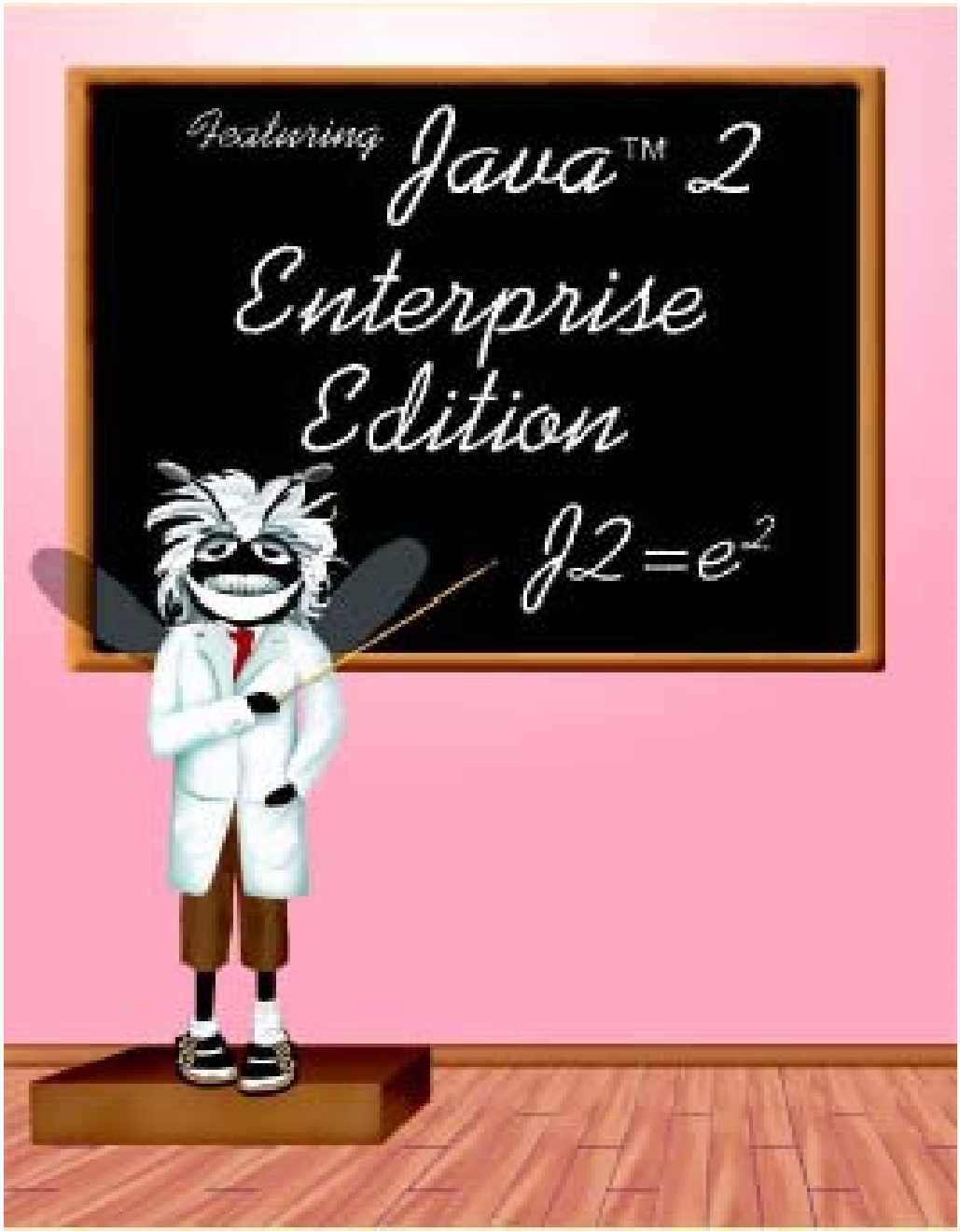
BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRỜNG ĐẠI HỌC S PHẠM KỸ THUẬT HNG YÊN

----------ooo----------



Chương 1

CÁC KHÁI NIỆM CƠ BẢN

BÀI 1. LÀM QUEN VỚI JAVA

I. Lịch sử java

Java là một ngôn ngữ lập trình được Sun Microsystems giới thiệu vào tháng 6 năm 1995. Từ đó, nó đã trở thành một công cụ lập trình của các lập trình viên chuyên nghiệp. Java được xây dựng trên nền tảng của C và C++. Do vậy nó sử dụng các cú pháp của C và các đặc trưng hướng đối tượng của C++.

Ban đầu Java được thiết kế để làm ngôn ngữ viết chương trình cho các sản phẩm điện tử dân dụng như đầu video, tivi, điện thoại, máy nhắn tin.. . Tuy nhiên với sự mãnh mẽ của Java đã khiến nó nổi tiếng đến mức vượt ra ngoài sự tưởng tượng của các nhà thiết kế ra nó.

Java khởi thuỷ tên là Oak- là cây sồi mọc ở phía sau văn phòng của nhà thiết kế chính ông Jame Gosling, sau này ông thấy rằng đã có ngôn ngữ lập trình tên Oak rồi, do vậy nhóm thiết kế quyết định đổi tên, "Java" là cái tên được chọn, Java là tên của một quán cafe mà nhóm thiết kế java hay đến đó uống.

II. Java em là ai

Java là ngôn ngữ lập trình hướng đối tượng, do vậy không thể dùng Java để viết một chương trình hướng chức năng. Java có thể giải quyết hầu hết các công

việc mà các ngôn ngữ khác có thể làm được.

Java là ngôn ngữ vừa biên dịch vừa thông dịch. Đầu tiên mã nguồn được biên dịch bằng công cụ JAVAC để chuyển thành dạng ByteCode. Sau đó được thực thi trên từng loại máy cụ thể nhờ chương trình thông dịch JAVA. Mục tiêu của các nhà thiết kế Java là cho phép người lập trình viết chương trình một lần nhưng có thể chạy trên bất cứ phần cứng cụ thể, thế nên khẩu hiệu của các nhà thiết kế Java là "Write One, Run Any Where".

Ngày nay, Java được sử dụng rộng rãi để viết chương trình chạy trên Internet. Nó là ngôn ngữ lập trình hướng đối tượng độc lập thiết bị, không phụ thuộc vào hệ điều hành. Java không chỉ dùng để viết các ứng dụng chạy đơn lẻ hay trong mạng mà còn để xây dựng các trình điều khiển thiết bị cho điện thoại di động, PDA, …

II. Một số đặc trưng của java

1.Đơn giản

Những người thiết kế mong muốn phát triển một ngôn ngữ dễ học và quen thuộc với đa số người lập trình. Java tựa như C++, nhưng đã lược bỏ đi các đặc trưng phức tạp, không cần thiết của C và C++ như: thao tác con trỏ, thao tác định nghĩa chồng toán tử (operator overloading),… Java không sử dụng lệnh "***goto***" cũng như file header (.h). Cấu trúc "***struct***" và "***union***" cũng được loại bỏ khỏi Java. Nên có người bảo Java là "C++--", ngụ ý bảo java là C++ nhưng đã bỏ đi những thứ phức tạp, không cần thiết.

2. Hướng đối tượng

Có thể nói java là ngôn ngữ lập trình hoàn toàn hướng đối tượng, tất cảc trong java đều là sự vật, đâu đâu cũng là sự vật.

3. Độc lập với hệ nền

Mục tiêu chính của các nhà thiết kế java là độc lập với hệ nền hay còn gọi

là độc lập phần cứng và hệ điều hành. Đây là khả năng một chương trình được viết tại một máy nhưng có thể chạy được bất kỳ đâu

Tính độc lập với phần cứng được hiểu theo nghĩa một chương trình Java nếu chạy đúng trên phần cứng của một họ máy nào đó thì nó cũng chạy đúng trên tất cả các họ máy khác. Một chương trình chỉ chạy đúng trên một số họ máy cụ thể được gọi là phụ thuộc vào phần cứng.

Tính độc lập với hệ điều hành được hiểu theo nghĩa một chương trình Java có thể chạy được trên tất cả các hệ điều hành. Một chương trình chỉ chạy được trên một số hệ điều hành được gọi là phụ thuộc vào hệ điều hành.

Các chương trình viết bằng java có thể chạy trên hầu hết các hệ nền mà không cần phải thay đổi gì, điều này đã được những người lập trình đặt cho nó một khẩu hiệu ***'viết một lần, chạy mọi nơi'***, điều này là không thể có với các ngôn ngữ lập trình khác.

Đối với các chương trình viết bằng C, C++ hoặc một ngôn ngữ nào khác, trình biên dịch sẽ chuyển tập lệnh thành mã máy (machine code), hay lệnh của bộ vi xử lý. Những lệnh này phụ thuộc vào CPU hiện tại trên máy bạn. Nên khi muốn chạy trên loại CPU khác, chúng ta phải biên dịch lại chương trình.

4. Mạnh mẽ Java là ngôn ngữ yêu cầu chặt chẽ về kiểu dữ liệu, việc ép kiểu tự động bừa bãi của C, C++ nay được hạn chế trong Java, điều này làm chương trình rõ ràng, sáng sủa, ít lỗi hơn.Java kiểm tra lúc biên dịch và cả trong thời gian thông dịch vì vậy Java loại bỏ một một số loại lỗi lập trình nhất định.Java không sử dụng con trỏ và các phép toán con trỏ. Java kiểm tra tất cả các truy nhập đến mảng, chuỗi khi thực thi để đảm bảo rằng các truy nhập đó không ra ngoài giới hạn kích thước.

Trong các môi trường lập trình truyền thống, lập trình viên phải tự mình cấp phát bộ nhớ. Trước khi chương trình kết thúc thì phải tự giải phóng bộ nhớ đã cấp. Vấn đề nảy sinh khi lập trình viên quên giải phóng bộ nhớ đã xin cấp trước đó. Trong chương trình Java, lập trình viên không phải bận tâm đến việc cấp phát

bộ nhớ. Qúa trình cấp phát, giải phóng được thực hiện tự động, nhờ dịch vụ thu nhặt những đối tượng không còn sử dụng nữa (garbage collection).

Cơ chế bẫy lỗi của Java giúp đơn giản hóa qúa trình xử lý lỗi và hồi phục sau lỗi.

5. Hỗ trợ lập trình đa tuyến

Đây là tính năng cho phép viết một chương trình có nhiều đoạn mã lệnh được chạy song song với nhau. Với java ta có thể viết các chương trình có khả năng chạy song song một cách dễ dàng, hơn thế nữa việc đồng bộ tài nguyên dùng chung trong Java cũng rất đơng giản. Điều này là không thể có đối với một số ngôn ngữ lập trình khác như C/C++, pascal …

6. Phân tán

Java hỗ trợ đầy đủ các mô hình tính toán phân tán: mô hình client/server, gọi thủ tục từ xa…

7. Hỗ trợ internet

Mục tiêu quan trọng của các nhà thiết kế java là tạo điều kiện cho các nhà phát triển ứng dụng có thể viết các chương trình ứng dụng internet và web một cách dễ dàng, với java ta có thể viết các chương trình sử dụng các giao thức TCP, UDP một cách dễ dàng, về lập trình web phía máy khách java có công nghệ java applet, về lập trình web phía máy khách java có công nghệ servlet/JSP, về lập trình phân tán java có công nghệ RMI, CORBA, EJB, Web Service.

8. Thông dịch

Các chương trình java cần được thông dịch trước khi chạy, một chương trình java được biên dịch thành mã byte code mã độc lập với hệ nền, chương trình thông dịch java sẽ ánh xạ mã byte code này lên mỗi nền cụ thể, điều này khiến java chậm chạp đi phần nào.

III. Các kiểu ứng dụng Java

Với Java ta có thể xây dựng các kiểu ứng dụng sau:

1. Ứng dụng Applets

Applet là chương trình Java được tạo ra để sử dụng trên Internet thông qua các trình duyệt hỗ trợ Java như IE hay Netscape. Applet được nhúng bên trong trang Web. Khi trang Web hiển thị trong trình duyệt, Applet sẽ được tải về và thực thi tại trình duyệt.

2. Ứng dụng dòng lệnh (console)

Các chương trình này chạy từ dấu nhắc lệnh và không sử dụng giao diện đồ họa. Các thông tin nhập xuất được thể hiện tại dấu nhắc lệnh.

3. Ứng dụng đồ họa

Đây là các chương trình Java chạy độc lập cho phép người dùng tương tác qua giao diện đồ họa.

4. JSP/Servlet

Java thích hợp để phát triển ứng dụng nhiều lớp. Applet là chương trình đồ họa chạy trên trình duyệt tại máy trạm. Ở các ứng dụng Web, máy trạm gửi yêu cầu tới máy chủ. Máy chủ xử lý và gửi kết quả trở lại máy trạm. Các Java API chạy trên máy chủ chịu trách nhiệm xử lý tại máy chủ và trả lời các yêu cầu của máy trạm. Các Java API chạy trên máy chủ này mở rộng khả năng của các ứng dụng Java API chuẩn. Các ứng dụng trên máy chủ này được gọi là các JSP/Servlet. hoặc Applet tại máy chủ. Xử lý Form của HTML là cách sử dụng đơn giản nhất của JSP/Servlet. Chúng còn có thể được dùng để xử lý dữ liệu, thực thi các giao dịch và thường được thực thi thông qua máy chủ Web.

5. Ứng dụng cơ sở dữ liệu

Các ứng dụng này sử dụng JDBC API để kết nối tới cơ sở dữ liệu. Chúng có thể là Applet hay ứng dụng, nhưng Applet bị giới hạn bởi tính bảo mật.

6. Ứng dụng mạng

Java là một ngôn ngữ rất thích hợp cho việc xây dựng các ứng dụng mạng. Với thư viện Socket bạn có thể lập trình với hai giao thức: UDP và TCP.

7. Ứng dụng nhiều tầng

Với Java bạn có thể xây dựng phân tán nhiều tầng với nhiều hỗ trợ khác nhau như: RMI, CORBA, EJB, Web Service

8. Ứng dụng cho các thiết bị di động

Hiện nay phần lớn các thiết bị di động như: Điện thoại di động, máy trợ giúp cá nhân… đều hỗ trợ Java. Thế nên bạn có thể xây dựng các ứng dụng chạy trên các thiết bị di động này. Đây là một kiểu ứng dụng khá hấp dãn, bởi vì các thiết bị di động này ngày càng phổ biến và nhu cầu có các ứng dụng chạy trên đó, đặc biệt là các ứng dụng mang tính chất giải trí như game…

IV. Máy ảo Java (JVM-Java Virtual Machine)

Máy ảo là một phần mềm mô phỏng một máy tính thật (máy tính ảo). Nó có tập hợp các lệnh logic để xác định các hoạt động của máy tính và có một hệ điều hành ảo. Người ta có thể xem nó như một máy tính thật (máy tính có phần cứng ảo, hệ điều hành ảo). Nó thiết lập các lớp trừu tượng cho: Phần cứng bên dưới, hệ điều hành, mã đã biên dịch.

Trình biên dịch chuyển mã nguồn thành tập các lệnh của máy ảo mà không phụ thuộc vào phần cứng và hệ điều hành cụ thể. Trình thông dịch trên mỗi máy sẽ chuyển tập lệnh này thành chương trình thực thi. Máy ảo tạo ra một môi trường bên trong để thực thi các lệnh bằng cách:

1 Nạp các file .class

2 Quản lý bộ nhớ

3 Dọn "rác"

Việc không nhất quán của phần cứng làm cho máy ảo phải sử dụng ngăn xếp để lưu trữ các thông tin sau:

1 Các "Frame" chứa các trạng thái của các phương thức.

2 Các toán hạng của mã bytecode.

3 Các tham số truyền cho phương thức.

4 Các biến cục bộ.

Khi JVM thực thi mã, một thanh ghi cục bộ có tên "**Program Counter**" được sử dụng. Thanh ghi này trỏ tới lệnh đang thực hiện. Khi cần thiết, có thể thay đổi nội dung thanh ghi để đổi hướng thực thi của chương trình. Trong trường hợp thông thường thì từng lệnh một nối tiếp nhau sẽ được thực thi.

Một khái niệm thông dụng khác trong Java là trình biên dịch "**Just In Time**-JIT". Các trình duyệt thông dụng như Netscape hay IE đều có JIT bên trong để tăng tốc độ thực thi chương trình Java. Mục đích chính của JIT là chuyển tập lệnh bytecode thành mã máy cụ thể cho từng loại CPU. Các lệnh này sẽ được lưu trữ và sử dụng mỗi khi gọi đến.

BÀI 2 NỀN TẢNG CỦA JAVA

I. Tập ký tư dùng trong java

Mọi ngôn ngữ nói chung, ngôn ngữ lập trình nói riêng đều phải xây dựng trên một tập hợp chữ cái (hay còn gọi là bảng chữ cái), các kí tự được nhóm lại theo một cách nào đó để tạo thành các từ, các từ lại được nhóm lại thành các câu (trong ngôn ngữ lập trình gọi là câu lệnh), một chương trình máy tính tính là một tập các câu lệnh được bố trí theo một trật tự mà người viết ra chúng sắp đặt

Ngôn ngữ java được được xây dựng trên bảng chữ cái unicode, do vậy ta có thể dùng các kí tự unicode để đặt tên cho các định danh.

II. Từ khoá của Java

Mỗi ngôn ngữ lập trình có một tập các từ khoá, người lập trình phải sử dụng từ khoá theo đúng nghĩa mà người thiết kế ngôn ngữ đã đề ra, ta không thể định nghĩa lại nghĩa của các từ khoá, như sử dụng nó để đặt tên biến, hàm..

Sau đây là một số từ khoá thường gặp:

Từ khóa	Mô tả
abstract	Sử dụng để khai báo lớp, phương thức trừu tượng
boolean	Kiểu dữ liệu logic
break	Được sử dụng để kết thúc vòng lặp hoặc cấu trúc switch
byte	kiểu dữ liệu số nguyên
case	được sử dụng trong lện switch
cast	Chưa được sử dụng (để dành cho tương lai)
catch	được sử dụng trong xử lý ngoại lệ
char	kiểu dữ liệu ký tự
class	Dùng để khai báo lớp

const	Chưa được dùng
continue	được dùng trong vòng lặp để bắt đầu một vòng lặp mới
default	được sử dụng trong lệnh switch
do	được dùng trong vòng lặp điều kiện sau
double	kiểu dữ liệu số thực
else	khả năng lựa chọn thứ hai trong câu lệnh if
extends	chỉ rằng một lớp đực kế thừa từ một lớp khác
false	Gía trị logic
final	Dùng để khai báo hằng số, phương thức không thể ghi đè, hoặc lớp không thể kế thừa
finally	phần cuối của khối xử lý ngoại lệ
float	kiểu số thực
for	Câu lệnh lặp
goto	Chưa được dùng
if	Câu lệnh lựa chọn
implements	chỉ rằng một lớp triển khai từ một giao diện
import	Khai báo sử dụng thư viện
instanceof	kiểm tra một đối tượng có phải là một thể hiện của lớp hay không
interface	sử dụng để khai báo giao diện
long	kiểu số nguyên
native	Khai báo phương thức được viết bằng ngông ngữ biên dịch C++
new	tạo một đối tượng mới
null	một đối tượng không tồn tại
package	Dùng để khai báo một gói
private	đặc tả truy xuất
protected	đặc tả truy xuất
public	đặc tả truy xuất

return	Quay từ phương thức về chỗ gọi nó
short	kiểu số nguyên
static	Dùng để khai báo biến, thuộc tính tĩnh
super	Truy xuất đến lớp cha
switch	lệnh lựa chọn
synchronized	một phương thức độc quyền truy xuất trên một đối tượng
this	Ám chỉ chính lớp đó
throw	Ném ra ngoại lệ
throws	Khai báo phương thức ném ra ngoại lệ
true	Giá trị logic
try	sử dụng để bắt ngoại lệ
void	Dùng để khai báo một phương thức không trả về giá trị
while	Dùng trong cấu trúc lặp

III. Định danh (tên)

Tên dùng để xác định duy nhất một đại lượng trong chương trình. Trong java tên được đặt theo quy tắc sau:

- Không trùng với từ khoá
- Không bắt đầu bằng một số, tên phải bắt đầu bằng kí tự hoặc bắt đầu bằng kí $,_
- Không chứa dấu cách, các kí tự toán học như +, -, *,/, %..
- Không trùng với một định danh khác trong cùng một phạm vi

chú ý:

- Tên nên đặt sao cho có thể mô tả được đối tượng trong thực tế
- Giống như C/C++, java có phân biệt chữ hoa chữ thường
- Trong java ta có thể đặt tên với độ dài tuỳ ý
- Ta có thể sử dụng các kí tự tiếng việt để đặt tên

Quy ước về đặt tên trong java

Ta nên đặt tên biến, hằng, lớp, phương thức sao cho nghĩa của chúng rõ ràng, dễ hiểu, khoa học và mang tính ước lệ quốc tế. Do java có phân biệt chữ hoa, chữ thường nên ta phải cẩn thận và chú ý.

Sau đây là quy ước đặt tên trong java (chú ý đây chỉ là quy ước do vậy không bắt buộc phải tuân theo quy ước này):

- Đối với biến và phương thức thì tên bao giờ cũng bắt đầu bằng ký tự thường, nếu tên có nhiều từ thì ghép lại thì: ghép tất cả các từ thành một, ghi từ đầu tiên chữ thường, viết hoa kí tự đầu tiên của mỗi từ theo sau trong tên, ví dụ area, radius, readInteger…

- Đối với tên lớp, giao diện ta viết hoa các kí tự đầu tiên của mỗi từ trong tên, ví dụ lớp WhileTest, Circle

- Tên hằng bao giờ cũng viết hoa, nếu tên gồm nhiều từ thì chúng được nối với hau bởi kí tự ghạch dưới '_', ví dụ PI, MAX_VALUE

IV. Cấu trúc một chương trình java

- Mỗi ứng dụng Java bao gồm một hoặc nhiều đơn vị biên dịch (mỗi đơn vị biên dịch là một tệp tin có phần mở rộng Java)

- Mỗi đơn vị biên dịch bao gồm một hoặc nhiều lớp

- Mỗi ứng dụng độc lập phải có duy nhất một phương thức main (điểm bắt đầu của ứng dụng)

- Mỗi đơn vị biên dịch có nhiều nhất một lớp được khai báo là public, nếu như trong đơn vị biên dịch có lớp public thì tên của đơn vị biên dịch phải trùng với tên của lớp public (giống hệt nhau cả ký tự hoa lẫn ký tự thường)

- Bên trong thân của mối lớp ta khai báo các thuộc tính, phương thức của lớp đó, Java là ngôn ngữ hướng đối tượng, do vậy mã lệnh phải nằm trong lớp nào đó. Mỗi lệnh đều được kết thúc bằng dấu chấm phảy ";".

- Trong ngôn ngữ Java, lớp là một đơn vị mẫu có chứa dữ liệu và mã lệnh liên quan đến một thực thể nào đó. Khi xây dựng một lớp, thực chất bạn đang tạo ra một một kiểu dữ liệu. Kiểu dữ liệu mới này được sử dụng để xác định các biến mà ta thương gọi là "đối tượng". Đối tượng là các thể hiện (instance) của lớp. Tất cả các đối tượng đều thuộc về một lớp có chung đặc tính và hành vi. Mỗi lớp xác định một thực thể, trong khi đó mỗi đối tượng là một thể hiện thực sự.

- Khi ban khai báo một lớp, bạn cần xác định dữ liệu và các phương thức của lớp đó. Về cơ bản một lớp được khai báo như sau:

Cú pháp:

class classname

{ var_datatype variablename;

:

met_datatype methodname(parameter_list)

:

}

Trong đó:

class - *Từ khoá xác định lớp*

classname *- Tên của lớp*

var_datatype *- kiểu dữ liệu của biến*

variablename *- Tên của biến*

met_datatype *- Kiểu dữ liệu trả về của phương thức*

methodname *- Tên của phương thức*

parameter_list *– Các tham số được của phương thức*

- Bạn còn có thể định nghĩa một lớp bên trong một lớp khác. Đây là lớp xếp lồng nhau, các thể hiện (instance) của lớp này tồn tại bên trong thể hiện của một lớp che phủ chúng. Nó chi phối việc truy nhập đến các thành phần của lớp bao phủ chúng. Có hai loại lớp trong đó là lớp trong tĩnh "static" và lớp trong không tĩnh "non static"

+ Lớp trong tĩnh (static)

Lớp trong tĩnh được định nghĩa với từ khoá "**static".** Lớp trong tĩnh có thể truy nhập vào các thành phần tĩnh của lớp phủ nó.

+ Lớp trong không tĩnh (non static)

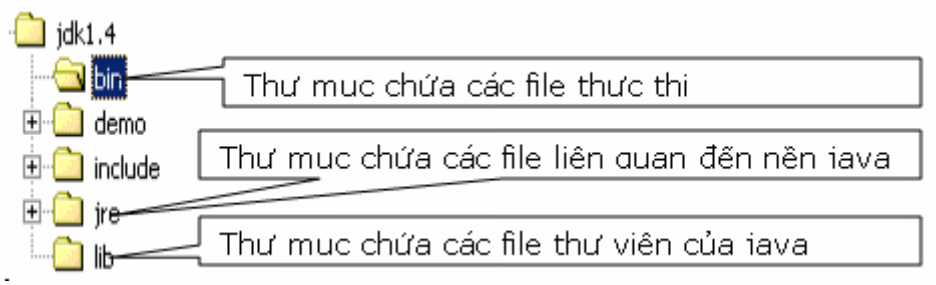
Lớp bên trong (không phải là lớp trong tĩnh) có thể truy nhập tất cả các thành phần của lớp bao nó, song không thể ngược lại.

V. Chương trình JAVA đầu tiên

Để có thể biên dịch và chạy các chương trình java ta phải cài

- JRE (Java Runtime Enviroment) môi trường thực thi của java, nó bao gồm: JVM (Java Virtual Machine) máy ảo java vì các chương trình java được thông dịch và chạy trên máy ảo java và tập các thư viện cần thiết để chạy các ứng dụng java.
- Bộ công cụ biên dịch và thông dịch JDK của Sun Microsystem

Sau khi cài đặt JDK (giả sử thư mục cài đặt là C:\JDK1.4) ta sẽ nhận được một cấu trúc thư mục như sau:



- Để biên dịch một chương trình java sang mã byte code ta dùng lệnh

C:\JDK1.4\BIN\javac TênTệp.java

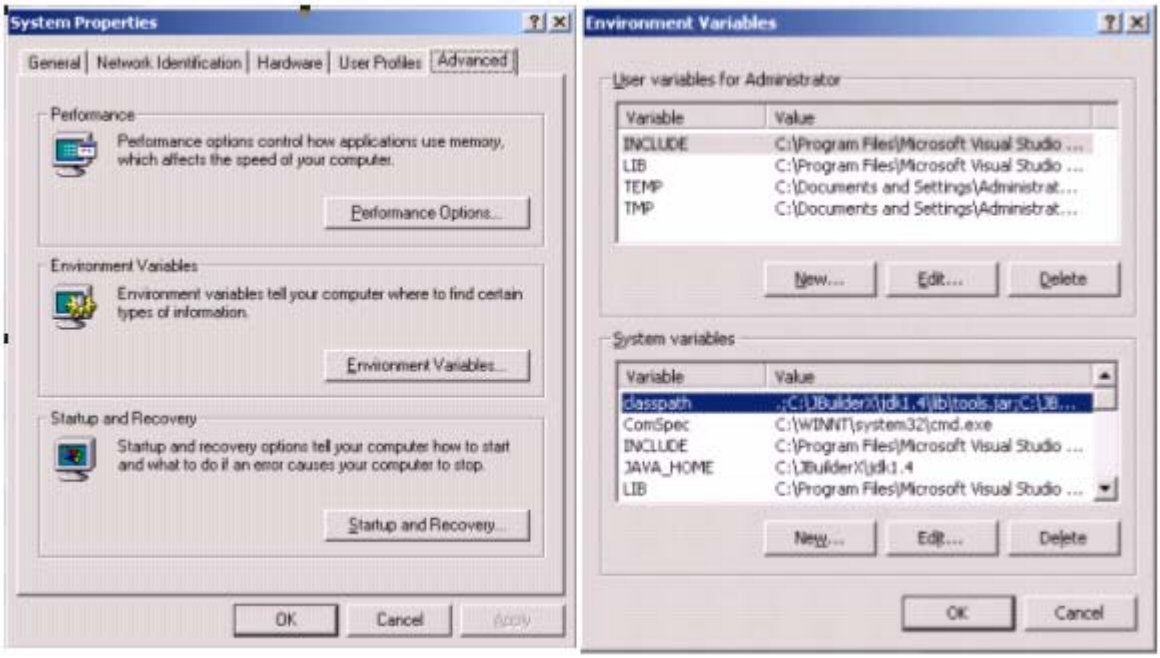
- Để thông dịch và chạy chương trình ta sử dụng lệnh

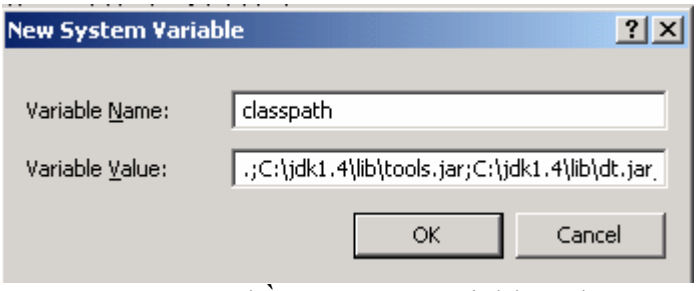
C:\JDK1.4\BIN\java TênTệp

Để biên dịch và chạy chương trình Java đơn giản ta nên thiết đặt hai biến môi trường "***paht***" và "***classpath***" như sau:

- Đối với dòng WinNT:

+ R-Click vào My Computer→ chọn Properties→ chọn Advanced→Enviroment Variables





+ Trong phần System variables chọn new để thêm biến môi trường mới, trong hộp thoại hiện ra gõ "classpath" vào ô Variable Name và ".;C:\jdk1.4\lib\tools.jar;C:\jdk1.4\lib\dt.jar;C:\jdk1.4\jre\lib\rt.jar" trong ô variable value (chú ý không gõ dấu " vào, mục đích để cho dễ nhìn mà thôi)

+ Cũng trong phần System variables tìm đến phần path trong danh sách→chọn edit để sửa lại giá trị hiện có, trong ô value ta thêm vào cuối ";C:\jdk1.4\bin"

Công việc đặt các biến môi trường đã xong, để thấy được tác dụng của các biến môi trường ta cần phải khởi động lại máy

- Đối với dòng Win9X:

Mở tệp C:\Autoexec.bat sau đó thêm vào hai dòng sau:

+classpath=.;C:\jdk1.4\lib\tools.jar;C:\jdk1.4\lib\dt.jar;C:\jdk1.4\jre\lib\rt.jar

+ path=…;c:\jdk1.4\bin

Khởi động lại máy để thấy được tác dụng của các biến môi trường này

Ví dụ đầu tiên: chương trình Hello World (chương trình khi chạy sẽ in ra màn hình lời chào Hello World)

Các bước:

- Mở một chương trình soạn thảo văn bản hỗ trợ asciii, như notepad, wordpad, EditPlus… và gõ vào các dòng sau:

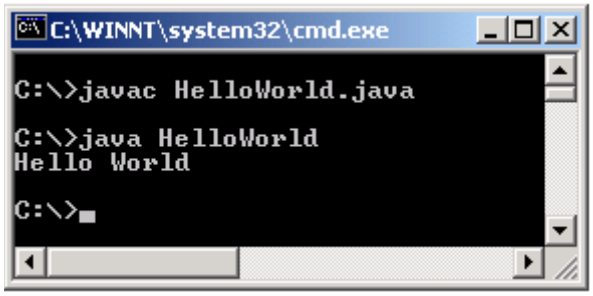
```
public class HelloWorld {
        public static void main(String[] args){
                System.out.println("Hello World");
        }
}
```

- Ghi lại với cái tên C:\HelloWorld.java (chú ý tên tệp phải trùng với tên lớp, kể cả chữ hoa chữ thường, phần mở rộng là java)

- Mở của sổ DOS Prompt

  + chuyển vào thư mục C:\

  + Gõ lệnh javac HelloWorld.java để biên dịch chương trình, nếu việc biên dịch thành công (chương trình không có lỗi cú pháp) thì ta sẽ thu được tệp HelloWorld.class trong cùng thư mục, nếu trong chương trình còn lỗi cú pháp thì trong bước này ta sẽ nhận được một thông báo lỗi và lúc này tệp HelloWorld.class cũng không được tạo ra



  + Gõ lệnh java HelloWorld (chú ý không gõ phần mở rộng) để chạy chương trình HelloWorld.

Sau khi thông dịch và chạy ta nhận được

VI. Chú thích trong chương trình

Trong java ta có 3 cách để ghi chú thích

Cách 1: sử dụng cặp /* và */ ý nghĩa của cặp chú thích này giống như của C, C++

Cách 2: sử dụng cặp // ý nghĩa của cặp chú thích này giống như của C, C++

Cách 3: sử dụng cặp /** và */, đây là kiểu chú thích tài liệu (không có trong C/C++), nó dùng để tạo ra tài liệu chú thích cho chương trình.

Với cách thứ nhất và cách ba ta có thể viết chú thích trên nhiều dòng, với cách chú thích hai ta chỉ có thể chú thích trên một dòng.

