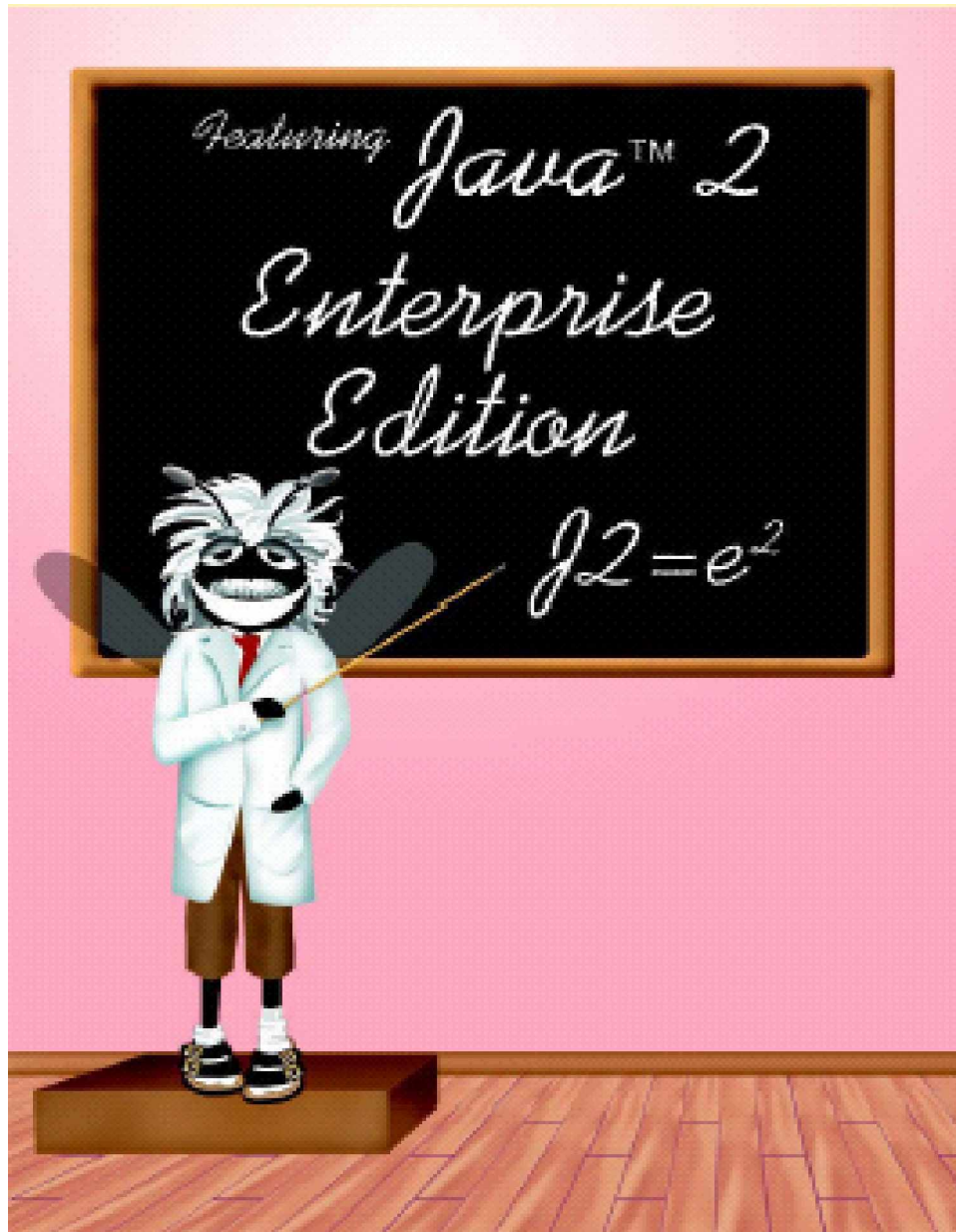


Tr-êng® i hác s- ph¹m kù thuËt h-ng yªn
khoa c«ng nghö th«ng tin

-----ooo-----



Sô c- ñng bµi gi¶ng Java

Nhóm biên soạn: 1. Hoàng Trọng Thế

2. Nguyễn Duy Tân

MỤC LỤC

Chương 1: CÁC KHÁI NIỆM CƠ BẢN.....	5
BÀI 1. LÀM QUEN VỚI JAVA.....	5
I. Lịch sử java	5
II. Java em là ai	5
II. Một số đặc trưng của java	5
III. Các kiểu ứng dụng Java.....	7
IV. Máy ảo Java (JVM-Java Virtual Machine)	7
BÀI 2 NỀN TẢNG CỦA JAVA	9
I. Tập ký tự dùng trong java	9
II. Từ khoá của Java	9
III. Định danh (tên)	10
IV. Cấu trúc một chương trình java	10
V. Chương trình JAVA đầu tiên	12
VI. Chú thích trong chương trình	14
VII. Kiểu dữ liệu	15
VII. Một số phép toán trên kiểu dữ liệu nguyên thủy	17
IX. Toán tử chuyển kiểu	23
X. Các hàm toán học	24
XI. Các phép toán trên kiểu kí tự	26
BÀI 3. ĐIỀU KHIỂN LƯỠNG CHƯƠNG TRÌNH	27
I. Cấu trúc rẽ nhánh	27
II. Cấu trúc lặp while và do-while	30
III. Cấu trúc for	32
IV. Lệnh break và continue	34
Chương 2: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG	37
BÀI 1. ĐỊNH NGHĨA LỚP	38
I. Khai báo lớp	38
II. Chi tiết về khai báo một phương thức	40
III. Từ khoá this	43
IV. Từ khoá super	43
V. Sử dụng lớp	44
VI. Điều khiển việc truy cập đến các thành viên của một lớp	44
BÀI 2. KHỞI ĐẦU VÀ DỌN DẸP	47
I. Phương thức tạo dựng (constructor)	47
II. Khởi khởi đầu vô danh và khởi khởi đầu tĩnh	49
III. Dọn dẹp: kết thúc và thu rác	51
