Chương 6 Applets

Sau khi kết thúc chương này, bạn có thể nắm được các nội dung sau:
Sau khi học xong chương này, bạn có thể nắm được các nội dung sau:

- Hiểu được các Applet của Java
- Phân biệt applet và các ứng dụng application
- Tìm hiểu chu trình sống của một applet
- Tạo các applet
- Hiển thị các hình ảnh sử dụng applet
- Truyền tham số cho applet
- Tìm hiểu ứng dụng của applet trong GUI

6.1 Java Applet

Applet là một chương trình Java có thể chạy trong trình duyệt web. Tất cả các applet đều là các lớp con của lớp 'Applet'.
Lớp Applet thuộc package 'java.applet'. Lớp Applet bao gồm nhiều phương thức để điều khiển quá trình thực thi của applet. Để tạo applet, bạn cần import hai gói sau:

- java.applet
- java.awt

6.2 Cấu trúc của một Applet

Một Applet định nghĩa cấu trúc của nó từ 4 sự kiện xảy ra trong suốt quá trình thực thi. Đối với mỗi sự kiện, một phương thức được gọi một cách tự động. Các phương thức này được minh hoạ trong bảng 6.1
Điều quan trọng là không phải lúc nào applet cũng bắt đầu từ ban đầu. Mà nó bắt đầu từ vị trí tiếp theo của quá trình thực thi trước đó.
Ngoài những phương thức cơ bản này, còn có những phương thức 'paint()' và 'repaint()'. Phương thức paint() dùng để hiển thị một đường thẳng (line), text, hoặc một hình ảnh trên màn hình. Đối số của phương thức này là đối tượng của lớp Graphics. Lớp này thuộc gói java.awt. Câu lệnh sau được dùng để import lớp Graphics:

import java.awt.Graphics;

Phương thức	Chức năng
init()	Được gọi trong quá trình khởi tạo applet. Trong quá trình khởi tạo, nó sẽ tạo đối tượng để cung cấp cho applet. Phương thức này được dùng để tải các hình ảnh đồ hoạ, khởi tạo các biến và tạo các đối tượng.
start()	Được gọi gọi khi một applet bắt đầu thực thi. Một khi quá trình khởi tạo hoàn tất, thì applet được khởi động. Phương thức này được dùng để khởi động lại applet sau khi nó đã ngừng trước đó
stop()	Được gọi khi ngừng thực thi một applet. Một applet bị ngừng trước khi nó bị huỷ.

destroy()	Được dùng để huỷ một applet. Khi một applet bị huỷ, thì bộ nhớ, thời gian thực thi của vi xử lý, không gian đĩa được trả về cho hệ thống.

Bảng 6.1: các phương thức của một applet

Phương thức 'repaint()' được dùng khi cửa sổ cần cập nhật lại. Phương thức này chỉ cần một thông số. Tham số này là đối tượng của lớp Graphics.

Applet sử dụng phương thức 'showStatus()' để hiển thị thông tin trên thanh trạng thái. Phương thức có tham số thuộc kiểu dữ liệu String. Để lấy các thông tin của applet, user có thể overide phương thức 'getAppletInfo()' của lớp Applet. Phương thức này trả về 1 đối tượng kiểu String.

Các phương thức của applet init(), start(), stop(), destroy(), và paint() được thừa kế từ một applet. Nhưng mặc định những phương thức này không thực thi một thao tác nào cả. Đây là ví dụ đơn giản của applet. Câu lệnh sau tạo một lớp có tên là 'Applet1', lớp này sẽ kế thừa tất cả các phương thức và biến của lớp 'applet'.

public class Applet1 extends Applet

Phương thức init() và paint() thường được dùng để thực hiện một số hàm để khởi tạo và vẽ applet. Phương thức 'g.drawstring()' chỉ ra vị trí mà đoạn văn bản được vẽ ở đâu trên màn hình.

Chương trình 6.1 hiển thị một chuỗi ở dòng 70 và cột 80:

Chương trình 6.1

```
import  java.awt.*;
import java.applet.*;
public class Applet1 extends Applet
{
        int num;
        public void init()
        {
                num = 6;
        }
        public void paint (Graphics g)
        {
                g.drawString ("Hello to Applet. Chapter " + num, 70, 80);
                showStatus (getAppletInfo());
                //Hiển thị  một chuỗi được trả về từ hàm getAppletInfo() trên thanh trạng
                thái
        }
        public String getAppletInfo() //user overrides
        {
                return "Created by Aptech";
        }
```

}

Sử dụng cú pháp sau để dịch một Applet:

javac Applet1.java

Để thực thi một applet, ta cần tạo một file HTML. File HTML này sử dụng thẻ applet. Thẻ applet này lấy tham số đầu tiên là đường dẫn của file applet.

Thẻ applet có hai thuộc tính sau:

- Width
- Height

Để truyền tham số vào applet, sử dụng param, sau đó là thẻ value. Sau đây là ví dụ của thẻ applet:

<applet code=Applet1 width=300 height=200>
</applet>

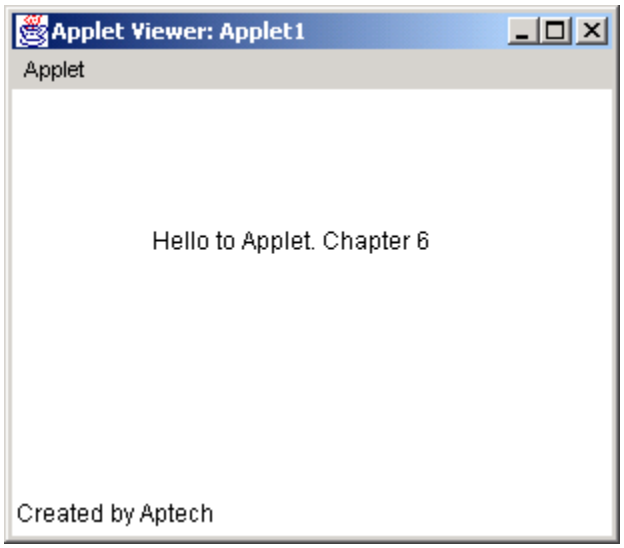
Lúc này, ta có thể thực thi applet này bằng cách dùng trình xem applet. Đây là công cụ của JDK. Để chạy file HTML trong trình xem applet, ta gõ câu lệnh sau:

Appletviewer abc.html // 'abc.html' là tên của file HTML

Một tuỳ chọn khác của applet là ta thêm thẻ applet như là một dòng chú thích trong đoạn code. Lúc đó, applet được dịch, và thực thi bằng cách sử dụng lệnh sau:

Appletviewer Applet1.java

Sau đây là kết quả của chương trình trên:



Hình 6.1 Applet

6.2.1 Sự khác nhau giữa Application và Applet

Sau đây là sự khác nhau giữa application và applet:

- Để thực thi các application chúng ta dùng trình thông dịch Java, trong khi đó applet có thể chạy được trên các trình duyệt (có hỗ trợ Java) hay sử dụng công cụ AppletViewer, công cụ này đi kèm với JDK.
- Quá trình thực thi của application bắt đầu từ phương thức 'main()'. Tuy nhiên applet thì không làm như vậy.
- Các application sử dụng 'System.out.println()' để hiển thị kết quả ra màn hình trong khi đó applet sử dụng phương thức 'drawstring()' để xuất ra màn hình.

Một điều đáng lưu ý là một chương trình Java đơn lẻ thì có thể vừa là application vừa là applet. Chức năng của applet được bỏ qua khi nó được thực thi như là một application và ngược lại.

Chương trình 6.2 sẽ minh hoạ điều này

Chương trình 6.2

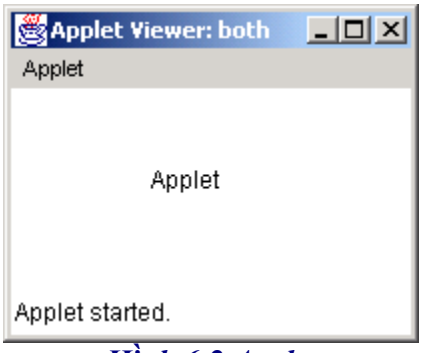
```
import java.applet.Applet;
import java.awt.*;
/*
<applet code = "both" width = 200 height = 100>
</applet>
*/

public class both extends Applet
{
    Button btn;
    public void init()
    {
        btn = new Button ("Click");
    }

    public void paint (Graphics g)
    {
        g.drawString ("Applet", 70, 50);
    }
    public static void main (String args[])
    {
        both app = new both();
        app.init();
        System.out.println("Application Main");
```

```
    }
}
```

Sau khi biên dịch chương trình, nó có thể được thực thi như là một applet bằng cách sử dụng cú pháp sau:

appletviewer both.java

Kết quả như sau:

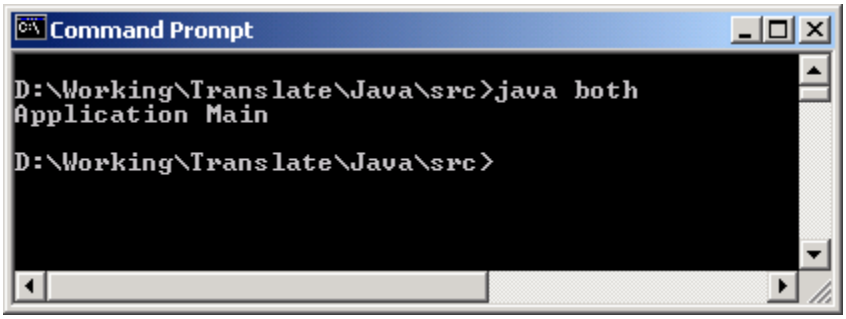


Hình 6.2 Applet

Nếu chạy chương trình trên như một application, thì sử dụng cú pháp sau:

java both

Kết quả là:



Hình 6.3 Application

Khi applet chạy trên trình duyệt web, đặc điểm này thực sự hữu ích khi bạn muốn tải applet trong một frame mới. Ví dụ: trong applet được tạo để chat, một số website sử dụng một cửa sổ chat riêng biệt để chat. Bạn cũng có thể kết hợp các đặc điểm của frame và applet vào trong một chương trình.

6.2.3 Những giới hạn bảo mật trên applet

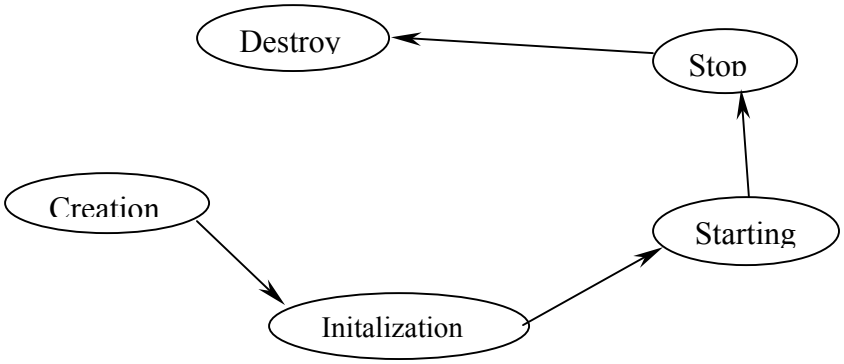
Có một số hạn chế mà applet không thể làm được. Bởi vì các applet của Java có thể phá hỏng toàn bộ hệ thống của user. Các lập trình viên Java có thể viết các applet để xoá file, lấy các thông tin các nhân của hệ thống…
Vì thế, các applet của java không thể làm các việc sau:

- Không thể đọc hoặc ghi file trên hệ thống file của user.
- Không thể giao tiếp với các site internet, nhưng chỉ có thể với các trang web có applet mà thôi.
- Không thể chạy bất cứ chương trình gì trên hệ thống của người đọc.
- Không thể tải bất cứ chương trình được lưu trữ trong hệ thống của user.

Những giới hạn trên chỉ đúng khi các applet được chạy trên trình duyệt Netscape Navigator hoặc Microsoft Internet Explorer.

6.3 Chu trình sống của một Applet

Chu trình sống của một Applet được mô tả ở sơ đồ dưới đây:



Hình 6.4 Chu trình sống của một applet

Trước tiên, applet được tạo.
Bước kế tiếp là khởi tạo. Điều này xảy ra khi một applet được nạp. Quá trình này bao gồm việc tạo các đối tượng mà applet cần. Phương thức init() được override để cung cấp các hành vi để khởi tạo.
Một khi applet được khởi tạo, applet sẽ được khởi động. Applet có thể khởi động ngay cả khi nó đã được ngừng trước đó. Ví dụ, nếu trình duyệt nhảy đến một liên kết nào đó ở trang khác, lúc đó applet sẽ bị ngừng, và được khởi động trở lại khi user quay về trang đó.

Sự khác nhau giữa quá trình khởi tạo và quá trình khởi động là một applet có thể khởi động nhiều lần, nhưng qúa trình khởi tạo thì chỉ xảy ra một lần.

Phương thức 'start()' được override để cung cấp các thao tác khởi động cho applet.

Phương thức 'stop()' chỉ được gọi khi user không còn ở trang đó nữa, hoặc trang đó đã được thu nhỏ lại ở dưới thanh taskbar.

Kế tiếp là phương thức 'destroy()'. Phương thức này giúp applet dọn dẹp trước khi nó được giải phóng khỏi vùng nhớ, hoặc trước khi truyệt duyệt kết thúc. Phương thức này được dùng để huỷ những luồng (thread) hay quá trình đang chạy.

Phương thức 'destroy()' khác với phương thức finalize() là phương thức destroy() chỉ dùng cho applet, trong khi finalize() là cách tổng quát để dọn dẹp applet.

Phương thức paint() cũng là một phương thức quan trọng khác. Phương thức này cho phép ta hiển thị một cái gì đó trên màn hình. Có thể là text, đường thẳng, màu nền, hoặc hình ảnh. Phương thức này xảy ra nhiều lần trong suốt quá trình applet tồn tại. Phương thức này thực thi một lần sau khi applet được khởi tạo. Nó sẽ lặp đi lặp lại khi di chuyển từ cửa sổ trình duyệt sang cửa sổ khác. Nó cũng xảy ra khi cửa sổ trình duyệt thay đổi vị trí của nó trên màn hình.

Phương thức 'paint()' có một tham số. Tham số này là đối tượng của lớp Graphics. Lớp Graphics thuộc lớp java.awt, chúng ta phải import trong đoạn code của applet. Chúng ta có thể sử dụng đoạn mã sau:

import java.awt.Graphics;

6.4 Truyền tham số cho Applet

Trong chương trình sau, chúng ta sẽ truyền tham số cho applet. Thành phần nút 'bNext' có tên được truyền như là một tham số. Phương thức 'init()' sẽ kiểm tra tham số có thên là 'mybutton'. Sau đó, nó tạo một nút với chuỗi đó như là tên của nút. Nếu không có tham số truyền vào, nút đó có tên mặc định là 'Default'.
Bây giờ chúng ta định nghĩa thẻ <PARAM> trong đoạn mã HTML như sau:

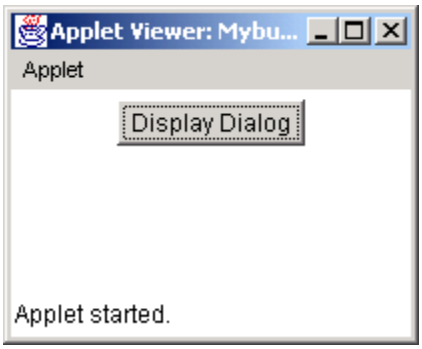
```
/*
<applet code="Mybutton1" width="100" height="100">
<PARAM NAME="mybutton" value="Display Dialog">
</applet>
*/
```

Chương trình 6.3

```
import java.awt.*;
import java.applet.*;
/*
<applet code="Mybutton1" width="200" height="100">
<PARAM NAME="mybutton" value="Display Dialog">
</applet>
*/
```

```
public class Mybutton1 extends Applet
{
    Button bNext;
    public void init()
    {
        /*getParameter returns the value of the specified pareameter in the form of a
        String object*/
        String str = getParameter("mybutton");
        //when no parameter is passed
        if (str==null)
                str = new String ("Default");
        //when parameter is passed
        bNext = new Button(str);
        add (bNext);
    }
}
```

Sau đây là kết quả của chương trình trên:



Hình 6.5: truyền tham số cho applet

Bây giờ chúng ta sẽ sử dụng lớp Graphics để vẻ các hình chẳng hạn như: đường thẳng, hình oval, và hình chữ nhật. Chúng ta sẽ học lớp Font trong các phần sau. Lớp này có thể dùng để in văn bản bằng bất cứ font nào.

6.5 Lớp Graphics

Java cung cấp gói AWT cho phép ta vẽ các hình đồ hoạ. Lớp Graphics bao gồm tập hợp rất nhiều phương thức. Nhưng phương thức này được sử dụng để vẽ bất cứ hình nào trong các hình sau:

- Oval
- Rectangle
- Square
- Circle
- Lines

- Text

Bạn có thể vẽ những hình này bằng bất cứ màu nào. Frame, Applet và canvas là các môi trường để hiển thị đồ hoạ.

Để vẽ bất cứ hình ảnh nào chúng ta cần phải có nền đồ hoạ (Graphical Background). Để có được một nền đồ hoạ, chúng ta goi phương thức 'getGraphics()' hay bất cứ phương thức nào trong các phương thức sau đây:

- repaint()
  Được gọi khi cần vẽ lại những đối tượng đã vẽ.
- update(Graphics g)
  Được gọi một cách tự động bởi phương thức 'repaint()'.
  Phương thức này sẽ xoá những đối tượng đã vẽ, và truyền nó cho đối tượng của lớp Graphics để gọi phương thức 'paint()';
- paint(Graphics g)
  Được gọi bởi phương thức update().

  Đối tượng được truyền cho phương thức này được dùng để vẽ. Phương thức này dùng để vẽ các hình ảnh đồ hoạ khác nhau.

Việc gọi phương thức paint() lặp đi lặp lại thông qua phương thức repaint() sẽ xoá đi các hình đã vẽ trước đó. Để vẽ các hình mới mà vẫn giữ lại những hình đã vẽ trước đó, chúng ta cần override lại phương thức upate().

```
Public void update (Graphics g)
{
        paint (g);
}
```

Ở đây, phương thức update() sẽ không xoá những đối tượng đã vẽ, nhưng chỉ gọi phương thức paint(). Để làm được điều này, nó truyền đối tương của lớp Graphics hoặc GraphicsContext cho phương thức paint(). Ở đây, đối tượng của lớp Graphics là 'g'.

6.5.1 Vẽ các chuỗi, các ký tự và các byte

Chương trình sau minh hoạ các vẽ các chuỗi, ký tự và các byte.
Để vẽ hoặc in một chuỗi, lớp Graphics cung cấp phương thức 'drawString()'. Cú pháp như sau:

```
DrawString (String str, int xCoor, int yCoor);
```

Ba tham số là:
- Chuỗi cần vẽ.
- Toạ độ X trên frame, nơi chuỗi cần được vẽ.
- Toạ độ Y trên frame, nơi chuỗi cần được vẽ.

Để vẽ hoặc xuất các ký tự trên frame, lớp Graphics cung cấp phương thức 'drawChars'. Cú pháp như sau:

DrawChars (char array[], int offset, int length, int xCoor, int yCoor);

Chú thích các tham số:

- Mẫng các ký tự.
- Vị trí bắt đầu, nơi các ký tự được vẽ.
- Số các ký tự cần được vẽ.
- Toạ độ X, nơi các ký tự cần được vẽ.
- Toạ độ Y, nơi các ký tự cần được vẽ.

Lớp Graphics cung cấp phương thức 'drawBytes()' để vẽ hoặc in các byte ra frame. Cú pháp của phương thức này như sau:

DrawBytes (byte array[], int offset, int length, int xCoor, int yCoor);

5 tham số của phương thức trên là:

- Mẫng các byte.
- Vị trí offset hay vị trí bắt đầu.
- Số byte cần vẽ.
- Toạ độ X.
- Toạ độ Y.

Đối với một ký tự hoặc một mẫng các byte, chúng ta có thể in một phần của mẫng mà thôi. Ở đây, toạ độ x và y là toạ độ tính theo dòng. Chương trình 6.4 minh hoạ cách vẽ chuỗi, các ký tự và các byte.

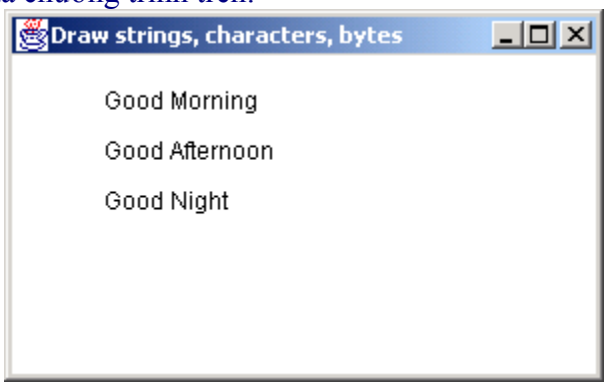
Chương trình 6.4

```
import java.awt.*;
public class DrawStrings extends Frame
{
    public DrawStrings()
    {
        super ("Draw strings, characters, bytes");
        setSize (300, 300);
        setVisible (true);
    }
    public void paint(Graphics g)
    {
        g.drawString ("Good Morning", 50, 50);
        g.drawString ("Good Afternoon", 50, 75);
        g.drawString ("Good Night", 50, 100);
```

```
        char ch[] = {};
    }
    public static void main (String args[])
    {
        new DrawStrings();
    }
}
```

Chương trình trên vẽ chuỗi, ký tự từ một mảng ký tự, và vẽ các byte từ mảng các byte. Bạn phải import gói java.awt để sử dụng các phương thức đồ hoạ có sẵn trong gói này. Ta phải làm điều này vì lớp Graphics nằm trong gói này.

Sau đây là kết quả của chương trình trên:



Hình 6.6 Strings, characters và bytes

6.5.2 Vẽ đường thẳng (Line) và Oval

Sau đây là cú pháp của các phương thức được sử dụng để vẽ đường thẳng và hình oval:

- drawLine (int x1, int y1, int x2, int y2);
- drawOval (int xCoor, int yCoor, int width, int height);
- setColor (Color c);
- fillOval (int xCoor, int yCoor, int width, int height);

Phương thức 'drawLine()' nhận các tham số sau:

- Toạ độ X, nơi bắt đầu vẽ (x1).
- Toạ độ Y, nơi bắt đầu vẽ (y1).
- Toạ độ X, nơi kết thúc vẽ (x2).
- Toạ độ Y, nơi kết thúc vẽ (y2).

Phương thức này bắt đầu vẽ tại toạ độ 'x1' và 'y1', và kết thúc tại toạ độ 'x2' và 'y2'. Để vẽ nhưng đường thẳng có màu, chúng ta thiết lập một màu nào đó. Phương thức 'setColor' dùng để thiết lập màu cho hình ảnh đồ hoạ. Trong chương trình này, chúng ta sử dụng câu lệnh sau để chọn màu xanh:

g.setColor (Color.blue);

Phương thức 'drawOval()' nhận 4 thông số sau:

- Toạ độ X.
- Toạ độ Y.
- Chiều rộng của hình Oval.
- Chiều cao của hình Oval.

Đối với hình oval rộng, thì giá trị của chiều rộng lớn hơn chiều cao, và ngược lại đối với hình oval cao.

Phương thức 'fillOval()' nhận 4 thông số, nhưng nó sẽ tô hình oval. Sử dụng phương thức setColor để tô hình oval;

g.setColor(Color.cyan);

Ở đây, hình oval sẽ được tô với màu cyan. Lớp Color cung cấp các màu khác nhau mà hệ thống có hổ trợ.