Chú ý: trong java ta có thể đặt chú thích ở đâu?, câu trả lời là: ở đâu có thể đặt được một dấu cách thì ở đó có thể đặt chú thích.

VII. Kiểu dữ liệu

1. Các kiểu dữ liệu nguyên thuỷ

Từ khoá	Mô tả	Kích cỡ	Tối thiểu	Tối đa	Lớp bao
(kiểu số nguyên)					
byte	số nguyên một byte	8 bit	-128	127	Byte
short	số nguyên ngắn	16 bit	-2^{15}	$2^{15}-1$	Short
int	số nguyên	32 bit	-2^{31}	$2^{31}-1$	Integer
long	số nguyên dài	64 bit	-2^{63}	$-2^{63}-1$	Long
(kiểu số thực)					
float	kiểu thực với độ chính xác đơn	32 bit	IEEE754	IEEE754	Float
double	Double-precision floating point	64 bit	IEEE754	IEEE754	Double
(kiểu khác)					
char	kiểu kí tự	16 bit	Unicode 0	Unicode $2^{16}-1$	Character

boolean	kiểu logic	true hoặc false	-	-	Boolean
void	-	-	-	-	Void

Đặc điểm của các biến có kiểu nguyên thủy là vùng nhớ của chúng được cấp phát ở phần stack. Do vậy việc truy xuất vào một biến kiểu nguyên thủy rất nhanh.

2. Kiểu tham chiếu

Trong Java có 3 kiểu dữ liệu tham chiếu

Kiểu dữ liệu	**Mô tả**
Mảng (Array)	Tập hợp các dữ liệu cùng kiểu.
Lớp (Class)	Là sự cài đặt mô tả về một đối tượng trong bài toán.
Giao diện (Interface)	Là một lớp thuần trừu tượng được tạo ra cho phép cài đặt đa thừa kế trong Java.

Đặc điểm của các biến kiểu tham chiếu là nó chứa địa chỉ của đối tượng mà nó trỏ đến.

Vùng nhớ của biến tham chiếu được cấp phát ở vùng nhớ stack còn vùng nhớ của đối tượng được cấp phát ở vùng nhớ heap. Việc truy xất vào vùng nhớ heap chậm hơn truy xất vào vùng nhớ stack tuy nhiên java có cơ chế cho phép truy cập vào vùng nhớ heap với tốc độ xấp xỉ bằng tốc độ truy cập vào vùng nhớ stack.

VIII. Khai báo biến

1. Khai báo biến

Tương tự ngôn ngữ C/C++, để khai báo biến trong java ta sử dụng cú pháp sau:

type name [=InitValue];

trong đó:

- type là kiểu dữ liệu cuả biến
- name là tên của biến, tên biến là một xâu kí tự được đặt theo quy tắc đặt tên của java
- InitValue là giá trị khởi tạo cho biến, đây là phần tuỳ chọn, nếu bỏ qua phần này thì giá trị ban đầu của biến được khởi tạo giá trị mặc định

Chú ý:

- Nếu cần khai báo nhiều biến có cùng một kiểu dữ liệu ta có thể đặt các khai báo các biến trên một dòng, các biến này được phân cách nhau bởi dấu phảy

- Java sẽ xử lý các biến không được khởi đầu giá trị như sau:

    + Đối với thuộc tính (biến được khai báo trong phạm vi của lớp) thì Java sẽ tự động khởi gán giá trị cho các biến theo quy tắc sau:

        + giá trị 0 cho kiểu dữ liệu số

        + false cho kiểu logic

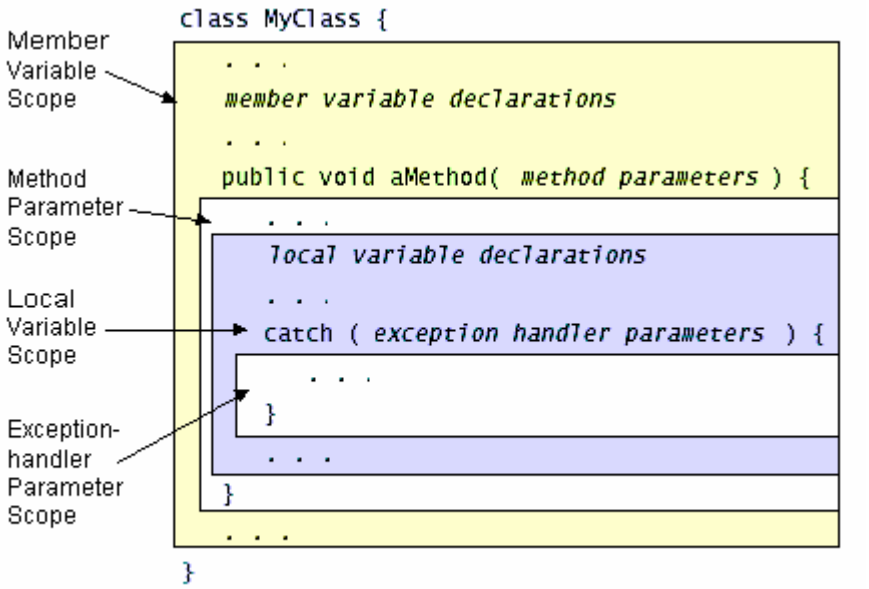
        + kí tự null (mã 0) cho kí tự

        + giá trị null cho kiểu đối tượng

    + Đối với các biến cục bộ thì biến không được khởi gán giá trị mặc định, tuy nhiên Java sẽ báo lỗi nếu ta sử dụng một biến chưa được nhận giá trị

2. phạm vi biến

Mỗi biến được khai báo ra có một phạm vi hoạt động, phạm vi của biến là nơi mà biến có thể được truy cập, điều này xác định cả tính thấy được và thời gian sống của biến.



- biến phạm vi lớp là biến được khai báo bên trong lớp nhưng bên ngoài các phương thức và hàm tạo, tuy nhiên việc khai báo phải xuất hiện trước khi biến được sử dụng
- biến phạm vi cục bộ là biến được khai báo bên trong một khối, phạm vi của biến tính từ điểm biến được khai báo cho đến cuối khối mà biến được khai báo

Ví dụ:

```
{
int i=1;
// chỉ có i sẵn sàng sử dụng
{
int j=10;
// cả i và j đều sẵn sàng
}
// chỉ có i sẵn sàng
// j không sẵn sàng vì nằm ngoài phạm vi
}
```

Chú ý: Ta không thể làm điều sau cho dù nó có thể trong C/C++

```
{
```

```
int i=1;
{
int i=10;// không được phép vì đã có một biến cùng tên với nó
}
}
```

những người thiết kế java cho rằng điều đó có thể gây lần lộn, do vậy họ đã quyết định không cho phép che giấu một biến ở phạm vi lớn hơn.

Chú ý: thời gian sống của các đối tượng không tuân theo quy luật thời gian sống của các biến kiểu nguyên thuỷ.

VII. Một số phép toán trên kiểu dữ liệu nguyên thuỷ

1. Phép gán

Cú pháp Biến=BiểuThức;

Phép gán được thực hiện bằng toán tử '=', nó có nghĩa là " hãy tính toán giá trị biểu thức bên phải dấu gán, sau đó đưa giá trị đó vào ô nhớ có tên nằm ở bên trái dấu gán'

Chú ý:

+ câu lệnh gán gồm một dấu '='
+ kiểu của biểu thức bên phải dấu gán phải tương thích với kiểu dữ liệu của biến
+ trong java ta có thể thực hiện một dẫy gán như sau:

```
i = j = 10;// cả i và j đều có giá trị 10
```

2. Toán tử toán học

Ngôn ngữ java cũng có các phép toán số học như các ngôn ngữ khác: + (phép cộng), - (phép trừ), * (phép nhân),/ (phép chia), % (phép toán chia lấy phần nguyên)

Ta mô tả tóm tắt các phép toán số học qua bảng tổng kết sau:

Phép toán	Sử dụng	Mô tả
+	op1 + op2	Cộng op1 vớiop2
-	op1 - op2	Trừ op1 cho op2
*	op1 * op2	Nhân op1 với op2
/	op1/ op2	chia op1 cho op2
%	op1 % op2	Tính phần dư của phép chia op1 cho op2

3. Toán tử tăng, giảm

Giống như ngôn ngữ C/C++, java cũng có phép toán tăng, giảm, ta có thể mô tả tóm tắt qua các bằng sau:

Phép toán	Sử dụng	Mô tả
++	op++	Tăng op lên 1 đơn vị, giá trị của op được tăng lên trước khi biểu thức chứa nó được tính
++	++op	Tăng op lên 1 đơn vị, giá trị của op được tăng lên sau khi biểu thức chứa nó được tính
--	op--	Giảm op xuống1 đơn vị, giá trị của op được giảm xuống trước khi biểu thức chứa nó được tính
--	--op	Giảm op xuống1 đơn vị, giá trị của op được giảm xuống sau khi biểu thức chứa nó được tính

Chú ý: nếu toán tử tăng trước, tăng sau(giảm trước, giảm sau) đứng một mình(không nằm trong biểu thức) thì chúng hoạt động như nhau, chúng chỉ khác nhau khi chúng nằm trong biểu thức

4. Phép toán quan hệ

Phép toán quan hệ bao giờ cũng cho kết quả boolean, phép toán quan hệ sẽ so sánh 2 giá trị, nó xác định mối quan hệ giữa chúng, ví dụ! = sẽ trả về true nếu 2 toán hạng là khác nhau.

Ta tóm tắt các phép toán qua bảng sau:

Phép toán	Sử dụng	Nhận về giá trị true khi
>	op1 > op2	op1 lớn hơn op2
>=	op1 >= op2	op1 lớn hơn hoặc bằng op2
<	op1 < op2	op1 nhỏ hơn op2
<=	op1 <= op2	op1 nhỏ hơn hoặc bằng op2
==	op1 == op2	op1 bằng op2
!=	op1! = op2	op1 khác op2

Ví dụ sử dụng các phép toán quan hệ

```
public class RelationalDemo {
public static void main(String[] args) {

// a few numbers
int i = 37;
int j = 42;
int k = 42;
System.out.println("Variable values...");
System.out.println(" i = " + i);
System.out.println(" j = " + j);
System.out.println(" k = " + k);

        //greater than
System.out.println("Greater than...");
System.out.println(" i > j = " + (i > j));      // false
System.out.println(" j > i = " + (j > i));// true
System.out.println(" k > j = " + (k > j));// false, they are equal
```

```
        //greater than or equal to
System.out.println("Greater than or equal to...");
System.out.println(" i >= j = " + (i >= j));// false
System.out.println(" j >= i = " + (j >= i));// true
System.out.println(" k >= j = " + (k >= j));// true

        //less than
System.out.println("Less than...");
System.out.println(" i < j = " + (i < j));// true
System.out.println(" j < i = " + (j < i));// false
System.out.println(" k < j = " + (k < j));// false

        //less than or equal to
System.out.println("Less than or equal to...");
System.out.println(" i <= j = " + (i <= j));// true
System.out.println(" j <= i = " + (j <= i));// false
System.out.println(" k <= j = " + (k <= j));// true

        //equal to
System.out.println("Equal to...");
System.out.println(" i == j = " + (i == j));// false
System.out.println(" k == j = " + (k == j));// true

        //not equal to
System.out.println("Not equal to...");
System.out.println(" i! = j = " + (i! = j));// true
System.out.println(" k! = j = " + (k! = j));// false

}
```

```
}
```

Đây là đầu ra của chương trình

```
Variable values...
i = 37
j = 42
k = 42
Greater than...
i > j = false
j > i = true
k > j = false
Greater than or equal to...
i >= j = false
j >= i = true
k >= j = true
Less than...
i < j = true
j < i = false
k < j = false
Less than or equal to...
i <= j = true
j <= i = false
k <= j = true
Equal to...
i == j = false
k == j = true
Not equal to...
i! = j = true
k! = j = false
```

5. Phép toán logic

Java hỗ trợ 6 phép toán logic được chỉ ra trong bảng sau:

Phép toán	Sử dụng	Nhận về giá trị true khi
&&	op1 && op2	Cả op1 và op2 đều là true, giá trị của op2 chỉ được tính khi op1 là true
||	op1 || op2	Hoặc op1 hoặc op2 là true, giá trị của op2 chỉ được tính khi op1 là false
!	! op	op là false
&	op1 & op2	Cả op1 và op2 đều là true, giá trị của op2 luôn được tính kể cả khi op1 là false
|	op1 | op2	Hoặc op1 hoặc op2 là true, giá trị của op2 luôn luôn được tính kể cả khi op1 là true
^	op1 ^ op2	Nếu op1 khác op2

Nhận xét:

+ Phép toán && (&) chỉ nhận giá trị true khi và chỉ khi cả hai toán hạng đều là true

+ Phép toán || (|) chỉ nhận giá trị false khi và chỉ khi cả hai toán hạng là false

+ Phép toán ^ chỉ nhận giá trị true khi và chỉ khi hai toán hạng khác nhau

6. phép toán thao tác trên bit

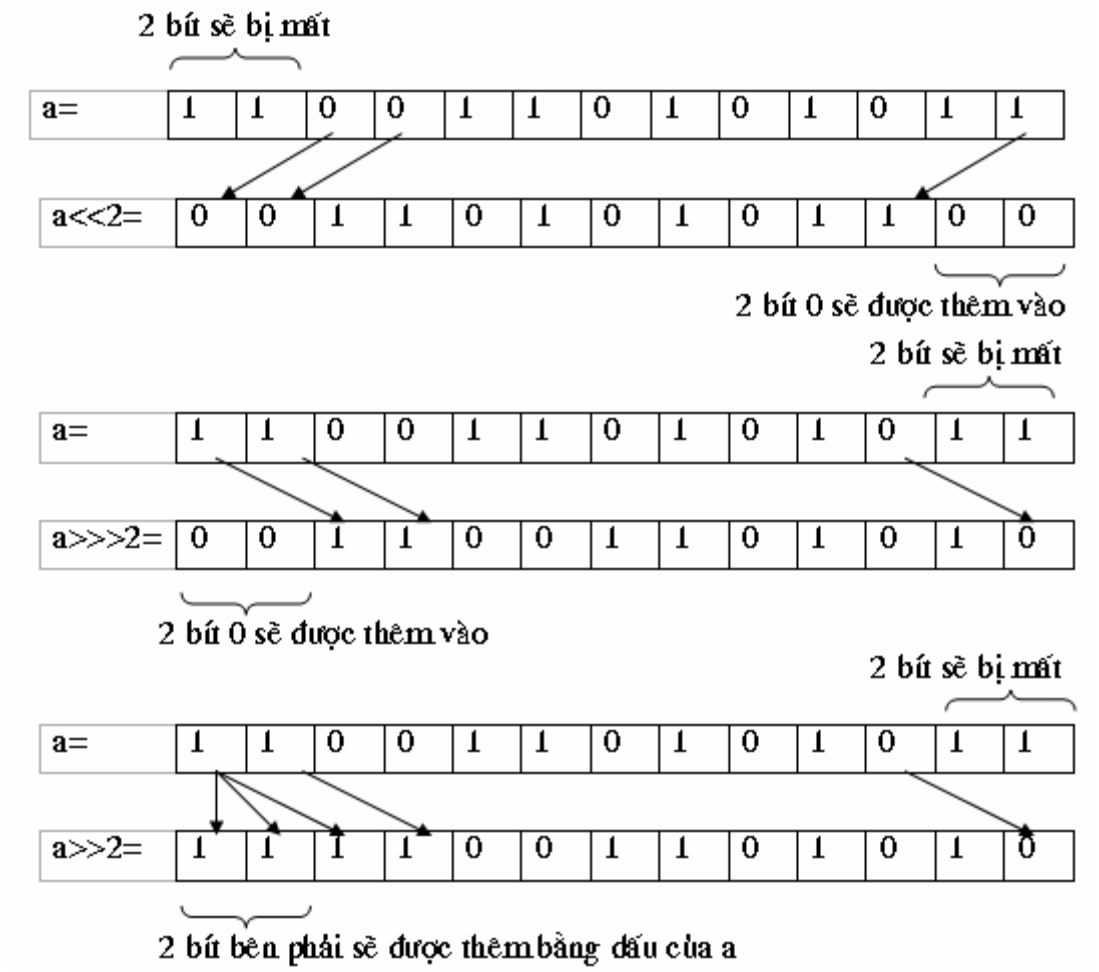
6.1. Phép toán dịch bit

Ta sẽ mô tả phép toán dịch chuyển qua bảng sau:

Phép toán	Sử dụng	Kết quả
>>	op1 >> op2	Dịch chuyển op1 sang phải op2 bit, op2 bit phía bên phải sẽ được điền bằng các bít 0
<<	op1 << op2	Dịch chuyển op1 sang trái op2 bit(giữ nguyên dấu của op1), op2 bit nằm bên trái sẽ được điền bằng các bít 0

>>>	op1>>> op2	Dịch chuyển op1 sang phải op2 bit, op2 bit

Sau đây là hình minh hoạ phép toán dịch bít



Ví dụ:

13>>1=6 vì 13=11012 do vậy khi dịch phải một bit ta sẽ được 1102=6

5<<1=10 vì 5=1012 do vậy khi dịch trái 1 bit ta sẽ được 10102=10

5<<2=100 vì 5=1012 do vậy khi dịch trái 2 bit ta sẽ được 101002=100

Nhận xét: phép toán dịch trái một bit chính là phép nhân với 2, còn dịch phải chính là phép chia cho 2

6.2. Phép toán logic trên bit

Các phép toán thao tác bit cho phép ta thao tác trên từng bit riêng lẻ trong một kiểu dữ liệu thích hợp, các phép toán thao tác bit thực hiện đại số boolean trên các bit tương ứng của 2 toán hạng để tạo ra kết quả

Ta tóm tắt các phép toán trong bảng sau:

Phép toán	Sử dụng	Thực hiện
&	op1 & op2	Thực hiện phép and các bit tương ứng của op1 với op2
|	op1 | op2	Thực hiện phép or các bit tương ứng của op1 với op2
^	op1 ^ op2	Thực hiện phép xor các bit tương ứng của op1 với op2
~	~op2	Thực hiện phép lật các bit của op2

Bảng giá trị chân lý của các phép toán đái số boolean:

Phép AND		
op1	**op2**	**Result**
0	0	0
0	1	0
1	0	0
1	1	1

Phép OR		
op1	**op2**	**Result**
0	0	0
0	1	1
1	0	1
1	1	1

Phép XOR		
op1	**op 2**	**Result**

0	0	0
0	1	1
1	0	1
1	1	0

Phép NOT	
op1	**Result**
0	1
1	0

Ví dụ:

1101// 13

& 1100// 12

1100// 12

1101// 13

| 1100// 12

1101// 13

1101// 13

^ 1100// 12

0001// 1

! 10101=01010

7. Toán tử gán tắt

Giống như C/C++ java cũng có toán tử gán, ta tóm tắt các toán tử gán qua bảng sau:

Phép gán	Sử dụng	Tương đương
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1/ = op2	op1 = op1/ op2
%=	op1 %= op2	op1 = op1 % op2
&=	op1 &= op2	op1 = op1 & op2
|=	op1 |= op2	op1 = op1 | op2
^=	op1 ^= op2	op1 = op1 ^ op2
<<=	op1 <<= op2	op1 = op1 << op2
>>=	op1 >>= op2	op1 = op1 >> op2
>>>=	op1 >>>= op2	op1 = op1 >>> op2

8. Thứ tự ưu tiên của các phép toán

Thứ tự ưu tiên của các phép toán xác định trình tự tính toán giá trị của một biểu thức, java có những quy tắc riêng để xác định trình tự tính toán của biểu thức, ta phải nhớ quy tắc sau:

- các phép toán một ngôi bao giờ cũng được thực hiện trước tiên
- trong một biểu thức có nhiều phép toán thì phép toán nào có độ ưu tiên cao hơn sẽ được thực hiện trước phép toán có độ ưu tiên thấp
- trong một biểu thức có nhiều phép toán có độ ưu tiên ngang nhau thì chúng sẽ

được tính theo trình tự từ trái qua phải

Ta có bảng tóm tắt thứ tự ưu tiên của các phép toán trong bảng sau:

postfix operators	[]. (params) expr++ expr--
unary operators	++expr --expr +expr -expr ~!
creation or cast	new (*type*)*expr*
multiplicative	*/ %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	==! =
Bitwise AND	&
Bitwise exclusive OR	^
Bitwise inclusive OR	|
Logical AND	&&
Logical OR	||
Conditional	?:
Assignment	= += -= *=/ = %= &= ^= |= <<= >>= >>>=

Trong bảng trên thứ tự ưu tiên của các phép toán được giảm từ trên xuống dưới, trên cùng một hàng thì chúng có độ ưu tiên ngang nhau.

1. Toán tử dấy

Không giống như C/C++, trong java chỗ duy nhất mà ta có thể đặt toán tử dấy là bên trong cặp ngoặc tròn của cấu trúc for(sẽ được mô tả chi tiết trong chương sau)

IX. Toán tử chuyển kiểu

9.1 Chuyển đổi kiểu không tường minh

Việc chuyển đổi kiểu thường được diễn ra một cách tự động trong trường hợp biểu thức gồm nhiều toán hạng có kiểu dữ liệu khác nhau. Điều này đôi khi làm cho bạn khá ngạc nhiên vì nhận được một kết quả không theo ý muốn. Ví dụ ta xét đoạn trình sau:

```
int two=2, three=3;
float result=1.5 +three/two;
```

kết quả nhận được của result là 2.5. Điều mà bạn mong muốn là 3.0 chứ không phải là 2.5. Kết quả 2.5 nhận được là do three và two là hai giá trị nguyên nên kết quả của phép chia three/two cho ta một giá trị nguyên bàng 1 chứ không phải là 1.5. Để nói rằng kết quả của phép chia three/two là một giá trị thực chứ không phải là một giá trị nguyên thì một trong hai toán hạng của phép chia này phải là một số thực. Do vậy ta cần phải chuyển kiểu của một trong hai toán hạng này hoặc cả hai thành số thực. Để nhận được kết quả đúng trong trường hợp này bạn cần viết như sau:

```
float result=1.5 +(float)three/two; hoặc
float result=1.5 +three/(float)two; hoặc
float result=1.5 +(float)three/(float)two;
```

Lý do mà ta viết như trên là nếu trong một phép toán có sự tham gia của nhiều toán hạng có kiểu khác nhau thì java sẽ chuyển kiểu tự động cho các toán hạng một cách tự động theo quy tắc sau:

byte -> short -> int -> long -> float -> double

9.2. Chuyển đổi kiểu tường minh

Để chuyển đổi kiểu một cách tường minh ta sử dụng cú pháp sau:

```
(type) biểu_thức;
```

khi gặp câu lệnh này java sẽ tính toán giá trị của biểu thức sau đó chuyển đổi kiểu giá trị của biểu thức thành kiểu type.

Ví dụ:

(int) 2.5 * 2 = 4

(int) 2.5 * 2.5 = 5

(int)(2.5 * 2.5) = 6

1+(float)5/2=1+5/(float)2=1+(float)5/(float)2=3.5

Chú ý:

1. Phép toán chuyển kiểu là phép toán có độ ưu tiên cao, nên (int)3.5*2≠(int)(3.4*2)
2. Cần chú ý khi chuyển một biểu thức kiểu dữ liệu có miền giá trị lớn sang một kiểu có miền giá trị nhỏ hơn. Trong trường hợp này có thể bạn sẽ bị mất thông tin.

X. Các hàm toán học

Các hàm toán học như sin, cos, sqrt được java viết sẵn trong lớp Math. Lớp này nằm trong gói java.lang (gói mặc định) do vậy bạn không cần phải thêm câu lệnh import ở đầu chương trình để có thể sử dụng lớp này. Các hàm này được viết là các phương thức tĩnh do vậy ta không cần phải tạo ra thể hiện của lớp Math.
Bảng sau liệt kê một số phương thức tĩnh trong lớp Math:

Tên phương thức	**Mô tả ý nghĩa**	**Kiểu tham số**	**Kiểu trả về**
sin(arg)	tính sin của arg	arg là một biểu thức kiểu double thể hiện một cung theo radians	double
cos(arg)	tính cos của arg	arg là một biểu thức kiểu double thể hiện một cung theo radians	double

tan(arg)	tính tang của arg	arg là một biểu thức kiểu double thể hiện một cung theo radians	double
asin(arg)	tính sin-1 (arcsin) arg	arg là một biểu thức kiểu double thể hiện một cung theo radians	double trong hệ radians
acos(arg)	tính cos-1 (arccosin) của arg	arg là một biểu thức kiểu double thể hiện một cung theo radians	double trong hệ radians
atan(arg)	tính tan-1 (arctang) của arg	arg là một biểu thức kiểu double thể hiện một cung theo radians	double trong hệ radians
atan2 (arg1,arg2)	tính tan-1 (arctang) của arg1/arg2	arg1,arg2 là các biểu thức kiểu double thể hiện một cung theo radians	double trong hệ radians
abs(arg)	tính trị tuyệt đối của arg	arg là một biểu thức kiểu int, long, float, hoặc double	The same type as the argument
max (arg1,arg2)	Nhận về giá trị lớn	arg1, arg2 là một	Nhận về kiểu cùng

	trong hai tham số	biểu thức kiểu int, long, float, hoặc double	kiùu với tham số
min (arg1,arg2)	Nhận về giá trị nhỏ trong hai tham số	arg1, arg2 lµ mét biùu thøc kiùu int, long, float, hoÆc double	Nhận về kiểu cùng kiùu với tham số
ceil(arg)	Nhận về giá trị nguyên nhỏ hơn hoặc bằng arg	arg là biểu thức kiểu float hoặc double	double
floor(arg)	Nhận về giá trị nguyên lớn hơn hoặc bằng arg	arg là biểu thức kiểu float hoặc double	double
round(arg)	Trả về giá trị nguyên gần arg nhất, giá trị này chính là giá trị của arg sau khi đã làm tròn	arg là biểu thức kiểu float hoặc double	Nhận về kiểu int nếu arg kiêu float, nhận về kiùu long nếu arg kiểu double
rint(arg)	Giống như round(arg)	arg là biểu thức kiểu double	double
sqrt(arg)	tính căn bậc hai của arg	arg là biểu thức kiểu double	double
pow (arg1,arg2)	tính arg1arg2	Cả arg1 và arg2 là các biểu thức kiểu	double

		double	
exp(arg)	tính earg	arg là biểu thức kiểu double	double
log(arg)	tính logarithmớ số e của arg	arg là biểu thức kiểu double	double
random()	Nhận về một số giả ngẫu nhiên nằm trong khoản [0, 1)	Không có tham số	double

Ví dụ về các hàm toán học trong lớp Math, bạn hãy gõ đoạn chương trình sau và cho chạy thử để thấy được kết quả tính toán của các hàm toán học.

XI. Các phép toán trên kiểu kí tự

Đối với kiểu kí tự ta có thể thực hiện các phép toán số học (như: +, -, *,/) và các phép toán quan hệ.

Ví dụ:

char kt1=’A’;

char kt2=tk1+a;// kt2 nhận ký tự B

char kt3=(char)33*2;// kt3 nhận ký tự B

(kt1>kt2)= false;

(kt2=kt3)= false;

BÀI 3 ĐIỀU KHIỂN LUỒNG CHƯƠNG TRÌNH

Chương trình là một dẫy các lệnh được bố trí thực hiện theo một trình tự nào đó, nhưng đôi khi ta muốn điều khiển luồng thực hiện của chương trình tuỳ thuộc vào điều kiện gì đó. Ngôn ngữ lập trình java cung cấp một số phát biểu cho phép ta điều khiển luồng thực hiện của chương trình, chúng được liệt kê trong bảng sau:

Kiểu lệnh	**Từ khoá**
Lặp	while, do-while, for
Quyết định	if-else, switch-case
Xử lý lỗi	try-catch-finally, throw
Rẽ nhánh	break, continue, label:, return

I. cấu trúc rẽ nhánh

1.1. phát biểu if

a) dạng khuyết

Cú pháp

```
if(Boolean-expression)
    statement;
```

sự hoạt động của cấu trúc if thiếu được mô ta qua sơ đồ sau:

b) dạng đủ

Cú pháp

```
if(Boolean-expression)
        statement1;
else
        statement2;
```

sự hoạt động của cấu trúc if thiếu được mô ta qua sơ đồ sau:

1.2. biểu thức điều kiện

Cú pháp:

Variable=booleanExpression? true-result-expression:

false-result-expression;

1.3. cấu trúc switch

a) Dạng khuyết

Cú pháp

```
switch(biểu_thức) {
case gt_1:
        lệnh 1; [ break;]
case gt_2:
        lệnh 2; [ break;]
…
case gt_n:
        lệnh n; [ break;]
```

}

Sau đây là sơ đồ khối mô tả sự hoạt động của cấu trúc rẽ nhánh switch dạng thiếu

b) Dạng đủ

Cú pháp

```
switch(biểu_thức) {
case gt_1:
        lệnh 1; [ break;]
case gt_2:
        lệnh 2; [ break;]
…
case gt_n:
        lệnh n; [ break;]
default:
        lệnh n+1;
}
```

Sau đây là sơ đồ khối mô tả sự hoạt động của cấu trúc switch dạng đủ

Chú ý:

- biểu_thức phải là một biểu thức có kiểu char, byte, short, int nhưng không thể là kiểu long, nếu biểu_thức có kiểu khác với các kiểu liệt kê ở trên thì java sẽ đưa ra một thông báo lỗi.
- Nếu biểu_thức bằng giá trị của gt_i thì các lệnh từ lệnh i cho đến lệnh n nếu không có default (lệnh n+1 nếu có default) sẽ được thực hiện.
- Câu lệnh break thoát ra khỏi cấu trúc switch.

Sơ đồ khối mô tả sự hoạt động của cấu trúc switch trong trường hợp có lệnh break

1.4 Toán tử điều kiện

Toán tử điều kiện là một loại toán tử đặc biệt vì nó gồm ba thành phần cấu thành biểu thức điều kiện. hay nói cách khác toán tử điều kiện là toán tử 3 ngôi.

Cú pháp :

biểu thức 1? biểu thức 2 : biểu thức 3;

Trong đó

biểu thức 1: Biểu thức 1 là một biểu thức logic. Tức là nó trả trả về giá trị True hoặc False

biểu thức 2: Giá trị trả về nếu biểu thức 1 nhận giá True.

biểu thức 3: Giá trị trả về nếu biểu thức 1 nhận giá trị False

Chú ý: Kiểu giá trị của biểu thức 2 và biểu thức 3 phải tương thích với nhau.

Ví dụ: Đoạn biểu thức điều kiện sau trả về giá trị "a là số chẵn" nếu như giá trị của biến a là số chẵn, ngược lại trả về giá trị "a là số lẻ" nếu như giá trị của biến a là số lẻ.

String result=a%2==0 ? "a là số chẵn" : "a là số lẻ";

II. Cấu trúc lặp while và do-while

1. Lặp kiểm tra điều kiện trước

Ta có thể sử dụng cấu trúc while để thực thi lặp đi lặp lại một lệnh hoặc một khối lệnh trong khi điều kiện đúng

Cú pháp:

while (BooleanExpression) {

statement;

}

ta có thể thấy được luồng thực hiện của chương trình thông qua sơ đồ khối sau:

trước tiên phát biểu while sẽ tính giá trị của biểu thức logic, nếu giá trị của biểu thức logic là đúng thì câu lệnh trong thân của while sẽ được thực hiện, sau khi thực hiện xong nó tính lại giá trị của biểu thức logic, nếu giá trị đúng nó lại tiếp tục thực hiện lệnh trong thân while cho đến khi giá trị của biểu thức sai.

Ví dụ:

```
public class WhileDemo {
public static void main(String[] args) {

String copyFromMe = "Copy this string until you " +
"encounter the letter 'g'.";
StringBuffer copyToMe = new StringBuffer();

int i = 0;
char c = copyFromMe.charAt(i);

while (c! = 'g') {
copyToMe.append(c);
c = copyFromMe.charAt(++i);
}
System.out.println(copyToMe);
}
```

}

Chú ý:

+ biểu thức bên trong cặp ngoặc tròn phải là một biểu thức logic (biểu thức trả về giá trị true hoặc false)

+ biểu thức điều kiện phải nằm trong cặp ngoặc tròn

+ sau từ khoá while ta chỉ có thể đặt được duy nhất một lệnh, do vậy để có thể thực hiện nhiều tác vụ sau while ta phải bao chúng trong một khối lệnh

+ bên trong thân của vòng lặp while ta nên có lệnh làm thay đổi giá trị của biểu thức logic, nếu không chúng ta sẽ rơi vào vòng lặp vô hạn.

