CHƯƠNG 8

Ứng dụng Dental Payment

Mục tiêu

Trong chương này, bạn sẽ tìm hiểu để:

- Sử dụng **CheckBox** cho phép người dùng lựa chọn.
- Sử dụng hộp thoại hiển thị các thông báo.
- Sử dụng toán tử lôgíc để tạo ra biểu thức điều kiện.

Nội dung chính

Giới thiệu về CheckBox và hộp thoại thông báo

Nhiều ứng dụng Visual Basic sử dụng các **hộp thoại thông báo** (**message dialog**) để hiển thị thông báo cho người dùng. Bạn thường gặp hộp thoại trong khi sử dụng máy tính, từ những hộp thoại hướng dẫn lựa chọn file hay nhập mật khẩu, đến những hộp thoại thông báo lỗi khi sử dụng ứng dụng. Trong chương này, bạn sẽ sử dụng hộp thoại thông báo để cung cấp thông tin về lỗi nhập liệu cho người dùng.

Bạn có thể thấy rằng `TextBox` cho phép người dùng nhập được mọi giá trị. Trong một vài trường hợp, có thể bạn muốn sử dụng điều khiển để cung cấp cho người dùng những tùy chọn được định nghĩa trước. Bạn có thể làm việc này bằng cách thêm các `CheckBox` vào ứng dụng. Bạn cũng sẽ học về toán tử lôgíc. Bạn có thể sử dụng toán tử này trong ứng dụng để tạo ra quyết định dựa trên dữ liệu do người dùng nhập vào.

8.1 Chạy thử ứng dụng Dental Payment

Có rất nhiều thủ tục mà các nha sĩ cần phải thực hiện và người trợ lý sẽ đưa cho bạn một hóa đơn được tạo ra bởi máy vi tính. Trong chương này, bạn sẽ lập trình ứng dụng để tạo hóa đơn cho một vài thủ tục nha khoa cơ bản. Ứng dụng này phải đáp ứng những yêu cầu sau:

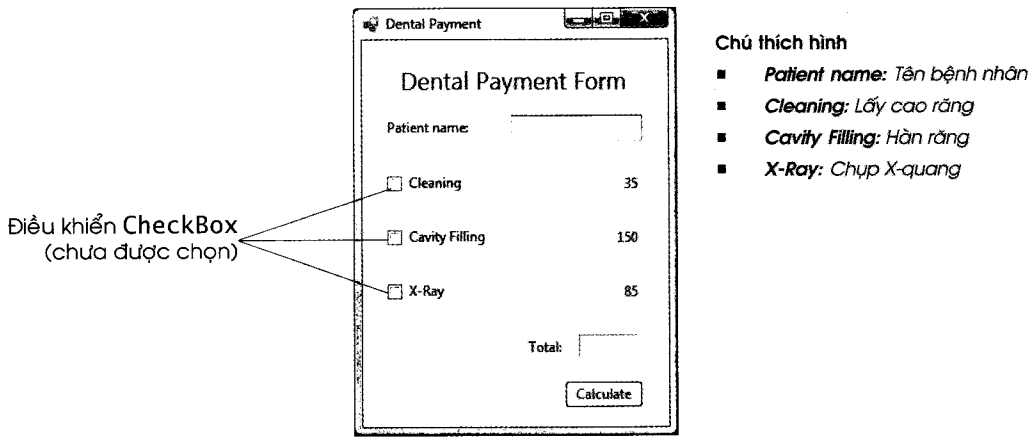
Yêu cầu đối với ứng dụng

Người quản lý văn phòng nha sĩ muốn tạo ra ứng dụng có thể dùng để tạo hóa đơn cho bệnh nhân. Ứng dụng này cho phép người dùng nhập tên bệnh nhân và chọn các dịch vụ được thực hiện trong suốt quá trình khám. Ứng dụng sẽ tính tổng chi phí phải trả. Nếu người dùng tính hóa đơn trước khi chọn bất kỳ dịch vụ nào, hay trước khi nhập tên bệnh nhân, thông báo lỗi sẽ được hiển thị, nhắc người dùng cần nhập liệu thông tin trường bị bỏ sót.

Trong ứng dụng **Dental Payment**, bạn sẽ sử dụng điều khiển `CheckBox` và hộp thoại thông báo lỗi để trợ giúp người dùng nhập liệu. Bạn bắt đầu bằng việc chạy thử ứng dụng đã hoàn thiện. Sau đó, bạn tìm hiểu những tính năng bổ sung của Visual Basic cần thiết để xây dựng cho mình một phiên bản của ứng dụng này.

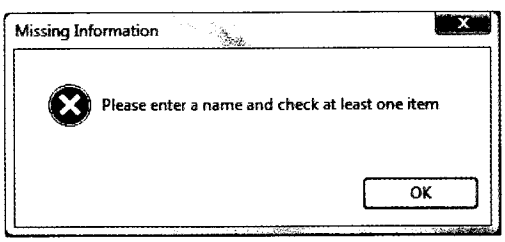
Chạy thử ứng dụng Dental Payment

1. ***Mở ứng dụng đã hoàn thiện.*** Mở thư mục `C:\Examples\Tutorial08\CompletedApplication\DentalPayment` để tìm ứng dụng **Dental Payment**. Nhấn đúp vào file `DentalPayment.sln` để mở ứng dụng trong Visual Basic IDE.
2. ***Chạy ứng dụng Dental Payment.*** Chọn **Debug > Start Debugging** để chạy ứng dụng này (xem Hình 8.1). Chú ý rằng có ba điều khiển hình vuông ở cột bên trái `Form`. Đây là các điều khiển **CheckBox**. CheckBox là một hình vuông nhỏ có thể trống hoặc chứa dấu chọn ở trong đó (☑). Khi CheckBox được chọn, dấu chọn sẽ xuất hiện trong hộp. CheckBox có thể được chọn đơn giản bằng cách nhấn vào hình vuông nhỏ hoặc nhấn vào dòng chữ mô tả CheckBox. Có thể bỏ chọn CheckBox bằng cách tương tự. Bạn sẽ học cách thêm điều khiển CheckBox vào `Form` ngay phần sau đây.



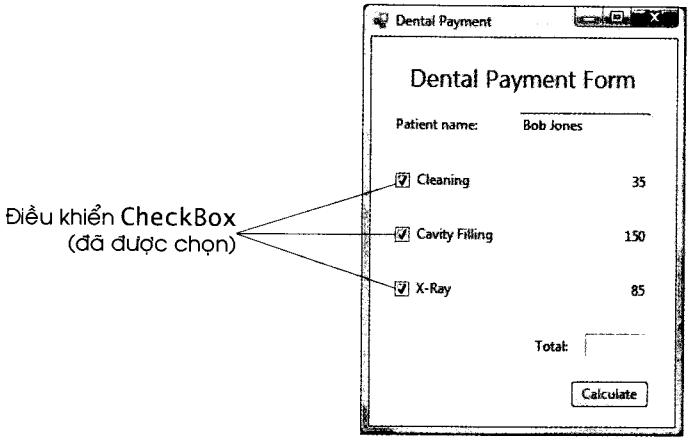
Hình 8.1 Chạy ứng dụng **Dental Payment** hoàn chỉnh.

3. ***Thử tính tổng số tiền trong khi không nhập dữ liệu.*** Để trống trường **Patient name:** và bỏ chọn bất kỳ CheckBox nào mà bạn vừa chọn. Nhấn vào `Button` **Calculate**. Chú ý rằng, hộp thoại thông báo sẽ xuất hiện yêu cầu bạn nhập dữ liệu (Hình 8.2). Đóng hộp thoại này bằng cách nhấn vào `Button` **OK**.

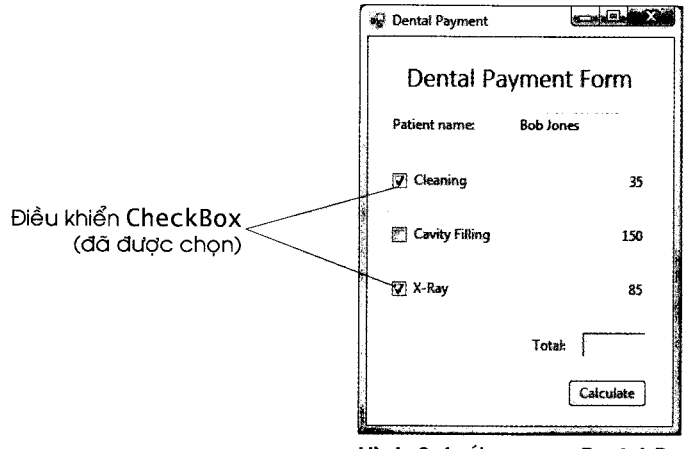


Hình 8.2 Hộp thoại thông báo xuất hiện khi không nhập tên hoặc không có CheckBox nào được chọn.

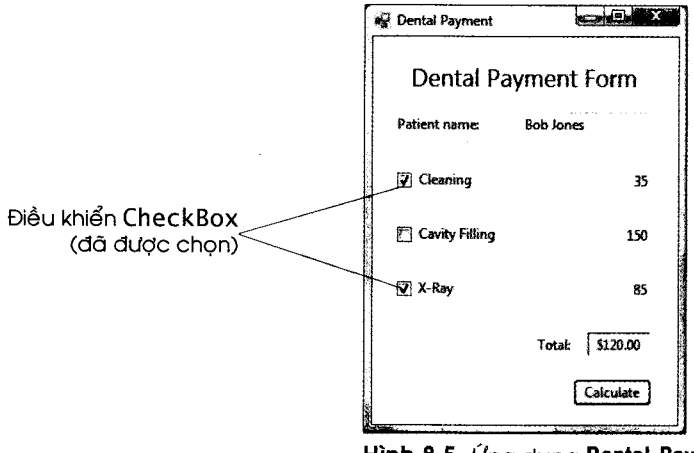
4. ***Nhập giá trị số lượng vào ứng dụng.*** Nhập Bob Jones vào trường **Patient name**. Chọn cả ba CheckBox bằng cách nhấn vào từng CheckBox một, dấu chọn sẽ xuất hiện trong mỗi CheckBox (Hình 8.3).
5. ***Bỏ chọn ở CheckBox Cavity Filling.*** Nhấn vào CheckBox **Cavity Filling** để bỏ chọn. Lúc này chỉ CheckBox **Cleaning** và **X-Ray** được chọn. (Hình 8.4).
6. ***Tính hóa đơn.*** Nhấn vào `Button` **Calculate**. Ứng dụng tính tổng chi phí của các dịch vụ được thực hiện trong suốt quá trình nha sĩ khám bệnh. Kết quả sẽ được hiển thị ở trường **Total:** (Hình 8.5).



Hình 8.3 Ứng dụng **Dental Payment** đã được nhập giá trị.



Hình 8.4 Ứng dụng **Dental Payment** có giá trị nhập vào thay đổi.



Hình 8.5 Ứng dụng **Dental Payment** tính giá trị tổng chi phí khám bệnh.

7. ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.
8. ***Đóng project.*** Đóng project bằng cách chọn **File > Close Project**.

8.2 Thiết kế ứng dụng Dental Payment

Nhớ lại rằng, mã giả là ngôn ngữ không chính quy giúp lập trình viên phát triển giải thuật. Đoạn mã giả sau mô tả các thao tác cơ bản trong ứng dụng **Dental Payment** khi người dùng nhấn vào Button **Calculate**:

```
Khi người dùng nhấn vào Button Calculate
    Xóa kết quả của lần tính toán trước
    Nếu người dùng không nhập tên bệnh nhân hoặc không chọn CheckBox
            nào thì
        Hiển thị hộp thoại thông báo
    Trái lại
        Khởi tạo giá trị tổng chi phí (total) là 0
        Nếu CheckBox "Cleaning" được chọn thì
            Cộng chi phí dịch vụ lấy cao răng vào tổng chi phí
        Nếu CheckBox "Cavity Filling" được chọn thì
            Cộng chi phí dịch vụ hàn răng vào tổng chi phí
        Nếu CheckBox "X-Ray" được chọn thì
            Cộng chi phí dịch vụ chụp X-quang vào tổng chi phí
        Định dạng giá trị tổng chi phí để hiển thị dưới dạng tiền tệ
        Hiển thị giá trị tổng chi phí
```

Bạn vừa chạy thử ứng dụng **Dental Payment** và tìm hiểu về đoạn mã giả biểu diễn ứng dụng này. Bạn sẽ sử dụng một bảng ACE để chuyển đoạn mã giả trên thành đoạn mã Visual Basic. Hình 8.6 liệt kê các hành động, điều khiển và sự kiện giúp bạn hoàn thành cho mình một phiên bản của ứng dụng này. Dữ liệu được nhập vào TextBox (`nameTextBox`) và các CheckBox (`cleanCheckBox`, `cavityCheckBox` và `xrayCheckBox`). Kết quả được hiển thị trong Label `totalResultLabel` khi Button (`CalculateButton`) được nhấn.

Bảng ACE cho ứng dụng Detal Payment

Hành động	Điều khiển/Lớp/ Đối tượng	Sự kiện
Hiển thị Label cho các điều khiển của ứng dụng	`titleLabel`, `nameLabel`, `totalLabel`, `cleanCostLabel`, `fillingCostLabel`, `xrayCostLabel`	Ứng dụng chạy
	`calculateButton`	`Click`
Xóa kết quả của lần tính toán trước	`totalResultLabel`	
Nếu người dùng không nhập tên bệnh nhân hoặc không chọn CheckBox nào thì	`nameTextBox`, `cleanCheckBox`, `cavityCheckBox`, `xrayCheckBox`	
Hiển thị hộp thoại thông báo	`MessageBox`	
Trái lại Khởi tạo giá trị tổng chi phí (total) là 0		

Hình 8.6 Bảng ACE của ứng dụng **Dental Payment**. (Phần 1/2)

Hành động	Điều khiển/Lớp/ Đối tượng	Sự kiện
Nếu CheckBox "Cleaning" được chọn Cộng chi phí dịch vụ lấy cao răng vào tổng chi phí	cleanCheckBox	
Nếu CheckBox "Cavity Filling" được chọn Cộng chi phí dịch vụ hàn răng vào tổng chi phí	cavityCheckBox	
Nếu CheckBox "X-Ray" được chọn Cộng chi phí dịch vụ chụp X-quang vào tổng chi phí	xrayCheckBox	
Định dạng giá trị tổng chi phí để hiển thị dưới dạng tiền tệ	String	
Hiển thị giá trị tổng chi phí	totalResultLabel	

Hình 8.6 Bảng ACE của ứng dụng **Dental Payment**. (Phần 2/2)

8.3 Sử dụng CheckBox

Mẹo thiết kế giao diện

Label của CheckBox nên có tính mô tả và càng ngắn càng tốt. Khi Label của CheckBox dài hơn một từ, hãy sử dụng kiểu viết hoa tiêu đề sách.

Như đã đề cập ở phần đầu, CheckBox là một hình vuông nhỏ chứa dấu chọn hoặc bỏ trống. Một CheckBox còn được gọi lại **button trạng thái (state button)** bởi vì nó có thể có trạng thái bật/tắt [đúng/sai]. Khi CheckBox được chọn sẽ xuất hiện dấu chọn. Cùng một thời điểm, có thể chọn nhiều CheckBox hoặc không chọn CheckBox nào. Đoạn văn bản xuất hiện bên cạnh CheckBox được gọi là nhãn CheckBox (**CheckBox label**).

Bạn có thể xác định xem CheckBox ở trạng thái bật (được chọn) hay không bằng cách sử dụng **thuộc tính Checked (Checked property)**. Nếu CheckBox được chọn, thuộc tính Checked có giá trị là True; ngược lại, nó có giá trị False. [*Lưu ý*: CheckBox cũng có thể có trạng thái vô định nếu thuộc tính ThreeState có giá trị True. Chúng ta sẽ không thảo luận về loại CheckBox như vậy trong cuốn sách này].

Bây giờ bạn sẽ tạo ứng dụng **Dental Payment** từ template đã được cung cấp sẵn. Phần dưới đây minh họa cách thêm CheckBox vào ứng dụng. Khi nhấn vào Button **Calculate**, ứng dụng bạn xây dựng trong hai phần tiếp theo sẽ không hiển thị hộp thoại trong trường hợp TexBox để trống hoặc tất cả CheckBox không được chọn. Bạn sẽ học cách hiển thị hộp thoại ở Phần 8.4.

Thêm CheckBox vào Form

1. ***Copy template vào thư mục làm việc.*** Copy thư mục C:\Examples\Tutorial08\TemplateApplication\DentalPayment vào thư mục C:\SimplyVB2008.
2. ***Mở file template của ứng dụng Dental Payment.*** Nhấn đúp vào file DentalPayment.sln trong thư mục DentalPayment để mở ứng dụng trong IDE Visual Basic. Nhấn đúp vào file DentalPayment.vb trong cửa sổ **Solution Explorer** nếu form không xuất hiện.
3. ***Thêm điều khiển CheckBox vào Form.*** Thêm một CheckBox vào Form bằng cách nhấn đúp vào biểu tượng

   ☑ CheckBox

   ở trên **Toolbox**. Lặp lại quá trình cho tới khi có ba CheckBox được thêm vào Form.

4. ***Tùy chỉnh các CheckBox***. Với ứng dụng này, bạn sẽ thay đổi các thuộc tính `Auto-Size`, `Location`, `Text`, `Size` và `Name` của mỗi `CheckBox`. Đầu tiên thiết lập giá trị `False` cho thuộc tính `AutoSize` của cả ba `CheckBox`. Tiếp đến, thay đổi thuộc tính `Size` cho cả ba `CheckBox` này là `122,24`. Thay đổi thuộc tính `Name` cho `CheckBox` đầu tiên là `cleanCheckBox`, thiết lập `Location` của nó là `22,113` và giá trị thuộc tính `Text` là `Cleaning`. Thay đổi thuộc tính `Name` cho `CheckBox` thứ hai là `cavityCheckBox` và thiết lập `Location` của nó là `22,160` và giá trị thuộc tính `Text` là `CavityFilling`. Thay đổi thuộc tính `Name` cho `CheckBox` cuối cùng là `xrayCheckBox` và thiết lập `Location` của nó là `22,207` và thuộc tính `Text` là `X-Ray`.
5. ***Lưu project***. Chọn **File > Save All** để lưu mã đã được sửa đổi.

Mẹo thiết kế giao diện

Căn lề các CheckBox theo cả chiều dọc và chiều ngang.

Sau khi đặt các `CheckBox` này lên `Form` và thiết lập thuộc tính của chúng, bạn cần phải viết mã cho xử lý sự kiện để thực hiện chức năng ứng dụng khi người dùng lựa chọn các `CheckBox` và nhấn vào **Calculate**.

Thêm xử lý sự kiện cho Button Calculate

1. ***Thêm xử lý sự kiện cho sự kiện `Click` của `calculateButton`***. Nhấn đúp vào `Button` **Calculate** trên `Form` để tạo ra xử lý sự kiện cho sự kiện `Click` của điều khiển này.
2. ***Thêm lệnh `If...Then` để tính hóa đơn cho bệnh nhân***. Thêm dòng 6-28 trên Hình 8.7 vào ứng dụng của bạn. Hãy chắc chắn rằng bạn đã thêm cả các dòng trắng và các ký tự nối dòng như Hình 8.7 để đoạn mã dễ đọc hơn và số dòng trong mã của bạn khớp với trên hình.

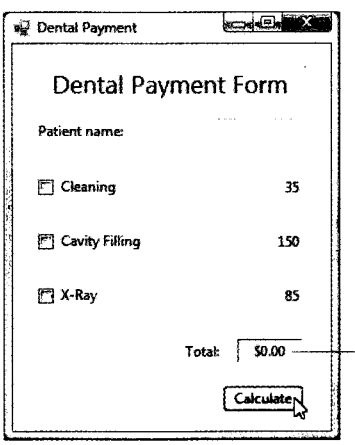
   Dòng 7 xóa đoạn văn bản hiển thị trên `Label` đầu ra từ lần tính toán trước. Dòng 10 khai báo biến `total` kiểu `Decimal` để lưu tổng chi phí của bệnh nhân. Biến này được khởi tạo giá trị là 0. Dòng 12-25 định nghĩa ba lệnh `If...Then` để xác định xem người dùng đã chọn `CheckBox` nào chưa. Mỗi một lệnh `If...Then` so sánh thuộc tính `Checked` của một `CheckBox` với giá trị `True`. Với mỗi lệnh `If...Then`, giá trị phí dịch vụ tính theo đô la được cộng vào biến `total` nếu `CheckBox` hiện tại được chọn. Ví dụ nếu `CheckBox` `cleanCheckBox` được chọn (dòng 13), dòng 14 sử dụng hàm `Val` để lấy giá trị từ `cleanCostLabel` và cộng thêm vào biến `total`. Dòng 28 hiển thị giá trị biến `total` (dưới định dạng tiền tệ) trong `totalResultLabel`.

Thêm đoạn mã được đánh dấu

```
' xử lý sự kiện Click
Private Sub calculateButton_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles calculateButton.Click

   ' xóa nội dung văn bản hiển thị trên Label
   totalResultLabel.Text = ""

   ' biến total chứa số tiền hóa đơn
   Dim total As Decimal = 0

   ' nếu bệnh nhân làm trắng răng
   If cleanCheckBox.Checked = True Then
      total += Val(cleanCostLabel.Text)
   End If

   ' nếu bệnh nhân hàn răng
   If cavityCheckBox.Checked = True Then
      total += Val(fillingCostLabel.Text)
   End If

   ' nếu bệnh nhân chụp x-quang
   If xrayCheckBox.Checked = True Then
      total += Val(xrayCostLabel.Text)
   End If

   ' hiển thị tổng chi phí
   totalResultLabel.Text = String.Format("{0:C}", total)
End Sub ' calculateButton_Click
```

Hình 8.7 Sử dụng thuộc tính Checked.

3. ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng của bạn. Chú ý rằng người dùng không bắt buộc phải nhập tên và chọn bất kỳ CheckBox nào trước khi nhấn vào Button **Calculate**. Nếu không có CheckBox nào được chọn, hóa đơn sẽ hiển thị giá trị $0.00 (Hình 8.8).
4. ***Chọn một CheckBox***. Chọn CheckBox **Cleaning** và nhấn vào Button **Calculate**. Trường **Total:** lúc này hiển thị **$35.00**.
5. ***Đóng ứng dụng.*** Đóng cửa sổ ứng dụng bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.



Ứng dụng tính toán hóa đơn là $0.00 nếu không có CheckBox nào được chọn

Hình 8.8 Kết quả ứng dụng khi không nhập liệu.

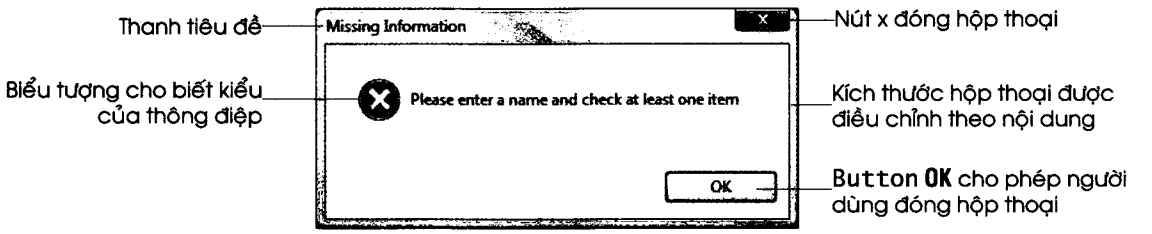
TỰ ÔN TẬP

1. Thuộc tính_____thiết lập nhãn cho CheckBox.
   a) Text b) Value
   c) Label d) Checked
2. Thuộc tính nào chỉ định CheckBox có được chọn hay không?
   a) Selected b) Checked
   c) Clicked d) Check

Đáp án: **1)** a. **2)** b.

8.4 Sử dụng hộp thoại để hiển thị thông báo

Trong ứng dụng hoàn thiện, hộp thoại hiển thị thông báo sẽ xuất hiện nếu người dùng tính tổng chi phí mà không lựa chọn một loại dịch vụ nào hoặc không nhập tên. Trong phần này, bạn sẽ học cách hiển thị hộp thoại khi tên bệnh nhân không được nhập. Khi đóng hộp thoại, điều khiển chuyển về `Form` ứng dụng. Hộp thoại thông báo trong ứng dụng của bạn sẽ được hiển thị trên Hình 8.9.



Hình 8.9 Hộp thoại được hiển thị bởi ứng dụng.

Hộp thoại thông báo có một thanh tiêu đề và một nút x đóng hộp thoại. Hộp thoại này còn chứa một thông báo (`Please enter a name and check at least one item` - Hãy nhập tên và chọn ít nhất một dịch vụ), `Button` **OK** cho phép người dùng **đóng** hộp thoại và một biểu tượng cho biết kiểu thông báo. (Trong trường hợp này, biểu tượng (⊗) chỉ ra rằng có một sự cố đã xảy ra).

Mẹo thiết kế giao diện

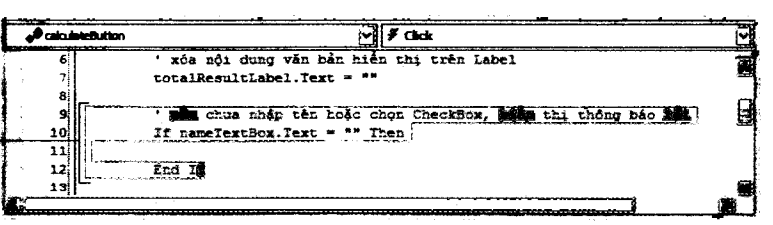
Nội dung văn bản được hiển thị trên hộp thoại nên có tính mô tả và càng ngắn càng tốt.

Hộp thoại thông báo được định nghĩa bởi lớp **MessageBox** và được hiển thị bằng cách sử dụng phương thức **MessageBox.Show**. Hộp thoại thông báo này được tùy chỉnh thông qua các đối số truyền vào `MessageBox.Show`. Phần dưới đây minh họa hộp thoại thông báo được hiển thị dựa vào điều kiện.

Hiển thị hộp thoại thông báo sử dụng MessageBox.Show

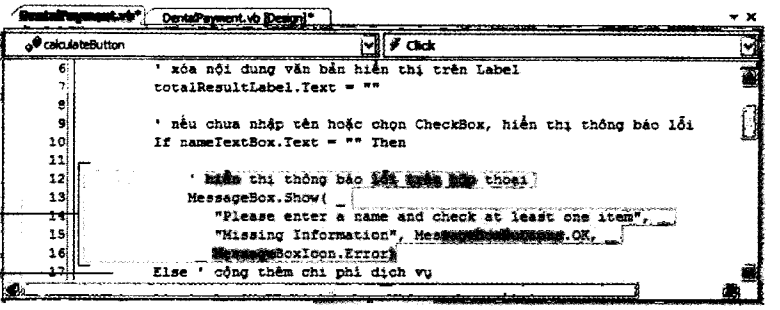
1. ***Thêm lệnh If...Then vào xử lý sự kiện cho sự kiện Click của calculateButton.*** Thông báo này chỉ hiển thị khi người dùng không nhập tên bệnh nhân. Ở phần sau, bạn sẽ thêm đoạn mã để xác định nếu không có CheckBox nào được chọn. Đặt con trỏ ở dòng 8 và ấn phím *Enter*. Sau đó chèn dòng 9-12 trên Hình 8.10 vào xử lý sự kiện của bạn. Hãy chắc chắn rằng, bạn đã chèn một dòng trắng sau từ khóa `End If`.

   Dòng 10 kiểm tra xem liệu dữ liệu đã được nhập vào `TextBox` **Patient name:** chưa. Nếu chưa được nhập liệu, biểu thức `nameTextBox.Text = ""` có giá trị `True`. Bạn sẽ thêm thân lệnh cho lệnh `If...Then` ở *Bước 2*.



Hình 8.10 Thêm lệnh `If...Then` vào xử lý sự kiện `Click` của `calculateButton` để hiển thị hộp thoại thông báo.

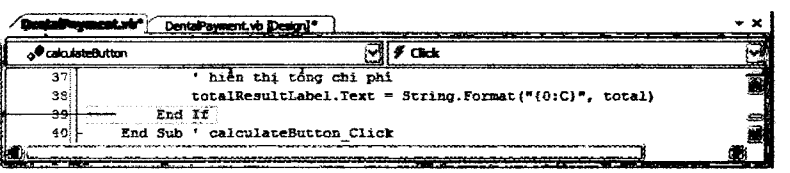
2. ***Thêm đoạn mã hiển thị hộp thoại thông báo.*** Thêm dòng 12-16 trong Hình 8.11 vào thân lệnh `If...Then` bạn vừa tạo trong bước trước. Đổi từ khóa `End If` (ở dòng 12 trên Hình 8.10) thành từ khóa `Else` (dòng 17 trên Hình 8.11). Chú ý rằng, đoạn mã bạn đã thêm vào sự kiện `Click` trước đây (Hình 8.7) bây giờ là thân mệnh đề `Else` của lệnh `If...Then...Else` mới này. Mệnh đề `Else` được đánh dấu có một lỗi cú pháp vì lệnh `If... Then...Else` hiện tại đang thiếu từ khóa `End If`. Bạn sẽ thêm các từ khóa này ở *Bước 3*.



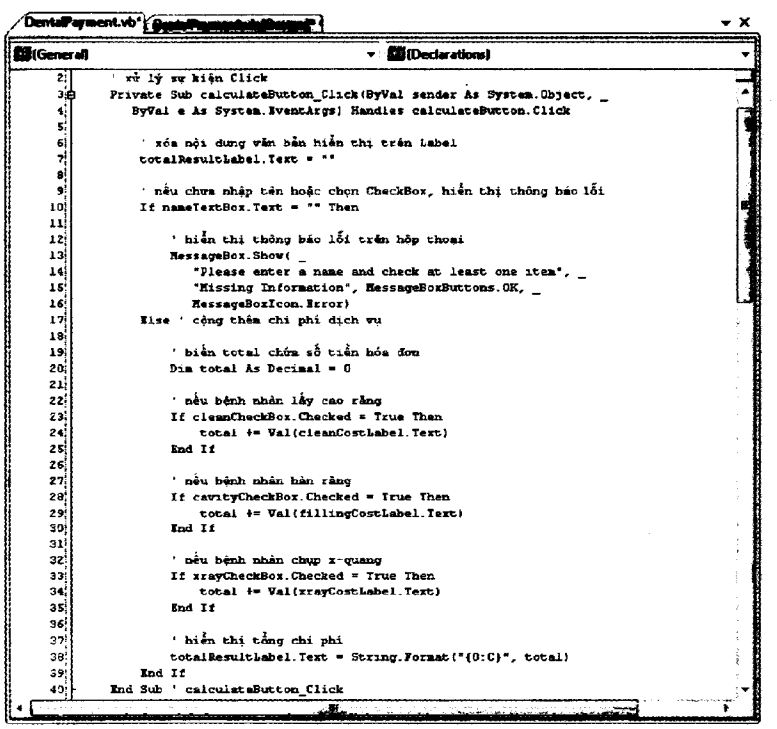
Hình 8.11 Đoạn mã hiển thị hộp thoại thông báo tới người dùng.

Dòng 13-16 gọi phương thức `MessgeBox.Show` sử dụng bốn đối số phân tách nhau bởi dấu phẩy. Đối số đầu tiên là đoạn văn bản hiển thị trên hộp thoại, đối số thứ hai là đoạn văn bản hiển thị trên thanh tiêu đề, đối số thứ ba cho biết những `Button` nào được hiển thị ở cuối hộp thoại, đối số thứ tư là biểu tượng gì sẽ xuất hiện ở bên trái của đoạn văn bản trong hộp thoại. Chúng ta sẽ thảo luận về hai đối số cuối cùng chi tiết hơn trong phần sau.

3. ***Kết thúc lệnh If...Then...Else.*** Kéo thanh cuộn tới cuối đoạn mã xử lý sự kiện của bạn. Chèn từ khóa `End If` (dòng 39 trên Hình 8.12) để kết thúc lệnh `If...Then...Else`. Hình 8.13 hiển thị toàn bộ phương thức `calculateButton_Click` sau khi đoạn mã mới được thêm vào. So sánh đoạn mã trong Hình 8.13 với đoạn mã của bạn để đảm bảo rằng bạn vừa thêm đoạn mã mới chính xác.

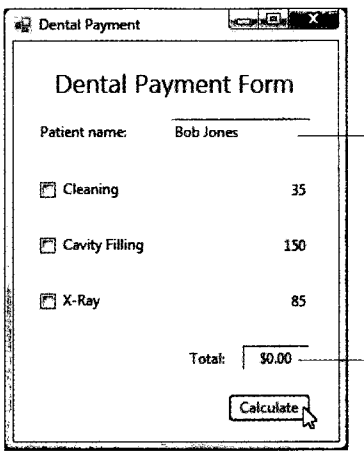


Hình 8.12 Kết thúc lệnh `If...Then...Else`.



Hình 8.13 Đoạn mã xử lý sự kiện `calculateButton_Click` hoàn chỉnh.

4. ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng. Chú ý rằng người dùng không phải chọn ít nhất một `CheckBox` trước khi nhấn `Button` **Calculate** nhưng phải nhập tên bệnh nhân vào `TextBox` **Patient name:**. Nếu không `CheckBox` nào được chọn, hóa đơn sẽ có giá trị là `$0.00` (Hình 8.14). Trong phần tiếp theo, bạn chỉnh sửa đoạn mã để kiểm tra xem người dùng đã chọn `CheckBox` nào chưa. [*Lưu ý*: Bạn không thể tương tác với `Form` cho đến khi bạn đóng hộp thoại thông báo.]
5. ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.



Hình 8.14 Tổng số tiền được tính khi không có `CheckBox` nào được chọn.

Trong ví dụ này, bạn đã truyền bốn đối số vào phương thức `MessageBox.Show`. Như chúng ta đã thảo luận, đối số thứ ba cho biết các `Button` nào sẽ hiển thị trên hộp thoại. Bạn đã truyền một trong các hằng **`MessageBoxButtons`** vào phương thức `MessageBox.Show`. Trong nội dung cuốn sách này, bạn chỉ sử dụng hằng `MessageBoxButtons.OK`. Hình 8.15 liệt kê các hằng `MessageBoxButtons`. Chú ý rằng, có thể hiển thị vài `Button` một lúc. Đối số thứ tư cho biết biểu tượng sẽ hiển thị trên hộp thoại. Để thiết lập biểu tượng hiển thị, bạn truyền một trong các hằng **`MessageBoxIcon`** vào phương thức `MessageBox.Show`. Hình 8.16 liệt kê danh sách các hằng biểu tượng có sẵn trong Visual Basic.

Hằng `MessageBoxButtons`	Mô tả
`MessageBoxButtons.OK`	`Button` **OK**. Cho phép người dùng hồi đáp lại ứng dụng rằng đã nhận thông báo.
`MessageBoxButtons.OKCancel`	`Button` **OK** và **Cancel**. Cho phép người dùng tiếp tục hay bỏ qua một hành động.
`MessageBoxButtons.YesNo`	`Button` **Yes** và **No**. Cho phép người dùng hồi đáp lại câu hỏi.
`MessageBoxButtons.YesNoCancel`	`Button` **Yes**, **No** và **Cancel**. Cho phép người dùng hồi đáp một câu hỏi hoặc bỏ qua một thao tác.
`MessageBoxButtons.RetryCancel`	`Button` **Retry** và **Cancel**. Cho phép người dùng thử lại hoặc bỏ qua một thao tác vừa thất bại.

Hình 8.15 Các hằng `MessageBoxButtons` của hộp thoại thông báo (1/2).

Hằng MessageBoxButtons	Mô tả
MessageBoxButtons.AbortRetryIgnore	Button **Abort**, **Retry** và **Ignore**. Khi một thao tác trong một dãy các thao tác thất bại, những Button này cho phép người dùng dừng toàn bộ dãy thao tác hoặc thử thực hiện lại thao tác thất bại hoặc bỏ qua thao tác thất bại và tiếp tục thực hiện những thao tác còn lại.

Hình 8.15 Các hằng MessageBoxButtons của hộp thoại thông báo (2/2).

Hằng MessageBoxIcon	Biểu tượng	Mô tả
MessageBoxIcon.Exclamation		Biểu tượng này chứa hình dấu chấm than. Được dùng để cảnh báo cho người dùng về vấn đề có thể xảy ra.
MessageBoxIcon.Information		Biểu tượng này chứa chữ "i". Được dùng để hiển thị thông tin về trạng thái của ứng dụng.
MessageBoxIcon.None		Không có biểu tượng nào được hiển thị.
MessageBoxIcon.Error		Biểu tượng này chứa dấu x màu trắng trong hình tròn màu đỏ. Dùng để cảnh báo người dùng về lỗi hoặc một tình huống nghiêm trọng.

Hình 8.16 Một số hằng MessageBoxIcon của hộp thoại thông báo.

TỰ ÔN TẬP

1. Gọi phương thức_____lớp MessageBox để hiển thị hộp thoại thông báo.
   a) Display b) Message
   c) Open d) Show
2. Biểu tượng của hộp thoại thông báo có chữ "i" được dùng _____.
   a) để hiển thị thông tin về trạng thái ứng dụng
   b) để cảnh báo người dùng về một vấn đề có khả năng xảy ra
   c) để đưa ra câu hỏi cho người dùng
   d) để cảnh báo cho người dùng về các tình huống cực kỳ nghiêm trọng

Đáp án: **1)** d. **2)** a.

8.5 Toán tử lôgíc

Cho đến thời điểm này, bạn chỉ mới học các **điều kiện đơn** (**simple condition**), như count <= 10, total > 1000 và number <> value. Mỗi lệnh lựa chọn bạn đã sử dụng chỉ đánh giá một điều kiện với một trong các toán tử >, <, >=, <=, = hoặc <>.

Mẹo tránh lỗi

Hãy viết điều kiện đơn giản nhất có thể bằng cách dùng số lượng toán tử lôgíc tối thiểu. Điều kiện có nhiều toán tử lôgíc có thể gây khó hiểu và gây ra các lỗi khó phát hiện trong ứng dụng.

Để xử lý nhiều điều kiện hiệu quả hơn, Visual Basic cung cấp **toán tử lôgíc** (**logical operator**) để tạo ra những điều kiện phức tạp bằng cách kết hợp các điều kiện đơn giản. Các toán tử lôgíc gồm **And**, **AndAlso**, **Or**, **OrElse**, **Xor**, **Not**. Chúng ta sẽ xét ví dụ sử dụng một số trong các toán tử này. Sau khi học về toán tử lôgíc, bạn sẽ sử dụng chúng để tạo ra điều kiện phức tạp trong ứng dụng **Dental Payment** để xác nhận rằng người dùng đã lựa chọn ít nhất một CheckBox.

Sử dụng toán tử AndAlso

Trong trường hợp bạn muốn *cả hai* điều kiện đều có giá trị là đúng thì mới thực thi có thể sử dụng toán tử `AndAlso` như sau:

```
If genderTextBox.Text = "Female" AndAlso age > = 65 Then
   seniorFemales +=1
End If
```

Lệnh `If...Then` này chứa hai điều kiện đơn. Điều kiện `genderTextBox.Text = "Female"` xác định xem một người có phải là nữ không và điều kiện `age >= 65` xác định xem người đó có phải là người cao tuổi không. Toán tử = và >= có mức độ ưu tiên cao hơn toán tử `AndAlso`. Lệnh `If...Then` này xét điều kiện kết hợp

```
genderTextBox.Text = "Female" AndAlso age > =65
```

Điều kiện này đạt giá trị `True` *khi và chỉ khi* cả hai điều kiện đơn đúng - ví dụ, `genderTextBox.Text` có giá trị `Female` và `age` có giá trị lớn hơn hoặc bằng 65. Khi điều kiện kết hợp này có giá trị đúng, biến `seniorFemales` tăng thêm 1. Tuy nhiên, nếu một hoặc cả hai điều kiện sai, ứng dụng sẽ bỏ qua không thực thi câu lệnh tăng thêm 1 và thực hiện lệnh tiếp theo lệnh `If...Then` này. Điều kiện kết hợp trên có thể thêm cặp ngoặc đơn phụ để dễ đọc hơn (tuy nhiên không cần thiết).

```
(genderTextBox.Text = "Female") AndAlso (age > =65)
```

Hình 8.17 minh họa kết quả sử dụng toán tử `AndAlso` với hai biểu thức. Bảng này liệt kê tất cả bốn tổ hợp `True`, `False` có thể cho *biểu thức 1* và *biểu thức 2* tương ứng với toán hạng bên trái và toán hạng bên phải. Bảng này gọi là **bảng chân lý (truth table)**. Biểu thức chứa toán tử quan hệ, toán tử so sánh bằng, toán tử lôgíc có giá trị `True` hoặc `False`.

Biểu thức 1	Biểu thức 2	Biểu thức 1 AndAlso Biểu thức 2
False	False	False
False	True	False
True	False	False
True	True	True

Hình 8.17 Bảng chân lý của toán tử `AndAlso`.

Sử dụng toán tử OrElse

Bây giờ, xét một toán tử `OrElse`. Trong trường hợp bạn muốn một *hoặc* cả hai điều kiện đúng thì mới thực thi, hãy sử dụng toán tử `OrElse`, như đoạn mã sau đây:

```
If (semesterAverage >= 90) OrElse (finalExam >= 90) Then
   MessageBox.Show("Student grade is "A", "Student Grade",_
      MessageBoxButtons.OK, MessageBoxIcon.Information)
End If
```

Lệnh này cũng chứa hai điều kiện đơn. Điều kiện `semesterAverage >= 90` được đánh giá để xác định xem sinh viên đó xứng đáng với điểm "A" trong học kỳ này vì kết quả nổi bật trong suốt kỳ học. Điều kiện `finalExam >= 90` được đánh giá để xác định xem sinh viên có xứng đáng với điểm "A" vì kết quả nổi bật trong kỳ thi. Lệnh `If...Then` xét điều kiện kết hợp

```
(semesterAverage >= 90) OrElse (finalExam >= 90)
```

và thưởng cho sinh viên điểm "A" nếu có một hoặc cả hai điều kiện đúng, nghĩa là sinh viên làm tốt trong suốt kỳ học hoặc làm tốt kỳ thi cuối khóa hoặc cả hai. Chú

Mẹo tránh lỗi

Khi viết điều kiện chứa các toán tử AndAlso và OrElse, sử dụng dấu ngoặc đơn để đảm bảo điều kiện được đánh giá đúng. Nếu không, lỗi lôgíc có thể xảy ra vì toán tử AndAlso có mức độ ưu tiên cao hơn OrElse.

ý rằng đoạn văn bản "Student grade is A" được hiển thị, ngoại trừ trường hợp cả hai điều kiện có giá trị False. Hình 8.18 cung cấp bảng chân lý cho toán tử OrElse. Chú ý rằng toán tử AndAlso có mức độ ưu tiên cao hơn toán tử OrElse.

Biểu thức 1	Biểu thức 2	Biểu thức 1 OrElse Biểu thức 2
False	False	False
False	True	True
True	False	True
True	True	True

Hình 8.18 Bảng chân lý của toán tử OrElse.

Đánh giá theo cơ chế tối ưu

Biểu thức chứa toán tử AndAlso được đánh giá cho đến khi biết giá trị của nó là đúng hay sai. Ví dụ, đánh giá biểu thức sau

```
(genderTextBox.Text = "Female") AndAlso (age >= 65)
```

sẽ bị dừng ngay nếu genderTextBox.Text không bằng "Female" (lúc đó đã đủ biết toàn bộ biểu thức này có giá trị False). Trong trường hợp này, đánh giá biểu thức thứ hai là không cần thiết vì biểu thức đầu sai thì toàn bộ biểu thức sẽ này sai. Biểu thức thứ hai chỉ được đánh giá khi và chỉ khi genderTextBox.Text bằng "Female" (có nghĩa là toàn bộ biểu thức có thể đạt giá trị True nếu điều kiện age >= 65 là đúng).

Tương tự, biểu thức chứa toán tử OrElse được đánh giá cho đến khi biết giá trị đúng sai của nó. Ví dụ, đánh giá biểu thức

```
If (semesterAverage >= 90) OrElse (finalExam >= 90) Then
```

sẽ bị dừng ngay lập tức nếu semesterAverage lớn hơn hoặc bằng 90 (có nghĩa là toàn bộ biểu thức có giá trị True). Trong trường hợp này, đánh giá biểu thức thứ hai là không cần thiết, vì khi biểu thức đầu tiên đúng thì toàn bộ biểu thức là đúng.

Phương pháp này đánh giá các biểu thức lôgíc cần thao tác ít hơn, do đó tốn ít thời gian hơn. Tính năng đánh giá biểu thức AndAlso và OrElse như thế này được gọi là **đánh giá theo cơ chế tối ưu (short-circuit evaluation)**. Visual Basic cũng cung cấp các toán tử And và Or không đánh giá theo cơ chế tối ưu như trên - luôn đánh giá toán hạng bên phải bất kể đã biết giá trị biểu thức điều kiện đúng hay sai. Trong ứng dụng Visual Basic, lợi ích hiệu năng sử dụng AndAlso và OrElse không đáng kể. Vấn đề thường gặp phải khi sử dụng AndAlso/OrElse thay vì And/Or là khi toán hạng bên phải chứa phần tử có thể gây ra kết quả không mong muốn (side effect), ví dụ như lời gọi hàm làm thay đổi giá trị của biến. Vì các phần tử này có thể không được thực hiện khi đánh giá theo cơ chế tối ưu, nên các lỗi lôgíc khó nhận biết có thể xảy ra. Vì vậy, bạn nên tránh viết các điều kiện chứa phần có thể gây ra kết quả không mong muốn như vậy.

Sử dụng toán tử Xor

Điều kiện chứa **toán tử lôgíc OR loại trừ (logical eXclusive OR - Xor**, hay còn gọi là toán tử logic Xor), có giá trị là True *khi và chỉ khi một toán hạng của nó có giá trị True và toán hạng còn lại có giá trị False*. Nếu tất cả các toán hạng có giá trị True hoặc tất cả có giá trị False, toàn bộ biểu thức điều kiện có giá trị là False. Hình 8.29 biểu diễn bảng chân lý của toán tử loại Xor. Toán tử này luôn đánh giá cả hai toán hạng (nghĩa là không có đánh giá theo cơ chế tối ưu).

Biểu thức 1	Biểu thức 2	Biểu thức 1 Xor Biểu thức 2
False	False	False
False	True	True
True	False	True
True	True	False

Hình 8.19 Bảng chân lý của toán tử lôgíc Xor.

Sử dụng toán tử Not

Toán tử Not của Visual Basic cho phép bạn "đảo ngược" giá trị của điều kiện. Khác với toán tử AndAlso, OrElse và Xor, mỗi toán tử này chứa *hai* biểu thức (gọi là các toán tử *hai ngôi*), toán tử Not là một toán tử *một ngôi* chỉ có *một* toán hạng. Toán tử Not được đặt trước một điều kiện để chọn một đường dẫn thực thi nếu điều kiện ban đầu (chưa sử dụng toán tử Not) đạt giá trị False. Toán tử Not được minh họa bởi đoạn mã sau đây:

```
If Not (grade = value) Then
   displayLabel.Text = "They are not equal!"
End If
```

Dấu ngoặc đơn bao quanh điều kiện grade = value để điều kiện dễ đọc hơn. Hầu hết lập trình viên thích viết

```
Not ( grade = value)
```

Hơn là

```
(grade <> value)
```

Hình 8.20 là bảng chân lý của toán tử Not. Trong phần tiếp theo, bạn sẽ sửa đổi ứng dụng **Dental Payment** bằng cách sử dụng biểu thức phức tạp hơn.

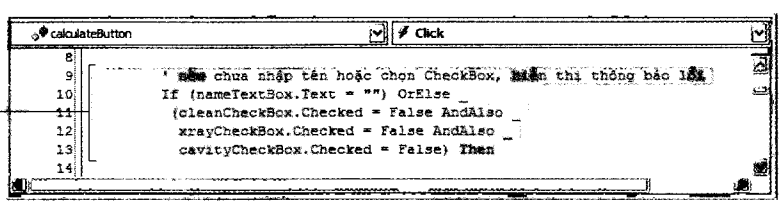
Biểu thức	Giá trị nghịch đảo của biểu thức
False	True
True	False

Hình 8.20 Bảng chân lý của toán tử Not.

Sử dụng toán tử lôgíc trong biểu thức phức tạp

1. ***Thêm biểu thức phức tạp vào xử lý sự kiện Click***. Thay dòng 9-10 trong file DentalPayment.vb bởi dòng 9-13 trong Hình 8.21.

Thêm đoạn mã được đánh dấu



Hình 8.21 Sử dụng toán tử lôgíc AndAlso và OrElse.

Dòng 10-13 định nghĩa một biểu thức lôgíc phức tạp hơn các biểu thức đã được học. Chú ý cách sử dụng OrElse và AndAlso. Nếu tên chưa được nhập hoặc không có CheckBox nào được chọn, hộp thoại sẽ hiển thị. Sau biểu thức ban đầu (nameTextBox.Text = ""), bạn sử dụng toán tử OrElse để chỉ ra rằng hoặc biểu thức ở bên trái (nameTextBox.Text = "") hoặc biểu thức ở bên phải (biểu thức kiểm tra không có CheckBox nào được chọn) phải đúng để toàn bộ biểu thức này có giá trị True để thực thi thân lệnh If...Then. Biểu thức phức tạp ở bên phải cũng sử dụng toán tử

`AndAlso` hai lần để xác định cả ba `CheckBox` không được chọn. Chú ý, vì `AndAlso` có độ ưu tiên cao hơn `OrElse`, nên dấu ngoặc ở dòng 10, 11 và 13 là không cần thiết.

2. ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng của bạn. Chú ý rằng người dùng phải nhập tên bệnh nhân và chọn ít nhất một `CheckBox` trước khi nhấn vào `Button` **Calculate**. Giao diện ứng dụng giống như Hình 8.1 và 8.4. Bây giờ, bạn đã sửa lại những nhược điểm trong ứng dụng **Dental Payment** lúc ban đầu.
3. ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

Hình 8.22 biểu diễn mã nguồn của ứng dụng **Dental Payment**. Những dòng mã chứa khái niệm lập trình mới mà bạn đã tìm hiểu trong chương này được đánh dấu.

```
1  Public Class DentalPaymentForm
2     ' xử lý sự kiện Click
3     Private Sub calculateButton_Click(ByVal sender As System.Object, _
4        ByVal e As System.EventArgs) Handles calculateButton.Click
5
6        ' xóa nội dung văn bản hiển thị trên Label
7        totalResultLabel.Text = ""
8
9        ' nếu chưa nhập tên hoặc chọn CheckBox, hiển thị thông báo
10       If (nameTextBox.Text = "") OrElse _
11       (cleanCheckBox.Checked = False AndAlso _
12       xrayCheckBox.Checked = False AndAlso _
13       cavityCheckBox.Checked = False) Then
14
15          ' hiển thị thông báo lỗi trên hộp thoại
16          MessageBox.Show( _
17             "Please enter a name and check at least one item", _
18             "Missing Information", MessageBoxButtons.OK, _
19             MessageBoxIcon.Error)
20       Else ' cộng thêm chi phí dịch vụ
21
22          ' biến total chứa số tiền hóa đơn
23          Dim total As Decimal = 0
24
25          ' nếu bệnh nhân làm trắng răng
26          If cleanCheckBox.Checked = True Then
27             total += Val(cleanCostLabel.Text)
28          End If
29
30          ' nếu bệnh nhân hàn răng
31          If cavityCheckBox.Checked = True Then
32             total += Val(fillingCostLabel.Text)
33          End If
34
35          ' nếu bệnh nhân chụp x-quang
36          If xrayCheckBox.Checked = True Then
37             total += Val(xrayCostLabel.Text)
38          End If
39
40          ' hiển thị tổng chi phí
41          totalResultLabel.Text = String.Format("{0:C}", total)
42       End If
43    End Sub ' calculateButton_Click
44 End Class ' DentalPaymentForm
```

Sử dụng các toán tử lôgíc và thuộc tính `Checked` của `CheckBox`

Hiển thị `MessageBox`

Xác định xem `cleanCheckBox` có được chọn không

Xác định xem `cavityCheckBox` có được chọn không

Xác định xem `xrayCheckBox` có được chọn không

Hình 8.22 Mã nguồn của ứng dụng **Dental Payment**.

TỰ ÔN TẬP

1. Toán tử một ngôi_____.
   a) chỉ có một toán hạng
   b) có hai toán hạng
   c) phải sử dụng từ khóa `AndAlso`
   d) có thể không chứa toán hạng nào
2. Toán tử_____được dùng để đảm bảo rằng cả hai điều kiện đều đúng.
   a) `Xor`
   b) `AndAlso`
   c) `Also`
   d) `OrElse`

Đáp án: 1) a. 2) b.

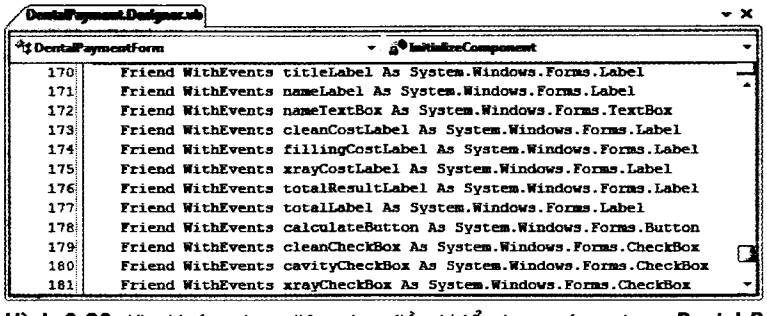
8.6 Mã do trình thiết kế tự sinh (Designer-Generated Code)

Trong Chương 6, bạn đã được học rằng tất cả các biến đều phải khai báo tên và kiểu trước khi sử dụng. Giống như các biến bạn đã từng khai báo, các điều khiển của giao diện cũng phải được khai báo trước khi sử dụng. Bạn có thể đang băn khoăn xem các biến này được khai báo ở đâu, vì bạn chưa từng nhìn thấy chúng trong bất kỳ một ví dụ nào cho đến thời điểm hiện tại. Một ưu điểm của Visual Basic là khi bạn làm việc trong chế độ **Design** để xây dựng và thiết kế giao diện đồ họa cho ứng dụng, Visual Basic sẽ tự động khai báo các điều khiển cho bạn. Visual Basic cũng tự sinh các đoạn mã tạo ra các điều khiển và đoạn mã thiết lập các thuộc tính cho các điều khiển này mỗi khi bạn thay đổi các thuộc tính trong cửa sổ **Properties** hoặc khi bạn kéo và thay đổi kích thước điều khiển trên giao diện `Form`.

Để đoạn mã ứng dụng dễ đọc hơn, Visual Basic "ẩn" khai báo của các điều khiển và đoạn mã mà nó tự sinh trong một file riêng biệt bắt đầu với tên tương tự `.vb` của `Form` nhưng kết thúc bằng **`Designer.vb`** - trong chương này, tên file là `DentalPaymentForm.Designer.vb`. Bằng cách đặt đoạn mã này trong file riêng biệt, Visual Basic cho phép người dùng tập trung vào lôgíc ứng dụng hơn là các chi tiết buồn tẻ khi xây dựng ứng dụng.

Bạn có thể xem file riêng biệt này và thậm chí là sửa chúng - nhưng bạn không nên sửa. Để xem file `Designer.vb` của ứng dụng **Dental Payment**, nhấn vào `Button` **Show All Files** (đã được thảo luận trong Mục 2.5) trong cửa sổ **Solution Explorer**, sau đó nhấn vào dấu cộng (+) bên cạnh `DentalPayment.vb` để mở rộng các nút của nó. Nhấn đúp vào file `DentalPaymentForm.Designer.vb` để xem đoạn mã.

Hình 8.23 minh họa các khai báo mà IDE sinh ra cho tất cả các điều khiển được dùng trong ứng dụng **Dental Payment** (dòng 170-181). Chú ý rằng IDE khai báo kiểu điều khiển với tên kiểu đầy đủ - gồm namespace `System.Windows.Forms` được thêm vào trước kiểu điều khiển. Các điều khiển được khai báo trong Hình 8.23 được sinh ra bởi dòng 25-36 trong file `DentalPaymentForm.Designer.vb` của ứng dụng hoàn chỉnh.



Hình 8.23 Khai báo giao diện cho điều khiển trong ứng dụng **Dental Payment**.

Ở *Bước 3* và *Bước 4* trong phần *Thêm CheckBox vào Form* ở phần đầu chương này, bạn đã đặt ba điều khiển CheckBox lên Form và thiết lập một số thuộc tính cho các CheckBox này. Hình 8.24 minh họa lệnh do IDE tạo ra dựa trên hành động của bạn. Ví dụ, bạn thiết lập thuộc tính Size cho các CheckBox là 122,24. Dòng 123, 132 và 141 là các lệnh được sinh ra. Đây là lệnh thay đổi kích thước của các CheckBox. Tương tự, bạn đã thay đổi vị trí của các CheckBox này. Lệnh thay đổi vị trí của các CheckBox trên dòng 121, 130 và 139.

DentalPayment.Designer.vb

DentalPaymentForm — InitializeComponent

```
118         '
119         'cleanCheckBox
120         '
121         Me.cleanCheckBox.Location = New System.Drawing.Point(22, 113)
122         Me.cleanCheckBox.Name = "cleanCheckBox"
123         Me.cleanCheckBox.Size = New System.Drawing.Size(122, 24)
124         Me.cleanCheckBox.TabIndex = 9
125         Me.cleanCheckBox.Text = "Cleaning"
126         Me.cleanCheckBox.UseVisualStyleBackColor = True
127         '
128         'cavityCheckBox
129         '
130         Me.cavityCheckBox.Location = New System.Drawing.Point(22, 160)
131         Me.cavityCheckBox.Name = "cavityCheckBox"
132         Me.cavityCheckBox.Size = New System.Drawing.Size(122, 24)
133         Me.cavityCheckBox.TabIndex = 10
134         Me.cavityCheckBox.Text = "Cavity Filling"
135         Me.cavityCheckBox.UseVisualStyleBackColor = True
136         '
137         'xrayCheckBox
138         '
139         Me.xrayCheckBox.Location = New System.Drawing.Point(22, 207)
140         Me.xrayCheckBox.Name = "xrayCheckBox"
141         Me.xrayCheckBox.Size = New System.Drawing.Size(122, 24)
142         Me.xrayCheckBox.TabIndex = 11
143         Me.xrayCheckBox.Text = "X-Ray"
144         Me.xrayCheckBox.UseVisualStyleBackColor = True
```

Hình 8.24 Lệnh thiết lập thuộc tính của các CheckBox.

Tất cả những gì bạn làm ở chế độ **Design** đều ảnh hưởng tới file Designer.vb, nhưng IDE sẽ xử lý đoạn mã giao diện này cho bạn. Điều này sẽ đơn giản hóa công việc lập trình và giúp nâng cao năng suất lao động của bạn, đồng thời cũng loại trừ nhiều lỗi lập trình thường gặp và lỗi đánh máy.

Giao diện đồ họa là công cụ vô cùng hữu hiệu để giao tiếp với máy tính. Tuy nhiên, chúng yêu cầu sử dụng nhiều mã hơn. Lập trình trực quan trong chế độ **Design** cho phép IDE tạo ra phần lớn mã cho bạn, một file ẩn Designer.vb tách riêng phần giao diện, do đó bạn có thể tập trung vào lôgíc ứng dụng.

8.7 Tổng kết

Trong chương này, cụ thể trong ứng dụng **Dental Payment**, bạn đã sử dụng điều khiển CheckBox để cung cấp lựa chọn cho người dùng. CheckBox có thể được chọn bằng cách nhấn vào chúng. Khi CheckBox được chọn, biểu tượng hình vuông của CheckBox sẽ chứa dấu chọn. Bạn có thể kiểm tra xem CheckBox này được chọn hay chưa bằng cách sử dụng thuộc tính Checked.

Ứng dụng **Dental Payment** của bạn sử dụng hộp thoại để hiển thị thông báo tới người dùng khi thông tin không được nhập chính xác. Để bổ sung hộp thoại cho ứng dụng, bạn sử dụng phương thức Show của lớp MessageBox và các hằng trong lớp MessageBoxButtons và MessageBoxIcon để hiển thị hộp thoại thông báo chứa Button và biểu tượng. Bạn đã sử dụng lệnh If...Then...Else để tính chi phí dịch vụ nha khoa hoặc để hiển thị hộp thoại thông báo nếu người dùng nhập thiếu thông tin. Phần sau cuốn sách này, bạn sẽ tìm hiểu cách vô hiệu hóa điều khiển (chẳng hạn Button) để tránh trường hợp nhập liệu không hợp lệ.

Bạn đã học cách sử dụng toán tử lôgíc AndAlso, trong đó cả hai điều kiện phải đúng thì toàn bộ biểu thức điều kiện mới đúng, nếu một điều kiện sai thì toàn bộ biểu thức điều kiện sẽ sai. Bạn cũng đã được học về toán tử lôgíc OrElse.

Toán tử này yêu cầu có ít nhất một điều kiện của nó phải đúng thì toàn bộ biểu thức điều kiện mới đúng, nếu cả hai điều kiện đều sai thì toàn bộ biểu thức điều kiện sẽ sai. Toán tử lôgíc `Xor` yêu cầu chỉ có duy nhất một điều kiện đúng thì toàn bộ biểu thức điều kiện đúng, nếu cả hai điều kiện cùng đúng hoặc sai thì toàn bộ biểu thức điều kiện sẽ sai. Toán tử lôgíc `Not` phủ định kết quả kiểu `Boolean` của điều kiện - `True` trở thành `False` và `False` thành `True`. Sau đó bạn đã sử dụng toán tử `AndAlso` và `OrElse` để tạo ra các biểu thức phức tạp.

Cuối cùng, bạn đã học về file `Designer.vb` do IDE tự sinh ra để chứa mã khai báo cho các điều khiển của giao diện đồ họa. File này cũng chứa lệnh thiết lập các điều khiển tương ứng với các hành động bạn thực hiện ở chế độ **Design**.

Trong chương tiếp theo, bạn sẽ học về các cấu trúc điều khiển của Visual Basic. Đặc biệt, bạn sẽ sử dụng lệnh lặp, là lệnh cho phép lập trình viên chỉ ra một hành động hay một nhóm các hành động được thực thi nhiều lần.

TỔNG KẾT KỸ NĂNG

Thêm CheckBox vào Form

- Nhấn đúp vào CheckBox trên **Toolbox**.

Chọn CheckBox

- Nhấn chuột vào CheckBox khi ứng dụng đang chạy, dấu chọn sẽ xuất hiện trên hình vuông của CheckBox.

Bỏ chọn CheckBox

- Nhấn chuột vào CheckBox đã được chọn khi ứng dụng đang chạy để bỏ dấu chọn.

Kiểm tra xem CheckBox đã được chọn chưa

- Sử dụng thuộc tính Checked của CheckBox.

Hiển thị hộp thoại

- Sử dụng phương thức `MessageBox.Show`.

Kết hợp nhiều điều kiện

- Sử dụng toán tử lôgíc để tạo các điều kiện phức tạp bằng cách kết hợp nhiều điều kiện đơn.

THUẬT NGỮ

bảng chân lý (truth table) - Bảng hiển thị kết quả kiểu `Boolean` của toán tử lôgíc cho tất cả các tổ hợp `True`, `False` có thể của các toán hạng.

`Button` trạng thái - `Button` có trạng thái bật/tắt (hoặc đúng/sai).

đánh giá theo cơ chế tối ưu (short-circuit evaluation) - Việc đánh giá toán hạng bên phải trong biểu thức `AndAlso` và `OrElse` chỉ xảy ra khi điều kiện đầu tiên đáp ứng tiêu chuẩn điều kiện.

điều khiển CheckBox - Thành phần giao diện đồ họa có hình vuông nhỏ, có thể chứa dấu chọn trong đó hoặc bỏ trống.

điều kiện đơn (simple condition) - Chứa một biểu thức có giá trị `True` hoặc `False`.

file `Designer.vb` - File chứa khai báo và lệnh cho phần giao diện đồ họa của ứng dụng.

hằng `MessageBoxButtons` - Là định danh chỉ ra `Button` nào sẽ được hiển thị trong hộp thoại `MessageBox`.

hằng `MessageBoxIcon` - Là định danh chỉ ra biểu tượng nào được hiển thị trong hộp thoại `MessageBox`.

hộp thoại thông báo (message dialog) - Cửa sổ hiển thị thông báo tới người dùng hoặc thu thập thông tin từ người dùng.

lệnh lặp (repetition statement) - Lệnh cho phép lập trình viên chỉ định một hành động hoặc một chuỗi hành động được thực thi nhiều lần.

lớp `MessageBox` - Cung cấp phương thức để hiển thị hộp thoại thông báo.

nhãn của CheckBox - Đoạn văn bản xuất hiện bên cạnh CheckBox.

phương thức `MessageBox.Show` - Hiển thị hộp thoại thông báo.

thuộc tính Checked của điều khiển CheckBox - Cho biết CheckBox được chọn (giá trị `True`) hoặc không được chọn (giá trị `False`).

toán tử lôgíc (logical operator) - Toán tử (ví dụ `AndAlso`, `OrElse`, `Xor` và `Not`) được dùng để tạo ra các điều kiện phức tạp bằng cách kết hợp nhiều điều kiện đơn.

toán tử And - Toán tử lôgíc được sử dụng để đảm bảo rằng *cả hai* điều kiện đều phải đúng thì mới thực thi hành động. Không thực hiện đánh giá theo cơ chế tối ưu.

toán tử `AndAlso` - Toán tử lôgíc được sử dụng để đảm bảo rằng *cả hai* điều kiện đều phải đúng thì mới thực thi hành động. Thực hiện đánh giá theo cơ chế tối ưu.

toán tử `Not` (toán tử lôgíc phủ định) - Toán tử lôgíc cho phép bạn phủ định giá trị của điều kiện. Điều kiện có giá trị `True`, khi thực hiện phủ định lôgíc sẽ có giá trị `False`; điều kiện có giá trị `False`, khi phủ định lôgíc sẽ có giá trị `True`.

toán tử `Or` - Toán tử lôgíc được dùng để đảm bảo rằng ít nhất một điều kiện phải đúng thì mới thực thi hành động.

toán tử `OrElse` - Toán tử lôgíc được dùng để đảm bảo rằng ít nhất một điều kiện phải đúng thì mới thực thi hành động. Thực hiện đánh giá theo cơ chế tối ưu.

toán tử `Xor` - Toán tử lôgíc mà có giá trị `True` khi và chỉ khi một toán hạng có giá trị `True` và toán hạng còn lại có giá trị `False`.

HƯỚNG DẪN THIẾT KẾ GIAO DIỆN

CheckBox

- Nhãn của CheckBox nên mô tả CheckBox càng ngắn càng tốt. Khi nhãn của CheckBox chứa nhiều hơn một từ, nên sử dụng kiểu viết hoa tiêu đề sách.
- Bố trí các CheckBox sao cho chúng gióng hàng ngang và hàng dọc.

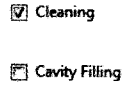
Hộp thoại thông báo

- Nội dung văn bản hiển thị trong hộp thoại nên có tính gợi tả và càng ngắn càng tốt.

ĐIỀU KHIỂN, SỰ KIỆN, THUỘC TÍNH & PHƯƠNG THỨC

CheckBox ☑ CheckBox Điều khiển này cho phép người dùng chọn một tùy chọn.

- ***Trên giao diện khi ứng dụng chạy***



- ***Thuộc tính***

`AutoSize` - Cho phép tự động điều chỉnh kích thước của CheckBox.

`Checked` - Chỉ định CheckBox này được chọn (tương ứng với giá trị `True`) hay không được chọn (tương ứng với giá trị `False`).

`Location` - Chỉ định vị trí của CheckBox ở trên `Form`.

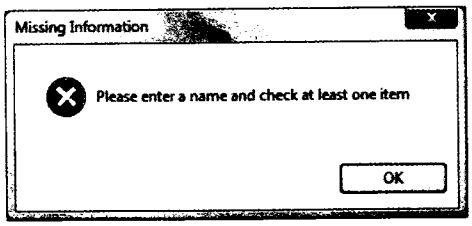
`Name` - Chỉ ra tên được sử dụng để truy cập CheckBox trong khi lập trình. Nên thêm hậu tố CheckBox vào cuối tên.

`Size` - Chỉ ra chiều rộng và chiều cao (bằng pixel) của CheckBox.

`Text` - Chỉ ra nội dung được hiển thị bên cạnh `TextBox`.

MessageBox Lớp này cho phép hiển thị hộp thoại thông báo.

- ***Trên giao diện khi ứng dụng chạy***

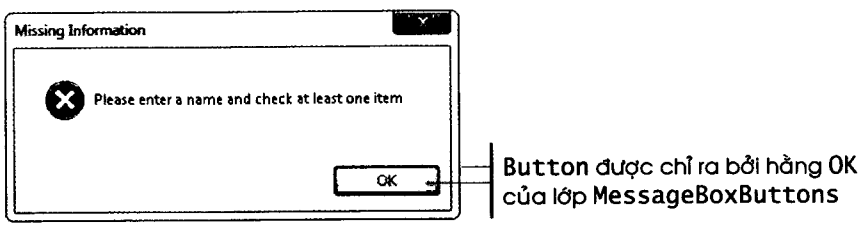


- ***Phương thức***

  Show - Hiển thị hộp thoại thông báo. Người dùng không thể tương tác với Form của ứng dụng cho đến khi đóng hộp thoại thông báo.

MessageBoxButtons Lớp này cung cấp hằng được dùng để chỉ ra các Button sẽ được hiển thị trong hộp thoại thông báo.

- ***Trên giao diện khi ứng dụng chạy***



- ***Hằng***

  OK - Button **OK**. Cho phép người dùng hồi đáp lại ứng dụng rằng đã nhận thông báo.

  OKCancel - Button **OK** và **Cancel**. Cho phép người dùng tiếp tục hay bỏ qua một hành động.

  YesNo - Button **Yes** và **No**. Cho phép người dùng hồi đáp lại câu hỏi.

  YesNoCancel - Button **Yes**, **No** và **Cancel**. Cho phép người dùng trả lời câu hỏi hoặc bỏ qua.

  RetryCancel - Button **Retry** và **Cancel**. Cho phép người dùng thử lại hoặc bỏ qua một thao tác vừa thất bại.

  AbortRetryCancel - Button **Abort**, **Retry** và **Ignore**. Khi một thao tác trong một dãy các thao tác thất bại, những Button này cho phép người dùng dừng toàn bộ dãy thao tác hoặc thử thực hiện lại thao tác thất bại hoặc bỏ qua thao tác thất bại và tiếp tục thực hiện những thao tác còn lại.

MessageBoxIcon Lớp này cung cấp các hằng dùng để chỉ ra biểu tượng nào sẽ hiển thị trong hộp thoại thông báo.

- ***Trên giao diện khi ứng dụng chạy***

- ***Hằng***

  Exclamation - Biểu tượng này chứa hình dấu chấm than. Được dùng để cảnh báo cho người dùng về vấn đề có thể xảy ra.

  Information - Biểu tượng này chứa chữ "i". Được dùng để hiển thị thông tin về trạng thái của ứng dụng.

  None - Không có biểu tượng nào được hiển thị trên hộp thoại thông báo.

`Error` - Biểu tượng này chứa dấu x màu trắng trong hình tròn màu đỏ. Dùng để cảnh báo người dùng về lỗi hoặc các tình huống nghiêm trọng.

CÂU HỎI TRẮC NGHIỆM

8.1 Tại một thời điểm, có thể lựa chọn bao nhiêu `CheckBox` trên giao diện đồ họa?

a) `0` b) `1`
c) `4` d) bất kỳ số lượng nào

8.2 Đối số đầu tiên truyền vào phương thức `MessageBox.Show` là_____.

a) nội dung hiển thị trong thanh tiêu đề của hộp thoại
b) hằng biểu diễn các `Button` sẽ được hiển thị trên hộp thoại
c) nội dung văn bản sẽ được hiển thị trên hộp thoại
d) hằng biểu diễn một biểu tượng sẽ hiển thị trên hộp thoại

8.3 Bạn có thể chỉ ra `Button` hoặc biểu tượng được hiển thị trên hộp thoại thông báo bằng cách sử dụng hằng `MessageBoxButtons` và hằng_____.

a) `MessageIcon` b) `MessageBoxImages`
c) `MessageBoxPicture` d) `MessageBoxIcon`

8.4 _____được dùng để tạo ra các điều kiện phức tạp.

a) Toán tử gán b) Biểu đồ hoạt động
c) Toán tử lôgíc d) Mã định dạng

8.5 Toán tử `AndAlso`_____.

a) thực hiện đánh giá theo cơ chế tối ưu
b) không phải là từ khóa
c) là toán tử so sánh
d) có giá trị `Fase` nếu cả hai toán hạng có giá trị `True`

8.6 Một `CheckBox` được chọn khi thuộc tính `Checked` của nó được thiết lập giá trị là_____.

a) `On` b) `True`
c) `Selected` d) `Checked`

8.7 Điều kiện *biểu thức thứ nhất* `AndAlso` *biểu thức thứ hai* có giá trị là `True` nếu_____.

a) *biểu thức thứ nhất* có giá trị `True` và *biểu thức thứ hai* có giá trị `False`
b) *biểu thức thứ nhất* có giá trị `False` và *biểu thức thứ hai* có giá trị `True`
c) cả *biểu thức thứ nhất* và *biểu thức thứ hai* đều có giá trị `True`
d) cả *biểu thức thứ nhất* và *biểu thức thứ hai* đều có giá trị `False`

8.8 Điều kiện *biểu thức thứ nhất* `OrElse` *biểu thức thứ hai* có giá trị là `False` nếu_____.

a) *biểu thức thứ nhất* có giá trị `True` và *biểu thức thứ hai* có giá trị `False`
b) *biểu thức thứ nhất* có giá trị `False` và b*iểu thức thứ hai* có giá trị `True`
c) cả *biểu thức thứ nhất* và *biểu thức thứ hai* đều có giá trị `True`
d) cả *biểu thức thứ nhất* và *biểu thức thứ hai* đều có giá trị `False`

8.9 Điều kiện *biểu thức thứ nhất* `Xor` *biểu thức thứ hai* đạt giá trị `True` nếu_____.

a) *biểu thức thứ nhất* có giá trị `True` và *biểu thức thứ hai* có giá trị `False`
b) *biểu thức thứ nhất* có giá trị `False` và *biểu thức thứ hai* có giá trị `True`
c) cả *biểu thức thứ nhất* và *biểu thức thứ hai* đều có giá trị `True`
d) Cả a và b

8.10 Điều kiện `Not(`*biểu thức thứ nhất* `AndAlso` *biểu thức thứ hai*`)` có giá trị `True` nếu____.

a) *biểu thức thứ nhất* có giá trị `True` và *biểu thức thứ hai* có giá trị `False`

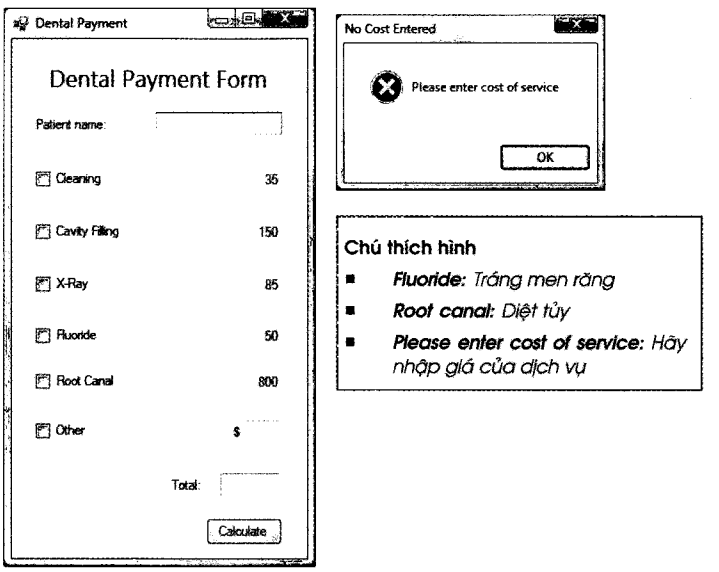
b) *biểu thức thứ nhất* có giá trị `False` và *biểu thức thứ hai* có giá trị `True`

c) cả *biểu thức thứ nhất* và *biểu thức thứ hai* đều có giá trị `True`

d) cả *biểu thức thứ nhất* và *biểu thức thứ hai* đều có giá trị `False`

BÀI TẬP

8.11 (***Nâng cấp ứng dụng Dental Payment***) Sửa ứng dụng **Dental Payment** trong chương này để thêm vào một vài dịch vụ mới, như Hình 8.25. Thêm chức năng thích hợp (sử dụng lệnh `If...Then`) để xác định xem có `CheckBox` mới nào được chọn không, nếu có, cộng thêm phí của những dịch vụ mới được chọn đó vào tổng tiền hóa đơn. Hiển thị thông báo lỗi trên hộp thoại nếu người dùng lựa chọn `CheckBox` **Other** mà không nhập giá của dịch vụ này.



Hình 8.25 Ứng dụng **Dental Payment** đã được nâng cấp.

a) ***Copy template của ứng dụng vào thư mục làm việc***. Copy thư mục `C:\Examples\Tutorial08\Exercises\DentalPaymentEnhanced` vào thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng***. Nhấn đúp vào file `DentalPaymentEnhanced.sln` trong thư mục `DentalPaymentEnhanced` để mở ứng dụng.

c) ***Thêm CheckBox, Label và TextBox***. Thêm hai `CheckBox` và hai `Label` vào `Form`. Những `CheckBox` mới này sẽ được gán nhãn là **Fluoride** và **Root Canal**. Thêm các `CheckBox` và `Label` này bên dưới `CheckBox` `X-Ray` và `Label` giá của nó. Giá điều trị cho tráng men răng là 50 USD, giá cho diệt tủy là 800 USD. Thêm `CheckBox` **Other** và `Label` chứa ký hiệu ($) vào `Form`, như Hình 8.25. Sau đó thêm một `TextBox` vào bên phải của `Label` ($) để người dùng có thể nhập phí thực hiện dịch vụ này.

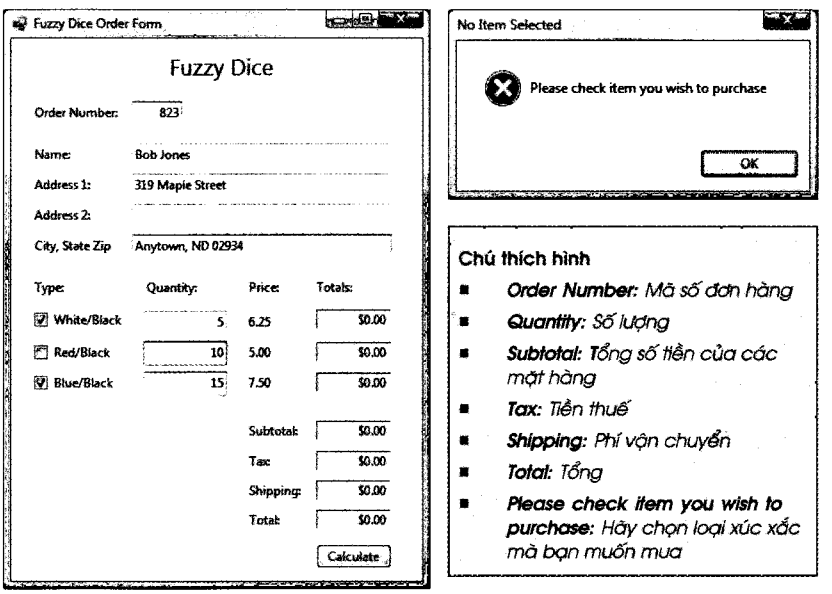
d) ***Thay đổi đoạn mã xử lý sự kiện `Click`***. Thêm đoạn mã vào xử lý sự kiện `calculateButton_Click` để xác định xem các `CheckBox` mới có được chọn không. Điều này có thể được thực hiện bằng cách sửa điều kiện phức trong lệnh `If...Then` đầu tiên. Thêm mệnh đề `ElseIf` để xác định xem có phải người dùng chọn `CheckBox` **Other** nhưng không nhập giá tiền không, nếu vậy, hiển thị một hộp thoại thông báo lỗi. Sử dụng lệnh `If...Then` để cập nhật tổng số tiền cho hóa đơn.

e) ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng. Kiểm tra ứng dụng bằng cách chọn một hoặc nhiều dịch vụ mới. Nhấn vào `Button` **Calculate** xác định xem tổng số tiền hóa đơn đã hiển thị chính xác chưa. Kiểm tra lại ứng dụng một lần nữa bằng cách chọn một số dịch vụ, sau đó chọn `CheckBox` **Other** và nhập giá trị tiền bằng đô la cho dịch vụ này. Nhấn vào `Button` **Calculate** và xác định xem tổng số tiền hóa đơn đã hiển thị đúng chưa, đã bao gồm cả giá của dịch vụ "khác" chưa.

f) ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

g) ***Đóng IDE***. Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

8.12 (***Ứng dụng Fuzzy Dice Order Form***) Viết ứng dụng cho phép người dùng xử lý các đơn đặt hàng xúc xắc, như Hình 8.26. Ứng dụng này tính tổng giá tiền của đơn đặt hàng, bao gồm cả thuế và phí vận chuyển. Các `TextBox` cho phép người dùng nhập mã số đơn hàng, tên khách hàng và địa chỉ chuyển hàng. Sử dụng `CheckBox` để lựa chọn màu xúc xắc. Ứng dụng cũng có một `Button`, khi nhấn vào `Button` này, ứng dụng sẽ tính tổng số tiền cho tất cả số xúc xắc và tính tổng số tiền của toàn bộ đơn đặt hàng này (bao gồm thuế và phí vận chuyển). Tỷ lệ thuế là 5%. Phí vận chuyển là `1,5` USD cho `20` cặp xúc xắc. Nếu nhiều hơn `20` cặp sẽ được miễn phí vận chuyển.