6.5.3 Vẽ hình chữ nhật (Rectangle) và hình chữ nhật bo góc (Rounded Rectangle)

Sau đây là cú pháp của các phương thức được dùng để vẽ hình chữ nhật và hình chữ nhật bo góc:

- drawRect (int xCoor, int yCoor, int width, int height);
- fillRect (int xCoor, int yCoor, int width, int height);
- drawRoundRect (int xCoor, int yCoor, int width, int height, int arcwidth, int archeight);
- fillRoundRect (int xCoor, int yCoor, int width, int height, int arcwidth, int archeight);

Phương thức 'drawRect()' được dùng để vẽ hình chữ nhật đơn giản. Phương thức này nhận 4 tham số sau:

- Toạ độ X
- Toạ độ Y
- Chiều rộng của hình chữ nhật
- Chiều cao của hình chữ nhật

Phương thức này vẽ hình chữ nhật có chiều rộng và chiều cao cho trước, bắt đầu tại toạ độ X, Y. Chúng ta có thể thiết lập màu của hình chữ nhật. Ở đây, chúng ta chọn màu đỏ. Câu lệnh sẽ như sau:

g.setColor (Color.red);

Phương thức 'drawRoundRect()' vẽ hình chữ nhật có các góc tròn. Phương thức này nhận 6 tham số, trong đó 4 tham số đầu thì giống với phương thức drawRect. Hai tham số khác là:

- arcwidth của hình chữ nhật
- archeight của hình chữ nhật

Ở đây, 'arcwidth' làm tròn góc trái và góc phải của hình chữ nhật. 'archeight' làm tròn góc trên đỉnh và góc đáy của hình chữ nhật. Ví dụ, arcwidth = 20 có nghĩa là hình chữ nhật được làm tròn cạnh trái và cạnh phải mỗi cạnh 10 pixel. Tương tự, archeight = 40 sẽ tạo ra hình chữ nhật được làm tròn từ đỉnh đến đáy 20 pixel.

Pixel là đơn vị đo. Nó là đơn vị nhỏ nhất trong vùng vẽ.

Để tô hay vẽ hình chữ nhật và hình chữ nhật bo góc, chúng ta sử dụng phương thức 'fillRect()' và 'fillRoundRect()'. Những phương thức này nhận các tham số giống với phương thức drawRect() và drawRoundRect(). Những phương thức này vẽ các hình ảnh với một màu cho trước hoặc mới màu hiện hành. Lệnh sau dùng để vẽ hình với màu xanh:

g.setColor(Color.green);

6.5.4 Vẽ hình chữ nhật 3D và vẽ hình cung (Arc)

Sau đây là cú pháp của các phương thức dùng để vẽ hình chữ nhật 3D và hình cung:

- draw3Drect (int xCoord, int yCoord, int width, int height, boolean raised);
- drawArc(int xCoord, int yCoord, int width, int height, int arcwidth, int archeight);
- fillArc(int xCoord, int yCoord, int width, int height, int arcwidth, int archeight);

Phương thức 'draw3Drect()' nhận 5 tham số. 4 tham số đầu thì tương tự với phương thức để vẽ hình chữ nhật. Tuy nhiên, giá trị của tham số thứ 5 quyết định là hình chữ nạh6t này có 3 chiều hay không. Tham số thứ 5 có kiểu dữ liệu là Boolean. Giá trị này True có nghĩa là hình chữ nhật là 3D.

Phương thức 'drawArc()' nhận 6 tham số sau:

- Toạ độ x
- Toạ độ y
- Chiều rộng của cung được vẽ.
- Chiều cao của cung được vẽ.
- Góc bắt đầu.
- Độ rộng của cung so với góc ban đầu.

Phương thức 'fillArc()' cũng nhận 6 tham số giống như phương thức drawArc(), nhưng nó vẽ cung và tô cung với màu hiện thời.

6.5.5 Vẽ hình PolyLine

Chương trình sau lấy các điểm từ hai mảng để vẽ một loạt các đường thẳng.

Cú pháp của phương thức này như sau:

- drawPolyline (int xArray[], int yArray[], int totalPoints);
- g.setFont (new Font("Times Roman", Font.BOLD, 15));

Phương thức 'drawPolyline()' nhận 3 tham số sau:

- Mẫng lưu trữ toạ độ x của các điểm.
- Mẫng lưu trữ toạ độ y của các điểm.
- Tổng số điểm cần vẽ.

Để vẽ các đường thẳng ta lấy các điểm từ hai mảng như sau:

(array1[0], array2[0]) (array1[1], array2[1]) (array1[2], array2[2])….

Số đường thẳng vẽ được luôn nhỏ hơn số truyền vào thông số thứ 3 của phương thức drawPoyline(). Ví dụ như: totalPoints - 1

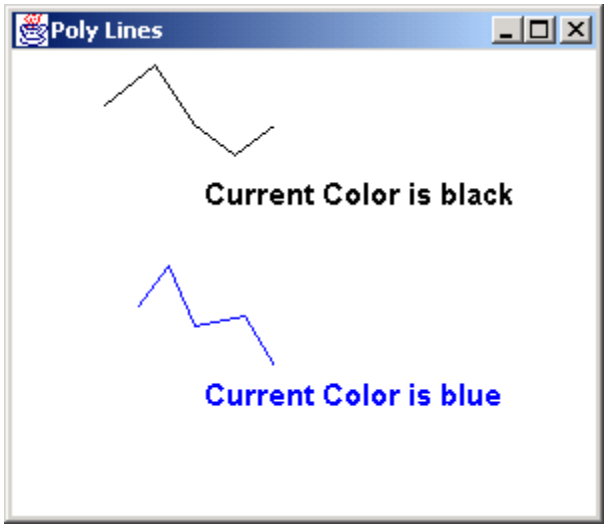
Chương trình 6.5 minh hoạ các vẽ polyline.

Chương trình 6.5

```
import java.awt.*;

class PolyLines extends Frame
{
    int x1[] = {50, 75, 95, 115, 135};
    int y1[] = {50, 30, 60, 75, 60};
    int x2[] = {67, 82, 95, 120, 135};
    int y2[] = {150, 130, 160, 155, 180};

    public PolyLines()//constructor
    {
        super ("Poly Lines");
        setSize (300, 300);
        setVisible (true);
    }
    public void paint (Graphics g)
    {
```

```
        g.drawPolyline (x1, y1, 5);
        g.setFont (new Font("Times Roman", Font.BOLD, 15));
        g.drawString("Current Color is black", 100, 100);
        g.setColor(Color.blue);
        g.drawPolyline (x2, y2, 5);
        g.drawString ("Current Color is blue", 100, 200);
    }
    public static void main (String args[])
    {
        new PolyLines();
    }
}
```

Kết quả của chương trình được minh hoạ ở hình 6.7



6.5.6 Vẽ và tô đa giác (Polygon)

Lớp Graphics cung cấp hai phương thức để vẽ đa giác. Phương thức đầu tiên nhận một đối tượng của lớp Polygon. Phương thức thức 2 lấy hai mẫng điểm, và tổng số điểm cần vẽ. Chúng ta sẽ sử dụng phương thức 2 để vẽ đa giác.

Cú pháp của **drawPolygon()** như sau:
drawPolygon(int x[], int y[], int numPoints);

Cú pháp của **fillPolygon()** như sau:
fillPolygon (int x[], int y[], int numPoints);

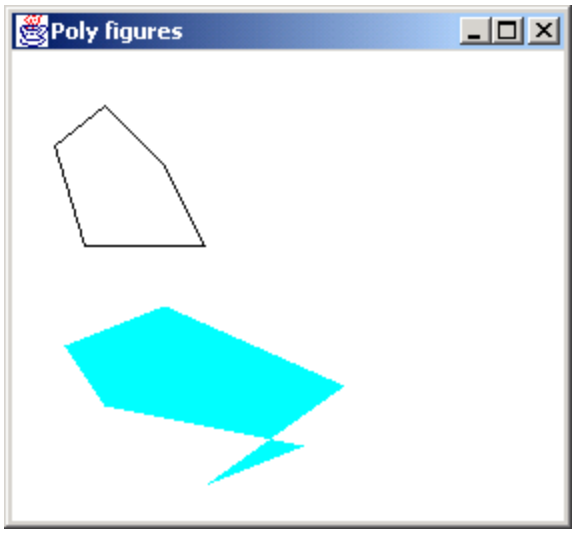
Chương trình dưới đây lấy các điểm từ 2 mảng để vẽ đa giác. Phương thức 'drawPolygon()' nhận 3 tham số sau giống như phương thức drawPolyline()

- Mảng lưu trữ toạ độ x của các điểm.
- Mảng lưu trữ toạ độ y của các điểm.
- Tổng số điểm cần vẽ.

Chương trình 6.6

```
import java.awt.*;
class PolyFigures extends Frame
{
    int x1[] = {50, 25, 40, 100, 80};
    int x2[] = {80, 30, 50, 150, 100, 170};
    int y1[] = {50, 70, 120, 120, 80};
    int y2[] = {150, 170, 200, 220, 240,190};

    public PolyFigures()
    {
        super ("Poly figures");
        setSize(300, 300);
        setVisible (true);
    }
    public void paint (Graphics g)
    {
        g.drawPolygon (x1, y1, 5);
        g.setColor (Color.cyan);
        g.fillPolygon (x2, y2, 6);
    }
    public static void main (String args[])
    {
        new PolyFigures();
    }
}
```

Sau đây là kết quả của chương trình trên:



Hình 6.8 Polygon

6.8 Điều khiển màu

Trong Java, chúng ta điều khiển màu bằng cách dùng 3 màu chính là đỏ (red), xanh lá cây (green), xanh dương (blue). Java sử dụng mô kiểu màu RGB. Đối tượng của lớp Color chứa 3 số nguyên cho các tham số red, green, blue. Bảng sau trình bày giá trị có thể có của các màu đó:

Thành phần	Phạmvi
Red	0-255
Green	0-255
Blue	0-255

Bảng 6.2 Phạm vi giá trị của các thành phần màu

Sử dụng các giá trị trên để tạo ra một màu tuỳ thích. Cú pháp của hàm dựng để tạo ra một màu như sau:

color (int red, int green, int blue);

Bảng sau hiển thị các giá trị của các màu thường gặp:

Màu	Red	Green	Blue
White	255	255	255
Light Gray	192	192	192
Gray	128	128	128
Dark Gray	64	64	64
Black	0	0	0
Pink	255	175	175

Orange	255	200	0
Yellow	255	255	0
Magenta	255	0	255

Bảng 6.3 Các giá trị RGB

Các đối tượng màu khác nhau có thể được tạo bằng những giá trị này. Những đối này có thể được dùng để vẽ hoặc tô các đối tượng đồ hoạ. Ví dụ, để tạo màu hồng, ta dùng lệnh sau:

color c = new Color (255, 175, 175);

Ta có thể thiết lập màu bằng cách dùng lệnh sau:

g.setColor (c); //g là đối tượng của lớp Graphics

Sử dụng kết hợp các giá trị RGB để tạo ra một màu tuỳ ý. Để cho dể hơn, lớp Color cung cấp sẵn một số màu.

color.white	color.black
color.orange	color.gray
color.lightgray	color.darkgray
color.red	color.green
color.blue	color.pink
color.cyan	color.magenta
color.yellow	

Bảng 6.4 Các màu thường gặp

Đoạn mã sau minh hoạ cách tạo một màu tuỳ ý:

```
Color color1 = new Color (230, 140, 60);
Color color4 = new Color (90, 210, 130);
g.setColor (color1);
int myred = color1.getRed ();
int mygreen = color1.getGreen ();
int myblue = color1.getBlue();
color1 = color1.darker();
color4 = color4.brighter();
```

6.7 Điều khiển Font

Java cung cấp lớp Font trong gói java.awt cho phép sử dụng các loại font khác nhau. Lớp này bao gồm một số phương thức.

Để sử dụng font, chúng ta nên kiểm tra xem hệ thống có hổ trợ hay không. Phương thức'getAllFont()' trả về tất cả các font mà hệ thống hổ trợ.

Trước tiên, khai báo một đối tượng của lớp GraphicsEnvironment như sau:

Graphicenvironment ge;

ge = GraphicsEnvironment.getLocalGraphicsEnvironment ();

Đối tượng này sử dụng cú pháp sau để lấy tất cả các font có trong mãng Font:

Font f[] = ge.getAllFonts();

Phương thức getAllFont() được sử dụng ở đây. Phương thức getAllFont() thuộc lớp GraphicsEnvironment. Đây là lớp trừu tượng, do đó ta không thể khởi tạo lớp này. để truy cập phương thức getAllFont(), chúng ta sử dụng phương thức 'getLoacalGraphicsEnvironment()' của lớp GraphicsEnvironment.

ge = GraphicsEnvironment.getLocalGraphicsEnvironment ();

Tham chiếu đến lớp này được gán cho biến ge. Biến này gọi phương thức getAllFont(). Chúng ta sử dụng các font khác nhau để hiển thị các chuỗi khác nhau. Phương thức getFont() trả về font mặc định dùng để hiển thị chuỗi, khi không có chọn font nào cả.

Font defaultFont = g.getFont (); //g là đối tượng Graphics
g.drawString ("Default Font is ", 30, 50);

Dialog là font mặc định của hệ thống.

Để thay đổi font mặc định của hệ thống thành font khác, chúng ta tạo đối tượng của lớp Font. Hàm dựng của Font lấy 3 tham số sau:

- Tên của font. Ta có thể lấy tên thông qua phương thức getFontList().
- Kiểu của font. Ví dụ: Font.BOLD, Font.PLAIN, Font.ITALIC.
- Kích thước font.

Cú pháp sau minh hoạ những thông số trên:

Font f1 = new Font ("SansSerif", Font.ITALIC, 16);
g.setFont (f1);

Ba tham số được truyền ở đây là: 'SanSerif' – tên của font, Font.BOLD – kiểu font, 14 là kích thước của font. Những thông số này tạo ra đối tượng f1. Chúng ta có thể kết hợp 2 kiểu font lại với nhau. Hãy xét ví dụ sau:

Font f3 = new Font ("Monospaced", Font.ITALIC+Font.BOLD, 20);

Ở đây kiểu font của f3 vừa đậm, vừa nghiêng.

6.8 Lớp FontMetrics

Lớp này xác định kích thước của các ký tự khác nhau thuộc các loại font khác nhau. Xác định kích thước bao gồm chiều cao (height), baseline, descent, và leading. Điều này rất cần thiết vì các ký tự khi in đều chiếm một kích thước riêng. Bạn cần tính kích thước cần thiết khi in các ký tự để tránh các ký tự ghi đè lên nhau.

- Height: chiều cao của font.
- Baseline (Dòng cơ sở): xác định cơ sở của các ký tự (không kể phần thấp nhất của ký tự)
- Ascent: khoảng cách từ đường baseline đến đỉnh của ký tự.
- Descent: khoảng các từ baseline đề đáy của ký tự.
- Leading: khoảng cách giữa các dòng chữ in.

Chương trình 6.7 minh hoạ việc sử dụng các phương thức khác nhau mà lớp FontMetrics có. Trong chương trình này, chúng ta sử dụng các phương thức khác nhau để xem xét chi tiết các loại font khác nhau. Lớp FontMetric là lớp trừu tượng. Phương thức getFontMetrics() có tham số là đối tượng của lớp Font, vì FontMetrics đi đôi với một font nào đó.

FontMetrics fm = g.getFontMetrics (f1);

Lệnh này tạo đối tượng fm của lớp FontMetrics, cùng với đối tượng f1. Bây giờ, chúng ta sử dụng fm để lấy chi tiết của font.

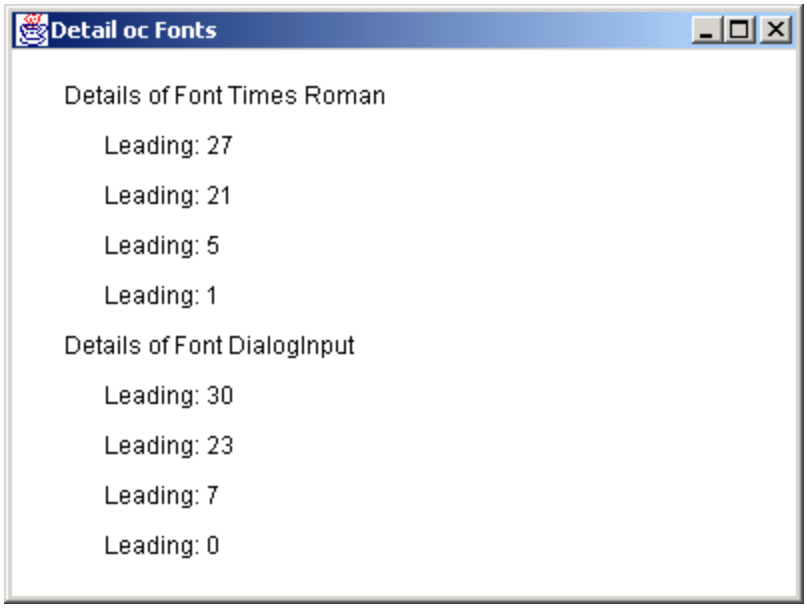
Các phương thức getHeight, getAscent(), getDescent(), và getLeading() trả về chi tiết của font. Phương thức getFont() của lớp FontMetrics trả về Font mà kết hợp với đối tượng của lớp FontMetrics. Phương thức getName() của lớp Font trả về tên Font.

Chương trình 6.7

```
import java.awt.*;

class FontMetricsUse extends Frame
{
    public FontMetricsUse()
    {
        super ("Detail oc Fonts");
        setSize (400, 300);
        setVisible(true);
    }

    public void paint (Graphics g)
    {
        Font f1 = new Font ("Times Roman", Font.PLAIN, 22);
        FontMetrics fm = g.getFontMetrics (f1);
```

```
        String name = fm.getFont().getName();
        g.drawString ("Details of Font " + name, 30, 50);
        g.drawString ("Leading: " + String.valueOf (fm.getHeight()), 50, 75);
        g.drawString ("Leading: " + String.valueOf (fm.getAscent()), 50, 100);
        g.drawString ("Leading: " + String.valueOf (fm.getDescent()), 50, 125);
        g.drawString ("Leading: " + String.valueOf (fm.getLeading()), 50, 150);

        Font f2 = new Font ("DialogInput", Font.PLAIN, 22);
        fm = g.getFontMetrics (f2);
        name = fm.getFont().getName();
        g.drawString ("Details of Font " + name, 30, 175);
        g.drawString ("Leading: " + String.valueOf (fm.getHeight()), 50, 200);
        g.drawString ("Leading: " + String.valueOf (fm.getAscent()), 50, 225);
        g.drawString ("Leading: " + String.valueOf (fm.getDescent()), 50, 250);
        g.drawString ("Leading: " + String.valueOf (fm.getLeading()), 50, 275);
    }
    public static void main (String args[])
    {
        new FontMetricsUse ();
    }
}
```

Kết quả của chương trình trên:



Hình 6.9 Lớp FontMetrics

Chương trình 6.8 minh hoạ cách lớp FontMetrics được sử dụng để in đoạn văn bản nhiều font, nhiều dòng. Trong chương trình này, chúng ta cần in văn bản nhiều font trên nhiều

dòng. Lớp FontMetrics giúp ta xác định khoảng cách cần thiết để in một dòng văn bản cho một font nào đó. Điều này thật cần thiết, bởi vì dòng thứ 2 được in ngay sau dòng thứ nhất.

Trước tiên chúng ta in msg1 sử dụng font Monospaced. Sau đó, chúng ta xuất msg2 sử dụng font iaglogInput. Để làm được điều này, chúng ta cần tính khỏng cách cần thiết để xuất msg1. Phương thức stringWidth() của lớp FontMetrics được dùng để tính ra tổng khoảng cách cần thiết để xuất msg1. khi chúng cộng thêm khoảng cách này vào biến x, chúng ta sẽ lấy được vị trí mà chúng ta bắt đầu in đoạn văn bản kế tiếp, msg2. Phương thức setFont() được dùng để thiết lập font để in văn bản.

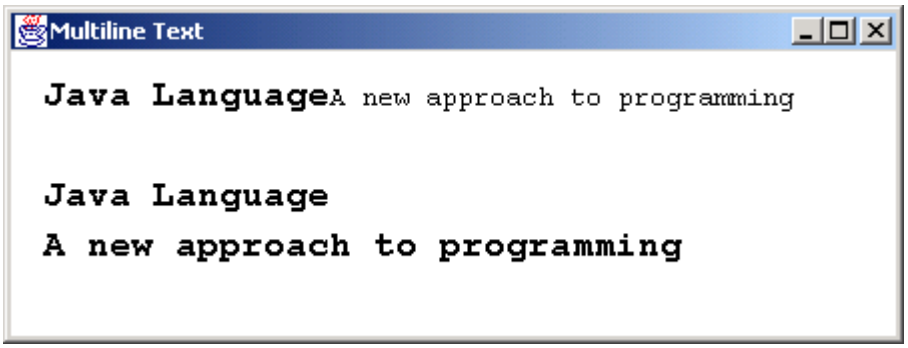
Kế đó, chúng ta xuất msg1 và msg2 trên các dòng khác nhau sử dụng chung 1 font Monospaced. Ở đây, chúng ta cần biết khoảng cách chiều cao của font, để in dòng kế tiếp. Phương thức getHeight() được dùng để làm điều này.

Chương trình 6.8

```
import java.awt.*;
class MultiFontMultiLine extends Frame
{
	public MultiFontMultiLine()
	{
		super ("Multiline Text");
		setSize (450, 200);
		setVisible (true);
	}

	public void paint (Graphics g)
	{
		Font f1 = new Font ("MonoSpaced", Font.BOLD, 18);
		Font f2 = new Font ("DialogInput", Font.PLAIN, 14);

		int x = 20;
		int y = 50;
		String msg1 = "Java Language";
		String msg2 = "A new approach to programming";

		FontMetrics fm = g.getFontMetrics(f1);
		g.setFont(f1);
		g.drawString (msg1, x, y);
		x = x + fm.stringWidth(msg1);
		g.setFont(f2);
		g.drawString (msg2, x, y);
		g.setFont(f1);
		y = 100;
		x = 20;
```

```
                int height = fm.getHeight();
                g.drawString (msg1, x, y);
                y += height;
                g.drawString (msg2, x, y);
        }
        public static void main (String args[])
        {
                new MultiFontMultiLine ();
        }
}
```

Kết quả của chương trình trên:



Hình 6.10 Văn bản được xuất nhiều font, nhiều dòng

6.9 Chọn mode để vẽ

Các đối tượng được vẽ bằng cách sử dụng mode vẽ. Khi một đối tượng mới được vẽ, nó sẽ đè lên các hình đã vẽ trước đây. Tương tự, khi các đối tượng được vẽ đi vẽ lại nhiều lần thì chúng sẽ xoá các đối tượng đã vẽ trước đó. Chỉ hiển thị nội dung của đối tượng mới. Để làm cho nội dung cũ và nội dung mới đều hiển thị trên màn hình, lớp Graphics cung cấp phương thức setXORMode (Color c);

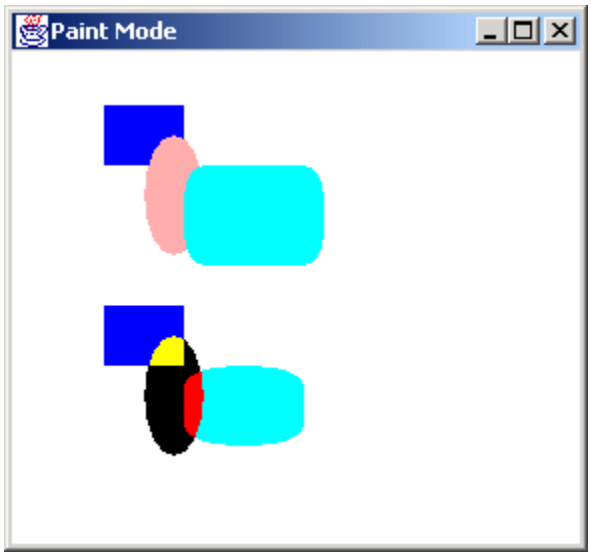
Chương trình 6.9 minh hoạ tiện lợi của của việc sử dụng phương thức setXORMode(). Ở đây, chúng ta sử dụng phương thức setXORMode() để tô các hình đồ hoạ khác nhau, mà không đè lên các hình khác. Kết quả là, khi sử dụng mode XOR thì hiển nhiên là tất cả các hình đều hiển thị đầy đủ. Điều này có nghĩa là các hình mới không đè lên các hình cũ. Thay vào đó, phần chung giữa các hình sẽ được hiển thị thành một màu khác. Nhưng khi không sử dụng mode XOR, hình mới hoàn toàn che khuất những hình trước đó.