+ câu lệnh trong thân cấu trúc while có thể không được thực hiện lần nào (do biểu thức lôgic ban đầu có giá trị false)

2. Lặp kiểm tra điều kiện sau

Cú pháp:

do {

statement(s);

} while (**expression*****);***

sự hoạt động của cấu trúc này được thể hiện qua sơ đồ sau:

Nhìn vào sơ đồ này ta thấy sự hoạt động của nó như sau:

b1) thực hiện lệnh

b2) sau khi thực hiện lệnh xong nó tính giá trị của biểu thức logic

b3) nếu biểu thức logic đúng nó quay trở lại b1, nếu sai thì b4

b4) kết thúc vòng lặp và thực hiện lệnh sau do-while

ví dụ:

public class DoWhileDemo {

```
public static void main(String[] args) {

String copyFromMe = "Copy this string until you " +
"encounter the letter 'g'.";
StringBuffer copyToMe = new StringBuffer();

int i = 0;
char c = copyFromMe.charAt(i);

do {
copyToMe.append(c);
c = copyFromMe.charAt(++i);
} while (c! = 'g');
System.out.println(copyToMe);
}
}
```

Chú ý:

+ biểu thức bên trong cặp ngoặc tròn phải là một biểu thức logic (biểu thức trả về giá trị true hoặc false)
+ biểu thức điều kiện phải nằm trong cặp ngoặc tròn
+ sau từ khoá do ta có thể đặt được nhiều lệnh
+ bên trong thân của vòng lặp do-while ta nên có lệnh làm thay đổi giá trị của biểu thức logic, nếu không chúng ta sẽ rơi vào vòng lặp vô hạn.
+ câu lệnh trong thân cấu trúc do-while được thực hiện ít nhất một lần

III. Cấu trúc for

đây la cấu trúc lặp phổ biến nhất trong các ngôn ngữ lập trình, mà nội dung cuả vòng lặp cần phải lặp đi lặp lại một số lần biết trước, cú pháp của nó như sau:

for (*initialization*; termination*; increment*) {

statement

}

Trong đó:

- initialization là giá trị khởi tạo trước khi vòng lặp bắt đầu, nó chỉ được thực hiện duy nhất một lần trước khi vòng lặp bắt đầu

- termination là điều kiện dùng để kết thúc quá trình lặp

- increment là câu lệnh dùng để điều khiển quá trình lặp

- statement là câu lệnh mà ta cần phải thực hiện lặp đi lặp lại.

Sơ đồ khối diễn giải sự hoạt động của cấu trúc for sau:

Nhận xét:

+ thân của cấu trúc lặp for ta chỉ có thể đặt được duy nhất một lệnh, do vậy để có thể thực hiện nhiều tác vụ trong thân for ta phải bao chúng trong khối lệnh

+ thân vòng lặp for có thể không được thực hiện lần nào

+ các phần initialization, termination, increment có thể khuyết tuy nhiên dấy phẩy dành cho nó vẫn phải có

+ số lần thực hiện initialization=1

+ số lần thực hiện termination = số lần lặp +1

+ số lần thực hiện increment = số lần lặp

+ ta có thể đặt một vài khai báo biến trong phần initialization, như ví dụ sau

+ ta có thể mô tả cấu trúc while thông qua cấu trúc for như sau

for(; Boolean_Expression;) *statement*;

Ví dụ: kiệt kê ra 128 các kí tự asciii đầu tiên

```
public class ListCharacters {
public static void main(String[] args) {
for( char c = 0; c < 128; c++)
```

```
if (c! = 26 )// ANSI Clear screen
System.out.println(
"value: " + (int)c +
" character: " + c);
}
}// /:~
```

Toán tử dẫy và vòng lặp for

Trong bài trước ta đã nhắc đến toán tử dẫy (toán tử dẫy là một dẫy các lệnh đơn được cách nhau bởi dấu phẩy), trong java chỗ duy nhất mà ta có thể đặt toán tử dẫy đó là bên trong cấu trúc lặp for, ta có thể đặt toán tử dẫy cả trong phần initialization lẫn phần increment

Ví dụ về toán tử dẫy

```
public class CommaOperator {
public static void main(String[] args) {
for(int i = 1, j = i + 10; i < 5;
i++, j = i * 2) {
System.out.println("i= " + i + " j= " + j);
}
}
}
```

Kết quả chạy chương trình sau:

```
i= 1 j= 11
i= 2 j= 4
i= 3 j= 6
i= 4 j= 8
```

IV. Lệnh break và continue

Bên trong thân của các cấu trúc lặp ta có thể điều khiển luồng thực hiện bằng cách

sử dụng lệnh break và continue, lệnh break sẽ chấm dứt quá trình lặp mà không thực hiện nốt phân còn lại của cấu trúc lặp, continue sẽ ngưng thực thi phần còn lại của thân vòng lặp và chuyển điều khiển về điểm bắt đầu của vòng lặp, để thực hiện lần lặp tiếp theo, ví dụ sau chỉ ra cách sử dụng break và continue bên trong cấu trúc lặp for và while

```
public class BreakAndContinue {
public static void main(String[] args) {
for(int i = 0; i < 100; i++) {
if(i == 74) break;// Out of for loop
if(i % 9! = 0) continue;// Next iteration
System.out.println(i);
}
int i = 0;
// An "infinite loop":
while(true) {
i++;
int j = i * 27;
if(j == 1269) break;// Out of loop
if(i % 10! = 0) continue;// Top of loop
System.out.println(i);
}
}
}
```

kết quả chạy chương trình sau:

```
0
9
18
27
```

36
45
54
63
72
10
20
30
40

Bên trong cấu trúc lặp for giá trị của i không thể đạt được giá trị 100 vì phát biểu break sẽ kết thúc vòng lặp khi i=74

Chú ý: Java không có lệnh nhẩy goto, tuy nhiên trong java vẫn có một vài vết tích của lệnh nhẩy goto (khét tiếng và được coi là nguồn sinh các lỗi) đó là lệnh break và continue

Nhãn của vòng lặp

Trong thực tế các vòng lặp có thể lồng vào nhau, mức độ lồng nhau không hạn chế, thế thì câu hỏi đặt ra là lệnh break sẽ thoát ra khỏi vòng lặp nào, câu trả lời là nó thoát ra khỏi vòng lặp mà lệnh break được đặt, thế thì làm cách nào ta có thể cho nó thoát ra khỏi một vòng lặp tuỳ ý nào đó, câu trả lời là java đã hỗ trợ cho ta một công cụ đó là nhãn của vòng lặp.

Nhãn là một cái tên sau đó có 2 dấu chấm

Ví dụ LabelName:

Chỗ duy nhất mà nhãn có ý nghĩa đó là ngay trước lệnh lặp, ta không thể có bất cứ một lệnh nào nằm giữa nhãn và lệnh lặp, ta mô tả sự hoạt động, cách sử dụng nhãn của vòng lặp thông qua ví dụ sau:

```
public class LabeledFor {
```

```
 public static void (String[] args) {
int i = 0;
outer:// Can't have statements here
for(; true; ) {// infinite loop
inner:// Can't have statements here
for(; i < 10; i++) {
prt("i = " + i);
if(i == 2) {
prt("continue");
continue;
}
if(i == 3) {
prt("break");
i++;// Otherwise i never
// gets incremented.
break;
}
if(i == 7) {
prt("continue outer");
i++;// Otherwise i never
// gets incremented.
continue outer;
}
if(i == 8) {
prt("break outer");
break outer;
}
for(int k = 0; k < 5; k++) {
if(k == 3) {
```

```
prt("continue inner");
continue inner;
}
}
}
}
// Can't break or continue
// to labels here
}
static void prt(String s) {
System.out.println(s);
}
}
```

kết quả chạy chương trình như sau:

```
i = 0
continue inner
i = 1
continue inner
i = 2
continue
i = 3
break
i = 4
continue inner
i = 5
continue inner
i = 6
```

continue inner

i = 7

continue outer

i = 8

break outer

Chương 2

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Lớp là khái niệm trọng tâm của lập trình hướng đối tượng, java là ngôn ngữ lập trình hướng đối tượng, một chương trình java gồm một tập các đối tượng, các đối tượng này phối hợp với nhau để tạo thành một ứng dụng hoàn chỉnh. Các đối tượng được mô tả qua khái niệm lớp, lớp là sự mở rộng khái niệm RECORD trong pascal, hay struct của C, ngoài các thành phần dữ liệu, lớp còn có các hàm (phương thức, hành vi), ta có thể xem lớp là một kiểu dữ liệu, vì vậy người ta còn gọi lớp là kiểu dữ liệu đối tượng. Sau khi định nghĩa lớp ta có thể tạo ra các đối tượng (bằng cách khai báo biến) của lớp vừa tạo, do vậy có thể quan niệm lớp là tập hợp các đối tượng cùng kiểu.

BÀI 1 ĐỊNH NGHĨA LỚP

I. Khai báo lớp

1.1. Một lớp được định nghĩa theo mẫu sau:

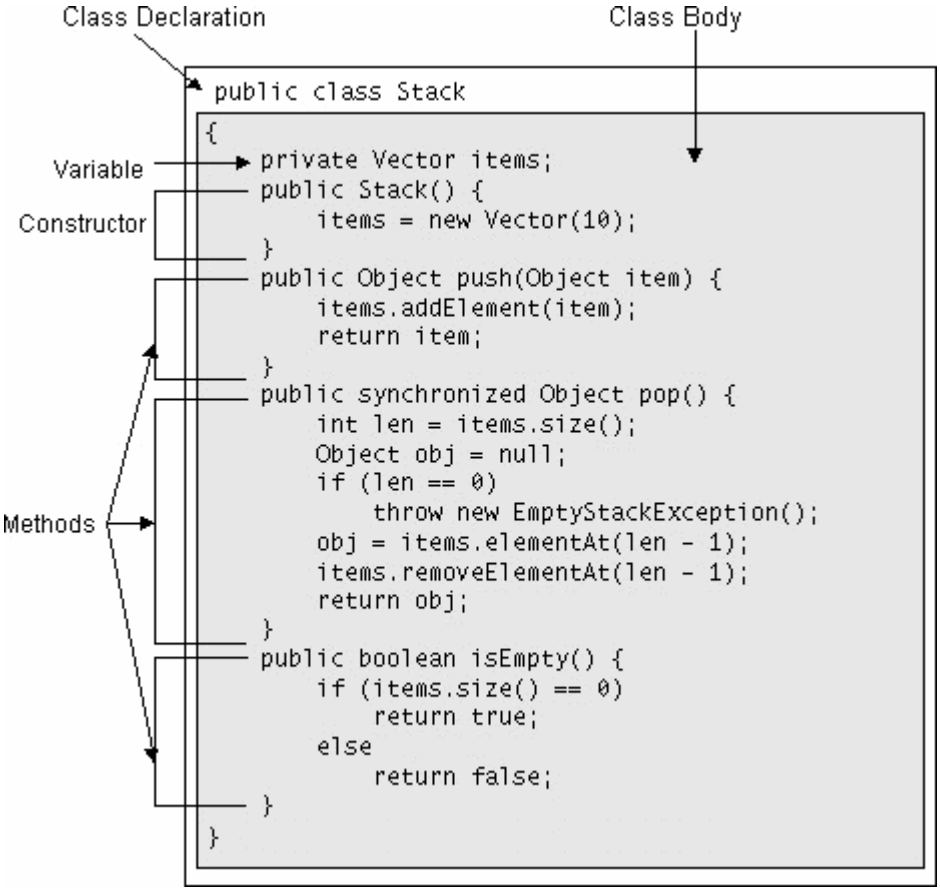
[pbulic][final][abstract] class <tên_lớp>{

// khai báo các thuộc tính

// khai báo các phương thức

}

sau đâu là ví dụ đơn giản định nghĩa lớp ngăn xếp:



Tổng quát: một lớp được khai báo dạng sau:

[public][<abstract><final>][class <Tên lớp>

[***extends*** <Tên lớp cha>] [***implements*** <Tên giao diện>] {

<Các thành phần của lớp, bao gồm: thuộc tính và phương thức>

}

Trong đó:

1) bởi mặc định một lớp chỉ có thể sử dụng bởi một lớp khác trong cùng một gói với lớp đó, nếu muốn gói khác có thể sử dụng lớp này thì lớp này phải được khai báo là lớp ***public***.
2) ***abstract*** là bổ từ cho java biết đây là một lớp trừu tượng, do vậy ta không thể tạo ra một thể hiện của lớp này
3) ***final*** là bổ từ cho java biết đây là một lớp không thể kế thừa
4) ***class*** là từ khoá cho chương trình biết ta đang khai báo một lớp, lớp này có tên là NameOfClass
5) ***extends*** là từ khoá cho java biết lớp này này được kế thừa từ lớp super
6) ***implements*** là từ khoá cho java biết lớp này sẽ triển khai giao diện Interfaces, đây là một dạng tương tự như kế thừa bội của java.

Chú ý:

1) Thuộc tính của lớp là một biến có kiểu dữ liệu bất kỳ, nó có thể lại là một biến có kiểu là chính lớp đó
2) Khi khai báo các thành phần của lớp (thuộc tính và phương thức) có thể dùng một trong các từ khoá ***private, public, protected*** để giứo hạn sự truy cập đến thành phần đó.

   – các thành phần private chỉ có thể sử dụng được ở bên trong lớp, ta không thể truy cập vào các thành phần ***private*** từ bên ngoài lớp

   – Các thành phần ***public*** có thể truy cập được cả bên trong lớp lẫn bên ngoài lớp.

   – các thành phần ***protected*** tương tự như các thành phần ***private***, nhưng có thể

truy cập được từ bất cứ lớp con nào kế thừa từ nó.
– Nếu một thành phần của lớp khi khai báo mà không sử dụng một trong 3 bổ từ ***protected, private, public*** thì sự truy cập là bạn bè, tức là thành phần này có thể truy cập được từ bất cứ lớp nào trong cùng gói với lớp đó.

3) Các thuộc tính nên để mức truy cập ***private*** để đảm bảo tính dấu kín và lúc đó để bên ngoài phạm vi của lớp có thể truy cập được đến thành phần ***private*** này ta phải tạo ra các phương thức phương thức get và set.
4) Các phương thức thường khai báo là public, để chúng có thể truy cập từ bất cứ đâu.
5) Trong một tệp chương trình (hay còn gọi là một đơn vị biên dịch) chỉ có một lớp được khai báo là public, và tên lớp public này phải trùng với tên của tệp kể cả chữ hoa, chữ thường

- Khai báo thuộc tính

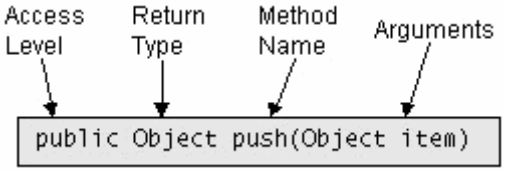
Trở lại lớp Stack

```
public class Stack {
private Vector items;
// a method with same name as a member variable
public Vector items() {
...
}
}
```

Trong lớp Stack trên ta có một thuộc tính được định nghĩa như sau:
private Vector items;

Việc khai báo như trên được gọi là khai báo thuộc tính hay còn gọi là biến thành viên lớp

Tổng quát việc khai báo một thuộc tính được viết theo mẫu sau:



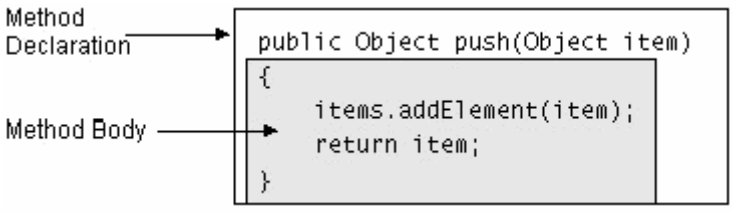
Trong đó:

- *accessLevel có thể là một trong các từ public, private, protected hoặc có thể bỏ* trống, ý nghĩa của các bổ từ này được mô tả ở phần trên
- - *static là từ khoá báo rằng đây là một thuộc tính lớp, nó là một thuộc tính sử* dụng chung cho cả lớp, nó không là của riêng một đối tượng nào.
- - *transient và volatile chưa được dùng*
- - *type là một kiểu dữ liệu nào đó*
- *name là tên của thuộc tính*

Chú ý: Ta phải phân biệt được việc khai báo như thế nào là khai báo thuộc tính, khai báo thế nào là khai báo biến thông thường? Câu trả lời là tất cả các khai báo bên trong thân của một lớp và bên ngoài tất cả các phương thức và hàm tạo thì đó là khai báo thuộc tính, khai báo ở những chỗ khác sẽ cho ta biến.

- Khai báo phương thức

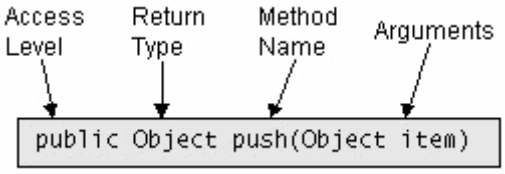
Trong lớp Stack trên ta có phương thức push dùng để đẩy một đối tượng vào đỉnh ngăn xếp, nó được định nghĩa như sau:



Cũng giống như một lớp, một phương thức cũng gồm có 2 phần: phần khai báo và

phần thân

- Phần khai báo gồm có những phần sau(chi tiết của khai báo được mô tả sau):



- Phần thân của phương thức gồm các lệnh để mô tả hành vi của phương thức, các hành vi này được viết bằng các lệnh của java.

II. Chi tiết về khai báo một phương thức

1. Tổng quát một phương thức được khai báo như sau:

accessLevel	//mô tả mức độ truy cập đến phương thức
static	//đây là phương thức lớp
abstract	//đây là phương thức không có cài đặt
final	//phương thức này không thể ghi đè
native	//phương thức này được viết trong một ngôn ngữ khác
synchronized	//đây là phương thức đồng bộ
returnType	//giá trị trả về của phương thức
MethodName	//tên của phương thức
throws exception	//khai báo các ngoại lệ có thể được nem ra từ phương thức

Trong đó:

- accessLevel có thể là một trong các từ khoá public, private, protected hoặc bỏ trống, ý nghĩa của các bổ từ này được mô tả trong phần khai báo lớp
- static là từ khoá báo cho java biết đây là một phương thức lớp
- abstract từ khoá cho biết đây là một lớp trừu tượng, nó không có cài đặt.
- final đây là từ khoá báo cho java biết đây là phương thức không thể ghi đè từ lớp

con

- native đây là từ khoá báo cho java biết phương thức này được viết bằng một ngôn ngữ lập trình nào đó không phải là java (thường được viết bằng C/C++)
- synchronized đây là một phương thức đồng bộ, nó rất hữu ích khi nhiều phương thức cùng truy cập đồng thời vào tài nguyên miền găng
- returnType là một kiểu dữ liệu, đây là kiểu trả về của phương thức, khi phương thức không trả về dữ liệu thì phải dùng từ khoá void
- MethodName là tên của phương thức, tên của phương thức được đặt theo quy tắc đặt tên của java
- throws là từ khoá dùng để khai báo các ngoại lệ có thể được ném ra từ phương thức, theo sau từ khoá này là danh sách các ngoại lệ có thể được phương thức này ném ra

Chú ý:

1) Nếu trong lớp có ít nhất một phương thức trừu tượng thì lớp đó phải là lớp trừu tượng
2) không có thuộc tính trừu tượng
3) ta không thể tạo đối tượng của lớp trừu tượng
4) khác với ngôn ngữ C/C++, java bắt buộc bạn phải khai báo giá trị trả về cho phương thức, nếu phương thức không trả về dữ liệu thi dùng từ khoá void (trong C/C++ khi ta không khai báo giá trị trả về thì mặc định giá trị trả về là int)

2. Nhận giá trị trả về từ phương thức

Ta khai báo kiểu giá trị trả về từ lúc ta khai báo phương thức, bên trong thân của phương thức ta phải sử dụng phát biểu return value; để nhận về kết quả, nếu hàm được khai báo kiểu void thì ta chỉ sử dụng phát biểu return; mệnh đề return đôi khi còn được dùng để kết thúc một phương thức.

3. Truyền tham số cho phương thức

Khi ta viết các phương thức, một số phương thức yêu cầu phải có một số tham số, các tham số của một phương thức được khai báo trong lời khai báo phương thức, chúng phải được khai báo chi tiết có bao nhiêu tham số, mỗi tham số cần phải cung cấp cho chúng một cái tên và kiểu dữ liệu của chúng.

Ví dụ: ta có một phương thức dùng để tính tổng của hai số, phương thức này được khai báo như sau:

```
public double tongHaiSo(double a, double b){
return (a + b);
}
```

1. Kiểu tham số

Trong java ta có thể truyền vào phương thức một tham số có kiểu bất kỳ, từ kiểu dữ liệu nguyên thuỷ cho đến tham chiếu đối tượng.

2. Tên tham số

Khi bạn khai báo một tham số để truyền vào phương thức thì bạn phải cung cấp cho nó một cái tên, tên nay được sử dụng bên trong thân của phương thức để tham chiếu đến tham số được truyền vào.

Chú ý: tên của tham số có thể trùng với tên của thuộc tính, khi đó tên của tham số sẽ "che" đi tên của phương thức, bởi vậy bên trong thân của phương thức mà có tham số có tên trùng với tên của thuộc tính, thì khi nhắc đến cái tên đó có nghĩa là nhắc đến tham số.

3. Truyền tham số theo trị

Khi gọi một phương thức mà tham số của phương thức có kiểu nguyên thuỷ, thì bản sao giá trị của tham số thực sự sẽ được chuyển đến phương thức, đây là đặc tính truyền theo trị (pass- by – value), nghĩa là phương thức không thể thay đổi giá trị của các tham số truyền vào.

Ta kiểm tra điều này qua ví dụ sau:

```
public class TestPassByValue {
public static void test(int t) {
t++;
System.out.println("Gia tri của t bi?n trong ham sau khi tang len 1 la " + t);
}

public static void main(String[] args) {
int t = 10;
System.out.println("Gia tri của t tru?c khi gọi ham = " + t);
test(t);
System.out.println("Gia tri của t truoc khi gọi ham = " + t);
}

}
```

ta se nhận được kết quả ra như sau:
Gia tri của t truoc khi gọi ham = 10
Gia tri của t bên trong ham sau khi tang len 1 la 11
Gia tri của t truoc khi gọi ham = 10

4. Thân của phương thức

Trong ví dụ sau thân của phương thức isEmpty và phương thức pop được in đậm và có mầu đỏ

```
class Stack {
static final int STACK_EMPTY = -1;
Object[] stackelements;
int topelement = STACK_EMPTY;
```

```
...
boolean isEmpty() {
if (topelement == STACK_EMPTY)
return true;
else
return false;
}
Object pop() {
if (topelement == STACK_EMPTY)
return null;
else {
return stackelements[topelement--];
}
}
```

III. Từ khoá this

Thông thường bên trong thân của một phương thức ta có thể tham chiếu đến các thuộc tính của đối tượng đó, tuy nhiên trong một số tình huống đặc biệt như tên của tham số trùng với tên của thuộc tính, lúc đó để chỉ các thành viên của đối tượng đó ta dùng từ khoá this, từ khoá this dùng để chỉ đối tượng này.

Ví dụ sau chỉ ra cho ta thấy trong tình huống này bắt buộc phải dùng từ khoá this vì tên tham số của phương thức tạo dựng lại trùng với tên của thuộc tính

```
class HSBColor {
int hue, saturation, brightness;
HSBColor (int hue, int saturation, int brightness) {
this.hue = hue;
this.saturation = saturation;
this.brightness = brightness;
```

}

IV. Từ khoá super

Khi một lớp được kế thừa từ lớp cha trong cả lớp cha và lớp con đều có một phương thức trùng tên nhau, thế thì làm thế nào có thể gọi phương thức trùng tên đó của lớp cha, java cung cấp cho ta từ khoá *super dùng để chỉ đối tượng của lớp cha*

Ta xét ví dụ sau

```
class ASillyClass {
   boolean aVariable;
   void aMethod() {
            aVariable = true;
   }
}

class ASillierClass extends ASillyClass {
   boolean aVariable;
   void aMethod() {
        aVariable = false;
        super.aMethod();
        System.out.println(aVariable);
        System.out.println(super.aVariable);
   }
}
```

trong ví dụ trên ta thấy trong lớp cha có phương thức tên là aMethod trong lớp con cũng có một phương thức cùng tên, ta còn thấy cả hai lớp này cùng có một thuộc tính tên aVariable để có thể truy cập vào các thành viên của lớp cha ta phải dùng từ khoá super.

Chú ý: ta không thể dùng nhiều từ khoá này để chỉ lớp ông, lớp cụ… chẳng hạn viết như sau là sai: super.super.add(1,4);

V. Sử dụng lớp

Sau khi khao một một lớp ta có thể xem lớp như là một kiểu dữ liệu, nên ta có thể tạo ra các biến, mảng các đối tượng, việc khai báo một biến, mảng các đối tượng cũng tương tự như khai báo một biến, mảng của kiểu dữ liệu nguyên thuỷ

Việc khai báo một biến, mảng được khai báo theo mẫu sau:

Tên_Lớp tên_biến;

Tên_Lớp tên_mang[kích thước mảng];

Tên_Lớp[kích thước mảng] tên_mang;

Về bản chất mỗi đối tượng trong java là một con trỏ tới một vùng nhớ, vùng nhớ này chính là vùng nhớ dùng để lưu trữ các thuộc tính, vùng nhớ dành cho con trỏ này thì được cấp phát trên stack, còn vùng nhớ dành cho các thuộc tính của đối tượng này thì được cấp phát trên heap.

VI. Điều khiển việc truy cập đến các thành viên của một lớp

Khi xây dựng một lớp ta có thể hạn chế sự truy cập đến các thành viên của lớp, từ một đối tượng khác.

Ta tóm tắt qua bảng sau:

Từ khoá	Truy cập trong chính lớp đó	Truy cập trong lớp con cùng gói	Truy cập trong lớp con khác gói	Truy cập trong lớp khác cùng gói	Truy cập trong lớp khác khác gói
private	X	-	-	-	-
protected	X	X	X	X	-
public	X	X	X	X	X

default	X	X	-	X	-

Trong bảng trên thì X thể hiện cho sự truy cập hợp lệ còn – thể hiện không thể truy cập vào thành phần này.

1. Các thành phần private

Các thành viên private chỉ có thể sử dụng bên trong lớp, ta không thể truy cập các thành viên private từ bên ngoài lớp này.

Ví dụ

```
class Alpha
{
private int iamprivate;
private void privateMethod()
{
System.out.println("privateMethod");
}
}

class Beta {
void accessMethod()
{
Alpha a = new Alpha();
a.iamprivate = 10;// không hợp lệ
a.privateMethod();// không hợp lệ
}
}
```

2. Các thành phần protected

Các thành viên protected sẽ được thảo luận trong chương sau

3. Các thành phần public

Các thành viên public có thể truy cập từ bất cứ đâu, ta se xem ví dụ sau:

```
package Greek;

public class Alpha {
public int iampublic;
public void publicMethod() {
System.out.println("publicMethod");
}
}

package Roman;

import Greek.*;

class Beta {
void accessMethod() {
Alpha a = new Alpha();
a.iampublic = 10;// hợp lệ
a.publicMethod();// hợp lệ
}
}
```

4. Các thành phần có mức truy xuất gói

khi ta khai báo các thành viên mà không sử dụng một trong các từ public, private, protected thì java mặc định thành viên đó có mức truy cập gói.

Ví dụ

```
package Greek;

class Alpha {
int iampackage;
void packageMethod() {
System.out.println("packageMethod");
}
}
```

```
package Greek;

class Beta {
void accessMethod() {
Alpha a = new Alpha();
a.iampackage = 10;// legal
a.packageMethod();// legal
}
}
```

BÀI 2 KHỞI ĐẦU VÀ DỌN DẸP

I. Phương thức tạo dựng (constructor)

1. Công dụng

Phương thức tạo dựng là một phương thức của lớp (nhưng khá đặc biệt) thường dùng để khởi tạo một đối tượng mới. Thông thường người ta thường sử dụng hàm tạo để khởi gán giá trị cho các thuộc tính của đối tượng và có thể thực hiện một số công việc cần thiết khác nhằm chuẩn bị cho đối tượng mới.

2. cách viết hàm tạo

a) đặc điểm của phương thức tạo dựng

- hàm tạo có tên trùng với tên của lớp
- hàm tạo không bao giờ trả về kết quả
- nó được java gọi tự động khi một đối tượng của lớp được tạo ra
- hàm tạo có thể có đối số như các phương thức thông thường khác
- trong một lớp có thể có nhiều hàm tạo

b) ví dụ

ví dụ 1: sử dụng hàm tạo để in ra màn hình xâu “Creating Rock”

```
class Rock {
Rock() {// This is the constructor
System.out.println("Creating Rock");
}
}

public class SimpleConstructor {
public static void main(String[] args) {
for(int i = 0; i < 10; i++)
new Rock();// call constructor
}
```

```
    }
```

ví dụ 2: sử dụng hàm tạo có đối

```
class Rock2 {
Rock2(int i) {
System.out.println(
"Creating Rock number " + i);
}
}

public class SimpleConstructor2 {
public static void main(String[] args) {
for(int i = 0; i < 10; i++)
new Rock2(i);// gọi hàm tạo có đối
}
}// /:~
```

3. Hàm tạo mặc định

Khi xây dựng một lớp mà không xây dựng hàm tạo thế thì java sẽ cung cấp cho ta một hàm tạo không đối mặc định, hàm tạo này thực chất không làm gì cả, nếu trong lớp đã có ít nhất một hàm tạo thì hàm tạo mặc định sẽ không được tạo ra, khi ta tạo ra một đối tượng thì sẽ có một hàm tạo nào đó được gọi, nếu trình biên dịch không tìm thấy hàm tạo tương ứng nó sẽ thông báo lỗi, điều này thường xẩy ra khi chúng ta không xây dựng hàm tạo không đối nhưng khi tạo dựng đối tượng ta lại không truyền vào tham số, như được chỉ ra trong ví dụ sau:

```
public class TestPassByValue {
public TestPassByValue(String s) {
```

```
System.out.println(s);
}
public static void main(String[] args) {
TestPassByValue thu = new TestPassByValue();
// lỗi vì lớp này không có hàm tạo không đối
TestPassByValue thu1 = new TestPassByValue("Hello World");
// không vấn đề gì
}
}
```

4. Gọi hàm tạo từ hàm tạo

Khi bạn viết nhiều hàm tạo cho lớp, có đôi lúc bạn muốn gọi một hàm tạo này từ bên trong một hàm tạo khác để tránh phải viết lặp mã. Để có thể gọi đến hàm tạo ta sử dụng từ khoá this.

Cú pháp this(danh sách đối số);

Ví dụ:

```
public class Test {
public Test ()
{
System.out.println("hàm tạo không đối");
}
public Test ( int i)
{
this();// gọi đến hàm tạo không đối của chính lớp này
}


public static void main(String[] args) {
TestPassByValue thu=new TestPassByValue(10);
}
```

}

Chú ý:

1) bên trong cấu tử ta chỉ có thể gọi được tối đa một cấu tử, điều này có nghĩa là ta không thể gọi được từ 2 cấu tử trở lên bên trong một cấu tử khác như được chỉ ra trong ví dụ sau:

```
public class TestPassByValue {
public TestPassByValue() {
System.out.println("Day la ham tao khong doi");
}

public TestPassByValue(int i) {
System.out.println("Day la ham tao doi so nguyen");
}

public TestPassByValue(String s) {
this();// không thể gọi hai hàm tạo trở lên bên trong một hàm tạo
this(10);
System.out.println("Day la ham tao doi so xau");

}

public static void main(String[] args) {
TestPassByValue thu = new TestPassByValue();//
TestPassByValue thu1 = new TestPassByValue("Hello World");//
}
```

}

2) khi gọi một hàm tạo bên trong một hàm tạo khác thì lời gọi hàm tạo phải là lệnh đầu tiên trong thân phương thức, nên ví dụ sau sẽ bị báo lỗi

```
public class Test{
public Test () {
System.out.println("Day la ham tao khong doi");
}
public Test (String s) {
System.out.println("Day la ham tao doi so xau");
this();// gọi đến cấu tử phải là lệnh đầu tiên
}

public static void main(String[] args) {
Test thu = new Test (“Hello World”);
}

}
```

nếu cho dịch ví dụ trên trình biên dịch sẽ phàn nàn

"Test.java": call to this must be first statement in constructor at line 7, column 9

II. Khối khởi đầu vô danh và khối khởi đầu tĩnh

1. Khối vô danh

Trong java ta có thể đặt một khối lệnh không thuộc một phương thức nào, nghĩa là khối này không thuộc bất cứ phương thức nào kể cả hàm tạo. khi đó khối lệnh này được gọi là khối vô danh, khối vô danh này được java gọi thực thi khi một đối tượng được tạo ra, các khối vô danh được gọi trước cả hàm tạo, thông thường ta hay sử dụng khối vô danh để khởi đầu các thuộc tính của lớp hoặc được

sử dụng để khởi tạo cho các thộc tính của một lớp vô danh(vì lớp vô danh không có tên do vậy ta không thể viết hàm tạo cho lớp này, trong trường hợp này khối vô danh là giải pháp duy nhất)

Ví dụ: ở ví dụ này ta có 3 khối vô danh, khi chạy java cho thực thi các khối vô danh này theo thứ tự từ trên xuống dưới

```
public class Untitled1{
// hàm tạo
public Untitled1 (){
        System.out.println ( "Day la ham tao" );
}
// bắt đầu khối vô danh
{
        System.out.println ( "khoi khoi dau thu 3 ");
}// kết thúc khối vô danh


//bắt đầu khối vô danh
{
        System.out.println ( "khoi khoi dau thu 1 ");
}//kết thúc khối vô danh

// bắt đầu khối vô danh
{
        System.out.println ( "khoi khoi dau thu 2 ");
}//kết thúc khối vô danh

public static void main ( String[] args )
```

```
{
        Untitled1 dt1 = new Untitled1 ();
        Untitled1 dt2 = new Untitled1 ();
}

}
```

khi chạy chương trình sẽ cho kết quả sau:

```
khoi khoi dau thu 3
khoi khoi dau thu 1
khoi khoi dau thu 2
Day la ham tao
khoi khoi dau thu 3
khoi khoi dau thu 1
khoi khoi dau thu 2
Day la ham tao
```

2. Khối khởi đầu tĩnh

Khối khởi đầu tĩnh là một khối lệnh bên ngoài tất cả các phương thức, kể cả hàm tạo, trước khối lệnh này ta đặt từ khoá static, từ khoá này báo cho java biết đây là khối khởi đầu tĩnh, khối này chỉ được gọi 1 lần khi đối tượng đầu tiên của lớp này được tạo ra, khối khởi đầu tĩnh này cũng được java gọi tự động trước bất cứ hàm tạo nào, thông thường ta sử dụng khối khởi đầu tĩnh để khởi đầu các thuộc tính tĩnh (static), sau đây là một ví dụ có 1 khối khởi đầu tĩnh và một khối vô danh, để bạn thấy được sự khác nhau giữa khối khởi đầu tĩnh và khối vô danh

```
public class Untitled1
{
```

```
public Untitled1 ()
{
        System.out.println ( "Đây là hàm tạo" );
}

static {// đây là khối khởi đầu tĩnh
        System.out.println ( "Đây là khối khởi đầu tĩnh");
        System.out.println("Khối này chỉ được gọi 1 lần khi thể hiện đầu tiên của lớp được tạo ra");
}

{//đây là khối vô danh
        System.out.println ( "Đây là khối vô danh ");
}


public static void main ( String[] args )
{
        Untitled1 dt1 = new Untitled1 ();/ / tạo ra thể hiện thứ nhất của lớp
        Untitled1 dt2 = new Untitled1 ();/ / tạo tiếp thể hiện thư 2 của lớp
}

}
```

khi cho chạy chương trình ta sẽ được kết quả ra như sau:

```
Đây là khối khởi đầu tĩnh
Khối này chỉ được gọi 1 lần khi thể hiện đầu tiên của lớp được tạo ra
Đây là khối vô danh
```

Đây là hàm tạo
Đây là khối vô danh
Đây là hàm tạo

Nhìn vào kết quả ra ta thấy khối khởi đầu tĩnh chỉ được java gọi thực hiện 1 lần khi đối tượng đầu tiên của lớp này được tạo, còn khối vô danh được gọi mỗi khi một đối tượng mới được tạo ra

III. Dọn dẹp: kết thúc và thu rác

1. Phương thức finalize

Java không có phương thức hủy bỏ. Phương thức finalize tương tự như phương thức hủy bỏ của C++, tuy nhiên nó không phải là phương thức hủy bỏ. Sở dĩ nó không phải là phương thức hủy bỏ vì khi đối tượng được hủy bỏ thì phương thức này chưa chắc đã được gọi đến. Phương thức này được gọi đến chỉ khi bộ thu rác của Java được khởi động và lúc đó đối tượng không còn được sử dụng nữa. Do vậy phương thức finalize có thể không được gọi đến.