Hình 8.26 Ứng dụng **Fuzzy Dice Order Form.**

a) ***Copy template của ứng dụng vào thư mục làm việc***. Copy thư mục `C:\Examples\Tutorial 08\Exercises\FuzzyDiceOrderForm` tới thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng***. Nhấn đúp vào file `FuzzyDiceOrderForm.sln` trong thư mục `FuzzyDiceOrderForm` để mở ứng dụng.

c) ***Thêm CheckBox vào Form***. Thêm ba `CheckBox` vào `Form`. Nhãn của các `CheckBox` này lần lượt là **White/Black**, **Red/Black**, **Blue/Black**.

d) ***Thêm xử lý sự kiện `Click` và viết mã***. Tạo xử lý sự kiện `Click` cho `Button` **Calculate**. Ứng dụng nên cảnh báo cho người dùng khi họ nhập thông tin vào trường số lượng mà không chọn `CheckBox` tương ứng. Để có thể tính tổng số tiền, người dùng phải nhập thông tin mã số đơn hàng, tên và địa chỉ chuyển hàng. Sử dụng toán tử lôgic để đảm bảo rằng tất cả các điều kiện trên được đáp ứng. Nếu không, hiển thị hộp thoại thông báo lỗi.

e) ***Tính tổng chi phí.*** Tính tổng số tiền của các mặt hàng, tiền thuế, phí vận chuyển, tổng toàn bộ chi phí (total) và hiển thị kết quả trên các `Label` tương ứng.

f) ***Chạy ứng dụng.*** Chọn **Debug > Start Debugging** để chạy ứng dụng. Kiểm tra ứng dụng bằng cách nhập số lượng vào các mục có `CheckBox` đã được chọn. Hãy chắc chắn rằng ứng dụng của bạn tính thuế là 5%. Kiểm tra xem phí vận chuyển đã được miễn cho đơn hàng có số lượng lớn hơn 20 cặp xúc xắc chưa. Kiểm tra lại xem đoạn mã chứa các toán tử lôgíc đã hoạt động đúng chưa bằng cách nhập số lượng vào `TextBox` tương ứng với `CheckBox` chưa được chọn. Ví dụ, trên Hình 8.26, số lượng (trường Quantity:) của loại xúc xắc **Red/Black** được nhập dữ liệu, nhưng `CheckBox` tương ứng không được chọn. Do đó hộp thoại thông báo hiển thị như trên Hình 8.26.

g) ***Đóng ứng dụng.*** Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

h) ***Đóng IDE.*** Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

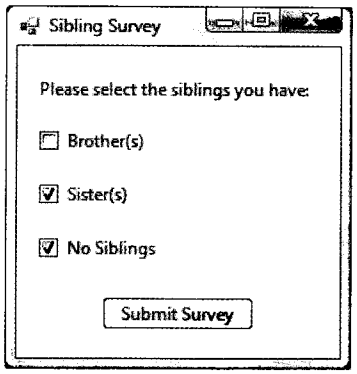
Đoạn mã này làm gì? ▶ **8.13** Giả sử rằng `nameTextBox` là `TextBox` và `otherCheckBox` là `CheckBox` đặt cạnh `TextBox` khác có tên là `otherTextBox`. Người dùng sẽ nhập giá trị vào các điều khiển này. Hãy cho biết đoạn mã dưới đây làm gì?

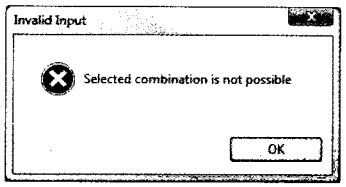
```
1  If (nameTextBox.Text = "" OrElse _
2      (otherCheckBox.Checked = True AndAlso _
3      otherTextBox.Text = "")) Then
4
5      MessageBox.Show("Please enter a name or value", _
6          "Input Error", MessageBoxButtons.OK, _
7          MessageBoxIcon.Error)
8
9  End If
```

Đoạn mã này có gì sai? ▶ **8.14** Giả sử rằng `nameTextBox` là `TextBox`. Tìm lỗi sai trong đoạn mã dưới đây:

```
1  If nameTextBox.Text = "John Doe" Then
2
3      MessageBox.Show("Welcome, John!", _
4          MessageBoxIcon.Exclamation)
5
6  End If
```

Sử dụng trình gỡ lỗi ▶ **8.15** (***Ứng dụng Sibling Survey***) Ứng dụng **Sibling Survey** hiển thị mối quan hệ anh em được chọn bởi người dùng. Nếu người dùng chọn ***CheckBox* Brother(s)** hoặc **Sister(s)** đồng thời chọn `CheckBox` **No Siblings**, người dùng sẽ được hỏi để xác minh lại lựa chọn trên. Trái lại, lựa chọn của người dùng sẽ được hiển thị trên `MessageBox`. Trong khi kiểm tra ứng dụng này, bạn sẽ thấy rằng ứng dụng chạy không chính xác. Sử dụng trình gỡ lỗi để tìm và sửa lỗi lôgíc trong đoạn mã này. Bài tập này nằm trong thư mục `C:\Examples\Tutorial08\Exercises\Debugger\SiblingSurvey`. Hình 8.27 hiển thị kết quả đúng của ứng dụng này.



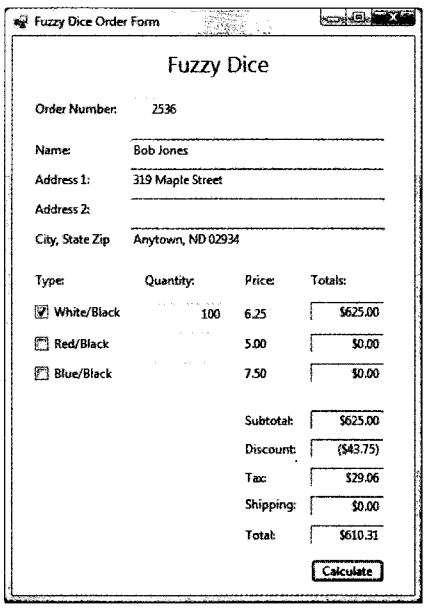


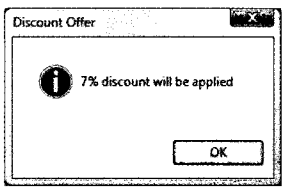
Chú thích hình

- ***Please select the siblings you have:*** *Hãy chọn quan hệ anh em ruột mà bạn có*
- ***Selected combinations is not possible:*** *Lựa chọn của bạn không hợp lệ*

Hình 8.27 Kết quả đúng của ứng dụng **Sibling Survey.**

Bài tập nâng cao ▶ **8.16** (***Chỉnh sửa ứng dụng Fuzzy Dice Order Form***) Sửa ứng dụng **Fuzzy Dice Order Form** trong Bài tập 8.12 để xác định xem khách hàng có nhận được 7% tiền giảm giá sản phẩm không (Hình 8.28). Những khách hàng đặt hàng với tổng số tiền lớn hơn 500 USD (không kể thuế và phí vận chuyển) sẽ được giảm giá.





Chú thích hình

- ***7% discount will applied:*** *Hóa đơn được giảm giá 7%*

Hình 8.28 Ứng dụng **Fuzzy Dice Order Form** đã được chỉnh sửa.

a) ***Mở ứng dụng***. Mở ứng dụng bạn đã tạo ra ở Bài tập 8.12.

b) ***Thêm Label để hiển thị thông tin về giảm giá***. Thêm hai `Label` vào `Form` để hiển thị số tiền giảm giá. Đặt các `Label` này phía dưới `Label` **Subtotal:** như Hình 8.28.

c) ***Xác định xem tổng số tiền của các mặt hàng có vượt quá 500 USD không***. Sử dụng lệnh `If...Then` để kiểm tra xem tổng số tiền của các mặt hàng có lớn hơn 500 USD không.

d) ***Hiển thị số tiền được giảm giá và tổng chi phí sau khi trừ đi số tiền được giảm giá***. Nếu khách hàng đặt hàng với số tiền nhiều hơn 500 USD, hiển thị hộp thoại thông báo, như Hình 8.28, thông báo cho người dùng là khách hàng được hưởng giảm giá 7%. Hộp thoại thông báo này chứa biểu tượng **Information** và `Button` **OK**. Tính giá trị 7% của tổng số tiền (chưa kể thuế và phí vận chuyển) và hiển thị số tiền giảm giá này trong trường **Discount:**. Lấy tổng số tiền của các mặt hàng trừ đi số tiền được giảm giá này và cập nhật kết quả lên trường **Total:**.

e) ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng của bạn. Hãy đảm bảo rằng ứng dụng tính và hiển thị số tiền giảm giá chính xác.

f) ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

g) ***Đóng IDE***. Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

CHƯƠNG 9

Ứng dụng Car Payment Calculator

Mục tiêu

Trong chương này, bạn sẽ tìm hiểu để:

- Sử dụng lệnh lặp **Do While...Loop** và **Do Until...Loop** để thực thi các lệnh trong ứng dụng lặp lại nhiều lần.
- Lệnh lặp được điều khiển bởi biến đếm.
- Hiển thị thông tin trên **ListBox**.
- Ghép chuỗi..

Nội dung chính

Giới thiệu về lệnh lặp Do While...Loop và Do Until...Loop

Chương này tiếp tục thảo luận về lập trình cấu trúc đã được đề cập trong Chương 7. Chúng tôi sẽ giới thiệu về **lệnh lặp (repetition statement)**, là lệnh điều khiển làm cho hành động được thực hiện lặp lại dựa trên giá trị của điều kiện. Bạn sẽ thực hiện nhiều tác vụ lặp lại dựa trên điều kiện. Ví dụ, mỗi lần lật một trang sách trong cuốn sách này (khi vẫn còn trang sách để đọc tiếp) là bạn đang lặp lại một tác vụ đơn giản có tên lật một trang sách dựa trên điều kiện vẫn còn trang sách để đọc tiếp.

Thực hiện lặp lại tác vụ là một phần quan trọng của lập trình cấu trúc. Lệnh lặp được sử dụng trong nhiều loại ứng dụng. Trong chương này, bạn học cách sử dụng các lệnh lặp `Do While...Loop` và `Do Until...Loop`. Bạn sẽ sử dụng lệnh lặp trong ứng dụng **Car Payment Calculator** do bạn xây dựng. Trong phần sau của chương, bạn sẽ được làm quen với các lệnh lặp khác nữa.

9.1 Chạy thử ứng dụng Car Payment Calculator

Vấn đề sau yêu cầu ứng dụng phải lặp lại một phép tính bốn lần - bạn sử dụng một lệnh lặp để giải quyết vấn đề này. Ứng dụng này phải đáp ứng những yêu cầu sau:

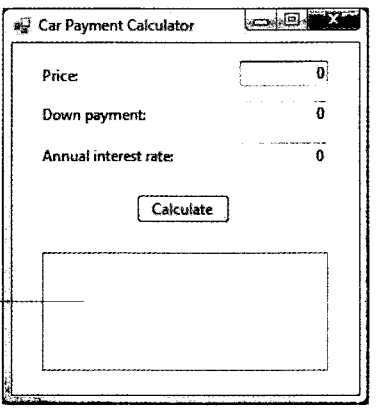
Yêu cầu đối với ứng dụng

Thông thường, ngân hàng đưa ra các khoản vay trả góp mua ô tô với kỳ hạn từ hai đến năm năm (24 đến 60 tháng). Người vay tiền phải trả một khoản tiền hàng tháng. Số tiền phải trả hàng tháng phụ thuộc vào khoảng thời gian vay, số tiền vay, và tỉ lệ lãi suất vay. Hãy tạo ứng dụng cho phép khách hàng nhập vào giá tiền của xe, số tiền trả trước và tỷ lệ lãi suất hàng năm của khoản vay. Ứng dụng sẽ hiển thị thời hạn khoản vay tính theo tháng và số tiền phải trả hàng tháng cho các khoản vay hai, ba, bốn hoặc năm năm. Các tùy chọn khác nhau cho phép người dùng dễ dàng so sánh kế hoạch hoàn trả và chọn cho mình kế hoạch thích hợp nhất.

Bạn bắt đầu bằng cách chạy thử ứng dụng đã hoàn thiện. Sau đó, bạn tìm hiểu những tính năng bổ sung cần thiết của Visual Basic để xây dựng cho mình một phiên bản của ứng dụng này.

Chạy thử ứng dụng Car Payment Calculator

1. ***Mở ứng dụng đã hoàn thiện.*** Mở thư mục `C:\Examples\Tutorial09\CompletedApplication\CarPaymentCalculator` để tìm ứng dụng **Car Payment Calculator**. Nhấn đúp vào file `CarPaymentCalculator.sln` để mở ứng dụng trong IDE Visual Basic.
2. ***Chạy ứng dụng.*** Chọn **Debug > Start Debugging** để chạy ứng dụng này (xem Hình 9.1). Chú ý rằng, điều khiển giao diện mới - điều khiển **`ListBox`** - cho phép người dùng xem và lựa chọn từ nhiều phần tử trên danh sách. Người dùng không thể thêm hoặc xóa các phần tử này khỏi `ListBox` bằng cách tương tác trực tiếp với nó. `ListBox` không cho phép nhập vào từ bàn phím - người dùng không thể thêm hoặc xóa các phần tử đã được chọn. Bạn phải viết đoạn mã để thêm hoặc xóa các phần tử này.

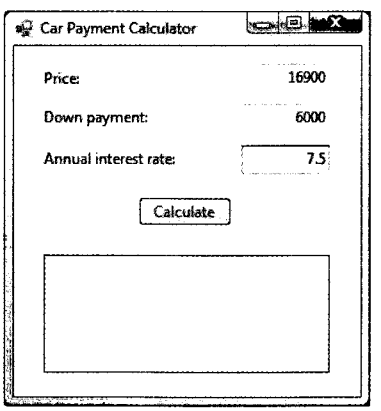


Chú thích hình

- ***Price:*** *Giá*
- ***Down payment:*** *Số tiền trả trước*
- ***Annual interest rate:*** *Tỉ lệ lãi suất hàng năm*

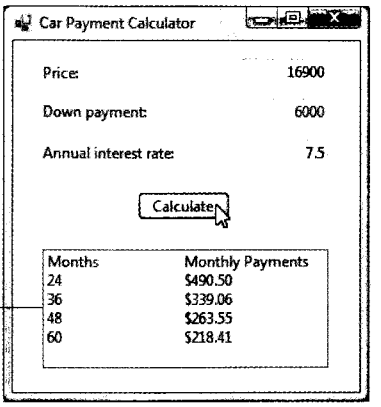
Hình 9.1 Ứng dụng **Car Payment Calculator** trước khi nhập dữ liệu.

3. ***Nhập các giá trị số lượng vào ứng dụng.*** Nhập giá trị 16900 vào `TextBox` **Price:**. Nhập 6000 vào `TextBox` **Down payment:**. Nhập `7.5` vào `TextBox` **Annual interest rate:**. Giao diện `Form` hiển thị giống như trên Hình 9.2.



Hình 9.2 Ứng dụng **Car Payment Calculator** sau khi nhập dữ liệu.

4. ***Tính số tiền phải trả hàng tháng.*** Nhấn vào `Button` **Calculate**. Ứng dụng hiển thị số tiền phải trả hàng tháng trên `ListBox` (Hình 9.3). Thông tin được hiển thị dưới dạng bảng.
5. ***Đóng ứng dụng.*** Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.
6. ***Đóng IDE.*** Đóng cửa sổ IDE Visual Basic bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.



Hình 9.3 Ứng dụng **Car Payment Calculator** hiển thị kết quả tính toán.

9.2 Lệnh lặp Do While...Loop

Lệnh lặp có thể lặp lại hành động tùy thuộc vào giá trị điều kiện (có thể có giá trị là đúng hoặc sai). Ví dụ, nếu bạn đi đến cửa hàng tạp hóa với một danh sách các mặt hàng cần mua, bạn mua lần lượt từng mặt hàng theo danh sách cho đến khi bạn có tất cả các mặt hàng này. Quá trình này được mô tả bởi đoạn mã giả sau:

```
Thực hiện các hành động sau đây khi vẫn còn mặt hàng trong danh sách
mặt hàng cần mua
    Đặt mặt hàng tiếp theo vào xe đẩy
    Gạch chéo tên mặt hàng này trong danh sách
```

Các lệnh này mô tả hành động lặp lại xảy ra trong suốt quá trình mua sắm. Điều kiện "vẫn còn mặt hàng trong danh sách mặt hàng cần mua" có thể đạt đúng hoặc sai. Nếu điều kiện đúng, thì hành động "Đặt mặt hàng đó vào xe đẩy" và "Gạch chéo tên mặt hàng này trong danh sách" được thực hiện theo thứ tự. Trong ứng dụng, những hành động này được thực hiện lặp lại nhiều lần nếu giá trị điều kiện vẫn còn đúng. Các lệnh này được viết lùi vào trong lệnh lặp tạo thành phần **thân lệnh** (**body**). Khi mặt hàng cuối cùng trong danh sách được đặt vào xe và bị gạch khỏi danh sách, điều kiện sẽ không còn đúng nữa. Lúc này, lệnh lặp kết thúc và lệnh đầu tiên sau lệnh lặp này sẽ được thực thi. Trong ví dụ mua hàng này, bạn sẽ tiếp tục thực hiện thanh toán tại quầy.

Lỗi lập trình thường gặp

Trong thân của lệnh `Do While... Loop`, bạn phải đặt vào đó hành động để cuối cùng điều kiện phải đạt được giá trị `False`. Nếu bạn không làm vậy, lệnh lặp không bao giờ kết thúc, gây ra lỗi lặp vô tận (infinite loop). Lỗi này làm "treo" ứng dụng. Khi xảy ra lỗi lặp vô hạn, quay trở lại cửa sổ IDE, chọn **Debug > Stop Debugging**.

Xét ví dụ về lệnh `Do While...Loop`, hãy xem đoạn ứng dụng được thiết kế để tìm lũy thừa của 3 đầu tiên lớn hơn 50.

```
Dim product As Integer = 3
Do While product <= 50
    product *= 3
Loop
```

Đoạn ứng dụng khai báo và khởi tạo biến `product` giá trị là 3, hãy tận dụng tính năng của Visual Basic cho phép bạn khởi tạo giá trị và khai báo biến đồng thời. Điều kiện trong lệnh `Do While...Loop` (`product <= 50`) được gọi là **điều kiện tiếp tục vòng lặp** (**loop-continuation condition**). Khi điều kiện này vẫn còn đúng, lệnh `Do While...Loop` thực thi lặp lại phần thân lệnh. Khi điều kiện tiếp tục vòng lặp không còn đúng nữa, lệnh `Do While...Loop` kết thúc thực thi và biến `product` chứa giá trị lũy thừa của 3 lớn hơn 50 đầu tiên.

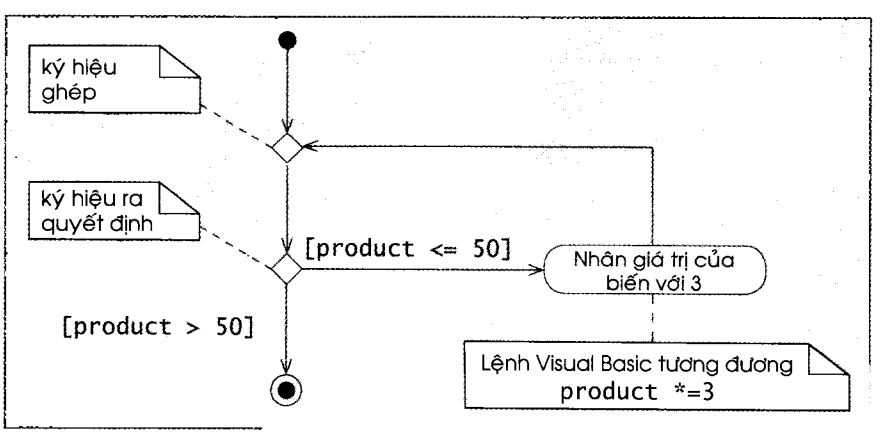
Hãy xem xét chi tiết sự thực thi của đoạn mã ở trên. Khi bắt đầu vào vòng lặp `Do While...Loop`, giá trị biến `product` là 3 và điều kiện tiếp tục vòng lặp (3 <=50) là đúng. Mỗi lần thực thi vòng lặp, biến `product` được nhân với 3 và nhận các giá trị lần lượt là 3, 9, 27 và 81. Khi biến `product` nhận giá trị 81, điều kiện trong lệnh `Do While...Loop` (`product <= 50`) có giá trị là `False`. Khi

lệnh lặp kết thúc, giá trị cuối cùng của biến product là 81. Đây là giá trị lũy thừa của 3 lớn hơn 50 đầu tiên. Ứng dụng tiếp tục thực thi lệnh tiếp theo sau lệnh Do While...Loop. Nếu điều kiện của lệnh Do While...Loop có giá trị khởi đầu là False, phần thân lệnh này sẽ không được thực thi và ứng dụng của bạn sẽ tiếp tục thực thi các lệnh sau từ khóa Loop. Phần tiếp theo mô tả từng bước khi lệnh lặp ở trên được thực thi.

Thực thi lệnh lặp Do While...Loop

1. Ứng dụng khai báo biến product và gán giá trị 3 cho biến này.
2. Ứng dụng chuyển đến thực hiện lệnh lặp Do While...Loop.
3. Điều kiện tiếp tục vòng lặp được kiểm tra. Điều kiện này có giá trị là True (product nhỏ hơn hoặc bằng 50), bởi vậy ứng dụng tiếp tục thực thi lệnh trong thân vòng lặp này.
4. Giá trị 3 hiện đang được lưu trong biến product sẽ được nhân 3 và kết quả này sẽ được gán cho biến product, lúc này biến product sẽ có giá trị là 9.
5. Điều kiện tiếp tục vòng lặp được kiểm tra. Điều kiện này có giá trị là True (product nhỏ hơn hoặc bằng 50), bởi vậy ứng dụng tiếp tục thực thi lệnh trong thân vòng lặp này.
6. Giá trị 9 hiện đang được lưu trong biến product sẽ được nhân 3 và kết quả này sẽ được gán cho biến product, lúc này biến product sẽ có giá trị là 27.
7. Điều kiện tiếp tục vòng lặp được kiểm tra. Điều kiện này có giá trị là True (product nhỏ hơn hoặc bằng 50), bởi vậy ứng dụng tiếp tục thực thi lệnh trong thân vòng lặp này.
8. Giá trị 27 hiện đang được lưu trong biến product sẽ được nhân 3 và kết quả này sẽ được gán cho biến product, lúc này biến product sẽ có giá trị là 81.
9. Điều kiện tiếp tục vòng lặp được kiểm tra. Điều kiện này có giá trị là False (product không nhỏ hơn hoặc bằng 50), bởi vậy ứng dụng thoát khỏi vòng lặp Do While...Loop và tiếp tục thực thi lệnh đầu tiên sau từ khóa Loop.

Hãy sử dụng biểu đồ hoạt động UML để minh họa luồng điều khiển của lệnh lặp Do While...Loop ở trên. Biểu đồ hoạt động UML trên Hình 9.4 chứa một trạng thái ban đầu, các mũi tên chuyển tiếp, một ký hiệu ghép (merge), một ký hiệu ra quyết định, hai điều kiện canh giữ (guard condition), một trạng thái hành động (action state), ba ghi chú và một trạng thái kết thúc. Trạng thái hành động này biểu diễn nhân giá trị biến product với 3.



Hình 9.4 Biểu đồ hoạt động UML minh họa lệnh lặp Do While...Loop.

Dễ dàng nhận thấy rằng biểu đồ hoạt động này biểu diễn lệnh lặp. Mũi tên chuyển tiếp xuất phát từ trạng thái hành động và trỏ về ký hiệu ghép, tạo một **vòng lặp** (**loop**). Các điều kiện canh giữ được kiểm tra mỗi lần lặp lại vòng lặp khi điều kiện canh giữ `product <= 50` vẫn đúng. Cuối cùng, điều kiện canh giữ `product > 50` đúng. Lúc này, lệnh `Do While...Loop` kết thúc, điều khiển chuyển tới lệnh tiếp theo trong ứng dụng ngay sau vòng lặp.

Hình 9.4 giới thiệu **ký hiệu ghép** (**merge symbol**) trong UML. Ngôn ngữ UML biểu diễn cả ký hiệu ghép và ký hiệu ra quyết định bằng hình thoi. Ký hiệu ghép kết hợp hai luồng hoạt động thành một luồng. Trong sơ đồ này, ký hiệu ghép kết hợp luồng từ trạng thái khởi đầu và luồng từ trạng thái hành động, cả hai luồng này chuyển đến điều kiện canh giữ tiếp tục vòng lặp để xác định xem thân vòng lặp có được bắt đầu (hay tiếp tục) thực thi hay không. Trong trường hợp này, biểu đồ hoạt động UML chuyển đến thực thi trạng thái hành động khi điều kiện canh giữ tiếp tục vòng lặp `product <= 50` là đúng.

Mặc dù, UML biểu diễn cả ký hiệu ra quyết định và ký hiệu ghép bằng hình thoi, nhưng các ký hiệu này có thể được phân biệt bằng số lượng mũi tên đến và mũi tên đi. Ký hiệu ra quyết định có một mũi tên trỏ tới hình thoi và có hai (hoặc nhiều hơn) mũi tên đi ra khỏi hình thoi này chỉ ra những chuyển tiếp có thể xảy ra từ điểm này. Hơn nữa, mỗi mũi tên đi ra khỏi ký hiệu ra quyết định có một điều kiện canh giữ bên cạnh. Ký hiệu ghép có hai (hoặc nhiều hơn) mũi tên chuyển tiếp trỏ tới hình thoi và chỉ một mũi tên chuyển tiếp đi ra khỏi hình thoi này, để cho thấy có nhiều hoạt động được gộp lại với nhau.

TỰ ÔN TẬP

1. Thân lệnh `Do While...Loop` thực thi_____.
   a) ít nhất một lần
   b) không bao giờ thực thi
   c) khi điều kiện của nó vẫn đúng
   d) khi điều kiện của nó có giá trị là sai
2. UML biểu diễn ký hiệu ghép và ký hiệu ra quyết định bằng_____.
   a) hình chữ nhật có hai cạnh bên hình vòng cung
   b) hình thoi
   c) hình tròn nhỏ màu đen
   d) hình bầu dục

Đáp án: **1)** c. **2)** b.

9.3 Lệnh lặp Do Until...Loop

Lỗi lập trình thường gặp

Nếu trong thân lệnh `Do Until...Loop` không hành động nào làm cho điều kiện của vòng lặp `Do Until...Loop` cuối cùng có giá trị là True sẽ gây ra lỗi lặp vô hạn.

Không giống như lệnh lặp `Do While...Loop`, lệnh lặp **`Do Until...Loop`** kiểm tra xem điều kiện có sai không trước khi tiếp tục thực hiện lặp, vòng lặp sẽ kết thúc khi điều kiện của nó có giá trị `True`. Điều kiện này gọi là **điều kiện kết thúcvòng lặp** (**loop-termination condition**). Ví dụ, bạn có thể xem việc mua sắm ở cửa hàng tạp hóa như sự lặp lại hành động lấy các mặt hàng có trong danh sách cho đến khi không còn mặt hàng nào trong danh sách nữa. Chú ý rằng, điều kiện "Không còn mặt hàng nào trong danh sách cần mua sắm" phải sai thì mới tiếp tục vòng lặp. Quá trình này được mô tả bởi đoạn mã giả sau:

```
Thực hiện các hành động sau cho đến tận khi không còn mặt hàng nào
      trong danh sách cần mua sắm
   Đặt mặt hàng tiếp theo vào xe đẩy
   Gạch chéo tên mặt hàng này trong danh sách
```

Những lệnh này mô tả các hành động lặp lại trong suốt thời gian mua sắm. Các lệnh trong phần thân `Do Until...Loop` được thực thi lặp lại chừng nào điều kiện kết thúc vòng lặp còn có giá trị `False`. Xét ví dụ về lệnh lặp `Do Until...Loop`, hãy xem lại đoạn ứng dụng được thiết kế để tìm lũy thừa của 3 đầu tiên lớn hơn 50:

```
Dim product As Integer = 3
Do Until product > 50
```

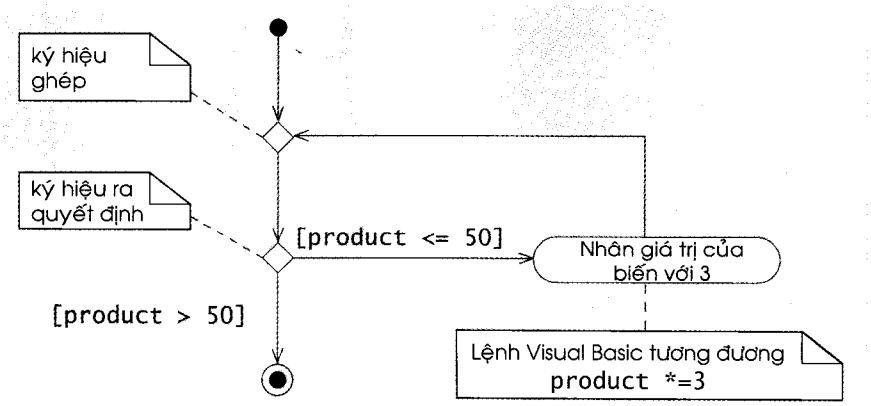
```
        Product *=3
    Loop
```

Phần tiếp theo mô tả từng bước lệnh lặp này thực thi.

Thực thi lệnh lặp Do Until...Loop

1. Ứng dụng khai báo biến `product` và gán giá trị 3 cho biến này.
2. Ứng dụng chuyển đến thực hiện lệnh lặp `Do Until...Loop`.
3. Điều kiện kết thúc vòng lặp được kiểm tra. Điều kiện này có giá trị là `False` (`product` không lớn hơn 50), bởi vậy ứng dụng tiếp tục thực thi lệnh trong thân vòng lặp này.
4. Giá trị 3 hiện đang được lưu trong biến `product` sẽ được nhân 3 và kết quả này sẽ được gán cho biến `product`, lúc này biến `product` sẽ có giá trị là 9.
5. Điều kiện kết thúc vòng lặp được kiểm tra. Điều kiện này có giá trị là `False` (`product` không lớn hơn 50), bởi vậy ứng dụng tiếp tục thực thi lệnh trong thân vòng lặp này.
6. Giá trị 9 hiện đang được lưu trong biến `product` sẽ được nhân 3 và kết quả này sẽ được gán cho biến `product`, lúc này biến `product` sẽ có giá trị là 27.
7. Điều kiện kết thúc vòng lặp được kiểm tra. Điều kiện này có giá trị là `False` (`product` không lớn hơn 50), bởi vậy ứng dụng tiếp tục thực thi lệnh trong thân vòng lặp này.
8. Giá trị 27 hiện đang được lưu trong biến `product` sẽ được nhân 3 và kết quả này sẽ được gán cho biến `product`, lúc này biến `product` sẽ có giá trị là 81.
9. Điều kiện kết thúc vòng lặp được kiểm tra. Điều kiện này có giá trị là `True` (`product` lớn hơn 50), bởi vậy ứng dụng thoát khỏi vòng lặp `Do Until... Loop` và tiếp tục thực thi lệnh đầu tiên sau từ khóa `Loop`.

Biểu đồ hoạt động UML trong Hình 9.5 minh họa luồng điều khiển cho lệnh lặp `Do Until...Loop`. Biểu đồ hoạt động này tương tự với biểu đồ hoạt động của lệnh lặp `Do While...Loop`. Một lần nữa, chú ý rằng (ngoài một trạng thái khởi đầu, các mũi tên chuyển tiếp, một trạng thái kết thúc và ba ghi chú) chỉ có các ký hiệu trên sơ đồ biểu diễn một trạng thái hành động, một quyết định và một ký hiệu ghép.



Hình 9.5 Biểu đồ hoạt động UML minh họa lệnh lặp `Do Until...Loop`.

TỰ ÔN TẬP

1. Lệnh lặp `Do Until...Loop` và lệnh lặp `Do While...Loop` khác nhau ở chỗ_____.
   a) lệnh lặp `Do While...Loop` lặp chừng nào điều kiện tiếp tục vòng lặp còn đạt giá trị `False`, trái lại lệnh lặp `Do Until...Loop` lặp chừng nào điều kiện tiếp tục vòng lặp còn đạt giá trị `True`
   b) lệnh lặp `Do Until...Loop` lặp chừng nào điều kiện kết thúc vòng lặp còn đạt giá trị `False`, trái lại lệnh lặp `Do While...Loop` lặp chừng nào điều kiện tiếp tục vòng lặp còn đạt giá trị `True`
   c) lệnh lặp `Do Until...Loop` luôn thực thi ít nhất một lần
   d) không hề khác nhau. Không có sự khác nhau giữa lệnh lặp `Do Until...Loop` và `Do While...Loop`
2. Các lệnh nằm trong thân lệnh `Do Until...Loop` được thực thi lặp lại chừng nào_____còn có giá trị `False`.
   a) điều kiện tiếp tục vòng lặp
   b) điều kiện thực hiện vòng lặp
   c) điều kiện kết thúc vòng lặp
   d) điều kiện vẫn còn vòng lặp

Đáp án: **1)** b. **2)** c.

9.4 Xây dựng ứng dụng Car Payment Calculator

Bạn đã được học về lệnh lặp `Do While...Loop` và `Do Until...Loop`, bây giờ bạn đã sẵn sàng để xây dựng ứng dụng **Car Payment Calculator**.

Đoạn mã giả dưới đây mô tả các thao tác cơ bản của ứng dụng **Car Payment Calculator** khi người dùng nhập thông tin và nhấn `Button` **Calculate**:

```
Khi người sử dụng nhấn vào Button Calculate
    Khởi tạo và gán giá trị cho thời hạn vay là hai năm
    Xóa mọi kết quả của lần tính toán trước trên ListBox
    Thêm dòng tiêu đề vào ListBox

    Lấy số tiền trả trước từ TextBox
    Lấy giá tiền của xe từ TextBox
    Lấy tỷ lệ lãi suất hàng năm từ TextBox

    Tính tổng số tiền vay (giá tiền của xe - số tiền trả trước)
    Tính tỷ lệ lãi suất vay mỗi tháng (tỷ lệ lãi suất hàng năm / 12)

    Thực hiện các hành động sau khi thời hạn vay nhỏ hơn hoặc bằng năm năm
        Chuyển thời hạn vay theo số năm sang theo số tháng

        Tính số tiền phải trả hàng tháng dựa trên tổng số tiền vay, tỷ lệ
            lãi suất mỗi tháng và thời hạn vay theo tháng

        Thêm kết quả vào ListBox
        Tăng thời hạn vay thêm một năm
```

Bạn vừa chạy thử ứng dụng **Car Payment Calculator** và tìm hiểu về biểu diễn mã giả của ứng dụng này. Bây giờ hãy sử dụng bảng Hành động/Điều khiển/Sự kiện (ACE) để giúp bạn chuyển đoạn mã giả sang ngôn ngữ Visual Basic. Hình 9.6 liệt kê các hành động, điều khiển, sự kiện giúp bạn tự hoàn thiện cho mình một phiên bản của ứng dụng này.

Chú ý rằng, đoạn mã giả lấy giá trị tiền trả trước, giá tiền xe và tỷ lệ lãi suất hàng năm, sau đó tính tổng số tiền vay và tỷ lệ lãi suất mỗi tháng trước lệnh lặp bởi vì chúng chỉ cần thực thi một lần. Các lệnh có kết quả khác trong mỗi lần lặp được viết bên trong lệnh lặp. Phần thân lệnh lặp bao gồm: Chuyển thời hạn vay từ số năm sang số tháng, tính số tiền phải trả hàng tháng dựa trên tổng số tiền vay, tỷ lệ lãi suất mỗi tháng và thời hạn vay theo tháng. Hiển thị kết quả tính toán và tăng thời hạn vay thêm một năm.

Bảng ACE cho ứng dụng Car Payment Calculator

Hành động	Điều khiển	Sự kiện
Hiển thị Label cho các điều khiển của ứng dụng	priceLabel, downPaymentLabel, interestLabel	Ứng dụng chạy
	calculateButton	Click
Khởi tạo và gán giá trị cho thời hạn vay là hai năm		
Xóa mọi kết quả của lần tính toán trước trên ListBox	paymentsListBox	
Thêm dòng tiêu đề vào ListBox	paymentListBox	
Lấy số tiền trả trước từ TextBox	downPaymentText-Box	
Lấy giá tiền của xe từ TextBoxt	priceTextBox	
Lấy tỷ lệ lãi suất hàng năm từ TextBox	interestTextBox	
Tính tổng số tiền vay (giá tiền của xe - số tiền trả trước)		
Tính tỷ lệ lãi suất vay mỗi tháng (tỷ lệ lãi suất hàng năm / 12)		
Thực hiện các hành động sau khi thời hạn vay nhỏ hơn hoặc bằng năm năm Chuyển thời hạn vay theo số năm sang theo số tháng		
Tính số tiền phải trả hàng tháng dựa trên tổng số tiền vay, tỷ lệ lãi suất mỗi tháng và thời hạn vay theo tháng		
Thêm kết quả vào ListBox	paymentListBox	
Tăng thời hạn vay thêm một năm		

Hình 9.6 Bảng ACE của ứng dụng **Car Payment Calculator**.

Ứng dụng sẽ hiển thị kết quả tính toán lên `ListBox`. Phần sau đây bạn thêm và tùy chỉnh `ListBox` để hiển thị kết quả.

Thêm ListBox vào ứng dụng Car Payment Calculator

1. ***Copy template vào thư mục làm việc của bạn.*** Copy thư mục `C:\Examples\Tutorial09\TemplateApplication\CarPaymentCalculator` vào thư mục `C:\SimplyVB2008`.
2. ***Mở file template của ứng dụng Car Payment Calculator.*** Nhấn đúp vào file `CarPaymentCalculator.sln` trong thư mục `CarPaymentCalculator` để mở ứng dụng trong IDE Visual Basic. `TextBox` dành cho người dùng nhập liệu và `Button` **Calculate** đã có sẵn trong file template.
3. ***Thêm điều khiển ListBox vào Form.*** Nhấn đúp vào ký hiệu điều khiển `ListBox`,

trên **Toolbox**. Thay đổi thuộc tính `Name` của `ListBox` thành `paymentsListBox`. Thiết lập thuộc tính `Location` của `ListBox` này là

Thói quen lập trình tốt

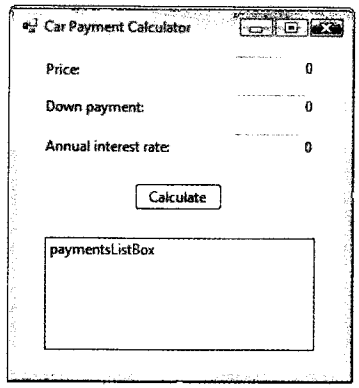
Thêm hậu tố `ListBox` vào sau tên của điều khiển `ListBox`.

Mẹo thiết kế giao diện

ListBox nên có kích thước đủ lớn để hiển thị toàn bộ nội dung hoặc đủ lớn để thanh cuộn có thể sử dụng dễ dàng.

24, 166 và thuộc tính Size là 230, 94. Trên Hình 9.7 là giao diện Form với điều khiển ListBox. Chú ý rằng ListBox này hiển thị thuộc tính Name trong chế độ **Design** - tên điều khiển không được hiển thị khi ứng dụng đang chạy.

4. ***Lưu project***. Chọn **File > Save All** để lưu lại thay đổi.

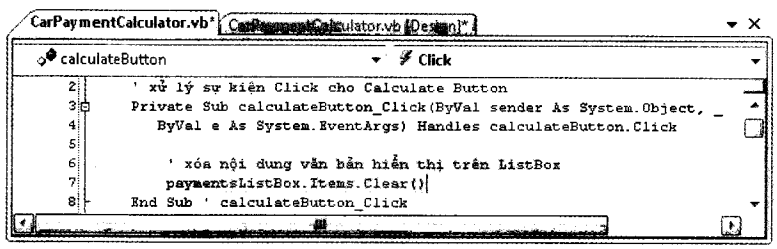


Hình 9.7 ListBox đã được thêm vào Form ứng dụng **Car Payment Calculator.**

Sau khi thêm ListBox, bạn phải thêm xử lý sự kiện vào ứng dụng này để hồi đáp lại sự kiện nhấn vào Button **Calculate** của người dùng. Xử lý sự kiện calculateButton_Click cập nhật nội dung của ListBox. Phần sau mô tả cách thêm các phần tử vào Listbox và cách xóa nội dung trên ListBox.

Sử dụng mã để thay đổi nội dung của một ListBox

1. ***Thêm xử lý sự kiện vào Button Calculate***. Nhấn đúp vào Button **Calculate** để tạo ra xử lý sự kiện calculateButton_Click trống.
2. ***Xóa nội dung hiển thị trên điều khiển ListBox***. Thêm dòng 6-7 trên Hình 9.8 vào calculateButton_Click. Mỗi lần người dùng nhấn vào Button **Calculate**, tất cả nội dung từng hiển thị trên ListBox này sẽ bị xóa. Để xóa tất cả nội dung trên ListBox, gọi phương thức **Clear** của thuộc tính **Items** (dòng 7). Có thể thêm hoặc xóa nội dung trên Listbox bằng cách sử dụng thuộc tính Items này. Thuộc tính Items trả lại đối tượng chứa danh sách các phần tử hiển thị trên ListBox. Chú ý rằng chúng ta vừa thêm chú thích ở dòng 2 và 8, ngắt khai báo calculateButton_Click thành hai dòng để mã dễ đọc hơn.

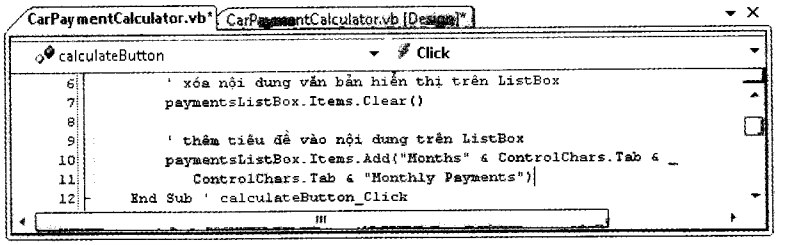


Hình 9.8 Xóa nội dung của ListBox.

Mẹo thiết kế giao diện

Sử dụng các tiêu đề trong một ListBox khi bạn hiển thị dữ liệu dạng bảng. Hãy thêm tiêu đề mô tả những thông tin được hiển thị trên ListBox để người dùng dễ hiểu.

3. ***Thêm nội dung vào điều khiển ListBox***. Thêm dòng 9-11 trên Hình 9.9 vào calculateButton_Click. ListBox này hiển thị số tháng và số tiền phải trả mỗi tháng. Để làm cho thông tin hiển thị rõ ràng hơn, đoạn văn bản - còn gọi là **tiêu đề (header)** - được thêm vào ListBox. Phương thức **Add** (dòng 10-11 trên Hình 9-9) thêm tiêu đề cột "Months" và "Monthly Payment" được phân tách bởi hai ký tự tab vào thuộc tính Items của ListBox.



Hình 9.9 Thêm tiêu đề vào `Listbox`.

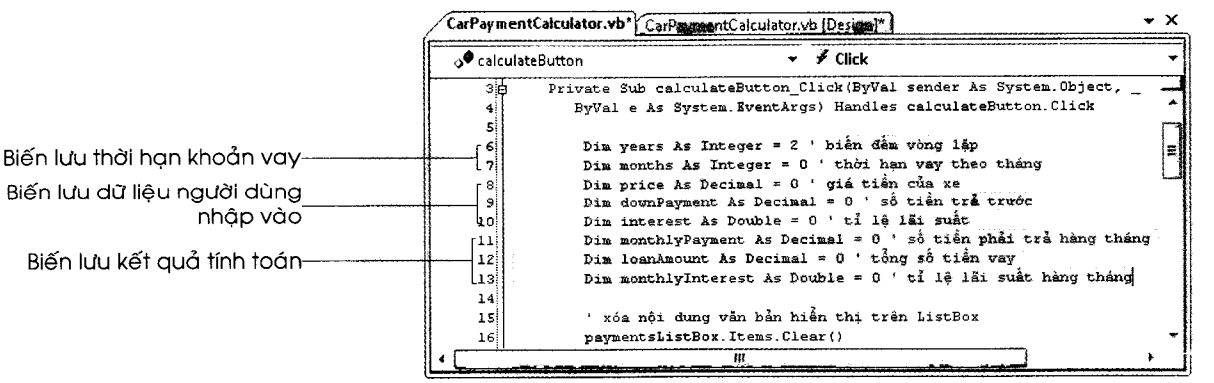
Ký hiệu & được gọi là **toán tử ghép chuỗi (string-concatenation operator)**. Toán tử này nối (hoặc ghép) hai toán hạng thành một giá trị chuỗi bằng cách thêm đoạn văn bản của toán hạng bên phải vào cuối đoạn văn bản của toán hạng bên trái. Ở dòng 10-11, tiêu đề được tạo ra bằng cách nối giá trị `"Months"` và `"Monthly Payments"` với hai hằng **`ControlChars.Tab`**. Hằng `ControlChars.Tab` chèn một ký tự tab vào chuỗi. Ứng dụng này sử dụng hai ký tự tab để phân tách giữa hai cột (Hình 9.3). [*Lưu ý*: Trong thư viện .NET Framework Class Library, kiểu `ControlChars` cung cấp hằng cho các ký tự đặc biệt, gồm ký tự `Tab`, `CrLf` và `Newline`. Những hằng này có hằng Visual Basic tương ứng, như `vbTab`, `vbCrLf`, `vbNewline`.]

4. ***Lưu project***. Chọn **File > Save All** để lưu mã đã được sửa đổi.

Bạn đã học cách thay đổi nội dung của `ListBox`, bây giờ bạn cần khai báo biến và lấy giá trị do người dùng nhập vào để thực hiện tính toán. Phần dưới đây chỉ cho bạn cách khởi tạo các biến của ứng dụng **Car Payment Calculator**. Ngoài ra, còn hướng dẫn bạn chuyển tỷ lệ lãi suất hàng năm thành tỷ lệ lãi suất hàng tháng và cách tính tổng số tiền vay.

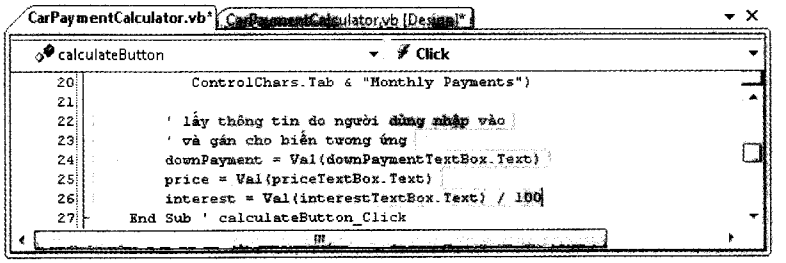
Khai báo biến và lấy thông tin do người dùng nhập vào

1. ***Khai báo biến***. Thêm dòng 6-13 trên Hình 9.10 vào ứng dụng ở phía trên đoạn mã bạn vừa thêm ở phần trước. Biến `years` và `months` lưu thời hạn vay tính theo năm và theo tháng vì tính toán yêu cầu thời hạn vay theo tháng, nhưng điều kiện tiếp tục vòng lặp lại tùy thuộc vào số năm. Biến `price`, `downPayment` và `interest` lưu dữ liệu người dùng nhập vào `TextBox`. Biến `monthlyPayment` lưu kết quả tính toán số tiền phải trả hàng tháng. Các biến `loanAmount` và `monthlyInterest` lưu kết quả tính toán.



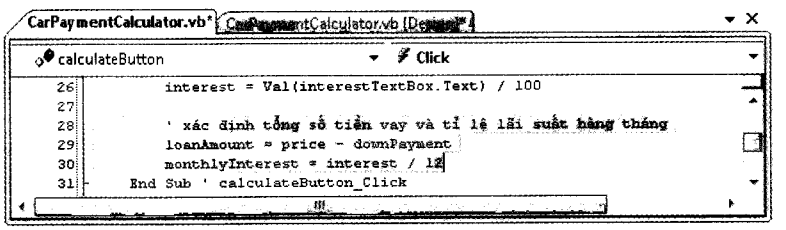
Hình 9.10 Các biến của ứng dụng **Car Payment Calculator**.

2. ***Lấy thông tin do người dùng nhập vào cần cho việc tính toán.*** Thêm dòng 22-26 trên Hình 9.11 vào bên dưới đoạn mã bạn đã thêm vào ở phần trước. Dòng 24-26 lấy số tiền trả trước (downPayment), giá (`price`) và tỷ lệ lãi suất hàng năm (`interest`) do người dùng nhập vào. Chú ý rằng dòng 26 chia tỷ lệ lãi suất cho 100 để thu được giá trị thập phân tương đương (ví dụ, 5% trở thành 0,05).



Hình 9.11 Lấy thông tin do người dùng nhập vào ứng dụng **Car Payment Calculator.**

3. ***Tính các giá trị được sử dụng trong tính toán.*** Ứng dụng tính tổng số tiền vay bằng cách lấy hiệu giữa giá tiền xe và số tiền trả trước. Thêm dòng 28-30 trên Hình 9.12 để tính tổng số tiền được vay (dòng 29) và tỷ lệ lãi suất hàng tháng (dòng 30). Các phép tính này chỉ cần thực hiện một lần nên chúng được đặt trước lệnh lặp `Do While...Loop`. Biến `loanAmount` và `monthlyInterest` được sử dụng để tính số tiền phải trả hàng tháng. Bạn thực hiện phép tính này ở phần ngay sau đây.



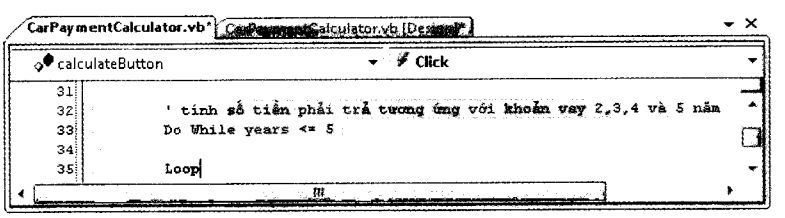
Hình 9.12 Tính tổng số tiền vay và tỷ lệ lãi suất hàng tháng.

4. ***Lưu project.*** Chọn **File > Save All** để lưu mã đã được sửa đổi.

Tiếp theo, bạn sẽ thêm lệnh lặp vào ứng dụng để tính số tiền phải trả hàng tháng cho bốn khoản vay theo các kỳ hạn khác nhau. Lệnh lặp này thực hiện tính toán cho các khoản vay kéo dài hai, ba, bốn, năm năm.

Tính số tiền phải trả hàng tháng sử dụng lệnh lặp Do While...Loop

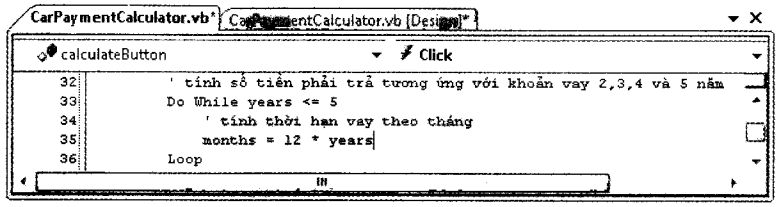
1. ***Thiết lập điều kiện tiếp tục vòng lặp.*** Thêm dòng 32-33 trên Hình 9.13 vào dưới dòng tính toán số tổng số tiền vay (`loanAmount`) và tỷ lệ lãi suất hàng tháng (`monthlyInterest`). Sau khi bạn viết mã của dòng 33 và bấm phím *Enter*, IDE đóng lệnh lặp bằng cách thêm từ khóa `Loop` ở dòng 35.



Hình 9.13 Điều kiện tiếp tục vòng lặp.

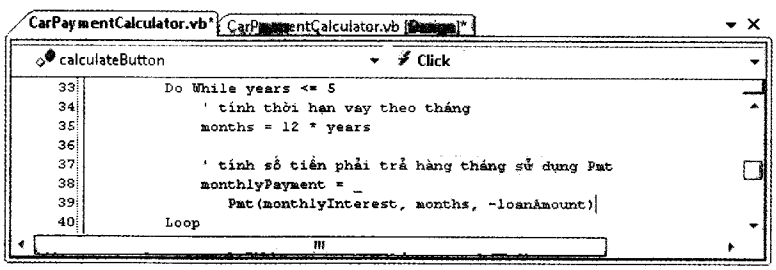
Nhớ lại rằng khoản vay ngắn hạn nhất trong ứng dụng này kéo dài hai năm, bởi vậy bạn khởi tạo biến `years` với giá trị 2 ở dòng 6 (Hình 9.10). Điều kiện tiếp tục vòng lặp (`years <=5`) trong Hình 9.13 chỉ ra rằng lệnh lặp `Do While...Loop` thực thi trong khi giá trị của biến `years` vẫn còn nhỏ hơn hoặc bằng 5. Vòng lặp này là một ví dụ về lệnh lặp được điều khiển bởi **biến đếm** (**counter-controlled repetition**). Kỹ thuật này sử dụng một biến gọi là **biến đếm** (**counter**) - trong trường hợp này là `years` - để kiểm soát số lần thực thi một tập hợp lệnh. Lệnh lặp được điều khiển bởi biến đếm còn được gọi là **lệnh lặp hữu hạn** (**definite repetition**) bởi vì số lần lặp được biết trước khi lệnh lặp bắt đầu thực thi. Trong ví dụ này, việc lặp lại kết thúc khi biến đếm (`years`) vượt quá giá trị 5.

2. ***Tính kỳ hạn thanh toán của khoản vay***. Thêm dòng 34-35 trên Hình 9.14 vào lệnh lặp `Do While...Loop` để tính kỳ hạn thanh toán (thời hạn của khoản vay tính theo tháng). Số tháng thay đổi qua mỗi lần lặp lại vòng lặp và kết quả tính toán thay đổi theo kỳ hạn thanh toán. Biến `months` nhận giá trị 24, 36, 48 và 60 trong mỗi lần lặp thành công.



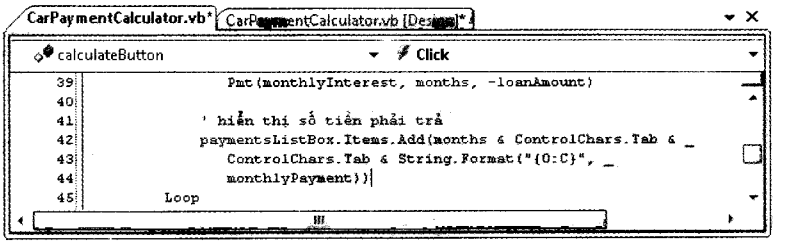
Hình 9.14 Chuyển thời gian vay tính theo đơn vị năm sang đơn vị tháng.

3. ***Tính số tiền phải trả hàng tháng***. Thêm dòng 37-39 trên Hình 9.15 vào lệnh lặp `Do While...Loop`. Dòng 38-39 (Hình 9.15) sử dụng hàm **`Pmt`** (có sẵn trong Visual Basic) trả lại giá trị kiểu `Double` cho biết số tiền phải trả hàng tháng của khoản vay có tỷ lệ lãi suất (`monthlyInterest`) không thay đổi trong khoảng thời gian (`months`) đã cho. Dòng 39 truyền tham số tỷ lệ lãi suất, thời hạn thanh toán (tính theo số tháng của kỳ hạn thanh toán) và tổng số tiền cho vay vào hàm `Pmt`. Chú ý truyền giá trị âm vào tổng số tiền cho vay. Số tiền cho vay được biểu diễn bởi giá trị âm, bởi vì chúng biểu diễn lượng tiền mặt bị lấy đi khỏi cá nhân hoặc tổ chức cho vay tiền. Giá trị kiểu `Double` do hàm `Pmt` trả lại được chuyển thành kiểu `Decimal` khi bạn gán nó cho biến `monthlyPayment` kiểu `Decimal`.



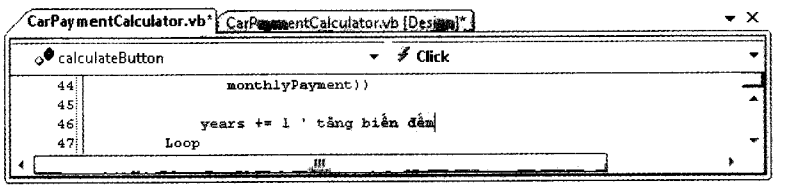
Hình 9.15 Hàm `Pmt` trả lại số tiền phải trả hàng tháng.

4. ***Hiển thị số tiền phải trả hàng tháng***. Thêm dòng 41-44 trên Hình 9.16 vào ứng dụng. Thời hạn phải trả tiền theo tháng và số lượng tiền phải trả mỗi tháng được hiển thị ở phía dưới dòng tiêu đề trên `ListBox`. Để thêm nội dung vào `ListBox`, gọi phương thức `Add` (dòng 42-44 trên Hình 9.16). Dòng 43-44 sử dụng phương thức `String.Format` để hiện thị giá trị `monthlyPayment` dưới định dạng tiền tệ. Chú ý rằng hai ký tự tab đảm bảo rằng số tiền phải trả hàng tháng được hiển thị ở cột thứ hai. Khoảng trắng được tạo ra bởi các ký tự tab làm cho kết quả của ứng dụng dễ đọc hơn.



Hình 9.16 Hiển thị số tháng và số tiền phải trả mỗi tháng.

5. ***Tăng biến đếm***. Thêm dòng 46 trên Hình 9.17 vào trước từ khóa kết thúc lệnh lặp. Dòng 46 tăng giá trị biến đếm (`years`). Biến `years` được tăng sau mỗi lần lặp cho đến tận khi nó bằng 6. Lúc này điều kiện tiếp tục vòng lặp (`year <= 5`) đạt giá trị `False` và kết thúc vòng lặp.



Hình 9.17 Tăng biến đếm.

6. ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng. Ứng dụng tính và hiển thị số tiền phải trả hàng tháng. Nhập các giá trị: giá xe, số tiền trả trước và tỷ lệ lãi suất hàng năm, sau đó nhấn vào `Button` **Calculate** để kiểm tra xem ứng dụng đã tính toán kết quả chính xác chưa.
7. ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.
8. ***Đóng IDE***. Đóng cửa sổ IDE Visual Basic bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

Hình 9.18 biểu diễn đoạn mã của ứng dụng **Car Payment Calculator**. Các dòng mã chứa các khái niệm lập trình mới mà bạn vừa học trong chương này được đánh dấu.

```
1  Public Class CarPaymentCalculatorForm
2      ' xử lý sự kiện Click cho Calculate Button
3      Private Sub calculateButton_Click(ByVal sender As System.Object,_
4      ByVal e As System.EventArgs) Handles calculateButton.Click
5
6          Dim years As Integer = 2 ' biến đếm vòng lặp
7          Dim months As Integer = 0 ' thời hạn vay theo tháng
8          Dim price As Decimal = 0 ' giá tiền của xe
9          Dim downPayment As Decimal = 0 ' số tiền trả trước
10         Dim interest As Double = 0 ' tỷ lệ lãi suất
11         Dim monthlyPayment As Decimal = 0 ' số tiền phải trả hàng tháng
12         Dim loanAmount As Decimal = 0 ' tổng số tiền vay
13         Dim monthlyInterest As Double = 0 ' tỷ lệ lãi suất hàng tháng
14
15         ' xóa nội dung văn bản hiển thị trên ListBox
16         paymentsListBox.Items.Clear()
17
18         ' thêm tiêu đề vào nội dung trên ListBox
19         paymentsListBox.Items.Add("Months" & ControlChars.Tab & _
20             ControlChars.Tab & "Monthly Payments")
21
22         ' lấy thông tin do người dùng nhập vào
23         ' và gán cho biến tương ứng
```

Xóa nội dung trên `ListBox` — dòng 16

Thêm tiêu đề vào `ListBox` — dòng 19

Hình 9.18 Đoạn mã ứng dụng **Car Payment Calculator**.(Phần 1/2)

```
24          downPayment = Val(downPaymentTextBox.Text)
25          price = Val(priceTextBox.Text)
26          interest = Val(interestTextBox.Text) / 100
27
28          ' xác định tổng số tiền vay và tỷ lệ lãi suất hàng tháng
29          loanAmount = price - downPayment
30          monthlyInterest = interest / 12
31
32          ' tính số tiền phải trả tương ứng với khoản vay 2,3,4 và 5 năm
33          Do While years <= 5
34              ' tính thời hạn vay theo tháng
35              months = 12 * years
36
37              ' tính số tiền phải trả hàng tháng sử dụng Pmt
38              monthlyPayment = _
39                  Pmt(monthlyInterest, months, -loanAmount)
40
41              ' hiển thị số tiền phải trả
42              paymentsListBox.Items.Add(months & ControlChars.Tab & _
43                  ControlChars.Tab & String.Format("{0:C}", _
44                  monthlyPayment))
45
46              years += 1 ' tăng biến đếm
47          Loop
48      End Sub ' calculateBu tton_Click
49  End Class ' CarPaymentCalculatorForm
```

Lệnh Do While...Loop thực hiện lặp thân lệnh khi biến years nhỏ hơn hoặc bằng 5

Tính kỳ hạn của khoản vay theo tháng

Tính số tiền phải trả mỗi tháng

Hiển thị số tháng và số tiền phải trả mỗi tháng

Tăng biến đếm years để điều kiện tiếp tục vòng lặp tiến dần đến giá trị sai

Hình 9.18 Đoạn mã ứng dụng **Car Payment Calculator**.(Phần 2/2)

TỰ ÔN TẬP

1. Lệnh lặp được điều khiển bởi biến đếm còn được gọi là_____bởi vì số lần lặp được biết trước khi bắt đầu thực thi vòng lặp.
   a) lệnh lặp hữu hạn
   b) lệnh lặp biết trước
   c) lệnh lặp theo thứ tự
   d) lệnh lặp sử dụng biến đếm
2. Dòng chữ được thêm vào `ListBox` để mô tả thông tin sẽ được hiển gọi là_____.
   a) tựa đề
   b) mở đầu
   c) tiêu đề
   d) chú giải

Đáp án: **1)** a. **2)** c.

9.5 Tổng kết

Trong chương này, bạn bắt đầu sử dụng lệnh lặp. Bạn đã sử dụng lệnh `Do While... Loop` và `Do Until...Loop` để lặp lại các hành động trong ứng dụng phụ thuộc vào điều kiện tiếp tục vòng lặp hoặc điều kiện kết thúc vòng lặp.

Lệnh lặp `Do While...Loop` thực thi chừng nào điều kiện tiếp tục vòng lặp còn có giá trị `True`. Khi điều kiện tiếp tục vòng lặp chuyển thành `Fasle`, quá trình lặp sẽ kết thúc. Vòng lặp vô hạn xảy ra khi điều kiện này không bao giờ chuyển thành `False`.

Lệnh lặp `Do Until...Loop` thực thi chừng nào điều kiện kết thúc vòng lặp còn có giá trị `False`. Khi điều kiện kết thúc vòng lặp chuyển thành `True`, quá trình lặp sẽ kết thúc. Vòng lặp vô hạn xảy ra khi điều kiện này không bao giờ chuyển thành `True`.

Bạn đã học về lệnh lặp được điều khiển bởi biến đếm, là lệnh lặp "biết" số lần lặp và tìm hiểu về biến đếm để đếm số lần lặp. Bạn đã sử dụng lệnh lặp để phát triển ứng dụng **Car Payment Calculator**. Trong ứng dụng này, bạn đã tính số tiền phải trả hàng tháng dựa vào tổng số tiền vay và tỷ lệ lãi suất cho thời hạn vay là hai, ba, bốn và năm năm.

Trong ứng dụng **Car Payment Calculator**, bạn đã sử dụng điều khiển `ListBox` để hiển thị kết quả cho các tùy chọn khoản vay mua ô tô. Bạn đã học về điều khiển `ListBox`, sử dụng để lưu trữ một danh sách các phần tử. Các phần tử có thể được lập trình để thêm vào và xóa khỏi `ListBox`. Các giá trị được thêm vào điều khiển `ListBox` bằng cách gọi phương thức `Add` của thuộc tính `Items` của điều khiển `ListBox`. Thuộc tính `Items` trả lại đối tượng chứa tất cả các giá trị được hiển thị trên `ListBox`.

Trong chương tiếp theo, bạn sẽ tìm hiểu hai lệnh lặp khác, tiếp tục khám phá lệnh lặp được điều khiển bởi biến đếm. Ứng dụng **Car Payment Calculator** minh họa một trường hợp thường dùng của các lệnh lặp - thực hiện cùng một phép tính với nhiều giá trị khác nhau. Ứng dụng tiếp theo giới thiệu một ứng dụng phổ biến khác của lệnh lặp - tính tổng một dãy số.

TỔNG KẾT KỸ NĂNG

Hiển thị giá trị lên `ListBox`

- Thuộc tính `Items` của điều khiển `ListBox` trả về đối tượng chứa các giá trị được hiển thị trên `ListBox`.
- Gọi phương thức `Add` để thêm giá trị vào thuộc tính `Items`.

Xóa nội dung hiển thị trên `ListBox`

- Phương thức `Clear` của thuộc tính `Items` xóa tất cả các giá trị có trên `ListBox`.

Lặp lại các hành động trong ứng dụng

- Sử dụng lệnh lặp phụ thuộc vào giá trị đúng hoặc sai của điều kiện tiếp tục vòng lặp hoặc điều kiện kết thúc vòng lặp.

Thực thi lệnh lặp với số lần lặp được biết trước

- Sử dụng lệnh lặp được điều khiển bởi biến đếm, trong đó biến đếm để xác định số lần thực thi một tập lệnh.

Sử dụng lệnh lặp `Do While...Loop`

- Lệnh lặp này thực thi khi điều kiện tiếp tục vòng lặp đạt giá trị `True`.
- Lặp vô hạn xảy ra khi điều kiện tiếp tục vòng lặp không bao giờ chuyển sang `False`.

Sử dụng lệnh lặp `Do Until...Loop`

- Lệnh lặp này thực thi khi điều kiện kết thúc vòng lặp đạt giá trị True.
- Lặp vô hạn xảy ra khi điều kiện kết thúc vòng lặp không bao giờ chuyển sang True.

Ghép chuỗi

- Sử dụng toán tử & để tạo ra một chuỗi mới từ hai chuỗi đã tồn tại. Nội dung toán hạng bên phải được thêm vào nội dung toán hạng bên trái để tạo ra chuỗi mới.

THUẬT NGỮ

biến đếm (counter) - Biến được dùng để xác định số lần thân lệnh lặp được thực thi.

điều khiển `ListBox` - Cho phép người dùng xem các phần tử của một danh sách. Các phần tử có thể được lập trình để thêm hoặc xóa khỏi danh sách.

điều kiện kết thúc vòng lặp (loop-termination condition) - Điều kiện được dùng trong lệnh lặp (ví dụ như lệnh `Do Until...Loop`) cho phép lệnh lặp tiếp tục khi điều kiện là `False` và lệnh lặp kết thúc khi điều kiện này có giá trị là `True`.

điều kiện tiếp tục vòng lặp (loop-continuation condition) - Điều kiện được dùng trong lệnh lặp (ví dụ như lệnh `Do While...Loop`) cho phép lệnh lặp tiếp tục khi điều kiện là `True` và lệnh lặp kết thúc khi điều kiện này có giá trị là `False`.

hàm `Pmt` - Hàm có sẵn trong Visual Basic, dựa vào tỷ lệ lãi suất, thời hạn vay và tổng số tiền vay, trả về số lượng tiền phải trả kiểu `Double`.

hằng `ControlChars.Tab` - Biểu diễn một ký tự tab.

lệnh lặp (repetition statement) - Cho phép bạn thực hiện lặp lại một hành động hoặc các hành động dựa vào giá trị điều kiện.

lệnh lặp `Do Until...Loop` - Lệnh điều khiển thực thi một tập lệnh trong phần thân lệnh lặp *cho đến khi* điều kiện kết thúc vòng lặp có giá trị `True`.

lệnh lặp `Do While...Loop` - Lệnh điều khiển thực thi một tập các lệnh trong phần thân lệnh lặp *khi* điều kiện tiếp tục vòng lặp có giá trị `True`.

lệnh lặp được điều khiển bởi biến đếm (counter-controlled repetition) - Kỹ thuật sử dụng biến đếm để xác định số lần thân lệnh lặp thực thi. Còn được gọi là lệnh lặp hữu hạn.

ký hiệu ghép (trong ngôn ngữ UML) - Ký hiệu hình thoi trong UML sử dụng để gộp hai luồng hoạt động thành một luồng.

phần thân của lệnh điều khiển - Tập các lệnh nằm bên trong lệnh điều khiển.

phương thức `Add` của `Items` - Thêm một phần tử vào điều khiển `ListBox`.

phương thức `Clear` của `Items` - Xóa nội dung hiển thị trên điều khiển `ListBox`.

thuộc tính Items của điều khiển `ListBox` - Trả lại đối tượng chứa tất cả các giá trị của `ListBox`.

tiêu đề (header) - Dòng chữ nằm ở đầu `ListBox` để làm rõ thông tin được hiển thị trên `ListBox`.

toán tử ghép chuỗi (&) - Toán tử này nối hai toán hạng thành một giá trị chuỗi.

vòng lặp (loop) - Tên gọi khác của lệnh lặp.

vòng lặp vô hạn (infinite loop) - Là lỗi xảy ra khi lệnh lặp không bao giờ kết thúc.

HƯỚNG DẪN THIẾT KẾ GIAO DIỆN

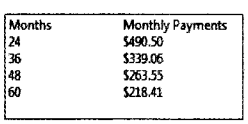
`ListBox`

- `ListBox` nên có kích thước đủ rộng để hiển thị toàn bộ nội dung hoặc đủ lớn để thanh cuộn có thể sử dụng dễ dàng.
- Nên sử dụng tiêu đề trong `ListBox` khi bạn hiển thị dữ liệu dạng bảng. Thêm tiêu đề mô tả thông tin được hiển thị trên `ListBox` sẽ giúp cho người dùng dễ hiểu hơn.

ĐIỀU KHIỂN, SỰ KIỆN, THUỘC TÍNH & PHƯƠNG THỨC

`ListBox` Điều khiển này cho phép người dùng xem và lựa chọn các phần tử trong một danh sách.

- ***Trên giao diện khi ứng dụng chạy***



- ***Thuộc tính***

  `Items` - Trả lại đối tượng chứa các phần tử được hiển thị trên `ListBox`.

  `Location` - Chỉ ra vị trí của `ListBox` trên `Form`.

  `Name` - Chỉ ra tên được sử dụng để truy cập `ListBox` trong khi lập trình. Tên nên được thêm hậu tố `ListBox` vào phía sau.

  `Size` - Chỉ ra chiều rộng và chiều cao (bằng pixel) của `ListBox`.

- ***Phương thức***

  `Items.Add` - Thêm một phần tử vào thuộc tính `Items`.

  `Items.Clear` - Xóa tất cả các giá trị có trong thuộc tính `Items` của `ListBox`.

CÂU HỎI TRẮC NGHIỆM

9.1 Lệnh_____thực thi cho đến khi điều kiện kết thúc vòng lặp đạt giá trị `True`.

a) `Do While...Loop` b) `Do Until...Loop`

c) `Do` d) `Loop`

9.2 Lệnh_____thực thi cho đến khi điều kiện tiếp tục vòng lặp đạt giá trị `False`.

a) `Do While...Loop` b) `Do Until...Loop`

c) `Do` d) `Loop`

9.3 Vòng lặp_____xảy ra khi điều kiện trong lệnh `Do While...Loop` không bao giờ có giá trị `False`.

a) vô hạn b) không xác định

c) lồng d) bất định

9.4 ____là biến giúp kiểm soát số lần thực thi tập lệnh.

a) Biến lặp b) Biến đếm

c) Vòng lặp d) Lệnh điều khiển lặp

9.5 Điều khiển_____cho phép người dùng thêm và xem các phần tử trong một danh sách.

a) `ListItems` b) `SelectBox`

c) `ListBox` d) `ViewBox`

9.6 Trong biểu đồ hoạt động UML, ký hiệu_____gộp hai luồng hoạt động thành một luồng.

a) ghép b) nối

c) trạng thái hành động d) ra quyết định

9.7 Thuộc tính_____trả về đối tượng chứa tất cả các giá trị của `ListBox`.

a) `All` b) `List`

c) `ListItemValues` d) `Items`

9.8 Phương thức_____của `Items` xóa tất cả các giá trị có trong danh sách.

a) `Remove` b) `Delete`

c) `Clear` d) `Del`

9.9 Phương thức_____của `Items` thêm một phần tử vào `ListBox`.

a) `Include` b) `Append`

c) `Add` d) Các lựa chọn trên đều sai

9.10 Hàm_____tính số tiền phải trả hàng tháng của một khoản vay dựa trên tỷ lệ lãi suất cố định.

a) `MonPmt` b) `Payment`

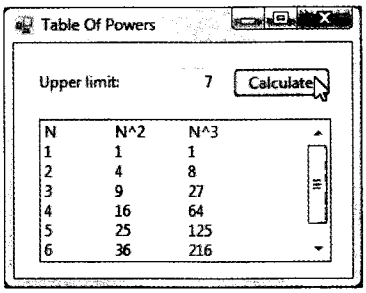
c) `MonthlyPayment` d) `Pmt`

BÀI TẬP

9.11 (***Ứng dụng Table of Powers***) Viết ứng dụng hiển thị một bảng chứa các số trong khoảng từ 1 tới một giới hạn trên, cùng với giá trị bình phương của mỗi số (ví dụ, số n lũy thừa 2 hoặc n ^ 2) và giá trị lập phương (số n lũy thừa 3 hoặc n ^ 3). Người dùng chỉ ra giới hạn trên và kết quả được hiển thị trên một `ListBox`, như Hình 9.19.

a) ***Copy template của ứng dụng vào thư mục làm việc***. Copy thư mục `C:\Examples\Tutorial09\Exercises\TableOfPowers` vào thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng***. Nhấn đúp vào file `TableOfPowers.sln` trong thư mục `TableOfPowers` để mở ứng dụng.



Hình 9.19 Giao diện ứng dụng **Table of Powers**.

c) ***Thêm ListBox***. Thêm `ListBox` vào ứng dụng, như Hình 9.19. `ListBox` này có tên là `resultsListBox`.

d) ***Thêm xử lý sự kiện cho TextBox Upper limit:***. Nhấn đúp vào `TextBox` **Upper limit:** để tạo ra xử lý sự kiện cho sự kiện `TextChanged` của `TextBox`. Trong xử lý sự kiện này, xóa nội dung có trong `ListBox`.

e) ***Thêm xử lý sự kiện cho Button Calculate***. Nhấn đúp vào `Button` **Calculate** để tạo ra xử lý sự kiện `calculateButton_Click` rỗng. Thêm đoạn mã được chỉ ra trong các bước còn lại vào xử lý sự kiện này.

f) ***Xóa nội dung trên ListBox***. Sử dụng phương thức `Clear` của thuộc tính `Items` để xóa dữ liệu của `ListBox` này.

g) ***Lấy giá trị giới hạn trên được cung cấp bởi người dùng***. Gán giá trị do người dùng nhập vào `TextBox` **Upper limit:** cho một biến. Chú ý rằng thuộc tính `Name` của `TextBox` này được thiết lập giá trị là `inputTextBox`.

h) ***Thêm tiêu đề***. Sử dụng phương thức `Add` của thuộc tính `Items` để thêm tiêu đề vào `ListBox`. Tiêu đề của ba cột như sau - `N`, `N^2` , `N^3`. Tiêu đề cột nên được phân tách bởi các ký tự tab.

i) ***Tính lũy thừa từ 1 tới giới hạn trên***. Sử dụng `Do Until...Loop` để tính giá trị bình phương và lập phương của mỗi số trong khoảng từ 1 đến giới hạn trên. Thêm một phần tử vào `ListBox` để chứa số hiện tại, giá trị bình phương và giá trị lập phương của số này.

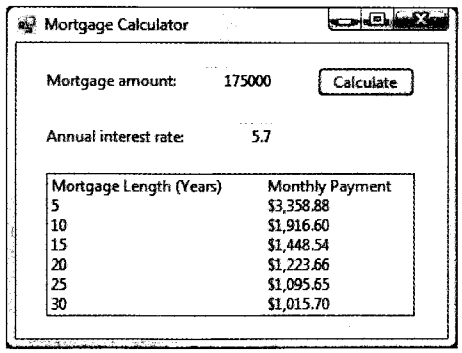
j) ***Tăng biến đếm***. Nhớ tăng biến đếm trong mỗi lần lặp.

k) ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng của bạn. Nhập giới hạn trên và nhấn vào `Button` **Calculate**. Kiểm tra xem bảng lũy thừa đã hiển thị đúng giá trị chưa.

l) ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

m) ***Đóng IDE***. Đóng cửa sổ IDE Visual Basic bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

9.12 (***Ứng dụng Mortgage Calculator***) Ngân hàng đưa ra các khoản vay thế chấp có thể hoàn trả trong vòng 5, 10, 15, 20, 25 hoặc 30 năm. Viết ứng dụng cho phép người dùng nhập vào giá của ngôi nhà (tổng số tiền thế chấp) và tỷ lệ lãi suất hàng năm. Khi người dùng nhấn vào `Button` **Calculate**, ứng dụng hiển thị một bảng chứa thời hạn vay thế chấp với số tiền phải trả hàng tháng, như Hình 9.20.



Hình 9.20 Giao diện ứng dụng **Mortgage Calculator**.

a) ***Copy template của ứng dụng vào thư mục làm việc***. Copy thư mục `C:\Examples\Tutorial09\Exercises\MortgageCalculator` vào thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng***. Nhấn đúp vào file `MortgageCalculator.sln` trong thư mục `MortgageCalculator` để mở ứng dụng.

c) ***Thêm `ListBox`***. Thêm `ListBox` vào ứng dụng, như Hình 9.20. `ListBox` này có tên là `resultsListBox`.

d) ***Thêm xử lý sự kiện cho Button Calculate***. Nhấn đúp vào `Button` **Calculate** để tạo ra xử lý sự kiện `calculateButton_Click` rỗng. Thêm đoạn mã được chỉ ra trong các bước còn lại vào xử lý sự kiện này.

e) ***Chuyển tỷ lệ lãi suất hàng năm sang tỷ lệ lãi suất hàng tháng***. Để chuyển tỷ lệ lãi suất hàng năm từ một giá trị phần trăm thành giá trị tương đương kiểu `Double`, chia tỷ lệ lãi suất hàng năm cho `100`. Sau đó chia tỷ lệ lãi suất kiểu `Double` này cho `12` để thu được tỷ lệ lãi suất hàng tháng.

f) ***Xóa nội dung hiển thị trên `ListBox`***. Sử dụng phương thức `Clear` của thuộc tính `Items` để xóa dữ liệu có sẵn trên `ListBox`.

g) ***Hiển thị tiêu đề***. Sử dụng phương thức `Add` để hiển thị tiêu đề trên `ListBox`. Tiêu đề cột là `"Mortgage Length (Years)"` và `"Monthly Payment"`, được phân tách bởi nhau một ký tự tab.

h) ***Sử dụng lệnh lặp***. Thêm lệnh `Do While...Loop` để tính sáu tùy chọn số tiền phải trả hàng tháng cho khoản vay thế chấp của người dùng. Mỗi tùy chọn có khoảng thời gian vay thế chấp khác nhau. Trong ví dụ này, các khoảng thời gian vay thế chấp là `5`, `10`, `15`, `20`, `25` và `30` năm.

i) ***Chuyển thời gian thế chấp tính theo năm sang tính theo tháng***. Chuyển số năm thành số tháng.

j) ***Tính số tiền phải trả hàng tháng cho sáu tùy chọn thế chấp khác nhau***. Sử dụng hàm `Pmt` để tính số tiền phải trả hàng tháng. Truyền tham số tỷ lệ lãi suất hàng tháng, số tháng vay thế chấp, tổng số tiền vay thế chấp vào hàm này. Nhớ rằng số tiền vay thế chấp phải là số âm vì nó biểu diễn một khoản tiền được trả bởi người cho vay.