Chương tìrnh 6.9

```
import java.awt.*;
class PaintMode extends Frame
{
        public PaintMode()
```

```
        {
                super ("Paint Mode");
                setSize (300, 300);
                setVisible (true);
        }
        public void paint (Graphics g)
        {
                g.setPaintMode ();
                g.setColor (Color.blue);
                g.fillRect (50,50,40, 30);
                g.setColor (Color.pink);
                g.fillOval (70, 65, 30, 60);
                g.setColor (Color.cyan);
                g.fillRoundRect (90, 80, 70, 50, 20, 30);
                g.setColor (Color.blue);
                g.fillRect (50, 150, 40, 30);
                g.setXORMode (Color.yellow);
                g.fillOval (70, 165, 30, 60);
                g.setXORMode (Color.magenta);
                g.fillRoundRect (90, 180, 60, 40, 50, 20);
        }

        public static void main (String args[])
        {
                new PaintMode();
        }
}
```

Kết quả của chương trình trên:



Hình 6.11 Paint mode

Tóm tắt

- Applet là chương trình Java chạy trong trình duyệt web.
- Chương trình Java đơn lẻ có thể vừa là applet, vừa là application.
- Lớp Graphics nằm trong gói AWT, bao gồm các phương thức được sử dụng để vẽ các hình đồ hoạ như oval, hình chữ nhật, hình vuông, hình tròn, đường thẳng và văn bản.
- Java sử dụng bảng màu RGB.
- Lớp Font trong gói java.awt cho phép sử dụng nhiều font khác nhau.
- Lớp FontMetrics xác định kích thước của các ký tự.

Chương 7 Xử lý ngoại lệ (Exception Handling)

Sau khi kết thúc chương này, bạn có thể nắm được các nội dung sau:

- Định nghĩa một ngoại lệ (exception)
- Hiểu được mục đích của việc xử lý ngoại lệ
- Hiểu được các kiểu ngoại lệ khác nhau trong Java
- Mô tả mô hình xử lý ngoại lệ
- Hiểu được các khối lệnh chứa nhiều catch
- Mô tả cách sử dụng các khối 'try', 'catch' và 'finally'
- Giải thích cách sử dụng các từ khoá 'throw' và 'throws'
- Tự tạo ra các ngoại lệ

7.1 Giới thiệu

Exception là một lỗi đặc biệt. Lỗi này xuất hiện vào lúc thực thi chương trình. Các trạng thái không bình thường xảy ra trong khi thi hành chương trình tạo ra các exception. Những trạng thái này không được biết trước trong khi ta đang xây dựng chương trình. Nếu bạn không phân phối các trạng thái này thì exception có thể bị kết thúc đột ngột. Ví dụ, việc chia cho 0 sẽ tạo một lỗi trong chương trình. Ngôn ngữ Java cung cấp bộ máy dùng để xử lý ngoại lệ rất tuyệt vời. Việc xử lý này làm hạn chế tối đa trường hợp hệ thống bị phá vỡ (crash) hay hệ thống bị ngắt đột ngột. Tính năng này làm cho Java là một ngôn ngữ lập trình mạnh.

7.2 Mục đích của việc xử lý ngoại lệ

Một chương trình nên có cơ chế xử lý ngoại lệ thích hợp. Nếu không, chương trình sẽ bị ngắt khi một exception xảy ra. Trong trường hợp đó, tất cả các nguồn tài nguyên mà hệ thống trước kia phân phối sẽ được di dời trong cùng trạng thái. Điều này gây lãng phí tài nguyên. Để tránh trường hợp này, tất cả các nguồn tài nguyên mà hệ thống phân phối nên được thu hồi lại. Tiến trình này đòi hỏi cơ chế xử lý ngoại lệ thích hợp.

Cho ví dụ, xét thao tác nhập xuất (I/O) trong một tập tin. Nếu việc chuyển đổi kiểu dữ liệu không thực hiện đúng, một ngoại lệ sẽ xảy ra và chương trình bị hủy mà không đóng lại tập tin. Lúc đó tập tin dễ bị hư hại và các nguồn tài nguyên được cấp phát cho tập tin không được thu hồi lại cho hệ thống.

7.3 Xử lý ngoại lệ

Khi một ngoại lệ xảy ra, đối tượng tương ứng với ngoại lệ đó được tạo ra. Đối tượng này sau đó được truyền cho phương thức là nơi mà ngoại lệ xảy ra. Đối tượng này chứa thông tin chi tiết về ngoại lệ. Thông tin này có thể được nhận về và được xử lý. Các môi trường runtime như 'IllegalAccessException', 'EmptyStackException' v.v... có thể chặn được các ngoại lệ. Đoạn mã trong chương trình đôi khi có thể tạo ra các ngoại lệ. Lớp 'throwable' được Java cung cấp là lớp trên nhất của lớp Exception , lớp này là lớp cha của các ngoại lệ khác nhau.

7.4 Mô hình xử lý ngoại lệ

Trong Java, mô hình xử lý ngoại lệ kiểm tra việc xử lý những hiệu ứng lề (lỗi), được biết đến là mô hình 'catch và throw'. Trong mô hình này, khi một lỗi xảy ra, một ngoại lệ sẽ bị chặn và được đưa vào trong một khối. Người lập trình viên nên xét các trạng thái ngoại lệ độc lập nhau từ việc điều khiển thông thường trong chương trình. Các ngoại lệ phải được bắt giữ nếu không chương trình sẽ bị ngắt.

Ngôn ngữ Java cung cấp 5 từ khoá sau để xử lý các ngoại lệ:

- try
- catch
- throw
- throws
- finally

Dưới đây là cấu trúc của mô hình xử lý ngoại lệ:

```
try
{
    // place code that is expected to throw an exception
}
catch(Exception e1)
{
    // If an exception is thrown in 'try', which is of type e1, then perform
    // necessary actions here, else go to the next catch block
}
catch(Exception e2)
{
    // If an exception is thrown in, try which is of type e2, then perform
```

```
    // necessary actions here, else go to the next catch block
}
catch(Exception eN)
{
    // If an exception is thrown in, try which is of type eN, then perform
    // necessary actions here, else go to the next catch block
}
finally
{
    // this book is executed, whether or not the exception is throw.
}
```

7.4.1 Các ưu điểm của mô hình 'catch và throw'

Mô hình 'catch và throw' có hai ưu điểm:

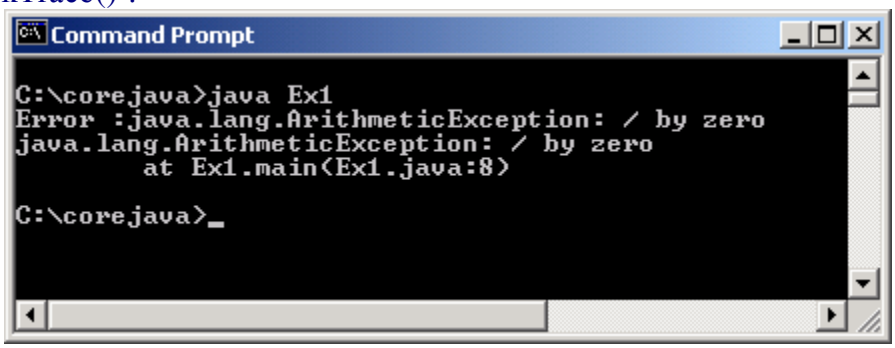
- Người lập trình viên phải phân phối trạng thái lỗi chỉ vào những nơi cần thiết. Không cần phải thực hiện tại mọi mức.
- Một thông báo lỗi có thể được in ra khi tiến hành xử lý ngoại lệ.

7.4.2 Các khối 'try' và 'catch'

Khối 'try-catch' được sử dụng để thi hành mô hình 'catch và throw' của việc xử lý ngoại lệ. Khối 'try' chứa một bộ các lệnh có thể thi hành được. Các ngoại lệ có thể bị chặn khi thi hành những câu lệnh này. Phương thức dùng để chặn ngoại lệ có thể được khai báo trong khối 'try'. Một hay nhiều khối 'catch' có thể theo sau khối 'try'. Các khối 'catch' này bắt các ngoại lệ bị chặn trong khối 'try'. Hãy nhìn khối 'try' dưới đây:

```
try
{
    doFileProcessing(); // user-defined method
    displayResults();
}
catch (Exception e) // exception object
{
    System.err.println("Error :" + e.toString());
    e.printStackTrace();
}
```

Ở đây, 'e' là đối tượng của lớp 'Exception'. Chúng ta có thể sử dụng đối tượng này để in các chi tiết về ngoại lệ. Các phương thức 'toString' và 'printStackTrace' được sử dụng để mô tả các exception phát sinh ra. Hình sau chỉ ra kết xuất của phương thức 'printStackTrace()'.



Hình 7.1 Khối Try và Catch

Để bắt giữ bất cứ ngoại lệ nào, ta phải chỉ ra kiểu ngoại lệ là 'Exception'.

catch(Exception e)

Khi ngoại lệ bị bắt giữ không biết thuộc kiểu nào, chúng ta có thể sử dụng lớp 'Exception' để bắt ngoại lệ đó.

Khối 'catch()' bắt giữ bất cứ các lỗi xảy ra trong khi thi hành phương thức 'doFileProcessing' hay 'display'. Nếu một lỗi xảy ra trong khi thi hành phương thức 'doFileProcessing()', lúc đó phương thức 'displayResults()' sẽ không bao giờ được gọi. Sự thi hành sẽ tiếp tục thực hiện khối 'catch'. Để có nhiều lớp xử lý lỗi hơn, như là 'LookupException' thay vì một đối tượng ngoại lệ chung (Exception e), lỗi thật sự sẽ là một instance của 'LookupException' hay một trong số những lớp con của nó. Lỗi sẽ được truyền qua khối 'try catch' cho tới khi chúng bắt gặp một 'catch' tham chiếu tới nó hay toàn bộ chương trình phải bị huỷ bỏ.

7.5 Các khối chứa nhiều Catch

Các khối chứa nhiều 'catch' xử lý các kiểu ngoại lệ khác nhau một cách độc lập. Chúng được liệt kê trong đoạn mã sau:

```
try
{
    doFileProcessing(); // user defined mothod
    displayResults(); // user defined method
}
```

```
catch(LookupException e) // e – Lookupexception object
{
    handleLookupException(e); // user defined handler
}
catch(Exception e)
{
    System.err.println("Error:" + e.printStackTrace());
}
}
```

Trong trường hợp này, khối 'catch' đầu tiên sẽ bắt giữ một 'LockupException'. Khối 'catch' thứ hai sẽ xử lý kiểu ngoại lệ khác với khối 'catch' thứ nhất.

Một chương trình cũng có thể chứa các khối 'try' lồng nhau. Ví dụ đoạn mã dưới đây:

```
try
{
    statement 1;
    statement 2;
    try
    {
        statement1;
        statement2;
    }
    catch(Exception e) // of the inner try block
    {

    }
}
catch(Exception  e) // of the outer try block
{
}
...
```

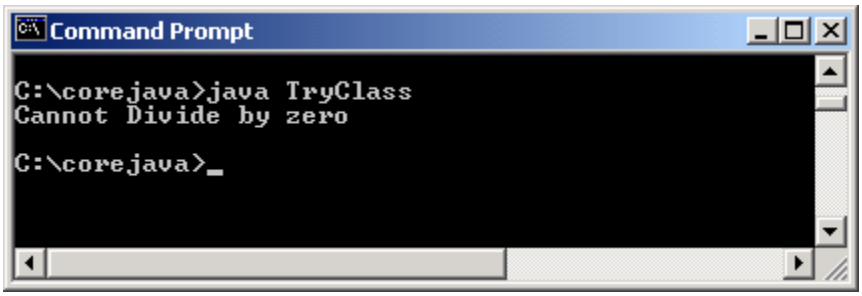
Khi sử dụng các 'try' lồng nhau, khối 'try' bên trong được thi hành đầu tiên. Bất kỳ ngoại lệ nào bị chặn trong khối 'try' sẽ bị bắt giữ trong các khối 'catch' theo sau. Nếu khối 'catch' thích hợp không được tìm thấy thì các khối 'catch' của các khối 'try' bên ngoài sẽ được xem xét. Nếu không, Java Runtime Environment xử lý các ngoại lệ.

chương trình 7.1 minh họa cách sử dụng các khối 'try' và 'catch'.

Chương trình 7.1

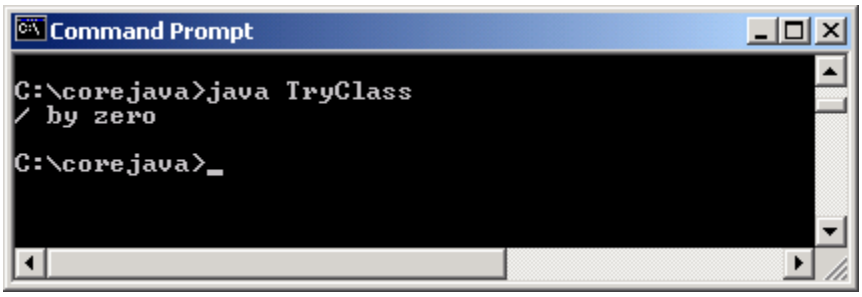
```
class TryClass
{
    public static void main(String args[])
    {
        int demo=0;
        try
        {
                System.out.println(20/demo);
        }
        catch(ArithmeticException a)
        {
                System.out.println("Cannot Divide by zero");
        }
    }
}
```

Kết xuất của chương trình:



Hình 7.2 ArithmeticException

Trong chương trình này, một số được chia cho 0. Đây không là toán tử số học hợp lệ. Do đó một ngoại lệ bị chặn và được bắt giữ trong khối catch. Khi người lập trình viên nhận biết được loại ngoại lệ nào có thể xảy ra, anh ta hay cô ta viết một câu lệnh trong khối 'catch'. Ở đây, 'a' được sử dụng như một đối tượng của Arithmaticexception để in các chi tiết về các toán tử ngoại lệ mà hệ thống cung cấp. Nếu bạn thay thế lệnh 'System.out.println' của khối 'catch' bằng lệnh '**System.out.println(a.getMessage())**' thì kết xuất của chương trình như sau:



Hình 7.3 Câu thông báo lỗi

Khi các khối 'try' được sử dụng mà không có các khối 'catch' nào, chương trình sẽ biên dịch mà không gặp sự cố nào nhưng sẽ bị ngắt khi thực thi. Bởi vì ngoại lệ đã xảy ra khi thực thi chương trình.

7.6 Khối 'finally'

Khi một ngoại lệ xuất hiện, phương thức đang được thực thi có thể bị dừng mà không được thi hành toàn vẹn. Nếu điều này xảy ra, thì các đoạn mã (ví dụ như đoạn mã với chức năng thu hồi tài nguyên có các lệnh đóng lại tập tin khai báo cuối phương thức) sẽ không bao giờ được gọi. Java cung cấp khối 'finally' để giải quyết việc này. Khối 'finally' thực hiện tất cả các việc thu dọn khi một ngoại lệ xảy ra. Khối này có thể được sử dụng kết hợp với khối 'try'. Khối 'finally' chứa các câu lệnh thu hồi tài nguyên về cho hệ thống hay lệnh in ra các câu thông báo. Các lệnh này bao gồm:

- Đóng tập tin.
- Đóng lại bộ kết quả (được sử dụng trong chương trình cơ sở dữ liệu).
- Đóng lại các kết nối được tạo trong cơ sở dữ liệu.

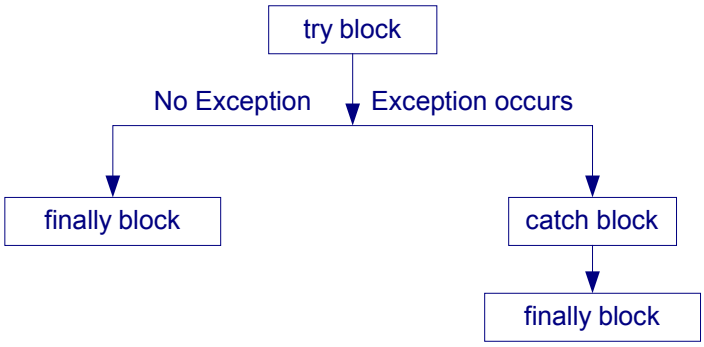
```
try
{
    doSomethingThatMightThrowAnException();
}
finally
{
    cleanup();
}
```

Phương thức 'cleanup()' được gọi nếu phương thức 'doSomethingThatMightThrowAnException()' chặn một ngoại lệ. Mặt khác 'cleanup()' cũng được gọi ngay khi không có ngoại lệ nào bị chặn và thi hành tiếp tục sau khối lệnh 'finally'.

Khối ‘finally’ là tuỳ ý, không bắt buộc. Khối này được đặt sau khối ‘catch’. Hệ thống sẽ duyệt từ câu lệnh đầu tiên của khối ‘finally’ sau khi gặp câu lệnh ‘return’ hay lệnh ‘break’ được dùng trong khối ‘try’.

Khối ‘finally’ bảo đảm lúc nào cũng được thực thi, bất chấp có ngoại lệ xảy ra hay không.

Hình 7.4 minh họa sự thi hành của các khối ‘try’, ‘catch’ và ‘finally’.



Hình 7.4 Khối lệnh ‘try’, ‘catch’ và ‘finally’

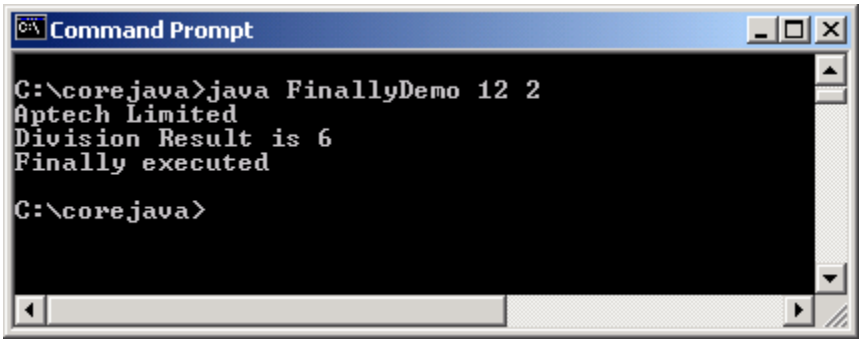
Hình 7.2 sử dụng khối ‘finally’. Ở đây, khối ‘finally’ được thi hành bất chấp ‘ArithmeticException’ có xảy ra hay không. Khối này khai báo các hoạt động thu dọn.

Chương trình 7.2

```
class FinallyDemo
{
    String name;
    int no1,no2;
    FinallyDemo(String args[])
    {
        try
        {
                name=new String("Aptech Limited");
                no1=Integer.parseInt(args[0]);
                no2=Integer.parseInt(args[1]);
                System.out.println(name);
                System.out.println("Division Result is" + no1/no2);
        }
        catch(ArithmeticException i)
```

```
        {
                System.out.println("Cannot Divide by zero");
        }
        finally
        {
                name=null; // clean up code
                System.out.println("Finally executed");
        }
    }
    public static void main(String args[])
    {
        new FinallyDemo(args);
    }
}
```

Kết xuất của chương trình:



Hình 7.5 Khối Finally

Trong ví dụ này, các câu lệnh trong khối 'Finally' luôn luôn thi hành, bất chấp ngoại lệ có xảy ra hay không. Trong kết xuất bên trên, khối 'finally' được thi hành mặc dù không có ngoại lệ xảy ra.

7.7 Các ngoại lệ được định nghĩa với lệnh 'throw' và 'throws'

Các ngoại lệ bị chặn với sự trợ giúp của từ khoá 'throw'. Từ khóa 'throw' chỉ ra một ngoại lệ vừa xảy ra. Toán tử của throw là một đối tượng của lớp, lớp này được dẫn xuất từ 'Throwable'.

Đoạn lệnh sau chỉ ra cách sử dụng của lệnh 'throw':

```
try
```

```
{
    if (flag<0)
    {
        throw new MyException(); // user-defined
    }
}
```

Một phương thức đơn có thể chặn nhiều ngoại lệ. Để xử lý những ngoại lệ này, ta cần cung cấp một danh sách các ngoại lệ mà phương thức chặn trong phần định nghĩa của phương thức. Giả sử rằng phương thức 'x()' gọi phương thức 'y()'. Phương thức 'y()' chặn một ngoại lệ không được xử lý. Trong trường hợp này, phương thức gọi 'x()' nên khai báo việc chặn cùng một ngoại lệ với phương thức được gọi 'y()'. Ta nên khai báo khối 'try catch' trong phương thức x() để đảm bảo rằng ngoại lệ không được truyền cho các phương thức mà gọi phương thức này.

Đoạn mã sau minh họa cách sử dụng của từ khoá 'throws' để xử lý nhiều ngoại lệ:

```
public class Example
{
    // multiple exceptions separated by a comma
    public void exceptionExample() throws ExException, LookupException
    {
        try
        {
                // statements
        }
        catch(ExException exmp)
        {
        }
        catch(LookupException lkpex)
        {
        }
    }
}
```

Trong ví dụ trên, phương thức 'exceptionExample' khai báo từ khoá 'throws'. Từ khoá này được theo sau bởi danh sách các ngoại lệ mà phương thức này có thể chặn – Trong trường hợp này là 'ExException' và 'LookupException'. Hàm xử lý ngoại lệ cho các phương thức này nên khai báo các khối 'catch' để có thể xử lý tất cả các ngoại lệ mà các phương thức chặn.

Lớp 'Exception' thực thi giao diện 'Throwable' và cung cấp các tính năng hữu dụng để phân phối các ngoại lệ. Ưu điểm của nó là tạo các lớp ngoại lệ được định nghĩa bởi người dùng. Để làm điều này, một lớp con của lớp Exception được tạo ra. Ưu điểm của lớp con là một kiểu ngoại lệ mới có thể bị bắt giữ độc lập từ các loại Throwable khác.

Chương trình 7.3 minh họa ngoại lệ được định nghĩa bởi người dùng 'ArraySizeException':

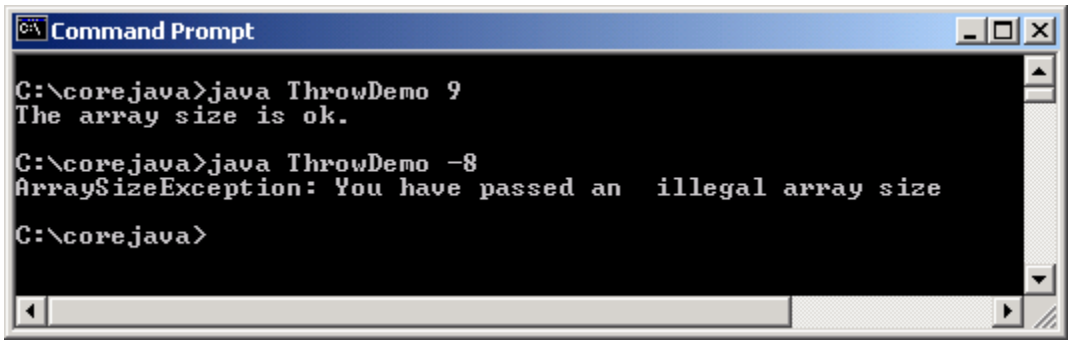
Chương trình 7.3

```
class ArraySizeException extends NegativeArraySizeException
{
    ArraySizeException() // constructor
    {
        super("You have passed an  illegal array size");
    }
}
class ThrowDemo
{
    int size, array[];
    ThrowDemo(int s)
    {
        size=s;
        try
        {
                checkSize();
        }
        catch(ArraySizeException e)
        {
                System.out.println(e);
        }
    }
    void checkSize() throws ArraySizeException
    {
        if (size < 0)
                throw new ArraySizeException();
        else
                System.out.println("The array size is ok.");
        array = new int[3];
        for (int i=0; i<3; i++)
                array[i] = i+1;
```

```
    }
    public static void main(String arg[])
    {
        new ThrowDemo(Integer.parseInt(arg[0]));
    }
}
```

Lớp được định nghĩa bởi người dùng 'ArraySizeException' là lớp con của lớp 'NegativeArraySizeException'. Khi một đối tượng được tạo từ lớp này, thông báo về ngoại lệ được in ra. Phương thức 'checkSize()' được gọi để chặn ngoại lệ 'ArraySizeException' mà được chỉ ra bởi mệnh đề 'throws'. Kích thước của mảng được kiểm tra trong cấu trúc 'if'. Nếu kích thước là số âm thì đối tượng của lớp 'ArraySizeException' được tạo. Phương thức 'call()' được bao quanh trong khối 'try-catch', là nơi mà giá trị của đối tượng được in ra. Phương thức 'call()' cần được bao trong khối 'try', để cho khối 'catch' tương ứng có thể in ra giá trị.

Kết xuất của chương trình được chỉ ra ở hình 7.6.