2. Cơ chế gom rác của java

Người lập trình C++ thường sử dụng toán tử new để cấp phát động một đối tượng, nhưng lại thường quên gọi toán tử delete để giải phóng vùng nhớ này khi không còn dùng đến nữa, điều này làm rò rỉ bộ nhớ đôi khi dẫn đến chương trình phải kết thúc một cách bất thường, quả thật đâu là một điều tồi tệ. Trong java ta không cần quan tâm đến điều đó, java có một cơ chế thu rác tự động, nó đủ thông minh để biết đối tượng tượng nào không dùng nữa, rồi nó tự động thu hồi vùng nhớ dành cho đối tượng đó.

Trong ngôn ngữ C++ khi một đối tượng bị phá huỷ, sẽ có một hàm được gọi tự động, hàm này được gọi là huỷ tử hay còn gọi là hàm huỷ, thông thường

hàm hàm huỷ mặc định là đủ là đủ để dọn dẹp, tuy nhiên trong một số trường hợp thì hàm huỷ mặc định lại không thể đáp ứng được, do vậy người lập trình C++, phải viết ra hàm huỷ riêng để làm việc đó, tuy nhiên java lại không có khái niệm hàm huỷ hay một cái gì đó tương tự.

BÀI 3 CÁC THÀNH PHẦN TĨNH

I. Thuộc tính tĩnh

Thuộc tính được khai báo với từ khoá static gọi là thuộc tính tĩnh

Ví dụ:

```
class Static{
static int i = 10;// Đây là thuộc tính tĩnh
int j = 10;// Đây là thuộc tính thường
…
}
```

+ Các thuộc tính tĩnh được cấp phát một vùng bộ nhớ cố định, trong java bộ nhớ dành cho các thuộc tính tĩnh chỉ được cấp phát khi lần đầu tiên ta truy cập đến nó.
+ Thành phần tĩnh là chung của cả lớp, nó không là của riêng một đối tượng nào cả.
+ Để truy xuất đến thuộc tính tĩnh ta có thể dùng một trong 2 cách sau:

tên_lớp.tên_thuộc_tính_tĩnh;

tên_đối_tương.tên_thuộc_tính_tĩnh;

cả 2 cách truy xuất trên đều có tác dụng như nhau

+ khởi gán giá trị cho thuộc tính tĩnh

thành phần tĩnh được khởi gán bằng một trong 2 cách sau:

- Sử dụng khối khởi đầu tĩnh (xem lại bài trước)
- Sử dụng khởi đầu trực tiếp khi khai báo như ví dụ trên

Chú ý: ta không thể sử dụng hàm tạo để khởi đầu các thuộc tính tĩnh, bởi vì hàm tạo không phải là phương thức tĩnh.

II. Phương thức tĩnh

Một phương thức được khai báo là static được gọi là phương thức tĩnh

Ví dụ:

```
class Static{
        static int i;// Đây là thuộc tính tĩnh
        // phương thức tĩnh
        static void println (){
                System.out.println ( i );
        }
}
```

+ Phương thức tĩnh là chung cho cả lớp, nó không lệ thuộc vào một đối tượng cụ thể nào
+ Lời gọi phương thức tĩnh xuất phát từ:
    tên của lớp: tên_lớp.tên_phương_thức_tĩnh(tham số);
    tên của đối tượng: tên_đối_tương. tên_phương_thức_tĩnh(tham số);

+ Vì phương thức tĩnh là độc lập với đối tượng do vậy ở bên trong phương thức tĩnh ta không thể truy cập các thành viên không tĩnh của lớp đó, tức là bên trong phương thức tĩnh ta chỉ có thể truy cập đến các thành viên tĩnh mà thôi.
+ Ta không thể sử dụng từ khoá this bên trong phương thức tĩnh

BÀI 4 NẠP CHỒNG PHƯƠNG THỨC

I. Khái niệm về phương thức bội tải

Java cho phép ta xây dựng nhiều phương thức trùng tên nhau, trong cùng một lớp, hiện tượng các phương thức trong một lớp có tên giống nhau được gọi là bội tải phương thức.

II. Yêu cầu của các phương thức bội tải

Do sử dụng chung một cái tên cho nhiều phương thức, nên ta phải cho java biết cần phải gọi phương thức nào để thực hiện, java dựa vào sự khác nhau về số lượng đối cũng như kiểu dữ liệu của các đối này để phân biệt các phương thức trùng tên đó.

Ví dụ:

```
public class OverloadingOrder {
static void print(String s, int i) {
System.out.println(
"String: " + s +
", int: " + i);
}
static void print(int i, String s) {
System.out.println(
"int: " + i +
", String: " + s);
}
public static void main(String[] args) {
print("String first", 11);
print(99, "Int first");
}
}// /:~
```

Chú ý:

1) nếu nếu java không tìm thấy một hàm bội tải thích hợp thì nó sẽ đưa ra một thông báo lỗi
2) ta không thể sử dụng giá trị trả về của hàm để phân biệt sự khác nhau giữa 2 phương thức bội tải
3) không nên quá lạm dụng các phương thức bội tải vì trình biên dịch phải mất thời gian phán đoán để tìm ra hàm thích hợp, điều này đôi khi còn dẫn đến sai sót
4) khi gọi các hàm nạp chồng ta nên có lệnh chuyển kiểu tường minh để trình biên dịch tìm ra hàm phù hợp một cách nhanh nhất
5) trong java không thể định nghĩa chồng toán tử như trong ngôn ngữ C++, có thể đây là một khuyết điểm, nhưng những người thiết kế java cho rằng điều này là không cần thiết, vì nó quá phức tạp.

BÀI 5 KẾ THỪA (INHERITANCE)

I. Lớp cơ sở và lớp dẫn xuất

- Một lớp được xây dựng thông qua kế thừa từ một lớp khác gọi là lớp dẫn xuất (hay còn gọi là lớp con, lớp hậu duệ), lớp dùng để xây dựng lớp dẫn xuất được gọi là lớp cơ sở (hay còn gọi là lớp cha, hoặc lớp tổ tiên)

- Một lớp dẫn xuất ngoài các thành phần của riêng nó, nó còn được kế thừa tất cả các thành phần của lớp cha

II. Cách xây dựng lớp dẫn xuất

Để nói lớp b là dẫn xuất của lớp a ta dùng từ khoá extends, cú pháp như sau:

class b ***extends*** a{

// phần thân của lớp b

}

III. Thừa kế các thuộc tính

Thộc tính của lớp cơ sở được thừa kế trong lớp dẫn xuất, như vậy tập thuộc tính của lớp dẫn xuất sẽ gồm: các thuộc tính khai báo trong lớp dẫn xuất và các thuộc tính của lớp cơ sở, tuy nhiên trong lớp dẫn xuất ta không thể truy cập vào các thành phần private, package của lớp cơ sở

IV. Thừa kế phương thức

Lớp dẫn xuất kế thừa tất cả các phương thức của lớp cơ sở trừ:

- Phương thức tạo dựng
- Phương thức finalize

V. Khởi đầu lớp cơ sở

Lớp dẫn xuất kế thừa mọi thành phần của lớp cơ, điều này dẫn ta đến một hình dung, là lớp dẫn xuất có cùng giao diện với lớp cơ sở và có thể có các thành phần mới bổ sung thêm. nhưng thực tế không phải vậy, kế thừa không chỉ là sao chép giao diện của lớp của lớp cơ sở. Khi ta tạo ra một đối tượng của lớp suy dẫn, thì nó chứa bên trong nó một sự vật con của lớp cơ sở, sự vật con này như thể ta đã tạo ra một sự vật tường minh của lớp cơ sở, thế thì lớp cơ sở phải được bảo đảm khởi đầu đúng, để thực hiện điều đó trọng java ta làm như sau:

Thực hiện khởi đầu cho lớp cơ sở bằng cách gọi cấu tử của lớp cơ sở bên trong cấu tử của lớp dẫn xuất, nếu bạn không làm điều này thì java sẽ làm giúp ban, nghĩa là java luôn tự động thêm lời gọi cấu tử của lớp cơ sở vào cấu tử của lớp dẫn xuất nếu như ta quên làm điều đó, để có thể gọi cấu tử của lớp cơ sở ta sử dụng từ khoá super

Ví dụ 1: ví dụ này không gọi cấu tử của lớp cơ sở một cách tường minh

class B

```
{
public B ()
{
        System.out.println ( "Ham tao của lop co so" );
}
}

public class A
extends B
{
public A ()
{// không gọi hàm tạo của lớp cơ sở tường minh
        System.out.println ( "Ham tao của lop dan xuat" );
}

public static void main ( String arg[] )
{
        A thu = new A ();
}
}
```

Kết quả chạy chương trình như sau:

```
Ham tao của lop co so
Ham tao của lop dan xuat
```

Ví dụ 2: ví dụ này sử dụng từ khoá super để gọi cấu tử của lớp cơ sở một cách tường minh

```
class B
{
```

```
public B ()
{
        System.out.println ( "Ham tao của lop co so" );
}
}

public class A
extends B
{
public A ()
{
        super();// gọi tạo của lớp cơ sở một cách tường minh
        System.out.println ( "Ham tao của lop dan xuat" );
}

public static void main ( String arg[] )
{
        A thu = new A ();
}
}
```

khi chạy chưng trình ta thấy kết quả giống hệt như ví dụ trên

Chú ý 1: nếu gọi tường minh cấu tử của lớp cơ sở, thì lời gọi này phải là lệnh đầu tiên, nếu ví dụ trên đổi thành

```
class B
{
public B ()
{
        System.out.println ( "Ham tao của lop co so" );
}
```

```
}

public class A
extends B
{
public A ()
{// Lời gọi cấu tử của lớp cơ sở không phải là lệnh đầu tiên
        System.out.println ("Ham tao của lop dan xuat");
        super ();

}

public static void main ( String arg[] )
{
        A thu = new A ();
}
}
```

nếu biên dịch đoạn mã này ta sẽ nhân được một thông báo lỗi như sau: "A.java": call to super must be first statement in constructor at line 15, column 15

Chú ý 2: ta chỉ có thể gọi đến một hàm tạo của lớp cơ sở bên trong hàm tạo của lớp dẫn xuất, ví dụ chỉ ra sau đã bị báo lỗi

```
class B
{
public B ()
{
        System.out.println ( "Ham tao của lop co so" );
}
```

```
public B ( int i )
{
        System.out.println ( "Ham tao của lop co so" );
}

}

public class A
extends B
{
public A ()
{
        super ();
        super ( 10 );/ / không thể gọi nhiều hơn 1 hàm tạo của lớp cơ sở
        System.out.println ( "Ham tao của lop dan xuat" );

}

public static void main ( String arg[] )
{
        A thu = new A ();
}
}
```

1. Trật tự khởi đầu

Trật tự khởi đầu trong java được thực hiện theo nguyên tắc sau: java sẽ gọi cấu tử của lớp cơ sở trước sau đó mới đến cấu tử của lớp suy dẫn, điều này có nghĩa là trong cây phả hệ thì các cấu tử sẽ được gọi theo trật tự từ gốc xuống dần đến lá

2. Trật tự dọn dẹp

Mặc dù java không có khái niệm huỷ tử như của C++, tuy nhiên bộ thu rác của java vẫn hoạt động theo nguyên tắc làm việc của cấu tử C++, tức là trật tự thu rác thì ngược lại so với trật tự khởi đầu.

VI. Ghi đè phương thức (Override)

Hiện tượng trong lớp cơ sở và lớp dẫn xuất có hai phương thức giống hệt nhau (cả tên lẫn bộ tham số) gọi là ghi đè phương thức (Override), chú ý Override khác Overload.

Gọi phương thức bị ghi đè của lớp cơ sở

Bên trong lớp dẫn xuất, nếu có hiện tượng ghi đè thì phương thức bị ghi đè của lớp cơ sở sẽ bị ẩn đi, để có thể gọi phương thức bị ghi đè của lớp cơ sở ta dùng từ khoá super để truy cập đến lớp cha, cú pháp sau:

super.*overriddenMethodName*();

Chú ý: Nếu một phương thức của lớp cơ sở bị bội tải (Overload), thì nó không thể bị ghi đè (Override) ở lớp dẫn xuất.

VI. Thành phần protected

Trong một vài bài trước ta đã làm quen với các thành phần private, public, sau khi đã học về kế thừa thì từ khoá protected cuối cùng đã có ý nghĩa.

Từ khoá protected báo cho java biết đây là thành phần riêng tư đối với bên ngoài nhưng lại sẵn sàng với các con cháu

VII. Từ khoá final

Từ khoá final trong java có nhiều nghĩa khác nhau, nghĩa của nó tuỳ thuộc vào ngữ cảnh cụ thể, nhưng nói chung nó muốn nói “cái này không thể thay đổi

được".

1. Thuộc tính final

Trong java cách duy nhất để tạo ra một hằng là khai báo thuộc tính là final
Ví dụ:

```
public class A
{
// định nghĩa hằng tên MAX_VALUE giá trị 100
static final int MAX_VALUE = 100;
public static void main ( String arg[] )
{
        A thu = new A ();
        System.out.println("MAX_VALUE= " +thu.MAX_VALUE);
}
}
```

Chú ý:

1) khi đã khai báo một thuộc tính là final thì thuộc tính này la hăng, do vậy ta không thể thay đổi giá trị của nó
2) khi khai báo một thuộc tính là final thì ta phải cung cấp giá trị ban đầu cho nó
3) nếu một thuộc tính vừa là final vừa là static thì nó chỉ có một vùng nhớ chung duy nhất cho cả lớp

2. Đối số final

Java cho phép ta tạo ra các đối final bằng việc khai báo chúng như vậy bên trong danh sách đối, nghĩa là bên trong thân của phương pháp này, bất cứ cố gắng nào để thay đổi giá trị của đối đều gây ra lỗi lúc dịch
Ví dụ sau bị báo lỗi lúc dịch vì nó cố gắng thay đổi giá trị của đối final
public class A

```
{
static public void thu ( final int i )
{
        i=i+1;//không cho phép thay đổi giá trị của tham số final
        System.out.println ( i );;
}

public static void main ( String arg[] )
{
        int i = 100;
        thu ( i );

}
}
```

chương trình này sẽ bị báo lỗi:

"A.java": variable i might already have been assigned to at line 5, column 9

3. Phương thức final

Một phương thức bình thường có thể bị ghi đè ở lớp dẫn xuất, đôi khi ta không muốn phương thức của ta bị ghi đè ở lớp dẫn xuất vì lý do gì đó, mục đích chủ yếu của các phương thức final là tránh ghi đè, tuy nhiên ta thấy rằng các phương thức private sẽ tự động là final vì chúng không thể thấy được trong lớp dẫn xuất lên chúng không thể bị ghi đè, nên cho dù bạn có cho một phương thức private là final thì bạn cũng chả thấy một hiệu ứng nào

4. Lớp final

Nếu bạn không muốn người khác kế thừa từ lớp của bạn, thì bạn hãy dùng từ khoá final để ngăn cản bất cứ ai muốn kế thừa từ lớp này.

Chú ý: do một lớp là final (tức không thể kế thừa)do vậy ta không thể nào ghi

đè các phương thức của lớp này, do vậy đừng cố gắng cho một phương thức của lớp final là final

BÀI 6 LỚP CƠ SỞ TRỪU TƯỢNG

Một lớp cơ sở trừu tượng là một lớp chỉ được dùng làm cơ sở cho các lớp khác, ta không thể tạo ra thể hiện của lớp này, bởi vì nó được dùng để định nghĩa một giao diện chung cho các lớp khác.

Phương thức trừu tượng

Một lớp trừu tượng có thể chứa một vài phương thức trừu tượng, do lớp trừu tượng chỉ làm lớp cơ sở cho các lớp khác, do vậy các phương thức trừu tượng cũng không được cài đặt cụ thể, chúng chỉ gồm có khai báo, việc cài đặt cụ thể sẽ dành cho lớp con

1. Chú ý:

1) nếu trong lớp có phương thức trừu tượng thì lớp đó phải được khai báo là trừu tượng
2) nếu một lớp kế thừa từ lớp trừu tượng thì: hoặc chúng phải ghi đè tất cả các phương thức ảo của lớp cha, hoặc lớp đó phải là lớp trừu tượng
3) không thể tạo ra đối tượng của lớp trừu tượng

BÀI 7 ĐA HÌNH THÁI

Đa hình thái trong lập trình hướng đối tượng đề cập đến khả năng quyết định trong lúc thi hành (runtime) mã nào sẽ được chạy, khi có nhiều phương thức trùng tên nhau nhưng ở các lớp có cấp bậc khác nhau.

Chú ý: khả năng đa hình thái trong lập trình hướng đối tượng còn được gọi với nhiều cái tên khác nhau như: tương ứng bội, kết ghép động,..

Đa hình thái cho phép các vấn đề khác nhau, các đối tượng khác nhau, các phương thức khác nhau, các cách giải quyết khác nhau theo cùng một lược đồ chung.

Các bước để tạo đa hình thái:

1. Xây dựng lớp cơ sở (thường là lớp cơ sở trừu tượng, hoặc là một giao diện), lớp này sẽ được các lớp con mở rộng(đối với lớp thường, hoặc lớp trừu tượng), hoặc triển khai chi tiết (đối với giao diện).
2. 2. Xây dựng các lớp dẫn xuất từ lớp cơ sở vừa tạo. trong lớp dẫn xuất này ta sẽ ghi đè các phương thức của lớp cơ sở(đối với lớp cơ sở thường), hoặc triển khai chi tiết nó (đối với lớp cơ sở trừu tượng hoặc giao diện).
3. Thực hiện việc tạo khuôn xuống, thông qua lớp cơ sở, để thực hiện hành vi đa hình thái

Khái niệm về tạo khuôn lên, tạo khuôn xuống

- Hiện tượng một đối tượng của lớp cha tham trỏ đến một đối tượng của lớp con thì được gọi là tạo khuôn xuống, việc tạo khuôn xuống luôn được java chấp thuận, do vậy khi tạo khuôn xuống ta không cần phải ép kiểu tường minh.
- Hiện tượng một đối tượng của lớp con tham trỏ tới một đối tượng của lớp cha thì được gọi là tạo khuôn lên, việc tạo khuôn lên là an toàn, vì một đối tượng của lớp con cũng có đầy đủ các thành phần của lớp cha, tuy nhiên việc tạo khuôn lên sẽ bị báo lỗi nếu như ta không ép kiểu một cách tường minh.

BÀI 8 GIAO DIỆN, LỚP TRONG, GÓI

Giao diện là một khái niệm được java đưa ra với 2 mục đích chính:

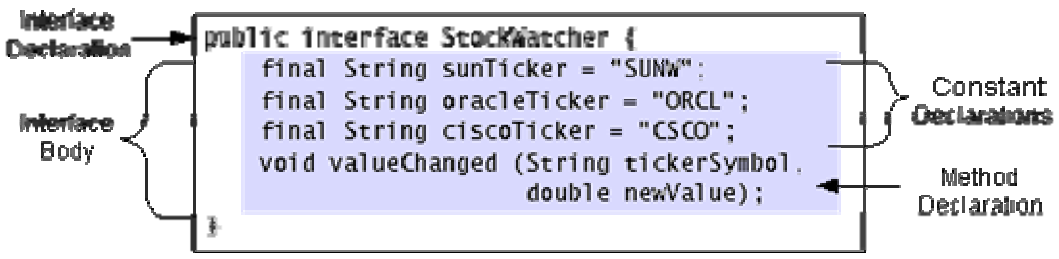
- Để tạo ra một lớp cơ sở thuần ảo, một lớp không có bất cứ hàm nào được cài đặt
- Thực hiện hành vi tương tự như kế thừa bội, bởi trong java không có khái niệm kế thừa bội, như của C++

Lớp trong cho ta một cách thức tinh vi để che giấu mã một cách tối đa, trong java ta có thể định nghĩa một lớp bên trong một lớp khác, thậm chí ta còn có thể tạo lớp trong, bên trong thân của một phương thức, điều này cho phép ta có thể tạo ra các lớp cục bộ, chỉ được sử dụng nội bộ bên trong một đơn vị đó. Ta không thể tạo ra một lớp trong, trong ngôn ngữ C++

I. Giao diện

Từ khoá interface đã đưa khái niệm abstract đi xa thêm một bước nữa. Ta có thể nghĩ nó như là một lớp abstract "thuần tuý", nó cho phép ta tạo ra một lớp thuần ảo, lớp này chỉ gồm tập các giao diện cho các lớp muốn dẫn xuất từ nó, một interface cũng có thể có các trường, tuy nhiên java tự động làm các trường này thành *static và final*

Để tạo ra một interface, ta dùng từ khoá interface thay vì từ khoá class. Một interface gồm có 2 phần: phần khai báo và phần thân, phần khai báo cho biết một số thông tin như: tên của interface, nó có kế thừa từ một giao diện khác hay không. Phần thân chứa các khai báo hằng, khai báo phương thức (nhưng không có cài đặt). Giống như một lớp ta cũng có thể thêm bổ từ public vào trước định nghĩa của interface. Sau đây là hình ảnh của một interface.



Nhưng do java tự động làm các trường thành final nên ta không cần thêm bổ từ này, do vậy ta có thể định nghĩa lại giao diện như sau:

Nhưng do java tự động làm các trường thành final nên ta không cần thêm bổ từ này

```
public interface StockWatcher
{
final String
sunTicker = "SUNW";
final String oracleTicker = "ORCL";
final String ciscoTicker = "CSCO";
void valueChanged(String tickerSymbol, double newValue);
}
```

1. Phần khai báo của giao diện

Tổng quát phần khai báo của một giao diện có cấu trúc tổng quát như sau:

Public	//giao diện này là công cộng
interface InterfaceName	//tên của giao diện
Extends SuperInterface	//giao diện này là mở rộng của 1 giao diện khác

{

InterfaceBody //thân của giao diện

}

Trong cấu trúc trên có 2 phần bắt buộc phải có đó là phần interface và *InterfaceName, các phần khác là tuỳ chọn.*

2. Phần thân

Phần thân khai báo các các hằng, các phương thức rỗng (không có cài đặt), các phương thức này phải kết thúc với dấu chấm phẩy ';', bởi vì chúng không có phần cài đặt

Chú ý:

1) Tất cả các thành phần của một giao diện tự động là public do vậy ta không cần phải cho bổ từ này vào.
2) Java yêu cầu tất cả các thành phần của giao diện phải là public, nếu ta thêm các bổ từ khác như private, protected trước các khai báo thì ta sẽ nhận được một lỗi lúc dịch
3) Tất cả các trường tự động là final và static, nên ta không cần phải cho bổ từ này vào.

3. Triển khai giao diện

Bởi một giao diện chỉ gồm các mô tả chúng không có phần cài đặt, các giao diện được định nghĩa để cho các lớp dẫn xuất triển khai, do vậy các lớp dẫn xuất từ lớp này phải triển khai đầy đủ tất cả các khai báo bên trong giao diện, để triển khai một giao diện bạn bao gồm từ khoá implements vào phần khai báo lớp, lớp của bạn có thể triển khai một hoặc nhiều giao diện (hình thức này tương tự như kế thừa bội của C++)

Ví dụ

public class StockApplet extends Applet **implements StockWatcher** {

```
.. .

public void valueChanged(String tickerSymbol, double newValue) {
if (tickerSymbol.equals(sunTicker)) {
.. .
} else if (tickerSymbol.equals(oracleTicker)) {
.. .
} else if (tickerSymbol.equals(ciscoTicker)) {
.. .
}
}
}
```

Chú ý:

1) Nếu một lớp triển khai nhiều giao diện thì các giao diện này được liệt kê cách nhau bởi dấu phẩy ‘,’
2) Lớp triển khai giao diện phải thực thi tất cả các phương thức được khai báo trong giao diện, nếu như lớp đó không triển khai, hoặc triển khai không hết thì nó phải được khai báo là abstract
3) Do giao diện cũng là một lớp trừu tượng do vậy ta không thể tạo thể hiện của giao diện
4) Một lớp có thể triển khai nhiều giao diện, do vậy ta có lợi dụng điều này để thực hiện hành vi kế thừa bội, vốn không được java hỗ trợ
5) Một giao diện có thể mở rộng một giao diện khác, bằng hình thức kế thừa

II. Lớp trong

Có thể đặt một định nghĩa lớp này vào bên trong một lớp khác. điều này được gọi là lớp trong. Lớp trong là một tính năng có giá trị vì nó cho phép bạn gộp nhóm các lớp về mặt logic thuộc về nhau và để kiểm soát tính thấy được của các

lớp này bên trong lớp khác. Tuy nhiên bạn phải hiểu rằng lớp trong không phải là là hợp thành

Ví dụ:

```
public class Stack {
private Vector items;

.. .//code for Stack's methods and constructors not shown...

public Enumeration enumerator() {
return new StackEnum();
}

class StackEnum implements Enumeration {
int currentItem = items.size() - 1;
public boolean hasMoreElements() {
return (currentItem >= 0);
}
public Object nextElement() {
if (!hasMoreElements())
throw new NoSuchElementException();
else
return items.elementAt(currentItem--);
}
}
}
```

Lớp trong rất hữu hiệu khi bạn bạn muốn tạo ra các lớp điều hợp (được bàn kỹ khi nói về thiết kế giao diện người dùng)

Bài 9 MẢNG, XÂU KÝ TỰ, TẬP HỢP

I. Mảng

1. Mảng 1 chiều

a) Khai báo

Cú pháp khai báo:

- KDL tên_mảng[];//Khai báo một con trỏ mảng

- KDL []tên_mảng;//như trên

- KDL tên_mảng[] = new KDL[spt];//Tạo ra một mảng có spt phần tử

Trong cú pháp trên thì:

- KDL là một kiểu dữ liệu bất kỳ như: kiểu nguyên thủy, kiểu đối tượng… nó xác định kiểu dữ liệu của từng phần tử của mảng.
- Spt là số phần tử của mảng.

Chú ý:

- Mảng trong Java là một đối tượng
- Cũng như các đối tượng khác, mảng phải được tạo ra bằng toán tử ***new*** như sau:

  Tên_mảng=new KDL[spt];
- Khi mảng được tạo ra thì mỗi phần tử của mảng sẽ nhận một giá trị mặc định, quy tắc khởi tạo giá trị cho các phần tử của mảng cũng chính là quy tắc khởi đầu giá trị cho các thuộc tính của đối tượng, tức là mỗi phần tử của mảng sẽ nhận giá trị:

  + 0 nếu KDL là kiểu số

  + '\0' nếu KDL là kí tự

  + false nếu KDL là boolean

  + null nếu KDL là một lớp nào đó.

Ví dụ 1. Khai báo một mảng số nguyên gồm 100 phần tử

Cách 1:

```
int mangInt[];//Khai báo một con trỏ đến mảng các số nguyên
```

mangInt=new int[100];//Tạo ra mảng

Cách 2:

int mangInt[]=new int[100];

Ví dụ 2: Giả sử ta có lớp SinhVien đã được định nghĩa, hãy khai báo một mảng gồm 100 đối tượng của lớp SinhVien

SinhVien arraySinhVien[]=new SinhVien[100];

Chú ý: Lúc này mỗi phần tử của mảng arraySinhVien là một con trỏ của lớp SinhVien và hiện giờ mỗi phần tử của mảng đang trỏ đến giá trị null. Để khởi tạo từng phần tử của mảng ta phải làm như sau:

arraySinhVien[0]=new SinhVien("sv01", "Nguyễn Văn An", "Hưng Yên");

arraySinhVien[1]=new SinhVien("sv02", "Nguyễn Thị Bình", "Bắc Giang");

….

arraySinhVien[99]=new SinhVien("sv100", "Đào Thị Mến", "Hà Nam");

Ngoài cách khai báo trên Java còn cho phép ta kết hợp cả khai báo và khởi gán các phần tử của mảng theo cách sau:

int[] mangInt = {1, 3, 5, 7, 9};

Tạo ra một mảng gồm 5 phần tử, các phần tử của mảng lần lượt được gán các giá trị là: 1, 3, 5, 7, 9

```
SinhVien[] mangSinhVien  = {
                        new SinhVien("sv01", "Nguyễn Văn A",
                "HY"),
                        new SinhVien("sv02", "Nguyễn Thị B", "HN"),
                        new SinhVien("sv03", "Đỗ Thị Q", "BG"),
                        null
                        };
```

Khai báo một mảng gồm 4 phần tử, giá trị của các phần tử lần lượt được khởi gán như sau:

mangSinhVien [0]=new SinhVien(“sv01”, “Nguyễn Văn A”, “HY”)

mangSinhVien [1]=new SinhVien(“sv02”, “Nguyễn Thị B”, “HN”)

mangSinhVien [2]=new SinhVien(“sv03”, “Đỗ Thị Q”, “BG”)

mangSinhVien [3]=null

b) Truy xất đến các phần tử của mảng một chiều

Để truy xuất đến phần tử thứ ind của mảng ta sử dụng cú pháp như sau:

Tên_mảng[ind-1]

Chú ý: Phần tử đầu tiên của mảng có chỉ số là 0.

Ví dụ:

int a[]=new int [3];//Khai báo và tạo ra mảng gồm 3 phần tử

Lúc này các phần tử của mảng lần lượt được truy xuất như sau:

- Phần tử đầu tiên của mảng là a[0]
- Phần tử thứ 2 của mảng là a[1]
- Phần tử thứ 3 đồng thời là phần tử cuối cùng của mảng là a[2]

c) Lấy về số phần tử hiện tại của mảng

Mảng trong Java là một đối tượng, do vậy nó cũng có các thuộc tính và các phương thức như các đối tượng khác. Để lấy về số phần tử của mảng ta sử dụng thuộc tính length như sau:

Tên_mảng.length

Ví dụ 1: Nhập vào một mảng và in ra màn hình

```
import com.theht.Keyboard;
class ArrayDemo{
   public static void main(String[] args) {
      //Nhập số phần tử của mảng
      System.out.print("Nhập số phần tử của mảng:");
      int n=Keyboard.readInt();
      //Khai báo mảng với số phần tử bằng n
```

```
        int a[]=new int[n];
        //Nhập dữ liệu cho mảng
        for(int i=0;i<a.length;i++){
           System.out.print("a[" + i + "]=");
           a[i]=Keyboard.readInt();
        }

        //In mảng ra màn hình
        System.out.println("Mảng vừa nhập là");
        for (int i = 0; i < a.length; i++)
           System.out.print(a[i] + "   ");

     }
  }
```

Ví dụ 2: Nhập vào một mảng số thực sau đó kiểm tra xem mảng có phải là một dãy tăng hay không?

```
  import com.theht.Keyboard;
  class ArrayDemo2{
     public static void main(String[] args) {
        //Nhập số phần tử của mảng
        System.out.print("Nhập số phần tử của mảng:");
        int n=Keyboard.readInt();
        //Khai báo mảng với số phần tử bằng n
        int a[]=new int[n];
        //Nhập dữ liệu cho mảng
        for(int i=0;i<a.length;i++){
           System.out.print("a[" + i + "]=");
           a[i]=Keyboard.readInt();
        }
```

```
        //Kiểm tra dãy tăng
        boolean kt=true;
        for (int i = 0; i < a.length-1; i++)
            if(a[i+1]-a[i]<0){
                kt=false;//thay đổi trạng thái cờ
                break;//Thoát khỏi vòng lặp
            }
        if(kt)
            System.out.println("Dãy tăng dần");
        else
            System.out.println("Dãy không phải tăng dần");
    }
}
```

2. Mảng nhiều chiều

a) Khai báo

Khai báo mảng N chiều trong Java được tiến hành như sau:

hoặc

hoặc

Trong đó:

- KDL là một kiểu dữ liệu bất kỳ: nguyên thuỷ hoặc lớp
- sp1, sp2, …, sptN lần lượt là số phần tử trên chiều thứ 1, 2, .., N

Ví dụ:

- Khai báo một con trỏ của mảng 2 chiều

```
int[][] a; hoặc int a[][];
```

- Khai báo và tạo ra mảng 2 chiều:

```
int[][] a = new int[2][3];  // Ma trận gồm 2 hàng, 3 cột
```

- Khai báo và khởi gán giá trị cho các phần tử của mảng 2 chiều:

```
int a[][]={
            {1, 2, 5}. //Các phần tử trên hàng thứ nhất
            {2, 4, 7, 9}. //Các phần tử trên hàng thứ hai
            {1, 7}. //Các phần tử trên hàng thứ ba
        }
```

Khai báo trên sẽ tạo ra một mảng hai chiều gồm: 3 hàng, nhưng trên mỗi hàng lại có số phần tử khác nhau, cụ thể là: trên hàng thứ nhất có 3 phần tử, hảng 2 gồm 4 phần tử và hàng thứ 3 gồm 2 phần tử.