BÀI 3. CÁC THÀNH PHẦN TĨNH	52
I. Thuộc tính tĩnh	52
II. Phương thức tĩnh	52
BÀI 4. NẠP CHỒNG PHƯƠNG THỨC	53
I. Khái niệm về phương thức bội tải	53

II. Yêu cầu của các phương thức bội tải	53
BÀI 5. KẾ THỪA (INHERITANCE)	54
I. Lớp cơ sở và lớp dẫn xuất	54
II. Cách xây dựng lớp dẫn xuất	54
III. Thừa kế các thuộc tính	54
IV. Thừa kế phương thức	54
V. Khởi đầu lớp cơ sở	54
VI. Ghi đè phương thức (Override)	56
VI. Thành phần protected	56
VII. Từ khoá final	56
BÀI 6. LỚP CƠ SỞ TRỪU TƯỢNG	61
BÀI 7. ĐA HÌNH THÁI	62
BÀI 8. GIAO DIỆN, LỚP TRONG, GÓI	63
I. Giao diện	63
II. Lớp trong	65
Bài 9. MẢNG, XÂU KÝ TỰ, TẬP HỢP	67
I. Mảng	67
II. Xâu ký tự	70
III. Lớp StringTokenizer	75
IV. Một số lớp cơ bản của Java	77
V. Các lớp tập hợp	83
Chương 3: XỬ LÝ NGOẠI LỆ	92
I. Mục đích của việc xử lý ngoại lệ	92
II. Mô hình sử lý ngoại lệ của java	92
III. Đặc tả ngoại lệ	93
III. Ném ra ngoại lệ	93
IV. Bắt ngoại lệ	93
V. Khối ‘finally’	95
VI. Một số lớp ngoại lệ chuẩn của Java	96
Chương 4: LẬP TRÌNH ĐA TUYẾN	97
I. Các kiến thức liên quan	97
II. Lập trình đa tuyến trong Java	98
Chương 5: NHẬP XUẤT (Input/Output)	104
I. Lớp luồng	105
II. Lớp File	113
Chương 6: LẬP TRÌNH ĐỒ HOẠ AWT	120
I. Giới thiệu về hệ thống đồ hoạ của Java	120
II. Trình quản lý hiển thị trong Java	124
III. Xử lý sự kiện trong Java	125
Chương 7 LẬP TRÌNH CƠ SỞ DỮ LIỆU	143
I. Tổng quan	143
II. Các kiểu trình điều khiển JDBC.....	143