Hình 7.6 Ngoại lệ tự định nghĩa

7.8 Danh sách các ngoại lệ

Bảng sau đây liệt kê một số ngoại lệ:

Ngoại lệ	**Lớp cha của thứ tự phân cấp ngoại lệ**
RuntimeException	Lớp cơ sở cho nhiều ngoại lệ java.lang
ArthmeticException	Trạng thái lỗi về số, ví dụ như 'chia cho 0'
IllegalAccessException	Lớp không thể truy cập
IllegalArgumentException	Phương thức nhận một đối số không hợp lệ
ArrayIndexOutOfBoundsExeption	Kích thước của mảng lớn hơn 0 hay lớn hơn kích thước thật sự của mảng

NullPointerException	Khi muốn truy cập đối tượng null
SecurityException	Việc thiết lập cơ chế bảo mật không được hoạt động
ClassNotFoundException	Không thể nạp lớp yêu cầu
NumberFormatException	Việc chuyển đối không thành công từ chuỗi sang số thực
AWTException	Ngoại lệ về AWT
IOException	Lớp cha của các ngoại lệ I/O
FileNotFoundException	Không thể định vị tập tin
EOFException	Kết thúc một tập tin
NoSuchMethodException	Phương thức yêu cầu không tồn tại
InterruptedException	Khi một luồng bị ngắt

Bảng 7.1 Danh sách một số ngoại lệ

Tóm tắt

- Bất cứ khi nào một lỗi xuất hiện trong khi thi hành chương trình, nghĩa là một ngoại lệ đã xuất hiện.
- Ngoại lệ phát sinh vào lúc thực thi chương trình theo trình tự mã.
- Mỗi ngoại lệ phát sinh ra phải bị bắt giữ , nếu không ứng dụng sẽ bị ngắt.
- Việc xử lý ngoại lệ cho phép bạn kết hợp tất cả tiến trình xử lý lỗi trong một nơi. Lúc đó đoạn mã của bạn sẽ rõ ràng hơn.
- Java sử dụng các khối 'try' và 'catch' để xử lý các ngoại lệ. Các câu lệnh trong khối 'try' chặn ngoại lệ còn khối 'catch' xử lý ngoại lệ.
- Các khối chứa nhiều catch có thể được sử dụng để xử lý các kiểu ngoại lệ khác nhau theo cách khác nhau.
- Từ khoá 'throw' liệt kê các ngoại lệ mà phương thức chặn.
- Từ khoá 'throw' chỉ ra một ngoại lệ vừa xuất hiện.
- Khối 'finally' khai báo các câu lệnh trả về nguồn tài nguyên cho hệ thống và in những câu thông báo.

ĐA TUYẾN

Mục tiêu:

Sau khi kết thúc chưiưng này, bạn có thể:

- Định nghĩa một luồng
- Mô tả đa tuyến
- Tạo và quản lý luồng
- Hiểu được vòng đời của luồng
- Mô tả một luồng hiểm
- Giải thích tập hợp các luồng ưu tiên như thế nào
- Giải thích được sự cần thiết của sự đồng bộ
- Hiểu được cách thêm vào các từ khoá synchronized (đồng bộ) như thế nào
- Liệt kê những điều không thuận lợi của sự đồng bộ
- Giải thích vai trò của các phương thức wait() (đợi), notify() (thông báo) và notifyAll().
- Mô tả một điều kiện bế tắc (deadlock).

1. Giới thiệu

Một luồng là một thuộc tính duy nhất của Java. Nó là đơn vị nhỏ nhất của đoạn mã có thể thi hành được mà thực hiện một công việc riêng biệt. Ngôn ngữ Java và máy ảo Java cả hai là các hệ thống đươc phân luồng

2. Đa tuyến

Java hổ trợ đa tuyến, mà có khả năng làm việc với nhiều luồng. Một ứng dụng có thể bao hàm nhiều luồng. Mỗi luồng được đăng ký một công việc riêng biệt, mà chúng được thực thi đồng thời với các luồng khác.

Đa tuyến giữ thời gian nhàn rỗi của hệ thống thành nhỏ nhất. Điều này cho phép bạn viết các chương trình có hiệu quả cao với sự tận dụng CPU là tối đa. Mỗi phần của chương trình được gọi một luồng, mỗi luồng định nghĩa một đường dẫn khác nhau của sự thực hiện. Đây là một thiết kế chuyên dùng của sự đa nhiệm.

Trong sự đa nhiệm, nhiều chương chương trình chạy đồng thời, mỗi chương trình có ít nhất một luồng trong nó. Một vi xử lý thực thi tất cả các chương trình. Cho dù nó có thể xuất hiện mà các chương trình đã được thực thi đồng thời, trên thực tế bộ vi xử lý nhảy qua lại giữa các tiến trình.

3. Tạo và quản lý luồng

Khi các chương trình Java được thực thi, luồng chính luôn luôn đang được thực hiện. Đây là 2 nguyên nhân quan trọng đối với luồng chính:

- Các luồng con sẽ được tạo ra từ nó.
- Nó là luồng cuối cùng kết thúc việc thực hiện. Trong chốc lát luồng chính ngừng thực thi, chương trình bị chấm dứt.

Cho dù luồng chính được tạo ra một cách tự động với chương trình thực thi, nó có thể được điều khiển thông qua một luồng đối tượng.

Các luồng có thể được tạo ra từ hai con đường:

- Trình bày lớp như là một lớp con của lớp luồng, nơi mà phương thức run() của lớp luồng cần được ghi đè. Lấy ví dụ:

```
Class Mydemo extends Thread
{
        //Class definition
        public void run()
        {
                //thực thi
        }
}
```

- Trình bày một lớp mà lớp này thực hiện lớp Runnable. Rồi thì định nghĩa phương thức run().

```
Class Mydemo implements Runnable
{
        //Class definition
        public void run()
        {
                //thực thi
        }
}
```

Chương trình 8.1 sẽ chỉ ra sự điều khiển luồng chính như thế nào

Chương trình 8.1

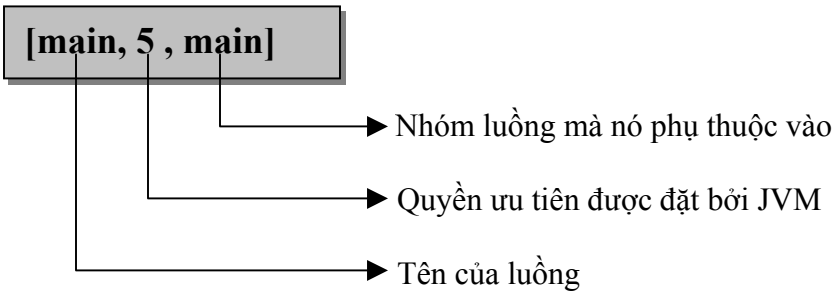
```
import java.io.*;
public class Mythread extends Thread{
/**
 * Mythread constructor comment.
 */
        public static void main(String args[]){
                Thread t = Thread.currentThread();
                System.out.println("The current Thread is :" + t);
                t.setName("MyJavaThread");
                System.out.println("The thread is now named: " + t);
                try{
                        for(int i = 0; i <3;i++){
                                System.out.println(i);
                                Thread.sleep(1500);
                        }
                }catch(InterruptedException e){
                        System.out.println("Main thread interupted");
                }
        }
}
```

Hình sau đây sẽ chỉ ra kết quả xuất ra màn hình của chương trình trên

```
Output
The current Thread is :Thread[main,5,main]
The thread is now named: Thread[MyJavaThread,5,main]
0
1
2
```

Hình 8.1 Luồng

Trong kết quả xuất ra ở trên



Mỗi luồng trong chương trình Java được đăng ký cho một quyền ưu tiên. Máy ảo Java không bao giờ thay đổi quyền ưu tiên của luồng. Quyền ưu tiên vẫn còn là hằng số cho đến khi luồng bị ngắt.

Mỗi luồng có một giá trị ưu tiên nằm trong khoảng của một Thread.MIN_PRIORITY của 1, và một Thread.MAX_PRIORITY của 10. Mỗi luồng phụ thuộc vào một nhóm luồng, và mỗi nhóm luồng có quyền ưu tiên của chính nó. Mỗi luồng được nhận một hằng số ưu tiên của phương thức Thread.PRIORITY là 5. Mỗi luồng mới thừa kế quyền ưu tiên của luồng mà tạo ra nó.

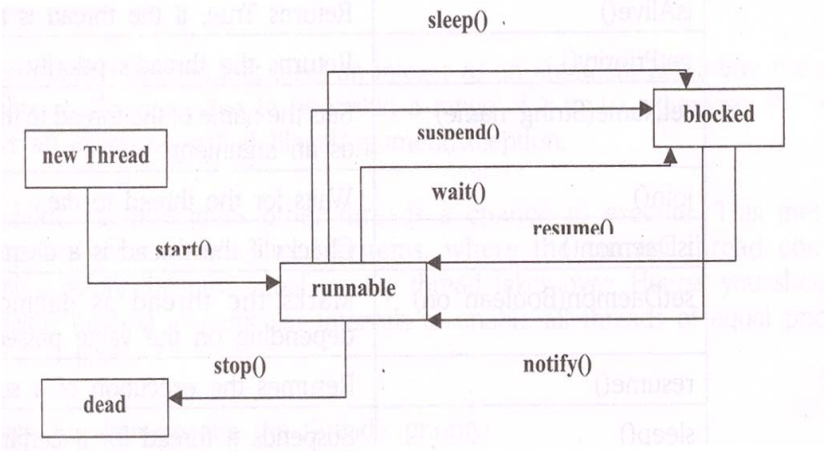
Lớp luồng có vài phương thức khởi dựng, hai trong số các phương thức khởi dựng được đề cập đến dưới đây:

- **public Thread(String threadname)**
  Cấu trúc một luồng với tên là "threadname"
- **public Thread()**

Cấu trúc một luồng với tên "Thread, được ràng buộc với một số; lấy ví dụ, Thread-1, Thread-2, v.v…

Chương trình bắt đầu thực thi luồng với việc gọi phương thức start(), mà phương thức này phụ thuộc vào lớp luồng. Phương thức này, lần lượt, viện dẫn phương thức run(), nơi mà phương thức định nghĩa tác vụ được thực thi. Phương thức này có thể viết đè lên lớp con của lớp luồng, hoặc với một đối tượng Runnable.

4. Vòng đời của Luồng



Hình 8.3 Vòng đời của luồng

5. Phạm vi của luồng và các phương thức của lớp luồng

Một luồng đã được tạo mới gần đây là trong phạm vi "sinh". Luồng không bắt đầu chạy ngay lập tức sau khi nó được tạo ra. Nó đợi phương thức start() của chính nó được gọi. Cho đến khi, nó là trong phạm vi "sẵn sàng để chạy". Luồng đi vào phạm vi "đang chay" khi hệ thống định rõ vị trí luồng trong bộ vi xử lý.

Bạn có thể sử dụng phương thức sleep() để tạm thời treo sự thực thi của luồng. Luồng trở thành sẵn sàng sau khi phương thức sleep kết thúc thời gian. Luồng Sleeping không sử dụng bộ vi xử lý. luồng đi vào phạm vi "waiting" (đợi) khi một luồng đang chạy gọi phương thức wait() (đợi).

Khi các luồng khác liên kết với các đối tượng, gọi phương thức notify(), luồng đi vào trở lại phạm vi "ready" (sẵn sàng) Luồng đi vào phạm vi "blocked" (khối) khi nó đang thực thi các phép toán vào/ra (Input/output). Nó đi vào phạm vi "ready" (sẵn sàng) khi các phương thức vào/ra nó đang đợi cho đến khi được hoàn thành. Luồng đi vào phạm vi "dead" (chết) sau khi phương thức run() đã được thực thi hoàn toàn, hoặc khi phương thức stop() (dừng) của nó được gọi.

Thêm vào các phương thức đã được đề cập trên, Lớp luồng cũng có các phương thức sau:

Phương thức	**Mục đích**
Enumerate(Thread t)	Sao chép tất cả các luồng hiện hành vào mảng được chỉ định từ nhóm của các luồng, và các nhóm con của nó.
getName()	Trả về tên của luồng
isAlive()	Trả về Đúng, nếu luồng là vẫn còn tồn tại (sống)
getPriority()	Trả về quyền ưu tiên của luồng
setName(String name)	Đặt tên của luồng là tên mà luồng được truyền như là một tham số.
join()	Đợi cho đến khi luồng chết.
isDaemon(Boolean on)	Kiểm tra nếu luồng là luồng một luồng hiếm.
resume()	Đánh dấu luồng như là luồng hiếm hoặc luồng người sử dụng phụ thuộc vào giá trị được truyền vào.
sleep()	Hoãn luồng một khoảng thời gian chính xác.
start()	Gọi phương thức run() để bắt đầu một luồng.

Bảng 8.1 Các phương thức của một lớp luồng

Bảng kế hoạch Round-robin (bảng kiến nghị ký tên vòng tròn) liên quan đến các luồng với cùng quyền ưu tiên được chiếm hữu quyền ưu tiên của mỗi luồng khác. Chúng chia nhỏ thời gian một cách tự động trong theo kiểu kế hoạch xoay vòng này.

Phiên bản mới nhất của Java không hổ trợ các phương thức Thread.suspend() (trì hoãn), Thread.resume() (phục hồi) và Thread.stop() (dừng), như là các phương thức resume() (phục hồi) và suspend() (trì hoãn) được thiên về sự đình trệ (deadlock), trong khi phương thức stop() không an toàn.

6. Thời gian biểu luồng

Hầu hết các chương trình Java làm việc với nhiều luồng. CPU chứa đựng cho việc chạy chương trình chỉ một luồng tại một khoảng thời gian. Hai luồng có cùng quyền ưu tiên trong một chương trình hoàn thành trong một thời gian CPU. Lập trình viên, hoặc máy ảo Java, hoặc hệ điều hành chắc chắn rằng CPU được chia sẻ giữa các luồng. Điều này được biết như là bảng thời gian biểu luồng.

Không có máy ảo Java nào thực thi rành mạch cho bảng thời gian biểu luồng. Một số nền Java hổ trợ việc chia nhỏ thời gian. Ở đây, mỗi luồng nhận một phần nhỏ của thời gian bộ vi xử lý, được gọi là định lượng. Luồng có thể thực thi tác vụ của chính nó trong suốt khoảng thời gian định lượng đấy. Sau khoảng thời gian này được vượt qua, luồng không được nhận nhiều thời gian để tiếp tục thực hiện, ngay cả nếu nó không được hoàn thành việc thực hiện của nó. Luồng kế tiếp của luồng có quyền ưu tiên bằng nhau này sẽ lấy khoảng thời gian thay đổi của bộ vi xử lý. Java là người lập thời gian biểu chia nhỏ tất cả các luồng có cùng quyền ưu tiên cao.

Phương thức setPriority() lấy một số nguyên (integer) như là một tham số có thể hiệu chỉnh quyền ưu tiên của một luồng. Đây là giá trị có phạm vi thay đổi từ 1 đến 10, mặc khác, phương thức đưa ra một ngoại lệ (bẫy lỗi) được gọi là IllegalArgumentException (Chấp nhận tham số trái luật)

Phương thức yield() (lợi nhuận) đưa ra các luồng khác một khả năng để thực thi. Phương thức này được thích hợp cho các hệ thống không chia nhỏ thời gian (non-time-sliced), nơi mà các luồng hiện thời hoàn thành việc thực hiện trước khi các luồng có quyền ưu tiên ngang nhau kế tiếp tiếp quản. Ở đây, bạn sẽ gọi phương thức yield() tại những khoản thời gian riêng biệt để có thể tất cả các luồng có quyền ưu tiên ngang nhau chia sẻ thời gian thực thi CPU.

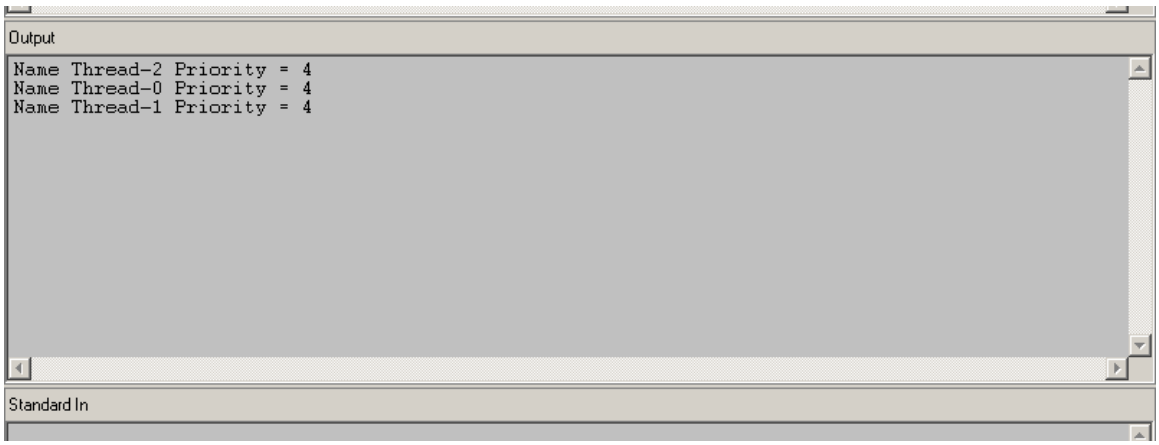
Chương trình 8.2 chứng minh quyền ưu tiên của luồng:

Chương trình 8.2

```
class PriorityDemo {
        Priority t1,t2,t3;
        public PriorityDemo(){
                t1 = new Priority();
                t1.start();
                t2 = new Priority();
                t2.start();
                t3 = new Priority();
                t3.start();
        }
        public static void main(String args[]){
```

```
                new PriorityDemo();
        }
        class Priority extends Thread implements Runnable{
                int sleep;
                int prio = 3;
                public Priority(){
                        sleep += 100;
                        prio++;
                        setPriority(prio);
                }
                public void run(){
                        try{
                                Thread.sleep(sleep);
                                System.out.println("Name "+ getName()+" Priority
= "+ getPriority());
                        }catch(InterruptedException e){
                                System.out.println(e.getMessage());
                        }
                }
        }

}
```

Kết quả hiển thị như hình 8.4



Hình 8.4 Quyền ưu tiên luồng

7. Luồng hiểm

Một chương trình Java bị ngắt chỉ sau khi tất cả các luồng bị chết. Có hai kiểu luồng trong một chương trình Java:

- Các luồng người sử dụng
- Luồng hiểm

Người sử dụng tạo ra các luồng người sử dụng, trong khi các luồng được chỉ định như là luồng "background" (nền). Luồng hiểm cung cấp các dịch vụ cho các luồng khác. Máy ảo Java thực hiện tiến trình thoát, khi và chỉ khi luồng hiểm vẫn còn sống. Máy ảo

Java có ít nhất một luồng hiểm được biết đến như là luồng "garbage collection" (thu lượm những dữ liệu vô nghĩa - dọn rác). Luồng dọn rác thực thi chỉ khi hệ thống không có tác vụ nào. Nó là một luồng có quyền ưu tiên thấp. Lớp luồng có hai phương thức để thỏa thuận với các luồng hiểm:

- public void setDaemon(boolean on)
- public boolean isDaemon()

8. Đa tuyến với Applets

Trong khi đa tuyến là rất hữu dụng trong các chương trình ứng dụng độc lập, nó cũng đáng được quan tâm với các ứng dụng trên Web. Đa tuyến được sử dụng trên web, cho ví dụ, trong các trò chơi đa phương tiện, các bức ảnh đầy sinh khí, hiển thị các dòng chữ chạy qua lại trên biểu ngữ, hiển thị đồng hồ thời gian như là một phần của trang Web v.vv… Các chức năng này cấu thành các trang web làm quyến rũ và bắt mắt.

Chương trình Java dựa trên Applet thường sử dụng nhiều hơn một luồng. Trong đa tuyến với Applet, lớp java.applet.Applet là lớp con được tạo ra bởi người sử dụng định nghĩa applet. Từ đó, Java không hỗ trợ nhiều kế thừa với các lớp, nó không thể thực hiện được trực tiếp lớp con của lớp luồng trong các applet. Tuy nhiên, chúng ta sử dụng một đối tượng của luồng người sử dụng đã định nghĩa, mà các luồng này, lần lượt, dẫn xuất từ lớp luồng. Một luồng đơn giản xuất hiện sẽ được thực thi tại giao diện (Interface) Runnable

Chương trình 8.3 chỉ ra điều này thực thi như thế nào:

Chương trình 8.3

```
import java.awt.*;
import java.applet.*;
public class Myapplet extends Applet implements Runnable {
        int i;
        Thread t;
/**
 * Myapplet constructor comment.
 */
        public void init(){
                t = new Thread(this);
                t.start();
        }
        public void paint(Graphics g){
                g.drawString(" i = "+i,30,30);
        }
        public void run(){
                for(i = 1;i<=20;i++){
                        try{
                                repaint();
                                Thread.sleep(500);
                        }catch(InterruptedException e){
                                System.out.println(e.getMessage());
                        }
```

```
            }
        }
    }
```

Trong chương trình này, chúng ta tạo ra một Applet được gọi là Myapplet, và thực thi giao diện Runnable để cung cấp khả năng đa tuyến cho applet. Sau đó, chúng ta tạo ra một thể nghiệm (instance) cho lớp luồng, với thể nghiệm applet hiện thời như là một tham số để thiết lập (khởi dựng). Rồi thì chúng ta viện dẫn phương thức start() của luồng thể nghiệm này. Lần lượt, rồi sẽ viện dẫn phương thức run(), mà phương thức này thực sự là điểm bắt đầu cho phương thức này. Chúng ta in số từ 1 đến 20 với thời gian kéo trễ là 500 miligiây giữa mỗi số. Phương thức sleep() được gọi để hoàn thành thời gian kéo trễ này. Đây là một phương thức tĩnh được định nghĩa trong lớp luồng. Nó cho phép luồng nằm yên (ngủ) trong khoản thời gian hạn chế.

Xuất ra ngoài có dạng như sau:

Hình 8.5 Đa tuyến với Applet

9. Nhóm luồng

Một lớp nhóm luồng (ThreadGroup) nắm bắt một nhóm của các luồng. Lấy ví dụ, một nhóm luồng trong một trình duyệt có thể quản lý tất cả các luồng phụ thuộc vào một đơn thể applet. Tất cả các luồng trong máy ảo Java phụ thuộc vào các nhóm luồng mặc định. Mỗi nhóm luồng có một nhóm nguồn cha. Vì thế, các nhóm từ một cấu trúc dạng cây. Nhóm luồng "hệ thống" là gốc của tất cả các nhóm luồng. Một nhóm luồng có thể là thành phần của cả các luồng, và các nhóm luồng.

Hai kiểu nhóm luồng thiết lập (khởi dựng) là:

- **public ThreadGroup(String str)**

Ở đây, "str" là tên của nhóm luồng mới nhất được tạo ra.

- **public ThreadGroup(ThreadGroup tgroup, String str)**

Ở đây, "tgroup" chỉ ra luồng đang chạy hiện thời như là luồng cha, "str" là tên của nhóm luồng đang được tạo ra.

Một số các phương thức trong nhóm luồng (ThreadGroup) được cho như sau:

- **public synchronized int activeCount()**

Trả về số lượng các luồng kích hoạt hiện hành trong nhóm luồng

- **public sunchronized int activeGroupCount()**

Trả về số lượng các nhóm hoạt động trong nhóm luồng

- **public final String getName()**

Trả về tên của nhóm luồng

- **public final ThreadGroup getParent()**

Trả về cha của nhóm luồng

10. Sự đồng bộ luồng

Trong khi đang làm việc với nhiều luồng, nhiều hơn một luồng có thể muốn thâm nhập cùng biến tại cùng thời điểm. Lấy ví dụ, một luồng có thể cố gắng đọc dữ liệu, trong khi luồng khác cố gắng thay đổi dữ liệu. Trong trường hợp này, dữ liệu có thể bị sai lạc.

Trong những trường hợp này, bạn cần cho phép một luồng hoàn thành trọn vẹn tác vụ của nó (thay đổi giá trị), và rồi thì cho phép các luồng kế tiếp thực thi. Khi hai hoặc nhiều hơn các luồng cần thâm nhập đến một tài nguyên được chia sẻ, bạn cần chắc chắn rằng tài nguyên đó sẽ được sử dụng chỉ bởi một luồng tại một thời điểm. Tiến trình này được gọi là "sự đồng bộ" (synchronization) được sử dụng để lưu trữ cho vấn đề này, Java cung cấp duy nhất, ngôn ngữ cấp cao hổ trợ cho sự đồng bộ này. Phương thức "đồng bộ" (synchronized) báo cho hệ thống đặt một khóa vòng một tài nguyên riêng biệt.