Chú ý: Với khai báo trên nếu ta liệt kê các phần tử của mảng theo trình tự từ trái qua phải và từ trên xuống dưới thì các phần tử lần lượt là:

a[0][0], a[0][1], a[0][2], a[1][0], a[1][1], a[1][2], a[1][3], a[2][0], a[2][1]

b) Truy xuất đến phần tử mảng nhiều chiều

tên_mảng[ind1][ind2]

Ví dụ 1: Nhập vào một ma trận và in ra màn hình

```
import com.theht.Keyboard;

class MaTram {
   public static void main(String[] args) {
      //Nhập số hàng và số cột
      System.out.print("Nhập số hàng:");
      int sh = Keyboard.readInt();
      System.out.print("Nhập số cột:");
      int sc = Keyboard.readInt();

      //Khai báo mảng hai chiều gồm sh hàng và sc cột
      float a[][] = new float[sh][sc];
```

```
        //Nhập dữ liệu cho mảng hai chiều
        for (int i = 0; i < a.length; i++)
            for (int j = 0; j < a[i].length; j++) {
                System.out.print("a[" + i + "," + j + "]=");
                //Nhập liệu cho phần tử hàng i, cột j
                a[i][j] = Keyboard.readFloat();
            }
        //In mảng hai chiều ra màn hình
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[i].length; j++)
                System.out.print(a[i][j] + "   ");
            System.out.println();
        }

    }
}
```

Ví dụ 2: Nhập vào ma trận vuông sau đó tính tổng các phần tử trên đường chéo chính.

II. Xâu ký tự

Việc xử lý các xâu ký tự trong Java được hỗ trợ bởi hai lớp ***String*** và ***StringBuffer***. *Lớp* ***String*** *dùng cho những xâu ký tự bất biến, nghĩa là* những xâu chỉ đọc và sau khi dược khởi tạo giá trị thì nội dung bên trong xâu không thể thay đổi được. Lớp ***StringBuffer*** được sử dụng đối với những xâu ký tự động, tức là có thể thay đổi được nội dung bên trong của xâu.

1. Lớp String

Chuỗi là một dãy các ký tự. Lớp String cung cấp các phương thức để thao tác với các chuỗi. Nó cung cấp các phương thức khởi tạo (constructor) khác nhau:

String str1 = new String( );

//str1 chứa một chuỗi rống.

String str2 = new String("Hello World");

//str2 chứa "Hello World"

char ch[] = {'A','B','C','D','E'};
String str3 = new String(ch);

//str3 chứa "ABCDE"

String str4 = new String(ch,0,2);

//str4 chứa "AB" vì 0- tính từ ký tự bắt đầu, 2- là số lượng ký tự kể từ ký tự bắt đầu.

Toán tử "+" được sử dụng để cộng chuỗi khác vào chuỗi đang tồn tại. Toán tử "+" này được gọi như là "nối chuỗi". Ở đây, nối chuỗi được thực hiện thông qua lớp "StringBuffer". Chúng ta sẽ thảo luận về lớp này trong phần sau. Phương thức "concat( )" của lớp String cũng có thể thực hiện việc nối chuỗi. Không giống như toán tử "+", phương thức này không thường xuyên nối hai chuỗi tại vị trí cuối cùng của chuỗi đầu tiên. Thay vào đó, phương thức này trả về một chuỗi mới, chuỗi mới đó sẽ chứa giá trị của cả hai. Điều này có thể được gán cho chuỗi đang tồn tại. Ví dụ:

String strFirst, strSecond, strFinal;

StrFirst = "Charlie";

StrSecond = "Chaplin";

//....bằng cách sử dụng phương thức concat( ) để gán với một chuỗi đang tồn tại.

StrFinal = strFirst.concat(strSecond);

Phương thức concat( ) chỉ làm việc với hai chuỗi tại một thời điểm.

Các phương thức của lớp String

Trong phần này, chúng ta sẽ xem xét các phương thức của lớp String.

- char charAt(int index) *Phương thức này trả về một ký tự tại vị trí index trong chuỗi.*

Ví dụ:

String name = new String("Java Language");

char ch = name.charAt(5);

Biến "ch" chứa giá trị "L", từ đó vị trí các số bắt đầu từ 0.

- boolean startsWith(String s) *Phương thức này trả về giá trị kiểu logic (Boolean), phụ thuộc vào chuỗi có bắt đầu với một chuỗi con cụ thể nào đó không.*

Ví dụ:

String strname = "Java Language";

boolean flag = strname.startsWith("Java");

Biến "flag" chứa giá trị true.

- boolean endsWith(String s) *Phương thức này trả về một giá trị kiểu logic (boolean), phụ thuộc vào chuỗi kết thúc bằng một chuỗi con nào đó không.*

Ví dụ:

String strname = "Java Language";
boolean flag = strname.endsWith("Java");

Biến "flag" chứa giá trị false.

- String copyValueOf( )

Phương thức này trả về một chuỗi được rút ra từ một mảng ký tự được truyền như một đối số. Phương thức này cũng lấy hai tham số nguyên. Tham số đầu tiên chỉ định vị trí từ nơi các ký tự phải được rút ra, và tham số thứ hai chỉ định số ký tự được rút ra từ mảng. Ví dụ:

char name[] = {'L','a','n','g','u','a','g','e'};

String subname = String .copyValueOf(name,5,2);

Bây giờ biến "subname" chứa chuỗi "ag".

- char [] toCharArray( )

Phương thức này chuyển chuỗi thành một mảng ký tự. Ví dụ:

String text = new String("Hello World");

char textArray[] = text.toCharArray( );

- int indexOf(String sunString)

Phương thức này trả về thứ tự của một ký tự nào đó, hoặc một chuỗi trong phạm vi một chuỗi. Các câu lệnh sau biểu diễn các cách khác nhau của việc sử dụng hàm.

String day = new String("Sunday");

int index1 = day.indexOf('n');

//chứa 2

int index2 = day.indexOf('z',2);

//chứa –1 nếu "z" không tìm thấy tại vị trí 2.

int index3 = day.indexOf("Sun");

//chứa mục 0

- String toUpperCase( )

Phương thức này trả về chữ hoa của chuỗi.

String lower = new String("good morning");

System.out.println("Uppercase: "+lower.toUpperCase( ));

- String toLowerCase( )

Phương thức này trả về chữ thường của chuỗi.

String upper = new String("JAVA");

System.out.println("Lowercase: "+upper.toLowerCase( ));

- String trim()

Phương thức này cắt bỏ khoảng trắng hai đầu chuỗi. Hãy thử đoạn mã sau để thấy sự khác nhau trước và sau khi cắt bỏ khoảng trắng.

String space = new String(" Spaces ");

System.out.println(space);

System.out.println(space.trim()); //Sau khi cắt bỏ khoảng trắng

- boolean equals(String s)

Phương thức này so sánh nội dung của hai đối tượng chuỗi.

String name1 = "Java", name2 = "JAVA";

boolean flag = name1.equals(name2);

Biến "flag" chứa giá trị false.

- Các phương thức valueOf được nạp chồng để cho phép chuyển một giá trị thành xâu

static String valueOf(Object obj)//Chuyển một đối tượng thành xâu, bằng cách gọi đến phương thức toString của đối tượng obj
*static String valueOf(char[] characters)//C*huyển mảng các ký tự thành xâu.
static String valueOf(boolean b)//Chuyển một giá trị logic thành xâu, xâu nhận được là "true" hoặc "false" tương ứng với giá trị true hoặc false của b
static String valueOf(char c)//Chuyển kí tự thành xâu
static String valueOf(int i)//chuyển một số nguyên thành xâu
static String valueOf(long l)//Chuyển một giá trị long thành xâu
static String valueOf(float f)//chuyển một giá trị float thành xâu
static String valueOf(double d)//chuyển một giá trị double thành xâu

2. Lớp StringBuffer

Lớp StringBuffer cung cấp các phương thức khác nhau để thao tác một đối

tượng dạng chuỗi. Các đối tượng của lớp này rất mềm dẻo, đó là các ký tự và các chuỗi có thể được chèn vào giữa đối tượng StringBuffer, hoặc nối thêm dữ liệu vào tại vị trí cuối. Lớp này cung cấp nhiều phương thức khởi tạo. Chương trình sau minh hoạ cách sử dụng các phương thức khởi tạo khác nhau để tạo ra các đối tượng của lớp này.

```
class StringBufferCons{

        public static void main(String args[]){

                StringBuffer s1 = new StringBuffer();

                StringBuffer s2 = new StringBuffer(20);

                StringBuffer s3 = new StringBuffer("StringBuffer");

                System.out.println("s3 = "+ s3);

                System.out.println(s2.length());   //chứa 0

                System.out.println(s3.length());  //chứa 12

                System.out.println(s1.capacity());  //chứa 16

                System.out.println(s2.capacity());  //chứa 20

 System.out.println(s3.capacity());  //chứa 28

}

}
```

"length()" và "capacity()" của StringBuffer là hai phương thức hoàn toàn khác nhau. Phương thức "length()" đề cập đến số các ký tự mà đối tượng thực chứa, trong khi "capacity()" trả về tổng dung lượng của một đối tượng (mặc định là 16)

và số ký tự trong đối tượng StringBuffer.

Dung lượng của StringBuffer có thể thay đổi với phương thức "ensureCapacity()". Đối số int đã được truyền đến phương thức này, và dung lượng mới được tính toán như sau:

NewCapacity = OldCapacity * 2 + 2

Trước khi dung lượng của StringBuffer được đặt lại, điều kiện sau sẽ được kiểm tra:

1 Nếu dung lượng(NewCapacity) mới lớn hơn đối số được truyền cho phương thức "ensureCapacity()", thì dung lượng mới (NewCapacity) được đặt.

2 Nếu dung lượng mới nhỏ hơn đối số được truyền cho phương thức "ensureCapacity()", thì dung lượng được đặt bằng giá trị tham số truyền vào.

Chương trình sau minh hoạ dung lượng được tính toán và được đặt như thế nào.

```
class test{

    public static void main(String args[]){

        StringBuffer s1 = new StringBuffer(5);

        System.out.println("Dung lượng của bộ nhớ đệm = "+s1.capacity()); //chứa 5

        s1.ensureCapacity(8);

        System.out.println("Dung lượng của bộ nhớ đệm = "+s1.capacity()); //chứa 12
```

```
        s1.ensureCapacity(30);

        System.out.println("Dung lượng của bộ nhớ đệm =
    "+s1.capacity()); //chứa 30

    }
}
```

Trong đoạn mã trên, dung lượng ban đầu của s1 là 5. Câu lệnh

s1.ensureCapacity(8);

Thiết lập dung lượng của s1 đến 12 =(5*2+2) bởi vì dung lượng truyền vào là 8 nhỏ hơn dung lượng được tính toán là 12 .

s1.ensureCapacity(30);

Thiết lập dung lượng của "s1" đến 30 bởi vì dung lượng truyền vào là 30 thì lớn hơn dung lượng được tính toán (12*2+2).

Các phương thức lớp StringBuffer

Trong phần này, chúng ta sẽ xem xét các phương thức của lớp StringBuffer với một chương trình.

- void append()

Phương thức này nối thêm một chuỗi hoặc một mảng ký tự vào cuối cùng của đối tượng StringBuffer. Ví dụ:

```
StringBuffer s1 = new StringBuffer("Good");

s1.append("evening");
```

Giá trị trong s1 bây giờ là "goodevening".

- void insert()

Phương thức này có hai tham số. Tham số đầu tiên là vị trí chèn. Tham số thứ hai có thể là một chuỗi, một ký tự (char), một giá trị nguyên (int), hay một giá trị số thực (float) được chèn vào. Vị trí chèn sẽ lớn hơn hay bằng 0, và nhỏ hơn hay bằng chiều dài của đối tượng StringBuffer. Bất kỳ đối số nào, trừ ký tự hoặc chuỗi, được chuyển sang chuỗi và sau đó mới được chèn vào. Ví dụ:

StringBuffer str = new StringBuffer("Java sion");

str.insert(1,'b');

Biến "str" chứa chuỗi "Jbava sion".

- char charAt()

Phương thức này trả về một giá trị ký tự trong đối tượng StringBuffer tại vị trí được chỉ định.Ví dụ:

StringBuffer str = new StringBuffer("James Gosling");

char letter = str.charAt(6); //chứa "G"

- void setCharAt(int index, char value)

Phương thức này được sử dụng để thay thế ký tự trong một StringBuffer bằng một ký tự khác tại một vị trí được chỉ định.

StringBuffer name = new StringBuffer("Jawa");

name.setCharAt(2,'v');

Biến "name" chứa "Java".

- void setLength()

Phương thức này thiết lập chiều dài của đối tượng StringBuffer. Nếu chiều dài được chỉ định nhỏ hơn chiều dài dữ liệu hiện tại của nó, thì các ký tự thừa sẽ bị cắt bớt. Nếu chiểu dài chỉ định nhiều hơn chiều dài dữ liệu thì các ký tự null được thêm vào phần cuối của StringBuffer

StringBuffer str = new StringBuffer(10);

str.setLength(str.length() +10);

- char [] getChars()

Phương thức này được sử dụng để trích ra các ký tự từ đối tượng StringBuffer, và sao chép chúng vào một mảng. Phương thức getChars() có bốn tham số sau:

Chỉ số đầu: vị trí bắt đầu, từ nơi mà ký tự được lấy ra.

Chỉ số kết thúc: vị trí kết thúc

Mảng: Mảng đích, nơi mà các ký tự được sao chép.

Vị trí bắt đầu trong mảng đích: Các ký tự được sao chép vào mảng đích từ vị trí này.

Ví dụ:

StringBuffer str = new StringBuffer("Leopard");

char ch[] = new char[10];

str.getChars(3,6,ch,0);

Bây giờ biến "ch" chứa "par"

- void reverse()

Phương thức này đảo ngược nội dung của một đối tượng StringBuffer, và trả về

một đối tượng StringBuffer khác. Ví dụ:

StringBuffer str = new StringBuffer(“devil”);

StringBuffer strrev = str.reverse();

Biến “strrev” chứa “lived”.

III. Lớp StringTokenizer

Một lớp StringTokenizer có thể sử dụng để tách một chuỗi thành các phần tử (token) nhỏ hơn. Ví dụ, mỗi từ trong một câu có thể coi như là một token. Tuy nhiên, lớp StringTokenizer đã đi xa hơn việc phân tách các từ trong câu. Để tách ra các thành token ta có thể tuỳ biến chỉ ra một tập dấu phân cách các token khi khởi tạo đối tượng StringTokenizer. Nếu ta không chỉ ra tập dấu phân cách thì mặc định là dấu trắng (space, tab, ...). Ta cũng có thể sử dụng tập các toán tử toán học (+, *, /, và -) trong khi phân tích một biểu thức.

Bảng sau tóm tắt 3 phương thức tạo dựng của lớp StringTokenizer:

Phương thức xây dựng	Ý nghĩa
StringTokenizer(String)	Tạo ra một đối tượng StringTokenizer mới dựa trên chuỗi được chỉ định.
StringTokenizer(String, Strìng)	Tạo ra một đối tượng StringTokenizer mới dựa trên (String, String) chuỗi được chỉ định và một tập các dấu phân cách.
StringTokenizer(String, String, boolean)	Tạo ra một đối tượng StringTokenizer dựa trên chuỗi được chỉ định, một tập các dấu phân cách, và một cờ hiệu cho biết nếu các dấu phân cách sẽ được trả về như các token hay không.

Các phương thức tạo dựng ở trên được minh họa trong các ví dụ sau:

StringTokenizer st1 = new StringTokenizer("A Stream of words");

*StringTokenizer st2 = new StringTokenizer("4*3/2-1+4", "+-*/", true);*

StringTokenizer st3 = new StringTokenizer("aaa,bbbb,ccc", ",");

Trong câu lệnh đầu tiên, StringTokenizer của "st1" sẽ được xây dựng bằng cách sử dụng các chuỗi được cung cấp và dấu phân cách mặc định. Dấu phân cách mặc định là khoảng trắng, tab, các ký tự xuống dòng. Các dấu phân cách này thì chỉ sử dụng khi phân tách văn bản, như với "st1".

Câu lệnh thứ hai trong ví dụ trên xây dựng một đối tượng StringTokenizer cho các biểu thức toán học bằng cách sử dụng các ký hiệu *, +, /, và -.

Câu lệnh thứ 3, StringTokenizer của "st3" sử dụng dấu phẩy như một dấu phân cách.

Lớp StringTokenizer cài đặt giao diện Enumeration. Vì thế, nó bao gồm các phương thức hasMoreElements() và nextElement(). Các phương thức có thể sử dụng của lớp StringTokenizer được tóm tắt trong bảng sau:

Phương thức	**Mục đích**
countTokens()	Trả về số các token còn lại.
hasMoreElements()	Trả về True nếu còn có token đang được đánh dấu trong chuỗi. Nó thì giống hệt như hasMoreTokens.
hasMoreTokens()	Trả về True nếu còn có token đang được đánh dấu trong chuỗi. Nó giống hệt như hasMoreElements.
nextElement()	Trả về token kế tiếp trong chuỗi. Nó thì giống như nextToken.

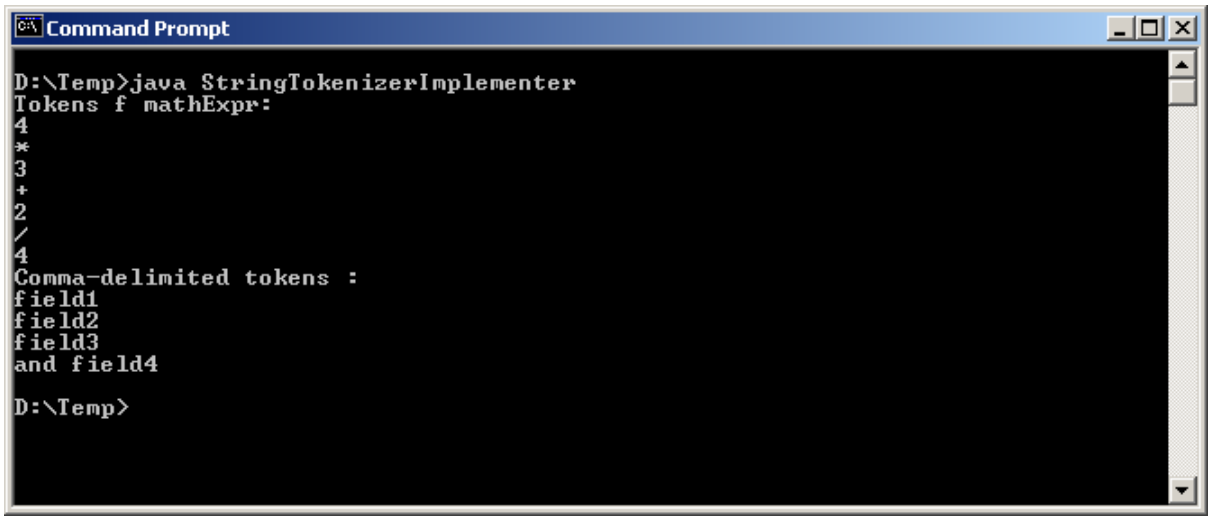
nextToken()	Trả về Token kế tiếp trong chuỗi. Nó thì giống như nextElement.
nextToken(String)	Thay đổi bộ dấu phân cách bằng chuỗi được chỉ định, và sau đó trả về token kế tiếp trong chuỗi.

Hãy xem xét chương trình đã cho ở bên dưới. Trong ví dụ này, hai đối tượng StringTokenizer đã được tạo ra. Đầu tiên, "st1" được sử dụng để phân tách một biểu thức toán học. Thứ hai, "st2" phân tách một dòng của các trường được phân cách bởi dấu phẩy. Cả hai tokenizer, phương thức hasMoreTokens() và nextToken() được sử dụng đế duyệt qua tập các token, và hiển thị các token.

```
import java.util.*;

public class StringTokenizerImplementer{

    public static void main(String args[]){

    // đặt một biểu thức toán học và tạo một tokenizer cho chuỗi đó.

    String mathExpr = "4*3+2/4";

    StringTokenizer st1 = new StringTokenizer(mathExpr,"*+/-", true);

    //trong khi vẫn còn các token, hiển thị  System.out.println("Tokens of mathExpr: ");

    while(st1.hasMoreTokens())

        System.out.println(st1.nextToken());

    //tạo một chuỗi của các trường được phân cách bởi dấu phẩy và tạo //một tokenizer cho chuỗi.
```

```
        String commas = "field1,field2,field3,and field4";
        StringTokenizer st2 = new StringTokenizer(commas,",",false);
        //trong khi vẫn còn token, hiển thị.
        System.out.println("Comma-delimited tokens : ");
        while (st2.hasMoreTokens())
                System.out.println(st2.nextToken());
        }
}
```

Kết quả chạy chương trình được mô tả như hình dưới.



IV. Một số lớp cơ bản của Java

Các lớp cơ bản, hay sử dụng của Java như: Object, các lớp Wrapper (lớp bao của các kiểu dữ liệu nguyên thủy), Math, String và lớp StringBuffer. Những lớp này được xây dựng trong gói java.lang (gói mặc định của Java, khi cần sử dụng các lớp trong gói này ta không cần phải import nó). Mọi lớp trong Java đều

là lớp con của lớp Object .

Cấu trúc phân cấp một số lớp trong gói java.lang

1. Lớp Object

Tất cả các lớp được xây dựng trong các chương trình Java đều hoặc là trực tiếp hoặc gián tiếp được mở rộng từ lớp Object. Đây là lớp cơ sở nhất, định nghĩa hầu như tất cả những phương thức phần cơ bản để các lớp con cháu của nó sử dụng trực tiếp hoặc viết đè.

Object cung cấp các phương thức sau:

int hashCode()

Khi các đối tượng được lưu vào các bảng băm (hash table), hàm này có thể sử dụng để xác định duy nhất giá trị cho mỗi đối tượng. Điều này đảm bảo tính nhất quán của hệ thống khi thực hiện chương trình.

Class getClass()

Trả lại tên lớp của đối tượng hiện thời.

boolean equals(Object obj)

Cho lại kết quả true khi đối tượng hiện thời và obj là cùng một đối tượng. Hàm này thường được viết đè ở các lớp con cho phù hợp với ngữ cảnh so sánh bằng nhau trong các lớp mở rộng đó.

protected Object clone() throws CloneNotSupportedException

Đối tượng mới được tạo ra có cùng các trạng thái như đối tượng hiện thời khi sử dụng clone(), nghĩa là tạo ra bản copy mới của đối tượng hiện thời.

String toString()

Nếu các lớp con không viết đè hàm này thì nó sẽ trả lại dạng biểu diễn văn bản (textual) của đối tượng. Hàm println() ở lớp PrintStream sẽ chuyển các đối số của nó sang dạng văn bản khi sử dụng hàm toString().

protected void finalize() throws Throwable

Hàm này được gọi ngay trước khi đối tượng bị dọn vào "thùng rác", nghĩa là trước khi đối tượng đó bị huỷ bỏ.

2. Các lớp bao kiểu nguyên thủy (Wrapper class)

Các giá trị nguyên thủy không phải là đối tượng trong Java. Để có thể thao tác được trên các giá trị nguyên thủy (giá trị số, kí tự và logic) thì gói *java.lang cung cấp các lớp bao gói (Wrapper) cho từng kiểu dữ liệu nguyên thủy* (gọi tắt là lớp bao). Các lớp bao có những đặc tính chung sau:

1. Các toán tử tạo lập chung. Các lớp bao (trừ lớp Character chỉ có một toán tử tạo lập) đều có hai toán tử tạo lập:
   - Toán tử tạo lập sử dụng giá trị nguyên thủy để tạo ra đối tượng tương ứng

```
Character charObj = new Character('a');
    Boolean boolObj   = new Boolean(true);
    Integer intObj    = new Integer(2002);
    Float floatObj    = new Float(3.14F);
    Double doubleObj  = new Double(3.14);
```

1 Toán tử thứ hai: chuyển các đối tượng lớp String biểu diễn cho các giá trị nguyên thủy về các lớp tương ứng. Các toán tử này sẽ ném ra ngoại lệ NumberFormatException khi giá trị String truyền vào hàm tạo không hợp lệ.

```
    Boolean boolObj = new Boolean("true");
    Integer intObj  = new Integer("2002");
    Float floatObj  = new Float("3.14F");
   Double doubleObj= new Double("3.14");
```

2. Có các hàm tiện ích chung: valueOf(String s), toString(), typeValue(),

equals().

2 Mỗi lớp (trừ lớp Character) đều định nghĩa hàm static valueOf(String s) trả lại đối tượng tương ứng. Các hàm này ném ra ngoại lệ NumberFormatException khi giá trị String truyền vào phương thức không hợp lệ.

```
Boolean boolObj = Boolean.valueOf("true");
Integer intObj  = Integer.valueOf("2002");
Float floatObj  = Float.valueOf("3.14F");
Double doubleObj= Double.valueOf("3.14");
```

3 Các lớp viết đè hàm toString() trả lại là các đối tượng String biểu diễn cho các giá trị nguyên thủy ở dạng xâu.

```
String charStr   = charObj.toString();       //  "a"
String boolStr   = boolObj.toString();       //  "true"
String intStr    = intObj.toString(); // "2002"
String doubleStr = doubleObj.toString(); // "3.14"
```

4 Các lớp định nghĩa hàm typeValue() trả lại các giá trị nguyên thủy tương ứng với các đối tượng nguyên thủy.

```
boolean b = boolObj.booleanValue();                    // true
int i  = intObj.intValue();                            // 2002
float f  = floatObj.floatValueOf();            // 3.14F
double d = doubleObj.doubleValueOf();  // 3.14
char c = charObj.charValue();                          // 'a'
```

5 Các lớp viết đè hàm equals() để thực hiện so sánh bằng nhau của các đối tượng nguyên thủy.

```
Character charObj = new Character('a');
boolean charTest  = charObj.equals('b');    // false
```

```
Integer intObj1   = Integer.valueOf(“2010”);
boolean intTest   = intObj.equals(intObj1); // false
```

Lớp Boolean

Lớp này định nghĩa hai đối tượng Boolean.TRUE, Boolean.FALSE biểu diễn cho hai giá trị nguyên thủy true và false tương ứng.

Lớp Character

Lớp Character định nghĩa hai giá trị cực tiểu, cực đại Character.MIN_VALUE, *Character.MAX_VALUE và các giá trị kiểu ký tự Unicode.* Ngoài ra lớp này còn định nghĩa một số hàm static để xử lý trên các ký tự:

```
static boolean isLowerCase(char ch)// true nếu ch là ký tự thường
static boolean isUpperCase(char ch)// true nếu ch là ký tự viết hoa
static boolean isDigit(char ch)  // true nếuch là chữ số
static boolean isLetter(char ch)// true nếu ch là chữ cái
static boolean isLetterOrDigit(char ch) // true nếu ch là chữ hoặc là số
static char  toUpperCase(char ch)// Chuyển ch về chữ viết hoa
static char  toLowerCase(char ch)// Chuyển ch về chữ viết thường
static char  toTitleCase(char ch)// Chuyển ch về dạng tiêu đề.
```

Các lớp bao kiểu số

Các lớp Byte, Short, Integer, Long, Float, Double là các lớp con của lớp Number. Trong các lớp này đều xác định hai giá trị:

```
<Lớp bao>.MIN_VALUE
<Lớp bao>.MAX_VALUE
```

là các giới hạn của các số trong kiểu đó. Ví dụ,

```
byte minByte = Byte.MIN_VALUE;       // -128
int maxInt   = Integer.MAX_VALUE;    // 2147483647
```

Trong mỗi lớp bao có hàm typeValue() để chuyển các giá trị của các đối tượng nguyên thủy về giá trị số:

byte byteValue()

short shortValue()

int intValue()

long longValue()

float floatValue()

double doubleValue()

Trong mỗi lớp bao còn có hàm static parseType(String s) để chuyển các giá trị được biểu diễn dưới dạng xâu về các giá trị số:

byte value1 = Byte.parseByte(“16”);

int value2   = Integer.parseInt(“2002”);

double value3 = Double.parseDouble(“3.14”);

Ví dụ: Viết chương trình để nhập vào một dãy số tùy ý và sắp xếp theo thứ tự tăng dần.

```
import java.io.*;
class SapXep{
        static int[] day;
        static void nhap(){
        String str;
        int n = day.length;
        DataInputStream stream = new DataInputStream(System.in);
        System.out.println("Nhap vao " + n + " so nguyen");
        for (int k = 0; k < n; k++){
                try{
                  System.out.print(k + ": ”);
                  str = stream.readLine();
                  day[k] = Integer.valueOf(str).intValue();
           }catch(IOException e){
```

```
                System.err.println("I/O Error!");
            }
          }
}
static void hienThi(){
int n = day.length;
for (int k = 0; k < n; k++)
        System.out.print(day[k] + " ");
System.out.println();
}
static void sapXep(){
int x, max, k;
for(int i =day.length-1; i > 0; i--){
        max = day[i];k = i;
        for (int j = 0; j < i; j++)
                if (max < day[j]){
                  max = day[j];
                  k = j;
             }

        day[k] = day[i];
        day[i] = max;
        }
}
public static void main(String[] args){
String str;
int n;
DataInputStream stream = new DataInputStream(System.in);
System.out.print("\nCho biet bao nhieu so nhap vao: ");
```

```
try{
  str = stream.readLine();
}catch(IOException e){
        System.err.println("I/O Error!");
        str = "0";
        }
n = Integer.valueOf(str).intValue();
SapXep.day = new int[n];
nhap();
sapXep();
System.out.println("Day so duoc sap xep: ");
hienThi();
}
}
```

Lớp Void

Lớp này ký hiệu cho đối tượng của lớp Class biểu diễn cho giá trị void.

3. Lớp Math

Lớp final class Math định nghĩa một tập các hàm tĩnh để thực hiện các chức năng chung của toán học như các phép làm tròn số, sinh số ngẫu nhiên, tìm số cực đại, cực tiểu, v.v.

Lớp final class Math còn cung cấp những hằng số như số e (cơ số của logarithm), số pi thông qua Math.E và Math.PI.

Các hàm làm tròn và xử lý các giá trị giới hạn

static int abs(int i)

static long abs(long l)

static float abs(float f)

static double abs(double d)

Hàm abs() được nạp chồng để trả lại giá trị tuyệt đối của đối số.

static double ceil(double d)

Hàm ceil() trả lại giá trị nhỏ nhất kiểu double mà không nhỏ hơn đối số và lại bằng số nguyên. Ví dụ ceil(3.14) cho giá trị 4.0 là số trần trên của đối số.

static double floor(double d)

Hàm floor() trả lại giá trị lớn nhất kiểu double mà không lớn hơn đối số và lại bằng số nguyên. Ví dụ floor(3.14) cho giá trị 3.0 là số sàn dưới của đối số.

static int round(float f d)

static long round(double d)

Hàm round() được nạp chồng để trả lại số nguyên gần nhất của đối số.

static int max(int a, int b)

static long max(long a, long b)

static float max(float a, float b)

static double max(double a, double b)

Hàm max() được nạp chồng để trả lại giá trị cực đại của hai đối số.

static int min(int a, int b)

static long min(long a, long b)

static float min(float a, float b)

static double min(double a, double b)

Hàm min() được nạp chồng để trả lại giá trị cực tiểu của hai đối số.

Các hàm lũy thừa

static double pow(double d1, double d2)

Hàm pow() trả lại giá trị là lũy thừa của d1 và d2 (d1d2).

static double exp(double d)

Hàm exp() trả lại giá trị là luỹ thừa cơ số e và số mũ d (ed).

static double log(double d)

Hàm log() trả lại giá trị là lô-ga-rit tự nhiên (cơ số e) của d.

static double   sqrt(double d)

Hàm sqrt() trả lại giá trị là căn bậc hai của d , hoặc giá trị NaN nếu đối số âm.