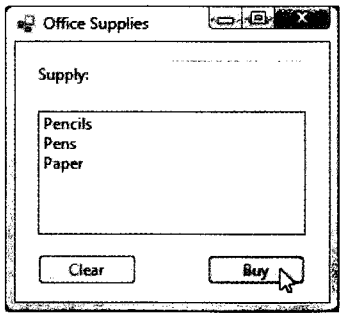
k) ***Hiển thị các kết quả***. Sử dụng phương thức `Add` của thuộc tính `Items` để hiển thị thời hạn khoản vay theo năm cùng với số tiền phải trả hàng tháng trên `ListBox`. Sử dụng ba ký tự tab để đảm bảo rằng số tiền phải trả hàng tháng được hiển thị ở cột thứ hai.

l) ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng của bạn. Nhập vào tổng số tiền vay thế chấp và tỷ lệ lãi suất hàng năm, sau đó nhấn vào `Button` **Calculate**. Kiểm tra xem số tiền phải trả hàng tháng được hiển thị đúng kết quả chưa.

m) ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

n) ***Đóng IDE***. Đóng cửa sổ IDE Visual Basic bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

9.13 (***Ứng dụng Office Supplies***) Tạo ứng dụng cho phép người dùng tạo danh sách các mặt hàng văn phòng phẩm cần phải mua, như Hình 9.21. Người dùng nhập các mặt hàng vào `TextBox` và nhấn vào `Button` **Buy** để thêm vào `ListBox`. `Button` **Clear** xóa bỏ tất các mặt hàng có trên danh sách.



Hình 9.21 Giao diện ứng dụng **Office Supplies**.

a) ***Copy template của ứng dụng vào thư mục làm việc***. Copy thư mục `C:\Examples\Tutorial09\Exercises\OfficeSupplies` tới thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng***. Nhấn đúp vào file `OfficeSupplies.sln` trong thư mục `OfficeSupplies` để mở ứng dụng.

c) ***Thêm `ListBox`***. Thêm một `ListBox` vào ứng dụng. `ListBox` này có tên là `suppliesListBox`. Thiết lập kích thước `ListBox` giống như trên Hình 9.21.

d) ***Thêm xử lý sự kiện cho `Button` Buy***. Nhấn đúp vào `Button` **Buy** để tạo ra xử lý sự kiện `buyButton_Click`. Xử lý sự kiện này lấy giá trị do người dùng nhập vào `TextBox`. Sau đó, dữ liệu này được lưu như là một phần tử trong `ListBox`. Sau khi giá trị do người dùng nhập vào được thêm vào `ListBox`, xóa nội dung trên `TextBox` **Supply:**.

e) ***Thêm xử lý sự kiện cho `Button` Clear***. Nhấn đúp vào `Button` **Clear** để tạo ra xử lý sự kiện `clearButton_Click`. Xử lý sự kiện này sử dụng phương thức `Clear` của thuộc tính `Items` để xóa `ListBox`.

f) ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng của bạn. Nhập một vài item vào `TextBox` **Supply:** và nhấn vào `Button` **Buy** sau mỗi lần nhập một mặt hàng. Kiểm tra xem từng mặt hàng này đã được thêm vào `ListBox` chưa. Nhấn vào `Button` **Clear** và kiểm tra xem tất cả các mặt hàng đã bị xóa khỏi `ListBox` chưa.

g) ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

h) ***Đóng IDE***. Đóng cửa sổ IDE Visual Basic bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

Đoạn mã này làm gì? ▶ **9.14** Kết quả của đoạn mã dưới đây là gì?

```
1  Dim x As Integer = 1
2  Dim mysteryValue As Integer = 1
3
4  Do While x < 6
5     mysteryValue *= x
6     x += 1
7  Loop
8
9  displayLabel.Text = mysteryValue
```

Đoạn mã này có gì sai? ▶ **9.15** Tìm lỗi sai trong đoạn mã dưới đây:

a) Giả sử biến x được khai báo và khởi tạo giá trị là 1. Vòng lặp tính tổng các số từ 1 đến 10.

```
1  Dim total As Integer = 0
2
3  Do Until x <= 10
4     total += x
5     x += 1
6  Loop
```

b) Giả sử biến `counter` được khai báo và khởi tạo giá trị là 1. Vòng lặp tính tổng các số từ 1 đến 100.

```
1  Do While counter <= 100
2     total += counter
3  Loop
4
5  counter += 1
```

c) Giả sử biến `counter` được khai báo và khởi tạo giá trị là 1000. Vòng lặp lặp từ 1000 đến 1.

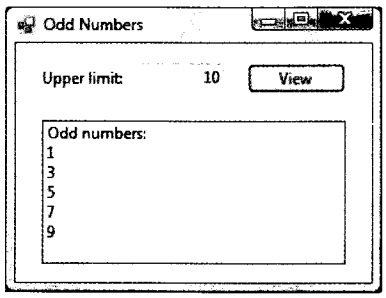
```
1  Do While > 0
2     numbersListBox.Items.Add(counter)
3     counter += 1
4  Loop
```

d) Giả sử rằng biến `counter` được khai báo và khởi tạo giá trị là 1. Vòng lặp thực thi năm lần, thêm các số từ 1-5 vào `ListBox`.

```
1  Do While counter < 5
2     numbersListBox.Items.Add(counter)
3     counter += 1
4  Loop
```

Sử dụng trình gỡ lỗi

▶ **9.16** (***Ứng dụng Odd Numbers***) Ứng dụng **Odd Numbers** hiển thị tất cả các số nguyên lẻ từ 1 tới một số do người dùng nhập vào. Copy ứng dụng **Odd Numbers** từ `C:\Examples\Tutorial09\Exercises\Debugger` vào thư mục làm việc của bạn. Chạy ứng dụng. Chú ý rằng vòng lặp vô hạn xảy ra sau khi bạn nhập giá trị vào `TextBox` **Upper limit:** và nhấn vào `Button` **View**. Chọn **Debug > Stop Debug** để đóng ứng dụng đang chạy. Sử dụng trình gỡ lỗi để tìm và sửa lỗi trong ứng dụng. Hình 9.22 hiển thị kết quả đúng của ứng dụng.

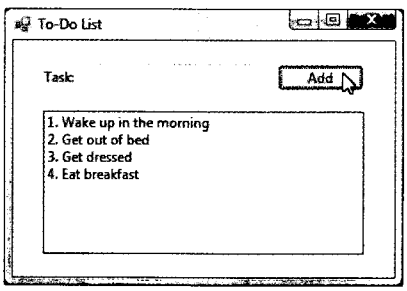


Hình 9.22 Kết quả đúng của ứng dụng **Odd Numbers**.

Bài tập nâng cao

▶ **9.17** (***Ứng dụng To-Do List***) Sử dụng `ListBox` như là một danh sách những việc phải làm. Nhập từng công việc vào `TextBox` và thêm chúng vào `ListBox` bằng cách nhấn vào `Button`. Các công việc được hiển thị trong một danh sách được đánh số, như Hình 9.23. Để làm việc này, chúng ta sẽ làm quen với thuộc tính `Count`, trả lại số lượng phần tử của thuộc tính `Items` của `ListBox`. Dưới đây là một ví dụ gọi và gán số lượng phần tử được hiển thị trong `sampleListBox` cho một biến `Integer`:

```
Count = sampleListBox.Items.Count
```



Hình 9.23 Giao diện ứng dụng **To-Do List**.

CHƯƠNG 10

Ứng dụng Class Average

Giới thiệu lệnh lặp Do...Loop While và Do...Loop Until

Mục tiêu

Trong chương này, bạn sẽ tìm hiểu để:

- Sử dụng lệnh **Do... Loop While**.
- Sử dụng lệnh **Do... Loop Until**.
- Tìm hiểu thêm về lệnh lặp được điều khiển bởi biến đếm.
- Chuyển focus tới một điều khiển.
- Kích hoạt hay vô hiệu hóa **Button**.

Nội dung chính

Trong chương này chúng ta sẽ tiếp tục thảo luận về các lệnh lặp đã đề cập ở Chương 9. Trong chương trước, chúng ta đã tìm hiểu về lệnh lặp `Do While... Loop` và `Do Until...Loop`, các lệnh này kiểm tra điều kiện tiếp tục vòng lặp và điều kiện kết thúc vòng lặp trước khi thực hiện lặp. Trong chương này, bạn sẽ được giới thiệu thêm hai lệnh lặp nữa là `Do...Loop While` và `Do...Loop Until`. Hai lệnh này kiểm tra điều kiện *sau* mỗi lần lặp. Do đó, thân của các lệnh lặp này được thực thi ít nhất một lần.

Bạn cũng sẽ học cách vô hiệu hóa hay kích hoạt điều khiển trên `Form`. Khi điều khiển (ví dụ như `Button`) bị vô hiệu hóa thì điều khiển này không còn hồi đáp lại thao tác của người dùng nữa. Bạn sử dụng tính năng này để ngăn người dùng gây ra lỗi trong ứng dụng. Chương này cũng giới thiệu các khái niệm chuyển focus của ứng dụng cho một điều khiển. Sử dụng focus hợp lý sẽ làm cho ứng dụng dễ sử dụng hơn.

10.1 Chạy thử ứng dụng Class Average

Ứng dụng này phải đáp ứng những yêu cầu sau:

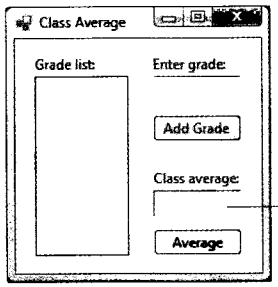
Yêu cầu đối với ứng dụng

Một giáo viên thường đưa ra các câu hỏi cho một lớp học gồm 10 sinh viên. Điểm của mỗi câu hỏi là một số nguyên trong khoảng từ 0 tới 100 (kể cả hai giá trị 0 và 100). Giáo viên này muốn bạn phát triển ứng dụng để tính điểm trung bình của cả lớp cho mỗi câu hỏi.

Điểm trung bình của cả lớp bằng tổng số điểm chia cho số sinh viên tham gia trả lời câu hỏi. Giải thuật giải quyết vấn đề này là nhập vào từng điểm số, tính tổng số điểm, thực hiện tính trung bình và hiển thị kết quả. Bạn bắt đầu bằng việc chạy thử ứng dụng đã hoàn thiện. Sau đó, bạn tìm hiểu những tính năng bổ sung cần thiết của Visual Basic để xây dựng cho mình một phiên bản của ứng dụng này.

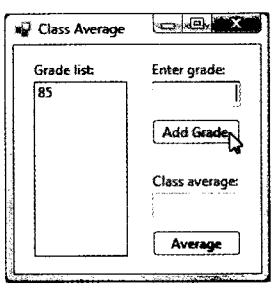
Chạy thử ứng dụng Class Average

1. ***Mở ứng dụng đã hoàn thiện.*** Mở thư mục `C:\Examples\Tutorial10\CompletedApplication\ClassAverage` để tìm ứng dụng **Class Average**. Nhấn đúp vào file `ClassAverage.sln` để mở ứng dụng trong IDE Visual Basic.
2. ***Chạy ứng dụng Class Average.*** Chọn **Debug > Start Debugging** để chạy ứng dụng này (xem Hình 10.1).



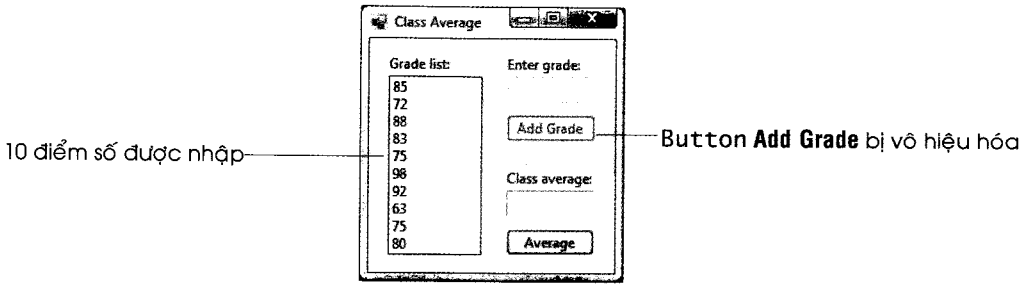
Hình 10.1 Giao diện ứng dụng **Class Average** đang chạy.

3. ***Nhập điểm cho mỗi câu hỏi.*** Nhập giá trị 85 là điểm số đầu tiên vào `TextBox` **Enter Grade:** và nhấn vào `Button` **Add Grade**. Điểm vừa nhập sẽ được thêm vào `ListBox`, như Hình 10.2. Sau khi bạn nhấn vào `Button` **Add Grade**, con trỏ xuất hiện ở `TextBox` **Enter grade:**. Khi một điều khiển được chọn (ví dụ, `TextBox` **Enter grade:**), thì ta gọi điều khiển đó nhận được **focus** của ứng dụng. Bạn sẽ học thiết lập focus cho điều khiển khi bạn xây dựng ứng dụng của chương này. Như bạn thấy, focus của ứng dụng chuyển sang `TextBox` **Enter grade:**, bạn có thể nhập một điểm khác mà không cần trỏ chuột vào `TextBox` này hoặc không cần bấm phím *tab*. Chuyển focus tới một điều khiển cụ thể sẽ cho người dùng biết thông tin nào được mong đợi tiếp theo. [*Lưu ý*: Nếu bạn nhấn vào `Button` **Average** trước khi 10 điểm được nhập, một lỗi thực thi sẽ xảy ra. Chọn **Debug > Stop Debugging** để đóng cửa sổ ứng dụng. Lặp lại bước 2. Bạn sẽ sửa lỗi này trong phần bài tập].

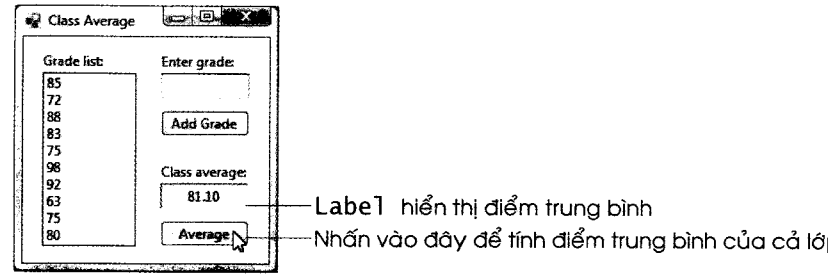


Hình 10.2 Nhập điểm vào ứng dụng **Class Average**.

4. ***Lặp lại* Bước 3 *chín lần.*** Nhập chín điểm khác trong khoảng từ 0 đến 100, nhấn vào `Button` **Add Grade** sau mỗi lần nhập. Sau khi có 10 điểm được hiển thị trong `ListBox` **Grade list:**, giao diện sẽ giống như Hình 10.3. Chú ý rằng `Button` **Add Grade** bị vô hiệu hóa sau khi bạn nhập đủ 10 điểm. Màu của `Button` này sẽ chuyển sang màu xám và khi nhấn vào `Button` này sẽ không gọi xử lý sự kiện thực thi lệnh.
5. ***Tính điểm trung bình của cả lớp.*** Nhấn vào `Button` **Average** để tính điểm trung bình của 10 câu hỏi. Điểm trung bình của cả lớp được hiển thị trên `Label` phía trên `Button` **Average** (Hình 10.14). Chú ý rằng `Button` **Add Grade** bây giờ đang ở trạng thái kích hoạt.

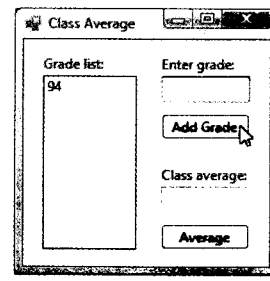


Hình 10.3 Ứng dụng **Class Average** sau khi 10 điểm số được nhập.



Hình 10.4 Hiển thị điểm trung bình của cả lớp.

6. ***Nhập một tập điểm khác***. Bạn có thể tính điểm trung bình của cả lớp với một tập 10 điểm khác mà không cần chạy lại ứng dụng. Nhập một điểm vào `TextBox` và nhấn vào `Button` **Add Grade**. Chú ý rằng `ListBox` **Grade list:** và trường **Class average:** bị xóa khi bạn nhập một tập điểm khác (Hình 10.5).



Hình 10.5 Nhập một tập điểm mới.

7. ***Đóng ứng dụng.*** Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.
8. ***Đóng IDE.*** Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

10.2 Lệnh lặp Do...Loop While

Lỗi lập trình thường gặp

Vòng lặp vô hạn xảy ra khi điều kiện tiếp tục vòng lặp trong lệnh `Do...Loop While` không bao giờ có giá trị `False`.

Lệnh lặp **`Do...Loop While`** giống với lệnh lặp `Do While...Loop`, cả hai lệnh này đều lặp khi điều kiện tiếp tục vòng lặp có giá trị `True`. Trong lệnh `Do While...Loop`, điều kiện tiếp tục vòng lặp được kiểm tra tại thời điểm bắt đầu vòng lặp, trước khi thân vòng lặp được thực hiện. Lệnh lặp `Do...Loop While` kiểm tra điều kiện tiếp tục vòng lặp *sau khi* thân vòng lặp thực hiện. Do vậy, trong lệnh lặp `Do...Loop While`, thân vòng lặp luôn luôn thực thi ít nhất một lần. Khi lệnh `Do...Loop While` kết thúc, ứng dụng tiếp tục thực thi lệnh tiếp theo sau từ khóa `Loop While`.

Để minh họa lệnh lặp Do...Loop While, hãy xem ví dụ đóng gói va li dưới đây. Trước khi bạn bắt đầu đóng gói, va li rỗng. Bạn đặt các vật vào va li, sau đó xác định xem va li đã đầy chưa. Chừng nào va li chưa đầy, bạn tiếp tục đặt các vật vào trong va li. Xét ví dụ về lệnh Do...Loop While được biểu diễn bởi đoạn mã bên dưới. Đoạn mã này hiển thị các số từ 1 đến 3 trên ListBox:

```
Dim counter As Integer = 1

Do
   displayListBox.Items.Add(counter)
   counter += 1
Loop While counter <= 3
```

Đoạn ứng dụng này khởi tạo giá trị 1 cho biến counter. Điều kiện tiếp tục vòng lặp của lệnh Do...Loop While là counter <= 3. Khi điều kiện tiếp tục vòng lặp là True, lệnh Do...Loop While thực thi. Khi điều kiện tiếp tục vòng lặp chuyển sang False (khi counter lớn hơn 3), lệnh Do...Loop While hoàn thành thực thi và displayListBox chứa các số từ 1 đến 3. Phần dưới đây mô tả từng bước lệnh lặp trên thực thi.

Thực thi lệnh lặp Do...Loop While

1. Ứng dụng khai báo biến counter và gán giá trị 1 cho biến này.
2. Ứng dụng chuyển đến thực thi lệnh lặp Do...Loop While.
3. Giá trị hiện đang được lưu trong biến counter (trong trường hợp này là 1) được thêm vào thuộc tính Items của displayListBox.
4. Giá trị biến counter được tăng thêm 1, lúc này biến counter có giá trị là 2.
5. Điều kiện tiếp tục vòng lặp được kiểm tra. Điều kiện này có giá trị là True (counter nhỏ hơn hoặc bằng 3), bởi vậy ứng dụng tiếp tục thực thi lệnh đầu tiên sau lệnh Do.
6. Giá trị hiện đang được lưu trong biến counter (trong trường hợp này là 2) được thêm vào thuộc tính Items của displayListBox.
7. Giá trị biến counter được tăng thêm 1, lúc này biến counter có giá trị là 3.
8. Điều kiện tiếp tục vòng lặp được kiểm tra. Điều kiện này có giá trị là True (counter nhỏ hơn hoặc bằng 3), bởi vậy ứng dụng tiếp tục thực thi lệnh đầu tiên sau lệnh Do.
9. Giá trị hiện đang được lưu trong biến counter (trong trường hợp này là 3) được thêm vào thuộc tính Items của displayListBox.
10. Giá trị biến counter được tăng thêm 1, lúc này biến counter có giá trị là 4.
11. Điều kiện tiếp tục vòng lặp được kiểm tra. Điều kiện này có giá trị là False (counter không nhỏ hơn hoặc bằng 3), bởi vậy ứng dụng thoát khỏi lệnh lặp Do...Loop While.

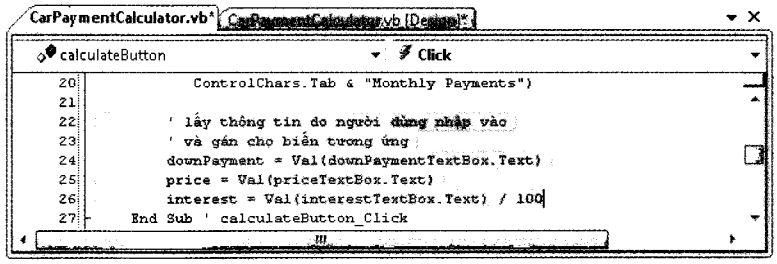
Mẹo tránh lỗi

Lựa chọn giá trị cuối cùng của điều kiện và toán tử quan hệ thích hợp có thể làm giảm khả năng xảy ra lỗi kết thúc vòng lặp sớm hoặc muộn. Ví dụ, trong lệnh Do While...Loop sử dụng để hiển thị lên màn hình các giá trị từ 1-10, điều kiện tiếp tục vòng lặp là counter <= 10 mà không phải là counter < 10 (điều kiện này sẽ gây ra lỗi kết thúc vòng lặp sớm) hay counter < 11 (điều kiện này đúng nhưng không rõ ràng).

Nếu bạn gõ nhầm điều kiện tiếp tục vòng lặp là counter < 3 hoặc counter <= 2, ListBox sẽ chỉ hiển thị giá trị 1 và 2. Toán tử quan hệ không đúng (dấu nhỏ hơn trong điều kiện counter < 3) hay giá trị cuối cùng của biến đếm vòng lặp không đúng (giá trị 2 trong counter <= 2) trong lệnh lặp, hay khởi tạo sai giá trị ban đầu (counter = 0) có thể gây ra lỗi **kết thúc vòng lặp sớm hoặc muộn** (**off-by-one error**). Lỗi này xảy ra khi vòng lặp thực thi ít hơn hoặc nhiều hơn một lần lặp so với số lần cần thiết.

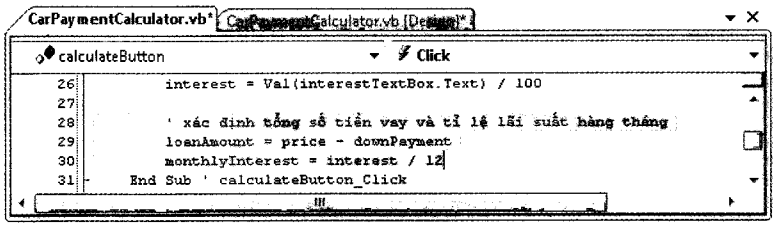
Hình 10.6 minh họa biểu đồ hoạt động UML cho lệnh Do...Loop While. Biểu đồ này rõ ràng chỉ ra rằng điều kiện canh giữ tiếp tục vòng lặp ([counter <= 3]) không được đánh giá cho đến tận sau khi vòng lặp thực hiện trạng thái hành động

2. ***Lấy thông tin do người dùng nhập vào cần cho việc tính toán.*** Thêm dòng 22-26 trên Hình 9.11 vào bên dưới đoạn mã bạn đã thêm vào ở phần trước. Dòng 24-26 lấy số tiền trả trước (`downPayment`), giá (`price`) và tỷ lệ lãi suất hàng năm (`interest`) do người dùng nhập vào. Chú ý rằng dòng 26 chia tỷ lệ lãi suất cho 100 để thu được giá trị thập phân tương đương (ví dụ, 5% trở thành 0,05).



Hình 9.11 Lấy thông tin do người dùng nhập vào ứng dụng **Car Payment Calculator.**

3. ***Tính các giá trị được sử dụng trong tính toán.*** Ứng dụng tính tổng số tiền vay bằng cách lấy hiệu giữa giá tiền xe và số tiền trả trước. Thêm dòng 28-30 trên Hình 9.12 để tính tổng số tiền được vay (dòng 29) và tỷ lệ lãi suất hàng tháng (dòng 30). Các phép tính này chỉ cần thực hiện một lần nên chúng được đặt trước lệnh lặp `Do While...Loop`. Biến `loanAmount` và `monthlyInterest` được sử dụng để tính số tiền phải trả hàng tháng. Bạn thực hiện phép tính này ở phần ngay sau đây.



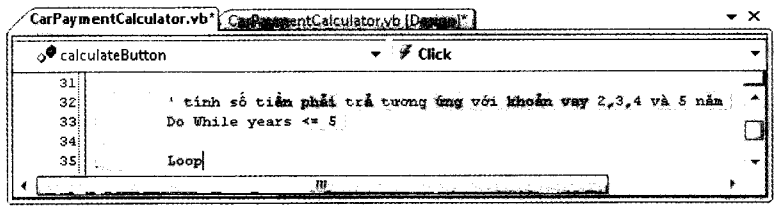
Hình 9.12 Tính tổng số tiền vay và tỷ lệ lãi suất hàng tháng.

4. ***Lưu project.*** Chọn **File > Save All** để lưu mã đã được sửa đổi.

Tiếp theo, bạn sẽ thêm lệnh lặp vào ứng dụng để tính số tiền phải trả hàng tháng cho bốn khoản vay theo các kỳ hạn khác nhau. Lệnh lặp này thực hiện tính toán cho các khoản vay kéo dài hai, ba, bốn, năm năm.

Tính số tiền phải trả hàng tháng sử dụng lệnh lặp Do While...Loop

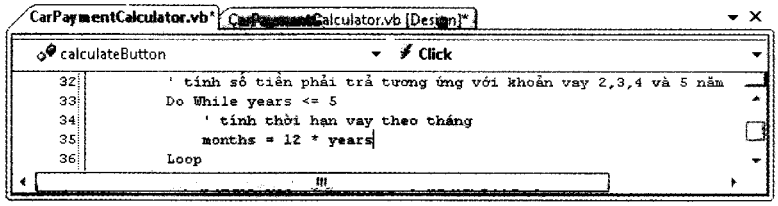
1. ***Thiết lập điều kiện tiếp tục vòng lặp.*** Thêm dòng 32-33 trên Hình 9.13 vào dưới dòng tính toán số tổng số tiền vay (`loanAmount`) và tỷ lệ lãi suất hàng tháng (`monthlyInterest`). Sau khi bạn viết mã của dòng 33 và bấm phím *Enter*, IDE đóng lệnh lặp bằng cách thêm từ khóa `Loop` ở dòng 35.



Hình 9.13 Điều kiện tiếp tục vòng lặp.

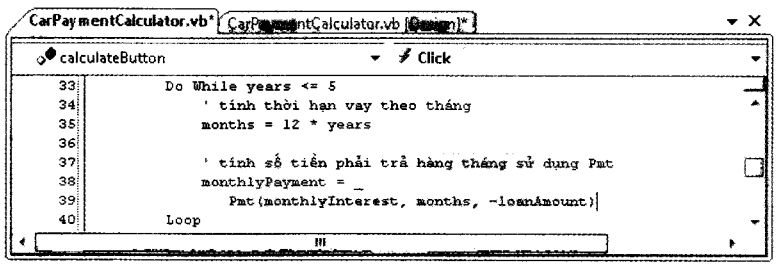
Nhớ lại rằng khoản vay ngắn hạn nhất trong ứng dụng này kéo dài hai năm, bởi vậy bạn khởi tạo biến `years` với giá trị 2 ở dòng 6 (Hình 9.10). Điều kiện tiếp tục vòng lặp (`years <=5`) trong Hình 9.13 chỉ ra rằng lệnh lặp `Do While...Loop` thực thi trong khi giá trị của biến `years` vẫn còn nhỏ hơn hoặc bằng 5. Vòng lặp này là một ví dụ về lệnh lặp được điều khiển bởi **biến đếm** (**counter-controlled repetition**). Kỹ thuật này sử dụng một biến gọi là **biến đếm** (**counter**) - trong trường hợp này là `years` - để kiểm soát số lần thực thi một tập hợp lệnh. Lệnh lặp được điều khiển bởi biến đếm còn được gọi là **lệnh lặp hữu hạn** (**definite repetition**) bởi vì số lần lặp được biết trước khi lệnh lặp bắt đầu thực thi. Trong ví dụ này, việc lặp lại kết thúc khi biến đếm (`years`) vượt quá giá trị 5.

2. ***Tính kỳ hạn thanh toán của khoản vay***. Thêm dòng 34-35 trên Hình 9.14 vào lệnh lặp `Do While...Loop` để tính kỳ hạn thanh toán (thời hạn của khoản vay tính theo tháng). Số tháng thay đổi qua mỗi lần lặp lại vòng lặp và kết quả tính toán thay đổi theo kỳ hạn thanh toán. Biến `months` nhận giá trị 24, 36, 48 và 60 trong mỗi lần lặp thành công.



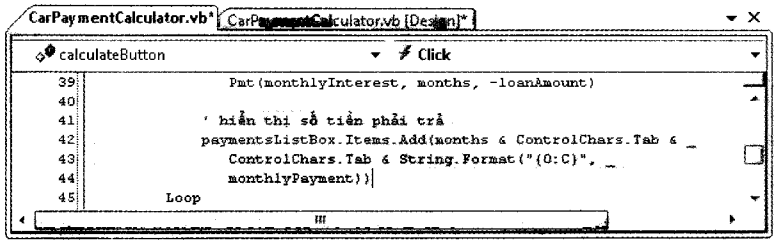
Hình 9.14 Chuyển thời gian vay tính theo đơn vị năm sang đơn vị tháng.

3. ***Tính số tiền phải trả hàng tháng***. Thêm dòng 37-39 trên Hình 9.15 vào lệnh lặp `Do While...Loop`. Dòng 38-39 (Hình 9.15) sử dụng hàm **`Pmt`** (có sẵn trong Visual Basic) trả lại giá trị kiểu `Double` cho biết số tiền phải trả hàng tháng của khoản vay có tỷ lệ lãi suất (`monthlyInterest`) không thay đổi trong khoảng thời gian (`months`) đã cho. Dòng 39 truyền tham số tỷ lệ lãi suất, thời hạn thanh toán (tính theo số tháng của kỳ hạn thanh toán) và tổng số tiền cho vay vào hàm `Pmt`. Chú ý truyền giá trị âm vào tổng số tiền cho vay. Số tiền cho vay được biểu diễn bởi giá trị âm, bởi vì chúng biểu diễn lượng tiền mặt bị lấy đi khỏi cá nhân hoặc tổ chức cho vay tiền. Giá trị kiểu `Double` do hàm `Pmt` trả lại được chuyển thành kiểu `Decimal` khi bạn gán nó cho biến `monthlyPayment` kiểu `Decimal`.



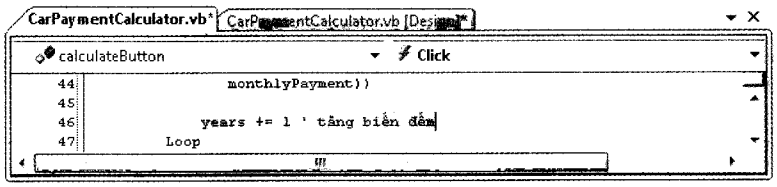
Hình 9.15 Hàm `Pmt` trả lại số tiền phải trả hàng tháng.

4. ***Hiển thị số tiền phải trả hàng tháng***. Thêm dòng 41-44 trên Hình 9.16 vào ứng dụng. Thời hạn phải trả tiền theo tháng và số lượng tiền phải trả mỗi tháng được hiển thị ở phía dưới dòng tiêu đề trên `ListBox`. Để thêm nội dung vào `ListBox`, gọi phương thức `Add` (dòng 42-44 trên Hình 9.16). Dòng 43-44 sử dụng phương thức `String.Format` để hiện thị giá trị `monthlyPayment` dưới định dạng tiền tệ. Chú ý rằng hai ký tự tab đảm bảo rằng số tiền phải trả hàng tháng được hiển thị ở cột thứ hai. Khoảng trắng được tạo ra bởi các ký tự tab làm cho kết quả của ứng dụng dễ đọc hơn.



Hình 9.16 Hiển thị số tháng và số tiền phải trả mỗi tháng.

5. ***Tăng biến đếm***. Thêm dòng 46 trên Hình 9.17 vào trước từ khóa kết thúc lệnh lặp. Dòng 46 tăng giá trị biến đếm (`years`). Biến `years` được tăng sau mỗi lần lặp cho đến tận khi nó bằng 6. Lúc này điều kiện tiếp tục vòng lặp (`year <= 5`) đạt giá trị `False` và kết thúc vòng lặp.



Hình 9.17 Tăng biến đếm.

6. ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng. Ứng dụng tính và hiển thị số tiền phải trả hàng tháng. Nhập các giá trị: giá xe, số tiền trả trước và tỷ lệ lãi suất hàng năm, sau đó nhấn vào `Button` **Calculate** để kiểm tra xem ứng dụng đã tính toán kết quả chính xác chưa.
7. ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.
8. ***Đóng IDE***. Đóng cửa sổ IDE Visual Basic bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

Hình 9.18 biểu diễn đoạn mã của ứng dụng **Car Payment Calculator**. Các dòng mã chứa các khái niệm lập trình mới mà bạn vừa học trong chương này được đánh dấu.

```
1  Public Class CarPaymentCalculatorForm
2      ' xử lý sự kiện Click cho Calculate Button
3      Private Sub calculateButton_Click(ByVal sender As System.Object,_
4      ByVal e As System.EventArgs) Handles calculateButton.Click
5
6          Dim years As Integer = 2 ' biến đếm vòng lặp
7          Dim months As Integer = 0 ' thời hạn vay theo tháng
8          Dim price As Decimal = 0 ' giá tiền của xe
9          Dim downPayment As Decimal = 0 ' số tiền trả trước
10         Dim interest As Double = 0 ' tỷ lệ lãi suất
11         Dim monthlyPayment As Decimal = 0 ' số tiền phải trả hàng tháng
12         Dim loanAmount As Decimal = 0 ' tổng số tiền vay
13         Dim monthlyInterest As Double = 0 ' tỷ lệ lãi suất hàng tháng
14
15         ' xóa nội dung văn bản hiển thị trên ListBox
16         paymentsListBox.Items.Clear()
17
18         ' thêm tiêu đề vào nội dung trên ListBox
19         paymentsListBox.Items.Add("Months" & ControlChars.Tab & _
20             ControlChars.Tab & "Monthly Payments")
21
22         ' lấy thông tin do người dùng nhập vào
23         ' và gán cho biến tương ứng
```

Xóa nội dung trên `ListBox` — dòng 16

Thêm tiêu đề vào `ListBox` — dòng 19

Hình 9.18 Đoạn mã ứng dụng **Car Payment Calculator**.(Phần 1/2)

```
24          downPayment = Val(downPaymentTextBox.Text)
25          price = Val(priceTextBox.Text)
26          interest = Val(interestTextBox.Text) / 100
27
28          ' xác định tổng số tiền vay và tỷ lệ lãi suất hàng tháng
29          loanAmount = price - downPayment
30          monthlyInterest = interest / 12
31
32          ' tính số tiền phải trả tương ứng với khoản vay 2,3,4 và 5 năm
33          Do While years <= 5
34              ' tính thời hạn vay theo tháng
35              months = 12 * years
36
37              ' tính số tiền phải trả hàng tháng sử dụng Pmt
38              monthlyPayment = _
39                  Pmt(monthlyInterest, months, -loanAmount)
40
41              ' hiển thị số tiền phải trả
42              paymentsListBox.Items.Add(months & ControlChars.Tab & _
43                  ControlChars.Tab & String.Format("{0:C}", _
44                  monthlyPayment))
45
46              years += 1 ' tăng biến đếm
47          Loop
48      End Sub ' calculateBu tton_Click
49  End Class ' CarPaymentCalculatorForm
```

Lệnh Do While...Loop thực hiện lặp thân lệnh khi biến years nhỏ hơn hoặc bằng 5

Tính kỳ hạn của khoản vay theo tháng

Tính số tiền phải trả mỗi tháng

Hiển thị số tháng và số tiền phải trả mỗi tháng

Tăng biến đếm years để điều kiện tiếp tục vòng lặp tiến dần đến giá trị sai

Hình 9.18 Đoạn mã ứng dụng **Car Payment Calculator**.(Phần 2/2)

TỰ ÔN TẬP

1. Lệnh lặp được điều khiển bởi biến đếm còn được gọi là_____bởi vì số lần lặp được biết trước khi bắt đầu thực thi vòng lặp.
   a) lệnh lặp hữu hạn
   b) lệnh lặp biết trước
   c) lệnh lặp theo thứ tự
   d) lệnh lặp sử dụng biến đếm
2. Dòng chữ được thêm vào `ListBox` để mô tả thông tin sẽ được hiển gọi là_____.
   a) tựa đề
   b) mở đầu
   c) tiêu đề
   d) chú giải

Đáp án: **1)** a. **2)** c.

9.5 Tổng kết

Trong chương này, bạn bắt đầu sử dụng lệnh lặp. Bạn đã sử dụng lệnh `Do While... Loop` và `Do Until...Loop` để lặp lại các hành động trong ứng dụng phụ thuộc vào điều kiện tiếp tục vòng lặp hoặc điều kiện kết thúc vòng lặp.

Lệnh lặp `Do While...Loop` thực thi chừng nào điều kiện tiếp tục vòng lặp còn có giá trị `True`. Khi điều kiện tiếp tục vòng lặp chuyển thành `Fasle`, quá trình lặp sẽ kết thúc. Vòng lặp vô hạn xảy ra khi điều kiện này không bao giờ chuyển thành `False`.

Lệnh lặp `Do Until...Loop` thực thi chừng nào điều kiện kết thúc vòng lặp còn có giá trị `False`. Khi điều kiện kết thúc vòng lặp chuyển thành `True`, quá trình lặp sẽ kết thúc. Vòng lặp vô hạn xảy ra khi điều kiện này không bao giờ chuyển thành `True`.

Bạn đã học về lệnh lặp được điều khiển bởi biến đếm, là lệnh lặp "biết" số lần lặp và tìm hiểu về biến đếm để đếm số lần lặp. Bạn đã sử dụng lệnh lặp để phát triển ứng dụng **Car Payment Calculator**. Trong ứng dụng này, bạn đã tính số tiền phải trả hàng tháng dựa vào tổng số tiền vay và tỷ lệ lãi suất cho thời hạn vay là hai, ba, bốn và năm năm.

Trong ứng dụng **Car Payment Calculator**, bạn đã sử dụng điều khiển `ListBox` để hiển thị kết quả cho các tùy chọn khoản vay mua ô tô. Bạn đã học về điều khiển `ListBox`, sử dụng để lưu trữ một danh sách các phần tử. Các phần tử có thể được lập trình để thêm vào và xóa khỏi `ListBox`. Các giá trị được thêm vào điều khiển `ListBox` bằng cách gọi phương thức `Add` của thuộc tính `Items` của điều khiển `ListBox`. Thuộc tính `Items` trả lại đối tượng chứa tất cả các giá trị được hiển thị trên `ListBox`.

Trong chương tiếp theo, bạn sẽ tìm hiểu hai lệnh lặp khác, tiếp tục khám phá lệnh lặp được điều khiển bởi biến đếm. Ứng dụng **Car Payment Calculator** minh họa một trường hợp thường dùng của các lệnh lặp - thực hiện cùng một phép tính với nhiều giá trị khác nhau. Ứng dụng tiếp theo giới thiệu một ứng dụng phổ biến khác của lệnh lặp - tính tổng một dãy số.

TỔNG KẾT KỸ NĂNG

Hiển thị giá trị lên `ListBox`

- Thuộc tính `Items` của điều khiển `ListBox` trả về đối tượng chứa các giá trị được hiển thị trên `ListBox`.
- Gọi phương thức `Add` để thêm giá trị vào thuộc tính `Items`.

Xóa nội dung hiển thị trên `ListBox`

- Phương thức `Clear` của thuộc tính `Items` xóa tất cả các giá trị có trên `ListBox`.

Lặp lại các hành động trong ứng dụng

- Sử dụng lệnh lặp phụ thuộc vào giá trị đúng hoặc sai của điều kiện tiếp tục vòng lặp hoặc điều kiện kết thúc vòng lặp.

Thực thi lệnh lặp với số lần lặp được biết trước

- Sử dụng lệnh lặp được điều khiển bởi biến đếm, trong đó biến đếm để xác định số lần thực thi một tập lệnh.

Sử dụng lệnh lặp `Do While...Loop`

- Lệnh lặp này thực thi khi điều kiện tiếp tục vòng lặp đạt giá trị `True`.
- Lặp vô hạn xảy ra khi điều kiện tiếp tục vòng lặp không bao giờ chuyển sang `False`.

Sử dụng lệnh lặp `Do Until...Loop`

- Lệnh lặp này thực thi khi điều kiện kết thúc vòng lặp đạt giá trị True.
- Lặp vô hạn xảy ra khi điều kiện kết thúc vòng lặp không bao giờ chuyển sang True.

Ghép chuỗi

- Sử dụng toán tử & để tạo ra một chuỗi mới từ hai chuỗi đã tồn tại. Nội dung toán hạng bên phải được thêm vào nội dung toán hạng bên trái để tạo ra chuỗi mới.

THUẬT NGỮ

biến đếm (counter) - Biến được dùng để xác định số lần thân lệnh lặp được thực thi.

điều khiển `ListBox` - Cho phép người dùng xem các phần tử của một danh sách. Các phần tử có thể được lập trình để thêm hoặc xóa khỏi danh sách.

điều kiện kết thúc vòng lặp (loop-termination condition) - Điều kiện được dùng trong lệnh lặp (ví dụ như lệnh `Do Until...Loop`) cho phép lệnh lặp tiếp tục khi điều kiện là `False` và lệnh lặp kết thúc khi điều kiện này có giá trị là `True`.

điều kiện tiếp tục vòng lặp (loop-continuation condition) - Điều kiện được dùng trong lệnh lặp (ví dụ như lệnh `Do While...Loop`) cho phép lệnh lặp tiếp tục khi điều kiện là `True` và lệnh lặp kết thúc khi điều kiện này có giá trị là `False`.

hàm `Pmt` - Hàm có sẵn trong Visual Basic, dựa vào tỷ lệ lãi suất, thời hạn vay và tổng số tiền vay, trả về số lượng tiền phải trả kiểu `Double`.

hằng `ControlChars.Tab` - Biểu diễn một ký tự tab.

lệnh lặp (repetition statement) - Cho phép bạn thực hiện lặp lại một hành động hoặc các hành động dựa vào giá trị điều kiện.

lệnh lặp `Do Until...Loop` - Lệnh điều khiển thực thi một tập lệnh trong phần thân lệnh lặp *cho đến khi* điều kiện kết thúc vòng lặp có giá trị `True`.

lệnh lặp `Do While...Loop` - Lệnh điều khiển thực thi một tập các lệnh trong phần thân lệnh lặp *khi* điều kiện tiếp tục vòng lặp có giá trị `True`.

lệnh lặp được điều khiển bởi biến đếm (counter-controlled repetition) - Kỹ thuật sử dụng biến đếm để xác định số lần thân lệnh lặp thực thi. Còn được gọi là lệnh lặp hữu hạn.

ký hiệu ghép (trong ngôn ngữ UML) - Ký hiệu hình thoi trong UML sử dụng để gộp hai luồng hoạt động thành một luồng.

phần thân của lệnh điều khiển - Tập các lệnh nằm bên trong lệnh điều khiển.

phương thức `Add` của `Items` - Thêm một phần tử vào điều khiển `ListBox`.

phương thức `Clear` của `Items` - Xóa nội dung hiển thị trên điều khiển `ListBox`.

thuộc tính Items của điều khiển `ListBox` - Trả lại đối tượng chứa tất cả các giá trị của `ListBox`.

tiêu đề (header) - Dòng chữ nằm ở đầu `ListBox` để làm rõ thông tin được hiển thị trên `ListBox`.

toán tử ghép chuỗi (&) - Toán tử này nối hai toán hạng thành một giá trị chuỗi.

vòng lặp (loop) - Tên gọi khác của lệnh lặp.

vòng lặp vô hạn (infinite loop) - Là lỗi xảy ra khi lệnh lặp không bao giờ kết thúc.

HƯỚNG DẪN THIẾT KẾ GIAO DIỆN

`ListBox`

- `ListBox` nên có kích thước đủ rộng để hiển thị toàn bộ nội dung hoặc đủ lớn để thanh cuộn có thể sử dụng dễ dàng.
- Nên sử dụng tiêu đề trong `ListBox` khi bạn hiển thị dữ liệu dạng bảng. Thêm tiêu đề mô tả thông tin được hiển thị trên `ListBox` sẽ giúp cho người dùng dễ hiểu hơn.

ĐIỀU KHIỂN, SỰ KIỆN, THUỘC TÍNH & PHƯƠNG THỨC

`ListBox`  Điều khiển này cho phép người dùng xem và lựa chọn các phần tử trong một danh sách.

- ***Trên giao diện khi ứng dụng chạy***

Months	Monthly Payments
24	$490.50
36	$339.06
48	$263.55
60	$218.41

- ***Thuộc tính***

  `Items` - Trả lại đối tượng chứa các phần tử được hiển thị trên `ListBox`.

  `Location` - Chỉ ra vị trí của `ListBox` trên `Form`.

  `Name` - Chỉ ra tên được sử dụng để truy cập `ListBox` trong khi lập trình. Tên nên được thêm hậu tố `ListBox` vào phía sau.

  `Size` - Chỉ ra chiều rộng và chiều cao (bằng pixel) của `ListBox`.

- ***Phương thức***

  `Items.Add` - Thêm một phần tử vào thuộc tính `Items`.

  `Items.Clear` - Xóa tất cả các giá trị có trong thuộc tính `Items` của `ListBox`.

CÂU HỎI TRẮC NGHIỆM

9.1 Lệnh_____thực thi cho đến khi điều kiện kết thúc vòng lặp đạt giá trị `True`.

a) `Do While...Loop`
b) `Do Until...Loop`
c) `Do`
d) `Loop`

9.2 Lệnh_____thực thi cho đến khi điều kiện tiếp tục vòng lặp đạt giá trị `False`.

a) `Do While...Loop`
b) `Do Until...Loop`
c) `Do`
d) `Loop`

9.3 Vòng lặp_____xảy ra khi điều kiện trong lệnh `Do While...Loop` không bao giờ có giá trị `False`.

a) vô hạn
b) không xác định
c) lồng
d) bất định

9.4 ____là biến giúp kiểm soát số lần thực thi tập lệnh.

a) Biến lặp
b) Biến đếm
c) Vòng lặp
d) Lệnh điều khiển lặp

9.5 Điều khiển_____cho phép người dùng thêm và xem các phần tử trong một danh sách.

a) `ListItems`
b) `SelectBox`
c) `ListBox`
d) `ViewBox`

9.6 Trong biểu đồ hoạt động UML, ký hiệu_____gộp hai luồng hoạt động thành một luồng.

a) ghép
b) nối
c) trạng thái hành động
d) ra quyết định

9.7 Thuộc tính_____trả về đối tượng chứa tất cả các giá trị của `ListBox`.

a) `All`
b) `List`
c) `ListItemValues`
d) `Items`

9.8 Phương thức_____của `Items` xóa tất cả các giá trị có trong danh sách.

a) `Remove`
b) `Delete`
c) `Clear`
d) `Del`

9.9 Phương thức_____của `Items` thêm một phần tử vào `ListBox`.

a) `Include`
b) `Append`
c) `Add`
d) Các lựa chọn trên đều sai

9.10 Hàm_____tính số tiền phải trả hàng tháng của một khoản vay dựa trên tỷ lệ lãi suất cố định.

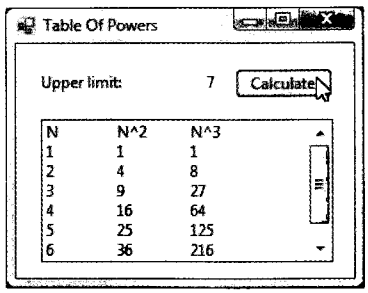
a) `MonPmt`
b) `Payment`
c) `MonthlyPayment`
d) `Pmt`

BÀI TẬP

9.11 (***Ứng dụng Table of Powers***) Viết ứng dụng hiển thị một bảng chứa các số trong khoảng từ 1 tới một giới hạn trên, cùng với giá trị bình phương của mỗi số (ví dụ, số n lũy thừa 2 hoặc *n* ^ 2) và giá trị lập phương (số n lũy thừa 3 hoặc *n* ^ 3). Người dùng chỉ ra giới hạn trên và kết quả được hiển thị trên một `ListBox`, như Hình 9.19.

a) ***Copy template của ứng dụng vào thư mục làm việc***. Copy thư mục `C:\Examples\Tutorial09\Exercises\TableOfPowers` vào thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng***. Nhấn đúp vào file `TableOfPowers.sln` trong thư mục `TableOfPowers` để mở ứng dụng.



Hình 9.19 Giao diện ứng dụng **Table of Powers**.

c) ***Thêm ListBox***. Thêm `ListBox` vào ứng dụng, như Hình 9.19. `ListBox` này có tên là `resultsListBox`.

d) ***Thêm xử lý sự kiện cho TextBox Upper limit:***. Nhấn đúp vào `TextBox` **Upper limit:** để tạo ra xử lý sự kiện cho sự kiện `TextChanged` của `TextBox`. Trong xử lý sự kiện này, xóa nội dung có trong `ListBox`.

e) ***Thêm xử lý sự kiện cho Button Calculate***. Nhấn đúp vào `Button` **Calculate** để tạo ra xử lý sự kiện `calculateButton_Click` rỗng. Thêm đoạn mã được chỉ ra trong các bước còn lại vào xử lý sự kiện này.

f) ***Xóa nội dung trên ListBox***. Sử dụng phương thức `Clear` của thuộc tính `Items` để xóa dữ liệu của `ListBox` này.

g) ***Lấy giá trị giới hạn trên được cung cấp bởi người dùng***. Gán giá trị do người dùng nhập vào `TextBox` **Upper limit:** cho một biến. Chú ý rằng thuộc tính `Name` của `TextBox` này được thiết lập giá trị là `inputTextBox`.

h) ***Thêm tiêu đề***. Sử dụng phương thức `Add` của thuộc tính `Items` để thêm tiêu đề vào `ListBox`. Tiêu đề của ba cột như sau - `N`, `N^2` , `N^3`. Tiêu đề cột nên được phân tách bởi các ký tự tab.

i) ***Tính lũy thừa từ 1 tới giới hạn trên***. Sử dụng `Do Until...Loop` để tính giá trị bình phương và lập phương của mỗi số trong khoảng từ 1 đến giới hạn trên. Thêm một phần tử vào `ListBox` để chứa số hiện tại, giá trị bình phương và giá trị lập phương của số này.

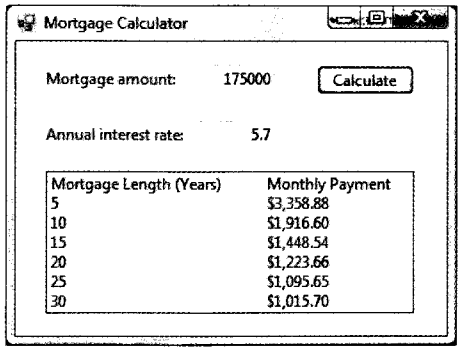
j) ***Tăng biến đếm***. Nhớ tăng biến đếm trong mỗi lần lặp.

k) ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng của bạn. Nhập giới hạn trên và nhấn vào `Button` **Calculate**. Kiểm tra xem bảng lũy thừa đã hiển thị đúng giá trị chưa.

l) ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

m) ***Đóng IDE***. Đóng cửa sổ IDE Visual Basic bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

9.12 (***Ứng dụng Mortgage Calculator***) Ngân hàng đưa ra các khoản vay thế chấp có thể hoàn trả trong vòng 5, 10, 15, 20, 25 hoặc 30 năm. Viết ứng dụng cho phép người dùng nhập vào giá của ngôi nhà (tổng số tiền thế chấp) và tỷ lệ lãi suất hàng năm. Khi người dùng nhấn vào `Button` **Calculate**, ứng dụng hiển thị một bảng chứa thời hạn vay thế chấp với số tiền phải trả hàng tháng, như Hình 9.20.



Chú thích hình

- ***Mortgage amount:*** *Số tiền thế chấp*
- ***Annual interest rate:*** *Tỷ lệ lãi suất hàng năm*

Hình 9.20 Giao diện ứng dụng **Mortgage Calculator**.

a) ***Copy template của ứng dụng vào thư mục làm việc***. Copy thư mục `C:\Examples\Tutorial09\Exercises\MortgageCalculator` vào thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng***. Nhấn đúp vào file `MortgageCalculator.sln` trong thư mục `MortgageCalculator` để mở ứng dụng.

c) ***Thêm `ListBox`***. Thêm `ListBox` vào ứng dụng, như Hình 9.20. `ListBox` này có tên là `resultsListBox`.

d) ***Thêm xử lý sự kiện cho Button Calculate***. Nhấn đúp vào `Button` **Calculate** để tạo ra xử lý sự kiện `calculateButton_Click` rỗng. Thêm đoạn mã được chỉ ra trong các bước còn lại vào xử lý sự kiện này.

e) ***Chuyển tỷ lệ lãi suất hàng năm sang tỷ lệ lãi suất hàng tháng***. Để chuyển tỷ lệ lãi suất hàng năm từ một giá trị phần trăm thành giá trị tương đương kiểu `Double`, chia tỷ lệ lãi suất hàng năm cho `100`. Sau đó chia tỷ lệ lãi suất kiểu `Double` này cho `12` để thu được tỷ lệ lãi suất hàng tháng.

f) ***Xóa nội dung hiển thị trên `ListBox`***. Sử dụng phương thức `Clear` của thuộc tính `Items` để xóa dữ liệu có sẵn trên `ListBox`.

g) ***Hiển thị tiêu đề***. Sử dụng phương thức `Add` để hiển thị tiêu đề trên `ListBox`. Tiêu đề cột là `"Mortgage Length (Years)"` và `"Monthly Payment"`, được phân tách bởi nhau một ký tự tab.

h) ***Sử dụng lệnh lặp***. Thêm lệnh `Do While...Loop` để tính sáu tùy chọn số tiền phải trả hàng tháng cho khoản vay thế chấp của người dùng. Mỗi tùy chọn có khoảng thời gian vay thế chấp khác nhau. Trong ví dụ này, các khoảng thời gian vay thế chấp là `5`, `10`, `15`, `20`, `25` và `30` năm.

i) ***Chuyển thời gian thế chấp tính theo năm sang tính theo tháng***. Chuyển số năm thành số tháng.

j) ***Tính số tiền phải trả hàng tháng cho sáu tùy chọn thế chấp khác nhau***. Sử dụng hàm `Pmt` để tính số tiền phải trả hàng tháng. Truyền tham số tỷ lệ lãi suất hàng tháng, số tháng vay thế chấp, tổng số tiền vay thế chấp vào hàm này. Nhớ rằng số tiền vay thế chấp phải là số âm vì nó biểu diễn một khoản tiền được trả bởi người cho vay.

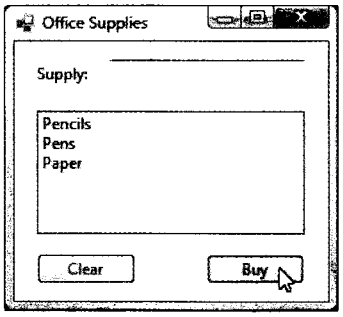
k) ***Hiển thị các kết quả***. Sử dụng phương thức `Add` của thuộc tính `Items` để hiển thị thời hạn khoản vay theo năm cùng với số tiền phải trả hàng tháng trên `ListBox`. Sử dụng ba ký tự tab để đảm bảo rằng số tiền phải trả hàng tháng được hiển thị ở cột thứ hai.

l) ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng của bạn. Nhập vào tổng số tiền vay thế chấp và tỷ lệ lãi suất hàng năm, sau đó nhấn vào `Button` **Calculate**. Kiểm tra xem số tiền phải trả hàng tháng được hiển thị đúng kết quả chưa.

m) ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

n) ***Đóng IDE***. Đóng cửa sổ IDE Visual Basic bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

9.13 (***Ứng dụng Office Supplies***) Tạo ứng dụng cho phép người dùng tạo danh sách các mặt hàng văn phòng phẩm cần phải mua, như Hình 9.21. Người dùng nhập các mặt hàng vào `TextBox` và nhấn vào `Button` **Buy** để thêm vào `ListBox`. `Button` **Clear** xóa bỏ tất các mặt hàng có trên danh sách.



Hình 9.21 Giao diện ứng dụng **Office Supplies**.

a) ***Copy template của ứng dụng vào thư mục làm việc***. Copy thư mục `C:\Examples\Tutorial09\Exercises\OfficeSupplies` tới thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng***. Nhấn đúp vào file `OfficeSupplies.sln` trong thư mục `OfficeSupplies` để mở ứng dụng.

c) ***Thêm `ListBox`***. Thêm một `ListBox` vào ứng dụng. `ListBox` này có tên là `suppliesListBox`. Thiết lập kích thước `ListBox` giống như trên Hình 9.21.

d) ***Thêm xử lý sự kiện cho `Button` Buy***. Nhấn đúp vào `Button` **Buy** để tạo ra xử lý sự kiện `buyButton_Click`. Xử lý sự kiện này lấy giá trị do người dùng nhập vào `TextBox`. Sau đó, dữ liệu này được lưu như là một phần tử trong `ListBox`. Sau khi giá trị do người dùng nhập vào được thêm vào `ListBox`, xóa nội dung trên `TextBox` **Supply:**.

e) ***Thêm xử lý sự kiện cho `Button` Clear***. Nhấn đúp vào `Button` **Clear** để tạo ra xử lý sự kiện `clearButton_Click`. Xử lý sự kiện này sử dụng phương thức `Clear` của thuộc tính `Items` để xóa `ListBox`.

f) ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng của bạn. Nhập một vài item vào `TextBox` **Supply:** và nhấn vào `Button` **Buy** sau mỗi lần nhập một mặt hàng. Kiểm tra xem từng mặt hàng này đã được thêm vào `ListBox` chưa. Nhấn vào `Button` **Clear** và kiểm tra xem tất cả các mặt hàng đã bị xóa khỏi `ListBox` chưa.

g) ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

h) ***Đóng IDE***. Đóng cửa sổ IDE Visual Basic bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

Đoạn mã này làm gì? ▶ **9.14** Kết quả của đoạn mã dưới đây là gì?

```
1  Dim x As Integer = 1
2  Dim mysteryValue As Integer = 1
3
4  Do While x < 6
5     mysteryValue *= x
6     x += 1
7  Loop
8
9  displayLabel.Text = mysteryValue
```

Đoạn mã này có gì sai? ▶ **9.15** Tìm lỗi sai trong đoạn mã dưới đây:

a) Giả sử biến x được khai báo và khởi tạo giá trị là 1. Vòng lặp tính tổng các số từ 1 đến 10.

```
1  Dim total As Integer = 0
2
3  Do Until x <= 10
4     total += x
5     x += 1
6  Loop
```

b) Giả sử biến `counter` được khai báo và khởi tạo giá trị là 1. Vòng lặp tính tổng các số từ 1 đến 100.

```
1  Do While counter <= 100
2     total += counter
3  Loop
4
5  counter += 1
```

c) Giả sử biến `counter` được khai báo và khởi tạo giá trị là 1000. Vòng lặp lặp từ 1000 đến 1.

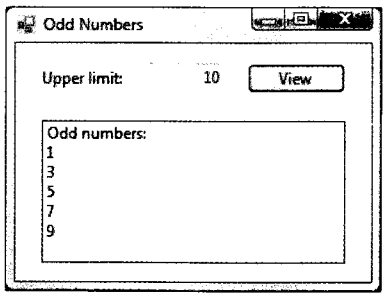
```
1  Do While > 0
2     numbersListBox.Items.Add(counter)
3     counter += 1
4  Loop
```

d) Giả sử rằng biến `counter` được khai báo và khởi tạo giá trị là 1. Vòng lặp thực thi năm lần, thêm các số từ 1-5 vào `ListBox`.

```
1  Do While counter < 5
2     numbersListBox.Items.Add(counter)
3     counter += 1
4  Loop
```

Sử dụng trình gỡ lỗi

▶ **9.16** (***Ứng dụng Odd Numbers***) Ứng dụng **Odd Numbers** hiển thị tất cả các số nguyên lẻ từ 1 tới một số do người dùng nhập vào. Copy ứng dụng **Odd Numbers** từ `C:\Examples\Tutorial09\Exercises\Debugger` vào thư mục làm việc của bạn. Chạy ứng dụng. Chú ý rằng vòng lặp vô hạn xảy ra sau khi bạn nhập giá trị vào `TextBox` **Upper limit:** và nhấn vào `Button` **View**. Chọn **Debug > Stop Debug** để đóng ứng dụng đang chạy. Sử dụng trình gỡ lỗi để tìm và sửa lỗi trong ứng dụng. Hình 9.22 hiển thị kết quả đúng của ứng dụng.

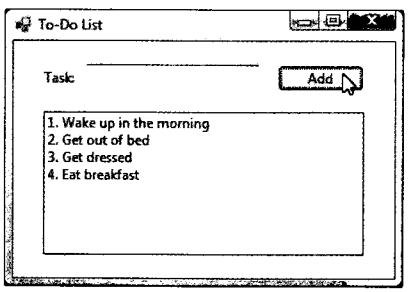


Hình 9.22 Kết quả đúng của ứng dụng **Odd Numbers**.

Bài tập nâng cao

▶ **9.17** (***Ứng dụng To-Do List***) Sử dụng `ListBox` như là một danh sách những việc phải làm. Nhập từng công việc vào `TextBox` và thêm chúng vào `ListBox` bằng cách nhấn vào `Button`. Các công việc được hiển thị trong một danh sách được đánh số, như Hình 9.23. Để làm việc này, chúng ta sẽ làm quen với thuộc tính `Count`, trả lại số lượng phần tử của thuộc tính `Items` của `ListBox`. Dưới đây là một ví dụ gọi và gán số lượng phần tử được hiển thị trong `sampleListBox` cho một biến `Integer`:

```
Count = sampleListBox.Items.Count
```



Hình 9.23 Giao diện ứng dụng **To-Do List**.

CHƯƠNG 10

Ứng dụng Class Average

Giới thiệu lệnh lặp Do...Loop While và Do...Loop Until

Mục tiêu

Trong chương này, bạn sẽ tìm hiểu để:

- Sử dụng lệnh **Do... Loop While.**
- Sử dụng lệnh **Do... Loop Until.**
- Tìm hiểu thêm về lệnh lặp được điều khiển bởi biến đếm.
- Chuyển focus tới một điều khiển.
- Kích hoạt hay vô hiệu hóa **Button**.

Nội dung chính

Trong chương này chúng ta sẽ tiếp tục thảo luận về các lệnh lặp đã đề cập ở Chương 9. Trong chương trước, chúng ta đã tìm hiểu về lệnh lặp `Do While... Loop` và `Do Until...Loop`, các lệnh này kiểm tra điều kiện tiếp tục vòng lặp và điều kiện kết thúc vòng lặp trước khi thực hiện lặp. Trong chương này, bạn sẽ được giới thiệu thêm hai lệnh lặp nữa là `Do...Loop While` và `Do...Loop Until`. Hai lệnh này kiểm tra điều kiện *sau* mỗi lần lặp. Do đó, thân của các lệnh lặp này được thực thi ít nhất một lần.

Bạn cũng sẽ học cách vô hiệu hóa hay kích hoạt điều khiển trên `Form`. Khi điều khiển (ví dụ như `Button`) bị vô hiệu hóa thì điều khiển này không còn hồi đáp lại thao tác của người dùng nữa. Bạn sử dụng tính năng này để ngăn người dùng gây ra lỗi trong ứng dụng. Chương này cũng giới thiệu các khái niệm chuyển focus của ứng dụng cho một điều khiển. Sử dụng focus hợp lý sẽ làm cho ứng dụng dễ sử dụng hơn.

10.1 Chạy thử ứng dụng Class Average

Ứng dụng này phải đáp ứng những yêu cầu sau:

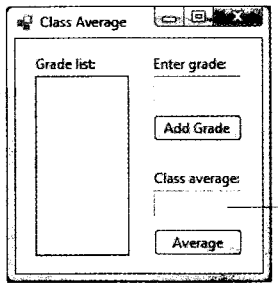
Yêu cầu đối với ứng dụng

Một giáo viên thường đưa ra các câu hỏi cho một lớp học gồm 10 sinh viên. Điểm của mỗi câu hỏi là một số nguyên trong khoảng từ 0 tới 100 (kể cả hai giá trị 0 và 100). Giáo viên này muốn bạn phát triển ứng dụng để tính điểm trung bình của cả lớp cho mỗi câu hỏi.

Điểm trung bình của cả lớp bằng tổng số điểm chia cho số sinh viên tham gia trả lời câu hỏi. Giải thuật giải quyết vấn đề này là nhập vào từng điểm số, tính tổng số điểm, thực hiện tính trung bình và hiển thị kết quả. Bạn bắt đầu bằng việc chạy thử ứng dụng đã hoàn thiện. Sau đó, bạn tìm hiểu những tính năng bổ sung cần thiết của Visual Basic để xây dựng cho mình một phiên bản của ứng dụng này.

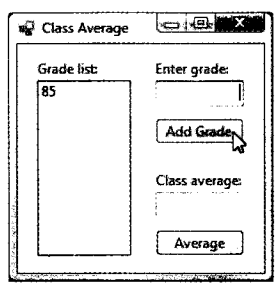
Chạy thử ứng dụng Class Average

1. ***Mở ứng dụng đã hoàn thiện***. Mở thư mục `C:\Examples\Tutorial10\CompletedApplication\ClassAverage` để tìm ứng dụng **Class Average**. Nhấn đúp vào file `ClassAverage.sln` để mở ứng dụng trong IDE Visual Basic.
2. ***Chạy ứng dụng Class Average***. Chọn **Debug > Start Debugging** để chạy ứng dụng này (xem Hình 10.1).



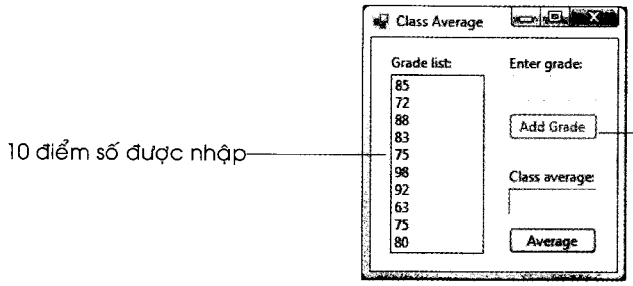
Hình 10.1 Giao diện ứng dụng **Class Average** đang chạy.

3. ***Nhập điểm cho mỗi câu hỏi***. Nhập giá trị 85 là điểm số đầu tiên vào `TextBox` **Enter Grade:** và nhấn vào `Button` **Add Grade**. Điểm vừa nhập sẽ được thêm vào `ListBox`, như Hình 10.2. Sau khi bạn nhấn vào `Button` **Add Grade**, con trỏ xuất hiện ở `TextBox` **Enter grade:**. Khi một điều khiển được chọn (ví dụ, `TextBox` **Enter grade:**), thì ta gọi điều khiển đó nhận được **focus** của ứng dụng. Bạn sẽ học thiết lập focus cho điều khiển khi bạn xây dựng ứng dụng của chương này. Như bạn thấy, focus của ứng dụng chuyển sang `TextBox` **Enter grade:**, bạn có thể nhập một điểm khác mà không cần trỏ chuột vào `TextBox` này hoặc không cần bấm phím *tab*. Chuyển focus tới một điều khiển cụ thể sẽ cho người dùng biết thông tin nào được mong đợi tiếp theo. [*Lưu ý*: Nếu bạn nhấn vào `Button` **Average** trước khi 10 điểm được nhập, một lỗi thực thi sẽ xảy ra. Chọn **Debug > Stop Debugging** để đóng cửa sổ ứng dụng. Lặp lại bước 2. Bạn sẽ sửa lỗi này trong phần bài tập].

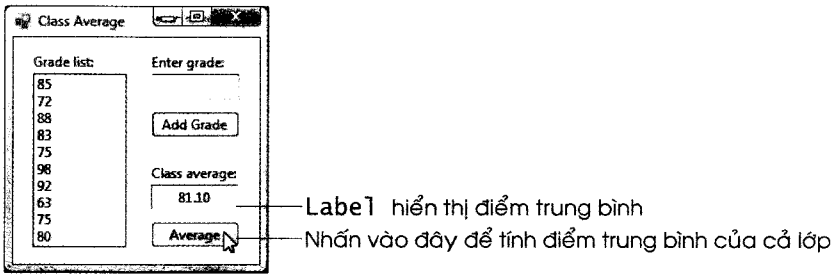


Hình 10.2 Nhập điểm vào ứng dụng **Class Average**.

4. ***Lặp lại* Bước 3 *chín lần***. Nhập chín điểm khác trong khoảng từ 0 đến 100, nhấn vào `Button` **Add Grade** sau mỗi lần nhập. Sau khi có 10 điểm được hiển thị trong `ListBox` **Grade list:**, giao diện sẽ giống như Hình 10.3. Chú ý rằng `Button` **Add Grade** bị vô hiệu hóa sau khi bạn nhập đủ 10 điểm. Màu của `Button` này sẽ chuyển sang màu xám và khi nhấn vào `Button` này sẽ không gọi xử lý sự kiện thực thi lệnh.
5. ***Tính điểm trung bình của cả lớp***. Nhấn vào `Button` **Average** để tính điểm trung bình của 10 câu hỏi. Điểm trung bình của cả lớp được hiển thị trên `Label` phía trên `Button` **Average** (Hình 10.14). Chú ý rằng `Button` **Add Grade** bây giờ đang ở trạng thái kích hoạt.

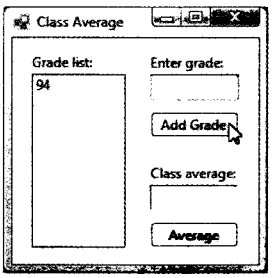


Hình 10.3 Ứng dụng **Class Average** sau khi 10 điểm số được nhập.



Hình 10.4 Hiển thị điểm trung bình của cả lớp.

6. ***Nhập một tập điểm khác.*** Bạn có thể tính điểm trung bình của cả lớp với một tập 10 điểm khác mà không cần chạy lại ứng dụng. Nhập một điểm vào `TextBox` và nhấn vào `Button` **Add Grade**. Chú ý rằng `ListBox` **Grade list:** và trường **Class average:** bị xóa khi bạn nhập một tập điểm khác (Hình 10.5).



Hình 10.5 Nhập một tập điểm mới.

7. ***Đóng ứng dụng.*** Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.
8. ***Đóng IDE.*** Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

10.2 Lệnh lặp Do...Loop While

Lỗi lập trình thường gặp

Vòng lặp vô hạn xảy ra khi điều kiện tiếp tục vòng lặp trong lệnh `Do...Loop While` không bao giờ có giá trị `False`.

Lệnh lặp **`Do...Loop While`** giống với lệnh lặp `Do While...Loop`, cả hai lệnh này đều lặp khi điều kiện tiếp tục vòng lặp có giá trị `True`. Trong lệnh `Do While...Loop`, điều kiện tiếp tục vòng lặp được kiểm tra tại thời điểm bắt đầu vòng lặp, trước khi thân vòng lặp được thực hiện. Lệnh lặp `Do...Loop While` kiểm tra điều kiện tiếp tục vòng lặp *sau khi* thân vòng lặp thực hiện. Do vậy, trong lệnh lặp `Do...Loop While`, thân vòng lặp luôn luôn thực thi ít nhất một lần. Khi lệnh `Do...Loop While` kết thúc, ứng dụng tiếp tục thực thi lệnh tiếp theo sau từ khóa `Loop While`.

Để minh họa lệnh lặp Do...Loop While, hãy xem ví dụ đóng gói va li dưới đây. Trước khi bạn bắt đầu đóng gói, va li rỗng. Bạn đặt các vật vào va li, sau đó xác định xem va li đã đầy chưa. Chừng nào va li chưa đầy, bạn tiếp tục đặt các vật vào trong va li. Xét ví dụ về lệnh Do...Loop While được biểu diễn bởi đoạn mã bên dưới. Đoạn mã này hiển thị các số từ 1 đến 3 trên ListBox:

```
Dim counter As Integer = 1

Do
    displayListBox.Items.Add(counter)
    counter += 1
Loop While counter <= 3
```

Đoạn ứng dụng này khởi tạo giá trị 1 cho biến counter. Điều kiện tiếp tục vòng lặp của lệnh Do...Loop While là counter <= 3. Khi điều kiện tiếp tục vòng lặp là True, lệnh Do...Loop While thực thi. Khi điều kiện tiếp tục vòng lặp chuyển sang False (khi counter lớn hơn 3), lệnh Do...Loop While hoàn thành thực thi và displayListBox chứa các số từ 1 đến 3. Phần dưới đây mô tả từng bước lệnh lặp trên thực thi.

Thực thi lệnh lặp Do...Loop While

1. Ứng dụng khai báo biến counter và gán giá trị 1 cho biến này.
2. Ứng dụng chuyển đến thực thi lệnh lặp Do...Loop While.
3. Giá trị hiện đang được lưu trong biến counter (trong trường hợp này là 1) được thêm vào thuộc tính Items của displayListBox.
4. Giá trị biến counter được tăng thêm 1, lúc này biến counter có giá trị là 2.
5. Điều kiện tiếp tục vòng lặp được kiểm tra. Điều kiện này có giá trị là True (counter nhỏ hơn hoặc bằng 3), bởi vậy ứng dụng tiếp tục thực thi lệnh đầu tiên sau lệnh Do.
6. Giá trị hiện đang được lưu trong biến counter (trong trường hợp này là 2) được thêm vào thuộc tính Items của displayListBox.
7. Giá trị biến counter được tăng thêm 1, lúc này biến counter có giá trị là 3.
8. Điều kiện tiếp tục vòng lặp được kiểm tra. Điều kiện này có giá trị là True (counter nhỏ hơn hoặc bằng 3), bởi vậy ứng dụng tiếp tục thực thi lệnh đầu tiên sau lệnh Do.
9. Giá trị hiện đang được lưu trong biến counter (trong trường hợp này là 3) được thêm vào thuộc tính Items của displayListBox.
10. Giá trị biến counter được tăng thêm 1, lúc này biến counter có giá trị là 4.
11. Điều kiện tiếp tục vòng lặp được kiểm tra. Điều kiện này có giá trị là False (counter không nhỏ hơn hoặc bằng 3), bởi vậy ứng dụng thoát khỏi lệnh lặp Do...Loop While.

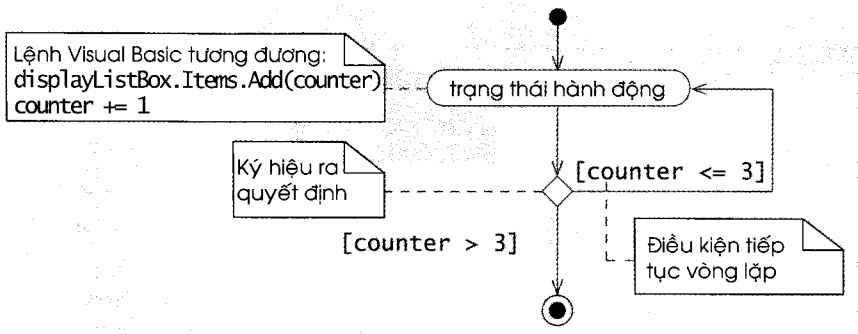
Mẹo tránh lỗi

Lựa chọn giá trị cuối cùng của điều kiện và toán tử quan hệ thích hợp có thể làm giảm khả năng xảy ra lỗi kết thúc vòng lặp sớm hoặc muộn. Ví dụ, trong lệnh Do While...Loop sử dụng để hiển thị lên màn hình các giá trị từ 1-10, điều kiện tiếp tục vòng lặp là counter <= 10 mà không phải là counter < 10 (điều kiện này sẽ gây ra lỗi kết thúc vòng lặp sớm) hay counter < 11 (điều kiện này đúng nhưng không rõ ràng).

Nếu bạn gõ nhầm điều kiện tiếp tục vòng lặp là counter < 3 hoặc counter <= 2, ListBox sẽ chỉ hiển thị giá trị 1 và 2. Toán tử quan hệ không đúng (dấu nhỏ hơn trong điều kiện counter < 3) hay giá trị cuối cùng của biến đếm vòng lặp không đúng (giá trị 2 trong counter <= 2) trong lệnh lặp, hay khởi tạo sai giá trị ban đầu (counter = 0) có thể gây ra lỗi **kết thúc vòng lặp sớm hoặc muộn** (**off-by-one error**). Lỗi này xảy ra khi vòng lặp thực thi ít hơn hoặc nhiều hơn một lần lặp so với số lần cần thiết.

Hình 10.6 minh họa biểu đồ hoạt động UML cho lệnh Do...Loop While. Biểu đồ này rõ ràng chỉ ra rằng điều kiện canh giữ tiếp tục vòng lặp ([counter <= 3]) không được đánh giá cho đến tận sau khi vòng lặp thực hiện trạng thái hành động

ít nhất một lần. Nhớ lại rằng trạng thái hành động có thể bao gồm một hoặc nhiều lệnh Visual Basic được thực thi tuần tự như trong ví dụ trên. Khi sử dụng lệnh lặp `Do...Loop While` để xây dựng ứng dụng, bạn sẽ cung cấp các trạng thái hành động và các điều kiện canh giữ thích hợp cho ứng dụng của bạn.



Hình 10.6 Lệnh lặp **Do...Loop While** trong biểu đồ hoạt động UML.

TỰ ÔN TẬP

1. Lệnh lặp Do...Loop `While` lặp lại khi điều kiện tiếp tục vòng lặp____.
   a) có giá trị `False` sau khi thân vòng lặp thực thi
   b) có giá trị `False` trước khi thân vòng lặp thực thi
   c) có giá trị `True` sau khi thân vòng lặp thực thi
   d) có giá trị `True` trước khi thân vòng lặp thực thi
2. Vòng lặp vô hạn xảy ra khi điều kiện tiếp tục vòng lặp trong lệnh `Do While`... `Loop` hay trong lệnh Do...Loop `While`____.
   a) không bao giờ có giá trị `True`
   b) không bao giờ có giá trị `False`
   c) có giá trị `False`
   d) được kiểm tra lặp lại nhiều lần

Đáp án: 1) c. 2) b.

10.3 Lệnh lặp Do...Loop Until

Lệnh lặp **Do...Loop Until** giống với lệnh Lệnh lặp Do Until... Loop, chỉ khác là điều kiện kết thúc vòng lặp của lệnh Do...Loop Until được kiểm tra *sau khi* thân vòng lặp được thực thi. Do vậy, thân vòng lặp thực thi ít nhất một lần. Khi lệnh Do...Loop Until kết thúc, ứng dụng tiếp tục thực thi lệnh sau từ khóa Loop Until.

Lỗi lập trình thường gặp

Vòng lặp vô hạn xảy ra khi điều kiện kết thúc vòng lặp của lệnh Do...Loop Until không bao giờ đạt giá trị True.

Chúng ta lại xét ví dụ đóng gói va li một lần nữa. Trước khi bạn bắt đầu đóng gói, hành lý rỗng. Bạn đặt một vật vào trong va li, sau đó xác định xem va li đã đầy chưa. Chừng nào điều kiện "va li đầy" là `False`, bạn tiếp tục đặt thêm các vật vào trong va li.

Xét đoạn ứng dụng dưới đây là một ví dụ về lệnh Do...Loop Until. Đoạn ứng dụng này hiển thị các số từ 1 đến 3 trên `ListBox`:

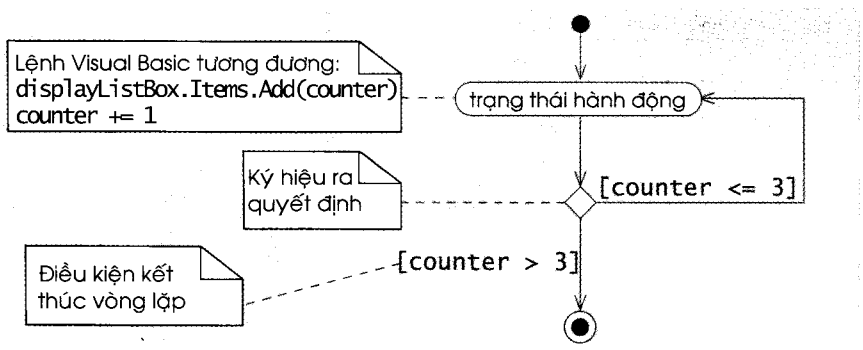
```
Dim counter As Integer = 1

Do
   displayListBox.Items.Add(counter)
   counter += 1
Loop Until counter > 3
```

Đoạn ứng dụng này khởi tạo giá trị 1 cho biến `counter` và điều kiện kết thúc vòng lặp của lệnh `Do...Loop Until` là `counter > 3`. Lệnh `Do...Loop Until` thực thi khi điều kiện kết thúc vòng lặp có giá trị `False`. Khi điều kiện kết thúc vòng lặp đạt giá trị `True`, lệnh `Do...Loop Until` kết thúc thực thi và `displayListBox` chứa các số từ 1 đến 3. Phần dưới đây mô tả từng bước lệnh lặp này thực thi.