Mấu chốt của sự đồng bộ hóa là khái niệm "monitor" (sự quan sát, giám sát), cũng được biết như là một bảng mã "semaphore" (bảng mã). Một "monitor" là một đối tượng mà được sử dụng như là một khóa qua lại duy nhất, hoặc "mutex". Chỉ một luồng có thể có riêng nó một sự quan sát (monitor) tại mỗi thời điểm được đưa ra. Tất cả các luồng khác cố gắng thâm nhập vào monitor bị khóa sẽ bị trì hoãn, cho đến khi luồng đầu tiên thoát khỏi monitor. Các luồng khác được báo chờ đợi monitor. Một luồng mà monitor của riêng nó có thể thâm nhập trở lại cùng monitor.

1. Mã đồng bộ

Tất cả các đối tượng trong Java được liên kết với các monitor (sự giám sát) của riêng nó. Để đăng nhập vào monitor của một đối tượng, lập trình viên sử dụng từ khóa synchronized (đồng bộ) để gọi một phương thức hiệu chỉnh (modified). Khi một luồng đang được thực thi trong phạm vi một phương thức đồng bộ (synchronized), bất kỳ luồng khác hoặc phương thức đồng bộ khác mà cố gắng gọi nó trong cùng thể nghiệm sẽ phải đợi.

Chương trình 8.4 chứng minh sự làm việc của từ khóa synchronized (sự đồng bộ). Ở đây, lớp "Target" (mục tiêu) có một phương thức "display()" (hiển thị) mà phương thức này lấy một tham số kiểu số nguyên (int). Số này được hiển thị trong phạm vi các cặp ký tự "< > # số # <>". Phương thức "Thread.sleep(1000) tạm dừng luồng hiện tại sau khi phương thức "display()" được gọi.

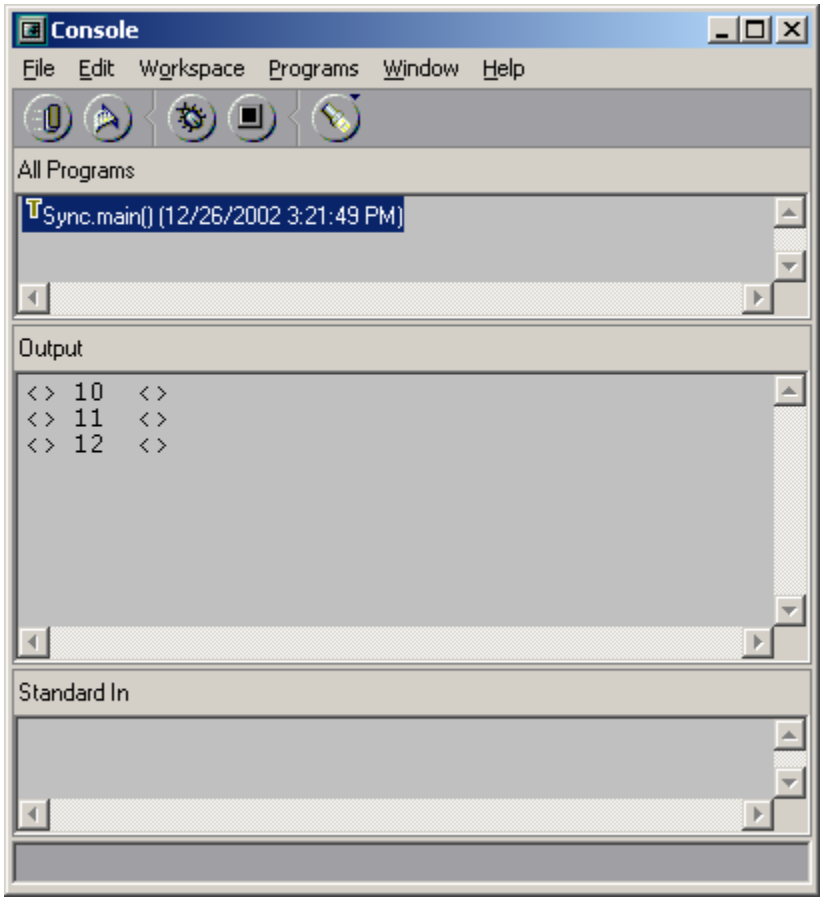
Thiết lập (khởi dựng) của liứp "Source" lấy một tham chiếu đến một đối tượng "t" của lớp "Target", và một biến số nguyên (integer). Ở đây, một luồng mới cũng được tạo ra. Luồng này gọi phương thức run() của đối tượng "t". Lớp chính "Synch" thể nghiệm lớp "Target" như là "target (mục tiêu), và tạo ra 3 đối tượng của lớp "Source" (nguồn). Cùng đối tượng "target" được truyền cho mỗi đối tượng "Source". Phương thức "join()" (gia nhập) làm luồng được gọi đợi cho đến khi việc gọi luồng bị ngắt.

Chương trình 8.4

```
class Target {

/**
 * Target constructor comment.
 */
	synchronized void display(int num) {
		System.out.print("<> "+num);
		try{
			Thread.sleep(1000);
		}catch(InterruptedException e){
			System.out.println("Interrupted");
		}
		System.out.println("  <>");
	}
}

class Source implements Runnable{
	int number;
	Target target;
	Thread t;
/**
 * Source constructor comment.
 */
	public Source(Target targ,int n){
		target = targ;
		number = n;
		t = new Thread(this);
		t.start();
	}
	public void run(){
		synchronized(target) {
			target.display(number);
		}
	}
}

class Sync {
/**
 * Sync constructor comment.
 */
	public static void main(String args[]){
		Target target = new Target();
		int digit = 10;
		Source s1 = new Source(target,digit++);
```

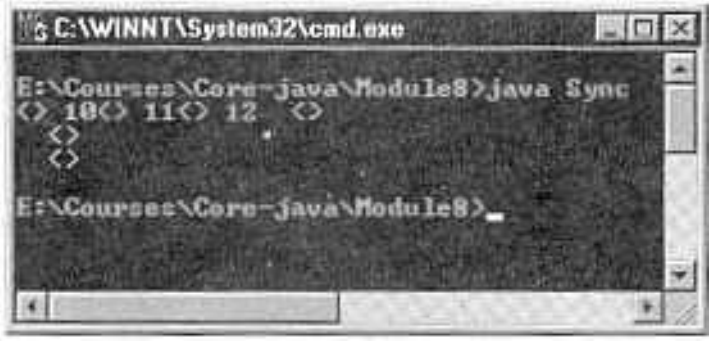
```
                Source s2 = new Source(target,digit++);
                Source s3 = new Source(target,digit++);
                try{
                        s1.t.join();
                        s2.t.join();
                        s3.t.join();
                }catch(InterruptedException e){
                        System.out.println("Interrupted");
                }
        }
}
```

Kết quả hiện thị như hình cho dưới đây:



Hình 8.6 Kết quả hiện thị của chương trình 8.4

Trong chương trình trên, có một "dãy số" đăng nhập được hiển thị "display()". Điều này có nghĩa là việc thâm nhập bị hạn chế một luồng tại mỗi thời điểm. Nếu từ khóa synchronized đặt trước bị bỏ quên trong phương thức "display()" của lớp "Target", tất cả luồng trên có thể cùng lúc gọi cùng phương thức, trên cùng đối tượng. Điều kiện này được biết đến như là "loại điều kiện" (race condition). Trong trường hợp này, việc xuất ra ngoài sẽ được chỉ ra như hình 8.7



Hình 8.7 Kết quả hiển thị của chương trình 8.7 không có sự đồng bộ

2. Sử dụng khối đồng bộ (Synchronized Block)

Tạo ra các phương thức synchronzed (đồng bộ) trong phạm vi các lớp là một con đường dễ dàng và có hiệu quả của việc thực hiện sự đồng bộ. Tuy nhiên, điều này không làm việc trong tất cả các trường hợp.

Hãy xem một trường hợp nơi mà lập trình viên muốn sự đồng bộ được xâm nhập vào các đối tượng của lớp mà không được thiết kế cho thâm nhập đa tuyến. Tức là, lớp không sử dụng các phương thức đồng bộ. Hơn nữa, mã nguồn là không có giá trị. Vì thế từ khoá synchronized không thể được thêm vào các phương thức thích hợp trong phạm vi lớp.

Để đồng bộ thâm nhập một đối tượng của lớp này, tất cả chúng gọi các phương thức mà lớp này định nghĩa, được đặt bên trong một khối đồng bộ. Tất cả chúng sử dụng chung một câu lệnh đồng bộ được cho như sau:

```
synchronized(object)
{
        // các câu lệnh đồng bộ
}
```

Ở đây, "object" (đối tượng) là một tham chiếu đến đối tượng được đồng bộ. Dấu ngoặc móc không cần thiết khi chỉ một câu lệnh được đồng bộ. Một khối đồng bộ bảo đảm rằng nó gọi đến một phương thức (mà là thành phần của đối tượng) xuất hiện chỉ sau khi luồng hiện hành đã được tham nhập thành công vào monitor (sự quan sát) của đối tượng.

Chương trình 8.5 chỉ ra câu lệnh đồng bộ sử dụng như thế nào:

Chương trình 8.5

```
class Target {

/**
 * Target constructor comment.
 */
        synchronized void display(int num) {
                System.out.print("<> "+num);
                try{
                        Thread.sleep(1000);
                }catch(InterruptedException e){
                        System.out.println("Interrupted");
                }
```

```
                System.out.println("  <>");
        }
}

class Source implements Runnable{
        int number;
        Target target;
        Thread t;
/**
 * Source constructor comment.
 */
        public Source(Target targ,int n){
                target = targ;
                number = n;
                t = new Thread(this);
                t.start();
        }
        // đồng bộ gọi phương thức display()
        public void run(){
                synchronized(target) {
                        target.display(number);
                }
        }
}

class Synchblock {
/**
 * Synchblock constructor comment.
 */
        public static void main(String args[]){
                Target target = new Target();
                int digit = 10;
                Source s1 = new Source(target,digit++);
                Source s2 = new Source(target,digit++);
                Source s3 = new Source(target,digit++);
                try{
                        s1.t.join();
                        s2.t.join();
                        s3.t.join();
                }catch(InterruptedException e){
                        System.out.println("Interrupted");
                }
        }
}
```

Ở đây, từ khóa synchronized không hiệu chỉnh phương thức "display()". Từ khóa này được sử dụng trong phương thức run() của lớp "Target" (mục tiêu). Kết quả xuất ra màn hình tương tự với kết quả chỉ ra ở hình số 8.6

3. Sự không thuận lợi của các phương thức đồng bộ

Người lập trình thường viết các chương trình trên các đơn thể luồng. Tất nhiên các trạng thái này chắc chắn không lợi ích cho đa tuyến. Lấy ví dụ, lụồng không tận dụng việc thực thi của trình biên dịch. Trình biên dịch Java từ Sun không chứa nhiều phương thức đồng bộ.

Các phương thức đồng bộ không thực thi tốt như là các phương thức không đồng bộ. Các phương thức này chậm hơn từ ba đến bốn lần so với các phương thức tương ứng không đồng bộ. Trong trạng thái nơi mà việc thực thi là có giới hạn, các phương thức đồng bộ bị ngăn ngừa.

11. Kỹ thuật "wait-notify" (đợi – thông báo)

Luồng chia các tác vụ thành các đơn vị riêng biệt và logic (hợp lý). Điều này thay thế các trường hợp (sự kiện) chương trình lặp. Các luồng loại trừ "polling" (kiểm soát vòng).

Một vòng lặp mà lặp lại việc một số điều kiện thường thực thi "polling" (kiểm soát vòng). Khi điều kiện nhận giá trị là True (đúng), các câu lệnh phúc đáp được thực hiện. Đây là tiến trình thường bỏ phí thời gian của CPU. Lấy ví dụ, khi một luồng sinh ra một số dữ liệu, và các luồng khác đang chi phối nó, luồng sinh ra phải đợi cho đến khi các luồng sử dụng nó hoàn thành, trước khi phát sinh ra dữ liệu.

Để tránh trường hợp kiểm soát vòng, Java bao gồm một thiết kế tốt trong tiến trình kỹ thuật truyền thông sử dụng các phương thức "wait()" (đợi), "notify()" (thông báo) và "notifyAll()" (thông báo hết). Các phương thức này được thực hiện như là các các phương thức cuối cùng trong lớp đối tượng (Object), để mà tất cả các lớp có thể thâm nhập chúng. Tất cả 3 phương thức này có thể được gọi chỉ từ trong phạm vi một phương thức đồng bộ (synchronized).

Các chức năng của các phương thức "wait()", "notify()", và "notifyAll()" là:

- Phương thức **wait()** nói cho việc gọi luồng trao cho monitor (sự giám sát), và nhập trạng thái "sleep" (chờ) cho đến khi một số luồng khác thâm nhập cùng monitor, và gọi phương thức "notify()".
- Phương thức **notify()** đánh thức, hoặc thông báo cho luồng đầu tiên mà đã gọi phương thức wait() trên cùng đối tượng.
- Phương thức **notifyAll()** đánh thức, hoặc thông báo tất cả các luồng mà đã gọi phương thức wait() trên cùng đối tượng.
- Quyền ưu tiên cao nhất luồng chạy đầu tiên.

Cú pháp của 3 phương thức này như sau:

final void wait() throws IOException

final void notify()

final void notifyAll()

Các phương thức wait() và notify() cho phép một đối tượng được chia sẻ để tạm ngừng một luồng, khi đối tượng trở thành không còn giá trị cho luồng. Chúng cũng cho phép luồng tiếp tục khi thích hợp.

Các luồng bản thân nó không bao giờ kiểm tra trạng thái của đối tượng đã chia sẻ.

Một đối tượng mà điều khiển các luồng khách (client) của chính nó theo kiểu này được biết như là một monitor (sự giám sát). Trong các thuật ngữ chặt chẽ của Java, một monitor là bất kỳ đối tượng nào mà có một số mã đồng bộ. Các monitor được sử dụng cho các phương thức wait() và notify(). Cả hai phương thức này phải được gọi trong mã đồng bộ.

Một số điểm cần nhớ trong khi sử dụng phương thức **wait():**

- Luồng đang gọi đưa vào CPU
- Luồng đang gọi đưa vào khóa
- Luồng đang gọi đi vào vùng đợi của monitor.

Các điểm chính cần nhớ về phương thức **notify()**

- Một luồng đưa ra ngoài vùng đợi của monitor, và vào trạng thái sẵn sàng.
- Luồng mà đã được thông báo phải thu trở lại khóa của monitor trước khi nó có thể bắt đầu.
- Phương thức notify() là không chính xác, như là nó không thể chỉ ra được luồng mà phải được thông báo. Trong một trạng thái đã trộn lẫn, luồng có thể thay đổi trạng thái của monitor trong một con đường mà không mang lại kết quả tốt cho luồng đã được đưa thông báo. Trong một số trường hợp này, các phương thức của monitor đưa ra 2 sự đề phòng:
    - Trạng thái của monitor sẽ được kiểm tra trong một vòng lặp "while" tốt hơn là câu lệnh if
    - Sau khi thay đổi trạng thái của monitor, phương thức notifyAll() sẽ được sử dụng, tốt hơn phương thức notify().

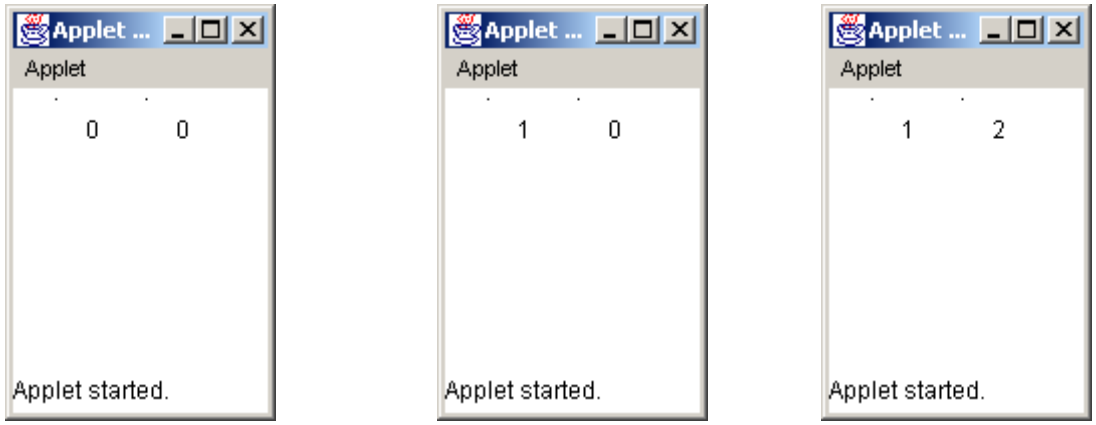
Chương trình 8.6 biểu thị cho việc sử dụng các phương thức notify(0 và wait():

Chương trình 8.6

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/*<applet code = "mouseApplet" width = "100" height = "100"> </applet> */
public class mouseApplet extends Applet implements MouseListener{
        boolean click;
        int count;
        public void init() {
                super.init();
                add(new clickArea(this)); //doi tuong ve duoc tao ra va them vao
                add(new clickArea(this));//doi tuong ve duoc tao ra va them vao
                addMouseListener(this);
        }
        public void mouseClicked(MouseEvent e) {

        }
        public void mouseEntered(MouseEvent e) {
```

```
        }
        public void mouseExited(MouseEvent e) {

        }
        public void mousePressed(MouseEvent e) {
                synchronized (this) {
                        click = true;
                        notify();
                }
                count++; //dem viec click
                Thread.currentThread().yield();
                click = false;

        }
        public void mouseReleased(MouseEvent e) {

        }
} //kết thúc Applet

class clickArea extends java.awt.Canvas implements Runnable{
        mouseApplet myapp;
        clickArea(mouseApplet mapp){
                this.myapp = mapp;
                setSize(40,40);
                new Thread(this).start();
        }
        public void paint(Graphics g){
                g.drawString(new Integer(myapp.count).toString(),15,20);

        }
        public void run(){
                while(true){
                        synchronized (myapp) {
                                while(!myapp.click){
                                        try{
                                                myapp.wait();
                                        }catch(InterruptedException ie){
                                        }
                                }
                        }
                        repaint(250);
                }

        }//end run
}
```

Không cần các phương thức wait() và notify(), luồng bức vẽ (canvas) không thể biết khi nào cập nhập hiển thị. Kết quả xuất ra ngoài của chương trình được đưa ra như sau:



Hình 8.8 Kết quả sau mỗi lần kích chuột

12. Sự bế tắt (Deadlocks)

Một "deadlock" (sự bế tắt) xảy ra khi hai luồng có một phụ thuộc vòng quanh trên một cặp đối tượng đồng bộ; lấy ví dụ, khi một luồng thâm nhập vào monitor trên đối tượng "ObjA", và một luồng khác thâm nhập vào monitor trên đối tượng "ObjB". Nếu luồng trong "ObjA" cố gắng gọi phương thức đồng bộ trên "ObjB", một bế tắt xảy ra.

Nó khó để gỡ lỗi một bế tắt bởi những nguyên nhân sau:

- Nó hiểm khi xảy ra, khi hai luồng chia nhỏ thời gian trong cùng một con đường
- Nó có thể bao hàm nhiều hơn hai luồng và hai đối tượng đồng bộ

Nếu một chương trình đa tuyến khóa kín thường xuyên, ngay lập tức kiểm tra lại điều kiện bế tắt.

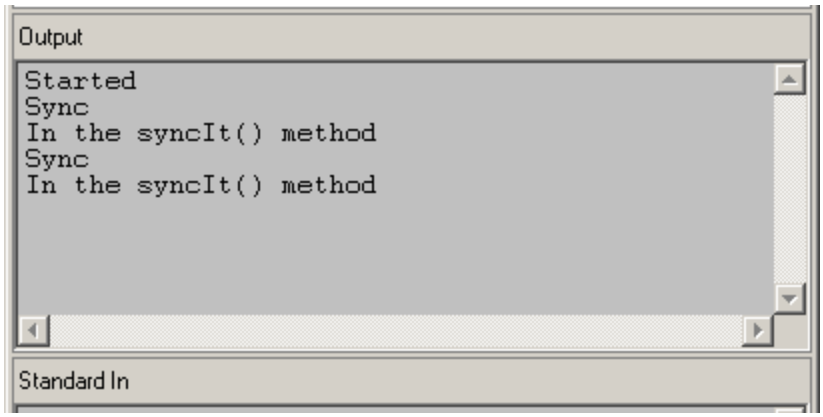
Chương trình 8.7 tạo ra điều kiện bế tắt. Lớp chính (main) bắt đầu 2 luồng. Mỗi luồng gọi phương thức đồng bộ run(). Khi luồng "t1" đánh thức, nó gọi phương thức "synchIt()" của đối tượng deadlock "dlk1". Từ đó luồng "t2" một mình giám sát cho "dlk2", luồng "t1" bắt đầu đợi monitor. Khi luồng "t2" đánh thức, nó cố gắng gọi phương thức "synchIt()" của đối tượng Deadlock "dlk2". Bây giờ, "t2" cũng phải đợi, bởi vì đây là trường hợp tương tự với luồng "t1". Từ đó, cả hai luồng đang đợi lẫn nhau, cả hai sẽ đánh thức. Đây là điều kiện bế tắt.

Chương trình 8.7

```
public class Deadlock implements Runnable{
        public static void main(String args[]){
                Deadlock dlk1= new Deadlock();
                Deadlock dlk2 = new Deadlock();
                Thread t1 = new Thread(dlk1);
                Thread t2 = new Thread(dlk2);

                dlk1.grabIt = dlk1;
                dlk2.grabIt = dlk2;
```

```
		t1.start();
		t2.start();
		System.out.println("Started");
		try{
			t1.join();
			t2.join();
		}catch(InterruptedException e){
			System.out.println("error occured");
		}
		System.exit(0);
	}
	Deadlock grabIt;
	public synchronized void run() {
		try{
			Thread.sleep(1500);
		}catch(InterruptedException e){
			System.out.println("error occured");
		}
		grabIt.syncIt();
	}
	public synchronized void syncIt() {
		try{
			Thread.sleep(1500);
			System.out.println("Sync");

		}catch(InterruptedException e){
			System.out.println("error occured");
		}
		System.out.println("In the syncIt() method");
	}
}
```

Kết quả của chương trình này được hiển thị như sau:



Hình 8.9 Sự bế tắt

13. Thu dọn "rác" (Garbage collection)

Thu dọn "rác" (Garbage collection) cải tạo hoặc làm trống bộ nhớ đã định vị cho các đối tượng mà các đối tượng này không sử dụng trong thời gian dài. Trong ngôn ngữ lập trình hướng đối tượng khác như C++, lập trình viên phải làm cho bộ nhớ trống mà đã không được yêu cầu trong thời gian dài. Tình trạng không hoạt động để bộ nhớ trống có thể là kết quả trong một số vấn đề. Java tự động tiến trình thu dọn rác để cung cấp giải pháp duy nhất cho vấn đề này. Một đối tượng trở nên thích hợp cho sự dọn rác nếu không có tham chiếu đến nó, hoặc nếu nó đã đăng ký rỗng.

Sự dọn rác thực thi như là một luồng riêng biệt có quyền ưu tiên thấp. Bạn có thể viện dẫn một phương thức gc() của thể nghiệm để viện dẫn sự dọn rác. Tuy nhiên, bạn không thể dự đoán hoặc bảo đảm rằng sự dọn rác sẽ thực thi một cách trọn vẹn sau đó.

Sử dụng câu lện sau để tắt đi sự dọn rác trong ứng dụng:

Java –noasyncgc ….

Nếu chúng ta tắt đi sự dọn rác, chương trình hầu như chắc chắn rằng bị treo do bởi việc đó.

1. Phương thức finalize() (hoàn thành)

Java cung cấp một con đường để làm sạch một tiến trình trước khi điều khiển trở lại hệ điều hành. Điều này tương tự như phương thức phân hủy của C++

Phương thức finalize(), nếu hiện diện, sẽ được thực thi trên mỗi đối tượng, trước khi sự dọn rác.

Câu lệnh của phương thức finalize() như sau:

protected void finalize() throws Throwable

Tham chiếu không phải là sự dọn rác; chỉ các đối tượng mới được dọn rác

Lấy thể nghiệm:

Object a = new Object();

Object b = a;

a = null;

Ở đây, nó sẽ sai khi nói rằng "b" là một đối tượng. Nó chỉ là một đối tượng tham chiếu. Hơn nữa, trong đoạn mã trích trên mặc dù "a' được đặt là rỗng, nó không thể được dọn rác, bởi vì nó vẫn còn có một tham chiếu (b) đến nó. Vì thế "a" vẫn còn với đến được, thật vậy, nó vẫn còn có phạn vi sử dụng trong phạm vi chương trình. Ở đây, nó sẽ không được dọn rác.

Tuy nhiên, trong ví dụ cho dưới đây, giả định rằng không có tham chiếu đến "a" tồn tại, đối tượng "a" trở nên thích hợp cho garbage collection.

Object a = new Object();
…
…
…
a = null;

Một ví dụ khác:

Object m = new Object();
Object m = null;

Đối tượng được tạo ra trong sự bắt đầu có hiệu lực cho garbage collection

Object m = new Object();

M = new Object();

Bây giờ, đối tượng căn nguyên có hiệu lực cho garbage collection, và một đối tượng mới tham chiếu bởi "m" đang tồn tại.

Bạn có thể chạy phương thức garbage collection, nhưng không có banỏ đảm rằng nó sẽ xảy ra.

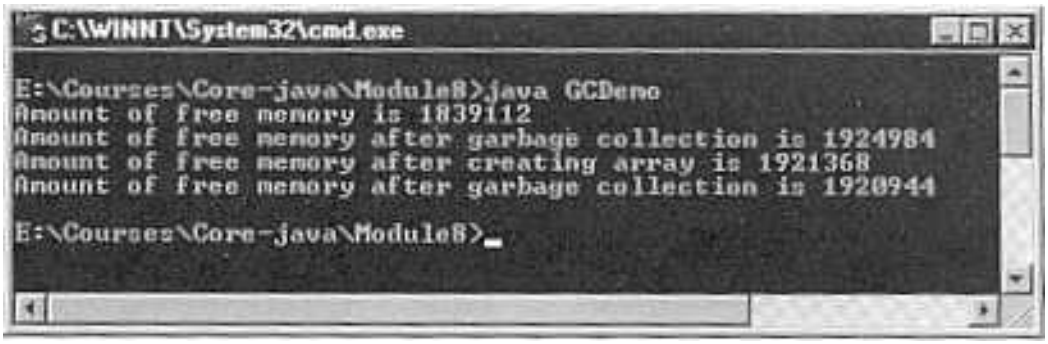
Chương trình 8.8 điển hình cho garbage collection.

Chương trình 8.8