Các hàm lượng giác

static double   sin(double d)

Hàm sin() trả lại giá trị là sine của góc d được cho dưới dạng radian.

static double   cos(double d)

Hàm cos() trả lại giá trị là cose của góc d được cho dưới dạng radian.

static double   tan(double d)

Hàm tan() trả lại giá trị là tangent của góc d được cho dưới dạng radian.

Hàm sinh số ngẫu nhiên

static double   random()

Hàm random() cho lại giá trị là số ngẫu nhiên trong khoảng từ 0.0 đến 1.0.

V. Các lớp tập hợp

Tập hợp (Collection) trong Java cho phép lưu lại tham chiếu đến các đối tượng. Các đối tượng bất kỳ có thể được lưu trữ, tìm kiếm và được thao tác như là các phần tử của tập hợp.

Phần giao diện

Giao diện (interface) Collection là cơ sở để phát triển, mở rộng thành các giao diện khác như Set, List, SortedSet và Map và giao diện cơ sở để mở rộng thành SortedMap. Hình sau mô tả cấu trúc phân cấp theo quan hệ kế thừa của các giao diện lõi.

Các giao diện lõi của cấu trúc Collection được mô tả trong bảng sau:

interface	Mô tả
Collection	*interface cơ sở định nghĩa tất cả các phép toán cơ* bản cho các lớp cần duy trì thực hiện và cài đặt chúng
Set	Mở rộng Collection để cài đặt cấu trúc tập hợp, trong đó không có phần tử được lặp và chúng không được sắp xếp
SortedSet	Mở rộng Set để cài đặt cấu trúc tập hợp được sắp, trong đó không có phần tử được lặp và chúng được sắp xếp theo thứ tự
List	Mở rộng Collection để cài đặt cấu trúc danh sách, trong đó các phần tử được sắp xếp theo thứ tự, và có lặp
Map	*interface cơ sở định nghĩa các phép toán để các* lớp sử dụng và cài đặt các ánh xạ từ khoá sang các giá trị
SortedMap	Mở rộng của Map để cài đặt các ánh xạ khoá theo thứ tự

Phần cài đặt

Gói java.util cung cấp tập các lớp cài đặt các giao diện lõi để tạo ra những cấu trúc dữ liệu thường sử dụng như: Vector, HashTable, HashSet, LinkedList, TreeSet*, v.v. Những lớp này và giao diện lõi được xây dựng* theo cấu trúc phân cấp như trong hình H6-3.

Các giao diện lõi và các lớp cài đặt chúng

Trong hình trên ký hiệu → biểu diễn cho quan hệ kế thừa giữa các giao diện và --> biểu diễn cho sự cài đặt các giao diện.

Phần thuật toán

Lớp java.util.Collection (cần phân biệt với giao diện Collection) cung cấp một số hàm *static thực hiện những thuật toán đa xạ cho những phép toán khác* nhau trên tập hợp, kể cả sắp xếp, tìm kiếm và dịch chuyển các phần tử.

Một số hàm trong số đó là:

static int binarySearch(List list, Object key)

Sử dụng thuật toán tìm kiếm nhị phân để xác định chỉ số của phần tử key trong danh sách list.

static void fill(List list, Object obj)

Thay thế tất cả các phần tử trong danh sách list bằng obj.

static void shuffle(List list)

Hoán vị các phần tử của danh sách list một cách ngẫu nhiên.

static void sort(List list)

Sắp xếp các phần tử trong danh sách list theo thứ tự tăng dần.

1. Collection

Giao diện Collection được xây dựng như là mẫu hợp đồng cho tất cả các cấu trúc tập hợp có thể dựa vào đó mà thực thi và cài đặt. Gói java.util cung cấp các lớp tập hợp và cài đặt hầu hết các hàm của *Collection*.

Các phép toán cơ sở

int size();Xác định kích thước của tập hợp.

boolean isEmpty();Trả lại true nếu tập hợp rỗng, ngược lại false.

boolean contains(Object obj);Trả lại true nếu tập hợp chứa obj, ngược lại false.

boolean add(Object obj); // Tùy chọn

boolean remove(Object obj); // Tùy chọn

Trả lại true nếu tập hợp thực hiện thành công việc bổ sung (loại bỏ) obj, ngược lại false.

Một số phép toán khác

Những phép toán sau thực hiện trên tập hợp như một đơn vị cấu trúc dữ liệu.

boolean cotainAll(Collection c); // Tùy chọn

Kiểm tra xem tập hợp hiện thời có chứa cả tập hợp c hay không.

boolean addAll(Collection c);// Tùy chọn

Thực hiện phép hợp hai tập hợp

boolean removeAll(Collection c); // Tùy chọn

Thực hiện phép trừ hai tập hợp

boolean retainAll(Collection c); // Tùy chọn

Thực hiện phép giao hai tập hợp

void clear() // Tùy chọn

Hủy bỏ đối tượng trong tập hợp các phép trên trả lại true nếu thực hiện thành công và cho kết quả thực hiện được minh họa như trong hình H6-4, trong đó a, b là hai tập hợp bất kỳ.

a.addAll(b) a.removeAll(b) a.retainAll(b)

Hình Các phép toán trên các tập hợp

2. Set (tập hợp)

Tập hợp Set là cấu trúc dữ liệu, trong đó không có sự lặp lại và không có sự sắp xếp của các phần tử. Giao diện Set không định nghĩa thêm các hàm mới mà chỉ giới hạn lại các hàm của Collection để không cho phép các phần tử của nó được lặp lại.

Giả sử a, b là hai tập hợp (hai đối tượng của các lớp cài đặt Set). Kết quả thực hiện trên a, b có thể mô tả như trong bảng sau:

Các hàm trong Set	Các phép hợp tương ứng
a.containsAll(b)	$b \subseteq a$? (Tập con)
a.addAll(b)	$a = a \cup b$ (Hợp tập hợp)
a.removeAll(b)	a = a - b (Hiệu tập hợp)
a.retainAll(b)	$a = a \cap b$ (Giao tập hợp)
a.clear()	$a = \varnothing$ (Tập rỗng)

Sau đây chúng ta xét một số lớp thực thi cài đặt giao diện Set.

HashSet

Một dạng cài đặt nguyên thủy của Set là lớp HashSet, trong đó các phần tử của nó là không được sắp. Lớp này có các toán tử tạo lập:

HashSet()

Tạo ra một tập mới không có phần tử nào cả (tập rỗng).

HashSet(Collection c)

Tạo ra một tập mới chứa các phần tử của tập hợp c nhưng không cho phép lặp.

HashSet(int initCapacity)

Tạo ra một tập mới rỗng có kích thước (khả năng chứa) là initCapacity.

HashSet(int initCapacity, float loadFactor)

Tạo ra một tập mới rỗng có kích thước (khả năng chứa) là initCapacity và yếu tố được nạp vào là loadFactor.

Ví dụ 6.4 Khi thực hiện các đối số được đưa vào sau tên chương trình theo dòng lệnh. Chương trình bắt đầu với tap1 là rỗng và lấy các ký tự của đối số đầu tiên để tạo ra tap2. So sánh hai tập đó, thông báo kết quả ra màn hình, sau đó cộng dồn tap2 vào tap1 và lại tiếp tục như thế đối với đối số tiếp theo cho đến hết.

```
import java.util.*;
public class TapKT {
    public static void main(String args[]){
    int nArgs = args.length;              // Số đối số của chương trình
    Set tap1 = new HashSet();  // Tạo ra tập thứ nhất là rỗng
    for (int i = 0; i < nArgs; i++){
        String arg = args[i]; // Lấy từng đối số của chương trình
        Set tap2 = new HashSet();// Tạo ra tập thứ 2
        int size = arg.length();        // Số ký tự trong mỗi đối số
        for (int j = 0; j < size; j++)// Tập thứ 2 chứa các ký tự của arg
          tap2.add(new Character(arg.charAt(j)));
        // Tạo ra tập tapChung chính bằng tap1
        Set tapChung = new HashSet(tap1);
        tapChung.retainAll(tap2);// tapChung = tap1 ∩ tap2
        boolean b = tapChung.size() == 0;
        if (b)
          System.out.println(tap2+" va "+tap1+" la roi nhau");
        else {
          // tap2 có phải là tập con của tap1?
```

```
                boolean isSubset = tap1.containsAll(tap2);
                // tap1 có phải là tập con của tap2?
                boolean isSuperset = tap2.containsAll(tap1);
                 // tap1 có bằng tap2?
                if (isSuperset && isSubset)
                 System.out.println(tap2 + " bang tap " + tap1);
                else if (isSubset)
                 System.out.println(tap2+" la tap con cua "+tap1);
                else if (isSuperset)
                 System.out.println(tap2+" la tap cha cua "+tap1);
                else
                 System.out.println(tap2 + " va " + tap1 + " co "
              + tapChung + " la phan chung");
                 }
                 tap1.addAll(tap2);// hợp tap2 vào tap1
              }
          }
        }
```

Dịch và thực hiện chương trình với các đối số như sau:

java TapKT em voi em anh

sẽ cho kết quả:

```
[m, e] va [ ] la roi nhau
        [v, i, o] va [m, e] la roi nhau
        [m, e] la tap con cua [m, v, i, e, o]
        [a, h, n] va [m, v, i, e, o] la roi nhau
```

3. List (danh sách)

Cấu trúc List là dạng tập hợp các phần tử được sắp theo thứ tự (còn được gọi là dãy tuần tự) và trong đó cho phép lặp (hai phần tử giống nhau). Ngoài những hàm mà nó được kế thừa từ Collection, List còn bổ sung thêm những hàm như:

Object get(int index)

Cho lại phần tử được xác định bởi index.

Object set(int index, Object elem) *// Tùy chọn*

Thay thế phần tử được xác định bởi index bằng elem.

void add(int index, Object elem) *// Tùy chọn*

Chèn elem vào sau phần tử được xác định bởi index.

Object remove(int index) *// Tùy chọn*

Bỏ đi phần tử được xác định bởi index.

boolean addAll(int index, Collection c) *// Tùy chọn*

Chèn các phần tử của tập hợp c vào vị trí được xác định bởi index.

int indexOf(Object elem)

Cho biết vị trí lần xuất hiện đầu tiên của elem trong danh sách.

int lastIndexOf(Object elem)

Cho biết vị trí lần xuất hiện cuối cùng của elem trong danh sách.

List subList(int fromIndex, int toIndex)

Lấy ra một danh sách con từ vị trí fromIndex đến toIndex .

ListIterator listIterator()

Cho lại các phần tử liên tiếp bắt đầu từ phần tử đầu tiên.

ListIterator listIterator(int index)

Cho lại các phần tử liên tiếp bắt đầu từ phần tử được xác định bởi *index*.

Trong đó ListIterator là giao diện mở rộng giao diện Iterator đã có trong

java.lang.

Các lớp ArrayList, Vector và LinkedList

Ba lớp này có những toán tử tạo lập để tạo ra những danh sách mới rỗng hoặc có các phần tử lấy theo các tập hợp khác.

Vector và ArrayList là hai lớp kiểu mảng động (kích thước thay đổi được). Hiệu suất sử dụng hai lớp này là tương đương nhau, tuy nhiên nếu xét theo nhiều khía cạnh khác thì ArrayList là cấu trúc hiệu quả nhất để cài đặt cấu trúc danh sách List.

Ví dụ 6.5 Hệ thống có một dãy N_DIGIT (5) chữ số bí mật. Hãy viết chương trình nhập vào N_DIGIT chữ số để đoán xem có bao nhiêu chữ số trùng và có bao nhiêu vị trí các chữ số trùng với dãy số cho trước.

```
import java.util.*;
public class NhapDoanSo {
        final static int N_DIGIT = 5;
        public static void main(String args[]){
                if(args.length != N_DIGIT) {
                 System.err.println("Hay doan " + N_DIGIT + " chu so!");
                 return;
                        }
                List biMat = new ArrayList();// Tạo danh sách biMat là rỗng
                biMat.add("5");                    // Bổ sung các số vào dãy biMat
                biMat.add("3");
                biMat.add("2");
                biMat.add("7");
                biMat.add("2");
                List doan = new ArrayList();// Tạo danh sách doan là rỗng
                for(int i = 0; i < N_DIGIT; i++)
                  doan.add(args[i]); // Đưa các số từ đối số chương trình vào doan
```

```
            List lap = new ArrayList(biMat);// Lưu lưu biMat sang lap
            int nChua = 0;
            // Đếm số các chữ số trùng nhau, nghĩa là thực hiện được phép bỏ đi
remove()
            for(int i = 0; i < N_DIGIT; i++)
              if (lap.remove(doan.get(i))) ++nChua;
            int nViTri = 0;
            ListIterator kiemTra = biMat.listIterator();
            ListIterator thu = doan.listIterator();
            // Tìm những vị trí đoán trúng trong hai dãy có lặp
            while (kiemTra.hasNext())// Khi còn phần tử tiếp theo
            // Kiểm tra xem lần lượt các vị trí của hai dãy có trùng nhau hay
không
              if (kiemTra.next().equals(thu.next())) nViTri++;
            // Thông báo kết quả ra màn hình
            System.out.println(nChua + " chu so doan trung.");
            System.out.println(nViTri + " vi tri doan trung.");
            }
      }
```

Dịch và thực hiện chương trình:

```
java NhapDoanSo 3 2 2 2 7
```

sẽ cho kết quả:

```
4 chu so doan trung
1 vi tri doan trung
```

4. Map (ánh xạ)

Map định nghĩa các ánh xạ từ các khoá (keys) vào các giá trị. Lưu ý: các khóa phải là duy nhất (không cho phép lặp). Mỗi koá được ánh xạ sang nhiều nhất một giá trị, được gọi là ánh xạ đơn.

Các ánh xạ không phải là các tập hợp, giao diện Map cũng không phải là mở rộng của các Collection. Song, phép ánh xạ có thể xem như là một loại tập hợp theo nghĩa: các khoá (key) tạo thành tập hợp và tập hợp các giá trị (value) hoặc tập các cặp <key, value>.

Giao diện Map khai báo những hàm sau:

Object put(Object key, Object value); ***// Tùy chọn***

Chèn vào một cặp <key, value>

Object get(Object key);

Đọc giá trị được xác định bởi key nếu có ánh xạ, ngược lại cho null nếu không có ánh xạ ứng với key.

Object remove(Object key); ***// Tùy chọn***

Loại bỏ ánh xạ được xác định bởi key.

boolean containsKey(Object key);

Cho giá trị true nếu key được ánh xạ sang một giá trị nào đó, ngược lại là *false*.

boolean containsValue(Object value);

Cho giá trị true nếu value được ánh xạ bởi một key nào đó, ngược lại là *false*.

int size();

Cho số các cặp ánh xạ <key, value>.

boolean isEmpty();

Cho giá trị true nếu ánh xạ rỗng, ngược lại là false.

Một số phép toán thường dùng

void putAll(Map t); *// Tùy chọn*

Sao lại các ánh xạ từ t.

void clear(); *// Tùy chọn*

Xoá đi tất cả các ánh xạ.

Set keySet();

Xác định tập các khoá.

Collection values();

Xác định tập hợp các giá trị.

Set entrySet();

Xác định tập các ánh xạ <key, value>.

Các lớp HashMap và HashTable

Hai lớp này cài đặt giao diện Map và được xây dựng trong java.lang. Chúng cho phép tạo ra các ánh xạ mới có thể rỗng hoặc có những kích thước tùy ý.

I. *Ví dụ 6.6 Viết chương trình nhập vào các trọng lượng và in ra tần suất của các trọng lượng đó trong các nhóm cách nhau 5 đơn vị (kg).*

```
import java.util.*;
public class NhomTrongluong {
        public static void main(String args[]){
                // Tạo ra một ánh xạ để lưu tần suất của mỗi nhóm
                Map demNhom = new HashMap();
                int nArgs = args.length;
                // Đọc các trọng lượng được nhập vào từ đối số và chia nhóm cách nhau 5 đơn vị.
                for(int i = 0; i < nArgs; i++){
                 double trongL = Double.parseDouble(args[i]);
                 Integer nhomTL=new Integer((int)Math.round(trongL/5)*5);
                 Integer demCu = (Integer)demNhom.get(nhomTL);
                // Tăng số lần trọng lượng trong cùng nhóm, nếu là lần đâu (demCu = null) thì đặt là 1.
                 Integer demMoi = (demCu == null)?
                        new Integer(1): new Integer(demCu.intValue()+1);
                 demNhom.put(nhomTL, demMoi);
              }
```

```
            // Lấy ra tập các giá trị từ ánh xạ demNhom
            List keys = new ArrayList(demNhom.keySet());
            // Sắp xếp lại theo các nhóm trọng lượng
            Collections.sort(keys);
            ListIterator keyIterator = keys.listIterator();
            // Tìm tần suất của các trọng lượng được nhập vào trong các nhóm
            while(keyIterator.hasNext()) {
                  Integer nhom = (Integer) keyIterator.next();
                  Integer dem = (Integer) demNhom.get(nhom);
                  int demInt = dem.intValue();
                  // Sử dụng hàm fill() của lớp Array để tạo ra xâu gồm
demInt các dấu '*'
                  char[] bar = new char[demInt];
                  Arrays.fill(bar, '*');
                  System.out.println(nhom+"\t" + new String(bar));
               }
            }
      }
```

Dịch và chạy chương trình NhomTrongLuong với các tham số:

```
java NhomTrongLuong 75 72 93 12 34
```

sẽ cho kết quả:

```
10      *
35      *
36      **
95      *
```

Như vậy, nhóm 10 kg có 1, 35 kg có 1, 75 kg có 2 và 95 kg có 1.

4. SortedSet (tập được sắp) và SortedMap (ánh xạ được sắp)

Các cấu trúc tập hợp (set) và ánh xạ (map) có giao diện đặc biệt là SortedSet và SortedMap như trong hình H6-5 để cài đặt những cấu trúc có các phần tử được sắp

theo thứ tự chỉ định.

Giao diện SortedSet

SortedSet là giao diện mở rộng của Set cung cấp các hàm để xử lý các tập được sắp.

SortedSet headSet(Object toElem);

Cho lại tập được sắp gồm những phần tử đứng trước toElem.

SortedSet tailSet(Object fromElem);

Cho lại tập được sắp gồm những phần tử cuối đứng sau fromElem.

SortedSet subSet(Object fromElem, Object toElem);

Cho lại tập được sắp gồm những phần tử kể từ fromElem đến toElem.

Object first();Cho lại phần tử đầu tiên (cực tiểu) của tập được sắp.

Object last();Cho lại phần tử cuối cùng (cực đại) của tập được sắp.

Comparator comparator()

Cho lại thứ tự so sánh của cấu trúc được sắp, cho null nếu các phần tử được sắp theo thứ tự tự nhiên (tăng dần)

Giao diện SortedMap

SortedMap là giao diện mở rộng của Map cung cấp các hàm để xử lý các ánh xạ được sắp theo thứ tự của khoá (key).

SortedMap headMap(Object toKey);Cho lại ánh xạ được sắp gồm những phần tử đứng trước toKey.

SortedMap tailMap(Object fromKey);Cho lại ánh xạ được sắp gồm những phần tử cuối đứng sau fromKey.

SortedMap subMap(Object fromKey, Object toKey);Cho lại ánh xạ được sắp gồm những phần tử kể từ fromKey đến toKey.

Object firstKey();Cho lại phần tử đầu tiên (cực tiểu) của ánh xạ được sắp.

Object lastKey();Cho lại phần tử cuối cùng (cực đại) của ánh xạ được sắp.

TreeSet và TreeMap

Hai lớp này cài đặt hai giao diện SortedSet và SortedMap tương ứng. Chúng có bốn loại toán tử tạo lập như sau:

TreeSet()

TreeMap()

Tạo ra những tập hoặc ánh xạ mới và rỗng, được sắp theo thứ tự tăng dần của các phần tử hoặc của khoá.

TreeSet(Comparator c)

TreeMap(Comparator c)

Tạo ra những tập hoặc ánh xạ mới được sắp và xác định thứ tự so sánh theo c.

TreeSet(Collection c)

TreeMap(Map m)

Tạo ra những tập hoặc ánh xạ mới được sắp và có các phần tử lấy từ c hoặc từ m tương ứng.

TreeSet(SortedSet s)

TreeMap(SortedMap m)

Tạo ra những tập hoặc ánh xạ mới được sắp và có các phần tử lấy từ s hoặc từ m tương ứng.

Chương 3

XỬ LÝ NGOẠI LỆ

Đối với người lập trình họ có thể gặp một trong các lỗi sau:

1 Lỗi cú pháp (syntac error)

2 Lỗi logic thuật toán

3 Lỗi lúc thực thi (runtime error)

- Đối với lỗi cú pháp người lập trình có thể phát hiện và sửa lỗi, dựa vào trình biên dịch, đây là lỗi dễ phát hiện và sửa chữa, tuy nhiêmn đây cũng là lỗi gây khó khăn và chán nản đối với người mới học lập trình.
- Đối với lỗi thuật toán, đây là lỗi khó phát hiện và sửa chữa nhất, tuy nhiên trong bài này ta không bàn luận về vấn đề này.
- Đối với lỗi lúc thực thi, ta hoàn toàn có thể kiểm soát được chúng, thông thường lỗi runtime thường do nguyên nhân khách quan như: truy cập vào một ổ đĩa nhưng ổ đĩa này lại chưa sẵn sàng, hay thực hiện phép chia nhưng mẫu số lại bằng 0, kết nối với máy tính ở xa nhưng máy đó lại không tồn tại…, khi một lỗi runtime xẩy ra JVM sẽ phát sinh một ngoại lệ, nếu một chương trình không cung cấp mã sử lý ngoại lệ có thể kết thúc không bình thường, trong bài hôm nay ta sẽ bàn về vấn đề sử lý ngoại lệ trong java.
- Mọi lớp biệt lệ trong java đều được dẫn xuất từ lớp cơ sở Throwable, ta có thể tạo ra lớp ngoại lệ riêng bằng cách mở rộng lớp *Throwable*

I. 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 ngoại lệ xảy ra. Trong trường hợp đó, tất cả các nguồn tài nguyên mà hệ thống đã cấp không được giải phóng. Điều này gây lãng phí tài nguyên. Để tránh trường hợp này, tất cả các nguồn tài nguyên mà hệ thống cấp nên được thu hồi lại. Tiến trình này đòi hỏi cơ chế xử lý ngoại lệ thích hợp.

Ví dụ, xét thao tác vào ra (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 tập tin lại. Lúc đó tập tin dễ bị hư hại và các nguồn tài nguyên được cấp phát cho tập tin không được trả lại cho hệ thống.

II. Mô hình sử lý ngoại lệ của java

Mô hình sử lý ngoại lệ của java dựa trên ba hoạt động chính: đặc tả *ngoại lệ, ném ra ngoại lệ, và bắt ngoại lệ.*

- Mỗi phương thức đều có thể phát sinh các ngoại lệ, các ngoại lệ có thể phát sinh cần được mô tả chi tiết trong lệnh khai báo của phương thức, việc khai báo này được gọi là đặc tả ngoại lệ.
- Khi một câu lệnh trong phương thức gây lỗi, mà người lập trình không cung cấp mã xử lý lỗi, thì ngoại lệ được chuyển đến phương thức gọi phương thức đó, việc này được gọi là ném ra biệt lệ, ta có thể ném ra biệt lệ một cách tường minh (điều này sẽ được giới thiếu sau).
- Sau khi JVM ném ra một ngoại lệ, thì hệ thống thi hành java bắt đầu tiến trình tìm mã xử lý lỗi. Mã xử lý lỗi hay còn gọi là mã xử lý biệt lệ, java runtime sẽ tìm mã xử lý lỗi bằng cách lần ngược trở lại chuỗi các phương thức gọi nhau, bắt đầu từ phương thức hiện tại. Chương trình sẽ kết thúc nếu không tìm thấy mã xử lý biệt lệ. Quá trình tìm kiếm này gọi là bắt biệt lệ.

III. Đặc tả ngoại lệ

Đặc tả ngoại lệ là khai báo cho trình biên dịch biết là phương thức này có thể gây ra ngoại lệ lúc thi hành.

Để khai báo biệt lệ ta sử dụng từ khoá throws trong khai báo phương thức, ví dụ:

public void myMethod() throws IOException, RemoteException

từ khoá throws chỉ cho trình biên dịch java biết rằng phương thức này có thể ném

ra ngoại lệ IOException và RemoteException, nếu một phương thức ném ra nhiều ngoại lệ thì các ngoại lệ được khai báo cách nhau bởi dấu phẩy ‘,’

III. Ném ra ngoại lệ

Một phương thức sau khi đã khai báo các biệt lệ, thì bạn (hoặc chương trình thực thi java) có thể ném ra các đối tượng biệt lệ, có kiểu mà ta đã khai báo trong danh sách throws. Cú pháp của lệnh ném ra ngoại lệ:

throw ExceptionObject;

Chú ý:

3 Bạn phải chú ý giữa lệnh khai báo biệt lệ và lệnh ném ra ngoại lệ

4 Một phương thức chỉ có thể ném ra các ngoại lệ mà nó đươc khai báo

IV. Bắt ngoại lệ

Một ngoại lệ (exception) trong chương trình Java là dấu hiệu chỉ ra rằng có sự xuất hiện một điều kiện không bình thường nào đó.

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 ngoại lệ này có thể là một ngoại lệ chuẩn của Java hoặc có thể là một ngoại lệ do ta tạo ra. Lớp ‘Throwable’ được Java cung cấp là cha của tất cả các ngoại lệ trong Java (lớp đầu tiên trong cây thừa kế).

Sau khi bạn đã biết cách khai báo và ném ra biệt lệ, thì phần việc quan trọng nhất là bắt và xử lý biệt lệ.

Vấn đề đối với người lập trình java là phải biết được đoạn mã nào của anh ta có thể gây ra lỗi. Khi họ đã khoanh vùng được đoạn mã có thể gây ra lỗi họ sẽ đặt đoạn mã, có khả năng gây ra lỗi đó trong khối try (thử làm), và đặt đoạn mã xử lý lỗi trong khối catch (bắt giữ). Khuôn dạng tổng quát như sau:

try{

```
// Các lệnh có khả năng gây lỗi
}
catch ( TypeException1 ex){
// Mã được thực thi khi một ngoại lệ TypeException1 được phát sinh trong khối try
}
catch ( TypeException2 ex){
// Mã được thực thi khi một ngoại lệ TypeException2 được phát sinh trong khối try
}
...
catch ( TypeExceptionN ex){
// Mã được thực thi khi một ngoại lệ TypeExceptionN được phát sinh trong khối try
} finally{
// khối lệnh nay luôn được thực hiện cho dù ngoại lệ có xảy ra trong khối try hay không.
}
```

Nếu không có một ngoại lệ nào phát sinh trong khối try thì các mệnh đề catch sẽ bị bỏ qua, trong trường hợp một trong các câu lệnh bên trong khối try gây ra một ngoại lệ thì, thì java sẽ bỏ qua các câu lệnh còn lại trong khối try để đi tìm mã xử lý ngoại lệ, nếu kiểu ngoại lệ so khớp với kiểu ngoại lệ trong mệnh đề catch, thì mã lệnh trong khối catch đó sẽ được thực thi, nếu không tìm thấy một kiểu ngại lệ nào được so khớp java sẽ kết thúc phương thức đó và chuyển biệt lệ đó ra phương thức đã gọi phương thức này quá trình này được tiếp tục cho đến khi tìm thấy mã xử lý biệt lệ, nếu không tìm thấy mã xử lý biệt lệ trong chuỗi các

phương thức gọi nhau, chương trình có thể chấm dứt và in thông báo lỗi ra luồng lỗi chuẩn System.err

Ví dụ

```
class TryClass{
    public static void main(String args[]){
        int n=0;
        try{
                System.out.println(1/n);
        }
        catch(ArithmeticException ex){
                System.out.println("Loi chia cho 0");
        }
    }
}
```

Khi chạy chương trình này ta se thu được một dòng in ra màn hình như sau:
Loi chia cho 0

Trong đoạn chương trình trên khi chia một số cho 0 sẽ gặp ngoại lệ ***ArithmeticException***, biết được ngoại lệ này có thể xẩy ra do vậy ta bắt nó và xử lý trong khối ***catch(ArithmeticException ex)***, ở đây ex là một đối tượng của lớp ***ArithmeticException*** chứa các thông tin về ngoại lệ xẩy ra, ta có thể lấy cá thông tin về ngoại lệ chẳng hạn như lấy về mô tả ngoại lệ như sau:
System.out.println(a.getMessage()).

V. Khối 'finally'

Khi một ngoại lệ xuất hiện, phương thức đang được thực thi có thể bị dừng mà

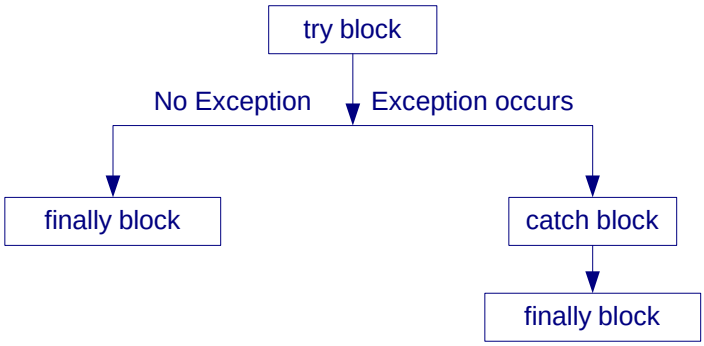
không được hoàn thành. Nếu điều này xảy ra, thì các đoạn mã phía sau (ví dụ như đoạn mã có chức năng thu hồi tài nguyên, như các lệnh đóng tập viết ở cuối phương thức) sẽ không bao giờ được gọi. Java cung cấp khối ***finally*** để giải quyết việc này. Thông thường khối 'finally' chứa các câu lệnh mang tính chất dọn dẹp như: đóng kết nối CSDL, đóng tệp tin,….

```
try{
    //Các lệnh có khả năng ném ra ngoại lệ
}
catch(Exception1 ex1){
…
}
catch(Exception2 ex2){
…
}
catch(Exceptionn exn){
…
}

finally{
    //Mã lệnh dọn dẹp
}
```

Khối 'finally' là tuỳ chọn, không bắt buộc phải có. Khối này được đặt sau khối 'catch' cuối cùng. Chương trình sẽ thực thi câu lệnh đầu tiên của khối 'finally' ngay sau khi gặp câu lệnh 'return' hay lệnh 'break' trong khối 'try'.

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

Hình minh họa sự thực hiện của các khối ‘try’, ‘catch’ và ‘finally’.



VI. Một số lớp ngoại lệ chuẩn của Java

Danh sách một số lớp ngoại lệ

Tên lớp ngoại lệ	Ý nghĩa
Throwable	Đây là lớp cha của mọi lớp ngoại lệ trong Java
Exception	Đây là lớp con trực tiếp của lớp Throwable, nó mô tả một ngoại lệ tổng quát có thể xẩy ra trong ứng dụng
RuntimeException	Lớp cơ sở cho nhiều ngoại lệ java.lang
ArthmeticException	Lỗi về số học, ví dụ như ‘chia cho 0’.
IllegalAccessException	Lớp không thể truy cập.
IllegalArgumentException	Đối số không hợp lệ.
ArrayIndexOutOfBoundsExeption	Lỗi truy cập ra ngoài mảng.
NullPointerException	Khi truy cập đối tượng null.
SecurityException	Cơ chế bảo mật không cho phép thực hiện.
ClassNotFoundException	Không thể nạp lớp yêu cầu.
NumberFormatException	Việc chuyển đổi từ chuỗi sang số không thành

	công.
AWTException	Ngoại lệ về AWT
IOException	Lớp cha của các lớp 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.

Chương 4

LẬP TRÌNH ĐA TUYẾN

I. Các kiến thức liên quan

1. Tiến trình (process)

Tiến trình là một thể hiện của một chương trình đang xử lý. Sở hữu một con trỏ lệnh, tập các thanh ghi và các biến. để hoàn thành tác vụ của mình, một tiến trình còn cần đến một số tài nguyên khác như: CPU, bộ nhớ, các tập tin, các thiết bị ngoại vi..

Cần phân biệt được giữa tiến trình và chương trình. Một chương trình là một thể hiện thụ động, chứa các chỉ thị điều khiển máy tính để thực hiện mục đích gì đó; khi cho thực thi chỉ thị này thì chương trình sẽ biến thành tiến trình

Có thể nói tóm tắt tiến trình là một chương trình chạy trên hệ điều hành và được quản lý thông qua một số hiệu gọi là thẻ

2. Tiểu trình (thread)

Một tiểu trình là một đơn vị xử lý cơ bản trong hệ thống. Mỗi tiểu trình xử lý tuần tự các đoạn code của nó, sở hữu một con trỏ lệnh, một tập các thanh ghi và một vùng nhớ stack riêng, các tiểu trình chia sẻ CPU với nhau giống như cách chia

sẻ giữa các tiến trình. Một tiến trình sở hữu nhiều tiểu trình, tuy nhiên một tiểu trình chỉ có thể thuộc về một tiến trình, các tiểu trình bên trong cùng một tiến trình chia sẻ nhau không gian địa chỉ chung, điều này có nghĩa là các tiểu trình có thể chia sẻ nhau các biến toàn cục của tiến trình. Một tiểu trình cũng có thể có các trạng thái giống như các trạng thái của một tiến trình.