Thực thi lệnh lặp Do...Loop Until

1. Ứng dụng khai báo biến `counter` và thiết lập giá trị 1 cho biến này.
2. Ứng dụng chuyển đến thực thi lệnh lặp `Do...Loop Until`.
3. Giá trị hiện đang được lưu trong biến `counter` (trong trường hợp này là 1) được thêm vào thuộc tính `Items` của `displayListBox`.
4. Giá trị biến `counter` được tăng thêm 1, lúc này biến `counter` có giá trị là 2.
5. Điều kiện kết thúc vòng lặp được kiểm tra. Điều kiện này có giá trị là `False` (`counter` không lớn hơn 3), bởi vậy ứng dụng tiếp tục thực thi lệnh đầu tiên sau lệnh `Do`.
6. Giá trị hiện đang được lưu trong biến `counter` (trong trường hợp này là 2) được thêm vào thuộc tính `Items` của `displayListBox`.
7. Giá trị biến `counter` được tăng thêm 1, lúc này biến `counter` có giá trị là 3.
8. Điều kiện kết thúc vòng lặp được kiểm tra. Điều kiện này có giá trị là `False` (`counter` không lớn hơn 3), bởi vậy ứng dụng tiếp tục thực thi lệnh đầu tiên sau lệnh `Do`.
9. Giá trị hiện đang được lưu trong biến `counter` (trong trường hợp này là 3) được thêm vào thuộc tính `Items` của `displayListBox`.
10. Giá trị biến `counter` được tăng thêm 1, lúc này biến `counter` có giá trị là 4.
11. Điều kiện kết thúc vòng lặp được kiểm tra. Điều kiện này có giá trị là `True` (`counter` lớn hơn 3), bởi vậy ứng dụng thoát khỏi lệnh lặp `Do... Loop Until`.

Biểu đồ hoạt động UML minh họa lệnh `Do...Loop Until` (Hình 10.7) chỉ rõ điều kiện kết thúc vòng lặp không được đánh giá cho đến tận khi thân vòng lặp được thực thi ít nhất một lần. Sơ đồ UML này có cùng điều kiện canh giữ giống như Hình 10.6. Sự khác nhau duy nhất là lệnh lặp `Do...Loop Until` tiếp tục thực thi khi điều kiện kết thúc vòng lặp có giá trị `False`. Khi điều kiện canh giữ đạt giá trị `True`, lệnh lặp kết thúc và điều khiển chương trình chuyển sang câu lệnh tiếp theo sau từ khóa `Loop Until`.



Hình 10.7 Biểu đồ hoạt động UML minh họa lệnh lặp `Do...Loop Until`.

TỰ ÔN TẬP

1. Lệnh Do...Loop `Until` kiểm tra điều kiện kết thúc vòng lặp_____.
   a) có giá trị `False` sau khi thân vòng lặp thực thi
   b) có giá trị `False` trước khi thân vòng lặp thực thi
   c) có giá trị `True` sau khi thân vòng lặp thực thi
   d) có giá trị `True` trước khi thân vòng lặp thực thi
2. Khi lệnh Do...Loop `Until` kết thúc, ứng dụng tiếp tục thực thi_____.
   a) mệnh đề Loop `Until`
   b) lệnh sau từ khóa Loop `Until`
   c) lệnh bên trong Do và Loop `Until`
   d) mệnh đề Do

Đáp án: 1) c. 2) b.

10.4 Tạo ứng dụng Class Average

Bạn đã học lệnh lặp Do...Loop `While` và Do...Loop `Until`, bây giờ bạn có thể bắt đầu phát triển ứng dụng **Class Average**. Đầu tiên, bạn sử dụng mã giả để liệt kê các hành động sẽ được thực thi và chỉ ra thứ tự thực hiện các hành động này. Bạn sử dụng lệnh lặp được điều khiển bởi biến đếm để nhập điểm. Nhớ lại rằng kỹ thuật này sử dụng biến đếm để xác định số lần một tập hợp lệnh được thực thi. Trong ví dụ này, việc lặp lại kết thúc khi biến đếm vượt quá giá trị 10, bởi vì chúng ta đã giả sử rằng người dùng chỉ nhập 10 điểm. Đoạn mã giả dưới đây mô tả các thao tác của ứng dụng **Class Average** khi nhấn vào `Button` **Add Grade** và khi nhấn vào `Button` **Average**.

```
Khi người dùng nhấn vào Button Add Grade
    Nếu điểm trung bình đã được tính trước đó
        Xóa nội dung trên Label hiển thị kết quả và ListBox

    Lấy giá trị điểm do người dùng nhập vào TextBox Enter grade:
    Hiển thị giá trị điểm này lên ListBox
    Xóa điểm trên TextBox Enter grade:
    Chuyển focus tới TextBox Enter grade:

    Nếu người dùng đã nhập đủ 10 điểm
        Vô hiệu hóa Button Add Grade
        Chuyển focus tới Button Average

Khi người dùng nhấn vào Button Average
    Gán giá trị 0 cho tổng điểm
    Gán giá trị 0 cho số lượng điểm

    Thực hiện các hành động sau

        Đọc giá trị điểm tiếp theo trên ListBox
        Cộng thêm giá trị điểm này vào tổng điểm
        Tăng số lượng điểm thêm 1
    Lặp lại khi số lượng điểm nhỏ hơn 10

    Tính điểm trung bình của cả lớp bằng cách chia tổng điểm cho 10
    Hiển thị giá trị điểm trung bình của cả lớp
    Kích hoạt Button Add Grade
    Chuyển focus tới TextBox Enter grade:
```

Bạn đã chạy thử ứng dụng **Class Average** và tìm hiểu về biểu diễn mã giả của ứng dụng này. Bây giờ bạn sẽ sử dụng bảng ACE để chuyển đoạn mã giả này thành mã Visual Basic. Hình 10.8 liệt kê các hành động, điều khiển và sự kiện giúp bạn hoàn thành cho mình phiên bản ứng dụng này.

Chúng ta sử dụng các `Label` mô tả là `gradeLabel`, `averageLabel` và `gradeListLabel` để gán nhãn cho các điều khiển của ứng dụng. Người dùng nhập điểm trên `gradeTextBox` và nhấn vào `addButton`. Sự kiện `Click` thêm

giá trị người dùng nhập trên `gradeTextBox` vào `ListBox` bằng cách sử dụng phương thức `gradesListBox.Items.Add`. Khi người dùng nhập vào 10 điểm và nhấn vào `averageButton`, ứng dụng lấy từng giá trị từ `ListBox`, cộng thêm vào tổng điểm và tính điểm trung bình của cả lớp bằng cách chia cho 10. Điểm trung bình của cả lớp được hiển thị trên `averageResultLabel`.

Bảng ACE cho ứng dụng Class Average

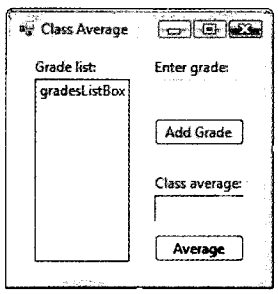
Hành động	Điều khiển	Sự kiện
Hiển thị Label cho các điều khiển của ứng dụng	gradeLabel, gradeListLabel, averageLabel,	
	addButton	Click
Nếu điểm trung bình đã được tính trước đó	averageResult-Label	
Xóa nội dung trên Label hiển thị kết quả và ListBox	averageResult-Label, gradesListBox	
Lấy giá trị điểm do người dùng nhập vào TextBox Enter grade:	gradeTextBox	
Hiển thị giá trị điểm này lên ListBox	gradesListBox	
Xóa điểm trên TextBox Enter grade:	gradeTextBox	
Chuyển focus tới TextBox Enter grade:	gradeTextBox	
Nếu người dùng đã nhập đủ 10 điểm	gradesListBox	
Vô hiệu hóa Button Add Grade	addButton	
Chuyển focus tới Button Average	averageButton	
	averageButton	Click
Gán giá trị 0 cho tổng điểm		
Gán giá trị 0 cho số lượng điểm		
Thực hiện các hành động sau		
Đọc giá trị điểm tiếp theo trên ListBox	gradesListBox	
Cộng thêm giá trị điểm này vào tổng điểm		
Tăng số lượng điểm thêm 1		
Lặp lại khi số lượng điểm nhỏ hơn 10		
Tính điểm trung bình cả lớp bằng cách chia tổng điểm cho 10		
Hiển thị giá trị điểm trung bình của cả lớp	averageResult-Label	
Kích hoạt Button Add Grade	addButton	
Chuyển focus tới TextBox Enter grade:	gradeTextBox	

Hình 10.8 Bảng ACE của ứng dụng **Class Average**.

Chúng ta vừa xây dựng giải thuật giải quyết vấn đề của ứng dụng **Class Average**. Bây giờ chúng ta bắt đầu thêm chức năng vào phần template của ứng dụng. Để hiển thị giá trị điểm mà người dùng nhập vào `TextBox` **Enter grade:** lên `ListBox` **Grade list:**, người dùng nhấn vào `Button` **Add Grade**. Nếu đã có điểm hiển thị trên `ListBox` **Grade list:** và điểm trung bình cả lớp trên `Label` **Class average:**, các giá trị này sẽ bị xóa đầu tiên. Phần dưới đây hướng dẫn bạn thêm chức năng này cho xử lý sự kiện của `Button` **Add Grade**.

Nhập điểm vào ứng dụng Class Average

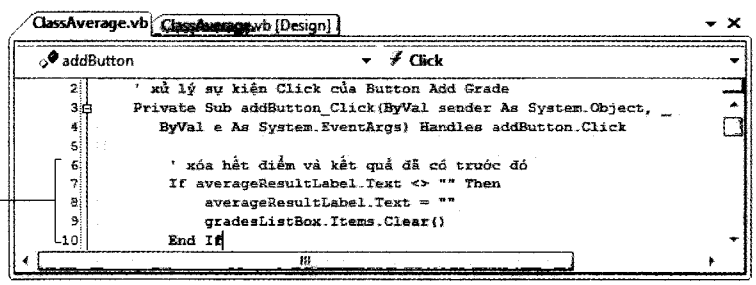
1. ***Copy template vào thư mục làm việc của bạn.*** Copy thư mục `C:\Examples\Tutorial10\TemplateApplication\ClassAverage` vào thư mục `C:\SimplyVB2008`.
2. ***Mở file template của ứng dụng Class Average.*** Nhấn đúp vào file `ClassAverage.sln` trong thư mục `ClassAverage` để mở ứng dụng trong IDE Visual Basic. Nhấn đúp vào file `ClassAverage.vb` trong cửa sổ **Solution Explorer** để hiển thị `Form` (Hình 10.9).



Hình 10.9 Giao diện ứng dụng **Class Average** ở chế độ **Design**.

3. ***Thêm một xử lý sự kiện cho Button Add Grade.*** Mỗi lần người dùng nhập một điểm số vào ứng dụng **Class Average**, họ phải nhấn vào `Button` **Add Grade**. Nhấn đúp vào `Button` **Add Grade** để tạo ra xử lý sự kiện `addButton_Click`.
4. ***Xóa nội dung hiển thị trên ListBox và kết quả từ lần tính toán trước trên Label Class Average:.*** Thêm dòng 6-10 (Hình 10.10) vào xử lý sự kiện `addButton_Click`. Hãy nhớ đặt chú thích trước mỗi xử lý sự kiện (dòng 2), và dùng ký tự nối dòng để phân tách các dòng quá dài (dòng 3-4). Để xác định xem đã có phép tính nào được thực hiện trước đó chưa, kiểm tra xem `averageResultLabel` có hiển thị điểm trung bình nào không bằng cách so sánh giá trị thuộc tính `Text` với chuỗi rỗng (dòng 7). Nếu `averageResultLabel` hiển thị kết quả của phép tính trước, hãy gán giá trị là chuỗi rỗng cho thuộc tính `Text` (dòng 8). Dòng 9 xóa các điểm đang hiển thị trên `ListBox`.

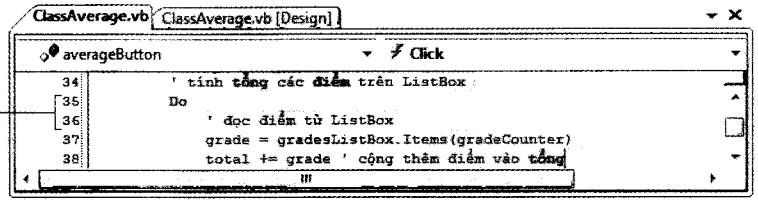
Xóa danh sách điểm và điểm trung bình của cả lớp



Hình 10.10 Xóa nội dung hiển thị trên `Label` đầu ra và `ListBox`.

5. ***Hiển thị từng điểm số trên điều khiển ListBox***. Thêm dòng 12-14 trên Hình 10.11 vào xử lý sự kiện `addButton_Click`, bên dưới lệnh `If...Then`. Dòng 13 thêm giá trị điểm được nhập trên `gradeTextBox` vào thuộc tính `Items` của `gradesListBox`. Điểm này sẽ được hiển thị trên `ListBox`.

Thêm điểm vào Listbox và xóa điểm do người dùng nhập vào TextBox



Hình 10.11 Thêm giá trị điểm do người dùng nhập vào `ListBox` và xóa nội dung trên `TextBox` **Enter grade:**.

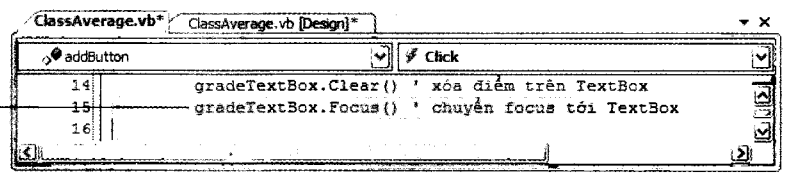
6. ***Chuẩn bị cho lần nhập điểm mới***. Phương thức `Clear` (dòng 14, Hình 10.11) xóa giá trị điểm trên `TextBox` để chuẩn bị cho lần nhập điểm mới vào ứng dụng. Sử dụng phương thức này có chức năng giống với lệnh gán chuỗi rỗng `""` hay `String.Empty` cho thuộc tính `Text` của `TextBox`.
7. ***Lưu project***. Chọn **File > Save All** để lưu mã đã được sửa đổi.

Bạn vừa thêm đoạn mã để hiển thị giá trị điểm được nhập ở `TextBox` **Enter grade:** lên `ListBox` khi người dùng nhấn vào `Button` **Add Grade**. Tiếp theo, bạn sẽ học cách chuyển focus tới `TextBox` này cho lần nhập giá trị điểm tiếp theo sau khi người dùng nhấn vào `Button` **Add Grade**. Phần dưới đây cũng chỉ cho bạn cách vô hiệu hóa `Button` **Add Grade** sau khi nhập đủ 10 điểm, khi đó chức năng của `Button` này không còn cần thiết nữa.

Chuyển Focus tới một điều khiển và vô hiệu hóa Button

1. ***Chuyển focus tới một điều khiển***. Thêm dòng 15 (Hình 10.12) vào xử lý sự kiện `addButton_Click`. Dòng 15 gọi phương thức **Focus** của `gradeTextBox` để đặt con trỏ vào `TextBox` này cho lần nhập điểm tiếp theo. Quá trình này được gọi là **chuyển focus (transferring the focus)**. Bây giờ focus được chuyển từ `Button` người dùng vừa nhấn tới `TextBox` mà người dùng sẽ nhập điểm số tiếp theo.

Chuyển focus của ứng dụng tới TextBox



Hình 10.12 Chuyển focus tới điều khiển `TextBox`.

Mẹo thiết kế giao diện

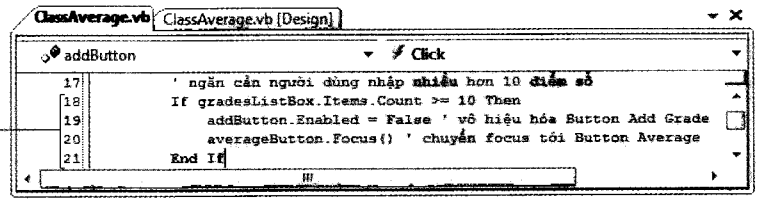
Hãy chuyển focus tới điều khiển sẽ được dùng tiếp theo.

Mẹo thiết kế giao diện

Hãy vô hiệu hóa `Button` khi chức năng của nó không nên cung cấp cho người dùng.

2. ***Vô hiệu hóa Button Add Grade để ngăn người dùng nhập nhiều hơn 10 điểm số***. Ứng dụng chỉ chấp nhận đúng 10 điểm số. Nếu số lượng điểm đã nhập vào là 10, ứng dụng sẽ ngăn không cho người dùng nhập thêm. Thêm dòng 17-21 trên Hình 10.13 vào xử lý sự kiện `addButton_Click`. Dòng 18 xác định xem số lượng điểm được nhập là 10 hay nhiều hơn, sử dụng toán tử so sánh `>=`. Thuộc tính **Count** của `Items` trả lại số lượng phần tử hiển thị trên `Listbox` **Grade list:**. Nếu số lượng điểm là 10, dòng 19 vô hiệu hóa `addButton` bằng cách thiết lập thuộc tính **Enabled** với giá trị `False`. Bây giờ, nhấn vào `Button` **Add Grade** đã bị vô hiệu hóa sẽ không làm xử lý sự kiện thực thi.

Vô hiệu hóa Button **Add Grade** và chuyển focus tới Button **Average**



Hình 10.13 Ứng dụng chỉ chấp nhận 10 điểm số.

3. ***Chuyển focus tới Button Average sau 10 lần nhập điểm***. Sau khi có 10 điểm được nhập, ứng dụng sẽ không chuyển focus tới TextBox. Thay vì vậy, dòng 20 gọi phương thức Focus để chuyển focus sang Button **Average**. Do đó, bạn có thể bấm phím *Enter* hoặc *phím khoảng trắng* (*Space*) để gọi xử lý sự kiện của Button **Average** mà không cần trỏ chuột vào Button này.
4. ***Lưu project***. Chọn **File > Save All** để lưu mã đã được sửa đổi.

Lỗi lập trình thường gặp

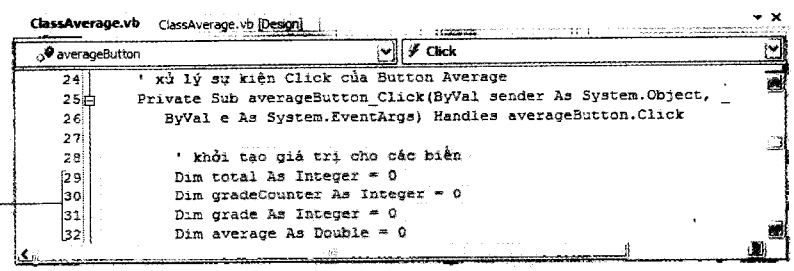
Điều khiển phải được kích hoạt thì mới có thể nhận được focus.

Sau khi nhập vào 10 điểm và hiển thị chúng trên ListBox, xử lý sự kiện của Button **Add Grade** sẽ chuyển focus tới Button **Average**. Khi người dùng nhấn vào Button **Average**, ứng dụng tính và hiển thị điểm trung bình của 10 điểm này. Phần dưới đây hướng dẫn bạn cách tính tổng điểm trước khi tính trung bình bằng lệnh lặp Do...Loop Until. Phần này cũng hướng dẫn cách hiển thị kết quả lên Label **Class average:**.

Tính điểm trung bình của cả lớp

1. ***Thêm xử lý sự kiện cho Button Average***. Nhấn đúp vào Button **Average** để tạo xử lý sự kiện averageButton_Click.
2. ***Khởi tạo giá trị các biến được dùng để tính điểm trung bình***. Thêm dòng 28-32 trong Hình 10.14 vào xử lý sự kiện averageButton_Click. Dòng 29 khai báo biến total kiểu Integer. Bạn sử dụng biến total để tính tổng của 10 điểm (bạn cần tính tổng này để tính điểm trung bình). Dòng 30 khai báo biến đếm (gradeCounter). Một điểm vô cùng quan trọng là các biến tổng hay biến đếm phải được khởi tạo giá trị thích hợp trước khi sử dụng. Nếu biến kiểu số không được khởi tạo trước khi sử dụng lần đầu tiên, Visual Basic sẽ tự động khởi tạo giá trị mặc định là 0 cho biến này. Tuy nhiên, trên Hình 10.14, tất cả các biến đều được khởi tạo giá trị là 0 bằng cách viết mã. Điều này làm cho chương trình rõ ràng hơn. Biến grade (dòng 31) lưu tạm thời giá trị điểm được đọc ra từ ListBox. Mặc dù các điểm được nhập vào là các số nguyên, nhưng kết quả tính trung bình là giá trị có dấu phẩy động (ví dụ kết quả là 81.10 trên Hình 10.4); do đó, bạn khai báo biến average kiểu Double (dòng 32) để lưu giá trị điểm trung bình này.

Khởi tạo giá trị cho các biến



Hình 10.14 Bước khởi tạo giá trị để tính điểm trung bình.

3. ***Tính tổng của các điểm đang được hiển thị trên ListBox***. Thêm dòng 34-40 trong Hình 10.15 vào xử lý sự kiện averageButton_Click. Lệnh

`Do...Loop Until` (dòng 35-40) tính tổng các điểm được đọc ra từ `ListBox`. Dòng 40 chỉ ra rằng lệnh này sẽ lặp cho đến tận khi giá trị `gradeCounter` lớn hơn hoặc bằng 10. [*Lưu ý*: Trong bài tập 10.12 bạn sẽ chỉnh sửa ứng dụng này để xử lý với mọi số lượng điểm nhập vào.] Dòng 37 đọc giá trị hiện tại từ `ListBox` bằng cách sử dụng thuộc tính `Items` của `ListBox` và lưu giá trị này vào biến `grade`. Các phần tử của `ListBox` được truy cập thông qua số thứ tự, bắt đầu từ 0. Dòng 38 sử dụng toán tử gán += để cộng giá trị biến `grade` với giá trị biến `total` và gán kết quả này trở lại biến `total`. Biến `gradeCounter` được tăng lên (dòng 39) để chỉ ra rằng điểm số tiếp theo sẽ được xử lý. (Biến đếm được tăng lên để đảm bảo rằng điều kiện ở dòng 40 cuối cùng sẽ đạt giá trị `True` và vòng lặp kết thúc.)

Sử dụng lệnh lặp `Do...Loop Until` để tính tổng của các điểm trong `ListBox`

```
34          ' tính tổng các điểm trên ListBox
35          Do
36              ' đọc điểm từ ListBox
37              grade = gradesListBox.Items(gradeCounter)
38              total += grade ' cộng thêm điểm vào tổng
39              gradeCounter += 1 ' tăng biến đếm
40          Loop Until gradeCounter >= 10
```

Hình 10.15 Lệnh `Do...Loop Until` tính tổng điểm.

Mẹo thiết kế giao diện

Kích hoạt `Button` đang bị vô hiệu hóa khi chức năng của nó cần được cung cấp cho người dùng.

4. ***Tính và hiển thị điểm trung bình***. Thêm dòng 42-45 trên Hình 10.16 vào xử lý sự kiện `averageButton_Click`. Dòng 42 gán kết quả tính trung bình cho biến `average`. Dòng 43 hiển thị giá trị biến `average`. Chú ý sử dụng ký tự chỉ thị định dạng F để hiển thị giá trị `average` dưới định dạng dấu phẩy động. Sau khi giá trị trung bình được hiển thị, người dùng có thể nhập thêm một tập hợp 10 điểm khác. Để làm được việc này, bạn cần kích hoạt `Button` **Add Grade** bằng cách thiết lập giá trị `True` cho thuộc tính `Enabled` (dòng 44). Dòng 45 chuyển focus tới `TextBox` **Enter grade:**.

Tính điểm trung bình của cả lớp, kích hoạt `Button` **Add Grade** và chuyển focus tới `TextBox` **Enter Grade:**

```
42          average = total / 10 ' tính điểm trung bình
43          averageResultLabel.Text = String.Format("{0:F}", average)
44          addButton.Enabled = True ' kích hoạt Button Add Grade
45          gradeTextBox.Focus() ' chuyển focus tới TextBox Enter grade:
46      End Sub ' averageButton_Click
```

Hình 10.16 Hiển thị kết quả tính trung bình.

5. ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng của bạn. Ứng dụng này có thể tính và hiển thị điểm trung bình của cả lớp. Sử dụng `TextBox` **Enter grade:** và `Button` **Add Grade** để nhập vào 10 điểm số (như trên Hình 10.3). Sau khi 10 điểm số đã được nhập, nhấn vào `Button` **Average** và kiểm tra xem giá trị trung bình đã hiển thị đúng chưa.
6. ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.
7. ***Đóng IDE***. Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

Hình 10.17 biểu diễn mã nguồn của ứng dụng **Class Average**. Những dòng mã được đánh dấu chứa khái niệm lập trình mới mà bạn đã tìm hiểu trong chương này.

```
1  Public Class ClassAverageForm
2     ' xử lý sự kiện Click của Button Add Grade
3     Private Sub addButton_Click(ByVal sender As System.Object, _
4        ByVal e As System.EventArgs) Handles addButton.Click
5
6        ' xóa hết điểm và kết quả đã có trước đó
7        If averageResultLabel.Text <> "" Then
8           averageResultLabel.Text = ""
9           gradesListBox.Items.Clear()
10       End If
11
12       ' hiển thị điểm trên ListBox
13       gradesListBox.Items.Add(Val(inputTextBox.Text))
14       gradeTextBox.Clear() ' xóa điểm trên TextBox
15       gradeTextBox.Focus() ' chuyển focus tới TextBox
16
17       ' ngăn cản người dùng nhập nhiều hơn 10 điểm số
18       If gradesListBox.Items.Count >= 10 Then
19          addButton.Enabled = False ' vô hiệu hóa Button Add Grade
20          averageButton.Focus() ' chuyển focus tới Button Average
21       End If
22    End Sub ' addButton_Click
23
24    ' xử lý sự kiện Click của Button Average
25    Private Sub averageButton_Click(ByVal sender As System.Object, _
26       ByVal e As System.EventArgs) Handles averageButton.Click
27
28       ' khởi tạo giá trị cho các biến
29       Dim total As Integer = 0
30       Dim gradeCounter As Integer = 0
31       Dim grade As Integer = 0
32       Dim average As Double = 0
33
34       ' tính tổng các điểm trên ListBox
35       Do
36          ' đọc điểm từ ListBox
37          grade = gradesListBox.Items(gradeCounter)
38          total += grade ' cộng thêm điểm vào tổng
39          gradeCounter += 1 ' tăng biến đếm
40       Loop Until gradeCounter >= 10
41
42       average = total / 10 ' tính điểm trung bình
43       averageResultLabel.Text = String.Format("{0:F}", average)
44       addButton.Enabled = True ' kích hoạt Button Add Grade
45       gradeTextBox.Focus() ' chuyển focus tới TextBox Enter grade:
46    End Sub ' averageButton_Click
47 End Class ' ClassAverageForm
```

Xóa điểm số trên `gradeTextBox`
Chuyển focus tới `gradeTextBox`
Vô hiệu hóa **`Button` Add Grade** và chuyển focus tới **`Button` Average**
Truy cập điểm trong `ListBox` thông qua thuộc tính `Items`
Sử dụng lệnh lặp `Do...Loop Until` để tính tổng điểm
Kích hoạt **`Button` Add Grade** và chuyển focus tới **`TextBox` Enter grade:**

Hình 10.17 Mã nguồn của ứng dụng **Class Average**.

TỰ ÔN TẬP

1. Nếu bạn không muốn `Button` gọi phương thức xử lý sự kiện khi nó được nhấn, hãy thiết lập thuộc tính_____giá trị là_____.
   a) `Enabled, False` b) `Enabled, True`
   c) `Disabled, True` d) `Disabled, False`
2. _____một `TextBox` thì `TextBox` này sẽ nhận dữ liệu do người dùng nhập vào.
   a) Kích hoạt b) Xóa
   c) Chuyển focus tới d) Vô hiệu

Đáp án: **1)** a. **2)** c.

10.5 Tổng kết

Trong chương này, bạn đã học cách sử dụng lệnh lặp `Do...Loop While` và `Do... Loop Until`. Chúng ta đã được giới thiệu về cú pháp và biểu đồ hoạt động UML (là biểu đồ giải thích mỗi lệnh này thực thi như thế nào). Bạn đã dùng lệnh

Do...Loop Until trong ứng dụng **Class Average**.

Lệnh lặp Do...Loop While thực thi chừng nào điều kiện tiếp tục vòng lặp đạt giá trị True. Lệnh lặp này luôn luôn thực thi ít nhất một lần. Khi điều kiện tiếp tục vòng lặp đạt giá trị False, lệnh lặp kết thúc. Lệnh này lặp vô hạn nếu điều kiện tiếp tục vòng lặp không bao giờ đạt giá trị False.

Lệnh lặp Do...Loop Until luôn luôn thực thi ít nhất một lần. Lệnh này thực thi chừng nào điều kiện kết thúc vòng lặp đạt giá trị False. Khi điều kiện kết thúc vòng lặp đạt giá trị True, việc lặp lại kết thúc. Lệnh này lặp vô hạn nếu điều kiện kết thúc vòng lặp không bao giờ đạt giá trị True.

Bạn cũng đã tìm hiểu kỹ thuật để tạo giao diện ứng dụng tiện dụng cho người dùng. Bây giờ bạn đã biết cách gọi phương thức Focus để chuyển focus tới một điều khiển nhằm chỉ ra hành động tiếp theo của người dùng cần phải thực hiện liên quan đến điều khiển này. Bạn cũng đã học cách vô hiệu hóa Button để nó không còn có hiệu lực đối với người dùng tại một thời điểm trong quá trình thực thi ứng dụng và bạn cũng đã học cách kích hoạt trở lại các Button này.

Trong chương tiếp theo, bạn tiếp tục tìm hiểu về lệnh lặp. Bạn sẽ học cách sử dụng lệnh lặp For...Next. Đây là một lệnh lặp được điều khiển bởi biến đếm vô cùng hữu ích.

TỔNG KẾT KỸ NĂNG

Lệnh lặp Do...Loop While

- Lặp khi điều kiện tiếp tục vòng lặp có giá trị là True.
- Kiểm tra điều kiện tiếp tục vòng lặp sau khi thân vòng lặp được thực thi.
- Luôn thực thi thân vòng lặp ít nhất một lần.
- Trở thành vòng lặp vô hạn khi điều kiện tiếp tục vòng lặp không bao giờ đạt giá trị False.

Lệnh lặp Do...Loop Until

- Lặp khi điều kiện kết thúc vòng lặp có giá trị False.
- Kiểm tra điều kiện kết thúc vòng lặp sau khi thân vòng lặp được thực thi.
- Luôn thực thi thân vòng lặp ít nhất một lần.
- Trở thành vòng lặp vô hạn khi điều kiện kết thúc vòng lặp không bao giờ đạt giá trị True.

Vô hiệu hóa Button

- Thiết lập giá trị Fasle cho thuộc tính Enabled.

Xác định số lượng Items có trong ListBox

- Sử dụng thuộc tính Count của thuộc tính Items của ListBox.

Kích hoạt Button

- Thiết lập giá trị True cho thuộc tính Enabled.

Chuyển Focus tới một điều khiển

- Gọi phương thức Focus.

THUẬT NGỮ

chuyển focus - Chuyển con trỏ đến một điều khiển trong ứng dụng.

lệnh lặp Do...Loop Until - Lệnh điều khiển thực thi một tập các lệnh ít nhất một lần cho đến khi điều kiện kết thúc vòng lặp đạt giá trị True sau khi thực thi vòng lặp lần đầu tiên.

lệnh lặp Do...Loop While - Lệnh điều khiển thực thi một tập hợp các lệnh ít nhất một lần khi điều kiện tiếp tục vòng lặp đạt giá trị True sau khi thực thi vòng lặp lần đầu tiên.

lỗi kết thúc vòng lặp sớm hoặc muộn (off-by-one error) - Lỗi lôgíc này xảy ra khi vòng lặp thực thi ít hơn hoặc nhiều hơn một lần lặp so với dự định.

phương thức Focus - Chuyển focus của ứng dụng tới điều khiển gọi phương thức này.

thuộc tính Count của Items - Trả lại số lượng phần tử của `ListBox`.

thuộc tính Enabled - Chỉ định điều khiển (ví dụ như `Button`) được kích hoạt (`True`) hay vô hiệu hóa (`False`).

HƯỚNG DẪN THIẾT KẾ GIAO DIỆN

Thiết kế tổng quan

- Chuyển focus tới điều khiển sẽ được sử dụng tiếp theo.

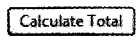
Button

- Vô hiệu hóa `Button` khi chức năng của nó không nên cung cấp cho người dùng.
- Kích hoạt `Button` đang bị vô hiệu hóa khi chức năng của nó cần được cung cấp cho người dùng.

ĐIỀU KHIỂN, SỰ KIỆN, THUỘC TÍNH & PHƯƠNG THỨC

Button ab Button Khi nhấn vào `Button` sẽ ra lệnh cho ứng dụng thực hiện một hành động.

- ***Trên giao diện khi ứng dụng chạy***



- ***Sự kiện***

  `Click` - Được kích hoạt khi người dùng nhấn vào `Button`.

- ***Thuộc tính***

  `Enabled` - Xác định xem có thực thi xử lý sự kiện của `Button` hay không khi nhấn vào `Button` này.

  `Location` - Chỉ định vị trí tương đối của `Button` so với góc trên cùng bên trái của `Form`.

  `Name` - Chỉ ra tên được sử dụng để truy cập `Button` trong khi lập trình. Tên nên được thêm hậu tố `Button` vào phía sau.

  `Size` - Chỉ định chiều rộng và chiều cao (bằng pixel) của `Button`.

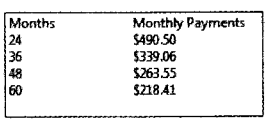
  `Text` - Chỉ ra đoạn văn bản được hiển thị trên `Button`.

- ***Phương thức***

  `Focus` - Chuyển focus của ứng dụng tới `Button` gọi phương thức này.

ListBox ListBox Điều khiển này cho phép người dùng xem và lựa chọn các phần tử trong một danh sách.

- ***Trên giao diện khi ứng dụng chạy***



- ***Thuộc tính***

  `Items` - Trả lại đối tượng chứa các phần tử được hiển thị trên `ListBox`.

  `Items.Count` - Trả lại số lượng phần tử có trong `ListBox`.

  `Location` - Chỉ định vị trí tương đối của `ListBox` so với góc trên cùng bên trái của `Form`.

  `Name` - Chỉ ra tên được sử dụng để truy cập `ListBox` trong khi lập trình. Tên nên được thêm hậu tố `ListBox` vào phía sau.

  `Size` - Chỉ định chiều rộng và chiều cao (bằng pixel) của `ListBox`.

- ***Phương thức***

  `Items.Add` - Thêm một phần tử vào thuộc tính `Items`.

  `Items.Clear` - Xóa tất cả phần tử có trong thuộc tính `Items` của `ListBox`.

TextBox TextBox Điều khiển này cho phép người dùng nhập liệu từ bàn phím.

- ***Trên giao diện khi ứng dụng chạy***

- ***Sự kiện***

  TextChanged - Được kích hoạt khi nội dung đoạn văn bản trong TextBox bị thay đổi.

- ***Thuộc tính***

  Location - Chỉ ra vị trí tương đối của TextBox so với góc trên cùng bên trái của Form.

  Name - Chỉ ra tên được sử dụng để truy cập TextBox trong khi lập trình. Tên nên được thêm hậu tố TextBox vào phía sau.

  Size - Chỉ định chiều rộng và chiều cao (bằng pixel) của TextBox.

  Text - Chỉ định đoạn văn bản được hiển thị ban đầu trên TextBox.

  TextAlign - Chỉ định cách thức căn lề đoạn văn bản trong TextBox.

  Width - Chỉ định chiều rộng (bằng pixel) của TextBox.

- ***Phương thức***

  Clear - Xóa nội dung đoạn văn bản đang hiển thị trên TextBox, gọi phương thức này.

  Focus - Chuyển focus của ứng dụng tới TextBox, gọi phương thức này.

CÂU HỎI TRẮC NGHIỆM

10.1 _____xảy ra khi điều kiện tiếp tục vòng lặp trong lệnh Do...Loop While không bao giờ đạt giá trị False.

a) Vòng lặp vô hạn
b) Vòng lặp được điều khiển bởi biến đếm
c) Lệnh điều khiển
d) Lệnh điều khiển lồng nhau

10.2 Thiết lập giá trị True cho thuộc tính_____để kích hoạt Button.

a) Disabled
b) Focus
c) Enabled
d) ButtonEnabled

10.3 Lệnh_____thực thi ít nhất một lần và tiếp tục thực thi cho đến khi điều kiện kết thúc vòng lặp đạt giá trị True.

a) Do While...Loop
b) Do...Loop Until
c) Do...Loop While
d) Do Until...Loop

10.4 Lệnh_____thực thi ít nhất một lần và tiếp tục thực thi cho đến tận khi điều kiện tiếp tục vòng lặp đạt giá trị False.

a) Do...Loop Until
b) Do Until...Loop
c) Do While...Loop
d) Do...Loop While

10.5 Phương thức_____chuyển focus tới một điều khiển.

a) GetFocus
b) Focus
c) Transfer
d) Activate

10.6 _____ chứa tổng của các giá trị.

a) Biến total
b) Biến đếm
c) Điều kiện
d) Vòng lặp

10.7 Thuộc tính_____của_____chứa số lượng phần tử của ListBox.

a) Count, ListBox
b) ListCount, Items
c) ListCount, ListBox
d) Count,Items

10.8 _____xảy ra khi vòng lặp thực thi ít hơn hoặc nhiều hơn một lần lặp so với số vòng lặp cần thiết.

a) Vòng lặp vô hạn
b) Vòng lặp được điều khiển bởi biến đếm
c) Lỗi kết thúc vòng lặp sớm hoặc muộn
d) Lệnh điều khiển lồng

10.9 Điều kiện kết thúc vòng lặp của lệnh lặp `Do...Loop Until` được đánh giá _____.

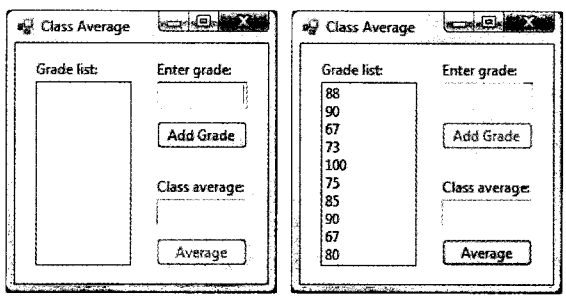
a) chỉ trong lần đầu thân vòng lặp được thực thi
b) trước khi thực thi thân vòng lặp
c) sau khi thực thi thân vòng lặp
d) Các lựa chọn trên đều sai

10.10 Nếu điều kiện tiếp tục vòng lặp được khởi tạo giá trị `False`, lệnh lặp `Do... Loop While`_____.

a) không bao giờ thực thi
b) thực thi khi điều kiện có giá trị là `False`
c) thực thi cho đến khi điều kiện đạt giá trị `True`
d) thực thi chỉ một lần

BÀI TẬP

10.11 (***Chỉnh sửa ứng dụng Class Average***) Sửa lại ứng dụng **Class Average** giống như Hình 10.18, để `Button` **Average** bị vô hiệu hóa cho đến khi số lượng điểm nhập vào là 10.



Hình 10.18 Ứng dụng **Class Average** đã chỉnh sửa.

a) ***Copy template của ứng dụng vào thư mục làm việc***. Copy thư mục `C:\Examples\Tutorial10\Exercises\ModifiedClassAverage` vào thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng***. Nhấn đúp vào file `ClassAverage.sln` trong thư mục `ModifiedClassAverage` để mở ứng dụng.

c) ***Ban đầu vô hiệu hóa `Button` Average***. Sử dụng cửa sổ **Properties** để vô hiệu hóa `Button` **Average** khi ứng dụng thực thi lần đầu tiên bằng cách thiết lập giá trị `False` cho thuộc tính `Enabled`.

d) ***Kích hoạt `Button` Average sau khi nhập đủ 10 điểm số***. Thêm mã vào xử lý sự kiện `addButton_Click` để kích hoạt `Button` **Average** sau khi có 10 điểm số được nhập.

e) ***Vô hiệu hóa `Button` Average sau khi thực hiện tính toán***. Thêm đoạn mã vào xử lý sự kiện `averageButton_Click` để vô hiệu hóa `Button` **Average** sau khi hiển thị kết quả tính toán.

f) ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng của bạn. Nhập vào 10 điểm số và đảm bảo rằng `Button` **Average** bị vô hiệu hóa cho đến khi tất cả 10 điểm số được nhập. Kiểm tra xem `Button` **Add Grade** đã bị vô hiệu hóa sau khi 10 điểm được nhập hay chưa. Khi `Button` **Average** được kích hoạt, nhấn vào `Button` này và kiểm tra xem

điểm trung bình hiển thị chính xác chưa. Sau khi tính trung bình, `Button` **Average** sẽ bị vô hiệu hóa trở lại và `Button` **Add Grade** được kích hoạt.

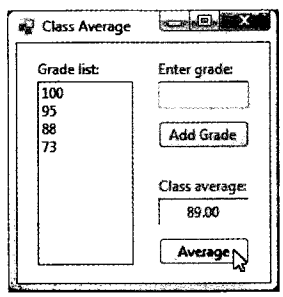
g) ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

h) ***Đóng IDE***. Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

10.12 (***Ứng dụng Class Average xử lý với mọi số lượng điểm nhập vào***) Viết lại ứng dụng **Class Average** để xử lý với mọi số lượng điểm nhập vào, như Hình 10.19. Chú ý rằng ứng dụng không biết người dùng nhập vào bao nhiêu điểm, bởi vậy `Button` phải luôn được kích hoạt.

a) ***Copy template của ứng dụng vào thư mục làm việc***. Copy thư mục `C:\Examples\Tutorial10\Exercises\UndeterminedClassAverage` vào thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng***. Nhấn đúp vào file `ClassAverage.sln` trong thư mục `UndeterminedClassAverage` để mở ứng dụng này.



Hình 10.19 Ứng dụng **Class Average** đã chỉnh sửa để xử lý với số lượng điểm nhập vào không được biết trước.

c) ***Không bao giờ vô hiệu hóa Button Add Grade***. Xóa đoạn mã vô hiệu hóa `Button` **Add Grade** khi nhập đủ 10 điểm từ xử lý sự kiện `addButton_Click`.

d) ***Tính tổng các điểm số trong ListBox***. Sửa đoạn mã trong xử lý sự kiện `averageButton_Click` để tăng biến `gradeCounter` cho đến khi giá trị biến này bằng số lượng điểm được nhập. Sử dụng `gradesListBox.Items.Count` để xác định số lượng phần tử trong `ListBox`. Kết quả số lượng được trả về bởi thuộc tính `Count` sẽ là 0 khi không có điểm nào được nhập. Sử dụng lệnh lựa chọn `If...Then` để tránh phép chia cho 0 và hiển thị một hộp thoại thông báo cho người dùng rằng chưa có điểm nào được nhập khi người dùng nhấn vào `Button` **Average**.

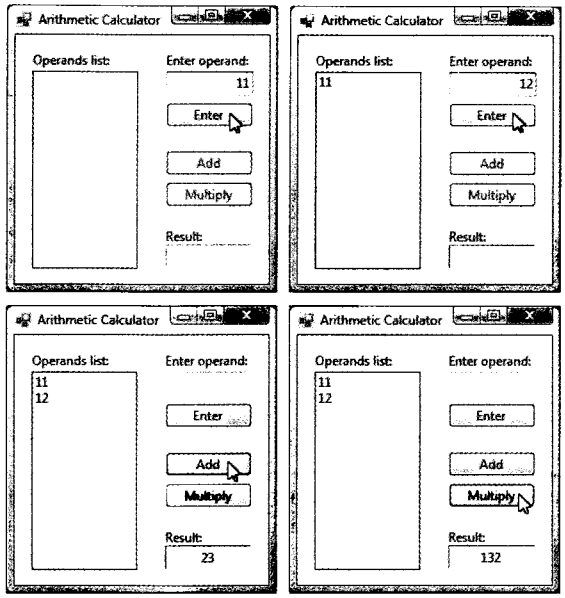
e) ***Tính điểm trung bình của cả lớp***. Chỉnh sửa đoạn mã trong xử lý sự kiện `averageButton_Click` để tính điểm trung bình bằng cách sử dụng số lượng điểm thực tế chứ không dùng giá trị 10.

f) ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng của bạn. Nhập vào 10 điểm số và nhấn vào `Button` **Average**. Kiểm tra xem điểm trung bình hiển thị đã chính xác chưa. Làm lại cùng các hành động trên nhưng lần này cho 15 điểm, sau đó cho 5 điểm. Sau mỗi lần, kiểm tra sự chính xác của kết quả hiển thị.

g) ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

h) ***Đóng IDE***. Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

10.13 (***Ứng dụng Arithmetic Calculator***) Viết ứng dụng cho phép người dùng nhập vào một dãy số và thao tác với chúng. Ứng dụng cho phép người dùng chọn cộng hoặc nhân các số này. Người dùng nhập từng số vào một `Textbox`. Sau khi nhập một số, người dùng nhấn vào `Button` **Enter**, số này sẽ được chèn vào `ListBox`. Giao diện của ứng dụng như trên Hình 10.20.



Hình 10.20 Ứng dụng **Arithmetic Calculator**.

a) ***Copy template của ứng dụng vào thư mục làm việc***. Copy thư mục `C:\Examples\Tutorial10\Exercises\ArithmeticCalculator` tới thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng***. Nhấn đúp vào file `ArithmeticCalculator.sln` trong thư mục `ArithmeticCalculator` để mở ứng dụng.

c) ***Thêm `ListBox` để hiển thị các con số được nhập***. Thêm `ListBox` có vị trí và kích thước như trên Hình 10.20.

d) ***Tạo xử lý sự kiện cho `Button` Enter***. Tạo xử lý sự kiện `Click` cho `Button` **Enter**. Nếu kết quả của phép tính trước đang được hiển thị, xử lý sự kiện sẽ xóa kết quả này, xóa các phần tử trên `ListBox` và vô hiệu hóa `Button` **Add** và **Multiply**. Sau đó thêm số vừa được nhập vào `ListBox` **Operands list:**. Khi `ListBox` chứa ít nhất hai con số, xử lý sự kiện sẽ kích hoạt các `Button` **Add** và **Multiply**.

e) ***Cộng các giá trị trong `ListBox`***. Thêm xử lý sự kiện `Click` cho `Button` **Add**. Xử lý sự kiện này, tính tổng tất cả các giá trị trong `ListBox` **Operands list:** và hiển thị kết quả lên `resultLabel`.

f) ***Nhân các giá trị trong `ListBox`***. Thêm xử lý sự kiện `Click` cho `Button` **Multiply**. Xử lý sự kiện này, tính tích tất cả các giá trị trong `ListBox` **Operands list:** và hiển thị kết quả lên `resultLabel`.

g) ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng. Nhập hai giá trị, sau đó nhấn vào `Button` **Add** và **Multiply**. Kiểm tra xem kết quả hiển thị đúng chưa. Đảm bảo rằng các `Button` **Add** và **Multiply** không được kích hoạt cho đến khi có hai giá trị được nhập. Nhập một giá trị mới và kiểm tra kết quả của lần tính toán trước và các phần tử trên `ListBox` có được xóa không. Nhập thêm hai giá trị nữa, sau đó nhấn vào `Button` **Add** và **Multiply**. Kiểm tra xem kết quả đã hiển thị đúng chưa.

h) ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

i) ***Đóng IDE***. Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

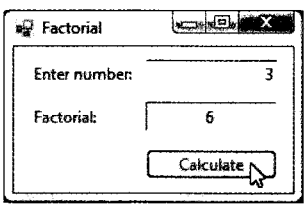
Đoạn mã này làm gì? ▶ **10.14** Kết quả của đoạn mã dưới đây là gì?

```
1  Dim y As Integer
2  Dim x As Integer
3  Dim mysteryValue As Integer
4
5  x = 1
6  mysteryValue = 0
7
8  Do
9     y = x ^ 2
10    displayListBox.Items.Add(y)
11    mysteryValue += 1
12    x += 1
13 Loop While x <= 10
14
15 resultLabel.Text = mysteryValue
```

Đoạn mã này có gì sai? ▶ **10.15** Tìm lỗi trong đoạn mã dưới đây. Đoạn mã này cộng giá trị của y với 10 và lưu kết quả vào biến z. Sau đó giảm giá trị y đi một đơn vị và lặp lại công việc này cho đến khi y nhỏ hơn 10. Cuối cùng, `resultLabel` sẽ hiển thị giá trị cuối cùng của z.

```
1  Dim y As Integer = 10
2  Dim z As Integer = 2
3
4  Do
5     z = y + 10
6  Loop Until y < 10
7
8  y -= 1
9
10 resultLabel.Text = z
```

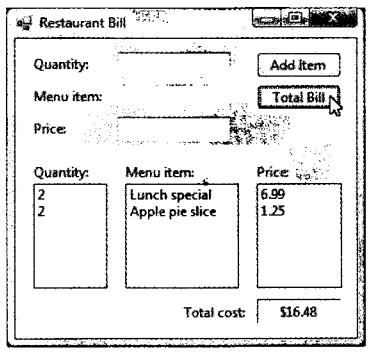
Sử dụng trình gỡ lỗi ▶ **10.16** (***Ứng dụng Factorial***) Ứng dụng **Factorial** tính giai thừa của một số nguyên do người dùng nhập vào. Giai thừa của một số nguyên là tích của các số nguyên từ 1 tới số nguyên đó. Ví dụ, giai thừa của 3 là 6 (1 x 2 x 3). Copy ứng dụng **Factorial** từ thư mục `C:\Examples\Tutorial10\Exercises\Factorial` vào thư mục làm việc của bạn. Trong khi xem xét ứng dụng, bạn sẽ nhận thấy rằng nó thực thi không chính xác. Sử dụng trình gỡ lỗi để tìm và sửa lỗi lôgíc trong ứng dụng. Hình 10.21 hiển thị kết quả đúng của ứng dụng **Factorial**.



Hình 10.21 Kết quả đúng của ứng dụng **Factorial**.

Bài tập nâng cao

▶ **10.17** (***Ứng dụng Restaurant Bill***) Phát triển ứng dụng tính hóa đơn cho nhà hàng. Người thu ngân có thể nhập các món ăn được gọi, số lượng món ăn và giá của từng món ăn. Khi người dùng nhấn vào `Button` **Add Item**, ứng dụng của bạn sẽ hiển thị số lượng, tên món ăn được gọi và giá của từng món ăn lên ba `ListBox`, như Hình 10.22. Khi người dùng nhấn vào `Button` **Total Bill**, ứng dụng tính tổng chi phí. Với mỗi món ăn trong `ListBox`, nhân chi phí của mỗi món ăn với số lượng được gọi.



Hình 10.22 Giao diện ứng dụng **Restaurant Bill**.

11

CHƯƠNG

Mục tiêu

Trong chương này, bạn sẽ tìm hiểu để:

- Thực thi lặp lại lệnh nhiều lần với lệnh lặp **For...Next**.
- Sử dụng điều khiển **NumericUpDown** để lấy dữ liệu do người dùng nhập vào.
- Sử dụng **TextBox** nhiều dòng để hiển thị thông tin.
- Sử dụng kiểu **String**.

Nội dung chính

Ứng dụng Interest Calculator

Giới thiệu lệnh lặp For...Next và điều khiển NumericUpDown

Như bạn đã học ở Chương 9 và 10, các ứng dụng thường cần lặp lại hành động. Sử dụng lệnh lặp Do cho phép bạn chỉ ra một điều kiện và kiểm tra điều kiện đó trước khi bắt đầu vòng lặp hoặc sau khi thực thi phần thân của vòng lặp. Trong ứng dụng **Car Payment Calculator** và ứng dụng **Class Average**, một biến đếm đã được sử dụng để xác định số lần lặp lại vòng lặp. Trên thực tế, việc sử dụng biến đếm trong các lệnh lặp phổ biến đến mức Visual Basic cung cấp một lệnh điều khiển được thiết kế đặc biệt cho mục đích này - lệnh lặp `For...Next`. Trong chương này, bạn sử dụng lệnh lặp `For...Next` để tạo một ứng dụng tính tiền lãi có tên là **Interest Calculator.**

11.1 Chạy thử ứng dụng Interest Calculator

Ứng dụng **Interest Calculator** tính số tiền trong tài khoản tiết kiệm của bạn. Bạn bắt đầu với một số tiền gửi nhất định và được trả lãi sau mỗi khoảng thời gian. Người dùng chỉ định số tiền gửi (số tiền vốn ban đầu trong tài khoản), tỷ lệ lãi suất và số năm theo đó tiền lãi sẽ được tính. Sau đó ứng dụng sẽ hiển thị kết quả tính toán. Ứng dụng này cần đáp ứng những yêu cầu sau:

Yêu cầu đối với ứng dụng

Bạn đang cân nhắc việc đầu tư 1000 USD vào một tài khoản tiết kiệm có lãi suất 5% mỗi năm và bạn muốn dự đoán sự phát triển của khoản đầu tư đó. Giả sử bạn để lại toàn bộ lãi trong tài khoản, hãy tính toán và hiển thị tổng số tiền trong tài khoản vào cuối mỗi năm sau một khoảng thời gian n *năm. Để tính toán, bạn hãy sử dụng công thức sau đây:*

$a = p(1 + r)^n$

Trong đó:

p là số tiền ban đầu gửi vào tài khoản (tiền vốn ban đầu).

r là tỷ lệ lãi suất hàng năm (chẳng hạn, 0,05 tương đương với 5%).

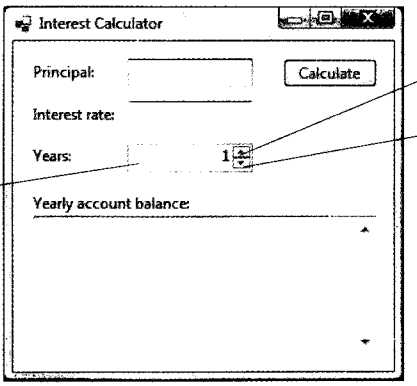
n là số năm.

a là tổng số tiền trong tài khoản vào cuối năm thứ n.

Bạn bắt đầu bằng việc chạy thử ứng dụng đã hoàn thiện. Sau đó, bạn sẽ tìm hiểu những tính năng bổ sung cần thiết của Visual Basic để xây dựng cho mình một phiên bản của ứng dụng này.

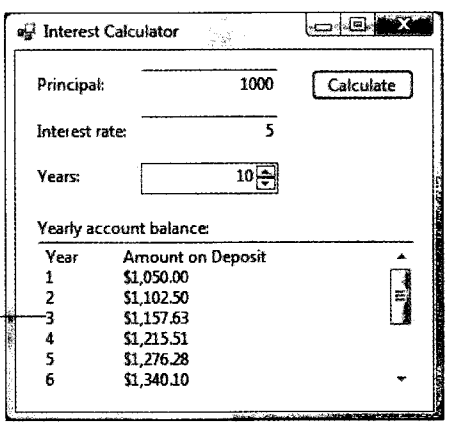
Chạy thử ứng dụng Interest Calculator

1. ***Mở ứng dụng đã hoàn thiện.*** **Mở thư mục** `C:\Examples\Tutorial11\CompletedApplication\InterestCalculator` để tìm ứng dụng **Interest Calculator**. Nhấn đúp vào `InterestCalculator.sln` để mở ứng dụng trong Visual Basic IDE.
2. ***Chạy ứng dụng Interest Calculator.*** **Chọn Debug > Start Debugging** để chạy ứng dụng.



Hình 11.1 Ứng dụng **Interest Calculator** hoàn thiện.

3. ***Nhập vốn ban đầu.*** Khi ứng dụng đã chạy, hãy nhập vốn ban đầu vào `TextBox` **Principal:**. Nhập 1000 như trong phần yêu cầu đối với ứng dụng đã trình bày.
4. ***Cung cấp tỷ lệ lãi suất.*** Tiếp theo, nhập giá trị vào `TextBox` **Interest Rate:**. Hãy nhập 5 vào `TextBox` **Interest Rate**.
5. ***Cung cấp khoảng thời gian đầu tư.*** Chọn số năm mà bạn muốn tính số tiền có trong tài khoản tiết kiệm. Trong trường hợp này, hãy chọn 10 bằng cách nhập từ bàn phím hoặc nhấn liên tục vào mũi tên đi lên của điều khiển `NumericUpDown` có nhãn **Years:** cho đến khi giá trị được hiển thị là 10.
6. ***Tính toán giá trị tài khoản.*** Sau khi bạn đã nhập thông tin cần thiết, nhấn `Button` **Calculate.** Tổng số tiền trong tài khoản vào cuối mỗi năm trong vòng 10 năm sẽ được hiển thị trong một `TextBox` nhiều dòng. Ứng dụng trông sẽ giống như Hình 11.2.



Hình 11.2 Kết quả hiển thị của ứng dụng **Interest Calculator**.

7. ***Đóng ứng dụng.*** Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.
8. ***Đóng IDE.*** Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

11.2 Những điều cơ bản về lệnh lặp được điều khiển bởi biến đếm

Trong Chương 9 và Chương 10, bạn đã học cách sử dụng lệnh lặp được điều khiển bởi biến đếm. Bốn yếu tố cơ bản của lệnh lặp được điều khiển bởi biến đếm là:

1. *tên* của *biến điều khiển* (hay biến đếm vòng lặp) dùng để xác định vòng lặp có tiếp tục được lặp lại hay không
2. *giá trị khởi tạo* của biến điều khiển
3. *bước tăng* (hoặc *bước giảm*) mà theo đó, biến điều khiển sẽ thay đổi qua mỗi lần lặp (tức là mỗi lần mà vòng được thực hiện).
4. *điều kiện* kiểm tra đối với *giá trị cuối cùng* của biến điều khiển (để xác định vòng lặp có tiếp tục được lặp lại hay không).

Ví dụ trong Hình 11.3 sử dụng bốn yếu tố của lệnh lặp được điều khiển bởi biến đếm. Lệnh `Do While...Loop` này giống như như lệnh lặp trong ứng dụng **Car Payment Calculator** ở Chương 9.

```
1  Dim years As Integer = 2 ' biến điều khiển
2
3  Do While years <= 5
4      months = 12 * years ' tính thời hạn vay theo tháng
5
6      ' tính số tiền phải trả hàng tháng
7      monthlyPayment = _
8          (monthlyInterest, months, -loanAmount)
9
10     ' hiển thị số tiền phải trả
11     paymentListBox.Items.Add(months & ControlChars.Tab & _
12         ControlChars.Tab & String.Format("{0:C}", monthlyPayment))
13
14     years += 1 ' tăng biến đếm
15 Loop
```

Hình 11.3 Ví dụ về lệnh lặp được điều khiển bởi biến đếm.

Hãy nhớ lại rằng ứng dụng **Car Payment Calculator** tính toán và hiển thị số tiền trả hàng tháng cho xe hơi trong các khoảng thời gian từ hai đến năm năm. Lệnh khai báo ở dòng 1 *đặt tên* cho biến điều khiển (`years`) và chỉ ra đó là kiểu dữ liệu `Integer`. Lệnh khai báo này cũng bao gồm phép khởi tạo, thiết lập *giá trị khởi tạo* cho biến là 2.

Hãy xem xét lệnh lặp `Do While...Loop` (dòng 3-15). Dòng 4 sử dụng biến `years` để tính toán số tháng mà theo đó số tiền trả cho xe được tính toán. Các dòng 7-8 sử dụng hàm `Pmt` để xác định tiền trả hàng tháng cho xe. Giá trị này phụ thuộc vào tỷ lệ lãi suất, thời hạn theo tháng của khoản vay, giá của chiếc xe và số tiền đã thanh toán trước. Dòng 11-12 hiển thị số tiền ở trong `ListBox`. Dòng 14 tăng biến điều khiển `years` thêm 1 đơn vị trong mỗi vòng lặp. Điều kiện trong lệnh `Do While...Loop` (dòng 3) kiểm tra giá trị của biến điều khiển có nhỏ hơn hoặc bằng 5 hay không, có nghĩa 5 là *giá trị cuối cùng* để điều kiện này vẫn còn đúng. Phần thân lệnh `Do While...Loop` thực hiện ngay cả khi biến điều khiển này có giá trị là 5. Vòng lặp kết thúc khi biến điều khiển lớn hơn 5 (nghĩa là, khi `years` có giá trị là 6).

TỰ ÔN TẬP

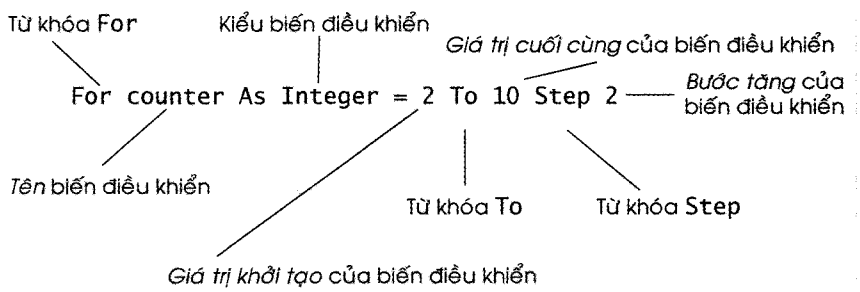
1. Lệnh lặp được điều khiển bởi biến đếm_____ biến điều khiển sau mỗi lần lặp.
   a) tăng b) khởi tạo
   c) giảm d) a hoặc c đúng
2. Thuộc tính nào của biến điều khiển xác định quá trình lặp có tiếp tục hay không?
   a) tên b) giá trị khởi tạo
   c) kiểu d) giá trị cuối cùng

Đáp án: **1)** d. **2)** d.

11.3 Giới thiệu lệnh lặp For...Next

Lệnh lặp **For...Next** cho phép bạn viết mã để thực hiện lệnh lặp được điều khiển bởi biến đếm dễ dàng hơn. Lệnh này chỉ ra cả bốn yếu tố cơ bản của phép lặp được điều khiển bởi biến đếm. Lệnh **For...Next** cần ít thời gian để viết mã hơn và dễ đọc hơn so với lệnh lặp Do tương đương.

Chúng ta hãy xem xét dòng đầu của lệnh lặp `For...Next` (Hình 11.4) mà ta gọi là **dòng tiêu đề For...Next**. Dòng tiêu đề này định rõ cả bốn yếu tố cơ bản của lệnh lặp được điều khiển bởi biến đếm. Dòng này có nghĩa là *"cho các giá trị của biến đếm `counter` bắt đầu từ 2 và kết thúc ở 10, hãy thực hiện các lệnh sau, sau đó cộng 2 vào biến `counter`".*



Hình 11.4 Các thành phần của dòng tiêu đề `For...Next`.

Một lệnh `For...Next` như sau

```
For counter As Integer = 2 To 10 Step 2
        Thân lệnh
Next
```

bắt đầu với từ khóa **For**. Sau đó lệnh khai báo và khởi tạo biến điều khiển (trong trường hợp này thì biến điều khiển `counter` kiểu `Integer` được khai báo và gán giá trị là 2). Chú ý là bạn không dùng từ khóa `Dim` để khai báo biến điều khiển trong dòng tiêu đề `For...Next`. Theo sau giá trị khởi tạo là từ khóa **To**, sau từ khóa này là giá trị cuối của biến điều khiển. Sau đó bạn có thể sử dụng từ khóa **Step** để chỉ ra giá trị bước tăng (hoặc giảm) của biến điều khiển sau mỗi lần thân vòng lặp được thực thi. Nếu muốn giảm giá trị của biến điều khiển sau mỗi vòng lặp, bạn chỉ cần dùng số âm phía sau từ khóa `Step`. Khi sử dụng giá trị `Step` âm, giá trị cuối phải nhỏ hơn giá trị khởi tạo, nếu không thì thân vòng lặp sẽ không được thực thi. Từ khóa `Step` là không bắt buộc. Nếu bạn bỏ qua từ khóa `Step`, biến điều khiển tự động tăng 1 đơn vị sau mỗi vòng lặp theo mặc định.

Lỗi lập trình thường gặp

Truy cập biến điều khiển của vòng lặp từ một đoạn mã nằm sau vòng lặp gây ra lỗi biên dịch vì biến không còn tồn tại.

Phần thân của lệnh `For...Next` được đặt sau dòng tiêu đề. Từ khóa **Next** đánh dấu điểm kết thúc của lệnh lặp `For...Next`, giống với từ khóa `Loop` đánh dấu điểm kết thúc của lệnh `Do While...Loop`. Bạn có thể viết tên biến điều khiển ở bên phải từ khóa `Next` (ví dụ như `Next counter`). Một số lập trình viên thường làm cách này để chương trình rõ ràng hơn, đặc biệt là trong các vòng lặp `For...Next` lồng

với nhau. Khi bạn khai báo biến điều khiển ở dòng tiêu đề của lệnh For...Next (như ta đã làm ở trên), biến điều khiển chỉ tồn tại cho đến khi vòng lặp kết thúc. Phần sau đây mô tả mỗi bước thực thi của lệnh lặp trên.

Thực thi lệnh lặp For...Next

1. Ứng dụng khai báo biến counter và gán giá trị cho biến là 2.
2. Điều kiện tiếp tục vòng lặp được kiểm tra. Điều kiện này có giá trị là True (vì biến counter có giá trị là 2, nhỏ hơn hoặc bằng 10) nên ứng dụng sẽ thực thi phần thân của lệnh lặp For...Next.
3. Giá trị của biến counter được tăng thêm 2 đơn vị; bây giờ biến counter có giá trị là 4.
4. Điều kiện tiếp tục vòng lặp được kiểm tra. Điều kiện này có giá trị là True (vì biến counter có giá trị là 4, nhỏ hơn hoặc bằng 10) nên ứng dụng thực thi phần thân của lệnh lặp For...Next.
5. Giá trị của biến counter được tăng thêm 2 đơn vị; bây giờ biến counter có giá trị là 6.
6. Điều kiện tiếp tục vòng lặp được kiểm tra. Điều kiện này có giá trị là True (vì biến counter có giá trị là 6, nhỏ hơn hoặc bằng 10) nên ứng dụng thực thi phần thân của lệnh lặp For...Next.
7. Giá trị của biến được tăng counter thêm 2 đơn vị; bây giờ biến counter có giá trị là 8.
8. Điều kiện tiếp tục vòng lặp được kiểm tra. Điều kiện này có giá trị là True (vì biến counter có giá trị là 8, nhỏ hơn hoặc bằng 10) nên ứng dụng thực thi phần thân của lệnh lặp For...Next.
9. Giá trị của biến counter được tăng thêm 2 đơn vị; biến counter có giá trị là 10.
10. Điều kiện tiếp tục vòng lặp được kiểm tra. Điều kiện này được đánh giá là True (vì biến counter có giá trị là 10, nhỏ hơn và bằng 10) nên ứng dụng thực thi phần thân của lệnh lặp For...Next.
11. Giá trị của biến counter được tăng thêm 2 đơn vị; biến counter có giá trị là 12.
12. Điều kiện tiếp tục vòng lặp được kiểm tra. Điều kiện này được đánh giá là False (vì biến counter có giá trị là 12, không nhỏ hơn hoặc bằng 10) nên ứng dụng thoát khỏi lệnh lặp For...Next.

Lệnh For...Next có thể được thay thế bởi lệnh lặp khác. Ví dụ, lệnh Do While...Loop tương đương cho ví dụ ở Hình 11.4 là

```
counter = 2

Do While counter <= 10
    thân lệnh
    counter += 2
Loop
```

Lưu ý rằng dòng tiêu đề lệnh For...Next (Hình 11.4) bao hàm điều kiện tiếp tục vòng lặp (counter <= 10). Điều kiện này được chỉ ra rõ ràng trong lệnh Do While...Loop trên đây. Giá trị ban đầu, giá trị kết thúc và bước tăng của dòng tiêu đề For...Next có thể chứa biểu thức số học. Biểu thức được đánh giá một lần (khi lệnh For...Next bắt đầu được thực thi) rồi được dùng như giá trị khởi đầu, giá trị kết thúc và bước tăng của dòng tiêu đề lệnh For...Next. Ví dụ, giả sử a=2 và b=10. Dòng tiêu đề:

```
For i As Integer = a To (4 * a * b) Step (b \ a)
```

tương đương với dòng tiêu đề

```
For i As Integer = 2 To 80 Step 5
```

Nếu như điều kiện tiếp tục vòng lặp ban đầu là `False` (ví dụ, nếu giá trị ban đầu lớn hơn so với giá trị kết thúc và bước tăng là dương) thì phần thân lệnh `For...Next` sẽ không được thực thi. Thay vào đó, chương trình tiếp tục thực thi lệnh tiếp theo sau lệnh `For...Next`.

Mẹo tránh lỗi

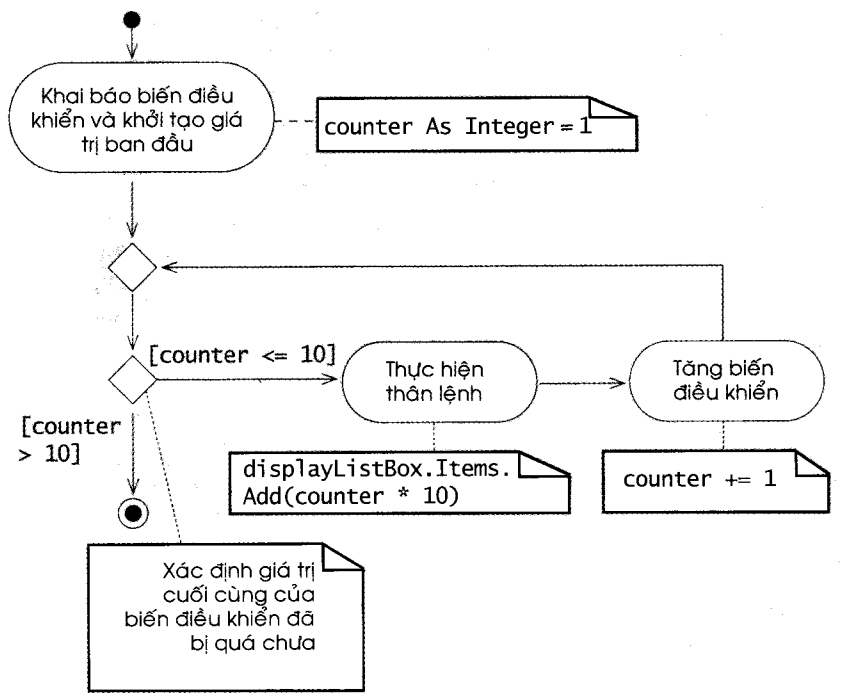
Mặc dù giá trị của biến điều khiển có thể được thay đổi trong thân vòng lặp `For...Next` nhưng ta nên tránh làm việc đó bởi cách làm này có thể dẫn đến những lỗi khó phát hiện.

Biến điều khiển thường được hiển thị hoặc sử dụng để tính toán trong thân lệnh `For...Next`, nhưng điều đó là không bắt buộc. Mục đích chính của biến này là điều khiển quá trình lặp chứ không dùng trong thân lệnh.

Biểu đồ hoạt động của lệnh `For...Next` tương tự như biểu đồ hoạt động của lệnh `Do While...Loop`. Ví dụ, biểu đồ hoạt động của lệnh `For...Next`

```
For counter As Integer = 1 To 10
    displayListBox.Items.Add(counter * 10)
Next
```

được minh họa trong Hình 11.5. Biểu đồ hoạt động này chỉ ra rằng phép khởi tạo chỉ xảy ra một lần và biến chỉ được tăng giá trị *sau* mỗi lần phần thân lệnh được thực thi. Hãy lưu ý rằng, ngoài các vòng tròn nhỏ và các mũi tên chuyển tiếp, biểu đồ hoạt động chỉ bao gồm các ký hiệu trạng thái hành động, ký hiệu ghép và các ký hiệu ra quyết định.



Hình 11.5 Biểu đồ hoạt động UML cho lệnh lặp `For...Next`.

Thói quen lập trình tốt

Cách dòng trên và dưới cũng như lùi mã phần thân lệnh điều khiển vào một chút để lệnh dễ đọc hơn.

Dòng tiêu đề của lệnh `For...Next` chỉ ra các yếu tố cần thiết của lệnh lặp được điều khiển bởi biến đếm. Để giúp bạn hiểu hơn về lệnh lặp mới này, hãy xem cách thay thế lệnh `Do While...Loop` trong Hình 11.3 bằng lệnh `For...Next`.

Mã đã được chuyển đổi thể hiện trên Hình 11.6. Khi lệnh `For...Next` bắt đầu thực thi, dòng 1 của Hình 11.6 khai báo biến điều khiển `years` kiểu `Integer` và khởi tạo giá trị là 2.

Mẹo tránh lỗi

Nếu bạn sử dụng `For...Next` cho lệnh lặp được điều khiển bởi biến đếm thì có thể tránh được các lỗi kết thúc vòng lặp sớm hoặc muộn (thường xảy ra khi vòng lặp thực thi nhiều hoặc ít hơn số lần lặp cần thiết) bởi giá trị kết thúc là rất rõ ràng.

Điều kiện tiếp tục vòng lặp được ngầm định là `years <= 5` (phụ thuộc vào giá trị cuối cùng của biến điều khiển) được kiểm tra ở dòng 1. Từ khóa `To` là bắt buộc trong lệnh `For...Next`. Giá trị đứng trước từ khóa này xác định giá trị ban

đầu của years; giá trị đứng sau nó chỉ ra giá trị được kiểm tra cho việc tiếp tục vòng lặp (trong trường hợp này là 5). Từ khóa Step là không bắt buộc và không được sử dụng ở đây. Step chỉ ra bước tăng (giá trị được tăng sẽ thêm vào years mỗi lần thân lệnh For...Next được thực thi). Nếu Step không có mặt thì bước tăng mặc định là 1.

Lỗi lập trình thường gặp

Không nên sử dụng biến đếm dấu chấm động trong lệnh lặp được điều khiển bởi biến đếm. Các biến kiểu này được biểu diễn một cách xấp xỉ trong bộ nhớ máy tính nên làm cho giá trị biến đếm không chính xác và sẽ làm sai kết quả phép kiểm tra, dẫn đến các lỗi lôgic.

```
1  For years As Interger = 2 To 5
2     months = 12 * years 'tính thời hạn vay theo tháng
3
4     'tính số tiền phải trả hàng tháng
5     value = Pmt(monathlyRate, months, -loanAmount)
6
7     'hiển thị số tiền phải trả
8     paymentListBox.Items.Add(months & ControlChars.Tab & _
9        ControlChars.Tab & String.Format("{0:C}", value))
10 Next
```

Hình 11.6 Đoạn mã của ứng dụng **Car Payment Calculator** sử dụng lệnh For...Next.

Giá trị khởi tạo của years là 2, vì thế điều kiện ngầm định để tiếp tục vòng lặp được thỏa mãn và các phép tính trong thân lệnh For...Next được thực thi.

Điều khiển chương trình đến dòng Next sau khi thực thi phần thân lệnh (dòng 10). Từ khóa này đánh dấu điểm kết thúc mỗi lần lặp trong lệnh For...Next. Khi gặp Next, giá trị của years được tăng thêm 1 (bước tăng mặc định) và chương trình lại kiểm tra điều kiện tiếp tục vòng lặp.

Chương trình tiếp tục lặp cho đến khi điều kiện tiếp tục vòng lặp là False (có nghĩa là Years >5), đến đây vòng lặp sẽ kết thúc.

Tự động xác định kiểu cục bộ

Trong mỗi lệnh For...Next đã trình bày ở trên, ta đã khai báo và khởi tạo biến điều khiển trong dòng tiêu đề For...Next. Trình biên dịch Visual Basic 2008 cung cấp một tính năng mới - được gọi là **tự động xác định kiểu cục bộ (local type inference)** - cho phép xác định kiểu của biến cục bộ dựa trên ngữ cảnh mà biến được khởi tạo. Biến cục bộ là bất cứ biến nào được khai báo trong thân của phương thức (ví dụ như trong phần thân xử lý sự kiện). Ví dụ, trong phép khai báo

```
Dim x = 7
```

trình biên dịch xác định biến x có kiểu Integer vì trình biên dịch cho rằng các giá trị số nguyên, ví dụ như 7, thuộc kiểu Integer. Tương tự, trong phép khai báo

```
Dim y = -123.45
```

trình biên dịch xác định biến y thuộc kiểu Double vì trình biên dịch cho rằng các giá trị số có dấu chấm động, như 123.45, thuộc kiểu Double.

Bạn cũng có thể sử dụng tính năng tự động xác định kiểu cục bộ này cho biến điều khiển trong tiêu đề của lệnh For...Next. Ví dụ như trên Hình 11.6, dòng 1 có thể viết lại như sau

```
For years = 2 To 5
```

Trong trường hợp này, biến years kiểu Integer bởi nó được khởi tạo giá trị số nguyên (2). Chúng tôi thường khai báo kiểu tường minh trong các ví dụ của cuốn sách này.

Tính năng tự động xác định kiểu cục bộ là một trong các tính năng mới của Visual Basic 2008, hỗ trợ cho Language Integrated Query (LINQ). Chúng tôi sẽ sử dụng tính năng tự động xác định kiểu cục bộ này khi trình bày các ví dụ LINQ trong Chương 20, 22.

TỰ ÔN TẬP

1. Nếu không có mệnh đề `Step` thì bước tăng mặc định của lệnh `For...Next` là_____.
   a) 2 b) 1
   c) 0 d) -1
2. Giá trị trước từ khóa `To` trong lệnh `For...Next` chỉ ra_____.
   a) giá trị khởi tạo của biến đếm b) giá trị cuối cùng của biến đếm
   c) bước tăng d) số lần vòng lặp lặp lại

Đáp án: 1) b. 2) a.

11.4 Các ví dụ sử dụng lệnh For...Next

Các ví dụ sau đây biểu diễn cách thay đổi giá trị biến điều khiển trong lệnh `For... Next`. Trong mỗi trường hợp, ta có dòng tiêu đề tương ứng:

a) Thay đổi biến điều khiển từ 1 đến 100 với bước tăng 1.

```
For i As Integer = 1 To 100
```

hoặc

```
For i As Integer = 1 To 100 Step 1
```

b) Thay đổi biến điều khiển từ 100 đến 1 với bước tăng -1 (giảm 1 đơn vị).

```
For i As Integer = 100 To 1 Step -1
```

c) Thay đổi biến điều khiển từ 7 đến 77 với bước tăng 7.

```
For i As Integer = 7 To 77 Step 7
```

d) Thay đổi biến điều khiển từ 20 đến -20 với bước tăng -2 (giảm 2 đơn vị).

```
For i As Integer = 20 To -20 Step -2
```

e) Thay đổi biến điều khiển theo chuỗi giá trị sau: `2, 5, 8, 11, 14, 17, 20.`

```
For i As Integer = 2 To 20 Step 3
```

f) Thay đổi biến điều khiển theo chuỗi giá trị sau: `99, 88, 77, 66, 55, 44, 33, 22, 11, 0.`

```
For i As Integer = 99 To 0 Step -11
```

TỰ ÔN TẬP

1. Đáp án nào dưới đây có dòng tiêu đề `For...Next` phù hợp để thay đổi biến điều khiển theo chuỗi giá trị `25, 20, 15, 10, 5`?
   a) `For i As Integer = 5 To 25 Step 5`
   b) `For i As Integer = 25 To 5 Step -5`
   c) `For i As Integer = 5 To 25 Step -5`
   d) `For i As Integer = 25 To 5 Step 5`
2. Câu nào trong số dưới đây mô tả đúng dòng tiêu đề `For...Next`

   ```
   For i As Integer = 81 To 102
   ```

   a) Thay đổi biến điều khiển từ 81 tới 102 với bước tăng 1.
   b) Thay đổi biến điều khiển từ 81 tới 102 với bước tăng 0.
   c) Thay đổi biến điều khiển từ 102 tới 81 với bước tăng -1.
   d) Thay đổi biến điều khiển từ 81 tới 102 với bước tăng 2.

Đáp án: 1) b. 2) a.

11.5 Xây dựng ứng dụng Interest Calculator

Giải pháp của chúng ta cho vấn đề này là tính tổng số tiền nhận được cuối mỗi năm trong vòng số năm cho trước bằng cách sử dụng lệnh `For...Next`.

Đoạn mã giả sau đây mô tả các thao tác cơ bản của ứng dụng **Interest Calculator**

khi Button **Calculate** được nhấn:

```
Khi người dùng nhấn Button Calculate
    Lấy về các giá trị vốn ban đầu, tỷ lệ lãi suất và số năm do người dùng nhập vào
    Lưu một chuỗi tiêu đề để sau này hiển thị trên nội dung TextBox

    Đối với mỗi năm (bắt đầu từ 1 và kết thúc ở số năm được nhập vào)
        Tính giá trị hiện thời của khoản đầu tư
        Ghép chuỗi có nội dung là số năm và giá trị hiện thời của khoản
            đầu tư vào chuỗi sẽ được hiển thị trong TextBox
    Hiển thị kết quả cuối cùng trên TextBox
```

Ứng dụng template mà chúng tôi cung cấp cho chương này gồm có Button **Calculate** cùng với hai Label và các TextBox tương ứng là: **Principal:** và **Interest Rate:**. Bạn sẽ sử dụng điều khiển **NumericUpDown** cho đầu vào được mô tả bởi Label **Years:**. Điều khiển NumericUpDown giới hạn lựa chọn của người dùng trong một phạm vi nhất định. Sau đó, bạn thêm TextBox nhiều dòng có thanh cuộn vào giao diện. Cuối cùng, bạn bổ sung khả năng xử lý cho lệnh For...Next. Lúc này, khi bạn đã chạy thử ứng dụng **Interest Calculator** và tìm hiểu về mã giả của ứng dụng, bạn dùng bảng ACE để chuyển đổi mã giả thành Visual Basic. Hình 11.7 liệt kê các hành động, điều khiển và sự kiện giúp bạn tự hoàn thiện cho mình một phiên bản của ứng dụng này.