```
class GCDemo
{
        public static void main(String args[])
        {
                int i;
                long a; ,
                Runtime r=Runtime.getRuntimeO;
                Long valuesD =new Long[200];
                System. out. print In ("Amount of free memory is" +
        r.freeMemoryO);
                r.gcO;
                System.out.println("Amount of free memory after garbage
        collection is " + r.freeMemoryO);
        for (a=IOOOO.i=O;i<200;a++.i++)
        {
                values[i] =new Long(a);
        }
        System.out.println("Amount of free memory after creating the array
" + r.freeMemoryO);
        for (i=O;i<200;i++)
        {
                values[i] =null;
        }
        System.out.println("Arnount of free memory after garbage collection is
" + r.freeMemoryO);
}
```

Chúng ta khai một mảng gồm 200 phần tử, trong đó kiểu dữ liệu là kiểu Long. Trước khi mảng được tạo ra, chúng ta phải xác định rõ số lượng bộ nhớ trống, và hiển thị nó. Rồi thì chúng ta viện dẫn phương thức gc() của thể nghiệm Runtime (thời gian thực thi) hiện thời. Điều này có thể hoặc không thể thực thi garbage collection. Rồi thì chúng ta tạo ra mảng, và đang ký giá trị cho các phần tử của mảng. Điều này sẽ giảm bớt số lượng bộ nhớ trống. Để làm các mảng phần tử thích hợp cho garbage collection, chúng ta đặt chúng rỗng. Cuối cùng, chúng ta sử dụng phương thức gc() để viện dẫn garbage collection lần nữa.

Kết quả xuất ra màn hình của chương trình trên như sau:



Hình 8.10 Garbage collection

Tổng kết

- Một luồng là đơn vị nhỏ nhất của đoạn mã thực thi được mà một tác vụ riêng biệt.
- Đa tuyến giữ cho thời gian rỗi là nhỏ nhất. Điều này cho phép bạn viết các chương trình có khả năng sử dụng tối đa CPU.
- Luồng bắt đầu thực thi sau khi phương thức start() được gọi
- Lập trình viên, máy ảo Java, hoặc hệ điều hành bảo đảm rằng CPU được chia sẻ giữa các luồng.
- Có hai loại luồng trong một chương trình Java:
  - Luồng người dùng
  - Luồng hiểm.
- Một nhóm luồng là một lớp mà nắm bắt một nhóm các luồng.
- Đồng bộ cho phép chỉ một luồng thâm nhập một tài nguyên được chia sẻ tại một thời điểm.
- Để tránh kiểm soát vòng, Java bao gồm một thiết kế tốt trong tiến trình kỹ thuật truyền thông sử dụng các phương thức "wait()" (đợi), "notify()" (thông báo) và "notifyAll()" (thông báo hết).
- Một "bế tắt" xảy ra khi hai luồng có mọt phụ thuộc xoay vòng trên một phần của các đối tượng đồng bộ
- Garbage collection là một tiến trình nhờ đó bộ nhớ được định vị để các đối tượng mà không sử dụng trong thời gian dài, có thể cải tạo hoặc làm rãnh bộ nhớ.

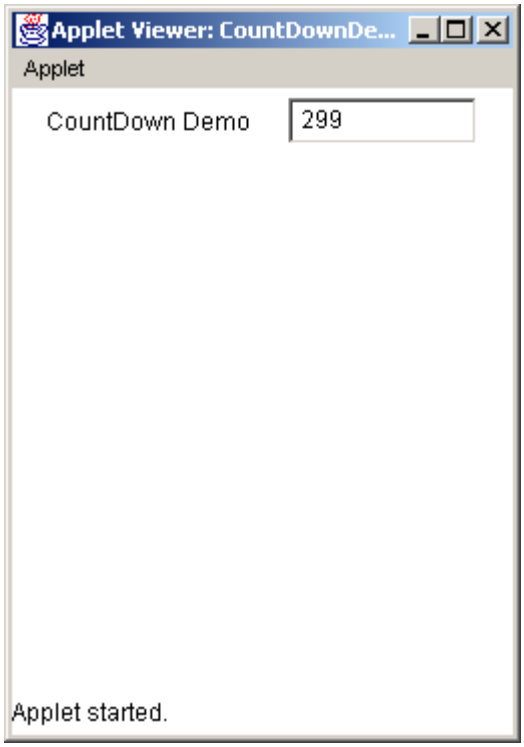
Kiểm tra lại sự hiểu biết của bạn

1. Một ứng dụng có thể chứa đựng nhiều luồng **Đúng/Sai**
2. Các luồng con được tạo ra từ luồng chính **Đúng/Sai**
3. Mỗi luồng trong một chương trình Java được đăng ký một quyền ưu tiên mà máy ảo Java có thể thay đổi. **Đúng/Sai**

4. Phương thức_____ có thể tạm thời ngừng việc thực thi luồng
5. Mặc định, một luồng có một quyền ưu tiên _____ một hằng số của _____
6. _____ luồng được dùng cho các luồng "nền", cung cấp dịch vụ cho luồng khác.
7. Trong luồng đồng bộ, một _____ là một đối tượng mà được sử dụng như là một khóa riêng biệt lẫn nhau.
8. _____ thường thực thi bởi một vòng lặp mà được sử dụng để lặp lại việc kiểm tra một số điều kiện.

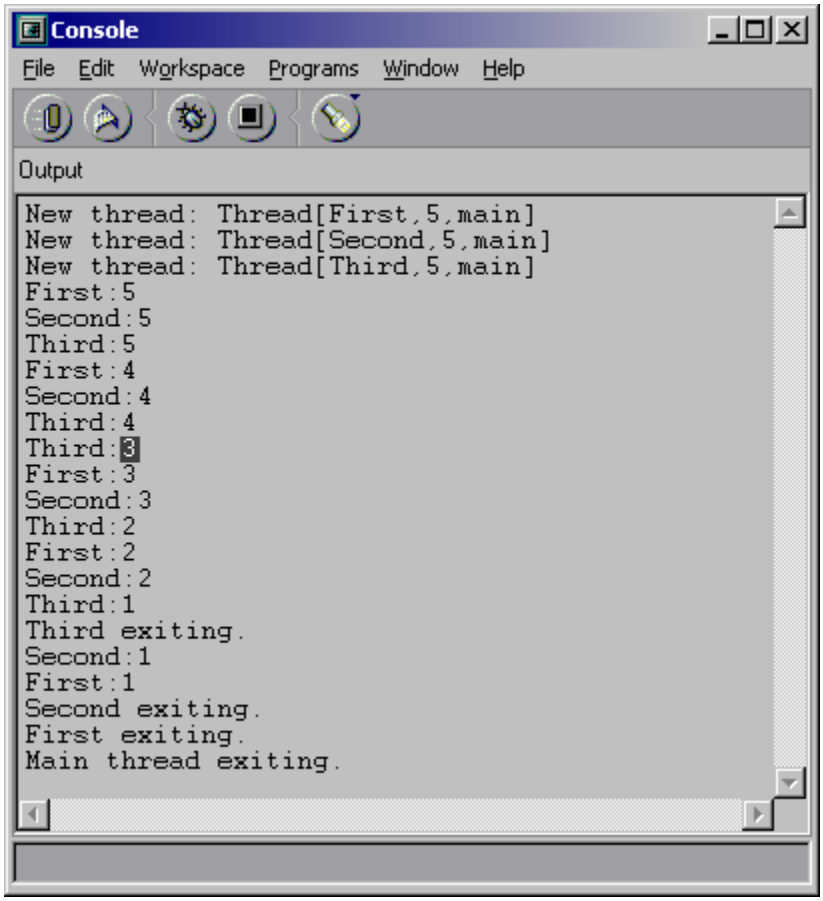
Bài tập:

1. Viết một chương trình mà hiển thị một sự đếm lùi từng giây cho đến không, như hình sau:



Ban đầu, số 300 sẽ được hiển thị. Giá trị sẽ được giảm dần cho đến 1 đến khi ngoài giá trị 0. Giá trị sẽ được trả lại 300 một lần nữa giảm đến trở thành 0.

2. Viết một chương trình mà hiển thị như hình dưới đây:



Tạo 3 luồng và một luồng chính trong "main". Thực thi mỗi luồng như một chương trình thực thi. Khi chương trình kết thúc, các câu lệnh thoát cho mỗi luồng sẽ được hiển thị. Sử dụng kỹ thuật nắm bắt lỗi.

LUỒNG I/O

Mục tiêu của môn học

Kết thúc chương, bạn có có thể :

- Đề cập đến các khái niệm về luồng
- Mô tả các lớp InputStream và OutputStream
- Mô tả I/O mảng Byte
- Thực hiện các tác vụ đệm I/O và lọc
- Dùng lớp RandomAccesFile.
- Mô tả các tác vụ chuỗi I/O và ký tự
- Dùng lớp PrinterWriter

9.1 Giới thiệu

Trong buổi học trước, chúng ta đã học về các dòng Synchronized. ngăn các dòng xẩy ra việc chia sẽ (dùng chung) các đối tượng một cách đồng thời. Toàn bộ tiến trình này được quản lý bởi cơ chế đợi thông báo (wait-notify). Phương thức wait() báo cho dòng gọi từ bỏ monitor và nhập vào trạng thái ngủ cho đến khi các dòng khác nhập vào cùng monitor và gọi phương thức notify(). Phương thức notify() và notifyAll() tạo ra dòng thông báo cho các dòng khác gọi phương thức wait() của cùng đối tượng. Trong bài học trước, chúng ta cũng học về các điều kiện bế tắc là gì và cách tránh chúng.

Chương này giới thiệu khái niệm về luồng. Chúng ta cũng thảo luận các lớp khác nhau trong gói java.io trợ giúp các tác vụ nhập xuất.

9.2 Các luồng

Theo thuật ngữ chung, luồng là một dòng lưu chuyển. trong thuật ngữ về kỹ thuật luồng là một lộ trình mà dữ liệu được truyền trong một chương trình. Một ứng dụng về các luồng ma ta đã quen thuộc đó là luồng nhập System.in .

Luồng là những dàn ống (pipelines) để gửi và nhận thông tin trong các chương trình java. Khi một luồng dữ liệu được gửi hoặc nhân, ta tham chiếu nó như đang "ghi" và "đọc" một luồng theo thứ tự nêu trên. Khi một luồng được đọc hay ghi, các dòng khác bị phong toả. Nếu có một lỗi xẩy ra khi đọc hay ghi luồng, một IOexception được kích hoạt. Do vậy, các câu lệnh luồng phải bao gồm khối try-catch.

Lớp 'java.lang.System' định nghĩa các luồng nhập và xuất chuẩn. chúng là các lớp chính của các luồng byte mà java cung cấp. Chúng ta cũng đã sử dụng các luồng xuất để xuất dữ liệu và hiển thị kết quả trên màn hình. Luồng I/O bao gồm:
:

- Lớp System.out: Luồng xuất chuẩn dùng để hiển thị kết quả trên màn hình.
- Lớp System.in: Luồng nhập chuẩn thường đến từ bàn phím và được dùng để đọc các ký tự dữ liệu.
- Lớp System.err: Đây là luồng lỗi chuẩn.

Các lớp 'InputStream' và 'OutputStream' cung cấp nhiều khả năng I/O khác nhau. Cả hai lớp này có các lớp con để thực hiện I/O thông qua các vùng đệm bộ nhớ, các tập tin và ống dẫn. Các lớp con của lớp InputStream thực hiện đầu vào, trong khi các lớp con của lớp OutputStream thực hiện kết xuất.

9.3Gói java.io

Các luồng hệ thống rất có ích. Tuy nhiên, chúng không đủ mạnh để dùng khi ứng phó với I/O thực tế. Gói java.io phải được nhập khẩu vì mục đích này. Chúng ta sẽ thảo luận tìm hiểu về các lớp thuộc gói java.io.

9.3.1 lớp InputStream

Lớp InputStream là một lớp trừu tượng. Nó định nghĩa cách nhận dữ liệu. Điểm quan trọng không nằm ở chổ dữ liệu đế từ đâu, mà là nó có thể truy cập. Lớp InputStream cung cấp một số phương pháp để đọc và dùng các luồng dữ liệu để làm đầu vào. Các phương thức này giúp ta tạo, đọc và xử lý các luồng đầu vào. Các phương thức được hiện trong bản 9.1

Tên phương thức	**Mô tả**
read()	Đọc các byte dữ liệu từ một luồng. Nếu như không dữ liệu nào là hợp lệ, nó khoá phương thức. Khi một phương thực được khoá, các dòng thực hiện được chờ cho đến khi dữ liệu hợp lệ.
read (byte [])	trả về byte được 'đọc' hay '-1', nếu như kết thúc của một luồng đã đến. nó kích hoạt IOException nếu lỗi xảy ra.
read (byte [], int, int)	Nó cũng đọc vào mảng byte. Nó trả về số byte thực sự được đọc. Khi kết thúc của một luồng đã đến. nó kích hoạt IOException nếu lỗi xảy ra.
available()	Phương pháp này trả về số lượng byte có thể được đọc mà không bị phong toả. Nó trả về số byte hợp lệ. Nó không phải là phương thức hợp lệ đáng tin cậy để thực hiện tiến trình xử lý đầu vào.
close()	Phương thức này đóng luồng. Nó dùng để phóng thích mọi tài nguyên kết hợp với luồng. Luôn luôn đóng luồng để chắc chắn rằng luồng xử lý được kết thúc. Nó kích hoạt IOException nếu lỗi xảy ra.
mark()	Đánh dấu vị trí hiện tại của luồng.
markSupporte()	trả về giá trị boolean nêu rõ luồng có hỗ trợ các khả năng mark và reset hay không. Nó trả về đúng nếu luồng hỗ trợ nó bằng không là sai.
reset()	Phương thức này định vị lại luồng theo vị

	trí được đánh dấu chót. Nó kích hoạt IOException nếu lỗi xảy ra.
skip()	Phương thức này bỏ qua 'n' byte đầu vào. '-n' chỉ định số byte được bỏ qua. Nó kích hoạt IOException nếu lỗi xảy ra. Phương thức này sử dụng để di chuyển tới vị trí đặc biệt bên trong luồng đầu vào.

Table 9.1 InputStream Class Methods

9.3.2 Lớp OutputStream

Lớp OutputStream cũng là lớp trừu tượng. Nó định nghĩa cách ghi các kết xuất đến luồng. Nó cung cấp tập các phương thức trợ giúp tạo ra, ghi và xử lý kết xuất các luồng. Các phương thức bao gồm:

Tên phương thức	**Mô tả**
write(int)	Phương thức này ghi một byte
write(byte[])	Phương thức này phong toả cho đến khi một byte được ghi. luồng chờ cho đến khi tác vụ ghi hoàn tất. Nó kích hoạt IOException nếu lỗi xảy ra.
write(byte[],int,int)	Phương thức này cũng ghi mảng các byte. Lớp OutputStream định nghĩa ba dạng quá tải của phương thức này để cho phép phương thức write() ghi một byte riêng lẻ, mảng các byte, hay một đoạn của một mảng.
flush()	Phương thức này xả sạch luồng. đệm dữ liệu được ghi ra luồng kết xuất. Nó kích hoạt IOException nếu lỗi xảy ra.
close()	Phương thức đóng luồng. Nó được dùng để giải phóng mọi tài nguyên kết hợp với luồng. Nó kích hoạt IOException nếu lỗi xảy ra.

Bảng 9.2 Các phương thức lớp OutputStream

9.3.3 Nhập và xuất mảng byte

Các lớp 'ByteArrayInputStream' và 'ByteArrayOutputStream' sử dụng các đệm bộ nhớ. Không cần thiết phải dùng chúng với nhau.

- Lớp ByteArrayInputStream

Lớp này tạo luồng đầu vào từ bộ nhớ đệm. Nó là mảng các byte. Lớp này không hỗ trợ các phương thức mới. Ngược lại nó chạy đè các phương thức của lớp InputStream như 'read() ', 'skip()', 'available()' và 'reset()'.

- Lớp ByteArrayOutputStream

Lớp này tạo ra luồng kết suất trên một mảng các byte. Nó cũng cung cấp các khả năng bổ sung để mảng kết suất tăng trưởng nhằm mục đích chừa chổ cho mảng được ghi. Lớp này cũng cung cấp các phương thức 'toByteArrray()' và 'toString()'. Chúng được dùng để chuyển đổi luồng thành một mảng byte hay đối tượng chuỗi.
Lớp ByteArrayOutputStream cũng cung cấp hai phương thức thiết lập. Một chấp nhận một đối số số nguyên dùng để ấn định mảng byte kết xuất theo một kích cỡ ban đầu. và thứ hai không chấp nhận đối số nào, và thiết lập đệm kết xuất với kích thước mặc định. lớp này cung cấp vài phương thức bổ sung, không được khai báo trong OutputStream:

- reset()

  Thiết lập lại kết xuất vùng đệm nhằm cho phép tiến trình ghi khởi động lại tại đầu vùng đệm.
- size()

  Trả về số byte hiện tại đã được ghi tới vùng đệm.
- writeto()

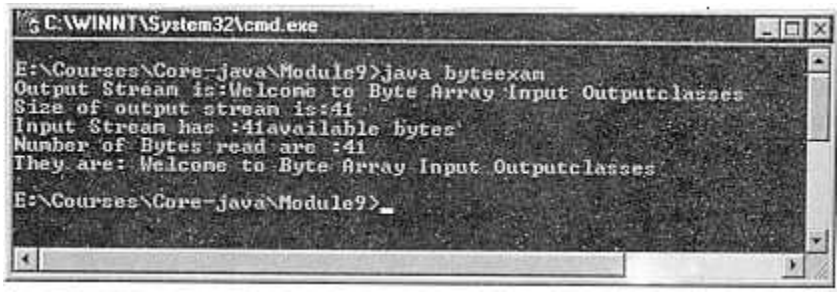
  Ghi nội dung của vùng đệm kết xuất ra luồng xuất đã chỉ định. Để thực hiện, nó chấp nhận một đối tượng của lớp OutputStream làm đối số.

Chương trình 9.1 sử dụng lớp 'ByteArrayInputStream' và 'ByteArrayOutputStream' để nhập và xuất:

Program 9.1

```
import java.lang.System;
import java.io.*;
public class byteexam
{
public static void main(String args[]) throws IOException
   {
        ByteArrayOutputStream os =new ByteArrayOutputStream();
        String s ="Welcome to Byte Array Input Outputclasses";
        for(int i=O; i<s.length( );i++)
        os. write (s.charAt(i) ) ;
        System.out.println("Output Stream is:" + os);
        System.out.println("Size of output stream is:"+ os.size());
        ByteArraylnputStream in;
        in = new ByteArraylnputStream(os.toByteArray());
        int ib = in.available();

        System.out.println("Input Stream has :" + ib + "available bytes");
        byte ibufl ] = new byte[ib];
        int byrd = in.read(ibuf, 0, ib);

        System.out.println("Number of Bytes read are :" + byrd);
        System.out.println("They are: " + new String(ibut));
   }
}
```

Hình 9.1 Xuất hiện kết xuất của chương trình:



Hình 9.1: sử dụng 1 sử dụng lớp 'ByteArrayInputStream' và 'ByteArrayOutputStream' cho nhập và xuất.

9.3.4 Nhập và xuất tập tin

Java hỗ trợ các tác vụ nhập và xuất tập tin với sự trợ giúp các lớp sau đây:

- File
- FileDescriptor
- FileInputStream
- FileOutputStream

Java cũng hỗ trợ truy cập nhập và xuất ngẫu nhiên hoặc trực tiếp bằng các lớp 'File','FileDescriptior', và 'RandomAccesFile'.

- Lớp File

  Lớp này được sử dụng để truy cập các đối tượng tập tin và thư mục. Các tập tin đặt tên theo qui ước đặt tên tập tin của hệ điều hành chủ. Các qui ước này được gói riêng bằng các hằng lớp File. Lớp này cung cấp các thiết lập các tập tin và các thư mục. Các thiết lập chấp nhận các đường dẫn tập tin tuyệt đối lẫn tương đối cùng các tập tin và thư mục. Tất cả các tác vụ thư mục và tập tin chung được thực hiện thông qua các phương thức truy cập của lớp *File*.

  Các phương thức:

  - Cho phép bạn tạo, xoá, đổi tên các file.
  - Cung cấp khả năng truy cập tên đường dẫn tập tin.
  - Xác định đối tượng có phải tập tin hay thư mục không.
  - Kiểm tra sự cho phép truy cập đọc và ghi.

  Giống như các phương thức truy cập, các phương thức thư mục cũng cho phép tạo, xoá, đặt tên lại và liệt kê các thư mục. Các phương pháp này cho phép các cây thư mục đang chéo bằng cách cung cấp khả năng truy cập các thư mục cha và thư mục anh em.

- Lớp FileDescriptor

Lớp này cung cấp khả năng truy cập các mô tả tập tin mà hệ điều hành duy trì khi các tập tin và thư mục đang được truy cập. Lớp này không cung cấp tầm nhìn đối với thông tin cụ thể do hệ điều hành duy trì. Nó cung cấp chỉ một phương thức có tên 'valid()', giúp xác định một đối tượng mô tả tập tin hiện có hợp lệ hay không.

- Lớp FileInputStream

Lớp này cho phép đọc đầu vào từ một tập tin dưới dạng một luồng. Các đối tượng của lớp này được tạo ra nhờ dùng một tập tin String, File, hoặc một đối tượng FileDescriptor làm một đối số. Lớp này chồng lên các phương thức của lớp InputStream. Nó cũng cung cấp các phương thức 'finalize()' và 'getFD()'.

Phương thức 'finalize()' được dùng để đóng luồng khi đang được bộ gôm rác Java xử lý. Phương thức 'getFD()' trả về đối tượng FileDescriptor biểu thị sự kết nối đến tập tin thực tế trong hệ tập tin đang được 'FileInputStream' sử dụng.

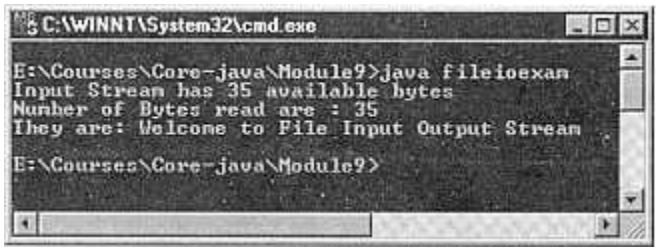
- Lớp FileOutputStream

Lớp này cho phép ghi kết xuất ra một luồng tập tin. Các đối tượng của lớp này cũng tạo ra sử dụng các đối tượng chuỗi tên tập tin, tập tin, FileDesciptor làm tham số. Lớp này chồng lên phương thức của lớp OutputStream và cung cấp phương thức 'finalize()' và getFD().

Chương trình 9.2