3. Hệ điều hành đơn nhiệm, đa nhiệm

- HĐH đơn nhiệm là HĐH chỉ cho phép 1 tiến trình chạy tại một thời điểm, ví dụ HĐH DOS là HĐH đơn nhiệm.
- - HĐH đa nhiệm cho phép nhiều tiến trình chạy tại một thời điểm, ví dụ HĐH windows, Unix, Linux là các HĐH đa nhiệm
- HĐH đa nhiệm ưu tiên: các tiến trình được cấp phát thời gian sử dụng CPU theo mức ưu tiên khác nhau
- HĐH đa nhiệm không ưu tiên: các tiến trình không có mức ưu tiên nào cả, chúng "tự giác" nhả quyền kiểm soát CPUsau khi kết thúc phần công việc

Chú ý: trong thực tế mỗi máy thường chỉ có 1 CPU, nên không thể có nhiều tiến trình chạy tại một thời điểm. Nên thông thường sự đa chương chỉ là giả lập. Chúng được giả lập bằng cách lưu trữ nhiều tiến trình trong bộ nhớ tại một thời điểm, và điều phối CPU qua lại giữa các tiến trình.

4. Các trạng thái của tiến trình

Trạng thái của một tiến trình tại một thời điểm được xác định bởi hoạt động hiện thời của tiến trình đó. Trong quá trình sống một tiến trình thay đổi trạng thái do nhiều nguyên nhân như: hết thời gian sử dụng CPU, phải chờ một sự kiện nào đó xẩy ra, hay đợi một thao tác nhập/xuất hoàn tất…

Tại một thời điểm một tiến trình có thể nhận một trong các trạng thái sau đây:

- Tạo mới: tiến trình đang được thành lập
- Running: các chỉ thị của tiến trình đang được xử lý, hay nói cách khác tiến

trình đang sở hữu CPU

- Blocked: tiến trình đang chờ được cấp tài nguyên, hay chờ một sự kiện nào đó xẩy ra
- Ready: tiến trình đang chờ cấp CPU để xử lý
- Kết thúc: tiến trình đã hoàn tất việc xử lý

5. Miền găng (Critical Section)

a) Vấn đề tranh chấp tài nguyên

Ta xét tình huống sau:

- giả sử A có 500$ trong tài khoản ngân hàng
- A quyết định rút ra 100$ từ tài khoản ngân hàng, thao tác của A gồm 2 bước:
  1) lấy ra 100$
  2) giảm số tài khoản đi 100$
- Tình huống giữa 2 thao tác 1 và 2, B trả A 300$, do vậy B cập nhật vào trong tài khoản của A là 800$ (=500$ +300$), sau đó A tiếp tục công việc 2, nó cập nhật lại trong tài khoản là 400$, như vậy B đã trả A 300$, nhưng A không nhận được.

b) Miền găng (Critical Section)

Đoạn chương trình trong đó có thể xẩy ra các mâu thuẫn truy xuất trên tài nguyên dụng chung được gọi là miền găng (Critical Section)

6. Khoá chết (deadlock)

Một tập các tiến trình được định nghĩa là ở trong tình trạng khoá chết nếu như, mỗi tiến trình trong tập hợp đều đều chờ đợi một số tài nguyên đang bị nắm

giữ bởi các tiến trình khác, như vậy không có tiến trình nào có thể tiếp tục xử lý, cũng như giải phóng tài nguyên cho các tiến trình khác sử dụng, tất cả các tiến trình trong tập hợp đều bị khoá vĩnh viễn!.

II. Lập trình đa tuyến trong Java

Với Java ta có thể xây dựng các chương trình đa luồng. Một ứng dụng có thể bao gồm nhiều luồng. Mỗi luồng được gán một công việc cụ thể, chúng được thực thi đồng thời với các luồng khác.

Có hai cách để tạo ra luồng

Cách 1: Tạo ra một lớp kế thừa từ lớp Thread và ghi đè phương thức run của lớp Thread như sau:

```
class MyThread extends Thread{
    public void run(){
    //Mã lệnh của tuyến
    }
}
```

Cách 2: Tạo ra một lớp triển khai từ giao diện Runnable, ghi đè phương thức run

```
class MyThread implements Runnable{
    public void run(){
    //Mã lệnh của tuyến
    }
}
```

1. Lớp Thread

Lớp Thread chứa phương thức tạo dựng Thread() cũng như nhiều phương thức hữu ích có chức năng chạy, khởi động, tạm ngừng, tiếp tục, gián đoạn và ngưng tuyến. Để tạo ra và chạy một tuyến ta cần làm 2 bước:

- Mở rộng lớp Thread và Ghi đè phương thức run()
- Gọi phương thức start() để bắt đầu thực thi tuyến

Lớp Thread không có nhiều phương thức lắm, chúng chỉ có một vài phương thức hữu dụng được liệt kê sau:

- ***public void run()***

được java gọi để thực thi tuyến thi hành, bạn phải ghi đè phương thức này để thực thi nhiệm vụ của tuyến, bởi vì phương thức run()của lớp Thread chỉ là phương thức rỗng

- ***public void start()***

  khi ta tạo ra tuyến nó chưa thực sự chạy cho đến khi, phương thức start() được gọi, khi start() được gọi thì phương thức run() cũng được kích hoạt

- ***public void stop()***

  có chức năng ngưng tuyến thi hành, phương thức này không an toàn, bạn nên gán null vào biến Thread để để dừng tuyến, thay vì sử dụng phương thức stop()

- ***public void suspend()***

  Có chức năng tạm ngừng tuyến, trong java 2, phương thức này ít được sử dụng, bởi vì phương thức này không nhả tài nguyên mà nó lắm giữ, do vậy có thể nguy cơ dẫn đến deadlock (khoá chết), bạn nên dùng phương thức wait(), để tạm ngừng tuyến thay vì sử dụng phương thức suspend()

- ***public void resume()***

  Tiếp tục vận hành tuyến nếu như nó đang bị ngưng, nếu tuyến đang thi hành thì phương thức này bị bỏ qua, thông thường phương thức này được dùng kết hợp với phương thức suspend(), kể từ java 2 phương thức này cùn với phương thức suspend()bị từ chối, do vậy bạn nên dùng phương thức notify () thay vì sử dụng phương thức resume()

- ***public static void sleep(long millis) Threadows InterruptedException***

  đặt tuyến thi hành vào trạng thái ngủ, trong khoảng thời gian xác định bằng mili giây. chú ý sleep() là phương thức tĩnh.

- ***public void interrupt()***
  làm gián đoạn tuyến thi hành
- ***public static boolean isInterrupt()***
  kiểm tra xem tuyến có bị ngắt không
- *public void setpriority(int p)*
  ấn định độ ưu tiên cho tuyến thi hành, độ ưu tiên được xác định là một số nguyên thuộc đoạn [1,10]
- ***public final void wait() throws InterruptException***
  đặt tuyến vào trạng thái chờ một tuyến khác, cho đến khi có một tuyến khác thông báo thì nó lại tiếp tục, đây là phương thức của lớp cơ sở Object
- ***public final void notify ()***
  đánh thức tuyến đang chờ, trên đối tượng này
- ***public final void notifyAll()*** đánh thức tất cả các tuyến đang chờ trên đối tượng này
- ***isAlive()*** Trả về True, nếu luồng là vẫn còn tồn tại (sống)
- ***getPriority()*** Trả về mức ưu tiên của luồng
- ***join()*** Đợi cho đến khi luồng kết thúc
- ***isDaemon()*** Kiểm tra nếu luồng là luồng một luồng chạy ngầm (deamon)
- setDeamon(boolean on) Đánh dấu luồng như là luồng chạy ngầm

ví dụ: ta tạo ra 2 tuyến thi hành song song, một tuyến thực hiện việc in 200 dòng "Đại học sư phạm kỹ thuật Hưng Yên", trong khi tuyến này đang thực thi thì có một tuyến khác vẫn tiếp tục in 200 dòng chữ "chào mừng bạn đến với java"

```
/**
 * <p>Title: </p>
 * <p>Description: Giao trinh ngon ngu lap trinh Java</p>
 * <p>Copyright: Copyright (c) 2004</p>
```

```
 * <p>Company: DHSPKT HY</p>
 * @author Hoang Trong The
 * @version 1.0
 */

public class Hello{
public static void main ( String[] args ){
        new ChaoDH ().start ();
        new ChaoJV ().start ();
}
}

class ChaoDH extends Thread{
public void run (){
        for ( int i = 1; i <= 200; i++ )
         System.out.println ( "Đại học sư phạm kỹ thuật Hưng Yên" );
}
}

class ChaoJV extends Thread{
public void run (){
        for ( int i = 1; i <= 200; i++ )
         System.out.println ( "chào mừng bạn đến với java" );
}
}
```

khi ta chạy chương trình thấy kết quả xen kẽ nhau như

```
        ……..
        Đại học sư phạm kỹ thuật Hưng Yên
```

Đại học sư phạm kỹ thuật Hưng Yên

chào mừng bạn đến với java

Đại học sư phạm kỹ thuật Hưng Yên

chào mừng bạn đến với java

Đại học sư phạm kỹ thuật Hưng Yên

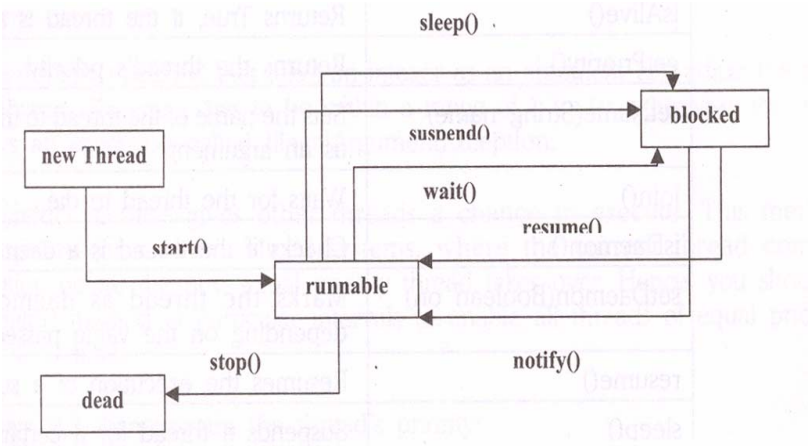
chào mừng bạn đến với java

chào mừng bạn đến với java

……...

2. Vòng đời của Thread

Hình sau thể hiện trạng thái của tuyến trong vòng đời của chúng



3. Luồng chạy ngầm (deamon)

Một chương trình Java kết thúc chỉ sau khi tất cả các luồng thực thi xong. Trong Java có hai loại luồng:

- Luồng người sử dụng
- Luồng chạy ngầm (deamon)

Người sử dụng tạo ra các luồng người sử dụng, trong khi các luồng deamon là các luồng chạy nền. Luồng deamon cung cấp các dịch vụ cho các luồng khác. Máy ảo Java thực hiện tiến trình thoát, khi đó chỉ còn duy nhất luồng deamon vẫn còn sống. Máy ảo Java có ít nhất một luồng deamon là luồng "garbage collection" (thu lượm tài nguyên - 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 làm việc với luồng deamon:

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

4. Giao diện Runnable

Ở mục trước bạn đã tạo ra các luồng thực hiện song song với nhau, trong java ta còn có thể tạo ra các tuyến thi hành song song bằng cách triển khai giao diện Runnable. Chắc bạn sẽ tự hỏi, đã có lớp Thread rồi tại sao lại còn có giao diện Runnable nữa, chúng khác gì nhau?, câu trả lời ở chỗ, java không hỗ trợ kế thừa bội, nếu chương trình của bạn vừa muốn kế thừa từ một lớp nào đó, lại vừa muốn đa tuyến thì bạn bắt buộc phải dùng giao diện Runnable, chẳng hạn như bạn viết các Applet, bạn vừa muốn nó là Applet, lại vừa muốn thực thi nhiều tuyến, thì bạn vừa phải kế thừa từ lớp Applet, nhưng nếu đã kế thừa từ lớp Applet rồi, thì bạn không thể kế thừa từ lớp Thread nữa.

Ta viết lại ví dụ trên, nhưng lần này ta không kế thừa lớp Thread, mà ta triển khai giao diện Runnable

```
public class Hello{
public static void main ( String[] args ){
        Thread t = new Thread ( new ChaoDH () );
        t.start ();
        Thread t1 = new Thread ( new ChaoJV () );
        t1.start ();
}
}

class ChaoDH implements Runnable{
```

```
public void run (){
        ChaoDH thu = new ChaoDH ();
        for ( int i = 1; i <= 200; i++ )        {
        System.out.println("Đại học sư phạm kỹ thuật Hưng Yên");
        }
}
}

class ChaoJV implements Runnable{
public void run (){
        for ( int i = 1; i <= 200; i++ )        {
         System.out.println ( "chào mừng bạn đến với java" );
        }
}
}
```

Cho chạy ví dụ này ta thấy kết quả ra không khác gì với ví dụ trước.

5. Thiết lập độ ưu tiên cho tuyến

Khi một tuyến được tạo ra, nó nhận một độ ưu tiên mặc định, đôi khi ta muốn điều chỉnh độ ưu tiên của tuyến để đạt được mục đích của ta, thật đơn giản, để đặt độ ưu tiên cho một tuyến ta chỉ cần gọi phương thức setPriority() và truyền cho nó một số nguyên số này chính là độ ưu tiên mà bạn cần đặt.

Ta viết lại ví dụ trên như sau: Thêm vào phương thức main() 2 dòng lệnh:

```
t.setPriority(1);//Tuyến này có độ ưu tiên là 1
t1.setPriority(10);// Tuyến này có độ ưu tiên là 1
```

Chạy lại chương trình này sau khi sửa và trước khi sửa ta thấy tuyến t1 được cấp thời gian sử dụng CPU nhiều hơn tuyến t, lý do là ta đã đặt độ ưu tiên của tuyến

t1, lớn hơn độ ưu tiên của tuyến t

Chú ý:

1) độ ưu tiên của một tuyến biểu thị bởi một số nguyên nằm trong đoạn từ 1 đến 10, một lỗi sẽ phát sinh nếu ta gán cho nó độ ưu tiên, nằm ngoài khoảng này
2) 2) nếu một tuyến không được đặt độ ưu tiên thì nó sẽ nhận độ ưu tiên mặc định (bằng 5), ta có thể kiểm tra điều này bằng cách gọi phương thức getPriority()

6. Nhóm tuyến (Thread Group)

- Nhóm tuyến là một tập hợp gồm nhiều tuyến, khi ta tác động đến nhóm tuyến (chẳng hạn như tạm ngưng, …) thì tất cả các tuyến trong nhóm đều nhận được cùng tác động đó, điều này là tiện lợi khi ta muốn quản lý nhiều tuyến thực hiện các tác vụ tương tự nhau.

- Để tạo một nhóm tuyến ta cần:

  + tạo ra một nhóm tuyến bằng cách sử dụng phương thức tạo dựng của lớp *ThreadGroup()*

  ThreadGroup g=new ThreadGroup("ThreadGroupName");

  ThreadGroup g=

  new ThreadGroup(ParentThreadGroup,"ThreadGroupName");

  Dòng lệnh trên tạo ra một nhóm tuyến g có tên là "ThreadGroupName", tên của tuyến là một chuỗi và không trùng với tên của một nhóm khác.

  + đưa các tuyến vào nhóm tuyến dùng phương thức tạo dựng của lớp Thread() :

  Thread =new Thread (g, new ThreadClass(),"ThisThread");

7. Đồng bộ các tuyến thi hành

Khi nhiều tuyến truy cập đồng thời vào tài nguyên dùng chung, mà tài nguyên này lại không thể chia sẻ, cho nhiều tuyến, khi đó tài nguyên dùng chung

có thể bị hỏng. 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.

Trong những trường hợp này, bạn cần cho phép một luồng hoàn thành trọn vẹn tác vụ của nó, và rồi thì mới cho phép các luồng kế tiếp thực thi. Khi hai hoặc nhiều hơn một 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.

Bởi trong java không có biến toàn cục, chúng ta chỉ có thuộc tính của đối tượng, tất cả các thao tác có thể dẫn đến hỏng hóc đều thực hiện qua phương thức, do vậy java cung cấp từ khoá ***synchronized***, từ khoá này được thêm vào định nghĩa của phương thức báo cho java biết đây là một phương thức đồng bộ, mỗi đối tượng sẽ có một bộ quản lý khoá, bộ quản lý khoá này chỉ cho 1 phương thức ***synchronized*** của đối tượng đó chạy tại một thời điểm

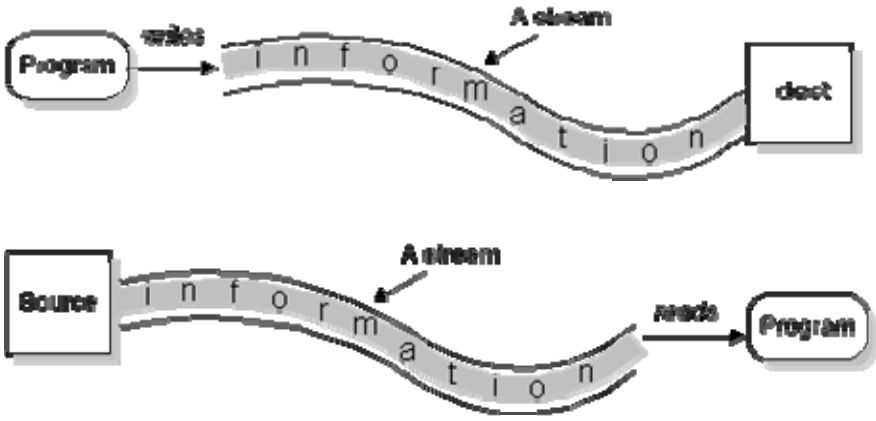
Mấu chốt của sự đồng bộ hóa là khái niệm “monitor” (giám sát), hay còn gọi “semaphore” (cờ hiệu). Một “monitor” là một đối tượng mà được khóa độc quyền. Chỉ một luồng có thể có monitor tại mỗi thời điểm. Tất cả các luồng khác cố gắng thâm nhập vào monitor sẽ bị trì hoãn, cho đến khi luồng đầu tiên thoát khỏi monitor. Các luồng khác được báo chờ đợi monitor. Một luồng có thể monitor một đối tượng nhiều lần.

Chương 5

NHẬP XUẤT

Một chương trình thường xuyên làm việc với dữ liệu, để có thể lưu trữ lâu dài chúng ta phải lưu trữ và nhận lại dữ liệu từ thiết bị lưu trữ ngoài, nguồn thông tin ngoài không chỉ gồm dữ liệu được lưu trữ trên đĩa từ, đĩa CD mà nó có thể là dữ liệu của một chương trình khác, hoặc có thể là được lưu trữ trên mạng… dù chúng được lưu trữ ở đâu chúng cũng chỉ có 1 số dạng như: đối tượng, kí tự, hình ảnh hoặc âm thanh, dù dữ liệu được lưu trữ dưới hình thức nào, lưu trữ ở đâu thì java đều trừu tượng hoá thành các luồng, điều này là rất tinh vi nó làm cho ta không cần phải quan tâm dữ liệu được lưu trữ ở đâu, dưới dạng thức như thế nào, nó đồng nhất mọi nguồn dữ liệu với nhau:

Để nhận về các thông tin, một chương trình mở một luồng liên kết với đối tượng nguồn(tệp tin, bộ nhớ, Socket) và đọc các thông tin tuần tự.
Tương tự để ghi thông tin ra các thiết bị ngoài bằng cách mở một luồng đến đối tượng đích và ghi thông tin ra một cách tuần tự như



Luồng là sự trừu tượng hoá ở mức cao, do vậy bất kể dữ liệu được đọc vào từ đâu hoặc ghi ra đâu, thì thuật toán đọc/ghi tuần tự đều tựa như sau:
Đọc vào

open a stream
while more information
read information
close the stream
Ghi ra
open a stream
while more information
write information
close the stream

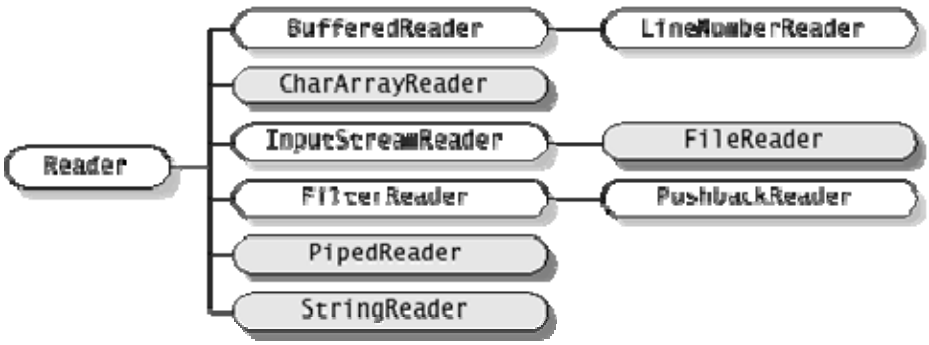
<u>Lớp luồng</u>

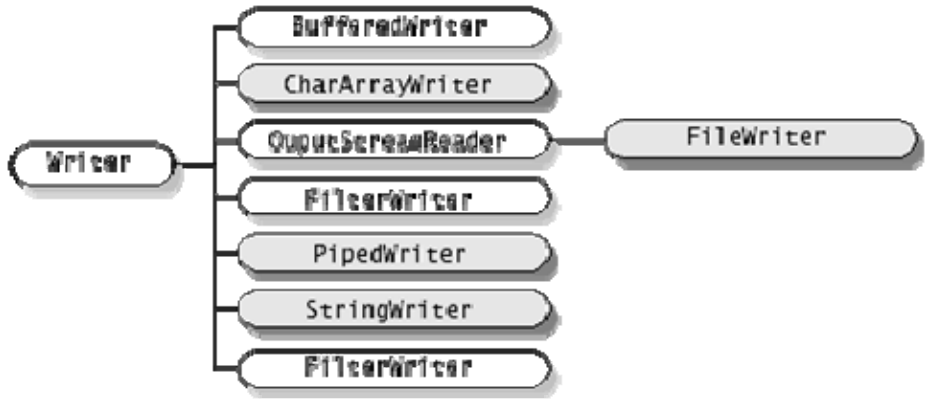
Java đưa ra nhiều lớp luồng, để xử lý mọi loại dữ liệu, java chia luồng ra thanh 2 loại: luồng byte (byte stream) và luồng kí tự (character stream), lớp InputStream và OutputStream là hai lớp cơ sở cho mọi luồng nhập xuất hướng byte, và lớp Reader/ Writer là hai lớp cơ sở cho việc đọc ghi hướng kí tự.

Lớp RandomAccessFile kế thừa từ lớp Object và triển khai giao diện, InputStream và OutputStream, đây là lớp duy nhất hỗ trợ cả đọc lẫn ghi.

<u>Lớp nhập, xuất hướng kí tự</u>

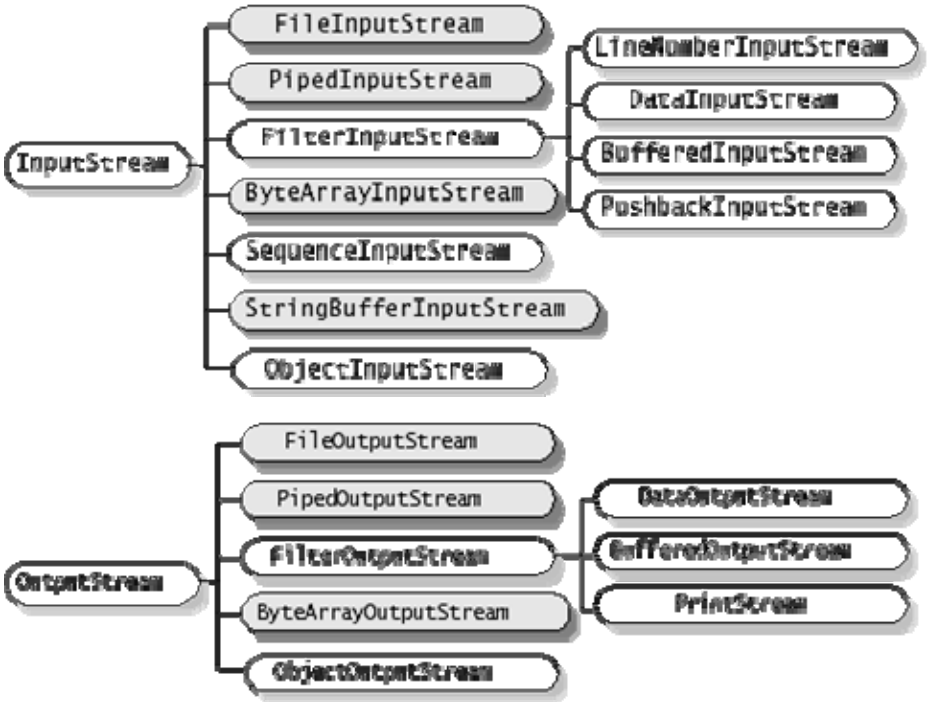
Reader và Writer là hai lớp cơ sở trừu tượng cho luồng hướng kí tự, hai lớp này cung cấp một giao diện chung cho tất cả các lớp đọc/ ghi hướng kí tự, mỗi lần đọc/ ghi ra luồng là đọc 2 byte tương ứng với một kí tự unicode, Sau đay là mô hình phân cấp các lớp đọc/ ghi hướng kí tự





<u>Luồng hướng byte</u>

Để có thể đọc ghi 1 byte, ta phải sử dụng luồng hướng byte, hai lớp InputStream và OutputStream là hai lớp cơ sở trừu tượng cho các luồng hướng byte, mỗi lân đọc/ ghi ra luồng là đọc/ ghi 8 bit dữ liệu ra luồng, Hình sau thể hiện mối quan hệ phân cấp giữa lớp đọc/ ghi hướng byte



Sự tương tự giữa hai luồng hướng byte và hướng kí tự

Lớp Reader và InputStream có một giao diện giống nhau, chúng chỉ khác nhau về

kiểu dữ liệu đọc vào, ví dụ lớp Reader có các phương thức sau giúp cho việc đọc một kí tự hoặc một mảng các kí tự

int read()

int read(char cbuf[])

int read(char cbuf[], int offset, int length)

thì trong lớp InputStream cũng có các phương thức với tên tương tự cho việc đọc một byte hoặc một mảng các byte

int read()

int read(byte cbuf[])

int read(byte cbuf[], int offset, int length)

Cũng tương tự vậy lớp Writer và OutputStream cũng có một giao diện tương tự nhau, ví dụ lớp Writer định nghĩa các phương thức để ghi một kí tự, một mảng các kí tự ra luồng

int write(int c)

int write(char cbuf[])

int write(char cbuf[], int offset, int length)

thì lớp OutputStream cũng có các phương thức tương ứng, để ghi một byte, một mảng byte ra luồng

int write(int c)

int write(byte cbuf[])

int write(byte cbuf[], int offset, int length)

Xử lý tệp tin

Để xử lý tệp tin ngoại trú, ta sử dụng các luồng liên quan đến tệp tin như FileInputStream và FileOutputStream cho việc đọc ghi tệp hướng byte, FileReader và FileWriter cho việc đọc ghi hướng kí tự, thông thường muốn sử dụng luồng tệp tin ta sử dụng hàm tạo của các lớp tương ứng để liên kết luồng với một tệp tin cụ thể.

public void FileInputStream (String FileName)
public void FileInputStream (File file)
public void FileOutputStream (String FileName)
public void FileOutputStream (File file)
public void FileWriter (String FileName)
public void FileWriter (File file)
public void FileReader (String FileName)
public void FileReader (File file)

Ví dụ: viết chương trình file copy, thực hiện việc copy một tệp, ta sẽ viết chương trình này sử dụng cả 2 luồng hướng byte và hướng kí tự

```
import java.io.*;

// chương trình copy sử dụng luồng hướng kí tự
public class CopyCharacter {
public static void main(String[] args) throws IOException {
        File inputFile = new File("C:/in.txt");
        File outputFile = new File("C:/out.txt");

FileReader in = new FileReader(inputFile);
FileWriter out = new FileWriter(outputFile);
int c;

while ((c = in.read())! = -1)
out.write(c);

in.close();
out.close();
}
```

```
}

import java.io.*;

// Chương trình copy sử dụng luồng hướng byte
public class CopyBytes {
public static void main(String[] args) throws IOException {
File inputFile = new File("farrago.txt");
File outputFile = new File("outagain.txt");

FileInputStream in = new FileInputStream(inputFile);
FileOutputStream out = new FileOutputStream(outputFile);
int c;

while ((c = in.read())! = -1)
out.write(c);

in.close();
out.close();
}
}
```

Luồng dữ liệu

Để đọc/ ghi các kiểu dữ liệu nguyên thuỷ, ta sử dụng luồng DataInputStream và DataOutputStream, lớp DataInputStream triển khai giao diện DataInput, còn lớp DataOutputStream triển khai giao diện DataOuput

Các phương thức sau được định nghĩa trong giao diện DataOutput

void write(byte[] b)	Ghi một mảng byte ra luồng
void write(byte[] b, int off, int len)	Ghi một mảng byte ra luồng kể từ vị trí off, len byte
void write(int b)	Ghi một byte ra luồng
void writeBoolean(boolean v)	Ghi một giá trị logic ra luồng
void writeByte(int v)	Ghi ra luồng phần thấp của v
void writeBytes(String s)	Ghi một xâu ra luồng
void writeChar(int v)	Ghi một kí tự ra luồng
void writeChars(String s)	Ghi một xâu kí tự ra luồng
void writeDouble(double v)	Ghi một số double ra luồng
void writeFloat(float v)	Ghi một số thực ra luồng
void writeInt(int v)	Ghi một số nguyên ra luồng
void writeLong(long v)	Ghi một số long ra luồng
void writeShort(int v)	Ghi một số short ra luồng
void writeUTF(String str)	Chi một xâu kí tự Unicode ra luồng

Các phương thức sau được định nghĩa trong giao diện DataInput:

boolean readBoolean()	đọc một giá trị logic từ luồng
byte readByte()	đọc một byte từ luồng
char readChar()	đọc một kí tự từ luồng
double readDouble()	đọc một số double từ luồng
float readFloat()	đọc một số float từ luồng
void readFully(byte[] b)	đọc một mảng byte từ luồng và ghi vào mảng
voidreadFully(byte[] b, int off, int len)	đọc len byte từ luồng và ghi vào mảng từ vị trí off
int readInt()	đọc một số nguyên

String readLine()	đọc một xâu kí tự cho đến khi gặp kí tự xuống dòng và bỏ qua kí tự xuống dòng
long readLong()	đọc một số long
short readShort()	đọc một số short
int readUnsignedByte()	đọc một số nguyên không dấu trong khoảng 0..255
in treadUnsignedShort()	đọc một số nguyên không dấu trong đoạn từ 0..65535
String readUTF()	đọc một xâu kí tự Unicode
int skipBytes(int n)	Bỏ qua n byte từ luồng

Sau đây là một ví dụ nhỏ về luồng nhập xuất dữ liệu, ví dụ này ghi dữ liệu ra tệp rồi lại đọc lai:

```
import java.io.*;

public class DataIODemo {
public static void main(String[] args) throws IOException {

// write the data out
DataOutputStream out = new DataOutputStream(new
FileOutputStream(“c:/TestIO.txt"));
// ghi số nguyên
out.writeInt(10);
// ghi số long
out.writeLong(123456789);
// ghi số thực chính xác kép
out.writeDouble(123.456789);
```

```
// ghi số thực chính xác đơn
out.writeFloat(123.456789f);
// ghi giá trị logic
out.writeBoolean(true);
// ghi một xâu
out.writeUTF("Day la mot xau ki tu");
out.close();
// read it in again
DataInputStream in = new DataInputStream(new
FileInputStream("c:/TestIO.txt"));

try {
// đọc lại số nguyên
System.out.println("Gia tri nguyen " + in.readInt());
// đọc lại số nguyên dài
System.out.println("Gia tri long " + in.readLong());
// đọc lại số thực chính xác kép
System.out.println("Gia tri double " + in.readDouble());
// đọc lại số thực chính xác đơn
System.out.println("Gia tri float " + in.readFloat());
        // đọc lại giá trị logic
System.out.println("Gia tri boolean " + in.readBoolean());
        // đọc lại một xâu unicode
System.out.println("Gia tri xau " + in.readUTF());
}

catch (EOFException e) {
System.out.println("loi");
}
```

```
in.close();
}
}
```

Luồng in ấn

Vì các luồng xuất ghi dữ liệu ra dưới dạng nhị phân do vậy bạn không thể dùng lệnh type, hoặc các chương trình soạn thảo asciii để xem được, trong java có thể sử dụng luồng in ấn để xuất dữ liệu ra dưới dạng asciii. Lớp PrintStream và PrintWriter sẽ giúp ta làm việc này. Hai lớp này thực hiện chức năng như nhau, đều xuất ra dữ liệu dạng asciii.