III. Các lớp trong JDBC API dùng để truy cập CSDL	146
IV. Kết nối CSDL với JDBC	147
V. Tương tác với CSDL	152
VI. Quản lý giao dịch	160
Chương 8: LẬP TRÌNH SOCKET	164
I. Các kiến thức liên quan	164
II. Khảo sát một số lớp trong gói java.net	164
2. Lớp URL và URI	171
3. Lớp Socket.....	176
4. Lớp ServerSocket.....	182
5. Lớp DatagramSocket	185
6. Lớp DatagramPacket	185
III. TCP Socket	185
Chương 9: JAVA MAIL	205
I. Các khái niệm MAIL	205
II. Gửi mail với thư viện có sẵn của java	206
III. Giới thiệu java mail API	207
III. Gửi mail	212
IV. Nhận mail	215
V. Xoá mail	221
Chương 10: LẬP TRÌNH PHÂN TÁN VỚI	222
JAVA RMI (REMOTE METHOD INVOKE).....	222
I. RMI và lập trình phân tán đối tượng	222
II. Gọi phương thức từ xa và các vấn đề phát sinh	223
III. Vai trò của các lớp trung gian	223
IV. Cài đặt đối tượng phân tán	224
Chương 11: JAVA JSP (JAVA SERVLET)	233
Chương 12: EJB (ENTERPRISE JAVA BEANS)	234
I. Tổng quan về JavaBean	234
II. Xây dựng thành phần JavaBean	234
III. Tìm hiểu Instrospector	247
IV. Đóng gói JavaBean	249

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 thủy 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 cả 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 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 này 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. Quá 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 quá 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 đơn 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 dẫn, 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:

- Ø Nạp các file .class
- Ø Quản lý bộ nhớ
- Ø 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:

- Ø Các “Frame” chứa các trạng thái của các phương thức.
- Ø Các toán hạng của mã bytecode.
- Ø Các tham số truyền cho phương thức.
- Ø Các biến cục bộ.

Khi JVM thực thi mã, một thanh ghi cục bộ có tên “**Program Counter**” được sử dụng. Thanh ghi này trở 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 tự 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 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
<i>abstract</i>	Sử dụng để khai báo lớp, phương thức trừu tượng
<i>boolean</i>	Kiểu dữ liệu logic
<i>break</i>	Được sử dụng để kết thúc vòng lặp hoặc cấu trúc switch
<i>byte</i>	kiểu dữ liệu số nguyên
<i>case</i>	được sử dụng trong lệnh switch
<i>cast</i>	Chưa được sử dụng (để dành cho tương lai)
<i>catch</i>	được sử dụng trong xử lý ngoại lệ
<i>char</i>	kiểu dữ liệu ký tự
<i>class</i>	Dùng để khai báo lớp
<i>const</i>	Chưa được dùng
<i>continue</i>	được dùng trong vòng lặp để bắt đầu một vòng lặp mới
<i>default</i>	được sử dụng trong lệnh switch
<i>do</i>	được dùng trong vòng lặp điều kiện sau
<i>double</i>	kiểu dữ liệu số thực
<i>else</i>	khả năng lựa chọn thứ hai trong câu lệnh if
<i>extends</i>	chỉ rằng một lớp được kế thừa từ một lớp khác
<i>false</i>	Giá trị logic
<i>final</i>	Dùng để khai báo hằng số, phương thức không thể ghi đè, hoặc lớp không thể kế thừa
<i>finally</i>	phần cuối của khối xử lý ngoại lệ
<i>float</i>	kiểu số thực
<i>for</i>	Câu lệnh lặp
<i>goto</i>	Chưa được dùng
<i>if</i>	Câu lệnh lựa chọn
<i>implements</i>	chỉ rằng một lớp triển khai từ một giao diện
<i>import</i>	Khai báo sử dụng thư viện
<i>instanceof</i>	kiểm tra một đối tượng có phải là một thể hiện của lớp hay không
<i>interface</i>	sử dụng để khai báo giao diện
<i>long</i>	kiểu số nguyên
<i>native</i>	Khai báo phương thức được viết bằng ngôn ngữ biên dịch C++
<i>new</i>	tạo một đối tượng mới