Bảng ACE cho ứng dụng Interest Calculator

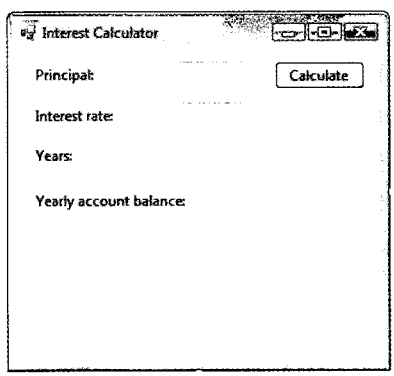
Hành động	Điều khiển	Sự kiện
Hiển thị Label cho các điều khiển của ứng dụng	principalLabel, rateLabel, yearsLabel, yearlyAccount-Label	Ứng dụng được chạy
	calculateButton	Click
Lấy về các giá trị vốn ban đầu, tỷ lệ lãi suất và số năm do người dùng nhập vào	principalText-Box, rateTextBox, yearUpDown	
Lưu một chuỗi tiêu đề để sau này hiển thị trên nội dung TextBox		
Đối với mỗi năm (bắt đầu từ 1 và kết thúc ở số năm được nhập vào) Tính giá trị hiện thời của khoản đầu tư		
Ghép chuỗi có nội dung là số năm và giá trị hiện thời của khoản đầu tư vào chuỗi sẽ được hiển thị trong TextBox		
Hiển thị kết quả cuối cùng trên TextBox	resultTextBox	

Hình 11.7 Bảng ACE cho ứng dụng **Interest Calculator.**

Trong phần sau đây, bạn sẽ bắt đầu xây dựng ứng dụng **Interest Calculator**. Đầu tiên, bạn thêm điều khiển `NumericUpDown` để người dùng có thể chỉ ra số năm. Điều khiển này có mũi tên lên và xuống, cho phép người dùng duyệt qua khoảng giá trị của điều khiển. Phần này cho bạn biết cách thiết lập giới hạn của khoảng giá trị này (giá trị lớn nhất và nhỏ nhất). Ta thiết lập 10 là giá trị lớn nhất và 1 là giá trị nhỏ nhất của điều khiển này. Thuộc tính **`Increment`** chỉ định giá trị hiện thời của điều khiển `NumericUpDown` sẽ thay đổi bao nhiêu khi người dùng nhấn mũi tên đi lên (để tăng) hoặc xuống (để giảm). Ứng dụng này thiết lập giá trị mặc định của thuộc tính `Increment` là 1.

Thêm và thay đổi điều khiển NumericUpDown

1. ***Copy template vào thư mục làm việc của bạn.*** Copy thư mục `C:\Examples\Tutorial11\TemplateApplication\InterestCalculator` vào thư mục `C:\SimplyVB2008`.
2. ***Mở file template của ứng dụng Interest Calculator.*** Nhấn đúp vào file `InterestCalculator.sln` ở thư mục `InterestCalculator` để mở ứng dụng trong Visual Basic IDE (Hình 11.8). Nhấn đúp vào file `InterestCalculator.vb` ở **Solution Explorer** nếu `Form` này vẫn chưa xuất hiện.



Hình 11.8 Ứng dụng **Interest Calculator** trong chế độ **Design**.

3. ***Thêm điều khiển NumericUpDown.*** Nhấn đúp vào `NumericUpDown`

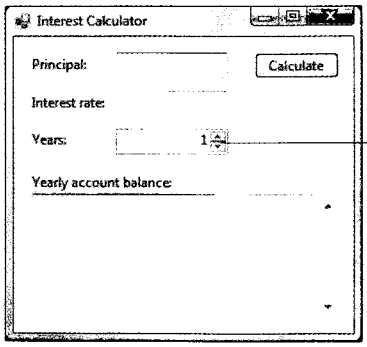
trên **Toolbox** để thêm vào `Form`. Thay đổi thuộc tính `Name` của điều khiển thành `yearUpDown`. Để mã dễ đọc hơn, bạn hãy thêm hậu tố `UpDown` vào sau tên điều khiển `NumericUpDown`.
4. ***Thiết lập vị trí và kích thước của điều khiển NumericUpDown.*** Thiết lập thuộc tính `Location` của `yearUpDown` là 91, 82 và thuộc tính `Width` là 100 để điều khiển này nằm thẳng hàng theo cả chiều ngang và chiều dọc với các `TextBox` bên trên.
5. ***Thiết lập thuộc tính TextAlign.*** Thiết lập giá trị thuộc tính `TextAlign` là `Right` để giá trị số trong điều khiển được căn lề phải.
6. ***Thiết lập giới hạn giá trị cho điều khiển NumericUpDown.*** Theo mặc định, điều khiển này thiếp lập 0 là giá trị nhỏ nhất và 100 là giá trị lớn nhất. Để thay đổi các giá trị này, thiết lập giá trị thuộc tính **`Maximum`** của điều khiển `NumericUpDown` **Years:** là 10, giá trị thuộc tính **`Minimum`** là 1. Thay đổi này sẽ giới hạn người dùng chỉ có thể lựa chọn số nằm trong khoảng từ 1 tới 10. Nếu người dùng nhập giá trị nhỏ hơn giá trị nhỏ nhất hoặc lớn hơn giá trị lớn nhất, giá trị sẽ được tự động sửa tương ứng thành giá trị nhỏ nhất hoặc giá trị lớn nhất khi người dùng chuyển focus. Lưu ý rằng điều khiển `NumericUpDown` sẽ hiển thị 1, giá trị thuộc tính `Minimum` của nó. `Form` của bạn trông sẽ giống như trong Hình 11.9.

Thói quen lập trình tốt

Thêm hậu tố UpDown vào sau tên điều khiển `NumericUpDown`.

Mẹo thiết kế giao diện

Sử dụng điều khiển `NumericUpDown` để giới hạn khoảng giá trị nhập vào của người dùng.



Hình 11.9 Điều khiển `NumericUpDown` được thêm vào ứng dụng **Interest Calculator.**

Mẹo thiết kế giao diện

Điều khiển `NumericUpDown` nên tuân thủ nguyên tắc thiết kế giao diện như `TextBox`. (Xem Phụ lục A)

7. ***Lưu project.*** Chọn **File > Save All** để lưu mã đã được sửa đổi.

Ứng dụng **Interest Calculator** hiển thị kết quả các phép tính toán trên điều khiển `TextBox` nhiều dòng - điều khiển có thể hiển thị văn bản nhiều hơn một dòng. Bạn có thể cấu hình `TextBox` để có thanh cuộn sao cho người dùng có thể cuộn lên xuống để thấy toàn bộ nội dung, nếu `TextBox` quá nhỏ để hiển thị. Trong phần tiếp theo, bạn sẽ tạo `TextBox` này.

Thêm và tùy chỉnh cho TextBox nhiều dòng có thanh cuộn

Mẹo thiết kế giao diện

Nếu muốn `TextBox` hiển thị nhiều dòng, hãy thiết lập giá trị cho thuộc tính `Multiline` là `True` và canh lề trái cho nội dung hiển thị trong đó bằng cách thiết lập giá trị cho thuộc tính `TextAlign` là `Left`.

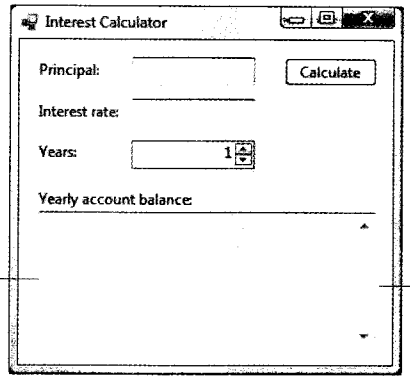
Mẹo thiết kế giao diện

Nếu dùng `TextBox` để hiển thị dữ liệu đầu ra, hãy thiết lập thuộc tính `ReadOnly` là `True` để người dùng không thể thay đổi nội dung của `TextBox`.

Mẹo thiết kế giao diện

Nếu cần hiển thị văn bản đầu ra có nhiều dòng, hãy giới hạn chiều cao của `TextBox` và sử dụng thanh cuộn dọc để người dùng có thể xem được toàn bộ nội dung.

1. ***Thêm TextBox vào Form.*** Nhấn đúp vào điều khiển `TextBox` ở **Toolbox** để thêm `TextBox` vào `Form`. Đặt tên cho `TextBox` này là `resultTextBox`.
2. ***Tạo TextBox nhiều dòng.*** Chọn thuộc tính **`Multiline`** của `TextBox` và thay đổi giá trị từ `False` sang `True` để `TextBox` có thể chứa nhiều dòng.
3. ***Thiết lập kích thước và vị trí của TextBox.*** Đặt thuộc tính `Location` của `TextBox` là `16, 140` và thuộc tính `Size` là `274, 111` để `Textbox` gióng với các điều khiển nằm phía trên.
4. ***Thiết lập thuộc tính ReadOnly.*** Để bảo đảm rằng người dùng không thể thay đổi nội dung trong `TextBox` **Yearly account balance:**, hãy đặt thuộc tính **`ReadOnly`** là `True`.
5. ***Thêm thanh cuộn dọc.*** Sử dụng thanh cuộn dọc cho phép bạn giữ kích cỡ của `TextBox` nhỏ trong khi người dùng vẫn thấy toàn bộ thông tin trong `TextBox` đó. Độ dài của văn bản có thể vượt quá chiều cao của `TextBox`, vì vậy hãy kích hoạt thanh cuộn dọc bằng cách thiết lập giá trị thuộc tính **`ScrollBars`** của `resultTextBox` là **`Vertical`**. Một thanh cuộn dọc sẽ xuất hiện ở bên phải của `TextBox`. Theo mặc định, thuộc tính `ScrollBars` có giá trị **None**. Bạn cũng có thể thiết lập giá trị thuộc tính `ScrollBars` là **`Horizontal`** hoặc **`Both`**. Thanh cuộn ngang sẽ xuất hiện ở bên dưới `TextBox`. Giá trị `Both` chỉ ra rằng thanh cuộn ngang và dọc đều sẽ được hiển thị - thanh cuộn ngang sẽ chỉ được hiển thị nếu văn bản vượt quá độ rộng của `TextBox`. Hãy lưu ý rằng, thậm chí nếu không có thanh cuộn, người dùng vẫn có thể di chuyển qua toàn bộ nội dung văn bản bằng cách sử dụng các phím mũi tên. Ban đầu thì thanh cuộn không được hiển thị trên `Form`. Thanh cuộn chỉ được kích hoạt khi cần thiết (nghĩa là, khi văn bản trong `TextBox` quá dài). `Form` của bạn trông sẽ giống như Hình 11.10.



Hình 11.10 TextBox nhiều dòng có thanh cuộn dọc được thêm vào Form.

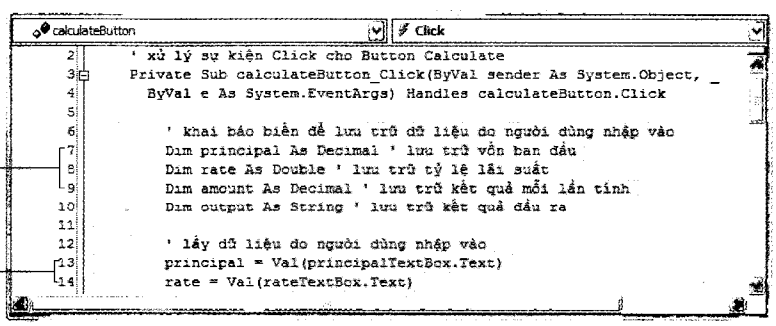
6. ***Lưu project***. Chọn **File > Save All** để lưu mã đã được sửa đổi.

Bây giờ bạn đã hoàn tất việc thiết kế giao diện, bạn sẽ thêm chức năng cho ứng dụng ở phần sau đây. Khi người dùng nhấn Button **Calculate**, bạn muốn ứng dụng lấy thông tin đã được nhập vào, sau đó xuất ra một bảng chứa tổng số tiền sẽ nhận được vào cuối mỗi năm. Bạn làm việc đó bằng cách bổ sung mã vào phần xử lý sự kiện Click của Button.

Thêm xử lý sự kiện Click

1. ***Tạo xử lý sự kiện***. Nhấn đúp vào Button **Calculate**, phần xử lý sự kiện Click của Button **Calculate** sẽ xuất hiện ở mã của ứng dụng.
2. ***Bổ sung mã vào xử lý sự kiện calculateButton_Click***. Thêm các dòng 6-14 trên Hình 11.11 vào phần xử lý sự kiện calculateButton_Click. Dòng 7-10 khai báo các biến cần thiết để lưu thông tin do người dùng nhập vào, kết quả tính toán và thông tin hiển thị. Biến principal lưu số vốn ban đầu do người dùng nhập vào và biến rate lưu tỷ lệ lãi suất. Biến amount lưu kết quả của việc tính toán tiền lãi.

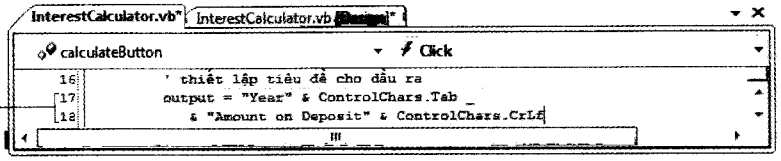
   Dòng 10 khai báo biến đầu ra kiểu String. Biến **String** lưu một dãy ký tự. Ký tự hay được dùng nhất là chữ cái và chữ số, mặc dù còn có nhiều ký tự khác như $, *, ^, tab và ký tự xuống dòng. Thực ra bạn đã dùng String trong suốt các bài tập - giá trị thuộc tính Text của Label và TextBox đều được biểu diễn dưới dạng String. Trên thực tế, khi bạn gán giá trị kiểu số, như Integer, cho thuộc tính Text của Label, giá trị Integer đó sẽ được tự động chuyển đổi thành kiểu String. Dòng 13-14 lấy giá trị vốn ban đầu và tỷ lệ lãi suất từ các TextBox.



Hình 11.11 Đoạn mã lấy và lưu thông tin do người dùng nhập vào.

TextBox nhiều dòng hiển thị các kết quả thành hai cột. Thêm các dòng 16-18 trên Hình 11.12 để gán dòng tiêu đề cho đầu ra. Dòng tiêu đề cho biết tên hai cột tương ứng là Year và Amount on Deposit.

Thêm tiêu đề vào String đầu ra



Hình 11.12 Đoạn mã thêm tiêu đề vào biến String.

Nhớ lại rằng bạn đã xóa nội dung trên Label ở Chương 6 bằng cách đặt thuộc tính Text thành chuỗi rỗng (""), đại diện cho String không chứa ký tự nào. Khi gán giá trị mới cho biến String, bạn phải bắt đầu và kết thúc văn bản bằng dấu ngoặc kép ("). Dấu ngoặc kép đóng và mở cần xuất hiện trên cùng một dòng, nếu không bạn sẽ gặp lỗi cú pháp. Chẳng hạn, nếu bạn muốn chứa từ Year trong biến String year, bạn hãy dùng lệnh sau:

```
year = "Year"
```

Bạn có thể thêm String hay ký tự vào cuối một String khác bằng cách sử dụng toán tử ghép chuỗi (&). Ở các dòng 17-18 trên Hình 11.12, bạn sử dụng hằng số ControlChars.Tab để chèn ký tự tab giữa từ Year và Amount on Deposit. Sau đó bạn thêm ký tự xuống dòng (ControlChars.CrLf) để phần văn bản tiếp theo xuất hiện trên dòng tiếp theo của đầu ra.

3. ***Lưu project.*** Chọn **File > Save All** để lưu mã đã sửa đổi.

Lệnh For...Next ở các dòng 21-26 trên Hình 11.13 thực hiện tính tổng số tiền nhận được theo số năm được nhập vào. Bạn sẽ viết lệnh For...Next trong phần tiếp theo.

Tính toán tổng số tiền người dùng nhận được sử dụng lệnh For...Next

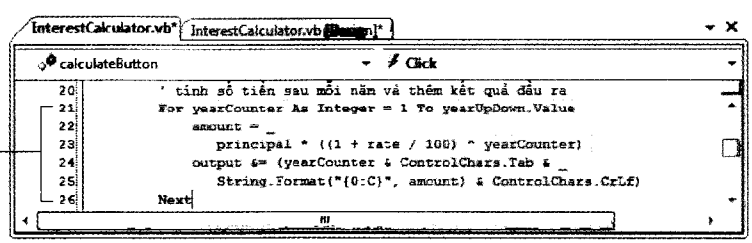
1. ***Khai báo, khởi tạo biến điều khiển và thiết lập phép kiểm tra tiếp tục vòng lặp.*** Thêm các dòng 20-26 của Hình 11.13 vào xử lý sự kiện calculateButton_Click. Lưu ý rằng từ khóa Next tự động xuất hiện khi bạn nhấn *Enter* ở cuối dòng 21.

Thói quen lập trình tốt

Cách một dòng trước và sau mỗi lệnh điều khiển để lệnh đó nổi bật trong mã.

Dòng 21 là tiêu đề lệnh For...Next, khai báo biến điều khiển yearCounter và khởi tạo cho biến này giá trị là 1. Giá trị sau từ khóa To thiết lập điều kiện vòng lặp. Vòng lặp này tiếp tục khi mà biến điều khiển nhỏ hơn hoặc bằng số năm do người dùng nhập vào.

Sử dụng lệnh For...Next để định dạng và thêm văn bản vào String đầu ra.



Hình 11.13 Lệnh For...Next của ứng dụng.

2. ***Tính toán lãi.*** Phần thân lệnh For...Next được thực thi một lần cho mỗi năm cho đến giá trị thuộc tính **Value** của điều khiển yearUpDown, tức số năm được người dùng chọn. Biến điều khiển yearCounter tăng từ 1 đến

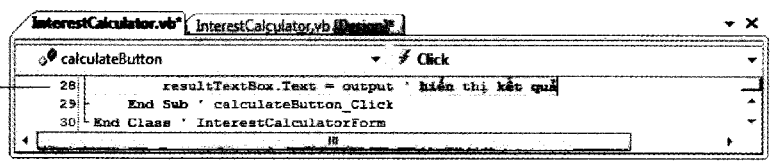
`yearUpDown.Value` với bước tăng 1 đơn vị. Thêm dòng 22-23 trên Hình 11.13 để thực hiện tính toán theo công thức

$$a = p(1 + r)^n$$

trong đó a là tổng số tiền nhận được, p là vốn gốc, r là tỷ lệ lãi suất và n là số năm. Lưu ý rằng phép tính ở dòng 23 chia tỷ lệ lãi suất cho 100 do người dùng nhập giá trị tỷ lệ lãi suất ở dạng phần trăm (chẳng hạn, người dùng nhập số `5.5` có nghĩa là 5,5%)

3. ***Thêm kết quả vào chuỗi đầu ra.*** Thêm dòng 24-25 trên Hình 11.13. Những dòng này thêm đoạn văn bản vào cuối `output` bằng toán tử **&=**. **Toán tử** &= (làm việc giống như toán tử +=) sẽ thêm toán hạng ở bên phải vào văn bản ở toán hạng bên trái. Giá trị mới này sau đó được gán cho biến ở toán hạng bên trái. Văn bản bao gồm giá trị `yearCounter` hiện thời, ký tự tab (`ControlChars.Tab`) để phân tách với cột thứ hai, kết quả của hàm `String.Format("{0:C}", amount)` và cuối cùng là ký tự xuống dòng (`ControlChars.CrLf`) để hiển thị đầu ra tiếp theo ở dòng tiếp theo. Hãy nhớ rằng mã định dạng C (currency) chỉ ra rằng đối số tương ứng của nó (`amount`) cần được hiển thị dưới định dạng tiền tệ.
4. ***Tới từ khóa Next.*** Sau khi phần thân lệnh đã được thực hiện, phần thực thi ứng dụng chạy tiếp đến từ khóa `Next` ở dòng 26. Biến đếm (`yearCounter`) tăng lên 1 đơn vị và vòng lặp bắt đầu lại một lần nữa bằng việc kiểm tra có tiếp tục vòng lặp hay không.
5. ***Kết thúc lệnh For...Next.*** Lệnh `For...Next` được thực thi cho đến khi biến điều khiển vượt quá số năm đã được người dùng chỉ ra.
6. ***Hiển thị kết quả tính toán.*** Sau khi thoát khỏi lệnh `For...Next`, nội dung đầu ra đã sẵn sàng để được hiển thị cho người dùng trên `resultTextBox`. Thêm dòng 28 trên Hình 11.14 để hiển thị dòng tiêu đề và kết quả trên `TextBox` nhiều dòng.

Hiển thị kết quả các phép tính toán được thực hiện bởi lệnh `For...Next` trong `TextBox` nhiều dòng



Hình 11.14 Đoạn mã hiển thị kết quả tính toán.

7. ***Chạy ứng dụng.*** Chọn **Debug > Start Debugging** để chạy ứng dụng. Bây giờ ứng dụng của bạn đã có thể tính toán số tiền gửi cho mỗi năm. Hãy nhập 1000 vào `TextBox` **Principal:**, 5 vào `TextBox` **Interest Rate:** và 10 vào điều khiển `NumericUpDown` **Year:**. Nhấn `Button` **Calculate** và kiểm tra xem kết quả có giống như trong Hình 11.2 hay không.
8. ***Đóng ứng dụng.*** Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.
9. ***Đóng IDE.*** Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

Hình 11.15 là mã nguồn của ứng dụng **Interest Calculator**. Các dòng mã chứa các khái niệm lập trình mới mà bạn vừa học trong chương này đều được đánh dấu.

```
1  Public Class InterestCalculatorForm
2      ' xử lý sự kiện Click cho Button Calculate
3      Private Sub calculateButton_Click(ByVal sender As System.Object, _
4          ByVal e As System.EventArgs) Handles calculateButton.Click
5
6          ' khai báo biến để lưu trữ dữ liệu do người dùng nhập vào
7          Dim principal As Decimal ' lưu trữ vốn ban đầu
8          Dim rate As Double ' lưu trữ tỷ lệ lãi suất
9          Dim amount As Decimal ' lưu trữ kết quả mỗi lần tính
10         Dim output As String ' lưu trữ kết quả đầu ra
11
12         ' lấy dữ liệu do người dùng nhập vào
13         principal = Val(principalTextBox.Text)
14         rate = Val(rateTextBox.Text)
15
16         ' thiết lập tiêu đề cho đầu ra
17         output = "Year" & ControlChars.Tab _
18             & "Amount on Deposit" & ControlChars.CrLf
19
20         ' tính số tiền sau mỗi năm và thêm kết quả đầu ra
21         For yearCounter = 1 To yearUpDown.Value
22             amount = _
23                 principal * ((1 + rate / 100) ^ yearCounter)
24             output &= (yearCounter & ControlChars.Tab & _
25                 String.Format("{0:C}", amount) & ControlChars.CrLf)
26         Next
27
28         resultTextBox.Text = output ' hiển thị kết quả
29     End Sub ' calculateButton_Click
30 End Class ' InterestCalculatorForm
```

Sử dụng thuộc tính `Value` của `yearUpDown` — dòng 7

Khai báo biến kiểu `String` — dòng 10

Thiết lập tiêu đề kiểu `String` cho `TextBox` — dòng 17–18

Lặp từ 1 cho đến giá trị được người dùng chỉ ra trên điều khiển `yearUpDown` — dòng 21

Nối thêm kết quả tính toán vào `String output` — dòng 24–25

Hiển thị kết quả trên `resultTextBox` — dòng 28

Hình 11.15 Ứng dụng **Interest Calculator**

TỰ ÔN TẬP

1. Thuộc tính_____ xác định giá trị hiện thời của điều khiển `NumericUpDown` sẽ thay đổi bao nhiêu khi người dùng nhấn mũi tên lên hoặc xuống.
   a) `Amount` b) `Step`
   c) `Increment` d) `Next`

2. Tiêu đề `For...Next` nào thay đổi biến điều khiển từ 1 đến 50 với bước tăng là 5?
   a) `For i = 1 To 50 Step 50`
   b) `For 1 To 50 Step 5`
   c) `For i = 1 To 50 Step = 5`
   d) `For i = 1 To 50 Step 5`

Đáp án: **1)** c. **2)** d.

11.6 Tổng kết

Trong chương này, bạn đã tìm hiểu các yếu tố cơ bản của lệnh lặp được điều khiển bởi biến đếm bao gồm tên biến điều khiển, giá trị ban đầu của biến điều khiển, bước tăng (hoặc bước giảm) mà theo đó biến điều khiển thay đổi sau mỗi vòng lặp và điều kiện kiểm tra giá trị của biến điều khiển. Sau đó ta đã tìm hiểu lệnh lặp `For...Next` là lệnh kết hợp những yếu tố cơ bản của lệnh lặp được điều khiển bởi biến đếm trong dòng tiêu đề của mình.

Sau khi làm quen với lệnh lặp `For...Next`, bạn đã thay đổi lệnh `Do While... Loop` của ứng dụng **Car Payment Calculator** thành lệnh `For...Next`. Tiếp đó bạn xây dựng ứng dụng **Interest Calculator**, sau khi phân tích mã giả và bảng ACE cho ứng dụng này. Trong giao diện của **Interest Calculator** bạn thêm thành phần thiết kế mới, bao gồm điều khiển `NumericUpDown` rất hữu ích cho việc giới hạn dữ liệu nhập vào là số và `TextBox` nhiều dòng có thanh cuộn.

Trong chương tới, bạn sẽ học cách sử dụng lệnh đa lựa chọn `Select Case`. Bạn đã biết lệnh lựa chọn `If...Then...Else` có thể được sử dụng để lựa chọn giữa các hành động khác nhau dựa trên giá trị của một điều kiện. Lệnh đa lựa chọn `Select Case` có thể tiết kiệm thời gian phát triển ứng dụng và làm cho mã dễ đọc hơn nếu có nhiều điều kiện. Bạn sẽ sử dụng lệnh đa lựa chọn `Select Case` để xây dựng ứng dụng **Security Panel**.

TỔNG KẾT KỸ NĂNG

Sử dụng lệnh lặp `For...Next`

- Khai báo biến điều khiển và chỉ rõ giá trị khởi tạo của biến này trước từ khóa `To`.
- Chỉ ra giá trị dùng để xác định việc tiếp tục vòng lặp sau từ khóa `To`.
- Sử dụng từ khóa không bắt buộc `Step` để chỉ ra bước tăng (hoặc bước giảm).
- Sử dụng từ khóa `Next` để đánh dấu điểm kết thúc vòng lặp.
- Sử dụng lệnh `For...Next` để loại trừ các lỗi kết thúc vòng lặp sớm hoặc muộn.

Tạo `TextBox` nhiều dòng có thanh cuộn dọc

- Thêm một `TextBox` vào `Form`.
- Thiết lập thuộc tính `Multiline` của `TextBox` là `True`.
- Thiết lập thuộc tính `ScrollBar` của `TextBox` là `Vertical`.

Thiết lập giá trị lớn nhất cho điều khiển `NumericUpDown`

- Sử dụng thuộc tính `Maximum` của `NumericUpDown`.

Thiết lập giá trị nhỏ nhất cho điều khiển `NumericUpDown`

- Sử dụng thuộc tính `Minimum` của `NumericUpDown`.

Thay đổi giá trị hiện thời trên điều khiển `NumericUpDown`

- Nhấn chuột vào mũi tên lên xuống của điều khiển `NumericUpDown` hoặc nhập vào giá trị mới.

Chỉ định giá trị hiện thời của điều khiển `NumericUpDown` sẽ thay đổi bao nhiêu khi người dùng nhấn mũi tên

- Sử dụng thuộc tính `Increment` của `NumericUpDown`.

Lấy giá trị của điều khiển `NumericUpDown`

- Sử dụng thuộc tính `Value` của `NumericUpDown`.

THUẬT NGỮ

điều khiển `NumericUpDown` - Cho phép bạn chỉ ra giá trị số đầu vào lớn nhất và nhỏ nhất. Điều khiển này cũng cho phép bạn chỉ ra bước tăng (hoặc giảm) khi người dùng nhấn mũi tên lên (hoặc xuống).

giá trị `Horizontal` của thuộc tính `ScrollBars` - Được sử dụng để hiển thị một thanh cuộn ngang ở đáy `TextBox`.

giá trị `None` của thuộc tính `ScrollBars` - Được thiết lập khi không muốn hiển thị thanh cuộn trên `TextBox`.

giá trị `Vertical` của thuộc tính `ScrollBars` - Được thiết lập khi muốn hiển thị thanh cuộn dọc ở bên phải `TextBox`.

kiểu dữ liệu `String` - Lưu trữ chuỗi ký tự.

lệnh lặp `For...Next` - Lệnh lặp này thực hiện tương tự như lệnh lặp được điều khiển bởi biến đếm. Lệnh lặp `For...Next` sử dụng cả bốn yếu tố cơ bản của lệnh lặp được điều khiển bởi biến đếm trên cùng một dòng mã (tên biến điều khiển, giá trị khởi tạo, giá trị của bước tăng hoặc bước giảm và giá trị cuối cùng).

thuộc tính `Increment` của điều khiển `NumericUpDown` - Chỉ định xem giá trị hiện tại của điều khiển `NumericUpDown` thay đổi bao nhiêu khi người dùng nhấn vào mũi tên lên (để tăng) hay mũi tên xuống (để giảm).

thuộc tính `Maximum` của điều khiển `NumericUpDown` - Cho biết giá trị đầu vào lớn nhất của điều khiển `NumericUpDown`.

thuộc tính Minimum của điều khiển NumericUpDown - Cho biết giá trị đầu vào nhỏ nhất của điều khiển NumericUpDown.

thuộc tính Multiline của điều khiển TextBox - Cho biết TextBox có khả năng hiển thị nhiều dòng văn bản hay không. Nếu giá trị thuộc tính này là True thì TextBox có thể chứa nhiều dòng văn bản, còn nếu giá trị thuộc tính là False thì TextBox chỉ có thể chứa một dòng văn bản.

thuộc tính ReadOnly của điều khiển TextBox - Cho biết người dùng có thể thay đổi giá trị của TextBox hay không.

thuộc tính ScrollBars của điều khiển TextBox - Cho biết TextBox có thanh cuộn hay không và nếu có thì là kiểu gì. Theo mặc định, thuộc tính ScrollBars được thiết lập giá trị None.

tiêu đề For...Next - Dòng đầu tiên của lệnh lặp For...Next. Tiêu đề For...Next xác định cả bốn yếu tố cơ bản trong lệnh lặp được điều khiển bởi biến đếm.

toán tử &= - Thay đổi String ở bên trái toán tử bằng cách thêm giá trị của String bên phải vào cuối String này.

tự động xác định kiểu cục bộ (local type inference) - Là tính năng của trình biên dịch Visual Basic 2008, cho phép xác định kiểu biến cục bộ dựa trên ngữ cảnh mà biến đó được khởi tạo.

từ khóa For - Bắt đầu lệnh For...Next.

từ khóa Step - Thành phần không bắt buộc của tiêu đề For...Next, chỉ ra bước tăng hoặc bước giảm.

từ khóa To - Sử dụng để chỉ định khoảng giá trị. Thường dùng trong tiêu đề For... Next để chỉ định các giá trị đầu và cuối của biến điều khiển trong lệnh.

HƯỚNG DẪN THIẾT KẾ GIAO DIỆN

TextBox

- Nếu muốn TextBox sẽ hiển thị nhiều dòng, hãy thiết lập giá trị thuộc tính Multiline là True và canh lề trái đầu ra bằng cách thiết lập giá trị thuộc tính TextAlign là Left.
- Nếu TextBox được dùng để hiển thị kết quả, thiết lập thuộc tính ReadOnly giá trị là True để đảm bảo rằng người dùng không thể thay đổi kết quả đầu ra.
- Nếu TextBox nhiều dòng hiển thị kết quả có quá nhiều dòng, hãy giới hạn chiều cao TextBox và sử dụng thanh cuộn dọc để cho phép người dùng xem được những dòng phía dưới.

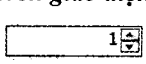
NumericUpDown

- Điều khiển NumericUpDown tuân thủ nguyên tắc thiết kế giao diện như đối với TextBox.
- Sử dụng điều khiển NumericUpDown để giới hạn khoảng giá trị số do người dùng nhập vào.

ĐIỀU KHIỂN, SỰ KIỆN, THUỘC TÍNH & PHƯƠNG THỨC

NumericUpDown NumericUpDown Điều khiển này cho phép bạn chỉ ra các giá trị số lớn nhất và nhỏ nhất có thể được nhập vào.

- ***Trên giao diện khi ứng dụng chạy***



- ***Thuộc tính***

  Increment - Chỉ ra giá trị hiện thời của điều khiển NumericUpDown sẽ thay đổi bao nhiêu khi người dùng nhấn mũi tên đi lên (để tăng) hoặc xuống (để giảm) trên điều khiển.

  Location - Chỉ ra vị trí tương đối của điều khiển NumericUpDown so với góc trên bên trái của Form.

`Maximum` - Cho biết giá trị nhập vào lớn nhất của điều khiển `NumericUpDown`.

`Minimum` - Cho biết giá trị nhập vào nhỏ nhất của điều khiển `NumericUpDown`.

`Name` - Chỉ ra tên để truy cập điều khiển `NumericUpDown` khi lập trình. Ta nên thêm hậu tố `UpDown` vào sau tên.

`Size` - Chỉ ra chiều rộng và chiều cao (theo pixel) của điều khiển `NumericUpDown`.

`TextAlign` - Chỉ ra cách canh lề văn bản trên điều khiển `NumericUpDown`.

`Value` - Chỉ ra giá trị của điều khiển `NumericUpDown`.

`Width` - Chỉ ra chiều rộng (theo pixel) của `NumericUpDown`.

`TextBox` `TextBox` Điều khiển này cho phép người dùng nhập dữ liệu từ bàn phím.

- ***Trên giao diện khi ứng dụng chạy***

  0

- ***Sự kiện***

  `TextChanged` - Được kích hoạt khi văn bản trong `TextBox` bị thay đổi.

- ***Thuộc tính***

  `Location` - Chỉ ra vị trí tương đối của `TextBox` so với góc trên bên trái của `Form`.

  `Multiline` - Cho biết `TextBox` có khả năng hiển thị nhiều dòng văn bản hay không.

  `Name` - Chỉ ra tên để truy cập `TextBox` trong khi lập trình. Bạn nên thêm hậu tố `TextBox` vào phía sau tên.

  `ReadOnly` - Cho biết người dùng có thể thay đổi nội dung trong `TextBox` không.

  `ScrollBars` - Cho biết `TextBox` nhiều dòng có chứa thanh cuộn dọc và/ hoặc thanh cuộn ngang hay không.

  `Size` - Chỉ ra chiều rộng và chiều cao của `TextBox`.

  `Text` - Chỉ ra văn bản được hiển thị trong `TextBox`.

  `TextAlign` - Chỉ ra cách căn lề văn bản trên `TextBox`.

  `Width` - Chỉ ra chiều rộng của `TextBox` (theo pixel)

- ***Phương thức***

  `Clear` - Xóa nội dung văn bản trên `TextBox`.

  `Focus` - Chuyển focus của ứng dụng đến `TextBox` gọi hàm này.

CÂU HỎI TRẮC NGHIỆM

11.1 "Hello" thuộc kiểu dữ liệu_____.

a) `String` b) `StringLiteral`

c) `Character` d) `StringText`

11.2 Một_____cung cấp khả năng nhập hoặc hiển thị nhiều dòng văn bản trên cùng một điều khiển.

a) `TextBox` b) `NumericUpDown`

c) `MultilineTextBox` d) `NumericUpDown` nhiều dòng

11.3 Tiêu đề `For...Next` chỉ ra_____.

a) biến điều khiển và giá trị ban đầu của nó

b) bước tăng hoặc bước giảm

c) điều kiện tiếp tục vòng lặp

d) cả bốn yếu tố cơ bản của lệnh lặp được điều khiển bởi biến đếm

11.4 _____là không bắt buộc trong tiêu đề `For...Next` khi bước tăng của biến điều khiển là 1.

a) Từ khóa `To` b) Giá trị ban đầu của biến điều khiển

c) Từ khóa `Step` d) Giá trị cuối cùng của biến điều khiển

11.5 Thiết lập thuộc tính `ScrollBars` của `TextBox` giá trị là _____ để chỉ tạo ra thanh cuộn dọc.

a) `True` b) `Vertical`

c) `Up` d) `Both`

11.6_____được dùng để xác định xem vòng lặp `For...Next` có tiếp tục được lặp lại nữa không.

a) Giá trị đầu tiên của biến điều khiển b) Từ khóa `For`

c) Từ khóa `Step` d) Giá trị cuối cùng của biến điều khiển

11.7 Trong vòng lặp `For...Next`, biến điều khiển tăng (hoặc giảm)_____

a) sau khi thân của vòng lặp được thực thi

b) khi gặp từ khóa `To`

c) khi điều kiện tiếp tục vòng lặp là `False`

d) khi phần thân vòng lặp đang được thực thi

11.8 Thiết lập giá trị các thuộc tính_____của điều khiển `NumericUpDown` để chắc chắn rằng người dùng không thể nhập giá trị không hợp lệ vào điều khiển.

a) `Increment` và `Locked` b) `ScrollBars` và `ReadOnly`

c) `Minimum` và `Maximum` d) `ReadOnly` và `InValid`

11.9 Thuộc tính_____và_____giới hạn giá trị người dùng có thể nhập trên điều khiển `NumericUpDown`.

a) `Maximum`, `Minimum` b) `Top`, `Bottom`

c) `High`, `Low` d) `Max`, `Min`

11.10 Tiêu đề `For...Next`_____có thể được sử dụng để thay đổi biến điều khiển theo giá trị các số lẻ từ 1 đến 9.

a) `For i As Integer = 1 To 10 Step 1`

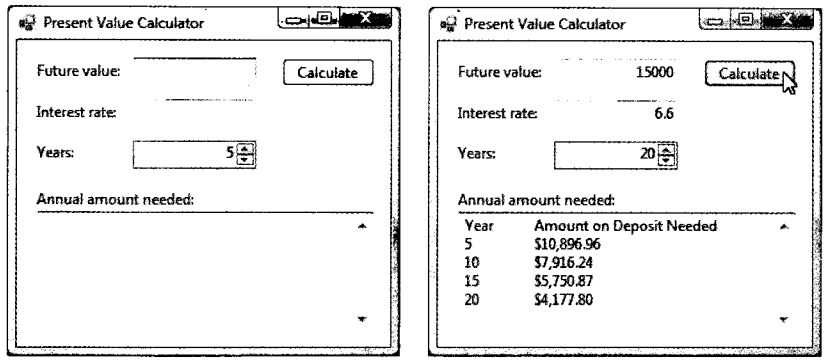
b) `For i As Integer = 1 To 10 Step 2`

c) `For i As Integer = 1 To 10 Step -1`

d) `For i As Integer = 1 To 10 Step -2`

BÀI TẬP

11.11 (***Ứng dụng Present Value Calculator***) Một ngân hàng muốn cho khách hàng của họ biết số tiền họ cần đầu tư ("giá trị hiện tại" – present value) để đạt được một số tiền mong muốn nào đó ("giá trị tương lai" - future value) trong 5, 10, 15, 20, 25 hoặc 30 năm sau. Người dùng nhập vào số tiền mong muốn của họ (số tiền họ muốn có sau một số năm nhất định), tỷ lệ lãi suất và thời hạn đầu tư tính bằng năm. Hãy tạo một ứng dụng tính toán và hiển thị tiền vốn ban đầu (khoản đầu tư ban đầu) cần thiết để người dùng có được số tiền mong muốn. Ứng dụng của bạn cần cho phép người dùng đầu tư tiền cho 5, 10, 15, 20, 25 hay 30 năm. Chẳng hạn, một khách hàng muốn có số tiền là 15.000 USD sau khoảng thời gian 5 năm với tỷ lệ lãi suất là 6,6% thì cần phải đầu tư 10.896,96 USD như ta thấy trên Hình 11.16. Hãy sử dụng toán tử &= và vòng lặp `For...Next` để thực hiện.



Chú thích hình

- ***Future value:*** *Giá trị tương lai*
- ***Interest rate:*** *Tỷ lệ lãi suất*
- ***Years:*** *Số năm*
- ***Amount on Deposit Needed:*** *Số tiền cần đầu tư*

Hình 11.16 Giao diện của ứng dụng **Present Value Calculator**.

a) ***Copy template của ứng dụng vào thư mục làm việc.*** Copy thư mục `C:\Examples\Tutorial11\Exercises\PresentValue` sang thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng.*** Nhấn đúp vào file `PresentValue.sln` ở thư mục `PresentValue` để mở ứng dụng.

c) ***Thêm điều khiển NumericUpDown.*** Định vị và thiết lập kích cỡ điều khiển `NumericUpDown` tuân thủ theo hướng dẫn thiết kế giao diện. Đặt giá trị thuộc tính `Name` của điều khiển `NumericUpDown` là `yearUpDown`. Thiết lập giá trị bước tăng của điều khiển `NumericUpDown` là 5. Để chỉ cho phép người dùng chọn một khoảng thời gian trong dãy giá trị, hãy đặt giá trị thuộc tính `ReadOnly` của điều khiển `yearUpDown` là `True` để người dùng phải chọn giá trị bằng cách sử dụng mũi tên lên xuống của điều khiển `yearUpDown` có các giá trị cách nhau 5 đơn vị.

d) ***Thêm TextBox nhiều dòng.*** Thêm `TextBox` vào `Form`, bên dưới điều khiển `NumericUpDown` và đặt giá trị thuộc tính `Name` là `resultTextBox`. Thiết lập để `TextBox` hiển thị nhiều dòng và có thanh cuộn. Thiết lập lại kích thước và vị trí của `TextBox` trên `Form` theo hướng dẫn thiết kế giao diện. Đảm bảo người dùng không thể thay đổi được văn bản trên `TextBox`.

e) ***Thêm xử lý sự kiện Click và viết mã.*** Thêm xử lý sự kiện `Click` cho `Button` **Calculate**. Sử dụng công thức tính toán sau để tính số tiền cần đầu tư:

$$p = a / (1 + r)^n$$

trong đó

p là số tiền cần thiết để đạt được số tiền mong muốn trong tương lai

a là số tiền mong muốn

r là tỷ lệ lãi suất hàng năm (chẳng hạn, .05 tương đương với 5%)

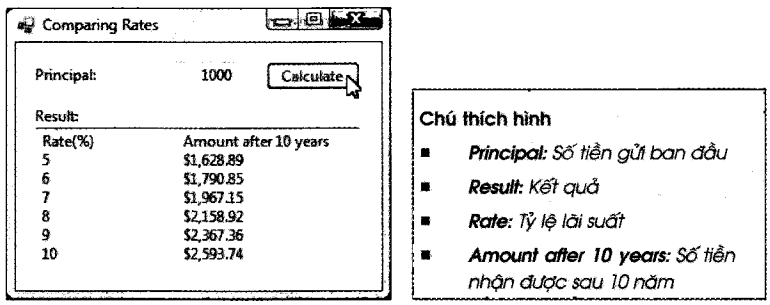
n là số năm

f) ***Chạy ứng dụng.*** Chọn **Debug > Start Debugging** để chạy ứng dụng. Nhập giá trị tương lai, tỷ lệ lãi suất và số năm. Nhấn `Button` **Calculate** và xác nhận rằng khoảng thời gian và số tiền đầu tư cần thiết tương ứng là đúng. Chạy thử ứng dụng một lần nữa, lần này nhập 30 cho số năm. Xác nhận rằng thanh cuộn dọc xuất hiện và hiển thị toàn bộ nội dung đầu ra.

g) ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

h) ***Đóng IDE***. Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

11.12 (***Ứng dụng Comparing Rates***). Viết một ứng dụng tính toán số tiền có trong tài khoản sau 10 năm với tỷ lệ lãi suất là 5-10%. Đối với ứng dụng này, người dùng phải cung cấp số vốn ban đầu.



Hình 11.17 Giao diện của ứng dụng **Comparing Rates**.

a) ***Copy template của ứng dụng vào thư mục làm việc***. Copy thư mục `C:\Examples\Tutorial11\Exercises\ComparingRates` vào thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng***. Nhấn đúp vào file `ComparingRates.sln` ở thư mục `ComparingRates` để mở ứng dụng.

c) ***Thêm TextBox nhiều dòng***. Thêm một `TextBox` vào `Form` bên dưới `Label` **Result:**. Thiết lập để `TextBox` hiển thị nhiều dòng. Thay đổi kích cỡ và vị trí của `TextBox` trên `Form` sao cho tuân thủ theo hướng dẫn thiết kế giao diện (Hình 11.17). Đảm bảo rằng người dùng không thể thay đổi nội dung trên `TextBox`.

d) ***Thêm xử lý sự kiện Click và viết mã***.Thêm xử lý sự kiện `Click` cho `Button` **Calculate**. Trong chế độ **Code**, viết mã cho ứng dụng sao cho khi nhấn `Button` **Calculate** thì `TextBox` nhiều dòng hiển thị số tiền có trong tài khoản sau 10 năm tương ứng cho các tỷ lệ lãi suất 5, 6, 7, 8, 9 và 10%. Sử dụng công thức tính sau đây:

$$a = p(1 + r)^n$$

trong đó

a là số tiền nhận được cuối năm thứ n

p là số tiền đầu tư ban đầu

r là tỷ lệ lãi suất hàng năm (chẳng hạn, .05 tương đương với 5%)

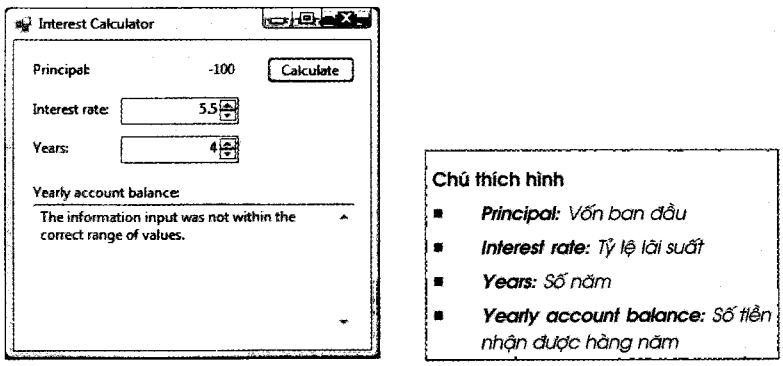
n là số năm

e) ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng. Nhập số tiền vốn ban đầu cho tài khoản và nhấn `Button` **Calculate**. Xác nhận rằng chương trình hiển thị đúng số tiền có được sau 10 năm tương ứng với các tỷ lệ lãi suất từ 5-10%.

f) ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

g) ***Đóng IDE***. Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

11.13 (***Kiểm tra tính hợp lệ dữ liệu đầu vào của ứng dụng Interest Calculator***). Nâng cấp ứng dụng **Interest Calculator** để thêm tính năng kiểm tra lỗi. Kiểm tra xem người dùng có nhập những giá trị hợp lệ cho vốn ban đầu và tỷ lệ lãi suất hay không. Nếu người dùng nhập một giá trị không hợp lệ thì hiển thị thông báo trên `TextBox` nhiều dòng. Hình 11.18 biểu diễn cách ứng dụng xử lý khi có một giá trị đầu vào không hợp lệ.



Hình 11.18 Ứng dụng **Interest Calculator** với tính năng kiểm tra lỗi.

a) ***Copy template của ứng dụng vào thư mục làm việc***. Copy thư mục `C:\Examples\Tutorial11\Exercises\InterestCalculatorEnhancement` vào thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng***. Nhấn đúp vào file `InterestCalculator.sln` ở thư mục `InterestCalculatorEnhancement` để mở ứng dụng.

c) ***Thêm điều khiển NumericUpDown***. Thay thế `TextBox` **Interest rate:** bằng điều khiển `NumericUpDown`. Cho phép người dùng nhập các giá trị chỉ trong khoảng 0-100. Thiết lập giá trị thuộc tính `DecimalPlaces` là `1` để cho phép người dùng nhập vào giá trị có một chữ số phần thập phân. Thiết lập giá trị cho thuộc tính `Interval` là `0.1`.

d) ***Thay đổi xử lý sự kiện Click***. Thay đổi mã trong xử lý sự kiện `Click` của `Button` **Calculate** để kiểm tra đầu vào. Vốn ban đầu phải là một số dương. Đồng thời đọc tỷ lệ lãi suất từ `NumericUpDown` **Interest Rate:**.

e) ***Hiển thị thông báo lỗi***. Hiển thị văn bản "`The information input was not within the correct range of values.`" (Giá trị nhập vào không nằm trong dãy giá trị hợp lệ) trên `resultTextBox` nếu giá trị vốn ban đầu không hợp lệ.

f) ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng. Nhập dữ liệu không hợp lệ cho vốn ban đầu và tỷ lệ lãi suất. Dữ liệu không hợp lệ cho tỷ lệ lãi suất tự động được điều chỉnh về phạm vi giữa các giá trị nhỏ nhất và lớn nhất có thể của điều khiển. Dữ liệu không hợp lệ của vốn ban đầu có thể bao gồm số âm hoặc các giá trị bắt đầu bằng chữ cái. Xác nhận rằng khi nhập dữ liệu vốn ban đầu không hợp lệ và nhấn `Button` **Calculate** thì ứng dụng đưa ra thông báo lỗi như Hình 11.18.

g) ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

h) ***Đóng IDE***. Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

Đoạn mã này làm gì? ▶ **11.14** Hãy cho biết giá trị của biến `result` sau khi đoạn mã dưới đây được thực thi. Giả thiết rằng `power`, `result` và `number` đều được khai báo kiểu `Integer`.

```
1  power = 5
2  number = 10
3  result = number
4
5  For i As Integer = 1 To (power-1)
6     resrult *= number
7  Next
```

Đoạn mã này có gì sai? ▶ **11.15** Hãy tìm và sửa lỗi mỗi trường hợp sau đây:

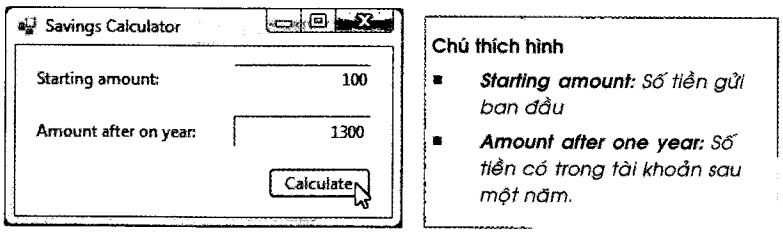
a) Lệnh này dùng để hiển thị tất cả các số từ 100 tới 1 trong `ListBox` theo thứ tự giảm dần.

```
1  For counter As Integer = 100 to 1
2          displayListBox.Items.Add(counter)
3      Next
```

b) Lệnh này phải hiển thị các số lẻ từ 19 đến 1 trong `ListBox` theo thứ tự giảm dần.

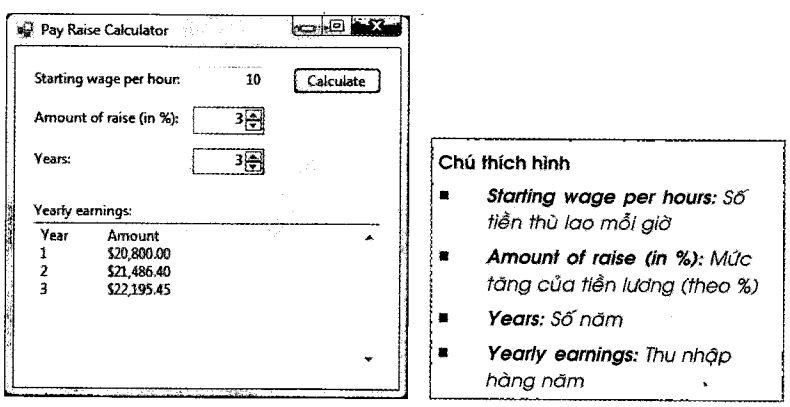
```
1  For counter As Integer = 19 To 1 By -1
2          displayListBox.Items.Add(counter)
3      Next
```

Sử dụng trình gỡ lỗi ▶ **11.16** (*Ứng dụng **Savings Calculator***) Ứng dụng **Savings Calculator** tính toán số tiền người dùng sẽ có trong tài khoản sau một năm. Ứng dụng lấy số tiền gửi ban đầu từ người dùng và giả định rằng người dùng sẽ gửi thêm 100USD vào tài khoản mỗi tháng trong vòng một năm. Tiền lãi không được cộng vào số dư tài khoản. Khi chạy thử ứng dụng, bạn nhận thấy rằng số tiền người dùng có trong tài khoản cuối năm được tính không đúng. Sử dụng trình gỡ lỗi để xác định và sửa lỗi lôgíc trong chương trình. Hình 11.19 hiển thị kết quả đúng khi ứng dụng đã được sửa lỗi.



Hình 11.19 Đầu ra đúng cho ứng dụng **Savings Calculator.**

Bài tập nâng cao ▶ **11.17** (*Ứng dụng **Pay Raise Calculator***) Phát triển một ứng dụng tính tiền lương cho công nhân mỗi năm trong vòng số năm do người dùng nhập vào. Công nhân nhận tiền lương theo giờ và được tăng lương mỗi năm một lần. Để đơn giản, giả sử rằng công nhân làm 40 giờ một tuần, 52 tuần một năm. Người dùng nhập vào tiền công mỗi giờ và mức tăng của tiền lương mỗi năm (tính bằng % theo năm).



Hình 11.20 Giao diện của ứng dụng **Pay Raise Calculator.**

a) ***Copy template của ứng dụng vào thư mục làm việc***. Copy thư mục `C:\Examples\Tutorial11\Exercises\PayRaise` vào thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng***. Nhấn đúp vào file `PayRaise.sln` ở thư mục `PayRaise` để mở ứng dụng.

c) ***Thêm các điều khiển vào Form***. Thêm hai điều khiển `NumericUpDown` vào `Form`. Điều khiển `NumericUpDown` thứ nhất cho phép người dùng chỉ ra tỷ lệ phần trăm mức tăng của lương. Người dùng chỉ có thể nhập vào tỷ lệ phần trăm trong khoảng 3-8%. Tạo điều khiển `NumericUpDown` thứ hai để người dùng chọn số năm trong khoảng từ 1-50. Sau đó thêm điều khiển `TextBox` nhiều dòng và thiết lập thuộc tính `ScrollBar` của `TextBox` này để hiển thị thanh cuộn dọc. Hãy chắc chắn rằng người dùng không thể thay đổi nội dung trong các điều khiển `NumericUpDown` và `TextBox`. Hãy thay đổi kích thước và di chuyển các điều khiển vừa tạo tuân thủ theo hướng dẫn thiết kế giao diện như trong Hình 11.20.

d) ***Viết xử lý sự kiện `Click` và mã***. Thêm xử lý sự kiện `Click` cho `Button` **Calculate**. Chuyển sang chế độ **Code** và viết mã dùng lệnh `For...Next` để tính tiền lương mỗi năm dựa trên mức tăng lương hàng năm.

e) ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng. Nhập tiền lương theo giờ khởi điểm, mức tăng lương hàng năm và số năm làm việc. Nhấn `Button` **Calculate** và đảm bảo `TextBox` **Yearly earnings:** hiển thị đúng số tiền công nhân nhận được sau mỗi năm.

f) ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

g) ***Đóng IDE***. Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

CHƯƠNG 12

Mục tiêu

Trong chương này, bạn sẽ tìm hiểu để:

- Sử dụng lệnh đa lựa chọn **Select Case**.
- Sử dụng lệnh **Case**.
- Sử dụng từ khóa **Is**.
- Lấy giá trị ngày tháng và thời gian hiện tại.
- Hiển thị ngày tháng và thời gian.
- Sử dụng thuộc tính **PasswordChar** của **TextBox**.

Nội dung chính

Ứng dụng Security Panel

Giới thiệu lệnh đa lựa chọn Select Case

Trong chương trước, bạn đã học cách sử dụng lệnh `For...Next`, lệnh ngắn gọn nhất để thực hiện vòng lặp điều khiển bởi biến đếm. Trong chương này, bạn sẽ tìm hiểu lệnh đa lựa chọn `Select Case`. Lệnh `Select Case` được sử dụng để đơn giản hóa mã có nhiều lệnh `ElseIf` liên tục khi ứng dụng phải chọn giữa nhiều hành động để thực hiện.

12.1 Chạy thử ứng dụng Security Panel

Trong chương này, bạn dùng lệnh đa lựa chọn `Select Case` để xây dựng ứng dụng **Security Panel**. Ứng dụng này phải đáp ứng những yêu cầu sau đây:

Yêu cầu đối với ứng dụng

Một phòng thí nghiệm muốn lắp đặt bảng an ninh phía ngoài phòng. Chỉ nhân viên được cho phép mới có thể vào trong phòng bằng cách sử dụng mã an ninh. Sau đây là các mã an ninh hợp lệ (còn được gọi là mã truy cập) và các nhóm nhân viên tương ứng:

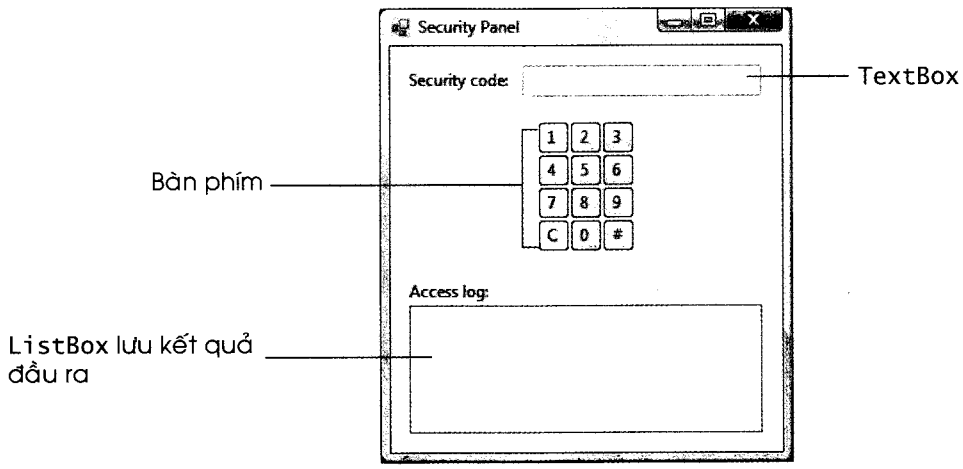
Giá trị	*Nhóm*
1645-1689	*Kỹ thuật viên*
8345	*Nhân viên tạp vụ*
9998, 1006-1008	*Nhà khoa học*

Mỗi khi nhập mã an ninh, truy cập sẽ được chấp nhận hoặc bị từ chối. Thông tin mỗi lần truy cập đều được ghi vào cửa sổ bên dưới bàn phím. Nếu truy cập được chấp nhận thì thông tin về thời gian truy cập và nhóm (nhà khoa học, nhân viên tạp vụ, v.v..) được ghi vào cửa sổ. Nếu truy cập bị từ chối thì thông tin về thời gian và thông báo "Access Denied" sẽ được ghi vào cửa sổ. Ngoài ra, người dùng có thể nhập mã truy cập là một chữ số bất kỳ để yêu cầu nhân viên bảo vệ đến trợ giúp. Sau đó, thông tin về ngày tháng, thời gian và thông báo "Assistance Requested" cũng được ghi vào cửa sổ để xác định yêu cầu đã được tiếp nhận.

Bạn bắt đầu bằng việc chạy thử ứng dụng đã hoàn thiện. Sau đó, bạn tìm hiểu những tính năng bổ sung cần thiết của Visual Basic để xây dựng cho mình một phiên bản của ứng dụng này.

Chạy thử ứng dụng Security Panel

1. ***Mở ứng dụng đã hoàn thiện.*** Mở thư mục `C:\Examples\Tutorial12\CompletedAplication\SecurityPanel` và tìm đến ứng dụng **Security Panel**. Nhấn đúp vào file `SecurityPanel.sln` để mở ứng dụng trong Visual Basic IDE.
2. ***Chạy ứng dụng Security Panel.*** Chọn **Debug > Start Debugging** để chạy ứng dụng (Hình 12.1). Ở phía trên của `Form`, có một `TextBox` hiển thị dấu sao *(*)* cho mỗi chữ số trong mã an ninh được nhập từ qua bàn phím trên giao diện. Hãy lưu ý rằng bàn phím trên giao diện trông giống như một bàn phím thật. `Button C` xóa dữ liệu nhập hiện thời và `Button #` ra lệnh cho ứng dụng xử lý mã an ninh vừa được nhập vào. Kết quả sẽ được hiển thị ở `ListBox` ở phía dưới `Form`.

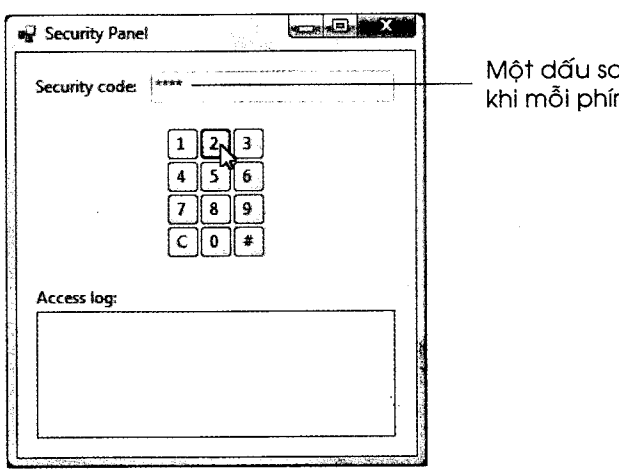


Hình 12.1 Thực thi ứng dụng **Security Panel.**

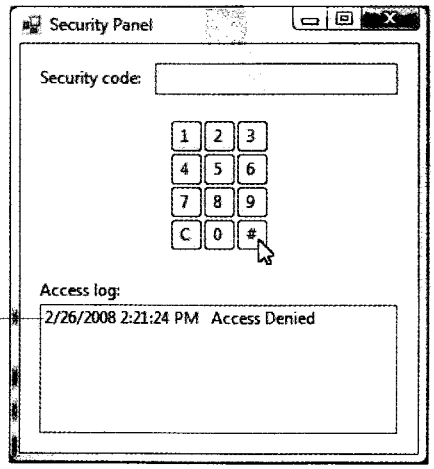
3. ***Nhập mã an ninh không hợp lệ.*** Dùng bàn phím để nhập mã an ninh không hợp lệ 1212. Lưu ý rằng một dấu sao (*) được hiển thị trong `TextBox` (Hình 12.2) khi mỗi phím số được nhấn trên `Form`. Tiếp theo, hãy nhấn `Button #`. Một thông báo cho biết truy cập bị từ chối xuất hiện trên `ListBox` như trong Hình 12.3. Lưu ý rằng `TextBox` bị xóa nội dung khi ta nhấn `Button #`.

Mẹo thiết kế giao diện

Nếu giao diện mô phỏng một thiết bị có thật, thiết kế của giao diện đó cần tương tự như hình dáng vật lý của thiết bị đó.



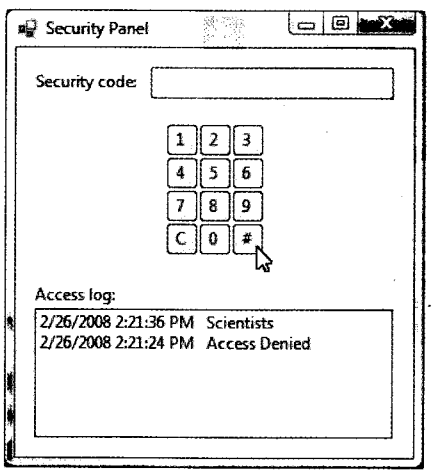
Hình 12.2 Các dấu sao được hiển thị trên trường **Security code:**.



Thông báo cho biết mã an ninh không hợp lệ đã được nhập

Hình 12.3 Security Panel hiển thị thông báo **Access Denied.**

4. ***Sử dụng Button C.*** Nhấn vài phím số, sau đó nhấn `Button` **C**. Lưu ý rằng tất cả các dấu sao trong `TextBox` biến mất. Người dùng thường nhấn nhầm phím, vì vậy `Button` **C** cho phép người dùng nhập lại từ đầu.
5. ***Nhập mã an ninh hợp lệ.*** Dùng bàn phím nhập 1006, sau đó nhấn `Button` `#`. Lưu ý thông báo thứ hai xuất hiện trong `ListBox` như Hình 12.4.



Thông báo hiển thị khi nhập vào mã an ninh hợp lệ

Hình 12.4 Ứng dụng **Security Panel** thông báo mã an ninh hợp lệ.

6. ***Đóng ứng dụng.*** Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.
7. ***Đóng IDE.*** Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

12.2 Giới thiệu lệnh đa lựa chọn Select Case

Trong mục này bạn sẽ tìm hiểu cách sử dụng **lệnh đa lựa chọn `Select Case`.** Với mục đích so sánh, chúng tôi cung cấp một lệnh đa lựa chọn `If...Then...Else` hiển thị thông báo dựa trên điểm số của sinh viên.

```
If grade = "A" Then
      displayLable.Text  =  "Excellent!"
ElseIf grade = "B" Then
      displayLable.Text  =  "Very good!"
```

```
ElseIf grade = "C" Then
       displayLable.Text  =  "Good."
ElseIf grade = "D" Then
       displayLable.Text  =  "Poor."
ElseIf grade = "F" Then
       displayLable.Text  =  "Failure."
Else
       displayLable.Text  =  "Invalid Grade."
End If
```

Lệnh này được dùng để đưa ra kết quả đúng khi lựa chọn giữa nhiều giá trị khác nhau. [*Lưu ý:* các phép so sánh chuỗi có phân biệt chữ hoa chữ thường - "A" khác với "a".] Tuy nhiên sử dụng lệnh `Select Case` sẽ đơn giản hóa mỗi câu lệnh có dạng

```
If grade = "A" Then
```

bằng một câu như

```
Case "A"
```

và loại bỏ các từ khóa `If` và `ElseIf`.

Lệnh đa lựa chọn `Select Case` sau đây có cùng một chức năng như lệnh `If...Then...Else` trên.

```
Select Case grade
       Case "A"
              displayLable.Text  =  "Excellent!"
       Case "B"
              displayLable.Text  =  "Very good!"
       Case "C"
              displayLable.Text  =  "Good."
       Case "D"
              displayLable.Text  =  "Poor."
       Case "F"
              displayLable.Text  =  "Failure."
       Case Else
              displayLable.Text  =  "Invalid Grade."
End Select
```

Thói quen lập trình tốt

Visual Basic tự động lùi đầu dòng các lệnh ở thân mỗi mệnh đề Case để mã nguồn dễ đọc hơn.

Lỗi lập trình thường gặp

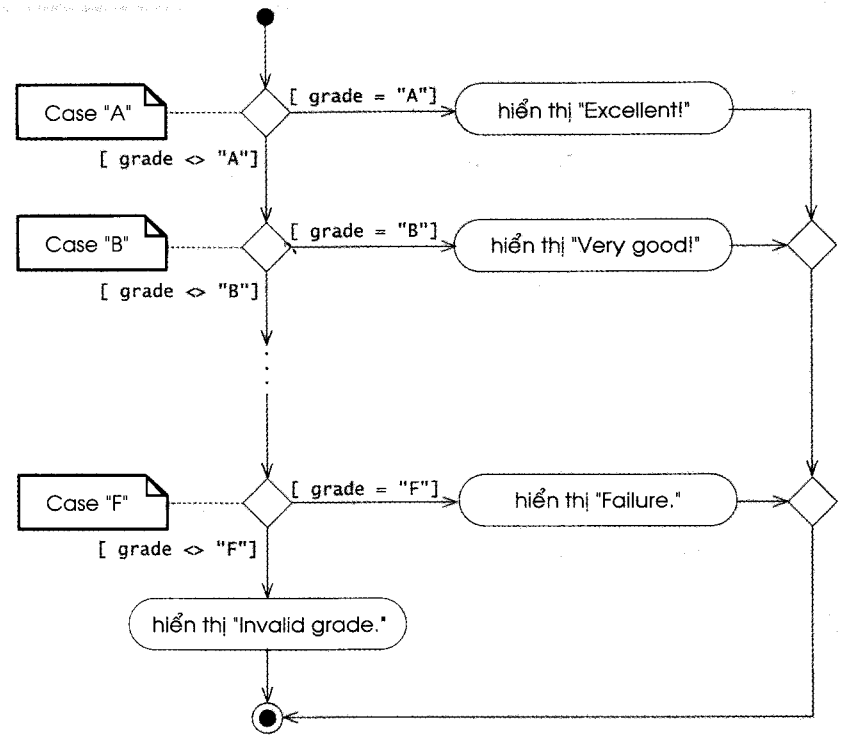
Khi sử dụng tùy chọn Case Else trong lệnh Select Case, không đặt Case Else sau mệnh đề Case cuối cùng là một lỗi cú pháp.

Lỗi lập trình thường gặp

Các lệnh Case có cùng một giá trị trong các danh sách biểu thức sẽ gây ra lỗi lôgíc. Khi chạy, chỉ có phần thân mệnh đề Case đầu tiên thỏa mãn với điều kiện được thực thi.

Lệnh `Select Case` bắt đầu với từ khóa `Select Case` và sau đó là một **biểu thức kiểm tra** (còn được gọi là **biểu thức điều khiển**) và kết thúc bằng từ khóa **`End Select`**. Biểu thức kiểm tra được chỉ ra một lần ở dòng đầu tiên của lệnh `Select Case` và được sử dụng trong mỗi lệnh `Case`. Lệnh `Select Case` trên có chứa năm **lệnh `Case`** và **lệnh `Case Else`** không bắt buộc. Mỗi lệnh `Case` chứa từ khóa `Case` và sau đó là một **danh sách biểu thức**. Danh sách biểu thức có thể chứa mọi kiểu dữ liệu có sẵn, chẳng hạn như chuỗi ("A") hay các giá trị số (707 hay 9.9). Mỗi danh sách biểu thức của lệnh `Case` được so sánh với `grade` – biểu thức điều khiển của lệnh `Select Case`. Mặc dù một lệnh `Select Case` không bị giới hạn số lệnh `Case` nhưng lệnh này chỉ được phép có một `Case Else`.

Hình 12.5 minh họa biểu đồ hoạt động UML của lệnh đa lựa chọn `Select Case` này. Điều kiện thứ nhất được đánh giá là `grade = "A"`. Nếu điều kiện này là `True` thì văn bản `"Excellent!"` được hiển thị và ứng dụng tiếp tục thực thi lệnh đầu tiên sau lệnh `Select Case`. Nếu điều kiện được đánh giá là `False`, nghĩa là `grade <> "A"`), lệnh tiếp tục với việc kiểm tra điều kiện tiếp theo, `grade = "B"`. Nếu điều kiện này là `True`, văn bản `"Very good!"` sẽ được hiển thị và điều khiển tiếp tục thực thi lệnh thứ nhất sau lệnh `Select Case`. Nếu điều kiện này là `False` (nghĩa là, `grade <> "B"`) thì lệnh tiếp tục kiểm tra điều kiện tiếp theo. Quá trình này tiếp tục cho đến khi tìm được một lệnh `Case` thích hợp hoặc cho đến khi điều kiện cuối cùng được đánh giá là `False` (`grade <> "F"`). Nếu khả năng thứ hai xảy ra, phần thân lệnh `Case Else` được thực thi và văn bản `"Invalid grade"` sẽ được hiển thị. Sau đó ứng dụng tiếp tục thực thi lệnh đầu tiên sau lệnh `Select Case`.



Hình 12.5 Biểu đồ hoạt động UML của lệnh đa lựa chọn `Select Case`.

TỰ ÔN TẬP

1. `Select Case` là lệnh_____lựa chọn.
   a) đa
   b) hai
   c) đơn
   d) Các câu trả lời trên đều sai
2. Phần thân của mệnh đề `Case Else` được thực thi khi nào?
   a) Mỗi lần lệnh `Case Else` thực thi
   b) Khi có nhiều hơn một mệnh đề `Case` đúng
   c) Khi tất cả các mệnh đề `Case` trong lệnh `Select Case` đều đúng
   d) Các câu trả lời trên đều sai

Đáp án: **1)** a. **2)** d.

12.3 Xây dựng ứng dụng Security Panel

Ứng dụng **Security Panel** chứa 10 `Button` để hiển thị các chữ số. (Ta gọi chúng là các `Button` số.) Bạn sẽ tạo xử lý sự kiện cho mỗi sự kiện `Click` của `Button`. Các `Button` này tạo thành bàn phím trên giao diện. Mã giả sau đây mô tả xử lý sự kiện cho mỗi sự kiện `Click` của các `Button` số này:

```
Nếu một Button số được nhấn thì
    Ghép giá trị số của Button vào giá trị thuộc tính Text của TextBox
```

Ở phần tiếp theo của chương này, bạn sẽ chuyển đổi đoạn mã giả này thành mã Visual Basic và tạo các xử lý sự kiện `Click` cho mỗi `Button` số. Sau đó người dùng có thể sử dụng các `Button` này để nhập các chữ số và và các chữ số này sẽ được nối vào sau văn bản trong `TextBox`.

Cùng với các `Button` số, ứng dụng này còn chứa `Button` **`C`** và `Button #`. `Button` **`C`** để xóa nội dung trên `TextBox`. Đoạn mã giả cho xử lý sự kiện của `Button #` như sau:

```
Khi người dùng nhấn Button # thì
```

```
Lấy giá trị mã an ninh do người dùng nhập vào
Xóa nội dung trên TextBox
Chọn trường hợp đúng dựa trên mã truy cập
    Trong trường hợp mã truy cập nhỏ hơn 10
        Lưu văn bản "Assistance Requested" vào biến String
    Trong trường hợp mã truy cập nằm trong dãy 1645 đến 1689
        Lưu văn bản "Technicians" vào biến String
    Trong trường hợp mã truy cập bằng 8345
        Lưu văn bản "Custodians" vào biến String
    Trong trường hợp mã truy cập bằng 9998 hoặc trong dãy 1006
    đến 1008
        Lưu văn bản "Scientists" vào biến String
    Trong trường hợp không thỏa mãn các trường hợp trên đây
        Lưu văn bản "Access Denied" vào biến String
Chèn thông báo chứa thời gian hiện thời và nội dung biến của
    String vào nội dung hiển thị của ListBox
```

Bây giờ khi đã chạy thử ứng dụng **Security Panel** và tìm hiểu về mã giả của ứng dụng, bạn dùng bảng ACE để chuyển đổi mã giả thành Visual Basic. Hình 12.6 liệt kê các hành động, điều khiển, sự kiện giúp bạn tự hoàn thiện cho mình một phiên bản của ứng dụng này. Dòng đầu chỉ ra rằng bạn sử dụng `Label` để định danh cho các điều khiển `TextBox` và `ListBox`. Dòng thứ hai chỉ ra các `Button` để mô phỏng bàn phím số. Khi các `Button` này được nhấn, giá trị của chúng được nối với văn bản thuộc tính `Text` của `TextBox`. Dòng tiếp theo cho biết rằng `TextBox securityCode` lưu mã an ninh do người dùng nhập vào. Hai dòng tiếp theo cho biết rằng người dùng có thể nhấn `Button` **Clear** để xóa `TextBox`. Các dòng còn lại của bảng cho biết chức năng của `enterButton`.

Bảng ACE cho ứng dụng Security Panel

Hành động	Điều khiển	Sự kiện
Hiển thị Label cho các trường của ứng dụng	secrityCode-Label, accessLogLabel	
	oneButton, twoButton, threeButton, fourButton, fiveButton, sixButton, sevenButton, eightButton, nineButton, zeroButton	Click
Nối chữ số của các Button vào sau giá trị thuộc tính Text của TextBox	securityCode-TextBox	
	clearButton	Click
Xóa nội dung trên TexBox	securityCode-TextBox	
	enterButton	
Lấy giá trị mã an ninh do người dùng nhập vào	securityCode-TextBox	
Xóa nội dung trên TextBox	securityCode-TextBox	
Chọn trường hợp đúng dựa trên mã truy cập		

Hình 12.6 Bảng ACE của ứng dụng **Security Panel** (Phần 1/2).

Hành động	Điều khiển	Sự kiện
Trong trường hợp mã truy cập nhỏ hơn 10 Lưu văn bản "Assistance Requested" vào biến String		Click
Trong trường hợp mã truy cập nằm trong dãy 1645 đến 1689 Lưu văn bản "Technicians" vào biến String		
Trong trường hợp mã truy cập bằng 8345 Lưu văn bản "Custodians" vào biến String		
Trong trường hợp mã truy cập bằng 9998 hoặc trong dãy 1006 đến 1008 Lưu văn bản "Scientists" vào biến String		
Trong trường hợp không thỏa mãn các trường hợp trên đây Lưu văn bản "Access Denied" vào biến String		
Chèn thông báo chứa thời gian hiện thời và nội dung biến của String vào nội dung hiển thị của ListBox	logEntryListBox	

Hình 12.6 Bảng ACE của ứng dụng **Security Panel** (Phần 2/2).

Bây giờ khi đã quen với lệnh đa lựa chọn `Select Case`, bạn sẽ dùng lệnh này để xây dựng ứng dụng **Security Panel**.

Sử dụng thuộc tính PasswordChar

1. ***Copy template vào thư mục làm việc.*** Copy thư mục `C:\Examples\Tutorial12\TemplateAplication\SecurityPanel` vào thư mục `C:\SimplyVB2008`.
2. ***Mở file template của ứng dụng Security Panel.*** Nhấn đúp vào file `SecurityPanel.sln` trong thư mục `SecurityPanel`.
3. ***Hiển thị ký tự * trong TextBox.*** Chọn `TextBox` **Security Code:**, trong cửa sổ **Properties** thiết lập giá trị thuộc tính `PasswordChar` của `TextBox` là *. Có thể giấu hoặc ngụy trang văn bản hiển thị trên `TextBox` bằng ký tự được chỉ ra trong thuộc tính **PasswordChar**. Các **ký tự ngụy trang (masking character)** được hiển thị thay cho văn bản thực sự mà người dùng nhập vào. Tuy nhiên, thuộc tính `Text` của `TextBox` vẫn chứa văn bản do người dùng nhập vào. Chẳng hạn, nếu người dùng nhập vào 5469, `TextBox` hiển thị **** nhưng vẫn lưu 5469 trong thuộc tính `Text`. Bây giờ mọi ký tự được hiển thị trên `TextBox` này đều là ký tự *.

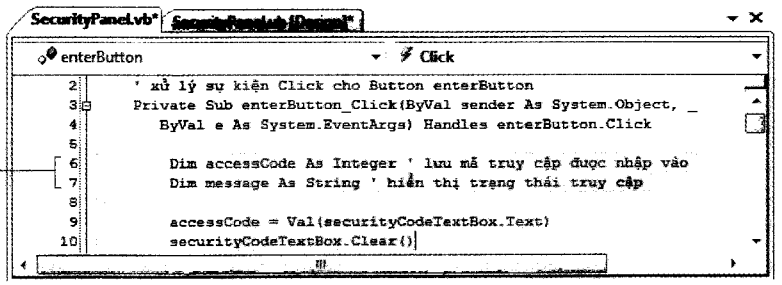
Mẹo thiết kế giao diện

Che dấu password hoặc các thông tin nhạy cảm khác trong `TextBox`.

4. ***Vô hiệu hóa TextBox.*** Lý do chính trong việc sử dụng `TextBox` thay cho `Label` để hiển thị mã truy cập là sử dụng thuộc tính `PasswordChar`. Để ngăn chặn người dùng thay đổi nội dung của `TextBox`, hãy thiết lập giá trị `False` cho thuộc tính **Enabled**. Bạn cũng có thể thực hiện điều này bằng cách sử dụng thuộc tính `ReadOnly`.
5. ***Tạo xử lý sự kiện enterButton_Click.*** Nhấn đúp vào `Button #` để tạo xử lý sự kiện `enterButton_Click`.

6. ***Khai báo và khởi tạo các biến.*** Thêm các dòng 6-10 trên Hình 12.7 vào xử lý sự kiện `enterButton_Click`. Dòng 6-7 khai báo các biến `accessCode` và `message` để lưu mã an ninh của người dùng (mã truy cập) và thông báo được hiển thị đối với người dùng (dựa trên mã truy cập được nhập vào). Dòng 9 thiết lập giá trị biến `accessCode` là mã an ninh do người dùng nhập vào. Ban đầu `securityCodeTextBox` chứa một chuỗi rỗng. Nếu người dùng nhấn `Button #` mà không nhập mã an ninh, hàm `Val` trả về 0. Dòng 10 xóa nội dung của `TextBox` **Security code:**.