Một số phương thức của lớp PrintStream:

boolean checkError()	dồn hết dữ liệu ra và kiểm tra lỗi luồng
void close()	đóng luồng
void flush()	dồn dữ liệu trong vùng đệm ra
void print(boolean b)	ghi giá trị logic ra luồng
void print(char c)	ghi kí tự
void print(char[] s)	ghi một mange kí tự
void print(double d)	ghi một số thực độ chính xác kép
void print(float f)	ghi một số thực
void print(int i)	ghi một số nguyên
void print(long l)	ghi một số nguyên dài
void print(Object obj)	ghi một đối tượng
void print(String s)	ghi một xâu
void println()	tạo ra một dòng trống
void println(boolean x)	ghi giá trị logic ra luồng và xuống dòng
void println(char x)	ghi kí tự và xuống dòng
void println(char[] x)	ghi một mange kí tự và xuống dòng

void println(double x)	ghi một số thực độ chính xác kép và xuống dòng
void println(float x)	ghi một số thực và xuống dòng
void println(int x)	ghi một số nguyên và xuống dòng
void println(long x)	ghi một số nguyên dài và xuống dòng
void println(Object x)	ghi một đối tượng và xuống dòng
void println(String x)	ghi một xâu và xuống dòng
protected void setError()	đặt trạng thái lỗi của luồng là true
void write(byte[] buf, int off, int len)	ghi mảng byte từ vị trí off len kí byte ra luồng
void write(int b)	ghi một byte ra luồng

Hàm tạo của lớp PrintStream:

PrintStream(OutputStream out) tạo ra một luồng mới

PrintStream(OutputStream out, boolean autoFlush) tạo ra một luồng mới với chức năng AutoFlush (tự dồn)

Một số phương thức của lớp PrintWriter

boolean checkError()	dồn hết dữ liệu ra và kiểm tra lỗi luồng
void close()	đóng luồng
void flush()	dồn dữ liệu trong vùng đệm ra
void print(boolean b)	ghi giá trị logic ra luồng
void print(char c)	ghi kí tự
void print(char[] s)	ghi một mange kí tự
void print(double d)	ghi một số thực độ chính xác kép
void print(float f)	ghi một số thực
void print(int i)	ghi một số nguyên
void print(long l)	ghi một số nguyên dài

void print(Object obj)	ghi một đối tượng
void print(String s)	ghi một xâu
void println()	tạo ra một dòng trống
void println(boolean x)	ghi giá trị logic ra luồng và xuống dòng
void println(char x)	ghi kí tự và xuống dòng
void println(char[] x)	ghi một mange kí tự và xuống dòng
void println(double x)	ghi một số thực độ chính xác kép và xuống dòng
void println(float x)	ghi một số thực và xuống dòng
void println(int x)	ghi một số nguyên và xuống dòng
void println(long x)	ghi một số nguyên dài và xuống dòng
void println(Object x)	ghi một đối tượng và xuống dòng
void println(String x)	ghi một xâu và xuống dòng
protected void setError()	đặt trạng thái lỗi của luồng là true
void write(byte[] buf, int off, int len)	ghi mảng byte từ vị trí off len kí byte ra luồng
void write(int b)	ghi một byte ra luồng
void write(int c)	Ghi một kí tự đơn
void write(String s)	Ghi một xâu
void write(String s, int off, int len)	Ghi một xâu len kí tự tính từ vị trí off

Các hàm tạo của lớp PrintWriter

- PrintWriter(OutputStream out) tạo ra một PrintWriter không có chức năng tự dồn từ một đối tượng OutputStream.
- PrintWriter(OutputStream out, boolean autoFlush) tạo ra một PrintWriter với chức năng tự dồn từ một đối tượng OutputStrea.
- PrintWriter(Writer out) tạo ra một PrintWriter không có chức năng tự dồn từ một đối tượng Writer

- PrintWriter(Writer out, boolean autoFlush) tạo ra một PrintWriter với chức năng tự dồn từ một đối tượng Writer

Sau đây là một ví dụ về luồng in ấn, ví dụ này in ra một tệp một số nguyên, một số thực và một xâu kí tự, sau khi chạy chương trình bạn có thể sử dụng lệnh type của DOS để xem

```
import java.io.*;

public class DataIODemo1 {
public static void main(String[] args) throws IOException {

// write the data out
PrintWriter out = new PrintWriter(new FileOutputStream("c:/a.txt"));
out.println(10);
out.println(1.2345);
out.print("xau ki tu");
out.close();
}
}
```

Luồng đệm

Vì các thao tác với ổ cứng, mạng thường lâu hơn rất nhiều so các thao tác với bộ nhớ trong, do vậy chúng ta cần phải có một kỹ thuật nào đó để tăng tốc độ đọc/ghi, kỹ thuật đó chính là vùng đệm, với vùng đệm ta sẽ giảm được số lần đọc ghi luồng, trong java ta có thể tạo ra vùng đệm với các lớp BufferInputStream, BufferOutputStream, BufferReader, BufferWriter, thông thường bạn sẽ nối các luồng của bạn vào luồng đệm.

Các phương thức tạo dựng luồng đệm:

public BufferInputStream(InputStream)

public BufferInputStream (InputStream in, int bufferSize)

public BufferOutputStream (OutputStream out)

public BufferOutputStream (OutputStream out, int bufferSize)

public BufferReader (Reader in)

public BufferReader (Reader in, int bufferSize)

public BufferWriter (Writer out)

public BufferWriter (Writer out, int bufferSize)

Tệp tin truy cập ngẫu nhiên

Tất cả các luồng xét trên chỉ có thể đọc, hoặc ghi, chúng không thể đọc ghi đồng thời, chỉ duy nhất có một lớp cho phép ta đọc ghi đồng thời, đó là lớp RandomAccessFile, lớp này triển khai giao diện InputData và OutputData, nên chúng có tất cả các phương thức của cả 2 lớp này, ngoài ra chúng còn có các phương thức sau:

- public void seek(long pos) chuyển con trỏ đến vị trí pos tính từ vị trí đầu tiên, chú ý vị trí đầu tiên tính từ 0
- public long getFilePointer() trả về vị trí con trỏ tệp tính bằng byte, kể ta đầu tệp
- public long length() trả về độ dài của tệp
- public void writeChar(int v) ghi kí tự unicode ra tệp với byte cao được ghi trước
- public final void writeChars(String s) ghi một xâu kí tự ra tệp

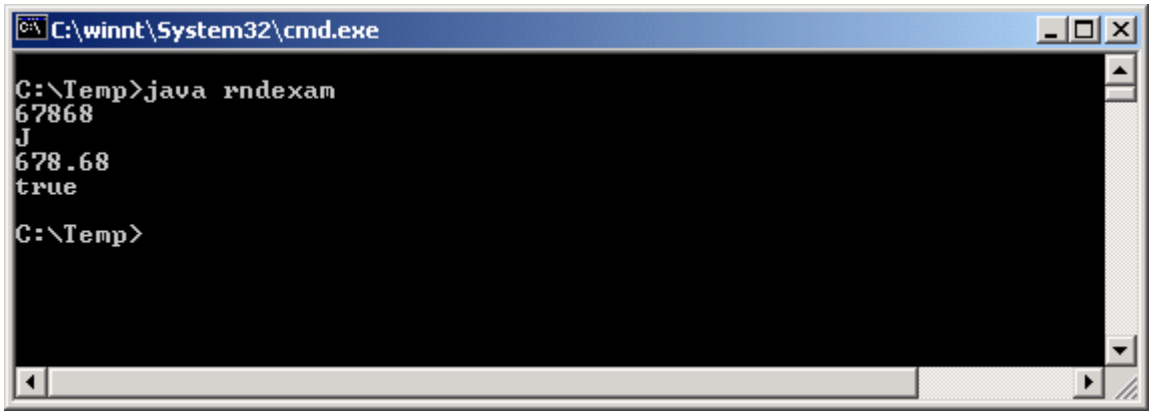
Tương tự giống C/C++ khi bạn mở một tệp truy cập ngẫu nhiên bạn phải chỉ rõ chế độ làm việc là đọc ‘r’, hay ghi ‘w’ hay đọc ghi đồng thời ‘rw’, ví dụ như bạn muốn mở tệp a.txt theo chế độ đọc ghi đồng thời thì bạn dùng cú pháp RandomAccessFile =new RandomAccessFile(“C:/ a.txt”, **“rw”**)

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 bên trong tập tin (bắt đầu từ 1).

Sau đó nó đọc giá trị số nguyên, ký tự và double từ tập tin và hiển thị chúng ra màn hình.

```
import java.lang.System;
import java.io.RandomAccessFile;
import java.io.IOException;

public class rndexam{
        public static void main (String args[ ]) throws IOException     {
                RandomAccessFile rf;
                rf = new RandomAccessFile("abc.txt", "rw");
                rf.writeBoolean(true);
                rf.writeInt(67868);
                rf.writeChars("J");
                rf.writeDouble(678.68);

                //Sử dụng phương thức seek() để di chuyển con trỏ đến byte thứ hai
                rf.seek(1);
                System.out.println(rf.readInt());
                System.out.println(rf.readChar());
                System.out.println(rf.readDouble());
                rf.seek(0);
                System.out.println(rf.readBoolean));
                rf.close();
        }
}
```

Kết quả xuất ra của chương trình:



Lớp File

Lớp File cung cấp giao diện chung để xử lý hệ thống tệp độc lập với môi trường của các máy tính. Một ứng dụng có thể sử dụng các chức năng chính của *File để xử lý tệp hoặc các thư mục (directory) trong hệ thống tệp.* Để xử lý các nội dung của các tệp thì sử dụng các lớp FileInputStream, *FileOutputStream và RandomAccessFile.*

Lớp File định nghĩa các thuộc tính phụ thuộc vào môi trường (platform) được sử dụng để xử lý tệp và tên gọi các đường dẫn trong các thư mục một cách độc lập với môi trường.

public static final char separatorChar

public static final String separator

Định nghĩa các ký hiệu hoặc các xâu sử dụng để ngăn cách các thành phần trong tên của đường dẫn. Ký hiệu '/' để ngăn cách cho Unix, '\' được sử dụng để ngăn cách các mục của đường dẫn trong Window.

Ví dụ: C:\book\Java là tên đường dẫn trong Window.

public static final char pathSeparatorChar

public static final String pathSeparator

Định nghĩa các ký hiệu hoặc các xâu sử dụng để ngăn cách các tệp hoặc tên thư mục trong danh sách của các đường dẫn. Ký hiệu ngăn cách ':' cho Unix, ';' được sử dụng để phân cách các đường dẫn trong Window.

Ví dụ: C:\book; C:\Java; D:\Anh\ABC; A:\Docs là danh sách các đường dẫn trong Window.

File(String pathName)

Gán đường dẫn có tên pathName cho đối tượng của lớp File. pathName có thể là đường dẫn tuyệt đối (có đủ cả tên ổ đĩa, tên của tất cả các mục lần lượt theo cây thư mục) hoặc đường dẫn tương *đối (bắt đầu từ thư mục hiện thời).*

Ví dụ:

File ch1 = new File(File.separator + "book" + File.separator + "chuong1");

File(File direct, String filename)

Gán tệp có tên filename ở thư mục direct cho đối tượng của lớp File. Nếu *direct là null thì filename được xử lý ở thư mục hiện thời,* ngược lại tên đường dẫn sẽ là đối tượng direct ghép với separator và filename.

Ví dụ:

File mucNhap = new File("book" + File.separator + "duThao");

File ch2 = new File(mucNhap, "chuong1");

Lớp File có thể sử dụng để truy vấn vào các hệ thống tệp để biết được các thông tin về tệp và các thư mục.

Lớp File cung cấp các hàm để nhận được các biểu diễn phụ thuộc vào môi trường của đường dẫn và các thành phần của đường dẫn.

String getName()

Hàm getName() cho lại tên của tệp trong thư mục chỉ định. Ví dụ, tên của "C:\java\bin\javac" là "javac".

String getPath()

Hàm getPath() cho lại tên của đường dẫn (tuyệt đối hoặc tương đối) chứa tệp chỉ định.

String getAbsolutePath()

Hàm getAbsolutePath() cho lại tên của đường dẫn tuyệt đối chứa tệp chỉ định.

long lastModified()

Hàm này cho lại thời gian dạng số long của lần sửa đổi tệp lần cuối cùng.

long length()

Hàm này cho lại kích thước của tệp tính theo byte.

boolean equals(Object obj)

Hàm này so sánh tên các đường dẫn của các đối tượng tệp, cho kết quả true nếu chúng đồng nhất với nhau.

boolean exists()

boolean isFile()

Hai hàm này cho kết quả true nếu tệp đó tồn tại trên đường dẫn hiện thời.

boolean isDirectory()

Hàm này cho kết quả true nếu thư mục đó tồn tại trên ổ đĩa hiện thời.

boolean createNewFile() throws IOException

Một tệp mới là rỗng được tạo ra ở thư mục hiện thời chỉ khi tệp này chưa có.

boolean mkdir()

boolean mkdirs()

Tạo ra các đường dẫn được xác định trong đối tượng của File.

boolean renameTo(File dest)

Hàm này đổi tên tệp hiện thời thành dest.

String[] list()

Hàm này hiển thị danh sách các tệp ở thư mục hiện thời.

boolean delete()

Hàm này xoá tệp hiện thời.

Ví dụ 8.1 Viết chương trình để đọc và hiển thị các tệp trong một thư mục.

```
import java.io.*;
public class DirectoryLister{
 public static void main(String args[]){
 File entry;
 if (args.length==0){
```

```
            System.err.println("Hay cho biet ten duong dan?");
            return;
    }
    entry = new File(args[0]);// Tạo ra đối tượng của File lấy tên từ agrs[0]
    listDirectory(entry);  // Hiển thị các mục trong danh mục chỉ định
  }
  public static void listDirectory(File entry) {
    try {
            if(!entry.exists()) {
                    System.out.println(entry.getName() +"not found.");
                    return;
            }
            if (entry.isFile()){
                    // Nếu đối số của chương trình là một tệp thì hiển thị tệp đó
                    System.out.println(entry.getCanonicalPath());
            } else if(entry.isDirectory()){
                    // Nếu đối số của chương trình là một danh mục thì đọc ra
(list())
                    String[] fileName = entry.list();
                    if (fileName ==null)return;
                    for (int i = 0; i<fileName.length; i++){
                            // Tạo ra đối tượng của File để xử lý từng mục
                            File item = new File(entry.getPath(), fileName[i]);
                            //  Gọi đệ qui hàm listDirectory()để hiển thị từng tệp.
                            listDirectory(item);
                    }
            }
    }catch(IOException e){System.out.println("Error:" +e);
      }
```

```
  }
}
```

Dịch xong có thể chạy trong môi trường DOS:

```
 java DirectoryLister  c:\users\lan
```

Tất cả các tệp trong danh mục c:\users\lan sẽ được hiện lên.

Chương 6

LẬP TRÌNH ĐỒ HOẠ

Một khả năng của Java là cho phép ta xây dựng các ứng dụng có giao diện đồ hoạ hay còn gọi là GUI (Grapthical User Interface). Khi Java được phát hành, các thành phần đồ hoạ được tập trung vào thư viện mang tên Abstract Window Toolkit (AWT). Đối với mỗi hệ nền, thành phần AWT sẽ được ánh xạ sang một thành phần nền cụ thể, bằng cách sử dụng trực tiếp mã native của hệ nền, chính vì vậy nó phụ thuộc rất nhiều vào hệ nền và nó còn gây lỗi trên một số hệ nền. Với bản phát hành Java 2, các thành phần giao diện được thay bằng tập hợp các thành phần linh hoạt, đa năng, mạnh mẽ, độc lập với hệ nền thuộc thư viện Swing. Phần lớn các thành phần trong thư viện Swing đều được tô vẽ trược tiếp trên canvas bằng mã lệnh của Java, ngoại trừ các thành phần là lớp con của lớp java.awt.Window hoặc Java.awt.Panel vốn phải được vẽ bằng GUI trên nền cụ thể. Thành phần Swing ít phụ thuộc vào hệ nền hơn do vậy ít gặp lỗi hơn và đặc biệt nó sử dụng ít tài nguyên của hệ thống hơn các thành phần trong thư viện awt. Mặc dù các thành phần awt vẫn được hỗ trợ trong Java 2 nhưng, tuy nhiên Sun khuyên bạn nên sử dụng các thành phần Swing thay cho các thành phần awt, tuy nhiên các thành phần trong thư viện Swing không thể thay tất cả các thành phần trong thư viện awt. Chúng chỉ thay thế một phần của awt như: Button, Panel, TextFeild, v.v. Còn các lớp trợ giúp khác trong awt như : Graphics, Color, Font, FontMetrics, v.v. vẫn không thay đổi. Bên cạnh đó các thành phần Swing còn sử dụng mô hình sử lý sự kiện của awt.

I. Giới thiệu về hệ thống đồ hoạ của Java

1. Giới thiệu chung

Thiết kế API cho lập trình đồ hoạ của Java là một ví dụ hoàn hảo về cách dùng

lớp, sự kế thừa và giao diện. API cho lập trình độ hoạ bao gồm một tập rất nhiều lớp nhằm trợ giúp xây dựng các thành phần giao diện khác nhau như: cửa sổ,nút ấn, ô văn bản, menu, hộp kiểm, v.v. Mối quan hệ kế thừa giữa các thành phần này được mô tả trong hình sau:

1. Componient Đây là lớp (trừu tượng) cha của mọi lớp giao diện người dùng. Lớp này cung cấp các thuộc tính, hành vi cơ bản nhất của tất cả các thành phần giao diện.
2. Container Là một vật chứa dùng để ghép nhóm các thành phần giao diện khác. Mỗi vật chứa có một lớp quản lý hiển thị, lớp quản lý hiển thị có trách nhiệm bố trí cách thức hiển thị các thành phần bên trong. Hai vật chứa hay được sử dụng nhất la JFrame và JPanel.
3. Jcomponient Là lớp cha của mọi thành phần Swing ligth weight, được vẽ trực tiếp lên canvas bằng mã lệnh Java.
4. Window Được sử dụng để tạo ra một cửa sổ, Thông thường ta hay sử dụng hai lớp con của nó là JFrame và JDialog.
5. JFrame là cửa sổ không lồng bên trong cửa sổ khác.
6. JDialog là một cửa sổ được hiển thị dưới dạng modal.
7. JAplet là lớp cha của mọi lớp ứng dụng aplet.
8. JPanel là một vật chứa, lưu giữ các thành phần giao diện người dùng.
9. Graphics là lớp trừu tượng cung cấp ngữ cảnh đồ hoạ để vẽ các đối tượng đồ hoạ như: Đường thẳng, đường tròn, hình ảnh…
10. Color lớp này biểu diễn một mầu sắc.
11. Font lớp này biểu thị cho một font đồ hoạ.
12. FontMetrics là một lớp trừu tượng dùng để xác định các thuộc tính của Font.

Tất cả các thành phần đồ hoạ trong thư viện Swing được nhóm trong gói javax.swing. Đa số các thành phần trong thư viện Swing đều có tiếp đầu ngữ là ‘J’, Ví dụ một nút lệnh trong thư viện Swing có tên là JButton, một memu có tên là

JMenu.

Tất cả những lớp khác được liệt kê trong hình dưới đây

Chú ý: Đừng pha trộn các thành phần giao diện swing và awt trong cùng một ứng dụng. Chẳng hạn như đừng nên đặt một JButton vào một Panel và đừng nên đặt Button vào JPanel. Việc làm này có thể gây lỗi.

Một lớp được kế thừa từ lớp JComponent được thể hiện trong hình sau:

2. Một số phương thức của lớp Component

Lớp Component cung cấp các thuộc tính, phương thức chung cho các lớp con của nó. Sau đây là một số phương thức của lớp ***Component*** :

- *Dimension getSize():*cho lại đối tượng thuộc lớp Dimension gồm width (chiều rộng), *height (chiều cao) xác định kích thước của một thành phần* tính theo pixel.
- void setSize(int width, int height) và void setSize(Dimension d) đặt lại kích thước của thành phần.

- *Point getLocation():* cho lại tọa độ (kiểu Point) trên cùng bên trái (tọa độ gốc) của thành phần đang xét.
- *void setLocation(int x, int y) và void setLocation(Point p)* đặt lại các tọa độ được chỉ định cho một thành phần.
- *Rectangle getBounds():*cho lại đường biên là hình chữ nhật Rectangle bao gồm

tọa độ gốc và chiều dài, chiều rộng của hình chữ nhật.

- void setBounds(int x, int y) và void setBounds(Rectangle r):đặt lại đường biên cho một thành phần.
- *void setForeground(Color c):*được sử dụng để đặt màu vẽ cho thành phần đồ họa
- *void setBackground(Color c):đặt màu nền cho thành phần đồ họa. Các* tham số của hai hàm này là đối tượng của lớp Color sẽ được giới thiệu ở phần sau.
- Font getFont(): được sử dụng để biết được font của các chữ đang xử lý trong thành phần đồ họa.
- *void setFont(Font f):*đặt lại font chữ cho một thành phần.
- *void setEnabled(boolean b):*Nếu đối số b của hàm getEnabled() là true thì thành phần đang xét hoạt động bình thường, nghĩa là có khả năng kích hoạt (enable), có thể trả lời các yêu cầu của người sử dụng và sinh ra các sự kiện như mong muốn. Ngược lại, nếu là false thì thành phần tương ứng sẽ không kích hoạt được, nghĩa là không thể trả lời được các yêu cầu của người sử dụng.
- Lưu ý: Tất cả các thành phần giao diện khi khởi tạo đều được kích hoạt
- *void setVisible(boolean b):*Một thành phần đồ họa có thể được hiển thị lên màn hình (nhìn thấy được) hoặc bị che giấu tùy thuộc vào đối số của hàm setVisible() là true hay false.

3. Lớp Container

Lớp Container là lớp con của lớp trừu tượng Component. Các lớp chứa (lớp con của Container) cung cấp tất cả các chức năng để xây dựng các giao diện đồ họa ứng dụng, trong đó có phương thức add() được nạp chồng dùng để bổ sung một thành phần vào vật chứa và phương thức remove() cũng được nạp chồng để gỡ bỏ một thành phần ra khỏi vật chứa.

4. Tạo ra Frame

Lớp JFrame là lớp con của lớp Frame (Frame là lớp con của lớp Window)

được sử dụng để tạo ra những cửa sổ cho các giao diện ứng dụng GUI.

Kịch bản chung để tạo ra một cửa sổ là:

1. Tạo ra một frame có tiêu đề gì đó, ví dụ "My Frame" :

   JFrame myWindow= new JFrame("My Frame");

2. Xây dựng một cấu trúc phân cấp các thành phần bằng cách sử dụng hàm myWindow.getContentPane().*add() để* bổ sung thêm JPanel hoặc những thành phần giao diện khác vào Frame:

Ví dụ: myWindow.getContentPane().*add(*new JButton("OK"));// Đưa vào một nút (JButton) có tên "*OK" vào frame*

3. Đặt lại kích thước cho frame sử dụng hàm setSize():
   myWindow.setSize(200, 300);// Đặt lại khung frame là 200 (300
4. Gói khung frame đó lại bằng hàm pack():
   myWindow.pack();
5. Cho hiện frame:
   myWindow.setVisible(true);

II. Trình quản lý hiển thị trong Java

Khi thiết kế giao diện đồ họa cho một ứng dụng, chúng ta phải quan tâm đến kích thước và cách bố trí (layout) các thành phần giao diện như: JButton, JCheckbox, *JTextField, v.v. sao cho tiện lợi nhất đối với người sử dụng. Java có các* lớp đảm nhiệm những công việc trên và quản lý các thành phần giao diện GUI bên trong các vật chứa.

Bảng sau cung cấp bốn lớp quản lý layout (cách bố trí và sắp xếp) các thành phần GUI.

Tên lớp	Mô tả
FlowLayout	Xếp các thành phần giao diện trước tiên theo hàng từ trái qua phải, sau đó theo cột từ trên xuống dưới. Cách sắp xếp này là mặc định đối với Panel, JPanel ,Applet và JApplet.
GridLayout	Các thành phần giao diện được sắp xếp trong các ô lưới hình chữ nhật lần lượt theo hàng từ trái qua phải và theo cột từ trên xuống dưới trong một phần tử chứa. Mỗi thành phần giao diện chứa trong một ô.
BorderLayout	Các thành phần giao diện (ít hơn 5) được đặt vào các vị trí theo các hướng: north (bắc), south (nam), west (tây), east (đông) và *center (trung tâm)). Cách sắp xếp này là mặc định* đối với lớp Window, Frame, Jframe, Dialog và JDialog.
GridBagLayout	Cho phép đặt các thành phần giao diện vào lưới hình chữ nhật, nhưng một thành phần có thể chiếm nhiều nhiều hơn một ô.
null	Các thành phần bên trong vật chứa không được sẵp lại khi kích thước của vật chứa thay đổi.

Các phương pháp thiết đặt layout

Để lấy về layout hay để đặt lại layout cho vật chứa, chúng ta có thể sử dụng hai phương thức của lớp Container:

LayoutManager getLayout();

void setLayout(LayoutManager mgr);

Các thành phần giao diện sau khi đã được tạo ra thì phải được đưa vào một phần tử chứa nào đó. Hàm add() của lớp Container được nạp chồng để thực hiện nhiệm vụ đưa các thành phần vào phần tử chứa.

Component add(Component comp)

Component add(Component comp, int index)

Cmponent add(Component comp, Object constraints)

Cmponent add(Component comp, Object constraints, int index)

Trong đó, đối số index được sử dụng để chỉ ra vị trí của ô cần đặt thành phần giao diện comp vào. Đối số constraints xác định các hướng để đưa comp vào phần tử chứa.

Ngược lại, khi cần loại ra khỏi phần tử chứa một thành phần giao diện thì sử dụng các hàm sau:

void remove(int index)

void remove(Component comp)

void removeAll()

1. Lớp FlowLayout

Lớp FlowLayout cung cấp các hàm tạo lập để sắp hàng các thành phần giao diện:

FlowLayout()

FlowLayout(int aligment)

FlowLayout(int aligment, int horizongap, int verticalgap)

public static final int LEFT

public static final int CENTER

public static final int RIGHT

Đối số aligment xác định cách sắp theo hàng: từ trái, phải hay trung tâm, *horizongap và verticalgap là khoảng cách tính theo pixel giữa các* hàng các cột. Trường hợp mặc định thì khoảng cách giữa các hàng, cột là 5 pixel.

2. Lớp GridLayout

Lớp GridLayout cung cấp các hàm tạo lập để sắp hàng các thành phần giao diện:

GridLayout()

GridLayout(int rows, int columns)

GridLayout(int rows, int columns, int hoiongap, int verticalgap)

Tạo ra một lưới hình chữ nhật có rows (columns ô có khoảng cách giữa các hàng các cột là horizongap, verticalgap. Một trong hai đối số *rows hoặc columns có thể là 0, nhưng không thể cả hai, GridLayout(1,0)* là tạo ra lưới có một hàng.

3.Lớp BorderLayout

Lớp BorderLayout cho phép đặt một thành phần giao diện vào một trong bốn hướng: *bắc (NORTH), nam (SOUTH), đông (EAST), tây (WEST) và ở giữa (CENTER).*

BorderLayout()

BorderLayout(int horizongap, int verticalgap)

Tạo ra một layout mặc định hoặc có khoảng cách giữa các thành phần (tính bằng pixel) là horizongap theo hàng và verticalgap theo cột.

Component add(Component comp)

void add(Component comp, Object constraint)

public static final String NORTH

public static final String SOUTH

public static final String EAST

public static final String WEST

public static final String CENTER

Trường hợp mặc định là CENTER, ngược lại, có thể chỉ định hướng để đặt các thành phần comp vào phần tử chứa theo constraint là một trong các hằng trên.

III. Xử lý sự kiện trong Java

Các ứng dụng với GUI thường được hướng dẫn bởi các sự kiện (event). Việc nhấn một nút, mở, đóng các Window hay gõ các ký tự từ bàn phím, v.v. đều

tạo ra các sự kiện (event) và được gửi tới cho chương trình ứng dụng. Trong Java các sự kiện được thể hiện bằng các đối tượng. Lớp cơ sở nhất, lớp cha của tất cả các lớp con của các sự kiện là lớp java.util.EventObject.

Hình H7-20 Các lớp xử lý các sự kiện

Các lớp con của AWTEvent được chia thành hai nhóm:

1. Các lớp mô tả về ngữ nghĩa của các sự kiện,
2. Các lớp sự kiện ở mức thấp.

1. Ý nghĩa của các lớp

a. ActionEvent

Sự kiện này được phát sinh bởi những hoạt thực hiện trên các thành phần của

GUI. Các thành phần gây ra các sự kiện hành động bao gồm:

1 *JButton - khi một nút button được khích hoạt,*

2 *JList - khi một mục trong danh sách được kích hoạt đúp,*

3 *JmenuItem, JcheckBoxMenu, JradioMenu - khi một mục trong thực đơn được chọn,*

4 *JTextField - khi gõ phím ENTER trong trường văn bản (text).*

b. ***AdjustmentEvent***

Sự kiện này xẩy ra khi ta điều chỉnh (adjustment) giá trị thanh cuốn (JScollBar)

1 Scrollbar - khi thực hiện một lần căn chỉnh trong thanh trượt Scrollbar.

Lớp này có phương thức int getValue(): cho lại giá trị hiện thời được xác định bởi lần căn chỉnh sau cùng.

c. ***ItemEvent***

Các thành phần của GUI gây ra các sự kiện về các mục gồm có:

1 JCheckbox - khi trạng thái của hộp kiểm tra Checkbox thay đổi.

2 CheckboxMenuItem - khi trạng thái của hộp kiểm tra Checkbox ứng với mục của thực đơn thay đổi.

3 JRadioButton- khi trạng thái của hộp chọn (Option) thay đổi.

4 JList - khi một mục trong danh sách được chọn hoặc bị loại bỏ chọn.

5 JCompoBox - khi một mục trong danh sách được chọn hoặc bị loại bỏ chọn.

Lớp ItemEvent có phương thức ***Object getItem():*** *Cho lại đối tượng được chọn hay vừa bị bỏ chọn.*

d. TextEvent

Các thành phần của GUI gây ra các sự kiện về text gồm có:

1 TextArea - khi kết thúc bằng nhấn nút ENTER,

2 TextField - khi kết thúc bằng nhấn nút ENTER.

e. ComponentEvent

Sự kiện này xuất hiện khi một thành phần bị ẩn đi/hiển ra hoặc thay thay đổi lại kích thước. Lớp ComponentEvent có phương thức:

Component getComponent()

Cho lại đối tượng tham chiếu kiểu Component.

f. ContainerEvent

Sự kiện này xuất hiện khi một thành phần được bổ sung hay bị loại bỏ khỏi vật chứa (Container).

g. FocusEvent

Sự kiện loại này xuất hiện khi một thành phần nhận hoặc mất focus.

h. KeyEvent

Lớp KeyEvent là lớp con của lớp trừu tượng InputEvent được sử dụng để xử lý các sự kiện liên quan đến các phím của bàn phím. Lớp này có các phương thức:

int getKeyCode()

- Đối với các sự kiện KEY_PRESSED hoặc KEY_RELEASED, hàm này được sử dụng để nhận lại giá trị nguyên tương ứng với mã của phím trên bàn phím.

char getKeyChar()

- Đối với các sự kiện KEY_PRESSED, hàm này được sử dụng để nhận lại giá trị nguyên, mã Unicode tương ứng với ký tự của bàn phím.

i. MouseEvent

Lớp MouseEvent là lớp con của lớp trừu tượng InputEvent được sử dụng để xử lý các tín hiệu của chuột. Lớp này có các phương thức:

int getX()

int getY()

Point getPoint()

Các hàm này được sử dụng để nhận lại tọa độ x, y của vị trí liên quan đến sự kiện do chuột gây ra.

void translatePoint(int dx, int dy)

Hàm translate() được sử dụng để chuyển tọa độ của sự kiện do chuột gây ra đến (dx, dy).

int getClickCount()

Hàm getClickCount() đếm số lần kích chuột.

j. PaintEvent

Sự kiện này xuất hiện khi một thành phần được vẽ lại, thực tế sự kiện này xẩy ra khi phương thức paint()/ update() được gọi đến.

k. WindowEvent

Sự kiện loại này xuất hiện khi thao tác với các Window, chẳng hạn như: đóng, phóng to, thu nhỏ.. một cửa sổ. Lớp này có phương thức:

Window getWindow()

Hàm này cho lại đối tượng của lớp Window ứng với sự kiện liên quan đến Window đã xảy ra.

Kiểu sự kiện	Nguồn gây ra sự kiện	Phương thức đang ký, gỡ bỏ đối tượng lắng nghe	Giao diện Listener lắng nghe tương ứng
AcitionEvent	JButton JList TexField	addComponentlistener removeActiontListener	AcitionListener
AdjustmentEvent	JScrollbar	addAdjustmentListener removeAdjustmentListener	AdjustmentListener
ItemEvent	JCheckbox	addItemListener	ItemListener

	JCheckboxMenuItem JRadioButton JList JCompoBox	removeItemListener	
TextEvent	JTextArea JTexField JTextPane JEditorPane	addTexListener removeTextListener	TextListener
ComponentEvent	Component	addComponentListener removeComponentListener	ComponentListener
ContainerEvent	Container	addContainerListener removeContainerListener	ContainerListener
FocusEvent	Component	addFocusListener removeFocusListener	FocusListener
KeyEvent	Component	addkeyListener removeKeyListener	KeyListener
MouseEvent	Component	addMouseListener remoMouseListener addMouseMotionListener remoMouseMotionListener	MouseMotionListener
WindowEvent	Window	addWindowListener removeWindowListener	WindowListener

3. Một số lớp điều hợp

Giao diện Listener lắng nghe	Lớp điều hợp tương ứng
AcitionListener	Không có lớp điều hợp tương ứng
AdjustmentListener	AdjustmentAdapter
ItemListener	Không có lớp điều hợp tương ứng
TextListener	Không có lớp điều hợp tương ứng
ComponentListener	ComponentAdapter
ContainerListener	ContainerAdapter
FocusListener	FocusAdapter
KeyListener	KeyAdapter
MouseMotionListener	MouseMotionAdapter
WindowListener	WindowAdapter