<i>null</i>	một đối tượng không tồn tại
<i>package</i>	Dùng để khai báo một gói
<i>private</i>	đặc tả truy xuất
<i>protected</i>	đặc tả truy xuất
<i>public</i>	đặc tả truy xuất
<i>return</i>	Quay từ phương thức về chỗ gọi nó
<i>short</i>	kiểu số nguyên
<i>static</i>	Dùng để khai báo biến, thuộc tính tĩnh
<i>super</i>	Truy xuất đến lớp cha
<i>switch</i>	lệnh lựa chọn
<i>synchronized</i>	một phương thức độc quyền truy xuất trên một đối tượng
<i>this</i>	Chỉ chỉ chính lớp đó
<i>throw</i>	Ném ra ngoại lệ
<i>throws</i>	Khai báo phương thức ném ra ngoại lệ
<i>true</i>	Giá trị logic
<i>try</i>	sử dụng để bắt ngoại lệ
<i>void</i>	Dùng để khai báo một phương thức không trả về giá trị
<i>while</i>	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 tùy 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 nhau bởi kí tự gạch 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 mỗi 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 phẩy “;”.

Một đơn vị biên dịch là một tệp tin .java

```
//Nhập khẩu các thư viện
import thư_viện;
//Xây dựng các lớp
public class clsMain{
...
public static void main(String args[])
{
//điểm bắt đầu của chương trình
}
...
}
class lớp1
{
...
}
class lớp2
{
...
}
```

- 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 kiểu dữ liệu. Kiểu dữ liệu mới này được sử dụng để xác định các biến mà ta thường 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 bạn 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)

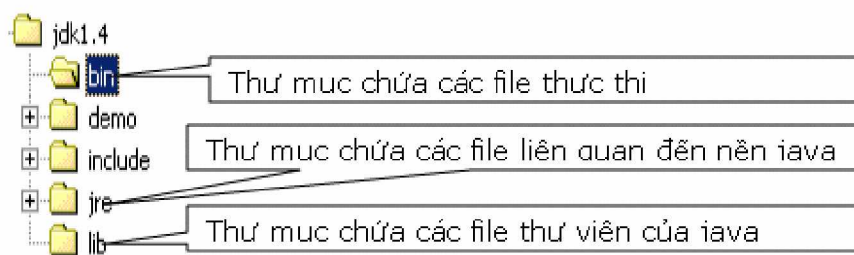
Lớp bên trong (không phải là lớp trong tĩnh) có thể truy nhập tất cả các thành phần của lớp bao nó, song không thể ngược lại.

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:



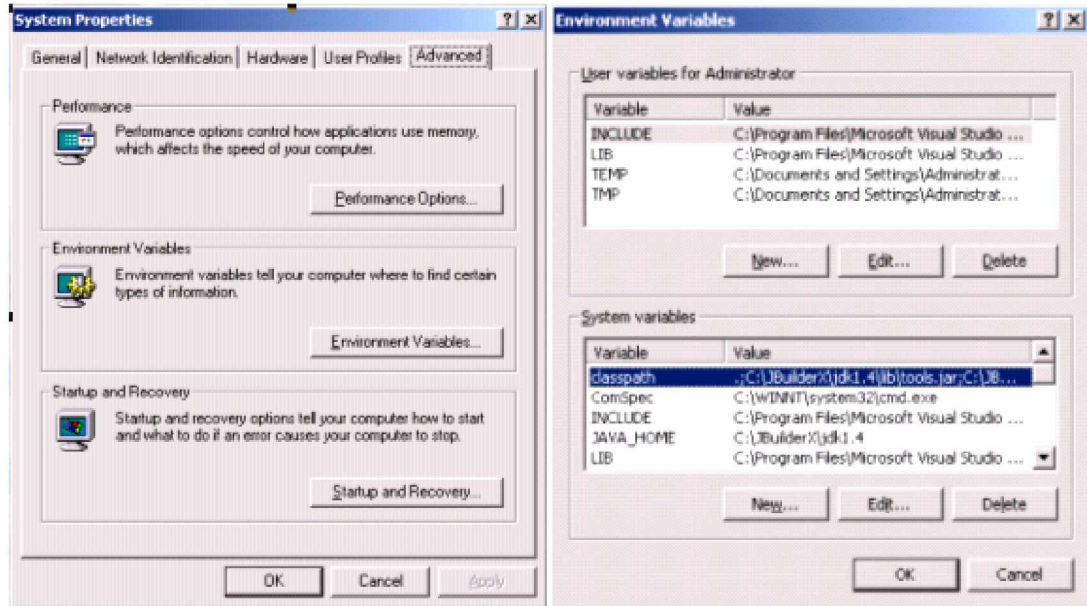
- Để biên dịch một chương trình java sang mã byte code ta dùng lệnh
C:\JDK1.4\BIN\javac TênTệp.java
- Để thông dịch và chạy chương trình ta sử dụng lệnh