Khai báo các biến của xử lý sự kiện



Hình 12.7 Các khai báo biến cho `enterButton_Click`.

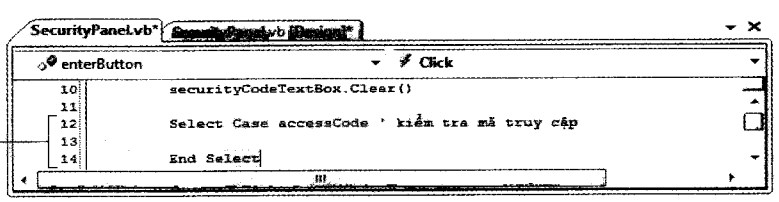
7. ***Lưu project.*** Chọn **File > Save All** để lưu mã đã được sửa đổi.

Bây giờ bạn đã thiết kế xong giao diện cho ứng dụng, khai báo các biến cho xử lý sự kiện và có được giá trị của biến `accessCode`, bạn hãy tiếp tục với việc viết lệnh `Select Case` như được thể hiện trong phần sau đây. Lệnh này quyết định quyền truy cập của người dùng dựa trên mã được nhập vào.

Thêm lệnh Select Case vào ứng dụng

1. ***Thêm lệnh Select Case vào enterButton_Click.*** Thêm dòng 12 trên Hình 12.8 vào xử lý sự kiện `enterButton_Click` và nhấn *Enter*. Từ khóa `End Select` (dòng 14) được tự động thêm vào bên dưới dòng `Select Case` mà bạn vừa thêm vào. Dòng 12 là dòng đầu tiên của lệnh `Select Case` có chứa biểu thức điều khiển `accessCode` - mã truy cập do người dùng nhập vào. Nhớ lại rằng biểu thức này (giá trị `accessCode`) được so sánh liên tục với mỗi mệnh đề `Case` cho đến khi gặp mệnh đề có biểu thức phù hợp hoặc gặp lệnh `End Select`. Nếu có một mệnh đề `Case` đúng thì phần thân của mệnh đề `Case` sẽ được thực thi và ứng dụng tiếp tục thực thi lệnh đầu tiên sau lệnh `End Select`.

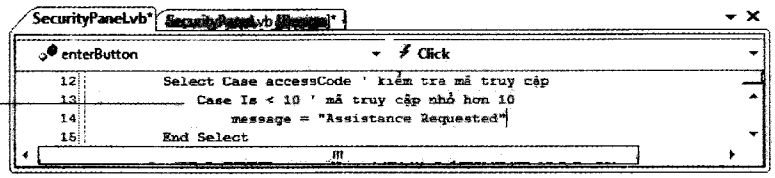
Tạo lệnh `Select Case`



Hình 12.8 Lệnh `Select Case`.

2. ***Thêm mệnh đề Case vào lệnh Select Case.*** Thêm các dòng 13-14 trên Hình 12.9 vào lệnh `Select Case`. Mệnh đề `Case` đầu tiên kiểm tra `accessCode` có nhỏ hơn 10 hay không. Từ khóa **`Is`** đi trước toán tử quan hệ hay toán tử so sánh bằng được dùng để so sánh biểu thức điều khiển với giá trị bên phải của toán tử. Trong trường hợp này, nếu giá trị trong `accessCode` nhỏ hơn 10, mã của phần thân của lệnh `Case` được thực thi và `message` được gán văn bản `"Assistance Requested"`, văn bản này sẽ được hiển thị sau khi phần thân của lệnh `Select Case` hoàn thành.

Từ khóa Is được dùng cho các phép so sánh quan hệ và so sánh bằng

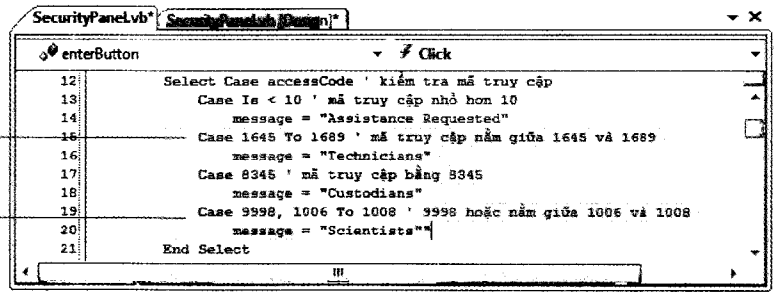


Hình 12.9 Mệnh đề Case đầu tiên được thêm vào lệnh Select Case.

3. ***Chỉ ra mệnh đề Case cho các mã truy cập còn lại.*** Thêm các dòng 15-20 trên Hình 12.10 vào lệnh Select Case.

Từ khóa To dùng để chỉ ra dãy các giá trị cần kiểm tra

Dấu phẩy để ngăn cách các biểu thức trong mệnh đề Case



Hình 12.10 Các mệnh đề Case được chỉ ra cho các mã truy cập còn lại.

Mệnh đề Case ở dòng 15-16 quyết định giá trị của accessCode có nằm trong dãy 1645-1689 hay không. Từ khóa To được sử dụng để xác định dãy này. Nếu người dùng nhập vào một mã truy cập nằm trong dãy thì phần thân của lệnh Case đặt giá trị message là "Technicians".

Mệnh đề Case tiếp theo (dòng 17-18) kiểm tra một số cụ thể. Nếu accessCode bằng với giá trị 8345, thì lệnh trong Case được thực thi. Việc chỉ ra một giá trị đơn trong một Case là phổ biến.

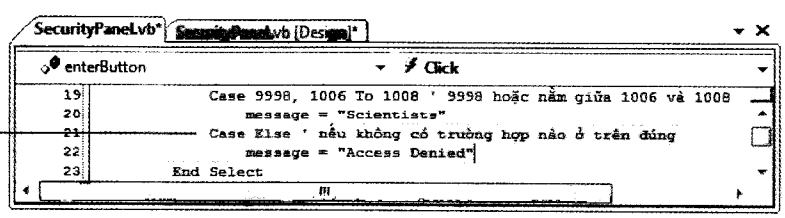
Mệnh đề Case tiếp theo (các dòng 19-20) quyết định accessCode có phải là 9998 hoặc là một số ở trong dãy 1006-1008 hay không. Hãy lưu ý rằng khi nhiều giá trị hay nhiều dãy giá trị được cung cấp trong một mệnh đề Case thì chúng được ngăn cách bởi các dấu phẩy.

Lỗi lập trình thường gặp

Nếu giá trị bên trái của từ khóa To trong mệnh đề Case lớn hơn giá trị ở bên phải thì mệnh đề Case này sẽ bị bỏ qua khi ứng dụng thực thi, điều này gây ra một lỗi lôgíc tiềm tàng. Trình biên dịch sẽ đưa ra cảnh báo đối với trường hợp này.

4. ***Thêm Case Else vào lệnh Select Case.*** Thêm dòng 21-22 trên Hình 12.11 vào lệnh Select Case. Các dòng này chứa mệnh đề Case Else là một tùy chọn và sẽ được thực thi khi biểu thức điều khiển không đúng với tất cả các mệnh đề Case trước nó. Nếu được sử dụng, mệnh đề Case Else phải nằm sau tất cả các mệnh đề Case khác. Trong ứng dụng của bạn, phần thân của mệnh đề Case Else thiết lập giá trị của biến message là "Access Denied". Từ khóa bắt buộc End Select (dòng 23 trong Hình 12.11) kết thúc lệnh Select Case.

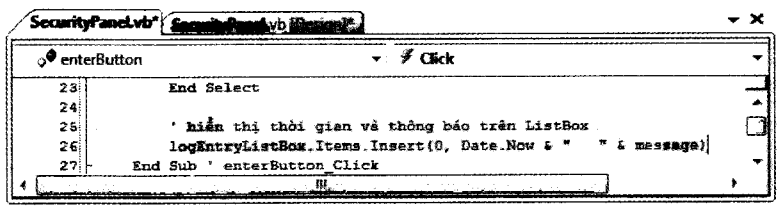
Mệnh đề Case Else được thực thi khi không có mệnh đề Case nào đúng



Hình 12.11 Mệnh đề Case Else của lệnh Select Case.

5. ***Hiển thị kết quả trong ListBox.*** Thêm các dòng 25-26 trên Hình 12.12 vào sau lệnh Select Case. Lệnh trên dòng 26 sử dụng phương thức **Insert** của thuộc tính Items để chèn thêm một phần tử vào ListBox tại một vị trí cụ thể. Phương thức Insert cần hai đối số - đối số đầu xác định vị trí chèn phần tử (bắt đầu từ 0 cho phần tử đầu tiên), đối số

thứ hai xác định phần tử cần được chèn. Dòng 26 chèn một `String` vào `logEntryListBox` bao gồm thông tin về ngày tháng và thời gian hệ thống hiện thời, theo sau là ba ký tự trống và giá trị gán cho `message`. Phần tử này là một `String` được chèn vào vị trí đầu tiên (0 như đối số đầu tiên đã chỉ ra) của `ListBox`, vì vậy các thông báo được hiển thị theo trật tự thời gian ngược. Phần đầu đối số thứ hai của phương thức `Insert` chứa biểu thức `Date.Now`. .NET Framework Class Library cung cấp kiểu **`Date`** có thể được sử dụng để lưu và hiển thị thông tin ngày tháng và thời gian. Thuộc tính **`Now`** trả về ngày tháng và thời gian của hệ thống trong thời điểm hiện tại. Giá trị này được truyền như một phần của `String` là đối số thứ hai vào phương thức `Insert` (dòng 26) làm cho giá trị này được đổi kiểu và hiển thị dưới dạng `String`. [*Lưu ý:* Định dạng được dùng để hiển thị `Date` dưới dạng `String` và giá trị của `String` này sẽ thay đổi phụ thuộc vào thiết lập vị trí cho máy tính của bạn.] Bạn sẽ tìm hiểu cách ngày tháng được lưu bằng kiểu `Date` trong Chương 14.



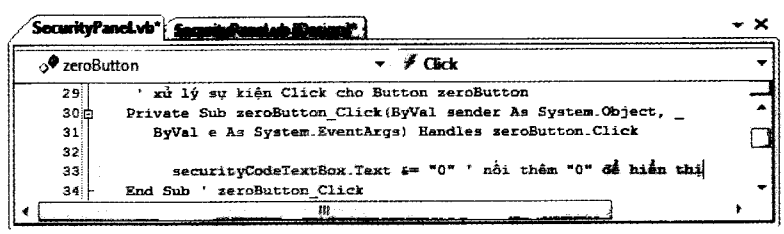
Hình 12.12 Cập nhật `ListBox` của ứng dụng **Security Panel**.

6. ***Lưu project.*** Chọn **File > Save All** để lưu mã đã được sửa đổi.

Bây giờ, khi đã định nghĩa xong xử lý sự kiện `enterButton_Click`, bạn tiếp tục tập trung vào các `Button` số. Bạn tạo các xử lý sự kiện cho mỗi `Button` số và `Button` **`C`**.

Lập trình các xử lý sự kiện còn lại

1. ***Tạo xử lý sự kiện `zeroButton_Click`.*** Trong chế độ **Design**, nhấn đúp `Button 0 (zeroButton)` để tạo xử lý sự kiện `zeroButton_Click`.
2. ***Viết mã cho xử lý sự kiện `zeroButton_Click`.*** Thêm dòng 33 trên Hình 12.13 vào xử lý sự kiện. Dòng 33 nối `String` `"0"` vào sau giá trị thuộc tính `Text` của `securityCodeTextBox`. Bạn làm việc này để thêm giá trị của `Button` số vào mã truy cập trên `TextBox`.



Hình 12.13 Xử lý sự kiện `zeroButton_Click`.

3. ***Định nghĩa xử lý sự kiện của các `Button` khác.*** Lặp lại các *Bước 1-2* cho các `Button` số còn lại (từ **1** đến **9**). Hãy chú ý thay thế các giá trị số của `Button` vào giá trị tương ứng trong dấu ngoặc kép (chẳng hạn `oneButton_Click` thì thiết lập giá trị `securityCodeTextBox` là `&= "1"`). Hình 12.14 thể hiện các xử lý sự kiện cho các `Button` `oneButton` và `twoButton`.

```
SecurityPanel.vb*  SecurityPanel.vb [Design]*
zeroButton                                  Click
    ' xử lý sự kiện Click của Button oneButton
Private Sub oneButton_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles oneButton.Click

      securityCodeTextBox.Text &= "1" ' nối thêm "1" để hiển thị
End Sub ' oneButton_Click

    ' xử lý sự kiện Click của Button twoButton
Private Sub twoButton_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles twoButton.Click

      securityCodeTextBox.Text &= "2" 'nối thêm "2" để hiển thị
End Sub ' twoButton_Click
```

Hình 12.14 Các xử lý sự kiện `oneButton_Click` và `twoButton_Click`.

4. ***Định nghĩa xử lý sự kiện `clearButton_Click`.*** Nhấn đúp vào Button **C** và thêm dòng 103 như trên Hình 12.15 để xóa nội dung TextBox **Security Code:**.

```
SecurityPanel.vb*
clearButton                                 Click
    ' xử lý sự kiện Click của Button clearButton
Private Sub clearButton_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles clearButton.Click

      securityCodeTextBox.Clear() ' xóa nội dung trên TextBox
End Sub ' clearButton_Click
```

Hình 12.15 Mã định nghĩa xử lý sự kiện `clearButton_Click`.

5. ***Chạy ứng dụng.*** Chọn **Debug > Start Debugging** để chạy ứng dụng. Nhập một vài mã an ninh và kiểm tra kết quả hiển thị trên ListBox.
6. ***Đóng ứng dụng.*** Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.
7. ***Đóng IDE.*** Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

Hình 12.16 giới thiệu mã nguồn của ứng dụng **Security Panel**. Các dòng mã chứa các khái niệm lập trình mới mà bạn vừa học trong chương này được in đậm. Hãy đọc mã này cẩn thận để chắc chắn rằng bạn đã thêm mã cho tất cả các xử lý sự kiện chính xác.

```
1  Public Class SecurityPanelForm
2     ' xử lý sự kiện Click cho Button enterButton
3     Private Sub enterButton_Click(ByVal sender As System.Object, _
4        ByVal e As System.EventArgs) Handles enterButton.Click
5
6        Dim accessCode As Integer ' lưu mã truy cập được nhập vào
7        Dim message As String ' hiển thị trạng thái truy cập
8
9        accessCode = Val(securityCodeTextBox.Text)
10       securityCodeTextBox.Clear()
11
12       Select Case accessCode ' kiểm tra mã truy cập
13          Case Is < 10 ' mã truy cập nhỏ hơn 10
14             message = "Assistance Requested"
15          Case 1645 To 1689 ' mã truy cập nằm giữa 1645 và 1689
16             message = "Technicians"
17          Case 8345 ' mã truy cập bằng 8345
```

Sử dụng `Select Case` để xác định mức truy cập của người dùng

Hình 12.16 Ứng dụng **Security Panel** (Phần 1/3).

Lấy giá trị thời gian hiện thời sử dụng `Date.Now`

Thêm văn bản vào `TextBox` đã bị vô hiệu hóa để chỉ có thể hiển thị nội dung

```
            message = "Custodians"
         Case 9998, 1006 To 1008 ' 9998 hoặc nằm giữa 1006 và 1008
            message = "Scientists"
         Case Else ' nếu không có trường hợp nào ở trên đúng
            message = "Access Denied"
      End Select

      ' hiển thị thời gian và thông báo trên ListBox
      logEntryListBox.Items.Insert(0, Date.Now & " " & message)
   End Sub ' enterButton_Click

   ' xử lý sự kiện Click cho Button zeroButton
   Private Sub zeroButton_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles zeroButton.Click

      securityCodeTextBox.Text &= "0" ' nối thêm "0" để hiển thị
   End Sub ' zeroButton_Click

   ' xử lý sự kiện Click của Button oneButton
   Private Sub oneButton_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles oneButton.Click

      securityCodeTextBox.Text &= "1" ' nối thêm "1" để hiển thị
   End Sub ' oneButton_Click

   ' xử lý sự kiện Click của Button twoButton
   Private Sub twoButton_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles twoButton.Click

      securityCodeTextBox.Text &= "2" ' nối thêm "2" để hiển thị
   End Sub ' twoButton_Click

   ' xử lý sự kiện Click của Button threeButton
   Private Sub threeButton_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles threeButton.Click

      securityCodeTextBox.Text &= "3" ' nối thêm "3" để hiển thị
   End Sub ' threeButton_Click

   ' xử lý sự kiện Click của Button fourButton
   Private Sub fourButton_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles fourButton.Click

      securityCodeTextBox.Text &= "4" ' nối thêm "4" để hiển thị
   End Sub ' fourButton_Click

   ' xử lý sự kiện Click của Button fiveButton
   Private Sub fiveButton_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles fiveButton.Click

      securityCodeTextBox.Text &= "5" ' nối thêm "5" để hiển thị
   End Sub ' fiveButton_Click

   ' xử lý sự kiện Click của Button sixButton
   Private Sub sixButton_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles sixButton.Click

      securityCodeTextBox.Text &= "6" ' nối thêm "6" để hiển thị
   End Sub ' sixButton_Click

   ' xử lý sự kiện Click của Button sevenButton
   Private Sub sevenButton_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles sevenButton.Click

      securityCodeTextBox.Text &= "7" ' nối thêm "7" để hiển thị
   End Sub ' sevenButton_Click

   ' xử lý sự kiện Click của Button eightButton
   Private Sub eightButton_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles eightButton.Click

      securityCodeTextBox.Text &= "8" ' nối thêm "8" để hiển thị
```

Hình 12.16 Ứng dụng **Security Panel** (Phần 2/3).

```
90      End Sub ' eightButton_Click
91
92      ' xử lý sự kiện Click của Button nineButton
93      Private Sub nineButton_Click(ByVal sender As System.Object, _
94      ByVal e As System.EventArgs) Handles nineButton.Click
95
96      securityCodeTextBox.Text &= "9" ' nối thêm "9" để hiển thị
97      End Sub ' nineButton_Click
98
99      ' xử lý sự kiện Click của Button clearButton
100     Private Sub clearButton_Click(ByVal sender As System.Object, _
101        ByVal e As System.EventArgs) Handles clearButton.Click
102
103        securityCodeTextBox.Clear() ' xóa nội dung trên TextBox
104     End Sub ' clearButton_Click
105 End Class ' SecurityPanelForm
```

Hình 12.16 Ứng dụng **Security Panel** (Phần 3/3).

TỰ ÔN TẬP

1. Một mệnh đề `Case` được thực thi đối với tất cả các giá trị lớn hơn một giá trị cụ thể thì phải có từ khóa_____ nằm trước toán tử >
   a) `Select` b) `Is`
   c) `Case` d) `All`
2. Sử dụng một _____ để ngăn cách các điều kiện trong mệnh đề `Case`.
   a) dấu chấm b) dấu sao
   c) dấu phẩy d) dấu hai chấm

Đáp án: 1) b. 2) c.

12.4 Tổng kết

Trong chương này, bạn đã tìm hiểu cách sử dụng lệnh đa lựa chọn `Select Case` và những điểm tương đồng của lệnh này với lệnh `If...Then...Else`. Bạn đã nghiên cứu biểu đồ hoạt động minh họa luồng điều khiển trong lệnh `Select Case`.

Sau đó bạn áp dụng những điều đã học được để xây dựng ứng dụng **Security Panel**. Bạn thiết lập thuộc tính `PasswordChar` của `TextBox` là * trong cửa sổ **Properties** để che giấu nội dung hiển thị trên `TextBox`. Bạn sử dụng lệnh `Select Case` để xác định người dùng có nhập mã an ninh đúng hay không. Bạn cũng đã định nghĩa mệnh đề `Case Else` để thực thi khi người dùng cung cấp một mã an ninh không hợp lệ. Bạn đã sử dụng `Date.Now` để lấy thời gian hiện thời của hệ thống. Cuối cùng, bạn đã học được cách chèn một phần tử vào một vị trí cụ thể của `ListBox` bằng cách sử dụng phương thức `Item.Insert`.

Trong chương tới bạn sẽ tìm hiểu cách xây dựng ứng dụng từ nhiều đoạn mã nhỏ, dễ quản lý và có thể tái sử dụng được gọi là thủ tục. Bạn sẽ sử dụng khả năng này để nâng cấp ví dụ mà bạn đã tạo ra ở chương trước trong cuốn sách này.

TỔNG KẾT KỸ NĂNG

Tạo lệnh `Select Case`

- Sử dụng từ khóa `Select Case` theo sau là biểu thức điều khiển.
- Sử dụng từ khóa `Case` theo sau là biểu thức để so sánh với biểu thức điều khiển.
- Định nghĩa các lệnh thực thi nếu biểu thức của mệnh đề `Case` đúng với biểu thức điều khiển.
- Sử dụng từ khóa `Case Else` theo sau là các lệnh được thực thi nếu biểu thức điều khiển không đúng với bất kỳ một mệnh đề `Case` nào. Mệnh đề `Case Else` (nếu được sử dụng) phải là mệnh đề `Case` cuối cùng.
- Sử dụng từ khóa `End Select` để kết thúc lệnh `Select Case`.

Che dấu nội dung do người dùng nhập vào TextBox

- Thiết lập giá trị thuộc tính PasswordChar của TextBox bằng ký tự mong muốn, thường là dấu sao (*), để che giấu nội dung do người dùng nhập vào.
- Lấy giá trị thực tế do người dùng nhập vào qua thuộc tính Text.

Lấy giá trị thời gian hiện thời

- Sử dụng thuộc tính Now của kiểu Date. Giá trị này khi được chuyển thành kiểu String thì sẽ hiển thị thời gian hiện thời với định dạng 12/31/2002 11:59:59 P.M. (hoặc có thể khác đi một chút tùy thuộc vào thiết lập vị trí của máy tính bạn).

Chèn phần tử vào đầu ListBox

- Sử dụng phương thức Insert của thuộc tính Items với đối số đầu tiên là 0.

THUẬT NGỮ

biểu thức điều khiển - Là biểu thức có giá trị được so sánh liên tục với các mệnh đề Case cho đến khi tìm được mệnh đề case phù hợp hoặc gặp lệnh End Select. Biểu thức điều khiển còn được gọi là biểu thức kiểm tra.

che giấu - Ẩn nội dung văn bản, chẳng hạn như mật khẩu hoặc những thông tin nhạy cảm khác không muốn để người khác nhìn thấy khi các thông tin này được nhập vào. Việc ngụy trang được thực hiện bằng cách sử dụng thuộc tính PasswordChar của TextBox mà bạn muốn che giấu dữ liệu. Khi đó dữ liệu thực sự được nhập được lưu vào thuộc tính Text của TextBox.

danh sách biểu thức - Các biểu thức được ngăn cách bởi dấu phẩy. Được sử dụng trong mệnh đề Case của lệnh Select Case khi một mệnh đề bất kỳ được thực thi dựa trên nhiều hơn một điều kiện.

kiểu Date - Là kiểu có các thuộc tính có thể được sử dụng để lưu trữ và hiển thị thông tin về thời gian.

ký tự ngụy trang - Dùng để hiển thị thay thế cho các ký tự trong TextBox khi muốn che giấu dữ liệu của TextBox để đảm bảo tính riêng tư.

lệnh đa lựa chọn - Thực hiện một trong nhiều hành động (hoặc nhiều chuỗi hành động) phụ thuộc vào giá trị của biểu thức điều khiển.

lệnh Select Case - Là lệnh đa lựa chọn được sử dụng để đưa ra quyết định bằng cách so sánh một biểu thức với một loạt các điều kiện. Sau đó thuật toán sẽ thực hiện các hành động khác nhau dựa trên các giá trị đó.

mệnh đề Case - Là mệnh đề có phần thân sẽ được thực thi nếu biểu thức điều khiển của Select Case đúng với biểu thức của mệnh đề Case này.

mệnh đề Case Else - Là mệnh đề không bắt buộc có phần thân sẽ được thực thi nếu như biểu thức điều khiển của lệnh Select Case không đúng với biểu thức của bất cứ một mệnh đề Case nào.

phương thức Insert của thuộc tính Items của ListBox - Chèn một phần tử vào một ListBox tại vị trí được chỉ định bởi đối số thứ nhất của phương thức này.

thuộc tính Enabled của TextBox - Xác định nội dung TextBox có bị thay đổi bởi người dùng hay không.

thuộc tính Now của kiểu Date - Trả lại ngày tháng và thời gian hệ thống hiện thời.

thuộc tính PasswordChar của TextBox - Chỉ ra ký tự ngụy trang cho TextBox.

từ khóa End Select - Là từ khóa kết thúc lệnh Select Case.

từ khóa Is - Từ khóa đi trước toán tử so sánh, dùng để so sánh biểu thức điều khiển của lệnh Select Case với một giá trị.

HƯỚNG DẪN THIẾT KẾ GIAO DIỆN

Thiết kế tổng thể

- Nếu giao diện mô phỏng thiết bị thực tế, thiết kế của nó phải bắt chước hình thức vật lý của thiết bị đó.

TextBox

- Che giấu thông tin mật khẩu hoặc những thông tin nhạy cảm khác trong các TextBox.

ĐIỀU KHIỂN, SỰ KIỆN, THUỘC TÍNH & PHƯƠNG THỨC

TextBox. TextBox Điều khiển này cho phép người dùng nhập dữ liệu từ bàn phím.

- ***Trên giao diện khi ứng dụng chạy***

- ***Sự kiện***

  TextChanged - Xảy ra khi văn bản trong TextBox thay đổi.

- ***Thuộc tính***

  Enabled - Cho biết người dùng có thể (True) hay không thể (False) nhập dữ liệu vào TextBox.

  Location - Chỉ ra vị trí tương đối của TextBox so với góc trên bên trái của Form.

  Multiline - Chỉ ra TextBox có khả năng hiển thị văn bản nhiều dòng hay không.

  Name - Chỉ ra tên dùng để truy cập TextBox trong khi lập trình. Tên nên được thêm hậu tố TextBox vào phía sau.

  PasswordChar - Chỉ ra ký tự ngụy trang khi hiển thị dữ liệu trong TextBox.

  ReadOnly - Cho biết người dùng có được thay đổi nội dung của TextBox hay không.

  ScrollBars - Chỉ ra TextBox có chứa các thanh cuộn hay không.

  Size - Chỉ ra chiều rộng và chiều cao (tính bằng pixel) của TextBox.

  Text - Chỉ ra văn bản được hiển thị trên TextBox.

  TextAlign - Chỉ ra cách văn bản trong TextBox được căn lề.

  Width - Chỉ ra chiều rộng (tính bằng pixel) của TextBox.

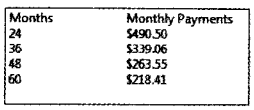
- ***Phương thức***

  Clear - Xóa nội dung văn bản của TextBox gọi hàm này.

  Focus - Chuyển focus của ứng dụng vào TextBox gọi hàm này.

ListBox ListBox Điều khiển này cho phép người dùng nhập dữ liệu từ bàn phím.

- ***Trên giao diện khi ứng dụng chạy***



- ***Thuộc tính***

  Items - Trả về đối tượng chứa các phần tử được hiển thị trong ListBox.

  Items.Count - Trả về số lượng các phần tử trong ListBox.

  Items.Item - Trả về giá trị của phần tử ở vị trí được chỉ ra của ListBox.

  Location - Chỉ ra vị trí tương đối của ListBox so với góc trên bên trái của Form.

  Name - Chỉ ra tên được sử dụng để truy cập ListBox trong khi lập trình. Tên nên được thêm hậu tố ListBox vào phía sau.

`Size` - Chỉ ra chiều rộng và chiều cao (tính bằng pixel) của `ListBox`.

- ***Phương thức***

`Items.Add` - Thêm một phần tử vào cuối danh sách các phần tử.

`Items.Clear` - Xóa tất cả các phần tử của `ListBox`.

`Items.Insert` - Chèn thêm phần tử được chỉ định bởi đối số thứ hai vào danh sách các phần tử tại vị trí được chỉ ra ở đối số thứ nhất (phần tử 0 là phần tử đầu tiên.

CÂU HỎI TRẮC NGHIỆM

12.1 Từ khóa_____báo hiệu sự kết thúc của lệnh `Select Case`.

a) `End Case`
b) `End Select`
c) `End Select Case`
d) `Case End`

12.2 Biểu thức_____trả lại thời gian và ngày tháng hệ thống hiện thời

a) `Date.DateTime`
b) `Date.SystemDateTime`
c) `Date.Now`
d) `Date.SystemTimeDate`

12.3 Có thể che giấu thông tin nhập vào `TextBox` bằng cách thiết lập giá trị thuộc tính_____ của `TextBox` bằng ký tự sẽ được hiển thị thay thế cho mọi ký tự do người dùng nhập vào.

a) `PrivateChar`
b) `Mask`
c) `MaskingChar`
d) `PasswordChar`

12.4 Lỗi nào sau đây là lỗi cú pháp?

a) Có hai mệnh đề `Case` trong cùng một lệnh `Select Case`
b) Có một mệnh đề `Case` mà trong đó giá trị bên trái từ khóa `To` lớn hơn giá trị bên phải
c) Mệnh đề `Case Else` đứng trước mệnh đề `Case` trong lệnh `Select Case`
d) Sử dụng từ khóa `Is` trong lệnh `Select Case`

12.5 Từ khóa dùng để chỉ ra một khoảng giá trị trong lệnh mệnh đề `Case`.

a) `Also`
b) `Between`
c) `To`
d) `From`

12.6_____ngăn cách các giá trị được kiểm tra trong mệnh đề `Case`.

a) Dấu phẩy
b) Dấu gạch dưới
c) Từ khóa `Also`
d) Dấu chấm phẩy

12.7 Phương thức_____chèn một giá trị vào vị trí cụ thể của `ListBox`.

a) `Append`
b) `Items.Insert`
c) `InsertArt`
d) `Items.Add`

12.8 Nếu giá trị bên trái của từ khóa `To` trong lệnh `Case` lớn hơn giá trị bên phải thì_____.

a) xảy ra lỗi cú pháp
b) phần thân của lệnh `Case` được thực thi
c) phần thân lệnh lệnh `Case` không bao giờ được thực thi
d) lệnh này sẽ gây ra lỗi khi chạy chương trình

12.9 Biểu thức đi sau từ khóa `Select Case` được gọi là_____.

a) điều kiện bảo vệ
b) biểu thức điều khiển
c) biểu thức lựa chọn
d) biểu thức mệnh đề `Case`

12.10 Để ngăn chặn người dùng thay đổi văn bản trong `TextBox`, hãy đặt thuộc tính_____của nó là `False`.

a) `Enabled`
b) `Text`
c) `TextChange`
d) `Editable`

BÀI TẬP

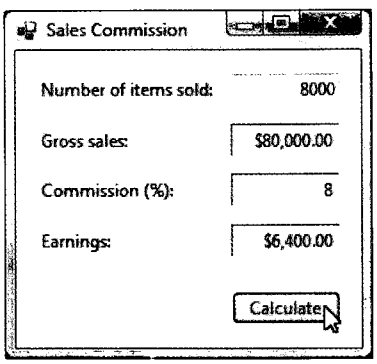
12.11 (***Ứng dụng Sales Commission Calculator***) Phát triển một ứng dụng tính tiền hoa hồng của nhân viên bán hàng nhận được dựa trên số mặt hàng đã bán được (Hình12.17). Giả sử tất cả các hàng hóa đều có một giá cố định là 10 $ cho mỗi mặt hàng. Sử dụng lệnh `Select Case` để tính tiền hoa hồng như sau:

Bán được ít hơn 10 mặt hàng thì hoa hồng nhận được là 1%

Bán được từ 10 đến 40 mặt hàng thì hoa hồng nhận được là 2%

Bán được từ 41 đến 100 mặt hàng thì hoa hồng nhận được là 4%

Bán được hơn 100 mặt hàng thì hoa hồng nhận được là 8%



Hình 12.17 Giao diện ứng dụng **Sales Commission Calculator**.

a) ***Copy template của ứng dụng vào thư mục làm việc.*** Copy thư mục `C:\Examples\Tutorial12\Exercises\SalesCommissionCalculator` vào thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng.*** Nhấn đúp vào file `SalesCommissionCalculator.sln` trong thư mục `SalesCommissionCalculator` để mở ứng dụng.

c) ***Định nghĩa một xử lý sự kiện cho sự kiện `Click` của `Button`.*** Tạo một xử lý sự kiện `Click` cho `Button` **Calculate**.

d) ***Hiển thị doanh số của nhân viên bán hàng.*** Trong xử lý sự kiện mới, hãy nhân số lượng mặt hàng bán được với 10 và hiển thị doanh số bán hàng (gross sales) dưới dạng tiền tệ.

e) ***Tính toán phần trăm hoa hồng của nhân viên bán hàng.*** Sử dụng lệnh `Select Case` để tính phần trăm hoa hồng (commission) cho mỗi nhân viên bán hàng từ số lượng mặt hàng mà nhân viên bán được. Tỷ lệ phần trăm này được áp dụng cho tất cả các mặt hàng mà nhân viên đã bán.

f) ***Hiển thị thu nhập của nhân viên bán hàng.*** Nhân tổng doanh số với phần trăm hoa hồng được xác định ở bước trước để tính thu nhập của nhân viên bán hàng (Earnings). Hãy nhớ chia cho 100 để có được số thập phân tương đương.

g) ***Chạy ứng dụng.*** Chọn **Debug > Start Debugging** để chạy ứng dụng. Nhập giá trị số lượng mặt hàng bán được (Number of items sold) và nhấn `Button` **Calculate**. Kiểm tra xem tổng doanh số, tỷ lệ phần trăm hoa hồng và thu nhập được hiển thị có đúng không dựa trên tỷ lệ hoa hồng tương ứng với số lượng mặt hàng bán được cho ở trên.

h) ***Đóng ứng dụng.*** Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

i) ***Đóng IDE.*** Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

12.12 (***Ứng dụng Cash Register***) Sử dụng bàn phím số từ ứng dụng **Security Panel** để xây dựng ứng dụng **Cash Register** (Hình 12.18). Ngoài các số, máy tính tiền còn

bao gồm một `Button` dấu chấm thập phân. Bên cạnh thao tác số, ta còn cần các `Button` **Enter, Delete, Clear** và **Total**. Để đơn giản hóa, tiền thuế được tính dựa trên tổng số tiền của các mặt hàng. Bạn hãy sử dụng lệnh `Select Case` để tính tiền thuế và sau đó cộng thêm tiền thuế vào tổng số tiền của các mặt hàng để tính tổng số tiền phải trả. Ứng dụng hiển thị tổng số tiền của các mặt hàng (Subtotal), tiền thuế (Tax) và tổng số tiền khách hàng phải trả (Total). Sử dụng tỷ lệ phần trăm của tiền thuế dựa trên tổng số tiền các mặt hàng sau đây:

Số tiền dưới 100 USD thì thuế là 10% (0,10)

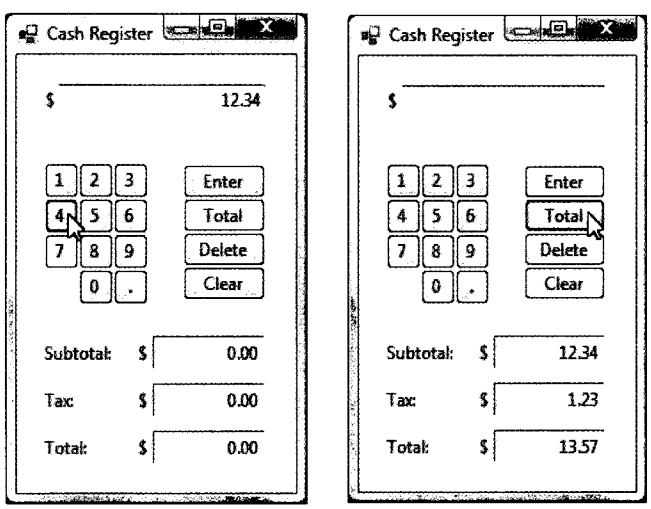
Số tiền lớn hơn hoặc bằng 100 USD và nhỏ hơn hoặc bằng 500 USD thì thuế là 7.5% (0,075)

Số tiền trên 500 USD thì thuế là 5% (0,05)

a) ***Copy template của ứng dụng vào thư mục làm việc.*** Copy thư mục `C:\Examples\Tutorial12\Exercises\CashRegister` vào thư mục `C:\SimplyVB2008` của bạn.

b) ***Mở file template của ứng dụng.*** Nhấn đúp vào file `CashRegister.sln` trong thư mục `CashRegister` để mở ứng dụng.

c) ***Định nghĩa các xử lý sự kiện cho các `Button` số và dấu chấm thập phân trên bàn phím.*** Tạo các xử lý sự kiện cho sự kiện `Click` của mỗi `Button`. Thiết lập để mỗi xử lý sự kiện nối giá trị thích hợp vào sau giá trị trên `TextBox` ở đầu `Form`.



Hình 12.18 Ứng dụng **Cash Register.**

d) ***Định nghĩa xử lý sự kiện cho sự kiện `Click` của `Button` Enter.*** Tạo xử lý sự kiện cho sự kiện `Click` của `enterButton`. Lập trình để xử lý sự kiện này cộng số tiền hiện thời vào tổng số tiền các mặt hàng, hiển thị tổng số tiền mới và xóa giá tiền mặt hàng hiện thời được nhập vào.

e) ***Định nghĩa xử lý sự kiện cho sự kiện `Click` của `Button` Total.*** Tạo xử lý sự kiện cho sự kiện `Click` của `totalButton`. Lập trình để xử lý sự kiện này sử dụng tổng số tiền các mặt hàng để tính số tiền thuế và tổng số tiền khách hàng phải trả. Hiển thị thông tin này ở các `Label` thích hợp.

f) ***Định nghĩa xử lý sự kiện cho sự kiện `Click` của `Button` Clear.*** Tạo xử lý sự kiện cho sự kiện `Click` của `clearButton`. Lập trình để xử lý sự kiện này xóa thông tin người dùng nhập vào và hiển thị giá trị `0,00` cho tổng số tiền của các mặt hàng, tiền thuế và tổng số tiền phải trả.

g) ***Định nghĩa xử lý sự kiện cho sự kiện `Click` của `Button` Delete.*** Tạo xử lý sự kiện cho sự kiện `Click` của `deleteButton`. Lập trình để xử lý sự kiện này chỉ xóa dữ liệu trên `TextBox`.

h) ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng. Sử dụng bàn phím để nhập các số tiền khác nhau, nhấn `Button` **Enter** sau khi số tiền của mặt hàng được nhập. Xác nhận rằng trường **Subtotal:** cập nhật lại giá trị sau mỗi lần nhập. Sau khi đã nhập một vài giá tiền của các mặt hàng, nhấn `Button` **Total** và xác nhận tiền thuế và tổng số tiền phải trả tương ứng được hiển thị. Nhập thêm một vài giá trị và nhấn `Button` **Delete** để xóa nội dung trên `TextBox` vừa nhập. Nhấn `Button` **Clear** để xóa tất cả các giá trị đầu ra.

i) ***Đóng ứng dụng.*** Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

j) ***Đóng IDE***. Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

12.13 (***Ứng dụng Income Tax Calculator***) Tạo một ứng dụng tính tiền thuế thu nhập một người phải trả dựa trên tiền lương. Thuế thu nhập được tính tùy thuộc vào các mức thu nhập khác nhau. Chẳng hạn, nếu người dùng có thu nhập 25.000 USD, họ phải trả 10% cho 7.825 USD và 15% cho 17.175 USD còn lại. Ứng dụng này được thể hiện như trên Hình 12.19. Sử dụng khoảng thu nhập và tỷ lệ thuế tương ứng sau đây:

Không vượt quá 7.825 USD thì thuế thu nhập là 10%

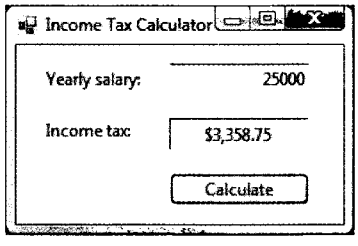
7.826 USD - 31.850 thì thuế thu nhập là 15%

31.851 USD - 77.100 thì thuế thu nhập là 25%

77.101 USD - 160.850 thì thuế thu nhập là 28%

160.850 USD - 349.700 thì thuế thu nhập là 33%

Trên 349.700 USD thì thuế thu nhập là 35%



Hình 12.19 Giao diện của **Income Tax Calculator.**

a) ***Copy template vào thư mục làm việc***. Copy thư mục `C:\Examples\Tutorial12\Exercises\IncomeTaxCalculator` vào thư mục `C:\SimplyVB2008` của bạn.

b) ***Mở file template của ứng dụng.*** Nhấn đúp vào file `IncomTaxCalculator.sln` trong thư mục `IncomeTaxCalculator` để mở ứng dụng.

c) ***Định nghĩa xử lý sự kiện cho sự kiện `Click` của `Button` Calculate.*** Sử dụng giao diện thiết kế để tạo xử lý sự kiện cho sự kiện `Click` của `calculateButton`. Lập trình cho xử lý sự kiện này sử dụng lệnh `Select Case` để xác định tỷ lệ phần trăm thuế thu nhập của người dùng, sau đó hiển thị kết quả trên `Label` đầu ra.

d) ***Sử dụng sự kiện `TextChanged` để xóa đầu ra.*** Nhấn đúp vào `TextBox` **Yearly salary:** để tạo sự kiện `TextChanged`. Xóa nội dung trên `Label` **Income tax:** khi người dùng thay đổi giá trị ở `TextBox`.

e) ***Chạy ứng dụng.*** Chọn **Debug > Start Debugging** để chạy ứng dụng. Nhập tiền lương và nhấn `Button` **Calculate**. Xác nhận rằng tiền thuế thu nhập được hiển thị đúng dựa trên các mức thu nhập được liệt kê ở phần mô tả của bài tập.

f) ***Đóng ứng dụng.*** Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

g) ***Đóng IDE***. Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

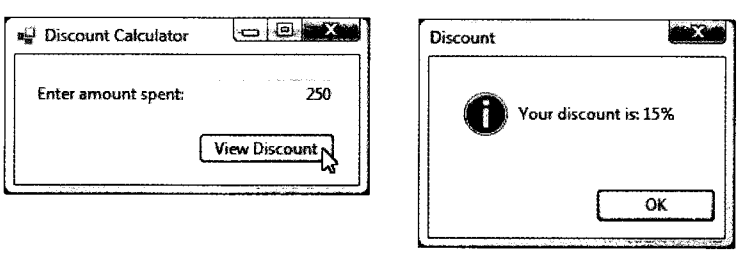
Đoạn mã này làm gì? ▶ **12.14** Hãy cho biết đầu ra của mã sau đây. Giả sử `donationButton` là một `Button`, `donationTextBox` là một `TextBox` và `massageLabel` là một `Label` hiển thị kết quả đầu ra.

```
1  Private Sub donationButton_Click(ByVal sender As  _
2      System.Object, ByVal e As System.EventArgs) _
3      Handles doantionButton.Click
4
5      Select Case Val(donationTextBox.Text)
6          Case 0
7              messageLabel.Text = "Please consider donating to our cause."
8          Case 1 To 100
9              messageLabel.Text = "Thank you for your donation."
10         Case Is > 100
11             messageLabel.Text = "Thank you very much for your donation!"
12         Case Else
13             messageLabel.Text = "Please enter a valid amount."
14     End Select
15 End Sub
```

Đoạn mã này có gì sai? ▶ **12.15** Lệnh `Select Case` này cần xác định một số `Integer` là chẵn hay lẻ. Hãy tìm lỗi trong mã sau đây:

```
1  Select Case value Mod 2
2      Case 0
3          outputLabel.Text = "Odd Integer"
4      Case 1
5          outputLabel.Text = "Even Integer"
6  End Select
```

Sử dụng trình gỡ lỗi ▶ **12.16** (***Ứng dụng Discount Calculator***) Copy thư mục `C:\Examples\Tutorial07\DiscountCalculator` vào thư mục `C:\SimplyVB2008`. Ứng dụng **Discount Calculator** xác định khoản chiết khấu người dùng nhận được dựa trên tổng số tiền đã mua hàng. Chiết khấu 15% được nhận đối với số tiền mua lớn hơn 200 USD, 10% đối với số tiền mua từ 150-200 USD, 5% đối với số tiền mua từ 100 USD đến 149 USD và 2% với số tiền mua từ 50-99 USD. Khi chạy thử ứng dụng, bạn nhận thấy ứng dụng không tính chiết khấu đúng cho mỗi giá trị tiền mua nhập vào. Sử dụng trình gỡ rối để tìm và sửa các lỗi lôgíc cho ứng dụng. Hình 12.20 hiển thị kết quả đầu ra đúng của ứng dụng.



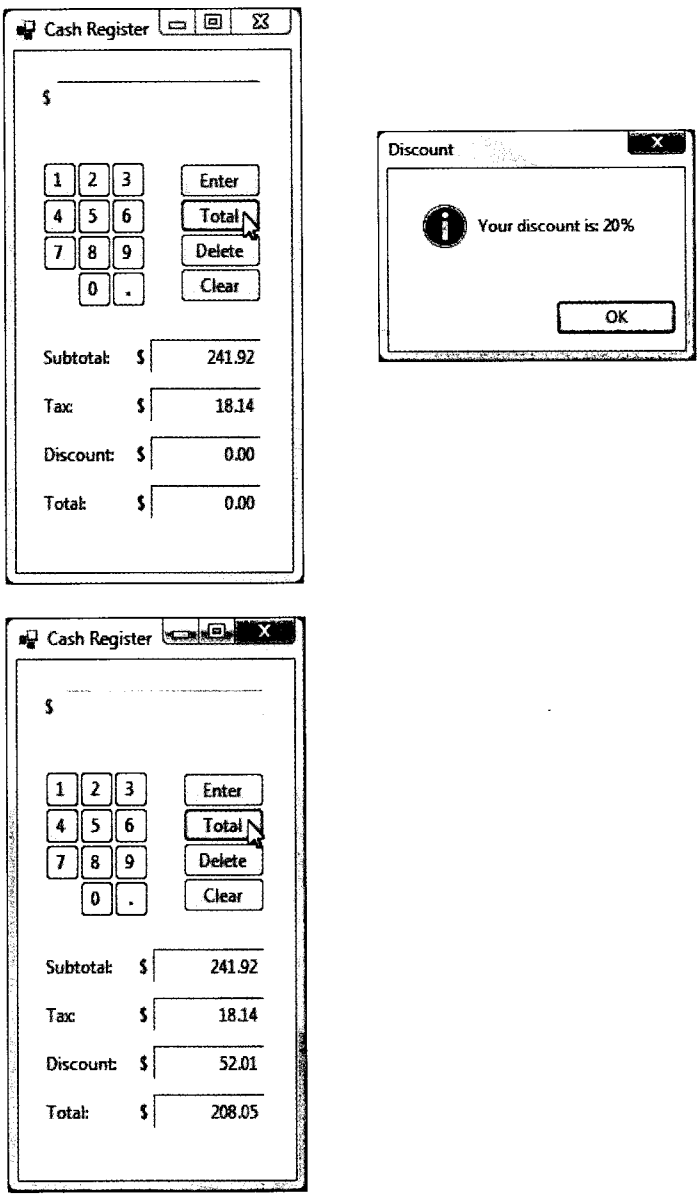
Hình 12.20 Đầu ra đúng cho ứng dụng **Discount Calculator.**

Bài tập nâng cao ▶ **12.17** ***(Nâng cấp ứng dụng Cash Register)***. Một cửa hàng đang chứa một lượng hàng hóa. Sửa đổi mã của ứng dụng **Cash Register** (Bài tập 12.12) để tính toán chiết khấu dựa trên tổng số tiền bán được (bao gồm cả tiền thuế). Chiết khấu được hiển thị trong hộp thoại hoặc trên `Label` **Discount:**. Hình 12.21 hiển thị giao diện của ứng dụng **Cash Register** sau khi sửa đổi. Cửa hàng đưa ra chiết khấu sau đây dựa trên tổng số tiền nhân viên bán được:

Dưới 200 USD thì chiết khấu là 10%

200 - 500 USD thì chiết khấu là 20%

Trên 500 USD thì chiết khấu là 30%



Hình 12.21 Giao diện của ứng dụng **Cash Register** sau khi đã sửa đổi.

CHƯƠNG 13

Nâng cấp ứng dụng Wage Calculator

Giới thiệu thủ tục Function và thủ tục Sub

Mục tiêu

Trong chương này, bạn sẽ tìm hiểu để:

- Xây dựng ứng dụng theo kiểu module hóa từ thủ tục.
- Sử dụng thủ tục có sẵn.
- Phân biệt thủ tục **Function** và thủ tục **Sub** và xác định khi nào nên dùng thủ tục gì.
- Tạo thủ tục **Function** và thủ tục **Sub**.
- Sử dụng các điều khiển gỡ lỗi trên thanh công cụ **Standard**.

Nội dung chính

Hầu hết các ứng dụng phần mềm trên thực tế đều lớn hơn nhiều so với các ứng dụng được giới thiệu trong những chương trước của cuốn sách này. Kinh nghiệm cho thấy rằng cách tốt nhất để phát triển và duy trì ứng dụng lớn là xây dựng chúng từ những đơn vị nhỏ hơn và dễ quản lý hơn. Kỹ thuật này có tên là chia để trị (**divide and conquer**) - hay còn được gọi là thành phần hóa (**componentization**). Những đơn vị nhỏ dễ quản lý này là các thành phần chương trình được gọi là các thủ tục (**procedure**). Các thành phần này đơn giản hóa việc thiết kế, phát triển và bảo trì ứng dụng lớn. Trong chương này, bạn sẽ tìm hiểu cách tạo hai dạng thủ tục là thủ tục `Function` và thủ tục Sub.

13.1 Chạy thử ứng dụng Wage Calculator đã nâng cấp

Sử dụng thủ tục để nâng cấp ứng dụng **Wage Calculator** mà bạn đã xây dựng trong Chương 7. Ứng dụng này phải đáp ứng những yêu cầu sau:

Yêu cầu đối với ứng dụng

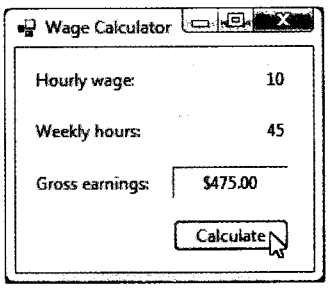
Nhớ lại vấn đề đã trình bày trong Chương 7: Một công ty trả lương tính tổng thu nhập trước thuế mỗi tuần cho nhân viên. Tổng thu nhập hàng tuần của nhân viên được tính dựa trên số giờ làm việc và tiền thù lao mỗi giờ. Cần tạo ra một ứng dụng với đầu vào là các thông tin này và tính tổng thu nhập (chưa chịu thuế) hàng tuần cho mỗi nhân viên. Trong ứng dụng này, giả sử rằng số giờ làm việc tiêu chuẩn trong một tuần là 40. Tiền lương tính cho 40 giờ làm hoặc ít hơn được tính bằng cách nhân lương mỗi giờ với số giờ làm việc trong mỗi tuần. Khi số giờ làm việc trong một tuần vượt quá 40 giờ thì số giờ vượt quá này được xem là thời gian làm thêm giờ và sẽ có thù lao gấp rưỡi so với giờ tiêu chuẩn. Lương cho giờ làm việc thêm đó được tính bằng cách nhân tiền thù lao mỗi giờ với 1,5 và nhân kết quả này với số giờ làm thêm. Tổng số tiền làm thêm giờ cộng với tổng số tiền làm trong 40 giờ làm việc tiêu chuẩn là tổng thu nhập trong tuần của nhân viên.

Ứng dụng hoàn thiện có chức năng giống như ứng dụng ở Chương 7 nhưng sử dụng thủ tục để tổ chức mã tốt hơn. Ứng dụng này tính toán thu nhập dựa trên tiền thù lao mỗi giờ của nhân viên và số giờ làm trong tuần. Nếu nhân viên làm việc 40 giờ hoặc ít hơn sẽ nhận được tiền lương là tiền thù lao mỗi giờ nhân với số giờ làm việc, còn nếu nhân viên làm việc nhiều hơn tiêu chuẩn là 40 giờ một tuần thì tính toán sẽ khác đi một chút. Trong chương này, bạn sẽ tìm hiểu các thủ tục thực thi việc tính toán dựa

trên giá trị được nhập vào có thể thay đổi trong mỗi lần ứng dụng chạy. Sau đó, bạn tìm hiểu những tính năng bổ sung của Visual Basic cần thiết để xây dựng cho mình một phiên bản của ứng dụng này.

Chạy thử ứng dụng Wage Calculator đã nâng cấp

1. ***Mở ứng dụng đã hoàn thiện.*** Mở thư mục `C:\Examples\Tutorial13\CompletedApplication\WageCalculator2` để tìm ứng dụng **Wage Calculator.** Nhấn đúp vào `fileWageCalculator2.sln` để mở ứng dụng trong Visual Basic IDE.
2. ***Chạy ứng dụng Wage Calculator.*** Chọn **Debug > Start Debugging** để chạy ứng dụng.
3. ***Nhập tiền thù lao mỗi giờ của nhân viên***. Nhập 10 vào `TextBox` **Hourly wage:** (Hình 13.1).



Chú thích hình
- ***Hourly wage:*** *Tiền thù lao mỗi giờ*
- ***Weekly hours:*** *Số giờ làm trong tuần*
- ***Gross earnings:*** *Tổng thu nhập trước thuế*

Hình 13.1 Ứng dụng **Wage Calculator** đang chạy.

4. ***Nhập số giờ làm việc của nhân viên. Nhập*** 45 vào `TextBox` **Weekly hours:.**
5. ***Tính toán lương. Nhấn*** `Button` **Calculate.** Kết quả (`$475.00`) hiển thị trong `Label` **Gross earnings:.**
6. ***Đóng ứng dụng.*** Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.
7. ***Đóng IDE.*** Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

13.2 Lớp và thủ tục

Chìa khóa để tạo ra các ứng dụng lớn là chia chúng thành những phần nhỏ hơn. Trong lập trình hướng đối tượng, những phần này là các lớp, sau đó các lớp lại được chia nhỏ hơn thành các **phương thức** (**method**). Trong lập trình Visual Basic, các phương thức được xây dựng bằng cách viết các thủ tục nên trong cuốn sách này, chúng tôi gọi thủ tục thay vì phương thức.

Lập trình viên thường kết hợp lớp và phương thức do người dùng định nghĩa (**programmer-defined**) và mã có sẵn (hay được định nghĩa sẵn) trong .NET Framework Class Library. Sử dụng mã có sẵn sẽ tiết kiệm thời gian, sức lực và tiền bạc. Tái sử dụng mã (**reusing code**) nâng cao hiệu quả làm việc của những người phát triển ứng dụng. Hình 13.2 giải thích một số phương thức Visual Basic có sẵn.

Bạn đã sử dụng một số lớp và phương thức có sẵn trong .NET Framework Class Library trong những chương trước. Chẳng hạn như tất cả điều khiển mà bạn đã dùng trong ứng dụng đều là những lớp được định nghĩa trong .NET Framework Class Library. Bạn cũng đã dùng phương thức trong các lớp của .NET Framework Class Library, chẳng hạn như phương thức `Format` của lớp `String` để hiển thị đầu ra của ứng dụng dưới định dạng đúng. Nếu không có phương thức `String.Format` thì bạn sẽ phải tự mình viết mã cho chức năng này - một nhiệm vụ cần

nhiều dòng lệnh và kỹ thuật lập trình phức tạp. Bạn sẽ tìm hiểu thêm nhiều hơn về các lớp và phương thức trong thư viện .NET Framework Class Library trong cuốn sách này.

Thủ tục	Mô tả	Ví dụ
`Math.Max(x,y)`	Trả về giá trị lớn hơn của x và y	`Math.Max(2.3, 12.7)` là `12.7` `Math.Max(-2.3, -12.7)` là `-2.3`
`Math.Min(x,y)`	Trả về giá trị nhỏ hơn của x và y	`Math.Min(2.3, 12.7)` là `2.3` `Math.Min(-2.3, -12.7)` là `-12.7`
`Math.Sqrt(x)`	Trả về căn bậc hai của x	`Math.Sqrt(9)` là `3.0` `Math.Sqrt(2)`là `1.4142135623731`
`Pmt(x, y, z)`	Tính toán khoản tiền vay phải trả hàng kỳ trong đó x là tỷ lệ lãi suất, y là thời hạn vay và z là khoản tiền vay ban đầu.	`Pmt(0.05, 12, -4000)` là `451.301640083261`

`Val(x)`	Trả về giá trị số của biến x	`Val("5")` là `5`
		`Val("5a8")` là `5`
		`Val("a5")` là `0`

`String.Format` (*formatString, listOfArguments*)	Trả về một `String` đã được định dạng trong đó tham số thứ nhất *formatString* chỉ ra định dạng và tham số thứ hai *listOfArguments* chỉ ra các giá trị cần định dạng	`String.Format("{0:C}", 1.23` là `"$1.23"`

Hình 13.2 Một số phương thức được định nghĩa sẵn trong Visual Basic.

Dù vậy, .NET Framework Class Library không thể cung cấp mọi tính năng mà bạn cần, vì vậy Visual Basic cho phép bạn tạo ra thủ tục do người dùng định nghĩa để đáp ứng những yêu cầu đặc biệt cho ứng dụng của bạn. Có hai loại thủ tục: **thủ tục `Function`** và **thủ tục `Sub`**. Trong mục tiếp theo, bạn sẽ tìm hiểu về thủ tục `Function`, còn thủ tục `Sub` bạn sẽ được tìm hiểu trong Mục 13.4. Trong suốt chương này, thuật ngữ "thủ tục" là để chỉ cả thủ tục `Function` và thủ tục `Sub`, trừ khi có lưu ý riêng.

TỰ ÔN TẬP

1. _____cung cấp cho lập trình viên các lớp có sẵn thực thi các nhiệm vụ thông thường.
   a) Framework Class Library
   b) Framework Code Library
   b) Từ khóa `preExisting`
   d) Từ khóa `Library`
2. Lập trình viên thường sử dụng_____.
   a) thủ tục do người dùng định nghĩa
   b) thủ tục có sẵn
   c) cả thủ tục do người dùng định nghĩa và thủ tục có sẵn
   d) không phải thủ tục do người dùng định nghĩa cũng không phải thủ tục có sẵn

Đáp án: 1) a. 2) c.

13.3 Thủ tục Function

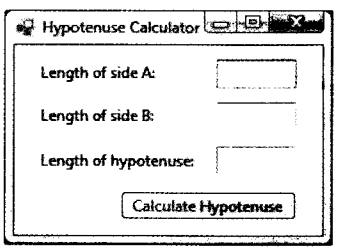
Mẹo thiết kế phần mềm

Sử dụng thủ tục để ứng dụng rõ ràng và dễ tổ chức hơn. Điều này không chỉ giúp người khác hiểu ứng dụng mà còn giúp bạn dễ dàng phát triển, kiểm thử và gỡ lỗi cho ứng dụng.

Những ứng dụng được giới thiệu ở phần trước của cuốn sách gọi các phương thức .NET Framework Class Library (như String.Format) để hỗ trợ ứng dụng hoàn thành tác vụ. Bây giờ bạn sẽ học cách tự định nghĩa thủ tục. Trước tiên bạn học cách tạo thủ tục trong hai ứng dụng nhỏ trước khi nâng cấp ứng dụng **Wage Calculator**. Ứng dụng thứ nhất sử dụng định lý Pytago để tính độ dài cạnh huyền của một tam giác vuông. Ứng dụng thứ hai tìm ra số lớn nhất trong ba số. Ta hãy nhớ lại định lý Pytago. Tam giác vuông (tam giác có một góc bằng 90 độ) luôn luôn thỏa mãn điều kiện sau đây: tổng bình phương hai cạnh bên bằng bình phương cạnh huyền. Trong ứng dụng này, sử dụng độ dài của hai cạnh bên A và B để tính độ dài cạnh huyền. Làm theo các bước ở phần tiếp theo để tạo ứng dụng.

Tạo ứng dụng Hypotenuse Calculator

1. ***Copy template của ứng dụng vào thư mục làm việc.*** Copy thư mục `C:\Examples\Tutorial13\TemplateApplication\HypotenuseCalculator` vào thư mục `C:\SimplyVB2008`.
2. ***Mở file template của ứng dụng Hypotenuse Calculator.*** Nhấn đúp vào file `HypotenuseCalculator.sln` trong thư mục `HypotenuseCalculator` để mở ứng dụng trong Visual Basic IDE. Khi bạn mở `Form` trong chế độ **Design**, bạn sẽ thấy giao diện như trên Hình 13.3. Khi ứng dụng này chạy, người dùng nhập độ dài hai cạnh bên của tam giác vuông vào các `TextBox` **Length of side A:** và **Length of side B:**, sau đó nhấn `Button` **Calculate Hypotenuse**. Ứng dụng sẽ tính toán độ dài của cạnh huyền và hiển thị kết quả trong `Label` **Length of hypotenuse:**.



Chú thích hình
- ***Length of side A:*** *Độ dài cạnh A*
- ***Length of side B:*** *Độ dài cạnh B*
- ***Length of hypotenuse:*** *Độ dài cạnh huyền*

Hình 13.3 Giao diện **Hypotenuse Calculator.**

3. ***Xem mã ứng dụng template.*** Chuyển sang chế độ **Code** và xem xét mã được cung cấp sẵn trong template như trên Hình 13.4.

Độ dài cạnh A, B và cạnh huyền

Bình phương độ dài cạnh A, B và cạnh huyền

Hộp thoại hiển thị báo lỗi nếu giá trị âm (hoặc giá trị không phải số) được nhập vào

```
(General)                                   (Declarations)
Public Class HypotenuseCalculatorForm
    ' xử lý sự kiện Click của Button Calculate
    Private Sub calculateButton_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles calculateButton.Click

        Dim sideA As Double ' độ dài của cạnh A
        Dim sideB As Double ' độ dài của cạnh B
        Dim hypotenuse As Double ' độ dài của cạnh huyền
        Dim squareSideA As Double ' bình phương độ dài của cạnh A
        Dim squareSideB As Double ' bình phương độ dài của cạnh B
        Dim squareHypotenuse As Double ' bình phương độ dài cạnh huyền

        sideA = Val(sideATextBox.Text)
        sideB = Val(sideBTextBox.Text)

        ' nếu dữ liệu đầu vào nhỏ hơn hoặc bằng 0
        If (sideA <= 0) OrElse (sideB <= 0) Then
            MessageBox.Show("You must enter a non-negative number", _
               "Invalid Input Entered", MessageBoxButtons.OK, _
               MessageBoxIcon.Error)
        Else

        End If
    End Sub ' calculateButton_Click
End Class ' HypotenuseCalculatorForm
```

Hình 13.4 Mã nguồn trong template của **Hypotenuse Calculator.**

Thói quen lập trình tốt

Thêm chú thích ở đầu thủ tục để chỉ ra mục đích của thủ tục đó.

Thói quen lập trình tốt

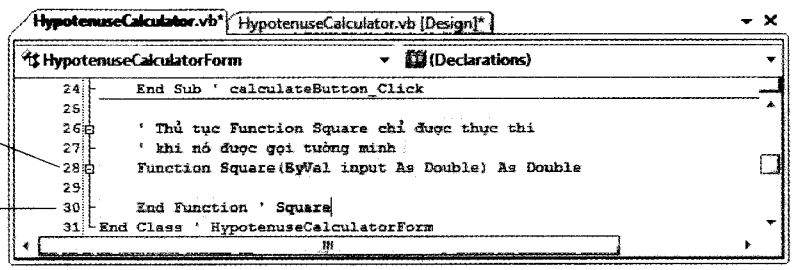
Thêm chú thích ở cuối thủ tục để chỉ ra thủ tục nào đã kết thúc.

Ta được cung cấp xử lý sự kiện chưa hoàn chỉnh cho `Button` **Calculate Hypotenuse**. Xử lý sự kiện này bao gồm sáu phép khai báo (dòng 6-11). Các biến `sideA` và `sideB` chứa độ dài cạnh A và cạnh B do người dùng nhập vào. Biến `hypotenuse` chứa độ dài cạnh huyền sẽ được tính toán sau đây. Biến `squareSideA` chứa bình phương độ dài cạnh A. Tương tự, các biến `squareSideB` và `squareHypotenuse` lưu các giá trị tương ứng là bình phương độ dài cạnh B và bình phương độ dài cạnh huyền. Dòng 13-14 lưu giá trị do người dùng nhập vào cho cạnh A và cạnh B. Dòng 17-23 là một lệnh `If...Then...Else`. Phần thân lệnh `If` hiển thị hộp thoại nếu giá trị âm (hoặc bằng 0) được nhập vào cho cạnh A hay cạnh B hoặc cả hai. Phần thân của lệnh `Else` (chỉ thực thi nếu bạn nhập giá trị lớn hơn không) sẽ tính toán độ dài cạnh huyền.

4. ***Tạo một thủ tục Function rỗng.*** Thêm dòng 26-28 trên Hình 13.5 vào sau xử lý sự kiện `calculateButton_Click`, rồi nhấn *Enter*. Từ khóa `End Function` sẽ được IDE tự động thêm vào (dòng 30). Bạn sẽ tìm hiểu các từ khóa này ngay sau đây. Hãy thêm chú thích ở dòng 30 để chỉ ra thủ tục kết thúc.

Tiêu đề thủ tục **Function**

Từ khóa **End Function** đánh dấu sự kết thúc của thủ tục **Function**



Hình 13.5 Thủ tục `Function Square`.

Thói quen lập trình tốt

Chọn tên thủ tục và tên tham số có nghĩa để ứng dụng dễ đọc hơn và giảm bớt những chú thích dài dòng.

5. ***Tìm hiểu thủ tục Function.*** Thủ tục bắt đầu ở dòng 28 (Hình 13.5) với từ khóa **`Function`**, tiếp theo là **tên thủ tục** (trong trường hợp này là `Square`). Tên thủ tục có thể là bất cứ định danh hợp lệ nào. Sau tên thủ tục là cặp ngoặc đơn chứa khai báo tham số.

Khai báo trong ngoặc đơn được gọi là **danh sách tham số (parameter list)**, ở đó các biến (được gọi là tham số (parameter)) sẽ được khai báo. Các tham số cho phép thủ tục nhận dữ liệu để thực hiện tác vụ của nó. Mặc dù danh sách tham số này chỉ chứa một phép khai báo, nhưng danh sách tham số trên thực tế có thể không chứa khai báo nào hoặc nhiều hơn một khai báo được ngăn cách bởi dấu phẩy. Danh sách tham số khai báo tên và kiểu của mỗi tham số. Lưu ý rằng khai báo tham số ở đây dùng từ khóa `ByVal` thay cho từ khóa `Dim`. Ta sẽ tìm hiểu từ khóa `ByVal` ngay sau đây. Các tham số chỉ được dùng trong thân thủ tục `Function`.

Mẹo thiết kế phần mềm

Để tăng khả năng tái sử dụng, mỗi thủ tục nên thực hiện một nhiệm vụ đơn lẻ. Tên thủ tục nên được định nghĩa rõ ràng và phải súc tích, nêu bật được nhiệm vụ của thủ tục.

Thủ tục `Function` trả về giá trị sau khi thực hiện nhiệm vụ. Để chỉ ra kiểu của giá trị trả về, tiếp sau danh sách tham số là từ khóa `As` cùng với kiểu dữ liệu (trong ví dụ này là `Double`). Kiểu dữ liệu đứng sau `As` được gọi là **kiểu trả về (return type)**, chỉ ra kiểu của giá trị được `Function` trả về (trong ví dụ này là `Double`). Dòng đầu tiên của thủ tục (bao gồm từ khóa `Function`, tên thủ tục, danh sách tham số và kiểu trả về) được gọi là **tiêu đề thủ tục (procedure header)**. Tiêu đề thủ tục cho `Square` khai báo một tham số đầu vào, có kiểu `Double` và kiểu trả về là `Double`.

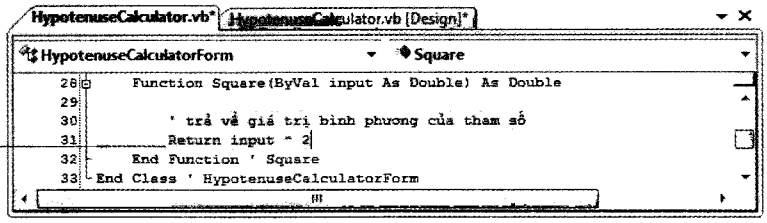
Thủ tục `Function` trên Hình 13.5 kết thúc ở dòng 30 bằng từ khóa **`End Function`**. Khai báo và lệnh xuất hiện sau tiêu đề thủ tục và trước từ khóa `End Function` tạo nên phần thân của thủ tục. Thân thủ tục chứa mã thực hiện các hành động, về cơ bản là thao tác trên tham số của danh sách tham số và trả về kết quả. Trong bước tiếp theo, bạn sẽ thêm các dòng lệnh vào thân thủ tục `Square`. Tiêu đề thủ tục, thân và từ khóa `End Function` tạo thành **định nghĩa thủ tục (procedure definition)**.

Thói quen lập trình tốt

Tên của thủ tục nên là động từ và có chữ cái đầu viết hoa. Mỗi từ tiếp theo trong tên cũng nên viết hoa chữ cái đầu. Quy ước đặt tên như vậy được gọi là quy ước Pascal.

6. ***Thêm mã vào thân của thủ tục Function.*** Thủ tục `Function` sẽ thực hiện chức năng tính bình phương của ứng dụng. Thêm các dòng 30-31 trên Hình 13.6 vào thân thủ tục `Square`.

Tính bình phương bằng toán tử ^



Hình 13.6 Định nghĩa thủ tục `Square`.

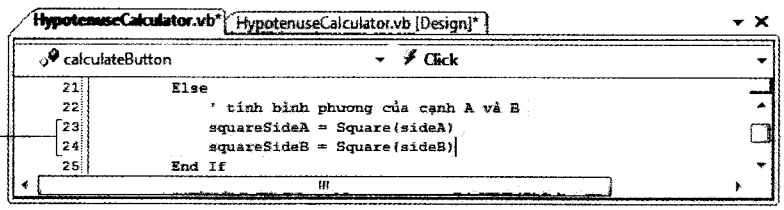
Thói quen lập trình tốt

Để một dòng trắng giữa các định nghĩa thủ tục để ứng dụng dễ đọc hơn.

Dòng 31 sử dụng toán tử ^ để tính bình phương đầu vào - tham số của thủ tục - và sử dụng lệnh **`Return`** để trả về giá trị. Lệnh này bắt đầu với từ khóa **`Return`**, tiếp theo là một biểu thức. Lệnh `Return` trả về kết quả của biểu thức đi sau từ khóa `Return`, trong trường hợp này là `input ^2` và kết thúc thực thi thủ tục. Giá trị này sẽ được trả về tại vị trí mà thủ tục được gọi. Bạn sẽ viết mã để gọi thủ tục trong bước tiếp theo.

7. ***Gọi thủ tục Square.*** Bây giờ thủ tục đã được tạo, bạn có thể gọi thủ tục này từ xử lý sự kiện. Thêm dòng 22-24 trên Hình 13.7 vào ứng dụng, trong phần `Else` của lệnh `If...Then...Else`. Những dòng này gọi thủ tục `Square` bằng cách sử dụng tên thủ tục, tiếp theo là cặp ngoặc đơn có chứa đối số của thủ tục. Trong trường hợp này, các đối số là các biến `sideA` và `sideB` chứa thông tin do người dùng nhập vào. Trong mỗi lần gọi, giá trị của đối số được truyền vào thủ tục `Square` và được lưu bởi tham số `input`.

Gọi thủ tục **Square**



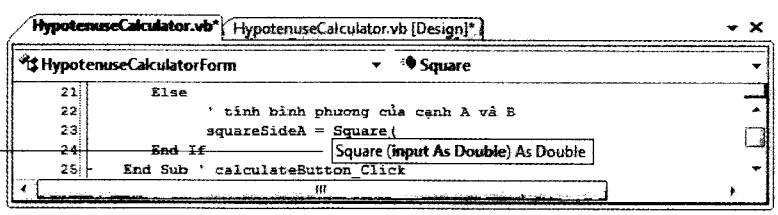
Hình 13.7 Gọi thủ tục `Square`.

Thói quen lập trình tốt

Chọn tên tham số có tính miêu tả để thông tin do tính năng Parameter Info cung cấp có ý nghĩa hơn.

Lưu ý rằng khi gõ dấu mở ngoặc đơn sau tên thủ tục thì Visual Basic IDE sẽ hiển thị một cửa sổ có chứa tên và kiểu đối số của thủ tục (Hình 13.8). Đó là tính năng ***Parameter Info*** của IDE, cung cấp cho bạn thông tin về các thủ tục và đối số của chúng. Tính năng *Parameter Info* hiển thị thông tin cho thủ tục do người dùng định nghĩa cũng như cho phương thức trong thư viện .NET Framework Class Library.

Cửa sổ *Parameter Info*



Hình 13.8 Cửa sổ *Parameter Info*.

Thủ tục được gọi (invoked) - nghĩa là được yêu cầu thực hiện tác vụ cụ thể - bằng **lời gọi thủ tục (procedure call)**. Lời gọi thủ tục chỉ ra tên thủ tục và cung cấp thông tin (**đối số**) mà thủ tục được gọi (**callee**) cần để thực hiện tác vụ. Mỗi đối số được gán cho tham số tương ứng của thủ tục khi thủ

tục được gọi. Số lượng đối số trong lời gọi phải bằng với số tham số có trong định nghĩa. Sau khi hoàn thành tác vụ, thủ tục được gọi trả lại quyền điều khiển cho **thủ tục gọi (caller)**. Chẳng hạn, chúng ta thường gọi thủ tục `Val` như sau:

```
result = Val(inputTextBox.Text)
```

trong đó `Val` là tên của thủ tục và giá trị của `inputTextBox.Text` là đối số truyền vào thủ tục này. `Val` sử dụng đối số để thực hiện tác vụ của mình (trả về giá trị của `inputTextBox.Text` dưới dạng một số).

Khi điều khiển chương trình đến dòng 23 trên Hình 13.7, ứng dụng gọi thủ tục `Function Square`. Tại đây, ứng dụng tạo bản sao cho giá trị của biến `sideA` (giá trị do người dùng nhập vào) và điều khiển chương trình được chuyển đến dòng đầu tiên của thủ tục `Square`.

Từ khóa **`ByVal`** trong khai báo tham số của thủ tục `Square` chỉ ra rằng, bản sao của giá trị đối số (độ dài cạnh A) sẽ được truyền vào thủ tục `Square`. `Square` nhận bản sao giá trị do người dùng nhập vào và lưu chúng trong tham số `input`. Khi gặp lệnh `Return` trong `Square`, giá trị ở bên phải của từ khóa `Return` được trả về dòng 23, nơi `Square` được gọi. Đến đây, thực thi thủ tục hoàn tất và tham số chứa bản sao giá trị bị hủy.

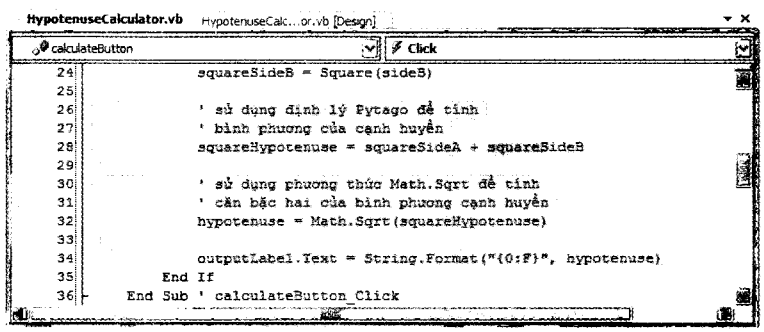
Điều khiển chương trình cũng được chuyển về vị trí này và ứng dụng tiếp tục với việc gán giá trị trả về của `Square` cho biến `squareSideA`. Điều tương tự diễn ra khi điều khiển chương trình gặp lệnh gọi thủ tục `Square` thứ hai ở dòng 24. Trong lần gọi này, giá trị được truyền cho `Square` là giá trị lưu trong biến `sideB` và giá trị trả về được gán cho biến `squareSideB`. Bây giờ chúng ta cần xác định cạnh huyền bằng cách sử dụng các giá trị của biến `squareSideA` và `squareSideB`.

8. ***Gọi phương thức có sẵn trong .NET Framework Class Library***. Thêm dòng 26-34 trên Hình 13.9 vào phần `Else` của lệnh `If...Then...Else` trong ứng dụng. Dòng 28 cộng bình phương của cạnh A và bình phương cạnh B được kết quả là bình phương cạnh huyền và giá trị này được gán cho biến `squareHypotenuse`. Dòng 32 gọi phương thức **`Sqrt`** của lớp `Math` trong .NET Framework Class Library (bằng cách dùng toán tử chấm). Phương thức này tính căn bậc hai bình phương cạnh huyền để tìm ra độ dài cạnh huyền, sau đó định dạng và hiển thị kết quả trên `outputLabel`.

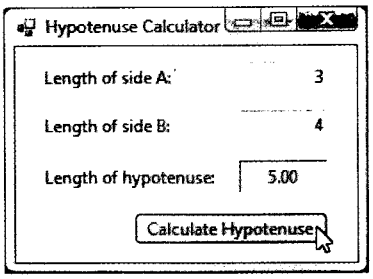
Mẹo tránh lỗi

Thủ tục nhỏ dễ kiểm thử, gỡ lỗi và dễ hiểu hơn so với thủ tục lớn.

9. ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng. Nhập 3 vào `TextBox` **Length of side A:** và 4 vào `TextBox` **Length of side B:**. Nhấn `Button` **Calculate Hypotenuse**. Kết quả được thể hiện trên Hình 13.10.



Hình 13.9 Hoàn chỉnh xử lý sự kiện `calculateButton_Click`.



Hình 13.10 Ứng dụng **Hypotenuse Calculator**.

10. ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.
11. ***Đóng IDE***. Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

Bạn vừa tạo thành công thủ tục `Function` và kiểm thử bằng cách chạy ứng dụng để chắc chắn rằng thủ tục chạy đúng. Thủ tục `Function` này bây giờ có thể được sử dụng trong bất cứ ứng dụng Visual Basic nào để tính toán bình phương của một số `Double`. Việc bạn cần làm chỉ là copy định nghĩa thủ tục vào ứng dụng của bạn. Đây là thí dụ về tái sử dụng mã, giúp bạn xây dựng ứng dụng nhanh hơn.

Như trong ứng dụng **Hypotenuse Calculator**, lời gọi thủ tục `Function` có định dạng như sau

tên (danh sách đối số)

Mẹo thiết kế phần mềm

Tiêu đề thủ tục và lời gọi thủ tục phải nhất quán về số, kiểu và trật tự tham số lượng.

Phải có một đối số trong danh sách đối số của lời gọi thủ tục cho mỗi tham số trong danh sách tham số của tiêu đề thủ tục đó. Các đối số cũng cần phải tương thích với kiểu tham số (có nghĩa là, Visual Basic cần phải có khả năng gán giá trị của một đối số cho tham số tương ứng). Chẳng hạn, một tham số kiểu `Double` có thể nhận giá trị `53547.350009`, `22` hoặc `.03546`, nhưng không thể nhận giá trị "`hello`" vì biến `Double` không thể chứa `String`. Nếu một thủ tục không nhận bất cứ giá trị nào, danh sách tham số là rỗng (có nghĩa là, tên thủ tục đứng trước cặp ngoặc đơn trống). Bạn sẽ tìm hiểu kỹ hơn về tham số thủ tục trong Chương 15.

Như bạn đã thấy trong ví dụ vừa rồi, lệnh

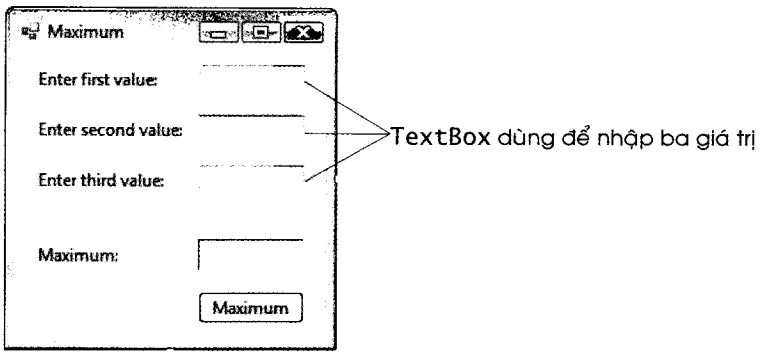
`Return` *biểu thức*

có thể xuất hiện ở bất cứ đâu trong thân thủ tục `Function` và trả về giá trị của *biểu thức* cho thủ tục gọi nó. Nếu cần thiết, Visual Basic sẽ cố gắng chuyển đổi giá trị của *biểu thức* thành kiểu trả về của thủ tục `Function`. `Function` trả về duy nhất một giá trị. Khi lệnh `Return` thực thi, điều khiển ngay lập tức trở về vị trí mà thủ tục `Function` được gọi.

Bây giờ bạn tạo thủ tục `Function` khác. Thủ tục này, là một phần của ứng dụng **`Maximum`**, sẽ trả về số lớn nhất trong ba số được người dùng nhập vào. Trong phần tiếp theo, bạn sẽ tạo ứng dụng **`Maximum`**.