```
import java..io.FileOutputStream;
import java.io.FileInputStream;
import java.io.File;
import java.io.IOException;
public class fileioexam
{
    public static void main(String args[ ]) throws IOException
      {
          // creating an output file abc.txt
          FileOutputStream os = new FileOutputStream("abc.txt");
          String s = "Welcome to File Input Output Stream " ;
          for(int i = 0; i< s.length( ); + +i) .
          os. write(s.charAt(i));
          os.close();
          II opening abc.txt for input
          FileInputStream is = new FileInputStream("abc.txt");
          int ibyts = is.available( );
          System.out.println("Input Stream has " + ibyts + " available bytes");
          byte ibuf[ ] = new byte[ibyts];
          int byrd = is.read(ibuf, 0, ibyts);
          System.out.println("Number of Bytes read are: " + byrd);
          System.out.println("They are: " + new String(ibuf));
          is.close();
          File fl = new File("abc.txt");
           fl.delete();
      }
}
```

Hình 9.2 hiện kết xuất của đoạn mã nguồn trên:



Hình 9.2 sử dụng FileInputStream, FileOutputStream, và các lớp File

9.3.5 Nhập xuất đã lọc

Một 'Filter' là một kiểu luồng sửa đổi cách điều quản một luồng hiện tồn tại. Các lớp, các luồng nhập xuất đã lọc của java sẽ giúp ta lọc I/O theo một số cách. Về cơ bản, các bộ lọc này dùng để thích ứng các luồng theo các nhu cầu của chương trình cụ thể.
Bộ lọc nằm giữa một luồng nhập và một luồng xuất. Nó thực hiện xử lý một tiến trình đặc biệt trên các byte được truyền từ đầu vào đến kết xuất. Các bộ lọc có thể phối hợp thực hiện dãy tuần tự các tuỳ chọn lọc ở đó mọi bộ lọc tác động như kết xuất của một bộ lọc khác.

- Lớp FilterInputStream

  Đây là lớp trừu tượng. Nó là cha của tất cả các lớp luồng nhập đã lọc. Lớp này cung cấp khả năng tạo ra một luồng từ luồng khác. Một luồng có thể được đọc và cung cấp dưới dạng kết xuất cho luồng khác. Biến 'in' được sử dụng để làm điều này. Biến này được dùng để duy trì một đối tượng tách biệt của lớp InputStream. Lớp FilterInputStream được thiết kế sao cho có thể tạo nhiều bộ lọc kết xích [chained filters]. Để thực hiện điều này chúng ta dùng vài tầng lồng ghép. đến lượt mỗi lớp sẽ truy cập kết xuất của lớp trước đó với sự trợ giúp của biến 'in'.

- Lớp FilterOutputStream

  Lớp này là một dạng bổ trợ cho lớp FilterInputStream. Nó là lớp cha của tất cả các lớp luồng xuất đã lọc. Lớp này tương tự như lớp FilterInputStream ở chỗ nó duy trì đối tượng của lớp OutputStream làm một biến 'out'. Dữ liệu ghi vào lớp này có thể sửa đổi theo nhu cầu để thực hiện tác vụ lọc và sau đó được chuyển gửi tới đối tượng OutputStream.

9.3.6 I/O có lập vùng đệm

Vùng đệm là kho lưu trữ dữ liệu. Chúng ta có thể lấy dữ liệu từ vùng đệm thay vì quay trở lại nguồn ban đầu của dữ liệu.
Java sử dụng cơ chế nhập/xuất có lập vùng đệm để tạm thời lập cache dữ liệu được đọc hoặc ghi vào/ra một luồng. Nó giúp các chương trình đọc/ghi các lượng dữ liệu nhỏ mà không tác động ngược lên khả năng thực hiện của hệ thống.

Trong khi thực hiện nhập có lập vùng đệm, số lượng byte lớn được đọc tại thời điểm này, và lưu trữ trong một vùng đệm nhập. khi chương trình đọc luồng nhập, các byte dữ liệu được đọc từ vùng đệm nhập.

Tiến trình lập vùng đệm kết xuất cũng thực hiện tương tự. khi dữ liệu được một chương trình ghi ra một luồng, dữ liệu kết xuất được lưu trữ trong một vùng đệm xuất. Dữ liệu được lưu trữ đến khi vùng đệm trở nên đầy hoặc các luồng kết xuất được xả trống. Cuối cùng kết xuất có lập vùng đệm được chuyển gửi đến đích của luồng xuất.

Các bộ lọc hoạt động trên vùng đệm. Vùng đệm được phân bố nằm giữa chương trình và đích của luồng có lập vùng đệm.

- Lớp BufferedInputStream

Lớp này tự động tạo ra và chứa đựng vùng đệm để hỗ trợ vùng đệm nhập. Nhờ đó chương trình có thể đọc dữ liệu từng luồng theo byte một mà không ảnh hưởng đến khả năng thực hiện của hệ thống. Bởi lớp 'BufferedInputStream' là một bộ lọc, nên có thể áp dụng nó cho một số đối tượng nhất định của lớp InputStream và cũng có thể phối hợp với các tập tin đầu vào khác.
Lớp này sử dụng vài biến để thực hiện các cơ chế lập vùng đệm đầu vào. Các biến này được khai báo là protected và do đó chương trình không thể truy cập trực tiếp. Lớp này định nghĩa hai phương thức thiết lập. Một cho phép chỉ định kích cỡ của vùng đệm nhập trong khi đó phương thức thiết lập kia thì không. Nhưng cả hai phương thức thiết lập đều tiếp nhận đối tượng của lớp InputStream và OutputStream làm đối số. lớp này chồng lên các phương thức truy cập mà InputStream cung cấp và không làm nảy sinh bất kì phương thức mới nào.
Lớp BufferedInputStream. Lớp này cũng định nghĩa hai phương thức thiết lập. nó cho phép chỉ định kích cỡ của vùng đệm xuất trong một phương thức thiết lập cũng như cung cấp một kích cỡ vùng đệm ngầm định. Nó chồng lên tất cả các phương thức của OutputStream và không làm nẩy sinh bất kì phương thức nào.

Chương trình 9.3 dưới đây mô tả cách dùng các luồng nhập/xuất có lập vùng đệm:

Chương trình 9.3
```
import javaJang. * ;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.SequenceInputStream;
import java.io.IOException;
publicI class buff exam
{
   public static void main(String args[ ]) throws IOException
  {
        // defining sequence input stream
        SequenceInputStream Seq3;
        FileInputStream Fis 1 ;
        Fisl = new FileInputStream("byteexam.java");
        FileInputStream Fis2;
        Fis2= new FileInputStream("fileioexam.java");
        Seq3 = new SequenceInputStream(Fisl, Fis2);
```

```
        // create buffered input and output streams
        BufferedInputStream inst;
        inst .= new BufferedInputStream(Seq3);
        BufferedOutputStream oust;
        oust= new BufferedOutputStream(System.out);
        inst.skip(lOOO);
        boolean eof = false;
         int bytcnt = 0;
         while(!eof)
        {
            int num = inst.read();
        if(num = = -1)
        {
          eof =true;
        }
        else
        {
           oust.write((char) num);
           + + bytcnt;
         }
}
 String bytrd ,= String.valueOf(bytcnt);
bytrd + = "bytes were read";
oust.write(bytrd.getBytes(). 0, bytrd.length());

// close all streams.
           inst.close();
           oust.close();
           Fisl.close();
           Fis2.close();
          }
}
```

Hình 9.3 hiện kết xuất của chương trình trên:

Hình 9.3 Sử dụng các lớp vùng đệm luồng nhập và xuất.

9.3.7 Lớp Reader và Writer

Đây là các lớp trừ tượng. Chúng nằm tại đỉnh của hệ phân cách lớp, hỗ trợ việc đọc và ghi các luồng ký tự unicode.java 1.1 thực tế đã giới thiệu các lớp này.

- Lớp Reader

Lớp này hỗ trợ các phương thức:

  - read( )
  - reset( )
  - skip( )
  - mark( )
  - markSupported( )
  - close( )

Lớp này cũng hỗ trợ phương thức gọi 'ready()'. Phương thức này trả về giá trị kiểu boolean nếu rõ tác vụ đọc kế tiếp có tiếp tục mà không phong toả hay không.

- Lớp Writer

Lớp này hỗ trợ các phương thức:

  - write( )
  - flush( )
  - close( )

9.3.8 Nhập/ xuất chuỗi và xâu ký tự

Các lớp 'CharArrayReader' và 'CharArrayWriter' cũng tương tự như các lớp ByteArrayInputStream và ByteArrayOutputStream ở chổ chúng hỗ trợ nhập/xuất từ các vùng đệm nhớ. Các lớp CharArrayReader và CharArrayWriter hỗ trợ nhập/ xuất ký tự 8 bit.

CharArrayReader không hỗ trợ bổ sung các phương pháp sau đây vào các phương thức của lớp Reader cung cấp. Lớp CharArrayWriter bổ sung các phương thức sau đây vào các phương thức của lớp Writer.

- reset( )

thiết lập lại vùng đệm

- size( )

trả về kích cỡ hiện hành của vùng đệm

- toCharArray( )

Trả về bản sao mảng ký tự của vùng đệm xuất

- toString( )

Chuyển đổi vùng đệm xuất thành một đối tượng String

- writeTo( )

Ghi vùng đệm ra một luồng xuất khác.

Lớp StringReader trợ giúp luồng nhập ký tự từ một chuỗi. Nó không bổ sung phương thức nào vào lớp Reader.

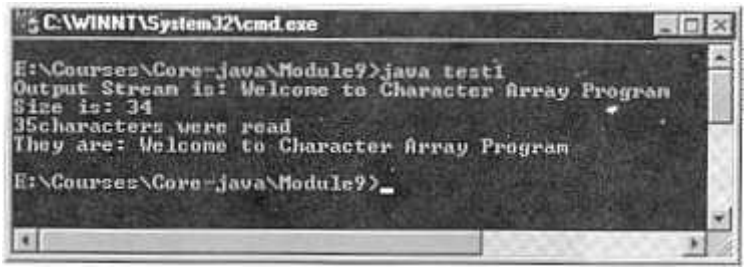
Lớp StringWriter trợ giúp ghi luồng kết xuất ký tự ra một đối tượng StringBuffer. Lớp này bổ sung hai phương thức có tên là 'getBuffer( )' và 'toString()' . Phương thức 'getBuffer( )' trả về đối tượng StringBuffer tương ứng với vùng đệm xuất, trong khi đó phương thức toString( ) trả về một bảng sao chuỗi của vùng đệm xuất.

Chương trình 9.4 dưới đây thực hiện các tác vụ nhập/xuất mảng ký tự:

Chương trình 9.4

```
import java.lang.System;
import java.io.CharArrayReader;
import java.io.CharArrayWriter;
import java.io.IOException;
public class testl
{
    public static void main(String args[ ]) throws IOException
    {
        CharArrayWriter ost = new CharArrayWriter( );
        String s = "Welcome to Character Array Program";

        for(int i= 0; i<s.length( ); ++i) ;
        osi.write(s.charAt(i));

        System.out.println("Output Stream is: " + ost);
        System.out.println("Size is: " + ost.size( ));

        CharArrayReader inst;
        inst = new CharArrayReader(ost.toCharArray( ));
        int a= 0;
        String Buffer sbI = new String Buffer(" ");

        while((a = inst.read( )) != -1)
        sbI.append((char) a);

        s = sbI.toString( );
        System.out.println(s.length() + "characters were read");
        System.out.println("They are:" + s);
    }
}
```

Hình 9.4 Hiện kết xuất chương trình:



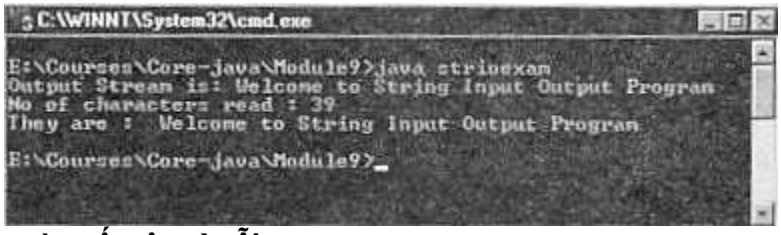
Hình 9.4 Các tác vụ nhập và xuất mảng các ký tự

Chương trình 9.5 Mô tả tiến trình nhập/xuất chuỗi.

Chương trình 9.5

```
import java.lang.System;
import java.io.StringReader;
import java.io.StringWriter;
import java.io.IOException;
import java.io. * ;

public class strioexam
{
  public static void main(String args[ ]) throws IOException
     {
        StringWriter ost = new StringWriter( );
        .String s = "Welcome to String Input Output Program";
        for(int i= 0; i <s.length( ); + +i)
        ost.write(s.charAt(i)) ;
        System.out.println("Output Stream is: " + ost);
        StringReader inst;
        inst = new StringReader(ost.toString( ));
        int a= 0;
        StringBuffer sb1 = new StringBuffer(" ");
        while((a = inst.read( )) ! = -1)
        sb1.append((char) a);
        s = sb1.toString( ); ,
        System.out.println("No of characters read: " +s.length( ));
        System.out.println("They are: " + s);
     }
}
```

Hình 9.5 Hiện kết xuất chương trình:



Hình 9.5 Nhập và xuất sâu chuỗi

9.3.9 Lớp PrinterWriter

Lớp 'PrintStream' thực hiện việc kết xuất dữ liệu. Lớp này có các phương thức bổ sung, trợ giúp cho việc in ấn dữ liệu cơ bản.

Lớp PrinterWriter' là một thay thể của lớp PrinterStream. Nó thực tế cải thiện lớp PrinterStream bằng cách dùng dấu tách dòng phụ thuộc nền tảng để in các dòng thay vì

ký tự '\n'. Lớp này cũng cấp hỗ trợ các ký tự Unicode so với PrinterStream. Phương thức 'checkError( )' được sử dụng kiểm tra kết xuất được xả sạch và và được kiểm ra các lỗi. Phương thức setError( ) được sử dụng để thiết lập lỗi điều kiện. Lớp PrintWriter cung cấp việc hỗ trợ in ấn các kiểu dữ liệu nguyên thuỷ, các mảng ký tự, các sâu chuỗi và các đối tượng.

9.3.10 Giao diện DataInput

Giao diện DataInput được sử dụng để đọc các byte từ luồng nhị phân và xây dựng lại các kiểu dữ liệu dạng nguyên thuỷ trong Java.
DataInput cũng cho phép chúng ta chuyển đổi dữ liệu từ định dạng sửa đổi UTF-8 tới dạng sâu chuỗi. Chuẩn UTF cho định dạng chuyển đổi Unicode. Nó là kiểu định dạng đặt biệt giải mã các giá trị Unicode 16 bit . UTF lạc quan ở mức thấp giả lập trong hầu hết các trường hợp, mức cao 8 bít Unicode sẽ là 0. Giao diện DataInput được định nghĩa là số các phương thức, các phương thức bao gồm việc đọc các kiểu dữ liệu nguyên thuỷ trong java.

Bảng 9.3 tóm lượt vài phương thức. Tất cả các phương thức được kính hoạt IOException trong trường hợp lỗi:

Tên phương thức	Mô tả
boolean readBoolean( )	Đọc một byte nhập, và trả về đúng nếu byte đó không phải là 0, và sai nếu byte đó là 0.
byte readByte( )	Đọc một byte
char readChar( )	Đọc và trả về một giá trị ký tự
short redShort( )	Đọc 2 byte và trả về giá trị short
long readLong( )	Đọc 8 byte và trả về giá trị long.
float readFloat( )	đọc 4 byte và trả về giá trị float
int readInt( )	Đọc 4 byte và trả về giá trị int
double readDouble( )	Đọc 8 byte và trả về giá trị double
String readUTF( )	Đọc một sâu chuỗi
String readLine( )	Đọc một dòng văn bản

Bảng 9.3 Các phương thức của giao diện DataInput

9.3.11 Giao diện DataOutput

Giao diện DataOutput được sử dụng để xây dựng lại các kiểu dữ liệu nguyên thuỷ trong java vào trong dãy các byte. nó ghi các byte này lên trên luồng nhị phân.
Giao diện DataOutput cũng cho phép chúng ta chuyển đổi một sâu chuỗi vào trong java được sửa đổi theo định dạng UTF-8 và ghi nó vào luồng.

Giao diện DataOutput định nghĩa số phương thức được tóm tắt trong bảng 9.4. Tất cả các phương thức sẽ kích hoạt IOException trong trường hợp lỗi.

Tên phương thức	Mô tả
void writeBoolean(Boolean b)	Ghi một giá trị Boolean vào luồng

void writeByte(int value)	Ghi giá trị 8 bit thấp
void writeChar(int value)	Ghi 2 byte giá trị kiểu ký tự vào luồng
void writeShort(int value)	Ghi 2 byte, biểu diễn lại giá trị dạng short
void writeLong(long value)	Ghi 8 byte, biểu diễn lại giá trị dạng long
void writeFloat(float value)	Ghi 4 byte, biểu diễn lại giá trị dạng float
void writeInt(int value)	ghi 4 byte
void writeDouble(double value)	Ghi 8 byte, biểu diễn lại giá trị dạng double
void writeUTF(String value)	Ghi một sâu dạng UTF tới luồng.

Bảng 9.4 Các phương thức của giao diện DataOutput

9.3.12 Lớp RandomAccessFile

Lớp RandomAccessFile cung cấp khả năng thực hiện I/O theo một vị trí cụ thể bên trong một tập tin. Trong lớp này, dữ liệu có thể đọc hoặc ghi ở vị trí ngẫu nhiên bên trong một tập tin thay vì một kho lưu trữ thông tin liên tục. hơn thế nữa lớp này có tên RandomAccess. Phương thưc 'seek( )' hỗ trợ truy cập ngẫu nhiên. Kết quả là, biến trỏ tương ứng với tạp tin hiện hành có thể ấn định theo vị trí bất kỳ trong tập tin.

Lớp RandomAccessFile thực hiện cả hai việc nhập và xuất. Do vây, có thể thực hiện I/O bằng các kiểu dữ liệu nguyên thuỷ. Lớp này cũng hỗ trợ cho phép đọc hoặc ghi tập tin cơ bản, điều này cho phép đọc tập tin theo chế độ chỉ đọc hoặc đọc-ghi. tham số 'r' hoặc 'rw' được gán cho lớp RandomAccessFile chỉ định truy cập 'chỉ đọc' và 'đọc-ghi'. Lớp này giới thiệu vài phương thức mới khác với phương pháp đã thừa kế từ các lớp DataInput và DataOutput.
Các phương thức bao gồm:

- **seek( )**
  Thiết lập con trỏ tập tin tới vị trí cụ thể bên trong tập tin.
- **getFilePointer( )**
  Trả về vị trí hiện hành của con trỏ tập tin.
- **length( )**
  Trả về chiều dài của tập tin tính theo byte.

Chương trình dưới đây minh hoạ cách dùng lớp RandomAccessFile. Nó ghi một giá trị boolean, một int, một char, một double tới một file có tên 'abc.txt'. Nó sử dụng phương pháp seek( ) để tìm vị trí định vị 1 bên trong tập tin. Sau đó nó đọc giá trị số nguyên, ký tự và double từ tập tin và hiển thị chúng ra màn hình.

Chương trình 9.6

```
import java.lang.System;
import java.io.RandomAccessFile;
import java.io.IOException;
public class mdexam
{
    public static void main (String args[ ]) throws IOException
     {
```