Tạo thủ tục `Function` trả về số lớn nhất trong ba số

1. ***Copy template của ứng dụng vào thư mục làm việc***. Copy thư mục `C:\Examples\Tutorial13\TemplateApplication\Maximum` vào thư mục `C:\SimplyVB2008`.
2. ***Mở file template của ứng dụng `Maximum`***. Nhấn đúp vào file `Maximum.sln` trong thư mục `Maximum` để mở ứng dụng trong Visual Basic IDE. Chuyển sang chế độ **Design** (Hình 13.11).

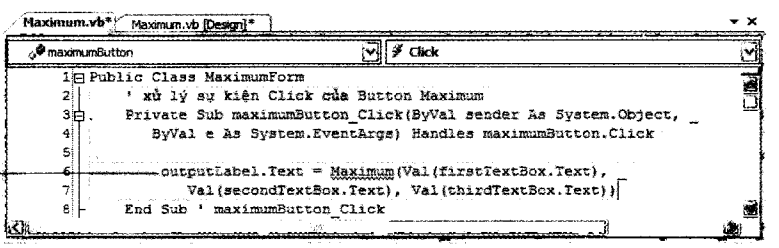


Hình 13.11 Ứng dụng `Maximum` trong chế độ **Design**.

Lỗi lập trình thường gặp

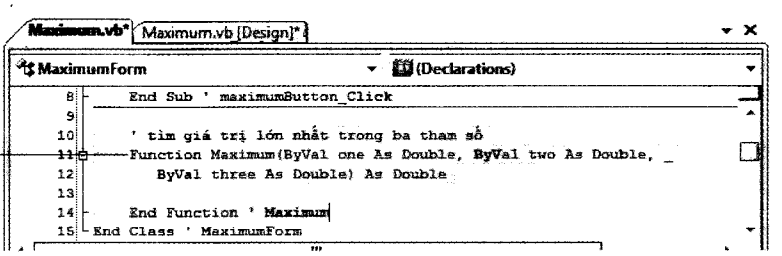
Gọi thủ tục không tồn tại hoặc gọi sai tên thủ tục trong lời gọi thủ tục sẽ gây lỗi biên dịch.

3. ***Tạo xử lý sự kiện cho Button Maximum.*** Nhấn đúp vào `Button` **Maximum** ***để tạo xử lý sự kiện cho sự kiện*** `Click` ***của*** `Button` ***này.*** Thêm chú thích ở dòng 2 trên Hình 13.12 và tách tiêu đề thành hai dòng, như dòng 3-4. Thêm dòng 6-7 vào xử lý sự kiện. Dòng 6-7 gọi thủ tục `Function Maximum` và truyền vào đó ba giá trị do người dùng nhập vào `TextBox` trên ứng dụng. Lưu ý rằng `Maximum` được gạch chân màu xanh, cho biết có lỗi biên dịch. Điều này xảy ra vì thủ tục `Function Maximum` chưa được định nghĩa. Bạn sẽ định nghĩa `Maximum` ở bước sau. Lỗi biên dịch này xuất hiện bất cứ khi nào bạn gọi thủ tục mà IDE Visual Basic không nhận ra. Gõ sai tên của thủ tục cũng gây lỗi biên dịch. Cuối cùng, hãy thêm chú thích sau từ khóa `End Sub` như dòng 8.



Hình 13.12 Gọi thủ tục `Function Maximum`.

4. ***Tạo thủ tục Function Maximum.*** Thêm dòng 10-12 trên Hình 13.13 vào sau xử lý sự kiện `maximumButton_Click`, rồi nhấn *Enter*. Lưu ý rằng từ khóa `EndFunction` được IDE tự động thêm vào. Danh sách tham số chỉ ra rằng giá trị của ba đối số truyền vào `Maximum` sẽ được lưu bởi các tham số `one`, `two` và `three`. Đứng sau danh sách tham số là từ khóa `As` và kiểu trả về `Double`.

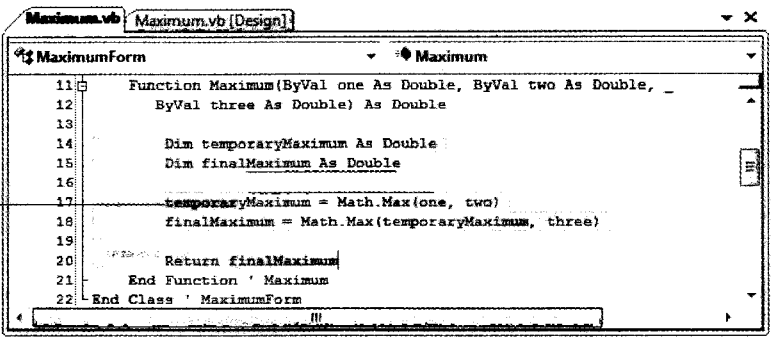


Hình 13.13 Thủ tục `Function Maximum`.

5. ***Thêm chức năng cho thủ tục Function Maximum.*** Thêm dòng 14-20 trên Hình 13.14 vào thân thủ tục `Maximum`. Dòng 14 tạo biến chứa số lớn hơn của hai số đầu được truyền vào thủ tục này. Số lớn hơn này được xác định ở dòng 17 bằng cách sử dụng phương thức **Max** của lớp `Math` trong .NET

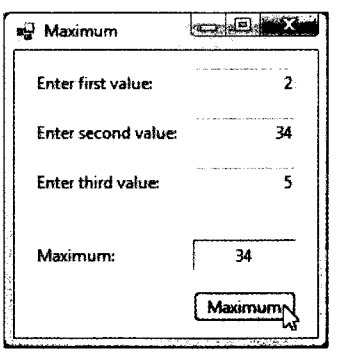
Framework Class Library. Phương thức này nhận vào hai số `Double` và trả về giá trị lớn nhất trong hai đối số đó. Giá trị trả về được gán cho biến `temporaryMaximum` ở dòng 17. Sau đó so sánh giá trị đó với tham số thứ ba của thủ tục `Function Maximum`, tham số `three`, ở dòng 18. Số lớn hơn được xác định ở dòng này, `finalMaximum`, là số lớn nhất trong ba số. Dòng 20 sử dụng lệnh `Return` để trả về giá trị này. Lệnh `Return` kết thúc việc thực thi thủ tục và trả kết quả của `finalMaximum` cho thủ tục gọi. Kết quả được trả về tại vị trí mà thủ tục `Maximum` được gọi (dòng 6 trên Hình 13.12) và được gán cho thuộc tính `Text` của `outputLabel`.

Gọi **Math.Max** để xác định số lớn hơn trong hai số



Hình 13.14 `Math.Max` trả về giá trị lớn hơn trong hai đối số.

6. Chạy ứng dụng. Chọn **Debug > Start Debugging** (Hình 13.15). Nhập giá trị số vào mỗi `TextBox` và nhấn `Button` **Maximum**. Lưu ý rằng số lớn nhất trong ba số được hiển thị ở `Label` đầu ra.



Hình 13.15 Ứng dụng **Maximum** đang chạy.

7. ***Đóng ứng dụng.*** Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.
8. ***Đóng IDE.*** Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

TỰ ÔN TẬP

1. Thủ tục được gọi bởi_____.
   a) thủ tục được gọi
   b) thủ tục gọi
   c) đối số
   d) tham số
2. Lệnh_____trong thủ tục `Function` trả về giá trị cho thủ tục gọi.
   a) `Return`
   b) `Back`
   c) `End`
   d) Các lựa chọn trên đều sai

Đáp án: 1) b. 2) a.

13.4 Sử dụng thủ tục Sub trong ứng dụng Wage Calculator

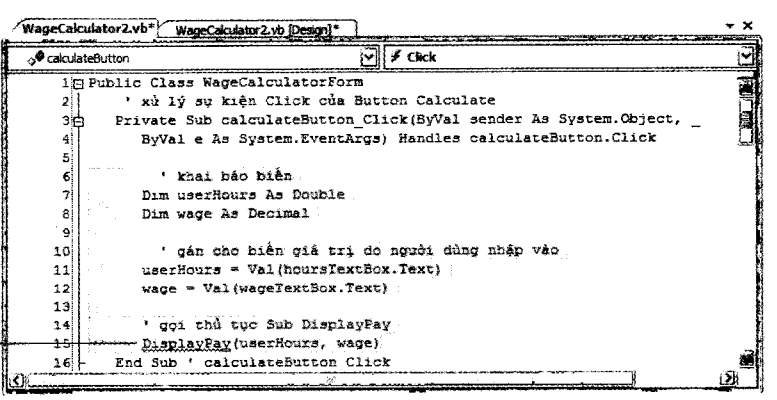
Xử lý sự kiện `Click` của `Button` **Calculate** trong phiên bản ban đầu của ứng dụng **Wage Calculator** (Chương 7) sẽ tính lương và hiển thị kết quả trên `Label` đầu ra. Trong phần tiếp theo, bạn sẽ viết thủ tục Sub `DisplayPay` để thực hiện nhiệm vụ này. Thủ tục Sub tương tự với thủ tục `Function`, chỉ có một điểm khác biệt quan trọng là thủ tục Sub không trả về giá trị cho thủ tục gọi nó. Khi người dùng nhấn vào `Button` **Calculate**, xử lý sự kiện `calculateButton_Click` sẽ gọi thủ tục Sub `DisplayPay`.

Tạo thủ tục Sub trong ứng dụng Wage Calculator

1. ***Copy template của ứng dụng vào thư mục làm việc.*** Copy thư mục `C:\Examples\Tutorial13\TemplateApplication\WageCalculator2` vào thư mục `C:\SimplyVB2008`.
2. ***Mở file template của ứng dụng Wage Calculator.*** Nhấn đúp vào file `WageCalculator2.sln` trong thư mục `WageCalculator2` để mở ứng dụng trong IDE Visual Basic.
3. ***Tạo xử lý sự kiện `calculateButton_Click`.*** Xem `Form` ứng dụng trong chế độ **Design**. Nhấn đúp vào `Button` **Calculate** để tạo xử lý sự kiện `Click`.
4. ***Thêm chức năng cho `calculateButton_Click`.*** Thêm dòng 6-15 trên Hình 13.16 vào xử lý sự kiện rỗng. Đoạn mã này gọi thủ tục `DisplayPay` để tính và hiển thị lương. Dòng 11-12 lấy giá trị người dùng nhập vào từ `TextBox` và gán chúng cho biến được khai báo ở dòng 7-8. Dòng 15 gọi thủ tục `DisplayPay`, bạn sẽ định nghĩa thủ tục này ngay sau đây. Lời gọi thủ tục này cần có hai đối số: số giờ làm việc (`userHours`) và tiền thu lao mỗi giờ (`wage`). Lưu ý rằng lời gọi thủ tục `DisplayPay` này được gạch chân màu xanh vì thủ tục này chưa được định nghĩa nên IDE thông báo lỗi biên dịch.
5. ***Tạo thủ tục Sub.*** Sau xử lý sự kiện `calculateButton_Click`, hãy thêm thủ tục Sub `DisplayPay` vào ứng dụng (dòng 18-39 trên Hình 13.17).

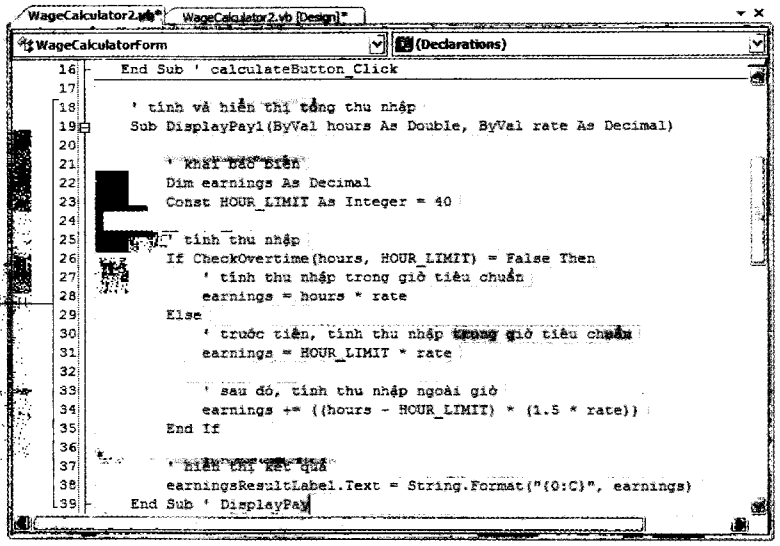
   Thủ tục `DisplayPay` nhận các giá trị đối số và lưu chúng trong các tham số `hours` và `rate`. Lưu ý rằng cú pháp của thủ tục Sub giống với cú pháp của thủ tục `Function` chỉ có một vài thay đổi nhỏ. Cụ thể là, các từ khóa `Function` và `End Function` được thay thế bằng các từ khóa tương ứng là **Sub** và **End Sub** (dòng 19 và 39 trên Hình 13.17). Một sự khác nhau nữa là thủ tục Sub không trả về giá trị nên không có kiểu trả về.

Gọi **DisplayPay**



Hình 13.16 `calculateButton_Click` gọi `DisplayPay`.

DisplayPay tính toán và hiển thị thu nhập cho người dùng



Hình 13.17 Định nghĩa thủ tục Sub `DisplayPay`.

Lưu ý rằng biến `earnings` và hằng số `HOUR_LIMIT` đã được truyền vào thủ tục `DisplayPay`. (Ở Hình 7.14 của Chương 7, chúng được đặt ở trong xử lý sự kiện `calculateButton_Click`). Các tham số và hằng số này không còn cần thiết trong `calculateButton_Click` nữa, vì vậy hãy xóa khỏi xử lý sự kiện.

Lỗi lập trình thường gặp

Khai báo biến trong thân thủ tục trùng tên với tham số trong tiêu đề thủ tục là lỗi biên dịch.

Dòng 26-35 định nghĩa lệnh `If...Then...Else`. Lệnh này sẽ xác định xem có tính thời gian làm ngoài giờ hay không. Điều kiện của lệnh này xác định `hours` có nhỏ hơn hoặc bằng `HOUR_LIMIT` hay không. Nếu có, thu nhập của nhân viên sẽ không bao gồm lương ngoài giờ. Trong trường hợp ngược lại, thu nhập của nhân viên sẽ được tính thêm cả lương ngoài giờ. Dòng 38 hiển thị kết quả (có định dạng tiền tệ) trên `Label`.

Khi gặp lệnh `End Sub` thì điều khiển chương trình sẽ được trả về cho lệnh gọi thủ tục `calculateButton_Click` (dòng 15 trên Hình 13.16).

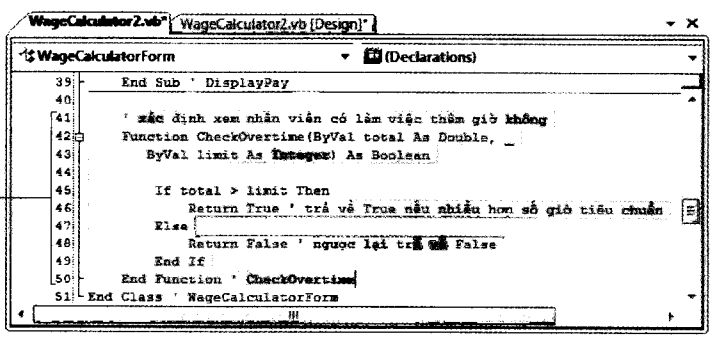
6. ***Lưu project.*** Chọn **File > Save All** để lưu mã đã được sửa đổi.

Phần tiếp theo sẽ hướng dẫn cách thêm thủ tục `Function CheckOvertime` vào ứng dụng **Wage Calculator**. Thủ tục `CheckOvertime` sẽ xác định nhân viên có làm việc ngoài giờ hay không.

Tạo thủ tục Function trong ứng dụng Wage Calculator

1. ***Tạo tiêu đề thủ tục Function.*** Thêm thủ tục `Function CheckOvertime` (dòng 41-50 trên Hình 13.18) vào sau định nghĩa thủ tục `DisplayPay`. Lưu ý rằng kiểu trả về của thủ tục là `Boolean` - giá trị mà thủ tục trả về phải có kiểu `Boolean` (một hằng số, biến hoặc biểu thức có giá trị là `True` hoặc `False`).

`CheckOvertime` xác định nhân viên có làm việc ngoài giờ hay không



Hình 13.18 Định nghĩa thủ tục `Function CheckOvertime`.

Lỗi lập trình thường gặp

Nếu một thủ tục `Function` không thể thực hiện việc trả về giá trị thì thủ tục đó sẽ trả về giá trị mặc định tùy thuộc vào kiểu trả về đã chỉ định trong phần tiêu đề (0 cho các kiểu số, `False` cho kiểu `Boolean`, `Nothing` cho kiểu tham chiếu) và thường gây ra các lỗi lô gíc.

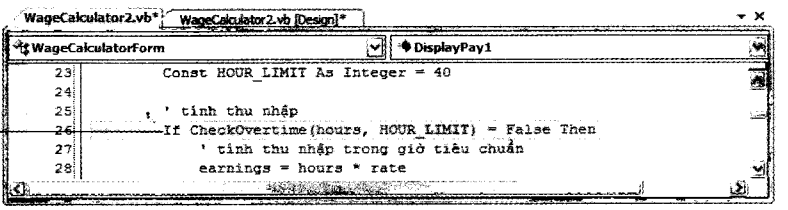
Khi chương trình gọi `CheckOvertime`, điều khiển chương trình sẽ chuyển đến đầu thủ tục này (dòng 42). Đối số đã được truyền vào thủ tục (qua lời gọi thủ tục mà bạn sẽ viết trong bước tiếp theo) và được lưu trong tham số `total` (tổng số giờ đã làm việc) và `limit` (số giờ làm việc tiêu chuẩn). Dòng 46 trả về giá trị `True` để chỉ ra rằng nhân viên có làm việc ngoài giờ; dòng 48 trả về giá trị `False` để chỉ ra rằng nhân viên vẫn đang làm việc trong giờ tiêu chuẩn. Điều khiển chương trình cùng với giá trị (`True` hoặc `False`) sẽ được trả về dòng lệnh nơi mà `CheckOvertime` được gọi.

2. ***Thay đổi thủ tục Sub DisplayPay***. Trong thủ tục `Sub DisplayPay`, hãy thay thế lệnh (dòng 26 Hình 13.17)

   ```
   If hours<= HOUR_LIMIT Then
   ```

   bằng dòng 26 trên Hình 13.19. Thay đổi `DisplayPay` để thủ tục này gọi thủ tục `Function CheckOvertime` để xác định xem nhân viên có làm việc ngoài giờ hay không.

Lời gọi cho thủ tục `CheckOvertime`



Hình 13.19 `DisplayPay` gọi thủ tục `Function CheckOvertime`.

Dòng 26 gọi thủ tục `Function CheckOvertime` trong điều kiện của lệnh `If`. Tại điểm này, ứng dụng copy giá trị của `hours` và `HOUR_LIMIT` (các đối số trong lời gọi thủ tục), điều khiển chương trình chuyển đến tiêu đề của thủ tục `Function CheckOvertime`.

Các tham số trong tiêu đề `CheckOvertime` được thiết lập giá trị ban đầu là bản sao giá trị của `hours` và `HOUR_LIMIT`. Giá trị trả về từ `CheckOvertime` được so sánh với `False` ở dòng 26. Lưu ý rằng dòng lệnh này cũng có thể được viết thành

```
If Not CheckOvertime(hours, HOUR_LIMIT) Then
```

Bây giờ khi `Button` **Calculate** được nhấn, `DisplayPay` sẽ được gọi và thực thi và thủ tục này lại gọi đến thủ tục `Function CheckOvertime`. Chuỗi lời gọi này được lặp lại mỗi lần người dùng nhấn vào `Button` **Calculate**.

3. ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng. Nhập tiền thù lao mỗi giờ và số giờ đã làm việc (dưới 40), sau đó nhấn `Button` **Calculate**. Hãy kiểm tra xem thu nhập đã được hiển thị đúng chưa. Thay đổi số giờ làm việc thành giá trị lớn hơn 40 và nhấn `Button` **Calculate** một lần nữa. Hãy kiểm tra và chắc chắn rằng ứng dụng hiển thị kết quả đúng.
4. ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

Hình 13.20 giới thiệu mã nguồn cho ứng dụng **Wage Calculator**. Các dòng mã chứa khái niệm lập trình mới mà bạn vừa học trong chương này được đánh dấu.

```
1  Public Class WageCalculatorForm
2      ' xử lý sự kiện Click của Button Calculate
3      Private Sub calculateButton_Click(ByVal sender As System.Object, _
4          ByVal e As System.EventArgs) Handles calculateButton.Click
5
6          ' khai báo biến
7          Dim userHours As Double
8          Dim wage As Decimal
9
10         ' gán cho biến giá trị do người dùng nhập vào
11         userHours = Val(hoursTextBox.Text)
12         wage = Val(wageTextBox.Text)
13
14         ' gọi thủ tục Sub DisplayPay
15         DisplayPay(userHours, wage)
16     End Sub ' calculateButton_Click
17
18     ' tính và hiển thị tổng thu nhập
19     Sub DisplayPay(ByVal hours As Double, ByVal rate As Decimal)
20
21         ' khai báo biến
22         Dim earnings As Decimal
23         Const HOUR_LIMIT As Integer = 40
24
25         ' tính thu nhập
26         If CheckOvertime(hours, HOUR_LIMIT) = False Then
27             ' tính thu nhập trong giờ tiêu chuẩn
28             earnings = hours * rate
29         Else
30             ' trước tiên, tính thu nhập trong giờ tiêu chuẩn
31             earnings = HOUR_LIMIT * rate
32
33             ' sau đó, tính thu nhập ngoài giờ
34             earnings += ((hours - HOUR_LIMIT) * (1.5 * rate))
35         End If
36
37         ' hiển thị kết quả
38         earningsResultLabel.Text = String.Format("{0:C}", earnings)
39     End Sub ' DisplayPay
40
41     ' xác định xem nhân viên có làm việc thêm giờ không
42     Function CheckOvertime(ByVal total As Double, _
43     ByVal limit As Integer) As Boolean
44
45     If total > limit Then
46         Return True ' trả về True nếu nhiều hơn số giờ tiêu chuẩn
47     Else
48         Return False ' ngược lại trả về False
49     End If
50     End Function ' CheckOvertime
51 End Class ' WageCalculatorForm
```

Lời gọi thủ tục **Sub** để tính toán và hiển thị thu nhập

Tiêu đề thủ tục **Sub** chỉ ra tên và kiểu tham số

Lời gọi thủ tục **Function** để xác định nhân viên có làm việc ngoài giờ hay không

Từ khóa **End Sub** cho biết điểm kết thúc của định nghĩa thủ tục **Sub**

Tiêu đề thủ tục **Function** chỉ ra tên và kiểu của tham số cũng như kiểu của giá trị trả về

Từ khóa **End Function** cho biết điểm kết thúc của định nghĩa thủ tục **Function**

Hình 13.20 Mã nguồn của ứng dụng **Wage Calculator.**

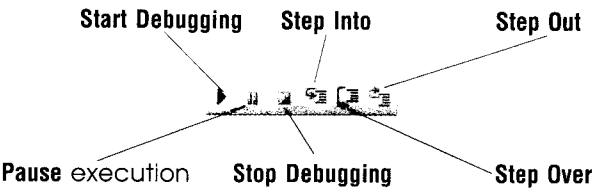
TỰ ÔN TẬP

1. Các đối số được truyền vào thủ tục có thể là_____.
   a) hằng số b) biểu thức
   c) biến d) Các lựa chọn trên đều đúng
2. _____là danh sách các khai báo trong tiêu đề thủ tục được ngăn cách bằng các dấu phẩy.
   a) Danh sách đối số b) Danh sách tham số
   c) Danh sách giá trị d) Danh sách biến

Đáp án: 1) d. 2) b.

13.5 Sử dụng trình gỡ lỗi: Các điều khiển gỡ lỗi.

Bây giờ bạn sẽ tiếp tục tìm hiểu về trình gỡ lỗi bằng việc tìm hiểu các điều khiển gỡ lỗi trên thanh công cụ **Standard** (Hình 13.21). Những `ToolStripButton` này giúp truy cập đến các chức năng trong menu **Debug** một cách dễ dàng. Nếu thanh công cụ **Standard** không xuất hiện trong IDE, hãy chọn **View > Toolbars > Standard**. Trong phần này, bạn sẽ học cách sử dụng các `ToolStripButton` gỡ lỗi để kiểm tra thủ tục có thực thi đúng hay không. Trong phần tiếp theo, chúng ta sẽ sử dụng `ToolStripButton` gỡ lỗi để kiểm tra ứng dụng **Wage Calculator**.

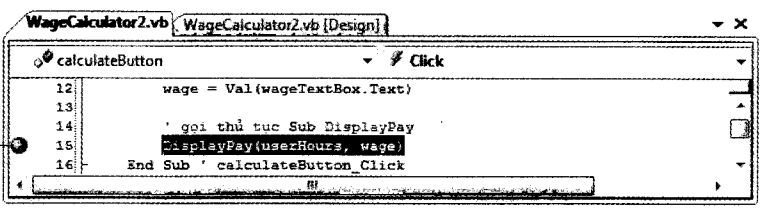


Hình 13.21 Điều khiển gỡ lỗi trên thanh công cụ **Standard**.

Sử dụng trình gỡ lỗi: Điều khiển gỡ lỗi

1. ***Mở ứng dụng đã hoàn thiện***. Nếu ứng dụng mẫu **Wage Calculator** chưa được mở, hãy nhấn đúp vào file `WageCalculator2.sln` trong thư mục `C:\SimplyVB2008\Calculator2` để mở ứng dụng.
2. ***Đặt breakpoint (điểm dừng)***. Đặt một breakpoint ở dòng 15 bằng cách nhấn vào thanh lề (Hình 13.22).
3. ***Khởi động trình gỡ lỗi***. Để khởi động trình gỡ lỗi, chọn **Debug > Start Debugging** hoặc nhấn `ToolStripButton` **Start Debugging** (▸) trên thanh công cụ **Standard**. Ứng dụng **Wage Calculator** sẽ được thực thi. Nhập giá trị `7.50` vào `TextBox` **Hourly wage:** và nhập 35 vào `TextBox` **Weekly hours:**. Nhấn `Button` **Calculate**.

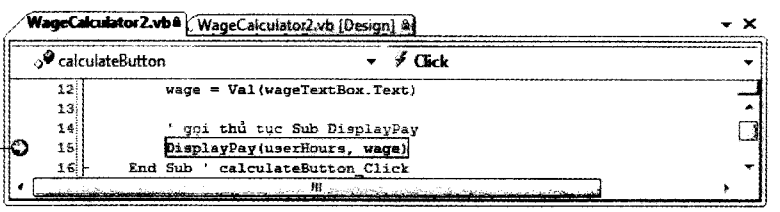
Breakpoint được đặt ở dòng chứa lời gọi thủ tục



Hình 13.22 Đặt breakpoint.

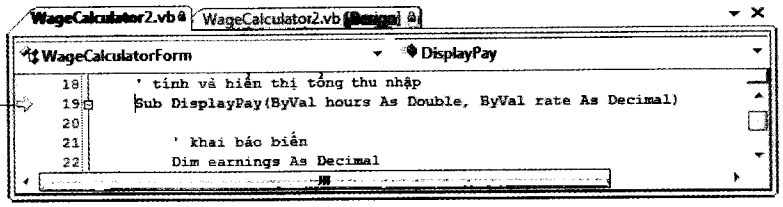
4. ***Sử dụng `ToolStripButton` Step Into***. `ToolStripButton` **Step Into** ( ) sẽ thực thi lệnh tiếp theo (dòng được đánh dấu) trong ứng dụng. Nếu lệnh tiếp theo là lời gọi thủ tục (Hình 13.23) và `ToolStripButton` **Step Into** được nhấn, điều khiển sẽ được chuyển cho thủ tục được gọi. `ToolStripButton` **Step Into** cho phép bạn đi vào một thủ tục và thực thi từng lệnh của thủ tục đó. ***Nhấn*** `ToolStripButton` **Step Into** để đi vào thủ tục `DisplayPay` (Hình 13.24).

Lệnh được thực thi tiếp theo là lời gọi thủ tục



Hình 13.23 Lệnh gọi thủ tục `DisplayPay`.

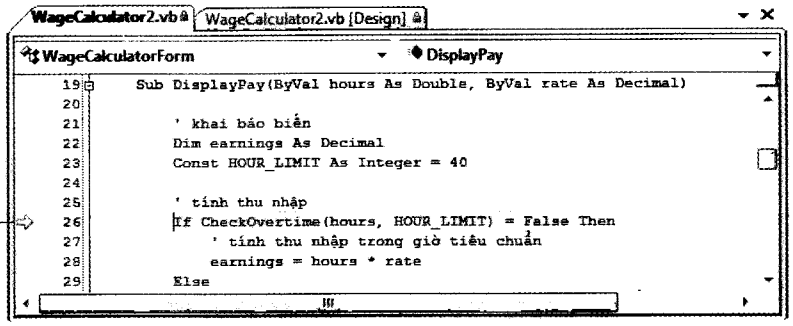
Chuyển điều khiển tới định nghĩa thủ tục



Hình 13.24 Sử dụng `ToolStripButton` **Step Into** trên thanh công cụ **Standard**.

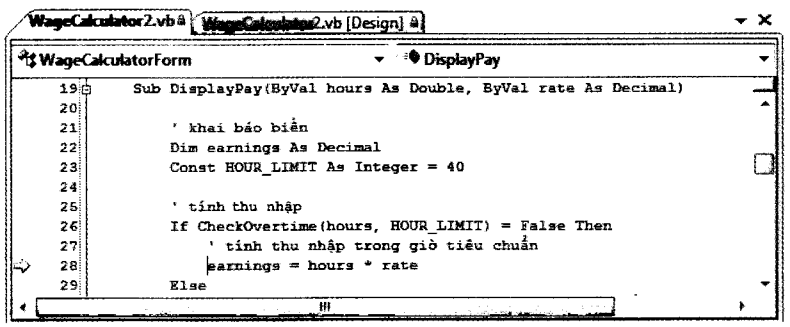
5. ***Nhấn ToolStripButton Step Over.*** Nhấn `ToolStripButton` **Step Over** (↷) để thực thi lệnh hiện thời (dòng 19 Hình 13.24) mà không cần đi vào lệnh đó và chuyển điều khiển đến dòng 26 (Hình 13.25).

Thủ tục `CheckOvertime` được thực thi mà không đi vào phần thân bên trong, nếu bạn nhấn vào `ToolStripButton` **Step Over**



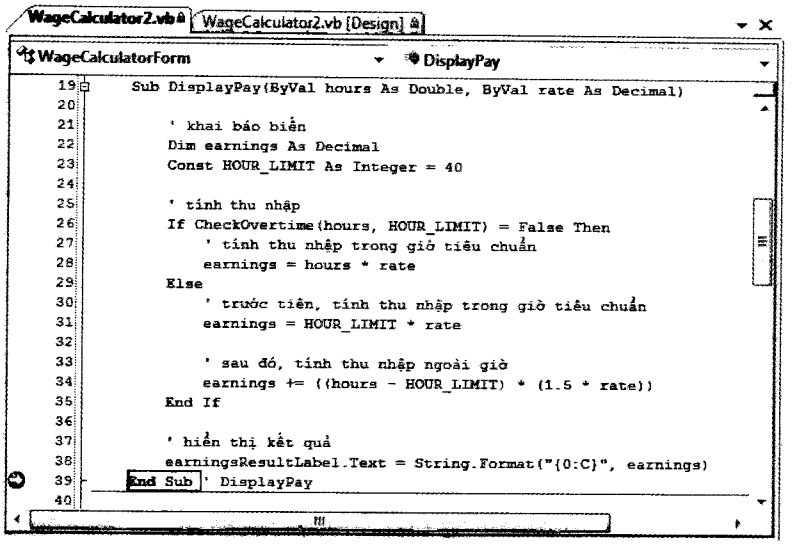
Hình 13.25 Sử dụng `ToolStripButton` **Step Over** trên thanh công cụ **Standard**.

6. ***Nhấn ToolStripButton Step Over một lần nữa.*** **Step Over** thực hiện giống như **Step Into** khi lệnh thực thi tiếp theo không chứa lời gọi thủ tục. Nếu lệnh tiếp theo chứa lời gọi thủ tục, thủ tục được gọi sẽ thực thi hoàn toàn mà không chuyển điều khiển vào thủ tục được gọi. Mũi tên vàng sẽ tiến đến dòng có thể thực thi tiếp theo trong thủ tục hiện thời (Hình 13.26).



Hình 13.26 Sử dụng `ToolStripButton` **Step Over** trên thanh công cụ **Standard** một lần nữa.

7. ***Đặt breakpoint.*** Đặt một breakpoint ở cuối thủ tục `DisplayPay` dòng 39 (End Sub) trên Hình 13.27. Bạn sẽ dùng đến breakpoint này ở bước tiếp theo.



Hình 13.27 Sử dụng `ToolStripButton` **Continue** trên thanh công cụ **Standard**.

8. ***Sử dụng ToolStripButton Continue.*** Nhấn `ToolStripButton` **Continue** ( ) sẽ thực thi cho đến khi gặp breakpoint tiếp theo. Lưu ý rằng có một lệnh có thể thực thi (dòng 38) trước breakpoint được đặt ở *Bước 7.* Nhấn `ToolStripButton` **Continue.** Điều khiển chuyển đến dòng 39 (Hình 13.27). Thuộc tính này đặc biệt hữu dụng khi có nhiều dòng lệnh trước breakpoint tiếp theo mà bạn lại không muốn thực thi tuần tự từng lệnh một.
9. ***Sử dụng ToolStripButton Stop Debugging.*** Nhấn `ToolStripButton` **Stop Debugging** ( ) để kết thúc gỡ lỗi và trả IDE về chế độ **Design.**
10. ***Khởi động trình gỡ lỗi.*** Chúng ta cần trình bày tính năng cuối cùng đòi hỏi bạn khởi động trình gỡ lỗi lần nữa. Khởi động trình gỡ lỗi và nhập giá trị đầu vào như bạn đã làm ở *Bước 3.*
11. ***Sử dụng ToolStripButton Step Into.*** Giữ lại breakpoint ở dòng 15 (Hình 13.22) và xóa breakpoint ở dòng 39. Lặp lại *Bước 4.*
12. ***Nhấn ToolStripButton Step Out.*** Sau khi bạn đã vào thủ tục `DisplayPay`, nhấn `ToolStripButton` **Step Out** ( ) để thực thi các lệnh còn lại trong thủ tục và trả điều khiển về dòng 15 chứa lời gọi thủ tục. Thông thường trong các thủ tục được gọi dài, bạn muốn xem một vài dòng lệnh quan trọng và sau đó tiếp tục gỡ lỗi mã của thủ tục gọi. Thuộc tính này hữu dụng cho những tình huống mà bạn không muốn tiếp tục dừng lại ở từng dòng lệnh của toàn bộ thủ tục được gọi.
13. ***Nhấn ToolStripButton Stop Debugging.*** Nhấn `ToolStripButton` **Stop Debugging** để kết thúc gỡ lỗi.
14. ***Đóng IDE.*** Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

TỰ ÔN TẬP

1. Trong quá trình gỡ lỗi, `ToolStripButton`_____thực thi các lệnh còn lại trong lời gọi thủ tục hiện thời và trả điều khiển chương trình về nơi thủ tục được gọi.

   a) **Step Into** b) **Step Out**

   c) **Step Over** d) **Steps**

2. `ToolStripButton`_____làm việc giống như `ToolStripButton` **Step Into** khi lệnh thực thi tiếp theo không chứa lời gọi thủ tục.

   a) **Step Into** b) **Step Out**

   c) **Step Over** d) **Steps**

Đáp án: 1) b. 2) c.

13.6 Tham số `Optional`

Chương trình phải gọi thủ tục lặp lại nhiều lần với cùng một giá trị đối số cho một tham số đặc biệt là điều thường gặp. Trong những trường hợp như vậy, bạn có thể chỉ ra tham số đó là **tham số `Optional`** - là tham số có giá trị mặc định. Trong lời gọi thủ tục, khi đối số cho tham số `Optional` được bỏ qua, trình biên dịch sẽ viết lại lời gọi thủ tục và chèn vào giá trị mặc định của tham số `Optional`. Có ba quy tắc sử dụng tham số `Optional`:

- Tham số `Optional` phải có giá trị mặc định.
- Giá trị mặc định phải là biểu thức không đổi (thường là giá trị cụ thể, chẳng hạn giá trị bằng số hoặc chuỗi ký tự).
- Tất cả các tham số xuất hiện sau tham số `Optional` trong danh sách tham số cũng phải là tham số `Optional`.

Hãy xem xét `Function BoxVolume` tính toán thể tích của hình hộp (chiều dài nhân chiều rộng nhân chiều cao):

```
Function BoxVolume( Optional ByVal length As Integer = 1, _
    Optional ByVal width As Integer = 1, _
    Optional ByVal height As Integer = 1 ) As Integer

    Return length * width * height
End Function ' BoxVolume
```

Trong trường hợp này, cả ba tham số đều là không bắt buộc do có từ khóa `Optional` trước khai báo của mỗi tham số. Lưu ý rằng mỗi tham số có giá trị mặc định là một số cụ thể được chỉ ra sau dấu = . Nếu `BoxVolume` được gọi với ít hơn ba tham số `Integer`, giá trị `1` sẽ được gán cho mỗi đối số bị bỏ qua. Bây giờ bạn có thể gọi `Function BoxVolume` bằng một vài cách khác nhau:

```
BoxVolume() ' trả về 1; sử dụng giá trị mặc định cho chiều dài, chiều rộng và chiều cao
BoxVolume(10) ' trả về 10; sử dụng giá trị mặc định cho chiều rộng và chiều cao
BoxVolume(10, 20) ' trả về 200, sử dụng giá trị mặc định cho chiều cao
BoxVolume(10, 20, 30) ' trả về 6000, không sử dụng giá trị mặc định
BoxVolume(, 20, 30) ' trả về 600; sử dụng giá trị mặc định cho chiều dài
BoxVolume(10,  , 30) ' trả về 300; sử dụng giá trị mặc định cho chiều rộng
```

Các đối số có thể được bỏ qua cho bất cứ tham số nào. Chẳng hạn, hai lời gọi phương thức cuối bỏ qua tham số chiều dài và chiều rộng. Lưu ý rằng dấu phẩy được sử dụng để giữ cho khi đối số được bỏ qua không phải là đối số cuối cùng trong lời gọi.

13.7 Tổng kết

Trong chương này bạn đã tìm hiểu về sự khác nhau giữa thủ tục `Function` và thủ tục `Sub`. Bạn cũng đã tìm hiểu về cách thủ tục có thể được sử dụng để tổ chức ứng dụng tốt hơn. Từ đây trở về sau, chúng ta thường xuyên đề cập đến các thủ tục `Function` và `Sub` chỉ đơn giản như những phương thức. Chương này đã giới thiệu khái niệm tái sử dụng mã, cho thấy sử dụng mã có sẵn sẽ tiết kiệm thời gian và công sức như thế nào. Bạn đã sử dụng mã có sẵn do .NET Framework Class

Library cung cấp và đã học cách tạo ra mã để có thể dùng lại được trong những ứng dụng khác.

Bạn đã học cú pháp tạo và gọi hai loại thủ tục. Bạn đã tìm hiểu các thành phần của thủ tục, bao gồm tiêu đề thủ tục, danh sách tham số và kiểu trả về cùng với lệnh `Return` - chỉ riêng với thủ tục `Function`. Sau khi học cách phát triển và viết thủ tục, bạn đã tìm hiểu trật tự thực thi bắt đầu từ dòng lệnh nơi thủ tục được gọi, sau đó thực thi thủ tục được gọi và cuối cùng trả điều khiển về điểm nó được gọi. Trong các ứng dụng của chương này, bạn đã tạo ba thủ tục `Function` (`Square`, `Maximum` và `CheckOvertime`) và một thủ tục `Sub` (`DisplayPay`).

Sau khi tạo thủ tục trong chương này, bạn đã học cách gỡ lỗi cho thủ tục trong ứng dụng bằng cách sử dụng `ToolStripButton` trên thanh công cụ **Standard**. Những `ToolStripButton` này (bao gồm **Step into**, **Step Out** và **Step Over**) có thể được sử dụng để xác định thủ tục có thực thi đúng hay không. Cuối cùng, bạn học cách khai báo và sử dụng tham số `Optional`.

Trong chương tiếp theo, bạn sẽ tìm hiểu về điều khiển `GroupBox` và `DateTimePicker`, sau đó sử dụng chúng để xây dựng ứng dụng **Shipping Time** để xử lý thông tin về thời gian hàng hóa được chuyển từ nơi này sang nơi khác.

TỔNG KẾT KỸ NĂNG

Tạo thủ tục Function

- Viết từ khóa `Function` vào đầu tiêu đề thủ tục.
- Chỉ ra một danh sách tham số bằng cách khai báo tên và kiểu tham số. Trong danh sách tham số, sử dụng từ khóa `ByVal` thay cho từ khóa `Dim`.
- Đặt từ khóa `As` và kiểu trả về sau dấu đóng ngoặc đơn (là dấu dùng để kết thúc danh sách tham số).
- Nhấn *Enter* để tự động thêm mệnh đề kết thúc `End Function`.
- Sử dụng lệnh `Return` để trả về giá trị.

Sử dụng thủ tục Function

- Sử dụng thủ tục `Function` khi cần trả về giá trị cho thủ tục gọi nó.

Trả về giá trị từ thủ tục Function

- Sử dụng từ khóa `Return` kèm theo giá trị cần trả về.

Tạo thủ tục Sub

- Viết từ khóa `Sub` vào đầu tiêu đề thủ tục.
- Chỉ ra danh sách tham số bằng cách khai báo tên và kiểu cho mỗi tham số. Trong danh sách tham số, sử dụng từ khóa `ByVal` thay thế từ khóa `Dim`.
- Nhấn *Enter* để tự động thêm mệnh đề kết thúc `End Sub`.
- Thêm mã cho thân thủ tục để thực hiện một nhiệm vụ cụ thể.

Gọi thủ tục

- Chỉ ra tên thủ tục và danh sách đối số.
- Kiểm tra để chắc chắn rằng danh sách đối số được truyền có số lượng, kiểu và trật tự đúng với số lượng, kiểu và trật tự của tham số trong định nghĩa thủ tục.

Sử dụng điều khiển gỡ lỗi trên thanh công cụ Standard

- Để đi qua từng lệnh trong thân thủ tục được gọi trong trình gỡ lỗi, hãy nhấn `ToolStripButton` **Step Into**.
- Để thoát ra khỏi thủ tục và trở về thủ tục gọi nó trong trình gỡ lỗi, hãy nhấn `ToolStripButton` **Step Out**.
- Khi bạn muốn thực thi thủ tục mà không muốn đi qua từng dòng lệnh của thủ tục đó trong trình gỡ lỗi, hãy nhấn `ToolStripButton` **Step Over**.

Chỉ ra các tham số Optional trong định nghĩa thủ tục

- Đặt từ khóa `Optional` trước khai báo tham số muốn có giá trị mặc định.
- Đặt dấu = và giá trị mặc định sau kiểu tham số.

THUẬT NGỮ

danh sách tham số (parameter list) - Là danh sách được ngăn cách bằng dấu phẩy được sử dụng trong thủ tục để khai báo tên và kiểu của mỗi tham số.

định nghĩa thủ tục (procedure definition) - Bao gồm tiêu đề, thân và lệnh kết thúc thủ tục.

đối số (argument) - Là thông tin được cung cấp cho lời gọi thủ tục.

gọi thủ tục (invoking a procedure) - Gọi đến một thủ tục và thực thi nhiệm vụ của thủ tục đó.

kiểu trả về - Kiểu dữ liệu của kết quả do thủ tục `Function` trả về.

kỹ thuật chia để trị (divide-and-conquer technique) - Xây dựng những ứng dụng lớn từ những phần nhỏ, dễ quản lý nhằm mục đích phát triển và bảo trì ứng dụng lớn được dễ dàng hơn.

lệnh `Return` - Dùng để trả về giá trị từ thủ tục.

lời gọi thủ tục - Gọi đến thủ tục, chỉ ra tên thủ tục và cung cấp đối số mà thủ tục được gọi cần để thực thi nhiệm vụ.

phương thức - Là thủ tục nằm trong lớp.

phương thức `Max` của lớp `Math` - Là phương thức trả về số lớn hơn trong hai đối số.

phương thức `Min` của lớp `Math` - Là phương thức trả về số nhỏ hơn trong hai đối số.

phương thức `Sqrt` của lớp `Math` - Là phương thức trả về giá trị căn bậc hai của đối số truyền vào.

tái sử dụng mã - Sử dụng mã có sẵn để xây dựng mã mới. Việc tái sử dụng mã sẽ tiết kiệm thời gian, sức lực và tiền bạc.

tên thủ tục - Đứng sau từ khóa `Sub` hoặc `Function` để phân biệt các thủ tục với nhau. Tên thủ tục có thể là bất cứ định danh hợp lệ nào.

tham số (parameter) - Là biến được khai báo trong danh sách tham số của thủ tục, được sử dụng trong thân thủ tục.

tham số `Optional` - Là tham số được chỉ định giá trị mặc định. Nếu đối số của tham số này bị bỏ qua trong lời gọi thủ tục, trình biên dịch sẽ gán giá trị mặc định cho đối số.

thân thủ tục - Gồm các khai báo và lệnh xuất hiện sau tiêu đề thủ tục nhưng trước các từ khóa `End Sub` hoặc `End Function`. Thân thủ tục chứa mã Visual Basic thực thi hành động, thường là thao tác với các tham số trong danh sách tham số.

thành phần hóa (componentization) - *Xem* kỹ thuật chia để trị

thủ tục (procedure) - Là tập hợp lệnh để thực hiện một nhiệm vụ cụ thể.

thủ tục do người dùng định nghĩa (programmer-defined procedure) - Thủ tục do lập trình viên tự định nghĩa để đáp ứng nhu cầu nhất định cho ứng dụng cụ thể.

thủ tục được gọi (callee) - Là thủ tục được gọi bởi thủ tục khác

thủ tục `Function` - Là thủ tục tương tự với thủ tục `Sub`, với một khác biệt quan trọng: thủ tục `Function` trả về giá trị cho thủ tục gọi, trong khi đó thủ tục Sub thì không.

thủ tục gọi (caller) - Thủ tục gọi đến thủ tục khác

thủ tục Sub - Là thủ tục tương tự với thủ tục `Function`, với một khác biệt quan trọng là thủ tục `Function` trả về giá trị, trong khi thủ tục `Sub` thì không.

tiêu đề thủ tục (procedure header) - Dòng đầu tiên của một thủ tục (bao gồm từ khóa `Sub` hoặc `Function`, tên thủ tục, danh sách tham số và kiểu trả về (chỉ riêng với thủ tục `Function`).

tính năng *Parameter Info* của IDE - cung cấp thông tin về thủ tục và đối số của chúng.

từ khóa `ByVal` - Từ khóa chỉ ra rằng thủ tục gọi sẽ truyền bản sao giá trị đối số trong lời gọi thủ tục cho thủ tục được gọi.

từ khóa `End Function` - Cho biết điểm kết thúc của thủ tục `Function`.

từ khóa `End Sub` - Cho biết điểm kết thúc của thủ tục `Sub`.

từ khóa `Function` - bắt đầu định nghĩa thủ tục `Function`.

từ khóa Return - biểu thị lệnh trả về giá trị cho thủ tục gọi.
từ khóa Sub - bắt đầu định nghĩa thủ tục Sub.

ĐIỀU KHIỂN, SỰ KIỆN, THUỘC TÍNH & PHƯƠNG THỨC

Math Lớp này cung cấp phương thức dùng để thực hiện phép tính toán số học thông thường.

- ***Phương thức***

  Min - Trả về giá trị nhỏ hơn trong hai số.

  Max - Trả về giá trị lớn hơn trong hai số.

  Sqrt - Trả về căn bậc hai của một số.

CÂU HỎI TRẮC NGHIỆM

13.1 Thủ tục được định nghĩa bắt đầu với từ khóa Sub _____.

a) phải chỉ ra kiểu trả về
b) không nhận đối số
c) trả về giá trị
d) không trả về giá trị

13.2 Kỹ thuật phát triển ứng dụng lớn từ những đoạn mã nhỏ, dễ quản lý được gọi là_____.

a) chia để trị
b) trả về giá trị
c) trực tuyến và truyền thống
d) thuật toán xây dựng khối

13.3 Đâu là sự khác nhau giữa thủ tục Sub và thủ tục Function?

a) Thủ tục Sub trả về giá trị, thủ tục Function thì không.
b) Thủ tục Function trả về giá trị, thủ tục Sub thì không.
c) Thủ tục Sub nhận các tham số, thủ tục Function thì không.
d) Thủ tục Function nhận các tham số, thủ tục Sub thì không.

13.4 Điều gì xảy ra sau khi thực hiện một lời gọi thủ tục?

a) Điều khiển chuyển đến thủ tục được gọi. Sau khi thực thi xong thủ tục này, ứng dụng quay về thủ tục gọi và thực hiện tiếp những câu lệnh phía sau lời gọi thủ tục.
b) Điều khiển chuyển đến thủ tục được gọi. Sau khi thực thi xong thủ tục này, ứng dụng tiếp tục thực thi lệnh nằm sau định nghĩa của thủ tục được gọi.
c) Thực thi lệnh đứng trước thủ tục gọi.
d) Ứng dụng kết thúc.

13.5 Function trả về_____giá trị.

a) không
b) một
c) một hoặc nhiều
d) không, một hoặc nhiều

13.6 Điều nào sau đây là đúng khi viết lời gọi thủ tục?

a) Số lượng đối số trong lời gọi thủ tục phải đúng với số lượng tham số trong tiêu đề thủ tục.
b) Kiểu đối số cần phải tương thích với kiểu tham số tương ứng của chúng.
c) Cả a và b đều đúng.
d) Cả a và b đều sai.

13.7 Lệnh nào sau đây trả về biến value từ thủ tục Function?

a) `Return Dim value`
b) `Return value As Integer`
c) `value Return`
d) `Return value`

13.8 ToolStripButton_____thực thi lệnh tiếp theo trong ứng dụng. Nếu lệnh tiếp theo có chứa lời gọi thủ tục, thủ tục được gọi sẽ thực thi toàn bộ mà không duyệt qua từng dòng lệnh.

a) **Step Into**
b) **Step Out**
c) **Step Over**
d) **Steps**

13.9 Dòng đầu tiên của thủ tục (bao gồm từ khóa `Sub` hay `Function`, tên thủ tục, danh sách tham số và kiểu trả về của thủ tục `Function`) là_____của thủ tục.

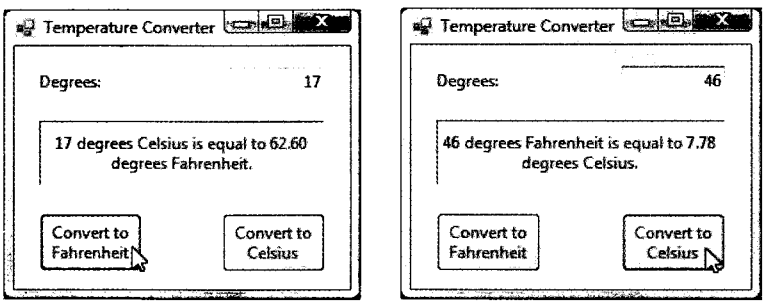
a) thân
b) tên
c) thủ tục gọi
d) tiêu đề

13.10 Phương thức_____của lớp `Math` tính toán căn bậc hai của giá trị đối số được truyền vào.

a) `SquareRoot`
b) `Root`
c) `Sqrt`
d) `Square`

BÀI TẬP

13.11 (***Ứng dụng Temperature Converter***) Viết ứng dụng để chuyển đổi số đo nhiệt độ (Hình 13.28). Ứng dụng này sẽ thực hiện hai kiểu chuyển đổi số đo nhiệt độ từ độ F sang độ C và ngược lại.



Hình 13.28 Giao diện ứng dụng **Temperature Converter.**

Chú thích hình

- ***Degrees:*** *độ*
- ***Convert to Fahrenheit:*** *Chuyển sang độ F*
- ***Convert to Celsius:*** *Chuyển sang độ C*

a) ***Copy template của ứng dụng vào thư mục làm việc.*** Copy thư mục `C:\Examples\Tutorial13\Exercises\TemperatureConversion` vào thư mục `C:\SimplyVB2008` của bạn.

b) ***Mở file template của ứng dụng.*** Nhấn đúp vào file `Temperature-Conversion.sln` trong thư mục `TemperatureConversion`.

c) ***Chuyển độ F sang độ C.*** Sử dụng công thức:

```
celsius = (5 / 9) * (fahrenheit - 32)
```

d) ***Chuyển độ C sang độ F.*** Sử dụng công thức:

```
fahrenheit = (9 / 5) * celsius + 32
```

e) ***Thêm xử lý sự kiện cho ứng dụng.*** Nhấn đúp vào mỗi `Button` để thêm xử lý sự kiện cho ứng dụng. Các xử lý sự kiện này gọi là thủ tục (mà bạn sẽ định nghĩa trong bước tiếp theo) để chuyển đổi số đo nhiệt độ được nhập vào sang độ F hay độ C. Mỗi xử lý sự kiện đều hiển thị kết quả trong `Label` đầu ra.

f) ***Thêm thủ tục `Function` cho ứng dụng.*** Tạo thủ tục `Function` để thực hiện mỗi lần chuyển đổi bằng cách sử dụng các công thức trên đây. Người dùng sẽ nhập số đo cần chuyển đổi.

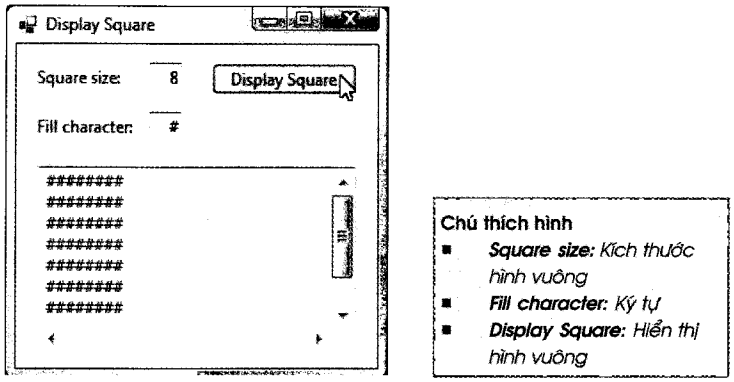
g) ***Định dạng nhiệt độ đầu ra.*** Để định dạng thông tin nhiệt độ, sử dụng phương thức `String.Format`. Sử dụng mã định dạng F để hiển thị nhiệt độ có hai chữ số thập phân.

h) ***Chạy ứng dụng.*** Chọn **Debug > Start Debugging**. Nhập vào giá trị nhiệt độ. Nhấn `Button` **Convert to Fahrenheit** và xác nhận rằng ứng dụng hiển thị kết quả đúng theo công thức đã cho. Tiếp tục nhấn `Button` **Convert to Celsius** và kiểm tra đầu ra có đúng không.

i) ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

j) ***Đóng IDE***. Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

13.12 (***Ứng dụng Display Square***) Viết ứng dụng hiển thị hình vuông được tạo thành bởi ký tự do người dùng nhập vào (Hình 13.29). Người dùng cũng phải nhập kích thước trên hình vuông.



Hình 13.29 Ứng dụng **Display Square.**

a) ***Copy template vào thư mục làm việc.*** Copy thư mục `C:\Examples\Tutorial13\Exercises\DisplaySquare` vào thư mục `C:\SimplyVB2008` của bạn.

b) ***Mở file template của ứng dụng.*** Nhấn đúp vào file `DisplaySquare.sln` trong thư mục `DisplaySquare`.

c) ***Thêm thủ tục Sub.*** Viết thủ tục `Sub DisplaySquare` để hiển thị hình vuông. Kích thước (độ dài mỗi cạnh) được chỉ ra bởi tham số `size` kiểu `Integer`. Ký tự điền vào hình vuông được chỉ ra bởi tham số kiểu `String fillCharacter`. Sử dụng lệnh `For...Next` được lồng trong lệnh `For...Next` khác để vẽ hình vuông. Lệnh `For...Next` bên ngoài chỉ ra hàng nào hiện thời đang được vẽ. Lệnh `For...Next` bên trong vẽ các ký tự cho hàng đó. Dùng `TextBox` nhiều dòng để hiển thị hình vuông này. Chẳng hạn, nếu cạnh là 8 và `fillCharacter` là # thì ứng dụng giống như trên Hình 13.29.

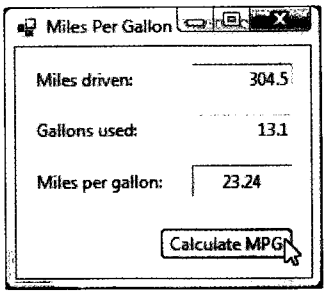
d) ***Tạo xử lý sự kiện cho sự kiện `Click` của `Button`.*** Nhấn đúp vào `Button` **Display Square** để tạo xử lý sự kiện. Lập trình để xử lý sự kiện gọi tới thủ tục `DisplaySquare`.

e) ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng. Nhập kích thước cho hình vuông (chiều dài mỗi cạnh) và ký tự điền vào hình vuông. Nhấn vào `Button` **Dislay Square**. Chương trình sẽ hiển thị hình vuông có kích thước mà bạn nhập vào và được tạo từ ký tự được chỉ định.

f) ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

g) ***Đóng IDE.*** Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

13.13 (***Ứng dụng Miles Per Gallon***) Lái xe thường muốn biết mỗi gallon xăng sẽ đi được bao nhiêu dặm để có thể ước lượng chi phí xăng dầu. Hãy phát triển ứng dụng cho phép người dùng nhập số dặm đi được và số gallon sử dụng, sau đó ứng dụng sẽ tính và hiển thị số dặm mà mỗi gallon đi được.



Chú thích hình
- ***Miles driven:*** *Số dặm*
- ***Gallons used:*** *Số Gallon sử dụng*
- ***Miles per gallon:*** *Số dặm cho mỗi gallon*
- ***MPG:*** *Miles per gallon*

Hình 13.30 Ứng dụng **Miles Per Gallon**.

a) ***Copy template của ứng dụng vào thư mục làm việc.*** Copy thư mục `C:\Examples\Tutorial13\Exercises\MilesPerGallon` vào thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng.*** Nhấn đúp vào file `MilesPerGallon.sln` trong thư mục `MilesPerGallon` để mở ứng dụng.

c) ***Tính toán số dặm cho mỗi gallon.*** Viết thủ tục `Function` `MilesPerGallon` nhận vào số dặm đi được và số gallon đã sử dụng (do người dùng nhập vào), tính toán và trả về số dặm cho mỗi gallon.

d) ***Hiển thị kết quả.*** Tạo xử lý sự kiện `Click` cho `Button` **Calculate MPG** gọi thủ tục `Function` `MilesPerGallon` và hiển thị kết quả trả về như trên Hình 13.30.

e) ***Chạy ứng dụng.*** Chọn **Debug > Start Debugging** để chạy ứng dụng. Nhập giá trị cho số dặm đi được và số gallon đã sử dụng. Nhấn `Button` **Calculate MPG** và kiểm tra kết quả được hiển thị.

f) ***Đóng ứng dụng.*** Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

g) ***Đóng IDE.*** Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

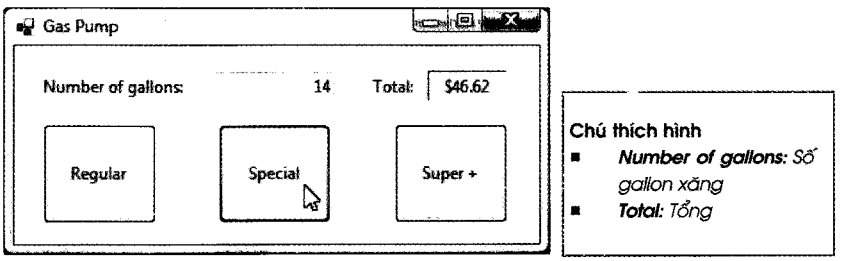
Đoạn mã này làm gì? ▶ **13.14** Đoạn mã sau thực hiện nhiệm vụ gì? Giả sử rằng thủ tục này được gọi với lệnh `Mystery(70, 80)`.

```
1  Sub Mystery(ByVal number1 As Integer, ByVal number2 As Integer)
2     Dim x As Integer
3     Dim y As Double
4
5     x = number1 + number2
6     y = x / 2
7
8     If y <= 60 Then
9        resultLabel.Text = "<= 60"
10    Else
11       resultLabel.Text = "Result is" & y
12    End If
13 End Sub ' Mystery
```

Đoạn mã này có gì sai? ▶ **13.15** Tìm lỗi trong mã sau. Mã này nhận giá trị `Integer` làm đối số và trả về giá trị của đối số nhân với 2.

```
1  Function TimesTwo(ByVal number As Integer) As Integer
2     Dim result As Integer
3
4     result = number * 2
5  End Function ' TimesTwo
```

Đoạn mã này có gì sai? ▶ **13.16** (***Ứng dụng Gas Pump***) Ứng dụng **Gas Pump** (Hình 13.31) tính toán chi phí xăng cho cây xăng. Cây xăng này tính phí `3,13` USD cho mỗi gallon xăng loại **Regular**, `3,33` USD cho mỗi gallon xăng loại **Special** và `3,45` USD cho mỗi gallon xăng loại **Super +**. Người dùng nhập số gallon và chọn loại xăng muốn mua. Ứng dụng gọi một thủ tục `Sub` để tính toán tổng số tiền từ số gallon được nhập vào và loại được chọn rồi hiển thị kết quả. Trong khi chạy thử ứng dụng, bạn nhận thấy rằng một trong ba loại xăng bị tính tiền sai.



Hình 13.31 Ứng dụng **Gas Pump** hiển thị kết quả đúng.

a) ***Copy template của ứng dụng vào thư mục làm việc***. Copy thư mục `C:\Examples\Tutorial13\Exercises\GasPump` vào thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng***. Nhấn đúp vào file `GasPump.sln` trong thư mục `GasPump` để mở ứng dụng.

c) ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng. Xác định loại xăng nào bị tính sai.

d) ***Đặt một breakpoint***. Đặt một breakpoint ở đầu xử lý sự kiện tính kết quả sai. Ví dụ, nếu `Button` **Regular** cung cấp đầu ra sai khi được nhấn, thêm một breakpoint ở đầu xử lý sự kiện `Click` của `Button` này. Sử dụng trình gỡ lỗi để tìm lỗi lôgíc trong ứng dụng.

e) ***Thay đổi ứng dụng.*** Khi đã tìm được lỗi, hãy sửa đổi ứng dụng để chương trình chạy đúng.

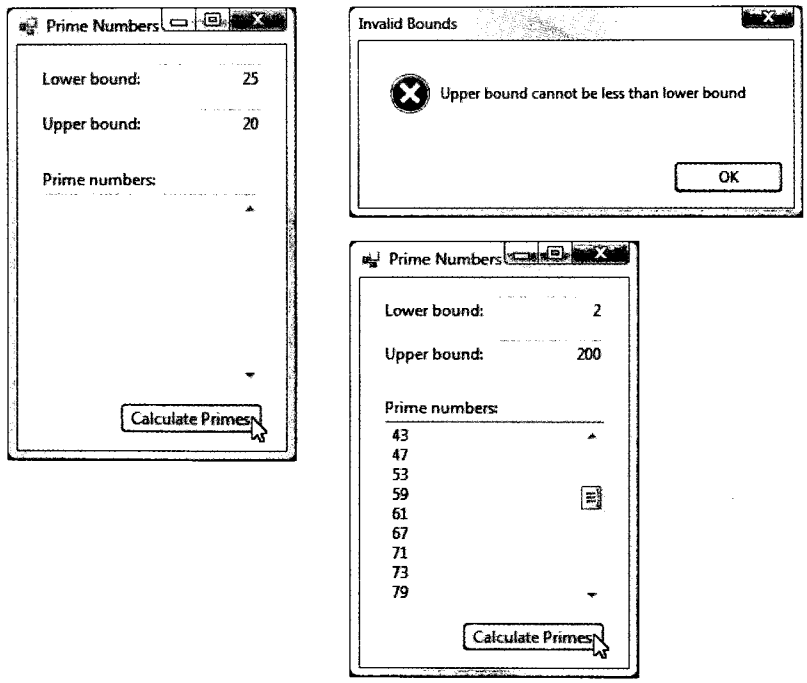
f) ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng. Nhập một số gallon và nhấn các `Button` **Regular**, **Special** và **Super +**. Sau khi nhấn mỗi `Button`, kiểm tra kết quả được hiển thị có đúng hay không dựa trên yêu cầu của ứng dụng.

g) ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

h) ***Đóng IDE***. Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

Bài tập nâng cao ▶ **13.17** (***Ứng dụng Prime Numbers)*** Một số `Integer` lớn hơn 1 được gọi là nguyên tố nếu chỉ chia hết cho 1 và chính nó. Ví dụ, 2, 3, 5 và 7 là số nguyên tố, còn 4, 6, 8 và 9 không phải là số nguyên tố. Viết một ứng dụng nhận vào hai số (là giới hạn dưới và giới hạn trên) và tìm tất cả các số nguyên tố nằm trong giới hạn được xác định.

a) Tạo ứng dụng. Tạo ứng dụng có tên `PrimeNumbers` có giao diện giống như trên Hình 13.32. Tạo xử lý sự kiện cho sự kiện `Click` của `Button` **Calculate Primes.**



Chú thích hình

- ***Lower bound:*** *Giới hạn dưới*
- ***Upper bound:*** *Giới hạn trên*
- ***Prime numbers:*** *Các số nguyên tố*

Hình 13.32 Ứng dụng **Prime Numbers**.

b) ***Kiểm tra số nguyên tố.*** Viết thủ tục `Function Prime` trả về `True` nếu một số là số nguyên tố, `False` nếu ngược lại. Để xác định một số có phải là số nguyên tố hay không, viết một lệnh `For...Next` đếm từ 2 đến căn bậc hai của số đó. Trong thân vòng lặp, sử dụng toán tử `Mod` (Chương 6) để xác định số đó có chia hết cho biến điều khiển hay không (phần dư bằng 0). Nếu có, số đó không phải là số nguyên tố.

c) ***Giới hạn đầu vào của người dùng.*** Cho phép người dùng nhập giới hạn dưới (`lower`) và giới hạn trên (`upper`). Không cho phép người dùng nhập giới hạn dưới nhỏ hơn hoặc bằng 1 hoặc giới hạn trên nhỏ hơn giới hạn dưới.

d) ***Hiển thị các số nguyên tố.*** Gọi thủ tục `Function Prime` từ phần xử lý sự kiện để xác định số nào giữa giới hạn trên và giới hạn dưới là số nguyên tố. Sau đó xử lý sự kiện hiển thị các số nguyên tố trong `TextBox` nhiều dòng có thanh cuộn như trên Hình 13.32.

e) ***Chạy ứng dụng.*** Chọn **Debug > Start Debugging** để chạy ứng dụng. Nhập một giới hạn dưới và giới hạn trên nhỏ hơn giới hạn dưới. Nhấn `Button` **Calculate Primes**. Bạn sẽ nhận được thông báo lỗi. Nhập các giới hạn âm rồi nhấn `Button` **Calculate Primes**. Một lần nữa, bạn lại nhận được thông báo lỗi. Nhập các giới hạn hợp lệ và nhấn `Button` **Calculate Primes**. Lần này, các số nguyên tố nằm trong khoảng sẽ được hiển thị.

f) ***Đóng ứng dụng.*** Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

g) ***Đóng IDE.*** Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

CHƯƠNG 14

Ứng dụng Shipping Time

Sử dụng Date và Timer

Mục tiêu

Trong chương này, bạn sẽ tìm hiểu để:

- Tạo và thao tác với biến **Date**.
- Thực thi mã sau những khoảng thời gian cố định sử dụng điều khiển **Timer**.
- Lấy giá trị đầu vào kiểu **Date** từ điều khiển **DateTimePicker**.
- Nhóm các điều khiển bằng điều khiển **GroupBox**.

Nội dung chính

Nhiều công ty, từ công ty hàng không đến công ty hàng hải, đều dựa vào thông tin về thời gian để thực hiện công việc hàng ngày của họ. Các công ty này thường đòi hỏi những ứng dụng thực hiện tính toán về thời gian đáng tin cậy. Trong chương này, bạn sẽ tạo ứng dụng thực thi việc tính toán với kiểu `Date` cơ sở - là kiểu dữ liệu cho phép bạn lưu và thao tác thông tin về thời gian. Bạn cũng sẽ tìm hiểu cách sử dụng điều khiển `DateTimePicker` để nhận thông tin về thời gian từ người dùng. Cuối cùng, bạn sẽ học cách sử dụng `Timer` - một điều khiển của Windows `Forms` thực thi mã nguồn sau những khoảng thời gian định kỳ.

14.1 Chạy thử ứng dụng Shipping Time

Trong chương này, bạn xây dựng ứng dụng **Shipping Time**. Ứng dụng này phải đáp ứng những yêu cầu sau đây:

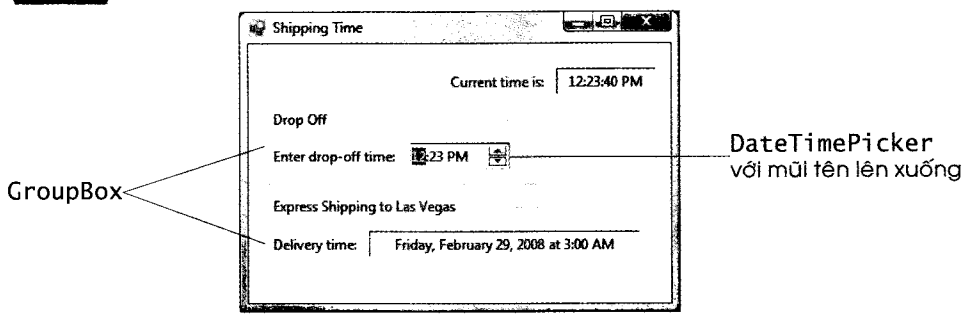
Yêu cầu đối với ứng dụng

Một nhà phân phối hải sản đã yêu cầu bạn tạo ứng dụng tính thời gian giao hàng hải sản tươi sống được chuyển từ Portland, Maine đến các trung tâm phân phối tại Las Vegas, Nevada - là những nơi chỉ chấp nhận hải sản vẫn còn tươi. Nhà phân phối đã thỏa thận với hãng hàng không địa phương để bảo đảm rằng hải sản được vận chuyển trên các chuyến bay cất cánh vào buổi trưa hoặc nửa đêm. Tuy nhiên, vì những lý do an ninh, sân bay yêu cầu nhà phân phối chuyển hàng hóa đến sân bay ít nhất là một giờ trước khi máy bay cất cánh. Khi nhà phân phối đưa ra thời gian chuyển hàng đến sân bay, ứng dụng phải hiển thị thời gian giao hàng ở Las Vegas. Ứng dụng này phải tính đến sự chênh lệch thời gian ba giờ đồng hồ (thời gian ở Las Vegas sớm hơn ba giờ so với địa điểm giao hàng) và thời gian bay sáu giờ giữa hai thành phố. Ứng dụng cho phép người dùng chọn thời gian chuyển hàng đến sân bay trong ngày hiện tại (hải sản phải được chuyển đi trong ngày để đảm bảo vẫn còn tươi sống). Ứng dụng cũng cần có đồng hồ đang chạy để hiển thị thời gian hiện thời.

Ứng dụng này tính toán thời gian giao hàng từ thời gian thả hàng tại sân bay có tính đến nhân tố thời gian vận chuyển và múi giờ. Bạn sử dụng điều khiển `DateTimePicker` để cho phép người dùng nhập thời gian thả hàng và sử dụng thuộc tính, phương thức của `Date` để tính toán thời gian giao hàng. Bạn bắt đầu bằng việc chạy thử ứng dụng đã hoàn thiện. Sau đó, bạn tìm hiểu những tính năng bổ sungcần thiết của Visual Basic để xây dựng cho mình một phiên bản của ứng dụng này.

Chạy thử ứng dụng Shipping Time đã được nâng cấp

1. ***Mở ứng dụng hoàn thiện.*** Mở thư mục `C:\Examples\Tutorial14\CompleteApplication\ShippingTime` để tìm ứng dụng **Shipping Time**. Nhấn đúp vào file `ShippingTime.sln` để mở ứng dụng trong Visual Basic IDE.
2. ***Chạy ứng dụng Shipping Time.*** Chọn **Debug > Start Debugging** để chạy ứng dụng (Hình 14.1).



Chú thích hình

- ***Current time is:*** *Thời gian hiện tại*
- ***Drop off time:*** *Thời gian thả hàng*
- ***Delivery time:*** *Thời gian giao hàng*

Hình 14.1 Ứng dụng **Shipping Time**.

3. ***Nhập thời gian thả hàng.*** Thời gian thả hàng mặc định được thiết lập là thời gian hiện thời của máy tính khi bạn chạy ứng dụng. Khi bạn thay đổi thời gian thả hàng, `Label` **Delivery time:** hiển thị thời gian giao hàng dựa trên thời gian mới. Lưu ý rằng nếu bạn chọn thời gian trước 11:00 AM., thì hàng hóa đến Las Vegas vào 3:00 PM. Nếu bạn chỉ định thời gian thả hàng ở khoảng giữa 11:00 AM và 11:00 PM, thì hàng hóa đến Las Vegas vào 3:00 AM. ngày hôm sau. Còn nếu bạn chỉ ra thời gian thả hàng sau 11:00 PM, thì hàng hóa sẽ đến vào 3:00 PM. ngày hôm sau.

   Thời gian hiển thị trong `Label` **Current time is:** sẽ được cập nhật thời gian hiện thời sau mỗi giây. Tuy nhiên, thời gian thả hàng được hiển thị trong `DateTimePicker` chỉ thay đổi nếu bạn chọn giá trị khác bằng cách sử dụng các mũi tên lên xuống hoặc nhập giá trị mới từ bàn phím.
4. ***Đóng ứng dụng.*** Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.
5. ***Đóng IDE.*** Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

14.2 Biến Date

Mẹo tránh lỗi

Luôn lưu ngày tháng trong biến `Date`. Lưu ngày tháng trong các biến có kiểu dữ liệu khác có thể gây ra lỗi hoặc làm mất dữ liệu.

Chọn kiểu dữ liệu hợp lý để lưu trữ thông tin có thể giảm bớt thời gian phát triển ứng dụng vì việc viết mã sẽ đơn giản hơn. Chẳng hạn, nếu bạn đang sử dụng số nguyên, biến kiểu `Integer` là lựa chọn tốt nhất cho bạn; nếu bạn cần lưu giá trị tiền tệ, bạn nên sử dụng các biến có kiểu `Decimal`. Nếu bạn muốn lưu thông tin thời gian (như ngày, tháng, năm, giờ, phút, giây ...) bạn có thể dùng từng biến riêng biệt để theo dõi tháng, ngày trong tuần, năm và những thông tin liên quan đến ngày tháng khác. Tuy nhiên, cách làm này sẽ là nhiệm vụ phức tạp và làm chậm quá trình phát triển những ứng dụng đòi hỏi nhiều thông tin về thời gian.

Khai báo một biến Date

Kiểu Date cơ bản sẽ đơn giản hóa việc thao tác, lưu trữ và hiển thị thông tin thời gian. Date là từ khóa của Visual Basic tương ứng với kiểu **DateTime** trong .NET Framework Class Library - chúng có thể được sử dụng thay cho nhau. Một biến **Date** lưu thông tin về một thời điểm (chẳng hạn, 12:00:00 AM. ngày 01 tháng 1, năm 2008). Bạn có thể truy cập các thuộc tính của một biến Date như ngày, giờ và phút bằng cách viết mã. Ứng dụng **Shipping Time** cần phép tính toán liên quan tới thời gian, vì vậy bạn dùng biến Date để lưu giữ và thao tác với những thông tin này.

Bạn sử dụng từ khóa New khi tạo giá trị Date. Trong dòng mã sau, lệnh

hàm khởi tạo của đối tượng Date

biến Date

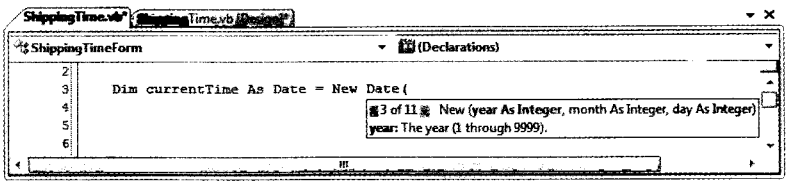
```
Dim delivery As Date = New Date(2003, 1, 1, 0, 0, 0)
```

khai báo một biến Date mới có tên delivery. Từ khóa New gọi hàm khởi tạo của Date. **Hàm khởi tạo (constructor)** là thủ tục khởi tạo giá trị cho đối tượng khi đối tượng này được tạo. Bạn sẽ học cách viết hàm khởi tạo trong Chương 19. Lưu ý rằng hàm khởi tạo trên lấy 6 đối số - year, month, day, hour, minute và second. Các giá trị này được mô tả trên Hình 14.2.

Đối số	Dãy các giá trị	Mô tả
Khởi tạo một biến Date sử dụng **New Date** *(year, month, day, hour, minute, second)*		
year	Các giá trị Integer 1-9999	Năm
month	Các giá trị Integer 1-12	Tháng của năm
day	Các giá trị Integer 1-*số ngày trong tháng*	Ngày của tháng. Mỗi tháng có 28 đến 31 ngày tùy thuộc vào tháng nào trong năm.
hour	Các giá trị Integer 0-23	Giờ trong ngày trên đồng hồ 24 giờ. Giá trị 0 đại diện cho 12:00 AM.
minute	Các giá trị Integer 0-59	Phút của giờ
second	Các giá trị Integer 0-59	Số giây đã trôi qua của phút hiện thời

Hình 14.2 Các đối số của hàm khởi tạo Date.

Kiểu Date thực ra có nhiều hàm khởi tạo nạp chồng. Nạp chồng phương thức (**method overloading**) cho phép bạn tạo ra nhiều phương thức có cùng tên nhưng khác về chữ ký (**signature**) - tức là khác nhau về số lượng và kiểu tham số hoặc khác nhau về trật tự kiểu của danh sách tham số. Khi phương thức nạp chồng được gọi, trình biên dịch sẽ chọn phương thức phù hợp bằng cách kiểm tra số lượng, kiểu và trật tự kiểu của danh sách đối số. Thường thì nạp chồng phương thức dùng để tạo một số phương thức có cùng tên, thực hiện những nhiệm vụ tương tự trên những tập hợp tham số khác nhau (ví dụ, nhiều hàm khởi tạo cho phép khởi tạo đối tượng theo nhiều cách khác nhau). Nếu một kiểu cung cấp nhiều hàm khởi tạo hoặc phương thức được nạp chồng, *IntelliSense* hiển thị tooltip có chứa một trong những lựa chọn nạp chồng có thể sử dụng. Bạn có thể xem các lựa chọn khác bằng cách nhấn vào bất cứ chỗ nào trên tooltip. Tooltip này xuất hiện sau khi bạn gõ phím ký tự mở ngoặc đơn sau tên của hàm khởi tạo hoặc phương thức. Hình 14.3 thể hiện hàm khởi tạo nạp chồng của kiểu Date có ba đối số - year, month và day. Như bạn có thể thấy trong tooltip, đây là một trong 11 hàm khởi tạo nạp chồng là của đối tượng Date.



Hình 14.3 *IntelliSense* hiển thị hàm khởi tạo nạp chồng của đối tượng Date.

Sử dụng các thành viên của đối tượng Date

Sau khi gán giá trị cho biến Date, bạn có thể truy cập các thuộc tính của biến bằng toán tử truy cập thành viên (dấu chấm) như dưới đây:

```
Dim year = delivery.Year ' lấy giá trị năm của biến Date delivery
Dim month = delivery.Month ' lấy giá trị tháng của biến Date delivery
Dim day = delivery.Day ' lấy giá trị ngày của biến Date delivery
Dim hour = delivery.Hour ' lấy giá trị giờ của biến Date delivery
Dim minute = delivery.Minute ' lấy giá trị phút của biến Date delivery
Dim second = delivery.Second ' lấy giá trị giây của biến Date delivery
```

Trong chương này, bạn sẽ dùng một vài thuộc tính và phương thức Date được truy cập thông qua toán tử truy cập thành viên.

Không thể thực hiện phép cộng cho biến Date giống như các kiểu dữ liệu số cơ sở như Integer hay Decimal - tuy nhiên bạn có thể so sánh Date bằng các toán tử quan hệ như đối với dữ liệu số. Thay cho việc sử dụng toán tử số học để thêm hoặc bớt giá trị của biến Date, bạn phải gọi phương thức bằng toán tử truy cập thành viên. Hình 14.4 minh họa thực thi các phép tính toán khác nhau với biến Date.

Lỗi lập trình thường gặp

Các phương thức của Date không thay đổi giá trị biến Date gọi chúng. Bạn phải gán kết quả của phương thức cho một biến kiểu Date để lưu giá trị sau khi thay đổi.

Lệnh Visual Basic 2008	Kết quả
Giả sử rằng delivery đã được khởi tạo với một giá trị Date	
`delivery = delivery.AddHours(3)`	Cộng thêm 3 giờ
`delivery = delivery.AddMinutes(-5)`	Trừ đi 5 phút.
`delivery = delivery.AddDays(1)`	Cộng thêm 1 ngày.
`delivery = delivery.AddMinutes(30)`	Cộng thêm 30 phút.
`delivery = delivery.AddHours(-12)`	Trừ đi 12 giờ

Hình 14.4 Các phương thức của biến Date thực hiện các phép tính toán khác nhau.

Lưu ý rằng phương thức "add" không làm thay đổi giá trị của biến Date gọi nó, mà mỗi phương thức "add" trả về một giá trị Date có chứa kết quả của phép tính toán. Để thay đổi giá trị của biến Date delivery, bạn phải gán cho biến delivery giá trị trả về của phương thức "add".

Visual Basic cung cấp một cách đơn giản để gán ngày và giờ hiện thời cho biến Date. Bạn có thể sử dụng thuộc tính **Now** để gán ngày và giờ hiện thời của máy tính vào biến Date:

```
Dim currentTime As Date = Date.Now
```

Lưu ý rằng phép gán này không cần từ khóa New vì thuộc tính Date.Now trả về giá trị Date. Cũng giống như phương thức MessageBox.Show và String.Format, bạn có thể truy cập thuộc tính Now của kiểu Date bằng cách viết toán tử truy cập thành viên và tên thuộc tính sau tên kiểu. Các phương thức và thuộc tính có thể được truy cập thông qua tên kiểu, thay vì qua một biến có kiểu đó, được gọi là thành viên chia sẻ (shared member). Tài liệu MSDN trực tuyến sử dụng ký hiệu **S** để chỉ thành viên chia sẻ.

Bây giờ khi đã làm quen với biến `Date`, bạn thiết kế ứng dụng **Shipping Time** bằng cách sử dụng hai điều khiển mới - điều khiển `GroupBox` và điều khiển `DateTimePicker`. Điều khiển **`GroupBox`** nhóm các điều khiển liên quan với nhau một cách trực quan bằng cách vẽ một hình chữ nhật bao quanh các điều khiển. `GroupBox` đặc biệt hữu dụng trong việc nhóm các lựa chọn liên quan đến nhau, ví dụ như các `CheckBox`. Điều khiển **`DateTimePicker`** cho phép người dùng nhập thông tin thời gian.

Tự ôn tập

1. Bạn có thể sử dụng phương thức_____để trừ một giá trị `Date` đi 2 ngày.
   a) `SubtractDays` b) `AddDays`
   c) `SubDays` d) `SubtractTime`
2. Phương thức của `Date` thực hiện tính toán_____.
   a) trả về giá trị `Date` mới b) thay đổi giá trị `Date`
   c) không trả về giá trị d) a hoặc b

Đáp án: **1)** b. **2)** a.

14.3 Tạo ứng dụng Shipping Time: Thiết kế thành phần

Bây giờ bạn đã sẵn sàng để bắt đầu phân tích vấn đề và xây dựng mã giả cho ứng dụng. Đoạn mã giả sau đây mô tả các thao tác cơ bản của ứng dụng **Shipping Time**:

```
Khi Form được load:
    Thiết lập khoảng thời gian thả hàng là thời điểm chạy ứng dụng
    Gọi thủ tục sub DisplayDeliveryTime để xác định và hiển thị thời gian
        giao hàng

Khi người dùng thay đổi thời gian thả hàng:
    Gọi thủ tục sub DisplayDeliveryTime để xác định và hiển thị thời gian
        giao hàng

Sau khi một giây trôi qua:
    Cập nhật và hiển thị thời gian hiện thời

Khi thủ tục DisplayDeliveryTime được gọi:
    Gọi hàm DepartureTime để xác định thời gian chuyến bay chuyển
        hàng cất cánh
    Thêm ba giờ để xác định thời gian giao hàng (vì phải tính đến 6 giờ
        bay và trừ đi 3 giờ do chênh lệch múi giờ)
    Hiển thị thời gian giao hàng

Khi thủ tục DepartureTime được gọi:
    Chọn trường hợp phù hợp dựa vào thời gian thả hàng
        Trong trường hợp thời gian thả hàng từ 0 đến 10h
            Hàng sẽ được chuyển trên chuyến bay trưa cùng ngày
        Trong trường hợp thời gian thả hàng là 23h
            Hàng sẽ được chuyển trên chuyến bay trưa của ngày tiếp theo
        Nếu không trường hợp nào ở trên đúng
            Hàng sẽ được chuyển trên chuyến bay nửa đêm cùng ngày
```

Bây giờ khi đã chạy thử ứng dụng **Shipping Time** và tìm hiểu mã giả của ứng dụng, bạn hãy sử dụng bảng ACE để chuyển đổi mã giả thành mã Visual Basic. Hình 14.5 liệt kê những hành động, điều khiển, sự kiện giúp bạn tự hoàn thiện cho mình một phiên bản của ứng dụng này.

Bảng ACE cho ứng dụng Shipping Time

Hành động	Điều khiển	Sự kiện/ Phương thức
Hiển thị Label cho các điều khiển của ứng dụng	currentTimeIsLabel, dropOffLabel, deliveryTimeLabel	Ứng dụng được chạy
	ShippingTime Form	Load
Thiết lập khoảng thời gian thả hàng là thời điểm chạy ứng dụng	dropOff-DateTimePicker	
Gọi thủ tục sub DisplayDeliveryTime để xác định và hiển thị thời gian giao hàng	dropOff-DateTimePicker, lasVegasTimeLabel	
	dropOff-DateTimePicker	ValueChanged
Gọi thủ tục sub DisplayDeliveryTime để xác định và hiển thị thời gian giao hàng	dropOff-DateTimePicker lasVegasTimeLabel	
	clockTimer	Tick
Cập nhật và hiển thị thời gian hiện thời	currentTimeLabel	
		Display-DeliverTime
Gọi hàm DepartureTime để xác định thời gian chuyến bay chuyển hàng cất cánh	dropOff-DateTimePicker	
Thêm ba giờ để xác định thời gian giao hàng		
Hiển thị thời gian giao hàng	lasVegasTimeLabel	
		Departure-Time
Chọn trường hợp phù hợp dựa vào thời gian thả hàng	dropOff-DateTimePicker	
Trong trường hợp thời gian thả hàng từ 0 đến 10h		
Hàng sẽ được chuyển trên chuyến bay trưa cùng ngày		
Trong trường hợp thời gian thả hàng là 23h		
Hàng sẽ được chuyển trên chuyến bay trưa của ngày tiếp theo		
Nếu không trường hợp nào ở trên đúng		
Hàng sẽ được chuyển trên chuyến bay nửa đêm cùng ngày		

Hình 14.5 Bảng ACE cho ứng dụng **Shipping Time.**

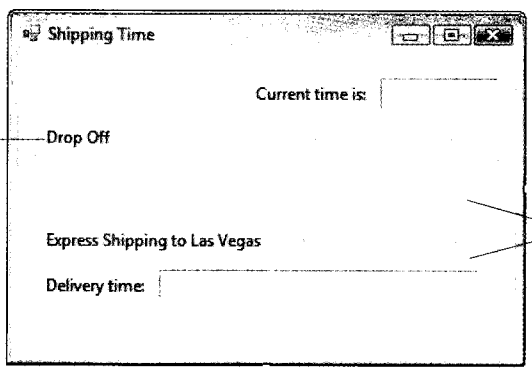
Phần tiếp sau sẽ hướng dẫn cách chèn điều khiển GroupBox vào ứng dụng.

Đặt các điều khiển vào GroupBox

1. ***Copy template của ứng dụng vào thư mục làm việc.*** Copy thư mục `C:\Examples\Tutorial14\TemplateApplication\ShippingTime` vào thư mục `C:\SimplyVB2008`.
2. ***Mở file template của ứng dụng Shipping Time.*** Nhấn đúp vào file `ShippingTime.sln` trong thư mục `ShippingTime` để mở ứng dụng trong Visual Basic IDE.
3. ***Hiển thị Form template.*** Nhấn đúp vào `ShippingTime.vb` trong cửa sổ **Solution Explorer** để hiển thị Form trong IDE.
4. ***Thêm điều khiển GroupBox vào Form.*** Template đã có GroupBox hiển thị thời gian giao chuyến hàng hải sản tươi sống. Thêm một GroupBox thứ hai để chứa thời gian thả hàng bằng cách nhấn đúp vào điều khiển GroupBox

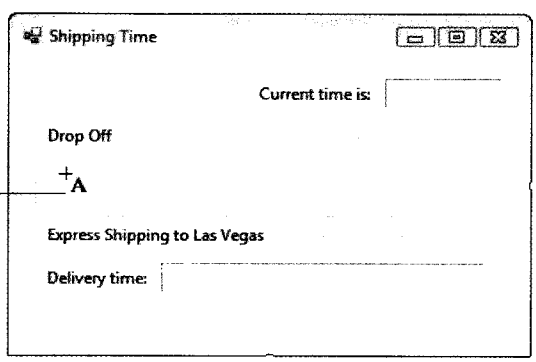
   GroupBox

   trong tab **Container** của **Toolbox**. Thay đổi thuộc tính `Text` thành `DropOff` và thuộc tính `Name` thành `dropOffGroupBox`. Đặt GroupBox bên trên GroupBox đã có và chỉnh sửa để chúng có cùng kích thước. Sau những thay đổi này, Form của bạn trông sẽ giống như Hình 14.6.



Hình 14.6 Điều khiển GroupBox trên Form **Shipping Time**.

5. ***Tạo Label trong GroupBox.*** Để thêm Label vào trong GroupBox, nhấn vào điều khiển Label trong **Toolbox**, sau đó nhấn vào bên trong GroupBox (Hình 14.7). Thay đổi thuộc tính `Text` của Label thành `Enter drop-off time:` và thuộc tính `Name` thành `dropOffLabel`. Sau đó thay đổi vị trí của Label bằng cách thiết lập giá trị thuộc tính `Location` bằng 6, 33 - giá trị này canh Label mới tạo thẳng hàng với Label **Delivery time:**.



Hình 14.7 Thêm Label vào GroupBox.

Mẹo thiết kế giao diện

Tiêu đề của GroupBox nên súc tích và sử dụng kiểu viết hoa tiêu đề sách.

Thói quen lập trình tốt

Thêm hậu tố GroupBox vào sau tên của điều khiển GroupBox.

Mẹo thiết kế giao diện

Sử dụng GroupBox để nhóm các điều khiển có liên quan trong cùng một hình chữ nhật có tiêu đề.

Lưu ý rằng các giá trị Location bạn nhập vào được tính từ góc trên bên trái của GroupBox chứ không phải của Form. Các đối tượng như Form, GroupBox và Panel (bạn sẽ dùng trong Chương 19) được gọi là **đối tượng chứa (container)**. Location của điều khiển được tính từ góc trên bên trái của đối tượng chứa nó.

Nếu GroupBox được đặt ở trên điều khiển đã có mặt trong Form, điều khiển sẽ ở phía sau GroupBox (nghĩa là, GroupBox che điều khiển bằng cách đè lên nó). Để tránh điều này, hãy xóa tất cả các điều khiển trong khu vực bạn muốn đặt điều khiển GroupBox trước khi chèn nó vào. Bạn cũng có thể kéo thả điều khiển đã có vào GroupBox hoặc thêm điều khiển cần thiết theo cách được mô tả ở trên.

6. ***Lưu project.*** Chọn **File > Save All** để lưu mã đã được sửa đổi.

Đến đây, bạn vừa thêm GroupBox và Label vào ứng dụng **Shipping Time** để hiển thị thời gian thả hàng. Trong phần tiếp theo, bạn sẽ thêm điều khiển DateTimePicker để lấy giá trị thời gian thả hàng từ người dùng.

Nhớ lại rằng DateTimePicker lấy về thông tin thời gian từ người dùng. DateTimePicker cho phép bạn chọn định dạng thời gian đã được định nghĩa trước (chẳng hạn, các định dạng ngày tháng như 12/31/2008 hay Friday, December 31, 2008; và định dạng giờ giấc như 2:00:00 PM) hoặc bạn có thể tự tạo định dạng của mình. Thông tin thời gian được lưu trong biến kiểu Date và bạn có thể thao tác trên đó bằng các phương thức của biến Date. Lưu ý rằng định dạng khác nhau sẽ giới hạn thông tin cho phép người dùng nhìn thấy nhưng không thay đổi giá trị Date lưu giữ trong DateTimePicker.

Tạo và tùy chỉnh DateTimePicker

Mẹo thiết kế giao diện

Mỗi DateTimePicker nên có một Label mô tả tương ứng.

Thói quen lập trình tốt

Thêm hậu tố DateTimePicker vào sau tên của điều khiển DateTimePicker.

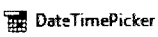
Mẹo thiết kế giao diện

Sử dụng DateTimePicker để lấy thông tin về ngày tháng và thời gian từ người dùng.

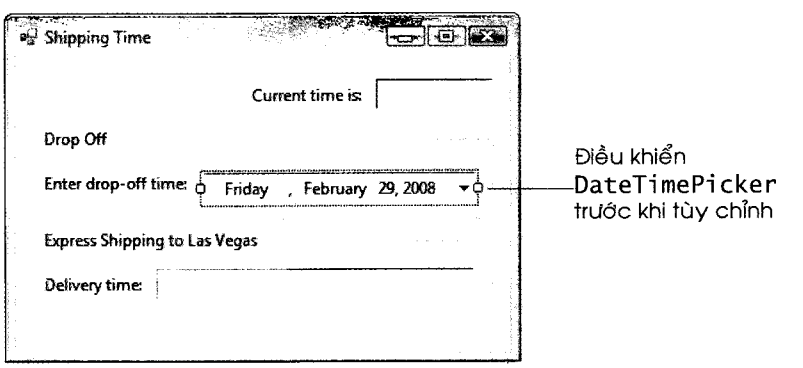
Mẹo tránh lỗi

Hãy cẩn thận khi sử dụng thuộc tính CustomFormat để định nghĩa cách hiển thị của DateTimePicker. Định dạng có thể khác đi tùy vào thiết lập vị trí cho máy tính của bạn.

1. ***Thêm DateTimePicker.*** Để thêm DateTimePicker vào ứng dụng, kéo điều khiển DateTimePicker



từ **Toolbox** vào bên phải của Label **Enter drop-off time:** để đặt DateTimePicker vào trong GroupBox. Form sẽ trông giống như Hình 14.8 (Điều khiển sẽ chứa thời gian hiện thời của máy tính).



Hình 14.8 Điều khiển DateTimePicker trên Form.

2. ***Tùy chỉnh DateTimePicker.*** Chọn DateTimePicker, thay đổi thuộc tính Name thành dropOffDateTimePicker.

   Canh lề DateTimePicker với Label mô tả tương ứng của nó. Tiếp theo, thay đổi thuộc tính **Format** thành Custom. Giá trị này chỉ ra rằng bạn sẽ tự định nghĩa cách ngày tháng xuất hiện trong DateTimePicker.

3. ***Tự định nghĩa cho định dạng hiển thị.*** Khi thuộc tính Format của DateTimePicker được đặt là Custom, điều khiển sẽ hiển thị thời gian theo định dạng tùy chỉnh mà bạn định nghĩa trong thuộc tính **CustomFormat**

Mẹo tránh lỗi

Nếu người dùng nhập thời gian vào, hãy sử dụng điều khiển `DateTimePicker` để ngăn cản người dùng nhập những giá trị không hợp lệ.

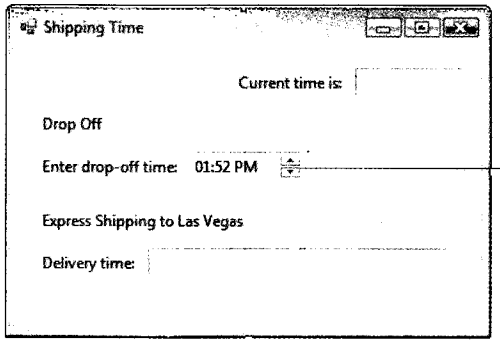
và thuộc tính CustomFormat chưa được gán giá trị. Lưu ý rằng bây giờ `DateTimePicker` hiển thị ngày tháng theo định dạng 1/1/2008, là mặc định khi thuộc tính `Format` được đặt giá trị `Custom` và thuộc tính `CustomFormat` chưa được gán giá trị.

Thay đổi giá trị của `CustomFormat` thành `hh:mm tt`. Lưu ý rằng thuộc tính `CustomFormat` có phân biệt chữ hoa chữ thường. "hh" hiển thị giờ là một số từ 01 đến 12, "mm" chỉ ra rằng số phút từ 00 đến 59 đứng sau dấu hai chấm. "tt" chỉ ra rằng AM hoặc PM sẽ xuất hiện tùy thuộc vào thời gian trong ngày. Bạn có thể tìm thấy danh sách định dạng ngày và giờ trên http://msdn2.microsoft.com/en-us/library/8kb3ddd4.aspx. Lưu ý rằng thuộc tính `Format` để tránh người dùng nhập vào chữ cái hay ký hiệu trong khi ứng dụng yêu cầu số - `DateTimePicker` không cho phép các giá trị khác ngoài những giá trị bạn đã chỉ ra trong thuộc tính `Format` hoặc `CustomFormat`. `DateTimePicker` cũng ngăn chặn người dùng chỉ ra thời gian không hợp lệ, chẳng hạn `32:15`. Hãy thay đổi kích thước `DateTimePicker` để phù hợp với định dạng vừa được định nghĩa này.

Mẹo thiết kế giao diện

Nếu chỉ muốn người dùng chỉ ra giờ giấc trong ngày, hãy thiết lập giá trị thuộc tính ShowUpDown của `DateTimePicker` là `True`. Nếu muốn người dùng chỉ ra giờ giấc lẫn ngày tháng, hãy thiết lập giá trị thuộc tính ShowUpDown của `DateTimePicker` là `False` để cho phép người dùng chọn ngày trong lịch tháng đổ xuống.

4. ***Sử dụng các mũi tên lên xuống trên DateTimePicker.*** Thiết lập giá trị `True` cho thuộc tính **ShowUpDown** của `DateTimePicker`. Thiết lập như thế này cho phép người dùng chọn ngày giờ bằng cách nhấn các mũi tên lên xuống xuất hiện bên phải điều khiển, giống với điều khiển `NumericUpDown`. Khi thuộc tính này được đặt là `False` (mặc định) thì một mũi tên trỏ xuống xuất hiện bên tay phải của điều khiển (Hình 14.8). Nhấn mũi tên này thì lịch tháng sẽ xuất hiện cho phép người dùng chọn một ngày (nhưng không chọn được thời gian). Hình minh họa lịch theo tháng được thể hiện trong phần Điều khiển, Sự kiện và Phương thức ở cuối chương này. Trong ứng dụng này, người dùng cần nhập thời gian trong ngày cho nên bạn thiết lập giá trị thuộc tính là `True` để cho phép người dùng chọn thời gian bằng cách nhấn vào mũi tên lên xuống (Hình 14.9).



Hình 14.9 Điều khiển `DateTimePicker` đã được thay đổi trên `Form`.

5. ***Lưu project.*** Chọn **File > Save All** để lưu mã đã được sửa đổi.

Điều khiển cuối cùng được bạn thêm vào `Form` là `Timer`. Bạn sử dụng điều khiển `Timer` để khởi tạo sự kiện giúp bạn hiển thị thời gian hiện thời trên `Form`.

Tạo điều khiển Timer

1. ***Thêm điều khiển Timer.*** Điều khiển `Timer` là đối tượng có thể thực thi mã sau mỗi mili giây (1/1000 giây) bằng cách tạo sự kiện **`Tick`**. Theo mặc định, `Timer` chạy mã sau mỗi 100 mili giây (1/10 giây). Mỗi lần sự kiện `Tick` xảy ra, xử lý sự kiện của nó được thực thi. Bạn có thể thay đổi "chu kỳ thức dậy" (thời gian giữa hai sự kiện `Timer Tick`) và mã sẽ thực thi (xử lý sự kiện cho sự kiện `Tick`) sao cho nhiệm vụ cụ thể được thực thi cho mỗi lần "thức dậy".

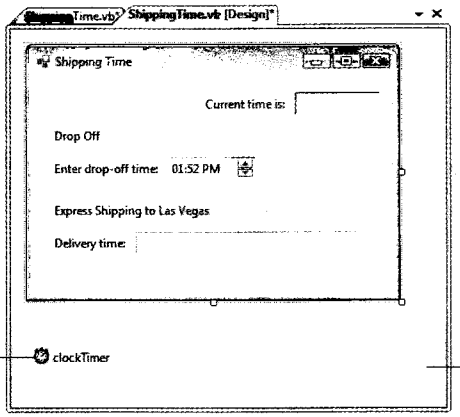
Thói quen lập trình tốt

Thêm hậu tố `Timer` vào sau tên của điều khiển `Timer`.

Thêm `Timer` vào `Form` bằng cách nhấn vào điều khiển `Timer`,

trong tab **Components** của **Toolbox** và kéo thả vào bất kỳ chỗ nào trên `Form`. Lưu ý rằng `Timer` không thực sự xuất hiện trên `Form` - nó xuất hiện trong khu vực bên dưới giao diện thiết kế `Form` được gọi là **khay thành phần** (Hình 14.10). Điều khiển `Timer` được đặt trong khay thành phần vì nó không phải là một phần của giao diện người dùng đồ họa - người dùng không bao giờ nhìn thấy điều khiển `Timer`.

2. ***Tùy chỉnh điều khiển Timer.*** Đặt lại tên cho `Timer` bằng cách thay đổi giá trị thuộc tính `Name` thành `ClockTimer`. Để cho phép `Timer` kích hoạt sự kiện `Tick`, thiết lập thuộc tính `Enabled` của `Timer` giá trị là `True`. Sau đó thiết lập giá trị thuộc tính **`Interval`** là 1000, giá trị này chỉ ra rằng số mili giây giữa các sự kiện `Tick` là 1000 mili giây = 1 giây.
3. ***Lưu project.*** Chọn **File > Save All** để lưu mã đã được sửa đổi.



Hình 14.10 Điều khiển `Timer` hiển thị trong khay thành phần.

TỰ ÔN TẬP

1. Theo mặc định, điều khiển `Timer` kích hoạt sự kiện `Tick` sau mỗi_____.
   a) giây b) 100 giây
   c) 100 mili giây d) nửa giây
2. Đặt giá trị_____cho thuộc tính `Format` của điều khiển `DateTimePicker` để chỉ ra cách ngày tháng xuất hiện trên `DateTimePicker`.
   a) `Custom` b) `Unique`
   c) `User` d) `Other`

Đáp án: 1) c. 2) a.

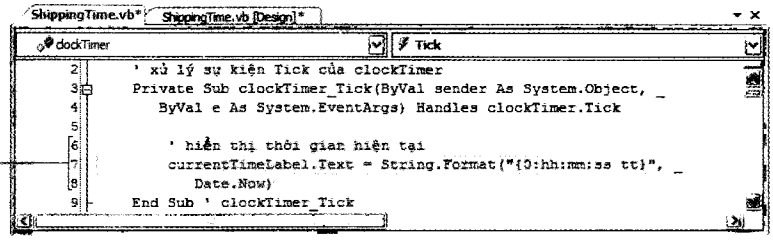
14.4 Tạo ứng dụng Shipping Time: Viết mã

Bây giờ khi đã hoàn chỉnh thiết kế giao diện cho ứng dụng **Shipping Time**, bạn sẽ hoàn chỉnh ứng dụng bằng việc viết mã. Bạn bắt đầu viết mã cho tính năng của ứng dụng bằng cách tạo đồng hồ trong ứng dụng dùng để cập nhật thời gian hiện thời sau mỗi giây. Sau đó bạn viết mã hiển thị thời gian giao hàng từ Portland đến Las Vegas. Bạn xây dựng tính năng này bằng cách thêm đoạn mã sẽ được thực thi khi `Form` được load hoặc bất cứ khi nào người dùng chỉ ra thời gian thả hàng mới. Trong phần tiếp theo, bạn sẽ viết mã để tạo đồng hồ.

Viết mã tạo đồng hồ cho ứng dụng Shipping Time

1. ***Thêm mã để xử lý sự kiện Tick của Timer.*** Nhấn đúp vào điều khiển `Timer` trong khay thành phần để tạo xử lý sự kiện rỗng cho sự kiện `Tick`. Thêm dòng 6-8 trên Hình 14.11 vào thân xử lý sự kiện. Hãy chắc chắn rằng xử lý sự kiện có định dạng như ở Hình 14.11 để bảo đảm rằng số dòng của bạn khớp với số dòng trên hình.

Hiển thị thời gian hiện thời



Hình 14.11 Thêm mã vào sự kiện `Tick`.

Dòng 3-9 định nghĩa xử lý sự kiện `Tick` thực thi một lần sau mỗi giây. Thuộc tính `Now` của `Date` lấy về thời gian hiện thời của máy tính. Xử lý sự kiện lấy thông tin này và định dạng nó theo khuôn mẫu "`{hh:mm:ss tt}`" sau đó gán giá trị này cho thuộc tính `Text` của `currentTimeLabel` để hiển thị cho người dùng. Nhớ lại rằng trong phương thức `String.Format`, 0 tương ứng với đối số sẽ được định dạng (nghĩa là, `Date.Now`) và văn bản đứng sau dấu hai chấm chứa thông tin định dạng cho đối số đó. Bạn đã quen với ký hiệu `hh:mm` và `tt`. Chuỗi `:ss` đứng sau `mm` chỉ ra rằng có một dấu hai chấm đứng trước số giây (00-59) sẽ được hiển thị.

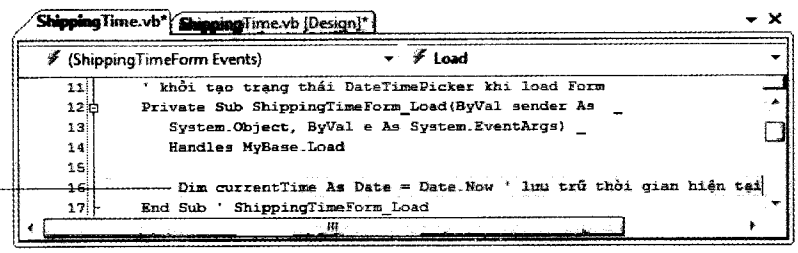
2. ***Lưu project.*** Chọn **File > Save All** để lưu mã đã được thay đổi.

Bây giờ khi đã viết mã cho đồng hồ của ứng dụng sử dụng sự kiện `Tick` của `Timer`, bạn sẽ viết mã để hiển thị thời gian giao hàng khi ứng dụng được mở. Bạn bắt đầu với việc tạo xử lý sự kiện cho sự kiện `Load` của ứng dụng.

Viết mã để hiển thị thời gian giao hàng

1. ***Thêm xử lý sự kiện ShippingTimeForm_Load.*** Khi ứng dụng chạy, `Form` được hiển thị. Tuy nhiên, đôi khi bạn cũng muốn thực hiện một số hành động đặc biệt khi ứng dụng được mở nhưng trước khi `Form` hiển thị. Để thực thi mã khi ứng dụng mở, hãy tạo xử lý sự kiện cho sự kiện **Load** của `Form`. Để tạo xử lý sự kiện cho sự kiện `Load`, hãy quay lại Windows `Form` Designer bằng cách nhấn vào tab **ShippingTime.vb[Design]**. Nhấn đúp vào một vùng trống trong `Form` để tạo xử lý sự kiện `Load`. Hãy thận trọng để không nhấn đúp vào điều khiển trên `Form`; việc này sẽ kích hoạt xử lý sự kiện của điều khiển đó. Bạn có thể nhấn đúp vào thanh tiêu đề của `Form` để đảm bảo rằng bạn không lỡ tay tạo xử lý sự kiện cho điều khiển khác.
2. ***Lưu ngày tháng hiện thời.*** Thêm dòng 16 trên Hình 14.12 vào xử lý sự kiện `Load` để lưu ngày tháng hiện thời vào biến `currentTime`. (Bạn lưu thông tin này để sau đó sử dụng trong xử lý sự kiện.) Hãy chắc chắn rằng bạn đã thêm các ghi chú và các ký tự nối dòng như trên Hình 14.12 để số dòng trong mã của bạn khớp với số dòng được hiển thị trong chương này.

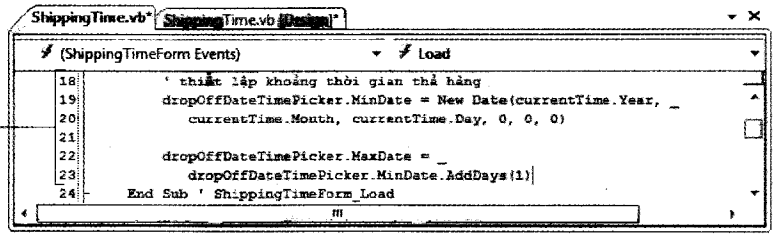
Lưu thời gian hiện thời trong `currentTime`



Hình 14.12 Lưu thời gian hiện thời.

3. ***Thiết lập giờ thả hàng.*** Thêm dòng 18-23 trên Hình 14.13 vào xử lý sự kiện ShippingTimeForm_Load. Các dòng này gán giá trị các thuộc tính **MinDate** và **MaxDate** cho dropOffDateTimePicker. Thuộc tính MinDate chỉ ra giá trị sớm nhất cho phép người dùng nhập vào. Thuộc tính MaxDate chỉ ra giá trị muộn nhất mà DateTimePicker cho phép người dùng nhập vào. Hai thuộc tính này cùng nhau tạo ra khoảng thời gian thả hàng mà người dùng có thể chọn.

Thiết lập khoảng thời gian thả hàng



Hình 14.13 Thiết lập các thuộc tính MinDate và MaxDate.

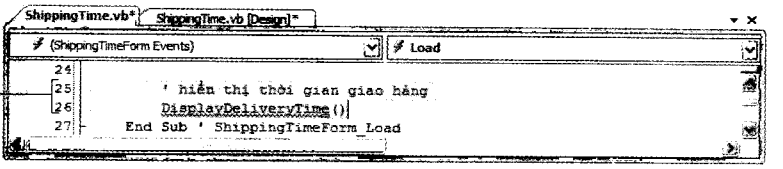
Để đảm bảo độ tươi sống, lô hàng hải sản phải được thả xuống sân bay trong ngày hôm đó; do vậy, thời gian thả hàng sớm nhất (MinDate) được đặt là 12:00 AM. của ngày hôm đó (dòng 19-20), thời gian thả hàng muộn nhất (MaxDate) được đặt là 12:00 AM. ngày hôm sau (dòng 22-23). Lưu ý rằng giá trị MaxDate được tính bằng cách thêm một ngày vào giá trị MinDate sử dụng phương thức AddDays. Nhớ lại rằng phương thức AddDays không thay đổi giá trị Date mà nó thao tác - phương thức này chỉ trả về giá trị Date mới. Giá trị này được gán cho thuộc tính MaxDate ở dòng 22.

Hàm khởi tạo Date (được gọi ở dòng 19) tạo giá trị lưu ngày tháng và thời gian lúc nửa đêm. Nhớ lại rằng tham số đầu tiên là năm, tham số thứ hai là tháng và tham số thứ ba là ngày. Ba tham số sau cùng là giờ, phút và giây. Thuộc tính Year của biến Date trả về giá trị của năm kiểu Integer (chẳng hạn, 2008). Thuộc tính Month trả về giá trị tháng của biến Date kiểu Integer (chẳng hạn, 6 cho tháng sáu). Cuối cùng, thuộc tính Day của biến Date trả về giá trị ngày trong tháng (một số kiểu Integer nằm giữa 1 và 31, tùy thuộc vào tháng và năm).

Kiểu Date cũng cung cấp thuộc tính **Today**. Thuộc tính này trả về ngày tháng hiện thời với thời gian được đặt giá trị 00:00:00. Bạn có thể sử dụng thuộc tính này để lấy về ngày tháng hiện thời khi không cần thông tin về thời gian. Bạn có thể sử dụng thuộc tính Today thay cho thuộc tính Now trong xử lý sự kiện Load của Form. Thuộc tính MinDate của DateTimePicker được đặt giá trị do hàm Date.Today (nửa đêm ngày hiện thời) trả về. Thuộc tính MaxDate được đặt giá trị bằng cách thêm một ngày vào thuộc tính MinDate. Ta đã sử dụng thuộc tính Now để bạn có cơ hội luyện tập nhiều hơn với hàm khởi tạo của Date và làm quen với các thuộc tính Year, Month và Day.

4. ***Gọi thủ tục DisplayDeliveryTime.*** Thêm dòng 25-26 trên Hình 14.14 để gọi thủ tục DisplayDeliveryTime. Lưu ý rằng DisplayDeliveryTime được gạch chân màu xanh. Đây là do lỗi biên dịch khi bạn gọi thủ tục chưa được định nghĩa. Bạn sẽ viết thủ tục này ở phần sau của chương. Thủ tục DisplayDeliveryTime tính thời gian giao hàng ở Las Vegas và hiển thị kết quả trên Label **Delivery time:**.

Hiển thị thời gian giao hàng



Hình 14.14 Gọi thủ tục `DisplayDeliveryTime`.

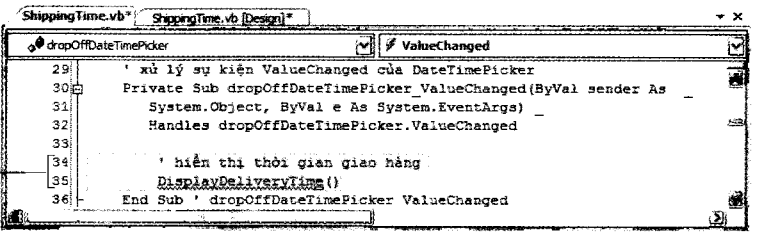
5. ***Lưu project.*** Chọn **File > Save All** để lưu mã đã được sửa đổi.

Đến đây bạn đã thêm chức năng gọi `DisplayDeliveryTime` để hiển thị thời gian giao hàng khi ứng dụng khởi chạy. Tuy nhiên, bạn sẽ cần cho phép người dùng chọn thời gian thả hàng và ngay lập tức thấy thời gian lô hàng hải sản sẽ được giao. Trong phần tiếp theo, bạn học cách quản lý sự kiện **`ValueChanged`** của `DateTimePicker` được kích hoạt khi người dùng thay đổi giá trị của `DateTimePicker`.

Lập trình xử lý sự kiện `ValueChanged`

1. ***Tạo xử lý sự kiện `ValueChanged`.*** Nhấn vào tab **[Design]** của file **ShippingTime.vb**. Nhấn đúp vào điều khiển `DateTimePicker` `dropOffDateTimePicker` để kích hoạt xử lý sự kiện.
2. ***Thêm mã vào xử lý sự kiện.*** Chèn dòng 34-35 trên Hình 14.15 vào xử lý sự kiện. Mã này chạy khi người dùng thay đổi thời gian trong `DateTimePicker`. Hãy chắc chắn đã thêm chú thích và ký tự xuống dòng như trên Hình 14.15 để số dòng trong mã của bạn khớp với số dòng trên hình.

Tính toán và hiển thị thời gian giao hàng



Hình 14.15 Chèn mã vào xử lý sự kiện `ValueChanged`.

Xử lý sự kiện `ValueChanged` cũng sử dụng thủ tục `DisplayDeliveryTime` để tính toán và hiển thị thời gian giao hàng ở Las Vegas. Trong phần tiếp theo, bạn sẽ viết thủ tục `DisplayDeliveryTime`, sau đó lỗi biên dịch sẽ không còn xuất hiện nữa.

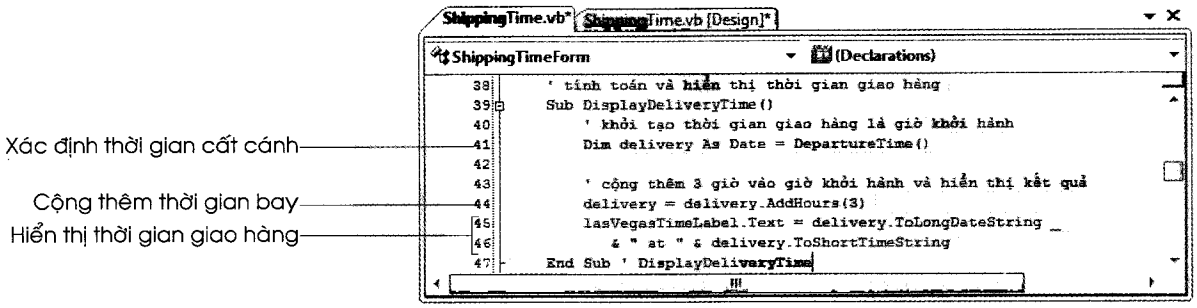
3. ***Lưu project.*** Chọn **File > Save All** để lưu mã đã sửa đổi.

Bạn cần phải định nghĩa thủ tục `DisplayDeliveryTime` vì bạn đã gọi thủ tục này trong hai xử lý sự kiện. Tiếp theo, bạn sẽ dùng phương thức của `Date` để tính toán và hiển thị thời gian giao hàng trong `Label` kết quả đầu ra.

Viết thủ tục `DisplayDeliveryTime`

1. ***Tạo thủ tục `DisplayDeliveryTime`.*** Thêm dòng 38-47 trên Hình 14.16 vào dưới xử lý sự kiện `ValueChanged`. Dòng 41 gọi thủ tục `DepartureTime`. Lưu ý rằng `DepartureTime` được gạch chân màu xanh, vì bạn gọi thủ tục chưa được định nghĩa nên IDE báo lỗi biên dịch. Bạn sẽ viết thủ tục này trong phần tiếp theo. Thủ tục `DepartureTime` xác định chuyến bay nào (nửa đêm hay giữa trưa) sẽ vận chuyển lô hàng hải sản. Thủ tục này trả về giá trị kiểu `Date` là thời gian xuất phát của chuyến bay. Dòng 41 lưu giá trị này vào biến `delivery` kiểu `Date`.

2. ***Tính toán và hiển thị thời gian giao hàng.*** Dòng 44 tính toán thời gian giao hàng bằng cách thêm ba giờ đồng hồ nữa vào thời gian xuất phát. Dòng 45-46 hiển thị thời gian giao hàng ở Las Vegas bằng cách gọi phương thức của `Date` là `ToLongDateString` và `ToShortTimeString`. Trong đó phương thức **`ToLongDateString`** trả về ngày tháng dưới dạng `String` có định dạng "Wednesday, October 30, 2008." Còn phương thức **`ToShortTimeString`** trả về thời gian dưới dạng `String` có định dạng "4:00 PM."



Hình 14.16 Thủ tục `DisplayDeliveryTime`.

3. ***Lưu project.*** Chọn **File > Save All** để lưu mã đã sửa đổi.

Khi tính thời gian giao hàng, bạn cần tính đến sự chênh lệch múi giờ và thời gian bay. Chẳng hạn, nếu bạn gửi lô hàng từ Portland, Maine đến Las Vegas, thì phải dịch chuyển về phía tây ba múi giờ (thời gian ở Las Vegas sớm hơn ba giờ) và mất sáu giờ vận chuyển. Nếu bạn thả hàng vào lúc 5:00 PM ở Portland, lô hàng sẽ rời bến vào chuyến bay lúc nửa đêm cùng ngày và đến Las Vegas vào lúc

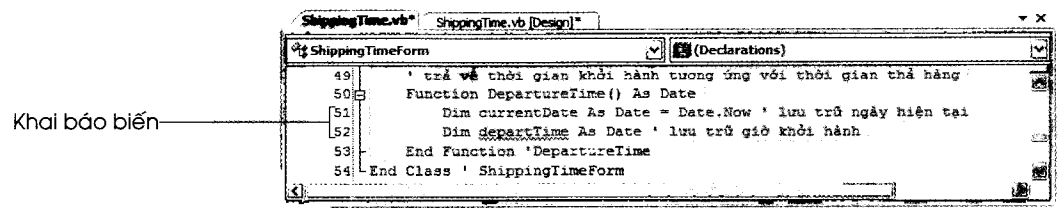
`12:00 AM +` (*thay đổi múi giờ + thời gian bay*) `= 12:00 AM + (-3 + 6)` *giờ*

tức là 3:00 AM ở Las Vegas. Tương tự, nếu lô hàng bay chuyến trưa cùng ngày, nó sẽ đến Las Vegas vào 3:00 PM.

Để hoàn chỉnh ứng dụng, bạn cần viết thủ tục `Function DepartureTime`. Bạn sử dụng lệnh `Select Case` và phương thức của `Date` để trả về thời gian cất cánh (giữa trưa hay nửa đêm) cho chuyến bay chở lô hàng hải sản.

Viết thủ tục DepartureTime

1. ***Viết thủ tục Departure Time.*** Chèn dòng 49-53 trên Hình 14.17 vào dưới thủ tục `DisplayDeliveryTime`. Dòng 51 lưu ngày tháng hiện thời vào biến `Date currentDate`. Dòng 52 khai báo biến `Date departTime` để lưu giá trị trả về của thủ tục `Function DepartureTime`.



Hình 14.17 Thêm thủ tục `DepartureTime` vào ứng dụng.

2. ***Xác định chuyến bay cho lô hàng.*** Chèn dòng 54-67 trên Hình 14.18 vào sau lệnh khai báo biến và trước lệnh `End Function`. Lệnh `Select Case` bắt đầu ở dòng 55 sử dụng giờ do người dùng chỉ ra trên `DateTimePicker` làm biểu thức điều khiển. Thuộc tính **`Value`** của `DateTimePicker` (kiểu `Date`) chứa giá trị do người dùng chọn. Thuộc tính `Hour` của `Date` trả về giờ của `Date` được lưu trong thuộc tính `Value` của `DateTimePicker`. Nhớ lại rằng thuộc tính `Hour` lưu giá trị giờ theo kiểu `Integer` trong khoảng từ 0 đến 23.

Sử dụng giá trị giờ được lưu trong `DateTimePicker` để xác định thời gian cất cánh

Thời gian cất cánh trưa cùng ngày

Thời gian cất cánh trưa ngày hôm sau

Thời gian cất cánh nửa đêm cùng ngày

```
52          Dim departTime As Date ' lưu trữ giờ khởi hành
53
54          ' xác định chuyến bay cho lô hàng
55          Select Case dropOffDateTimePicker.Value.Hour
56              Case 0 To 10 ' hàng sẽ trên chuyến bay trưa cùng ngày
57                  departTime = New Date(currentDate.Year, _
58                    currentDate.Month, currentDate.Day, 12, 0, 0)
59              Case 23 ' hàng sẽ trên chuyến bay trưa ngày hôm sau
60                  currentDate = currentDate.AddDays(1)
61                  departTime = New Date(currentDate.Year, _
62                    currentDate.Month, currentDate.Day, 12, 0, 0)
63              Case Else ' hàng sẽ trên chuyến bay nửa đêm
64                  currentDate = currentDate.AddDays(1)
65                  departTime = New Date(currentDate.Year, _
66                    currentDate.Month, currentDate.Day, 0, 0, 0)
67          End Select
```

Hình 14.18 Xác định giờ cất cánh của chuyến bay chở lô hàng hải sản.

Danh sách biểu thức của mệnh đề `Case` đầu tiên (dòng 56) xác định giá trị `DateTimePicker` có nằm giữa nửa đêm (`Hour = 0`) và 10:59 AM (`Hour = 10`) hay không. Nếu có, lô hàng sẽ bay chuyến giữa trưa cùng ngày đến Las Vegas. (Nhớ lại rằng lô hàng cần phải được thả xuống sân bay ít nhất là một giờ trước khi máy bay cất cánh). Thân mệnh đề `Case` đầu tiên (dòng 57-58) lưu thời gian khởi hành chuyến bay vào trưa cùng ngày vào biến `departTime`.

Danh sách biểu thức của mệnh đề `Case` tiếp theo (dòng 59) xác định giá trị trong `DateTimePicker` có nằm giữa 11:00 PM và 11:59 PM hay không (`Hour = 23`). Nếu thời gian thả hàng nằm giữa 11:00 PM và 11:59 PM, lô hàng sẽ bay chuyến bay trưa ngày hôm sau đến Las Vegas. Thân của mệnh đề `Case` này (dòng 60-62) lưu thời gian khởi hành chuyến bay trưa ngày hôm sau vào biến `departTime`.

Thân của lệnh `Case Else` sẽ thực thi nếu biểu thức điều khiển không đúng với mệnh đề `Case` ở trên (giá trị trong `DateTimePicker` nằm giữa 11:00 AM và 10:59 PM). Trong trường hợp này, lô hàng sẽ bay chuyến bay đêm cùng ngày đến Las Vegas. Thân mệnh đề `Case Else` (dòng 64-66) lưu thời gian cất cánh lúc nửa đêm vào biến `departTime`. Lưu ý rằng vì nửa đêm là thời điểm của ngày hôm sau, biến `Date` lưu thời điểm này sẽ chứa giá trị thuộc tính `Day` của hôm sau (dòng 64).

3. ***Trả về thời gian giao hàng.*** Thêm dòng 69 trên Hình 14.19 vào thủ tục `DepartureTime`, sau lệnh `End Select`. Dòng 69 trả về giá trị `Date` chứa thời gian chuyến bay cất cánh.

Trả về thời gian cất cánh

```
67          End Select
68
69          Return departTime ' trả về thời gian khởi hành
70      End Function ' DepartureTime
71  End Class ' ShippingTimeForm
```

Hình 14.19 Trả về thời gian chuyến bay cất cánh.

4. ***Chạy ứng dụng.*** Chọn **Debug > Start Debugging** để chạy ứng dụng.
5. ***Đóng ứng dụng.*** Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.
6. ***Đóng IDE.*** Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

Hình 14.20 thể hiện mã nguồn của ứng dụng **Shipping Time**. Các dòng mã chứa khái niệm lập trình mới mà bạn được học trong chương này được đánh dấu.

Sự kiện được kích hoạt khi Timer kích hoạt sự kiện Tick

Hiển thị thời gian hiện thời

Sự kiện xảy ra khi nạp Form

Thiết lập giá trị nhỏ nhất và lớn nhất cho DateTimePicker

Sự kiện được kích hoạt khi người dùng thay đổi giá trị của DateTimePicker

Tính toán và hiển thị thời gian giao hàng ở Las Vegas

Sử dụng lệnh Select Case để xác định thời gian cất cánh

```
1  Public Class ShippingTimeForm
2      ' xử lý sự kiện Tick của clockTimer
3      Private Sub clockTimer_Tick(ByVal sender As System.Object, _
4          ByVal e As System.EventArgs) Handles clockTimer.Tick
5
6          ' hiển thị thời gian hiện tại
7          currentTimeLabel.Text = String.Format("{0:hh:mm:ss tt}",_
8              Date.Now)
9      End Sub ' clockTimer_Tick
10
11     ' khởi tạo trạng thái DateTimePicker khi load Form
12     Private Sub ShippingTimeForm_Load(ByVal sender As _
13         System.Object, ByVal e As System.EventArgs) _
14         Handles MyBase.Load
15
16         Dim currentTime As Date = Date.Now ' lưu trữ thời gian hiện tại
17
18         ' thiết lập thời gian thả hàng
19         dropOffDateTimePicker.MinDate = New Date(currentTime.Year, _
20             currentTime.Month, currentTime.Day, 0, 0, 0)
21
22         dropOffDateTimePicker.MaxDate = _
23             dropOffDateTimePicker.MinDate.AddDays(1)
24
25         ' hiển thị thời gian giao hàng
26         DisplayDeliveryTime()
27     End Sub ' ShippingTimeForm_Load
28
29     ' xử lý sự kiện ValueChanged của DateTimePicker
30     Private Sub dropOffDateTimePicker_ValueChanged(ByVal sender As _
31         System.Object, ByVal e As System.EventArgs) _
32         Handles dropOffDateTimePicker.ValueChanged
33
34         ' hiển thị thời gian giao hàng
35         DisplayDeliveryTime()
36     End Sub ' dropOffDateTimePicker_ValueChanged
37
38     ' tính toán và hiển thị thời gian giao hàng
39     Sub DisplayDeliveryTime()
40         ' khởi tạo thời gian giao hàng là giờ khởi hành
41         Dim delivery As Date = DepartureTime()
42
43         ' cộng thêm 3 giờ vào giờ khởi hành và hiển thị kết quả
44         delivery = delivery.AddHours(3)
45         lasVegasTimeLabel.Text = delivery.ToLongDateString _
46         & " at " & delivery.ToShortTimeString
47     End Sub ' DisplayDeliveryTime
48
49     ' trả về thời gian khởi hành tương ứng với thời gian thả hàng
50     Function DepartureTime() As Date
51         Dim currentDate As Date = Date.Now ' lưu trữ ngày hiện tại
52         Dim departTime As Date ' lưu trữ giờ khởi hành
53
54         ' xác định chuyến bay cho lô hàng
55         Select Case dropOffDateTimePicker.Value.Hour
56             Case 0 To 10 ' hàng sẽ trên chuyến bay trưa cùng ngày
```

Hình 14.20 Mã nguồn ứng dụng **Shipping Time.** (Phần 1/2)

```
57              departTime = New Date(currentDate.Year, _
58                  currentDate.Month, currentDate.Day, 12, 0, 0)
59          Case 23 ' hàng sẽ trên chuyến bay trưa ngày hôm sau
60              currentDate = currentDate.AddDays(1)
61              departTime = New Date(currentDate.Year, _
62                  currentDate.Month, currentDate.Day, 12, 0, 0)
63          Case Else ' hàng sẽ trên chuyến bay nửa đêm
64              currentDate = currentDate.AddDays(1)
65              departTime = New Date(currentDate.Year, _
66                  currentDate.Month, currentDate.Day, 0, 0, 0)
67      End Select
68
69      Return departTime ' trả về thời gian khởi hành
70  End Function ' DepartureTime
71 End Class ' ShippingTimeForm
```

Hình 14.20 Mã nguồn ứng dụng **Shipping Time.** (Phần2/2)

TỰ ÔN TẬP

1. Phương thức `ToShortTimeString` được gọi bởi biến `Date` trả về giá trị của biến đó dưới định dạng_____.
   a) `11 h` b) `23:00`
   c) `11:00` d) `11:00 PM`
2. Các thuộc tính_____và_____của `DateTimePicker` lần lượt xác định ngày sớm nhất và muộn nhất có thể được chọn.
   a) `MinDate, MaxDate` b) `Now, Later`
   c) `Minimum, Maximum` d) `Early, Late`

Đáp án: **1)** d. **2)** a.

14.5 Tổng kết

Trong chương này, bạn đã tìm hiểu cách sử dụng kiểu `Date` để thao tác với thông tin thời gian. Bạn sử dụng các biến có kiểu này để tính toán và hiển thị thời gian giao hàng trong ứng dụng **Shipping Time**. Bạn đã sử dụng điều khiển `DateTimePicker` để cho phép người dùng nhập thông tin thời gian. Bạn đã quan sát cách điều khiển `DateTimePicker` hiển thị thời gian theo định dạng do người dùng tự định nghĩa và giới hạn giá trị nhập vào của người dùng. Bạn sử dụng điều khiển `GroupBox` để nhóm các điều khiển trên `Form` một cách trực quan. Bạn cũng đã học cách sử dụng điều khiển `Timer` để thực thi mã sau những khoảng thời gian nhất định được tính theo mili giây.

Sau đó bạn viết mã cho ba xử lý sự kiện mới để hoàn thành ứng dụng **Shipping Time**. Bạn đã biết rằng xử lý sự kiện `Load` của `Form` sẽ thực thi mã khi ứng dụng vừa được mở. Bạn đã sử dụng sự kiện này để thiết lập các giá trị khởi đầu cho ứng dụng. Sau đó bạn tìm hiểu cách sử dụng xử lý sự kiện `ValueChanged` của điều khiển `DateTimePicker` để thực thi mã khi giá trị của điều khiển này thay đổi. Bạn đã sử dụng xử lý sự kiện này để cập nhật thời gian giao hàng mỗi khi người dùng nhập vào thời gian thả hàng mới. Cuối cùng, bạn tìm hiểu về xử lý sự kiện `Tick` của điều khiển `Timer`. Bạn dùng xử lý sự kiện này để cập nhật và hiển thị thời gian hiện thời trên `Label` đồng hồ.

Trong chương tiếp theo, cuốn sách sẽ sử dụng ứng dụng **Fund Raiser** để giới thiệu hai khái niệm quan trọng - tham số và phạm vi của tham số. Tìm hiểu những khái niệm này sẽ giúp bạn hiểu cách Visual Basic theo dõi biến trong ứng dụng.

TỔNG KẾT KỸ NĂNG

Thực thi mã khi ứng dụng mở

- Sử dụng xử lý sự kiện `Load` của `Form` để thực thi mã khi ứng dụng vừa được mở.

Lưu và thao tác với thông tin thời gian

- Sử dụng biến `Date` (tương ứng với kiểu dữ liệu `DateTime` có sẵn) để lưu và

thao tác với thông tin thời gian. Biến Date lưu thông tin về một thời điểm (ví dụ 12:00:00 AM ngày 1/1/2003). Thông tin này có thể được định dạng để hiển thị theo kiểu ngắn hoặc dài được định nghĩa sẵn hoặc do người dùng tự định nghĩa.

Sử dụng biến Date

- Sử dụng từ khóa New để tạo giá trị Date mới.
- Sử dụng thuộc tính Date.Now để lấy về ngày giờ hiện thời dựa trên múi giờ của máy tính.
- Sử dụng toán tử truy cập thành viên (.) để truy cập thuộc tính của biến Date, ví dụ như Years, Hours...
- Sử dụng phương thức của kiểu Date, ví dụ như AddHours và AddDays để cộng hoặc trừ thời gian vào giá trị của biến Date. Sau đó gán giá trị do phương thức trả về cho biến Date.

Sử dụng điều khiển GroupBox

- Sử dụng điều khiển GroupBox để nhóm các điều khiển có liên quan một cách trực quan. Để thêm GroupBox vào Form, nhấn đúp vào điều khiển GroupBox trên **Toolbox** hoặc kéo điều khiển GroupBox từ **Toolbox** vào Form.
- Sử dụng thuộc tính Text để thiết lập tiêu đề cho GroupBox.

Đặt các điều khiển vào trong GroupBox

- Đặt điều khiển vào trong GroupBox bằng cách nhấn vào tên của điều khiển trên **Toolbox** rồi nhấn chuột vào trong GroupBox. Bạn cũng có thể kéo điều khiển từ **Toolbox** hoặc từ Form vào trong GroupBox.

Sử dụng điều khiển DateTimePicker

- Sử dụng điều khiển DateTimePicker để lấy thông tin ngày và giờ từ người dùng.
- Đặt thuộc tính Format thành Custom để chỉ ra rằng bạn sẽ tự định nghĩa định dạng ngày tháng xuất hiện trong DateTimePicker. Định dạng cụ thể sẽ được chỉ ra trong thuộc tính CustomFormat.
- Đặt thuộc tính ShowUpDown với giá trị True để cho phép người dùng chọn ngày tháng hoặc thời gian bằng cách nhấn mũi tên lên xuống. Nếu giá trị của thuộc tính này là False thì lịch tháng sẽ thả xuống để người dùng chọn ngày.
- Sử dụng xử lý sự kiện ValueChanged của DateTimePicker để thực thi mã khi giá trị của điều khiển được thay đổi.

Sử dụng điều khiển Timer

- Sử dụng điều khiển Timer để thực thi mã (xử lý sự kiện Tick) sau những khoảng thời gian nhất định.
- Để thêm điều khiển Timer vào Form, nhấn vào Timer trên **Toolbox** rồi nhấn chuột vào bất cứ đâu trên Form. Bạn cũng có thể nhấn đúp vào Timer trên **Toolbox**. Điều khiển Timer sẽ xuất hiện trên khay thành phần.
- Chỉ ra số mili giây giữa các sự kiện Tick bởi thuộc tính Inteval.
- Đặt giá trị thuộc tính Enabled là True để sự kiện Tick được kích hoạt sau mỗi khoảng thời gian.

THUẬT NGỮ

biến Date - Biến có kiểu Date dùng để chứa dữ liệu ngày tháng và thời gian.

chữ ký - Gồm các tham số của thủ tục và kiểu của chúng.

điều khiển DateTimePicker - Dùng để lấy về thông tin ngày tháng và thời gian từ người dùng.

điều khiển GroupBox - Dùng để nhóm các điều khiển có liên quan với nhau trên giao diện.

điều khiển Timer - Kích hoạt sự kiện Tick để thực thi mã sau những khoảng thời gian nhất định.

đối tượng chứa (container) - Một đối tượng chứa các đối tượng khác (ví dụ như `GroupBox` hoặc `Form`).

hàm khởi tạo (constructor) - Thủ tục để khởi tạo đối tượng khi cần tạo đối tượng.

khay thành phần (component tray) - Phần bên dưới Windows `Form` Designer chứa các điều khiển không được hiển thị trên giao diện (ví dụ như `Timer`).

kiểu cơ sở `DateTime` - Kiểu của thư viện .NET Framework Class Library tương ứng với từ khóa `Date`.

phương thức nạp chồng (method overloading) - Cho phép bạn tạo nhiều phương thức có cùng tên nhưng có chữ ký khác nhau.

phương thức `ToLongDateString` của kiểu `Date` - Trả về `String` chứa ngày tháng có định dạng giống như "Wednesday, October 30, 2002".

phương thức `ToShortTimeString` của kiểu `Date` - Trả về một `String` chứa giờ có định dạng giống như "4:00 PM."

sự kiện `Load` của một `Form` - Được kích hoạt khi ứng dụng thực thi.

sự kiện `Tick` của điều khiển `Timer` - Được kích hoạt sau mỗi khoảng thời gian - số mili giây được chỉ ra trong thuộc tính `Interval` của điều khiển `Timer` (nếu `Enabled` có giá trị `True`).

sự kiện `ValueChanged` của điều khiển `DateTimePicker` - Được kích hoạt khi người dùng chọn ngày hoặc giờ mới trên điều khiển `DateTimePicker`.

thuộc tính `CustomFormat` của điều khiển `DateTimePicker` - Là thuộc tính của `DateTimePicker` chứa chuỗi định dạng để hiển thị ngày tháng và/hoặc thời gian khi thuộc tính `Format` của `DateTimePicker` được thiết lập giá trị `Custom`.

thuộc tính `Format` của điều khiển `DateTimePicker` - Là thuộc tính của `DateTimePicker` cho phép bạn chỉ ra một định dạng đã được định nghĩa sẵn hoặc tự định nghĩa để hiển thị ngày và/hoặc giờ.

thuộc tính `Interval` của điều khiển `Timer` - Là thuộc tính của Timer cho phép bạn xác định số mili giây giữa các sự kiện Tick.

thuộc tính `MaxDate` của điều khiển `DateTimePicker` - Dùng để chỉ ra giá trị muộn nhất mà `DateTimePicker` cho phép người dùng nhập vào.

thuộc tính `MinDate` của điều khiển `DateTimePicker` - Dùng để chỉ ra giá trị sớm nhất mà `DateTimePicker` cho phép người dùng nhập vào.

thuộc tính `Now` của kiểu `Date` - Dùng để lấy về thời gian hiện thời của máy tính.

thuộc tính `ShowUpDown` của điều khiển `DateTimePicker` - Thuộc tính này khi có giá trị `True` sẽ cho phép người dùng ấn định giờ bằng mũi tên lên xuống, khi có giá trị `False` sẽ cho phép người dùng ấn định ngày tháng bằng lịch đổ xuống.

thuộc tính `Today` của kiểu `Date` - Trả về ngày hiện thời với thời gian được thiết lập là nửa đêm.

thuộc tính `Value` của điều khiển `DateTimePicker` - Lưu giá trị thời gian của điều khiển `DateTimePicker`.

từ khóa `New` - Dùng để gọi hàm khởi tạo khi tạo đối tượng.

HƯỚNG DẪN THIẾT KẾ GIAO DIỆN

`DateTimePicker`

- Sử dụng `DateTimePicker` để lấy về thông tin ngày tháng và thời gian từ người dùng.
- Mỗi `DateTimePicker` nên có một `Label` mô tả tương ứng.
- Nếu muốn người dùng chỉ ra giờ thì hãy thiết lập giá trị thuộc tính `ShowUpDown` của `DateTimePicker` là `True`, còn nếu muốn người dùng chỉ ra ngày thì hãy thiết lập giá trị thuộc tính `ShowUpDown` của điều khiển thành `False` để cho phép người dùng chọn ngày từ lịch tháng.

GroupBox

- Tiêu đề GroupBox nên ngắn gọn và sử dụng kiểu viết hoa tiêu đề sách.
- Sử dụng GroupBox để nhóm các điều khiển có liên quan trong cùng một hình chữ nhật có tiêu đề.

ĐIỀU KHIỂN, SỰ KIỆN, THUỘC TÍNH & PHƯƠNG THỨC

Date Kiểu dữ liệu này cung cấp các thuộc tính và phương thức để lưu và thao tác với thông tin thời gian.

- ***Thuộc tính***

  Day - Trả về ngày được lưu trong biến Date.

  Hour - Trả về giờ được lưu trong biến Date.

  Minute - Trả về phút được lưu trong biến Date.

  Month - Trả về tháng được lưu trong biến Date.

  Now - Trả về thời gian hiện thời của hệ thống.

  Second - Trả về giây được lưu trong biến Date.

  Today - Trả về ngày hiện thời của hệ thống với giờ được thiết lập là 00:00:00 (nửa đêm).

  Year - Trả về giá trị năm được lưu trong biến Date.

- ***Phương thức***

  AddDays - Tạo một giá trị Date mới sau (hoặc trước) giá trị của biến Date gọi phương thức này một số ngày được chỉ định.

  AddHours - Tạo một giá trị Date mới sau (hoặc trước) giá trị của biến Date gọi phương thức này một số giờ được chỉ định.

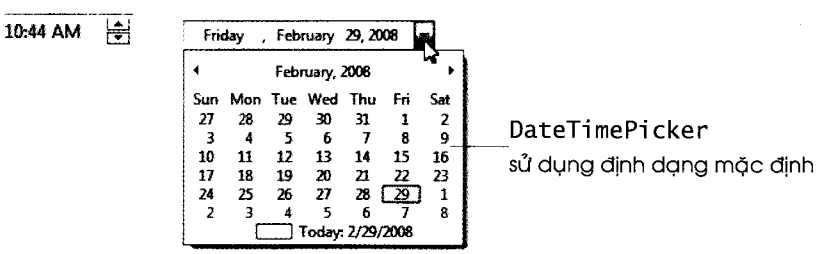
  AddMinutes - Tạo một giá trị Date mới sau (hoặc trước) giá trị của biến Date gọi phương thức này một số phút được chỉ định.

  ToLongDateString - Trả về một String chứa ngày theo định dạng "Wednesday, October 30, 2002".

  ToShortTimeString - Trả về một String chứa giờ theo định dạng "4:00 PM".

DateTimePicker DateTimePicker Điều khiển này được dùng để lấy thông tin về thời gian từ người dùng.

- ***Trên giao diện khi ứng dụng chạy***



- ***Sự kiện***

  ValueChanged - Được kích hoạt khi thuộc tính Value bị thay đổi.

- ***Thuộc tính***

  CustomFormat - Dùng để tự thiết lập chuỗi định dạng với mục đích hiển thị thời gian theo định dạng tự định nghĩa này.

  Format - Chỉ ra định dạng thời gian được hiển thị trên điều khiển. Long chỉ ra rằng ngày tháng được hiển thị theo định dạng "Monday, December 09,

2002". Short chỉ ra rằng ngày được hiển thị theo định dạng "12/9/2002". Time chỉ ra rằng giờ được hiển thị theo định dạng "8:39:53 PM". Custom cho phép bạn chỉ ra định dạng tự định nghĩa để hiển thị thời gian.

Location - Chỉ ra vị trí tương đối của điều khiển DateTimePicker so với góc trên bên trái của đối tượng chứa (Form hoặc GroupBox).

MinDate - Chỉ thời gian (ngày và/hoặc giờ) nhỏ nhất có thể được chọn.

MaxDate - Chỉ ra thời gian (ngày và/hoặc giờ) lớn nhất có thể được chọn.

Name - Chỉ ra tên dùng để truy cập điều khiển DateTimePicker khi lập trình. Tên nên được thêm hậu tố DateTimePicker vào phía sau.

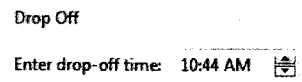
ShowUpDown - Nếu có giá trị là True thì mũi tên lên xuống sẽ được hiển thị trên điều khiển giúp người dùng chỉ định giờ. Nếu False, một mũi tên trỏ xuống được hiển thị để chọn ngày từ lịch thả xuống.

Size - Chỉ ra chiều rộng và chiều cao (bằng pixel) của điều khiển DateTimePicker.

Value - Lưu trữ thời gian của điều khiển DateTimePicker.

GroupBox GroupBox Điều khiển này nhóm các điều khiển có liên quan một cách trực quan vào hình chữ nhật có tiêu đề.

- ***Trên giao diện khi ứng dụng chạy***



- ***Thuộc tính***

  Location - Chỉ ra vị trí của điều khiển GroupBox so với góc trên bên trái của đối tượng chứa (VD: Form hoặc GroupBox).

  Name - Tên dùng để truy cập điều khiển GroupBox khi lập trình. Tên nên được thêm hậu tố GroupBox vào phía sau.

  Size - Chỉ ra chiều rộng và chiều cao (theo pixel) của điều khiển GroupBox.

  Text - Tiêu đề hiển thị trên GroupBox.

Timer Timer Điều khiển này "thức dậy" sau những khoảng thời gian cố định để thực thi mã trong xử lý sự kiện Tick.

- ***Sự kiện***

  Tick - Được kích hoạt sau mỗi số mili giây được chỉ định bởi thuộc tính Interval trôi qua.

- ***Thuộc tính***

  Enabled - Chỉ định Timer có chạy hay không. Giá trị mặc định là False.

  Interval - Khoảng thời gian (bằng mili giây) giữa hai sự kiện Tick.

  Name - Tên dùng để truy cập của điều khiển Timer khi lập trình. Tên nên được thêm hậu tố Timer vào sau.

CÂU HỎI TRẮC NGHIỆM

14.1 _____cho phép bạn lưu và thao tác thông tin ngày tháng dễ dàng.

a) Kiểu Date
b) Điều khiển DatePicker
c) Điều khiển GroupBox
d) Thuộc tính Now

14.2 Bạn có thể_____một biến Date.

a) cộng thêm giờ vào
b) cộng thêm ngày vào
c) trừ bớt giờ đi
d) Các lựa chọn trên đều đúng

14.3 Để trừ một ngày từ giá trị của biến day kiểu Date, gán giá trị được trả về từ_____cho biến day.

a) `day.addHours(-24)` b) `day.SubtractDays(1)`

c) `day.AddDays(-1)` d) Cả a và c

14.4 Có thể định dạng thời gian 3:45 và 35 giây vào buổi chiều là 3:45:35 PM với chuỗi định dạng_____.

a) `"hh:mm:ss"` b) `"hh:mm:ss tt"`

c) `"hh:mm:ss am:pm"` d) `"h:m:s tt"`

14.5 Sự kiện_____xảy ra trước khi Form được hiển thị.

a) `LoadForm` b) `InitializeForm`

c) `Load` d) `FormLoad`

14.6 Thuộc tính Interval của Timer thiết lập thời gian xảy ra sự kiện Tick tính bằng_____.

a) nano giây b) micro giây

c) mili giây d) giây

14.7 Để thiết lập giờ của biến time kiểu Date sớm hơn năm giờ, sử dụng_____.

a) `time = time.SubtractHours(5)`

b) `time = time.AddHours(-5)`

c) `time = time.AddHours(5)`

d) `time.AddHours(-5)`

14.8 _____là một đối tượng chứa.

a) `GroupBox` b) `Form`

c) `Timer` d) Cả a và b

14.9 Biến Date lưu giá trị giờ trong khoảng_____.

a) 1 đến 12 b) 0 đến 12

c) 0 đến 24 d) 0 đến 23

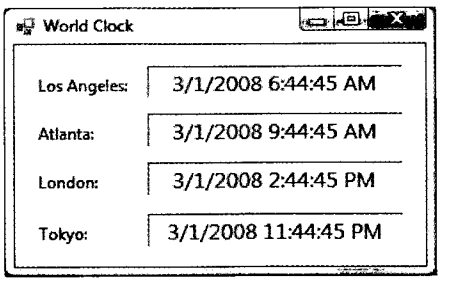
14.10 Thuộc tính_____của DateTimePicker chỉ định chuỗi định dạng để hiển thị thời gian.

a) `CustomFormat` b) `FormatString`

c) `Format` c) `Text`

BÀI TẬP

14.11 (***Ứng dụng World Clock***) Tạo ứng dụng hiển thị thời gian hiện thời ở Los Angeles, Atlanta, London và Tokyo. Sử dụng Timer để cập nhật đồng hồ sau mỗi giây. Giả sử rằng giờ địa phương của bạn là Atlanta. Atlanta muộn hơn Los Angeles ba giờ. London sớm hơn Atlanta năm giờ. Tokyo muộn hơn London chín giờ. Ứng dụng có giao diện giống như trên Hình 14.21.



Hình 14.21 Giao diện của ứng dụng **World Clock**.

a) ***Copy template của ứng dụng vào thư mục làm việc.*** Copy thư mục `C:\Examples\Tutorial14\Exercises\WorldClock` vào thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng.*** Nhấn đúp vào file `WorldClock.sln` trong thư mục `WorldClock` để mở ứng dụng.

c) ***Thêm `Timer` vào `Form`.*** Thêm điều khiển `Timer` vào ứng dụng **World Clock**. Thiết lập thuộc tính `Name` của `Timer` là `clockTimer`. Điều khiển `Timer` cần kích hoạt sự kiện `Tick` sau mỗi 1000 mili giây (một giây).

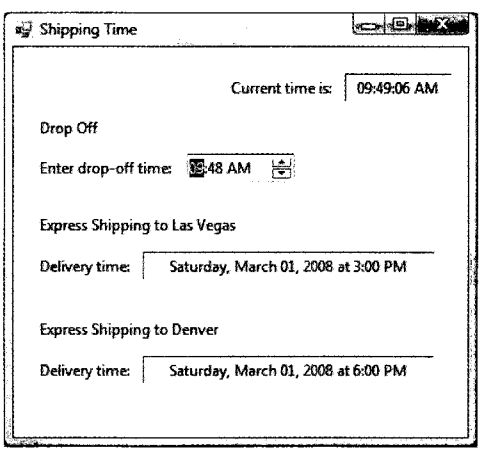
d) ***Thêm xử lý sự kiện `Tick` cho `clockTimer`.*** Thêm xử lý sự kiện `Tick` cho `clockTimer`. Xử lý sự kiện này sẽ tính toán và hiển thị thời gian hiện tại của Los Angeles, Atlanta, London và Tokyo.

e) ***Chạy ứng dụng.*** Chọn **Debug > Start Debugging** để chạy ứng dụng. Hãy xem đồng hồ trên máy của bạn và chắc chắn rằng thời gian Los Angeles sớm hơn giờ đó 3 tiếng, thời gian ở Atlanta giống với giờ trên đồng hồ của bạn, giờ ở London muộn hơn 5 tiếng và giờ ở Tokyo muộn hơn 14 tiếng (muộn hơn London 9 tiếng).

f) ***Đóng ứng dụng.*** Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

g) ***Đóng IDE.*** Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

14.12 (***Ứng dụng Shipping Time nâng cao***) Trong mùa đông, trung tâm phân phối tại Denver, Colorado cần nhận các chuyến hàng hải sản để cung cấp cho các khu nghỉ dưỡng trượt tuyết. Hãy nâng cấp ứng dụng **Shipping Time** bằng cách thêm điểm giao hàng tại Denver, Colorado. Denver nằm cách Portland, Maine hai múi giờ về phía tây, có nghĩa là sớm hơn Portland hai giờ. Vì không có các chuyến bay thẳng tới Denver, nên các chuyến hàng từ Portland tới đó cần tám giờ bay.



Hình 14.22 Giao diện ứng dụng **Shipping Time** nâng cao.

a) ***Copy template của ứng dụng vào thư mục làm việc.*** Copy thư mục `C:\Examples\Tutorial14\Exercises\ShippingTimeEnhanced` vào thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng.*** Nhấn đúp vào file `ShippingTime.sln` trong thư mục `ShippingTimeEnhanced` để mở ứng dụng.

c) ***Thêm một `GroupBox`.*** Thay đổi kích thước `Form` để có thể thêm vào `GroupBox` **Express Shipping to Denver** như trên Hình 14.22. Thêm `GroupBox` vào `Form`. Thay đổi thuộc tính `Text` của `GroupBox` để chỉ ra rằng `GroupBox` này chứa thời gian giao hàng tại Denver. Sửa lại kích thước và vị trí của `GroupBox` sao cho giống với giao diện trên Hình 14.22.

d) ***Thêm Label***. Trong GroupBox vừa tạo, hãy thêm Label đầu ra để hiển thị thời gian giao hàng hải sản tại Denver và Label mô tả tương ứng.

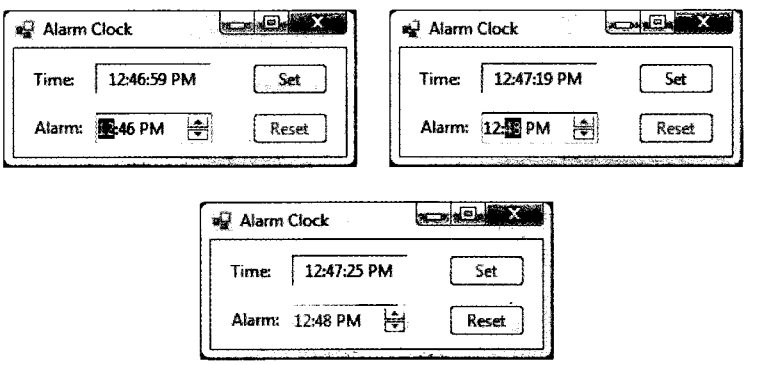
e) ***Viết mã cho thủ tục DisplayDeliveryTime***. Viết mã cho thủ tục DisplayDeliveryTime để tính và hiển thị thời gian giao hàng tại Denver.

f) ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng. Chọn những thời gian thả hàng khác nhau và chắc chắn rằng thời gian giao hàng đúng với cả Las Vegas và Denver.

g) ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

h) ***Đóng IDE***. Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

14.13 (***Ứng dụng Alarm***) Tạo ứng dụng cho phép người dùng đặt đồng hồ báo thức. Ứng dụng này cho phép người dùng đặt thời gian cho đồng hồ báo thức bằng cách sử dụng điều khiển DateTimePicker. Khi thời gian báo thức đã được đặt, người dùng không thể thay đổi DateTimePicker. Nếu đồng hồ báo thức đã được đặt và thời gian hiện thời trùng khớp với hoặc vượt quá thời gian trong DateTimePicker, hãy bật tiếng "beep" của máy tính. (Máy tính của bạn phải có phần cứng âm thanh cần thiết để có thể phát ra âm thanh). Người dùng có thể sử dụng Button **Reset** để hủy giờ báo thức. Ban đầu, khi ứng dụng khởi chạy thì Button này bị vô hiệu hóa.



Hình 14.23 Giao diện ứng dụng **Alarm**.

a) ***Copy template vào thư mục làm việc***. Copy thư mục C:\Examples\Tutorial14\Exercises\AlarmClock vào thư mục C:\SimplyVB2008.

b) ***Mở file template của ứng dụng***. Nhấn đúp vào file AlarmClock.sln trong thư mục AlarmClock để mở ứng dụng.

c) ***Thêm DateTimePicker***. Thêm điều khiển DateTimePicker vào Form. Thiết lập để DateTimePicker chỉ hiển thị giờ như trên Hình 14.23. Thay đổi kích thước và vị trí của DateTimePicker như trên Hình 14.23.

d) ***Viết mã để xử lý sự kiện Click cho Button Set***. Thêm xử lý sự kiện Click cho Button **Set**. Xử lý sự kiện này cần vô hiệu hóa Button **Set** và DateTimePicker, sau đó kích hoạt Button **Reset**.

e) ***Viết mã để xử lý sự kiện Tick cho Timer***. Định nghĩa xử lý sự kiện Tick cho Timer. Sự kiện Tick xảy ra sau mỗi 1000 mili giây (một giây). Cập nhật thời gian hiện thời sau mỗi giây. Nếu đồng hồ báo thức đã được hẹn giờ và thời gian hiện thời trùng khớp hoặc vượt quá thời gian trên DateTimePicker, hãy bật tiếng "beep" của máy tính bằng cách gọi hàm Beep. Để gọi hàm Beep, gõ Beep() trên một dòng lệnh riêng biệt. Nhớ lại rằng bạn có thể sử dụng toán tử quan hệ trên giá trị Date.

f) ***Viết mã để xử lý sự kiện Click cho Button Reset.*** Định nghĩa xử lý sự kiện `Click` của `Button` **Reset**. Khi `Button` **Reset** được nhấn, giao diện cần được thiết lập về trạng thái ban đầu.

g) ***Chạy ứng dụng.*** Chọn **Debug > Start Debugging** để chạy ứng dụng. Sử dụng `DateTimePicker` và `Button` **Set** để đặt thời gian báo thức. Đợi cho tới khi đồng hồ kêu tiếng beep. Nhấn `Button` **Reset** để thiết lập thời gian báo thức mới.

h) ***Đóng ứng dụng.*** Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

i) ***Đóng IDE.*** Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.

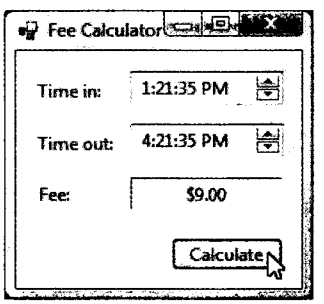
Đoạn mã này làm gì? ▶ **14.14** Đoạn mã này tạo biến `Date`. Hãy cho biết giá trị ngày của biến này.

```
Dim day as Date = New Date(2003, 1, 2, 3, 4, 5)
```

Đoạn mã này có gì sai? ▶ **14.15** Dòng mã sau có mục đích tạo biến `Date` và tăng giá trị giờ của nó thêm hai giờ. Tìm lỗi trong đoạn mã sau.

```
Dim currentDay As Date = Date.Now
currentDay.AddHours(2)
```

Bài tập nâng cao ▶ **14.16** (***Fee Calculator***) Tạo ứng dụng tính toán phí đỗ xe trong gara đỗ xe (Hình 14.24). Người dùng cần cung cấp giá trị **Time In:** và **Time out:** bằng cách sử dụng `DateTimePicker`. Ứng dụng sẽ tính toán phí đỗ xe trong gara trong khoảng thời gian nhất định. Giả sử rằng phí đỗ xe là 3USD/giờ. Khi tính toán tổng thời gian xe ở trong gara, bạn chỉ tính đến phút và bỏ qua giây. Để đơn giản, ta giả sử rằng không được phép đỗ xe qua đêm, có nghĩa là mỗi xe rời khỏi gara cùng ngày mà nó vào.



Hình 14.24 Giao diện ứng dụng **Fee Calculator**.

a) ***Copy template của ứng dụng vào thư mục làm việc.*** Copy thư mục `C:\Examples\Tutorial14\Exercises\FeeCalculator` vào thư mục `C:\SimplyVB2008`.

b) ***Mở file template của ứng dụng.*** Nhấn đúp vào file `FeeCalculator.sln` trong thư mục `FeeCalculator` để mở ứng dụng.

c) ***Thêm điều khiển DateTimePicker.*** Thêm hai điều khiển `DateTimePicker` vào `Form`. Thiết lập để `DateTimePicker` chỉ hiển thị thời gian. Đặt thuộc tính `Size` và `Location` của mỗi `DateTimePicker` để chúng giống như trên Hình 14.24.

d) ***Viết thủ tục Function Fee.*** Định nghĩa thủ tục `Function Fee` nhận vào hai tham số kiểu `Date` - giá trị của `DateTimePicker` **Time In:** và **Time Out:**. Với thông tin này, thủ tục `Fee` sẽ tính toán và trả về phí đỗ xe trong gara kiểu `Decimal`.

e) ***Viết mã để xử lý sự kiện Click cho Button Calculate.*** Thêm xử lý sự kiện `Click` cho `Button` **Calculate**. Xử lý sự kiện này sẽ gọi `Fee` để lấy về phí gửi xe. Sau đó nó sẽ hiển thị phí (được định dạng theo kiểu tiền tệ) trên `Label`.

f) ***Chạy ứng dụng***. Chọn **Debug > Start Debugging** để chạy ứng dụng. Sử dụng mũi tên lên xuống của `DateTimePicker` để chọn thời gian mà xe được đưa vào gara và thời gian xe ra khỏi gara. Nhấn `Button` **Calculate** và kiểm tra xem tiền phí đã được hiển thị đúng hay chưa.

g) ***Đóng ứng dụng***. Đóng ứng dụng đang chạy bằng cách nhấn vào nút x ở trên cùng, bên phải của ứng dụng.

h) ***Đóng IDE***. Đóng cửa sổ Visual Basic IDE bằng cách nhấn vào nút x ở trên cùng, bên phải của IDE.