```
        RandomAccessFile rf;
        rf= new RandomAccessFile("abc.txt", "rw");
        rf. writeBoolean(true);
        rf. writelnt( 67868) ;
        rf.writeChars("J");
        rf. writeDouble(678.68);
        / / making use of seek( ) method to move to a specific file location
        rf.seek(l);
        System.out.println(rf.readlnt( ));
        System.out.println(rf.readChar( ));
        System.out.println(rf.readDouble( ));
        rf.seek(0);
        System.out.println(rf.readBoolean( ));
        rf.close( );
    }
}
```

Hình 9.5 Hiện kết xuất chương trình:

Hình 9.6: Lớp RandomAccessFile

9.4Gói java.awt.print

Đây là gói mới mà java JDK 1.2 cung cấp. Nó thay thế khả năng in của JDK 1.1. Nó bao gồm dãy các giao diện:

- **Pageable**
- **Printable**
- **PrinterGraphics**

Giao diện 'Pageable' định nghĩa các phương thức được sử dụng cho đối tượng mô tả lại các trang sẽ được in. Nó cũng chỉ định số lượng trang sẽ được in cũng như sẽ được in trang hiện hành hay một miền trang.

Giao diện 'Printable' chỉ định phương thức print( )được dùng để in một trạng trên một đối tượng Graphics.

Giao diện 'PrinterGraphics' cung cấp khả năng truy cập đối tượng 'PrinterJob'. Nó cung cấp các lớp sau đây:

- **Paper**
- **Book**
- **PageFormat**

- **Printerjob**

Lớp ‘Page’ định nghĩa các đặc tính vật lý của giấy in. Ngoài ra nó cũng cung cấp khổ giấy và vùng vẽ.

Lớp ‘Book’ là một lớp con của đối tượng duy trì một danh sách các trang in. Lớp này cũng cung cấp các phương thức để bổ sung và quản lý các trang cũng như thực thi giao diện Pageable.

Lớp ‘PageFormat’ định nghĩa lề của trang như các lề ‘Top’, ‘Bottom’,’Left’ và ‘Right’. Nó cũng chỉ định kích cỡ và hướng in như ‘Portait’ (khổ dọc) hoặc ‘Landscape’ (khổ ngang).

Lớp ‘Printerjob’ là một lớp con của đối lượng khởi tạo, quản lý, và điều khiển yêu cầu máy in. Lớp này cũng chỉ định các tính chất in.

Dưới đây là ngoại lệ và lỗi mà gói java.awt.print kích hoạt:

- **PrinterException**
- **PrinterIOException**
- **PrinterAbortException**

‘PrinterException‘ mở rộng lớp java.lang.Exception nhằm cung cấp một lớp cơ sở để in các ngoại lệ liên quan.

‘PrinterIOException’ mở rộng lớp ‘PrinterException’ nêu rõ một lỗi trong I/O.

‘PrinterAbortException’ là lớp con của lớp PrinterException nêu rõ khối in đã được bỏ ngang.

Tóm tắt buổi học

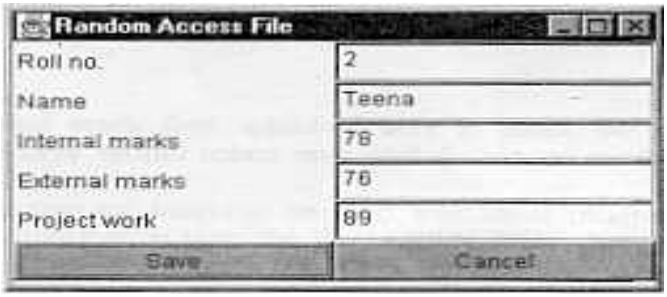
- Một luồng là một lộ trình qua đó dữ liệu di chuyển trong một chương trình java.
- Khi một luồng dữ liệu được gửi hoặc nhận.Chung ta xem nó như đang ghi và đọc một luồng theo thứ tự nêu trên.
- Luồng nhập/xuất bao gồm các lớp sau đây:
    - Lớp System.out
    - Lớp System.in
    - Lớp System.err
- Lớp InputStream là một lớp trừu tượng định nghĩa cách nhận dữ liệu.
- Lớp OutputStream cũng là lớp trừu tượng. Nó định nghĩa ghi ra các luồng được kết xuất như thế nào.
- Lớp ByteArrayInputStream tạo ra một luồng nhập từ vùng đệm bộ nhớ trong khi ByteArrayOutputStream tạo một luồng xuất trên một mãng byte.
- Java hổ trợ tác vụ nhập/xuất tập tin với sự trợ giúp của các File, FileDescriptor, FileInputStream và FileOutputStream.
- Các lớp Reader và Writer là lớp trừu tượng hỗ trợ đọc và ghi các luồng ký tự Unicode.
- CharArrayReader, CharArrayWriter khác với ByteArrayInputStream, ByteArrayOutputStream hỗ trợ định dạng nhập/xuất 8 bit, Trong khi ByteArrayInputStream, ByteArrayOutputStream hỗ trợ nhập/xuất 16bit.
- Lớp PrintStream thực thi một kết xuất. lớp này có phương thức bổ sung, giúp ta in các kiểu dữ liệu cơ bản.
- Lớp RandomAccessFile cung cấp khả năng thực hiện I/O tới vị trí cụ thể trong một tập tin.

Kiểm tra mức độ tiến bộ

1. ---------- là các dàn ống (pipelines) để gửi và nhận thông tin trong các chương trình java.
2. ---------- là luồng lỗi chuẩn.
3. Phương thức ---------- đọc các byte dữ liệu từ một luồng.
4. Phương thức ---------- trả về giá trị boolean, nêu rõ luồng có hỗ trợ các khả năng mark và reset hay không.
5. Phương thức ---------- xả sạch luồng.
6. Nhập/xuất mảng byte sử dụng các lớp ---------- và ----------
7. Lớp ---------- được sử dụng truy cập các đối tượng thư mục và tập tin.
8. ----------là một khi chưa để lưu giữ dữ liệu.

Bài tập

1. Viết chương trình nhận một dòng văn bản từ người dùng và hiển thị đoạn văn bản đó lên màn hình.
2. Viết chương trình sao chép nội dụng một tập tin tới tập tin khác.
3. Viết chương trình tạo ra một tập tin truy cập ngẫu nhiên. kết xuất hiển thị phía dưới đây.



Các bản ghi nên được lưu ở dạng tập tin '.dat', vì vậy người dùng truy cập chúng nhanh hơn.

THỰC THI BẢO MẬT

Mục tiêu bài học:

Cuối chương này bạn có thể

- Mô tả về công cụ JAR
- Tạo và xem một file JAR, và liệt kê và trích rút nội dung của file.
- Sử dụng chữ ký điện tử (Digital Signatures) để nhận dạng Applets
- Tạo bộ công cụ khóa bảo mật (Security key)
- Làm việc với chứng chỉ số (Digital Certificate)
- Tìm hiểu về gói Java.security

10.1 Giới thiệu:

Trong phần này, chúng ta sẽ tìm hiểu chi tiết về bảo mật Java applet. Chúng ta cũng thảo luận về mô hình bảo mật JDK 1.2 đáp ứng nhu cầu người dùng và nhà phát triển.

Java là một ngôn ngữ lập trình đầu tiên gởi các chương trình không tương tác như các file văn bản, file ảnh và các thông tin tĩnh thông qua World Wide Web. Các chương trình này, không giống như chương trình CGI, được chạy trên hệ thống của người dùng, hơn là chạy trên máy chủ Web (Web server). Bảo mật Java Applet là sự quan tâm chính giữa người dùng và nhà phát triển applet. Thiết tính bảo mật trong applet có thể dẫn tới sửa đổi hoặc phơi bày các dữ liệu nhạy cảm. Mô hình bảo mật của Java 2, hoặc JDK 1.2 rất hữu ích cho người dùng, cũng như cho nhà phát triển. Nó giúp người dùng duy trì mức độ bảo mật cao. Trong chương nay, chúng ta sẽ học mô hình bảo mật JDK 1.2.

10.2 Công cụ JAR:

Một file JAR là một file lưu trữ được nén do công cụ lưu trữ Java tạo ra. File này tương tự như chương trình PKZIP. Nó chứa nhiều file trong một file lưu trữ. Điều này cho phép tải trong trình duyệt hiệu quả. Dùng một jar với một applet cải tiện đáng kể khả năng thực hiện của trình duyệt. Vì tấc cả các tất cả các file được biên dịch trong một file đơn, trình duyệt chỉ cần thiết lập kết nối HTTP với web server. Nén file giảm 50% thời gian tải file.

Để khởi động công cụ JAR, dùng câu lệnh sau tại dấu nhắc lệnh:

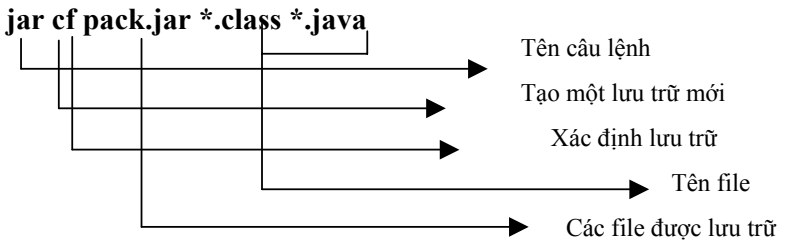
jar [options][manifest] jar-file input-file(s)

Tuỳ chọn	Mô tả
c	Tạo ra một lưu trữ mới
t	Ghi vào bảng nội dung cho lưu trữ
x	Trích dẫn file có tên từ lưu trữ
v	Tạo nguồn xuất đa dòng (verbose output) trên một lỗi chuẩn
f	Xác định tên file lưu trữ
m	Bao hàm thông tin chứng thực từ các file chứng thực xác định
o	Lưu trữ chỉ 'use no zip' nén
M	Không tạo các file chứng thực cho các mục (entries).

Bảng 0.1. công cụ jar

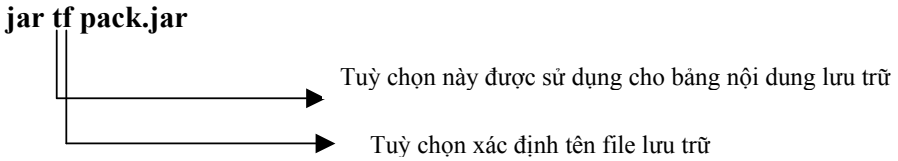
Một file chứng thực chứa thông về các file lưu trữ. File này là một tuỳ chọn. Thậm chí file không xác định thì JAR cũng tự động tạo ra. File jar được dùng như các lưu trữ. File này phải có phần mở rộng là '.jar' được xác định tại dòng lệnh. File đầu vào (input-file) là danh sách phân cách các file được đặt trong lưu trữ. Netscape Navigator và Internet Explorer hỗ trợ file JAR.

Câu lệnh sau lưu trữ tất cả các file class và file java bao gồm trong một thư mục xác định vào một file jar gọi là 'pack'



Hình 10.1 lệnh jar

Dùng lệnh sau tại dấu nhắc liệt kê các file trong file 'pack.jar'



Hình 10.2 Liệt kê các file trong file pack.jar

Để gộp file lưu trữ 'pack.jar vào trong một applet, mở trang HTML, và thêm thuộc tính ARCHIVE='pack.jar' vào thẻ applet, như sau:

<applet code="exr7.class" ARRCHIVE="pack.jar" height=125 width=350></applet>

Thuộc tính sẽ chỉ cho trình duyệt nạp lưu trữ 'pack.jar' để tìm file 'exr7.class'

Câu lệnh sau trích rút các file được nén trong file pack.jar:

jar xvf pack.jar

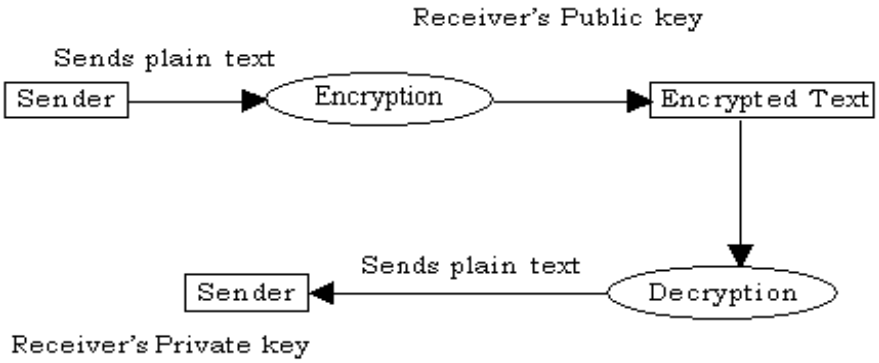
Mục chọn 'x' cho phép bạn trích rút nội dung của file.

10.3 Chữ ký điện tử (Digital Signature) cho định danh các applet:

Trong java, bảo mật applet trên web là phần rất quan trọng. Hacker có thể viết các applet nguy hiểm xuyên thủng hàng rào bảo mật. Vì thế, applet hạn chế sự can thiệt của các ngôn ngữ. Applet không hỗ trợ một số nét đặt trưng sau:

- Đọc và ghi file từ hệ thống nơi applet đang chạy.
- Lấy thông tin về một file từ hệ thống
- Xoá một file từ hệ thống.

Java 2 có thể thực hiện tất cả các đặc điểm trên, với các applet cung cấp từ một nhà cung cấp applet tin cậy, và được ký danh số (digitally signed).
Hình sau minh họa quá trình mã hoá khoá



Hình 10.3. Mã hoá dựa trên các khoá

Trong hình trên, khoá công cộng (public keys) được dùng mã hoá và giải mã. Cùng ý tưởng được sử dụng cho chữ ký số, thêm các tính năng bổ sung.
Một chữ ký số là một file mã hoá cung cấp chương trình nhận dạng chính xác nguồn gốc của file. Khóa bí mật tính giá trị từ file applet. Người giữ khoá bí mật kiểm tra nội dung của đối tượng.
Trong định danh số, một khóa riêng (private key) được sử dụng để mã hóa, và khoá công cộng, được dùng giải mã. Trong khi ký danh (sign) một đối tượng, người ký danh dùng thuật toán tóm lược thông báo như MD5 để tính bảng tóm lược của đối tượng. Bảng tóm lược được dùng như là dấu tay cho đối tượng. bảng tóm lược lần lượt được mã hoá dùng khóa riêng, đưa ra chử ký điện tử của đối tượng. Khoá công cộng của bộ ký duyệt dùng để mã hoá chữ ký và kiểm tra chúng. Kết quả của sự giải mã, giá trị tóm lược được đưa ra. Giá trị tóm lược của đối tượng được tính và so sánh với giá trị tóm lược được giải mã. Nếu giá trị tóm lược (digest) của đối tượng và giá trị tóm lược được mã hoá khớp với nhau, chữ ký được được xác nhận. Tài liệu mô tả chữ ký được gọi là "Chứng thực" (Certificate)
Thiết lập sự uỷ thác (trust), nhận dạng applet được chứng nhận. Chứng nhận các thực thể các sử dụng khóa công cộng đặt biệt. Quyền chứng thực (a certificate authority) được dùng thực hiện chứng nhân. Nhận được được chứng thực từ một CA (Certificate Authority), applet phải đệ trình tài liệu chứng thực sự nhận dạng của nó.
Hiện giờ các công ty đưa ra các dịch vụ xác nhận chứng thực sau:

- VeriSign
- Chứng thực Thawte

Bạn có thể thiết lập các mức bảo mật khác nhau. Một applet có thể đưa ra sự uỷ thác hoàn toàn, hoặc không uỷ thác, với sự giúp đỡ của tập các lớp gọi là "Quyền" (Permissions). Nhưng nhìn chung, mỗi applet được giới hạn một cách đầy đủ, trừ khi nhà phát triển ký danh applet. Điều này thiết lập cho nhà phát triển đáng tin cậy.

10.4 Khoá bảo mật Java (Java Security key).

Chúng ta cần tạo 3 công cụ, tên là, 'jar', 'jarsigner', và 'keytool', trước khi dùng các applet ký danh. Chúng ta cần tạo cặp khóa công cộng/riêng, và làm cho nó trở nên sẵn sàng với công cụ jarsigner.

Bây giờ, chúng ta sẽ tạo các công dụng của keystore.

- Keystore (Lưu trữ khoá)

  Keystore là một cơ sở dữ liệu khoá, chứa các chứng thực số dùng để nhận dạng các giá trị khoá công cộng.

- Keytool (Công cự khoá)

Keytool là công cụ khoá bảo mật của java, tạo và quản lý khóa công cộng, khoá riêng, và các chứng thực bảo mật. Nó cũng có thể thực hiện:

- Quản lý cặp khoá công cộng/riêng
- Lưu trữ các khoá công cộng
- Dùng các chứng thực để xác thực chứng thực khác.
- Xác thực (Authenticate) dữ liệu nguồn.

Tất cả thông tin mà keytool quản lý được lưu trữ trong cơ sở dữ liệu gọi là keystore. Sun có một keystore mật định dùng một định dạng file mới gọi là JKS (java key store Lưu trữ khoá java). Để kiểm nếu hệ thống bạn có một keystore dưới định dạng này, thực hiện câu lệnh sau tại dấu nhắc lệnh:

Keytool –list

Thông báo lỗi sau xuất hiện nếu bạn không có gì trong keystore của bạn.

Keytool error: keystore file does not exist: c:\windows\.keystore

JDK tìm keystore chính trong thư mục C:\windows\. Đây là một vị trí chung cho các file hệ thống quan trọng trên windows 95, 98 và NT systems.

Tuỳ chọn keystore cũng có thể được sử dụng trong lệnh keytool, như sau:

keytool –list keystore c:\java\try

Câu lệnh này chỉ cho JDK tìm keystore trong file được gọi là 'try' trong thư mục 'C:\java\try'. Nếu không tìm thấy, sẽ hiển thị thông báo lỗi như trên.

Mục '-genkey' có thể được sử dụng cùng với câu lệnh keytool để tạo cặp khoá công cộng/riêng. Bạn cũng có thể dùng một số các tuỳ chọn khác. Dạng đơn giản nhất như sau:

keytool –genkey –alias "I"

Bí danh (alias) có thể được dùng lưu trữ, thay thế hoặc xoá cặp khoá. Các bí danh keytool không phân biệt chữ hoa. Trong lệnh trên, chúng ta không sử dụng tuỳ chọn keystore. Nếu cùng câu lệnh sử dụng tuỳ chọn keystore, sẽ được viết lại như sau:

keytool –genkey –alias "I" –keystore "store"

Trong lệnh trên, cặp khoá sẽ được lưu trữ trong keystore 'store', và không lưu trong keystore mật định của hệ thống.

Sau khi nhập lệnh trên vào, và nhấn phím enter, keytool nhắc bạn nhập vào mật khẩu (password) cho keystore, như sau:

Enter keystore password

Nhập vào 'password' như yêu cầu.
Tiếp theo, keytool nhắc bạn nhập vào các thông tin bổ sung như:

What is your first and last name? (Tên và họ)
[unknown]
what is the name of your organization unit?

[unknown]: software Development.
What is the name of your organization? (Tên của tổ chức)
[Unknown]: ABC Consultants (tư vấn ABC)
What is the name of your city or Locality? (tên thành phố hoặc địa phương của bạn)
[Unknown]: California
What is the name of your State or Province? (tên bang hoặc tỉnh của bạn)
[Unknown]:United States of America
What is the two-letter country code for this unit?(Mã quốc gia với 2 ký tự)
[Unknown]: US

Khi bạn đã nhập vào các thông tin, keytool hiển thị thông tin sau:

Is <CN=Bob Fernandes, OU=Software Development, O=ABC Consultants, L=California, ST=United States of America, C=US>correct?
[no]:

Cuối cùng, keystool nhắc bạn nhập vào mật khẩu cho khoá riêng của bạn, như:

Enter key password for <I>
(RETURN if same as keystore password)

Thông tin trên được sử dụng để kết hợp sự phân biệt tên (name) X500 với bí danh (alias). Thông tin trên cũng có thể được đưa vào trực tiếp từ mục chọn '-dname'

Mật khẩu sau cùng phần biệt với mật khẩu keystore. Nó được dùng truy cập khoá riêng của cặp khoá công cộng. Mật khẩu có thể trực tiếp chỉ rõ bằng cách sử dụng tuỳ chọn '-keypass'. Nếu mật khẩu không chỉ rõ, mật khẩu keystore được sẽ được dùng. Tuỳ chọn '-keypasswd' dùng thay đổi mật khẩu. Tuỳ chọn '-keyalg' chỉ rõ thuật toán tạo cặp khoá.

Khi bạn tạo một khoá và bổ sung nó vào trong keystore, bạn có thể dùng tuỳ chọn '-list' của keytool để xem khoá có trong keystore hay không.

Để xoá cặp khoá từ cơ sở liệu, dùng lệnh sau:

keytool –delete –alias aliasName

'aliasName' chỉ tên của khoá được xoá.

Bây giờ, chúng ta tạo cặp khoá riêng/công cộng cho file JAR, chúng ta hãy ký danh nó. Lệnh jarsigner dùng để ký danh một file JAR. Nhập lệnh sau vào dấu nhắc DOS:

jarsigner –keystore keyStore –storepass storePassword –keypass keyPassword

Bảng sau cung cấp danh sách của JARFileNames và bí danh:

Tuỳ chọn	Mô tả
keyStore	Tên keystore sử dụng
storePassword	Mật khẩu keystore
keyPassword	Mật khẩu khoá riêng
JARFileName	Tên của file JAR được ký danh
Alias	Bí danh của bộ ký danh

Bảng 10.2 JARFileNames và bí danh

Để ký danh file JAR 'pack.jar', với keystore 'store', và mật khẩu để lưu trữ và các khoá riêng là 'password', dùng lệnh sau:

jarsigner –keystore store –storepass password –keypass password pack.jar pk
'pk' nghĩa là tên bí danh.

Nếu tuỳ chọn '-keystore' không chỉ rõ, thì keystore mật định được dùng.

Để chỉ rõ chữ ký của file JAR được định danh, dùng tuỳ chọn '-verify'.

jarsigner –verify pack.jar

'pack.jar' chỉ tên file JAR. Nếu chữ ký không hợp lệ, thì ngoại lệ sau được ném ra (thrown).

Jarsigner:java.util.zip.ZipException:invalid entry size (expected 900 but got 876 bytes)

Ngược lại, xuất hiện thông báo "jar verified" (jar được xác minh)

Quá trình xác thực kiểm tra theo các bước sau:

- Có file '.DSA' chứa chữ ký hợp lệ cho file chữ ký .SF không.
- Có các mục trong file chữ ký là các tóm lược hợp lệ cho mỗi mục tương ứng file kê khai (manifest file)

10.5 Chữ ký điện tử (digital Certificates)

Cho đến bây giờ, chúng ta đã học cách tạo và ký danh một file JAR. Bây giờ, chúng ta sẽ học cách xuất các chữ ký điện tử(digital certificates), sẽ sử dụng để xác thực chữ ký của các file JAR. Chúng ta cũng sẽ học các nhập chữ ký điển tử từ các file các.

Chữ ký điện tử là một file, một đối tượng, hoặc một thông báo được ký danh bởi quyền chứng thực (certificate authority). The CA (Certificate authority) cấp chứng nhận giá trị các khoá công cộng. Chứng nhận X.509 của tổ chức International Standards Organization là một dạng chứng nhận số phổ biến. Keytool hổ trợ những chứng nhận này.

Keytool ở bước đầu tiên cần nhận được một chứng nhận (certificate). Chúng ta dùng chứng nhận đó tạo cặp khoá 'công cộng/riêng' (private/public). Keytool nhập vào các chứng nhận đã được tạo và được ký danh. Keytool tự động gắn (bundle) khoá công cộng mới với một chứng nhận mới. Cùng thực thể đã tạo khoá công cộng ký danh chứng nhận này. Đó được gọi là **'self-signed certificates'** (Chứng nhận tự ký danh).Các chứng nhận này không phải là chứng nhận đáng tin cậy cho định danh. Tuy nhiên, chúng cần để tạo các yêu cầu ký danh chứng nhận (certificate-signing request).

Keytool và tuỳ chọn được sử dụng để tạo các chứng nhận trên. Câu lệnh sau giúp tạo các chứng nhận trên:

keytool –keystore store –alias mykey –certreq –file mykey.txt

Cặp khoá được tạo là 'mykey'. Tuỳ chọn '-file' chỉ tên file, mà yêu cầu ký danh chứng nhận dùng để lưu.

Dùng lệnh '-export' xuất các chứng nhận này như sau:
keytool –export –keystore store –alias pk –file mykey

Câu lệnh trên hiển thị dấu nhắc sau:

Enter keystore password

Chứng nhận đã lưu trữ trong <mykey>

Để nhập các chứng nhận khác vào keystore của bạn, nhập câu lệnh sau:

keytool import –keytool keystore –alias alias –file filename

Tên được chỉ như là tên file chứa chứng nhận được nhập vào (imported certificate).

Câu lệnh sau chỉ tên bí danh là 'alice' để nhập chứng nhận trong file 'mykey' vào keystore 'MyStore':

keytool –import –keystore MyStore –alias alice –file mykey

Câu lệnh trên hiển thị dấu nhắc sau:

Enter keystore password (Nhập vào mật khẩu keystore)

Kết quả xuất ra hiển thị hai tuỳ chọn –Owner và Issuer. Nó hiển thị tên công ty, nghề nghiệp, tổ chức, địa điểm, bang và tiền tệ. Nó cũng hiển thị số serial và thời gian có giá trị. Cuối cùng, nó hỏi có là chứng nhận uỷ thác không. Chứng nhận được chấp thuận cho sự uỷ thác của riêng bạn.

Dùng lệnh '-list' liệt kê nội dung của keystore như sau:

keystool –list –keystore Store

Câu lệnh trên yêu yều password keystore

Dùng tuỳ chọn '-alias' liệt kê một mục. Dùng lệnh -delete để xoá bí danh trong keystore, như sau:

keytool –delete –keystore Store –alias alias

Dùng lệnh '-printcert' in chứng nhận được lưu trữ trong file, theo cách sau:

keytool –printcert –file myfile

Dùng lệnh '-help' nhận về danh sách tất cả các lệnh keytool hổ trợ:

keytool -help

10.6 Các gói bảo mật java (JAVA Security packages)

Các gói bảo mật Java bao gồm:

- java.security

Đây là gói API nhân bảo mật (the core security API package). Chứa các lớp và giao diện (interface) hỗ trợ mã hoá (encryption), tính bảng tóm lược tài liệu và chữ ký điện tử.

- java.security.acl

Chứa các giao diện dùng cài đặt các chính sách điều kiển truy cập

- java.security.cert

Cung cấp sự hổ trợ cho chứng nhận X.509

- java.security.interfaces

Định nghĩa các giao diện truy cập thuật toán chữ ký điện tử (the digital signature algorithm)

- java.security.spec

Cung cấp các lớp độc lập và phục thuộc vào thuật toán cho các khoá.

Tóm tắt:

- Nếu khả năng bảo mật trong applet không đảm bảo, các dữ liệu nhạy cảm có thể được sữa đổi hoặc phơi bày.
- Mục đích chính của JAR là kết nối các file mà applet sử dụng trong một file nén đơn. Điều này cho phép các applet nạp vào trình duyệt một cách hiệu quả.
- Một file kê khai (manifest file) chứa thông tin về các file lưu trữ.
- Chữ ký điện tử là một mã hoá kèm với chương trình để nhận diện chính xác nơi nguồn gốc của file.
- Keystore là một cơ sở dữ liệu của các khoá.
- Keytool là công cụ khoá bảo mật của java.
- chứng nhận điện tử là một file, hoặc một đối tượng, hoặc một thông báo được ký danh bởi quyền chứng nhận (certificate authority)

Kiểm tra kiến thức:

1. File _____là file lưu trữ được nén.
2. Tuỳ chọn _____, khi dùng với công cụ jar, trích rút tên file từ một lưu trữ (file)
3. JAR tự động tạo file kê khai, thậm chí nó không được chỉ ra ***true/false***
4. Thuộc tính_____,khi dùng trong thẻ applet, chỉ cho trình duyệt nạp file jar lưu trữ cụ thể, và tìm file class được nhập vào.
5. Trong chữ ký điện tử, _____được dùng cho mã hoá và _____được dùng cho giải mã.
6. Tất cả các thông tin keytool quản lý, được lưu trữ trong một cơ sở dữ liệu gọi là _____
7. keytool ở bước đầu tiên cần nhận được một chứng nhận ***true/false***
8. Gói _____chứa giao diện (interfaces) dùng cài đặt các chính sách điều kiển truy cập.

Bài tập:

Tạo **các câu lệnh java** thực hiện các hành động sau:

1. Tạo một file jar 'core-java.jar' chứa các file lớp (class file) và các file nguồn.
2. Liệt kê nội dung của file jar.
3. Tạo file html cho file CardLayoutDemo.class,file lớp được chứa trong file jar.
4. trích rút (extract) file jar
5. Dùng lệnh keytool với tên bí danh và keystore để tạo ra cặp khoá công cộng/riêng mới
6. Ký danh file jar mới được tạo
7. Xác minh chữ ký (signature).
8. Xuất các chứng nhận (certificate)

9. Liệt kê nội dung của keystore
10. In các chứng nhận được lưu trong file.