C:\JDK1.4\BIN\java TênTệp

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

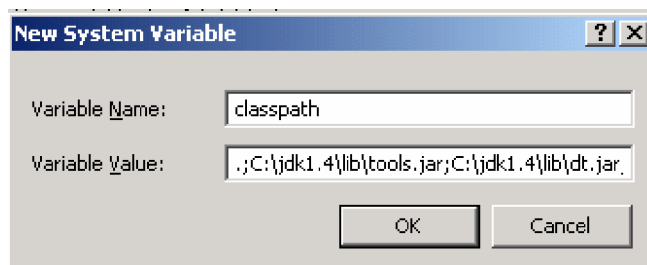
- Đối với dòng WinNT:

+ R-Click vào My Computer → chọn Properties → chọn Advanced → Environment 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)

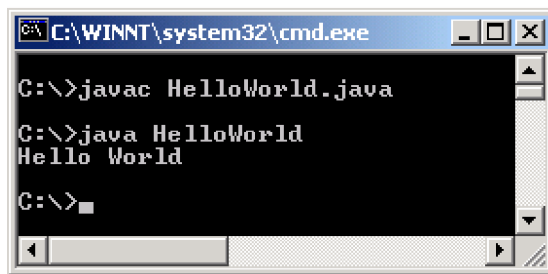
Các bước:

- Mở một chương trình soạn thảo văn bản hỗ trợ ascii, 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 cửa 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++

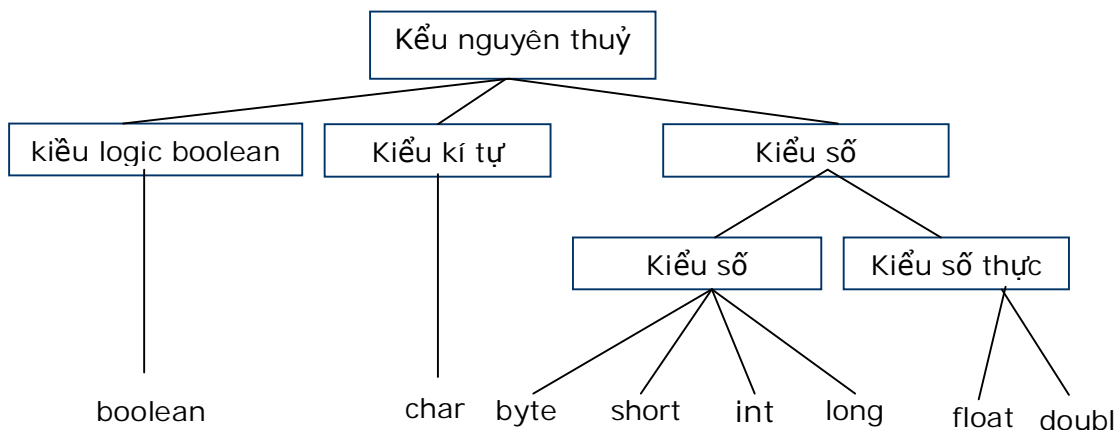
Cách 3: sử dụng cặp /** và */, đây là kiểu chú thích tài liệu (không có trong C/C++), nó dùng để tạo ra tài liệu chú thích cho chương trình.

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 thủy



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

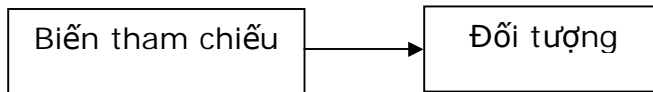
Đặc điểm của các biến có kiểu nguyên thủy là vùng nhớ của chúng được cấp phát ở phần stack. Do vậy việc truy xuất vào một biến kiểu nguyên thủy rất nhanh.

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 cập vào vùng nhớ heap chậm hơn truy cập 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:

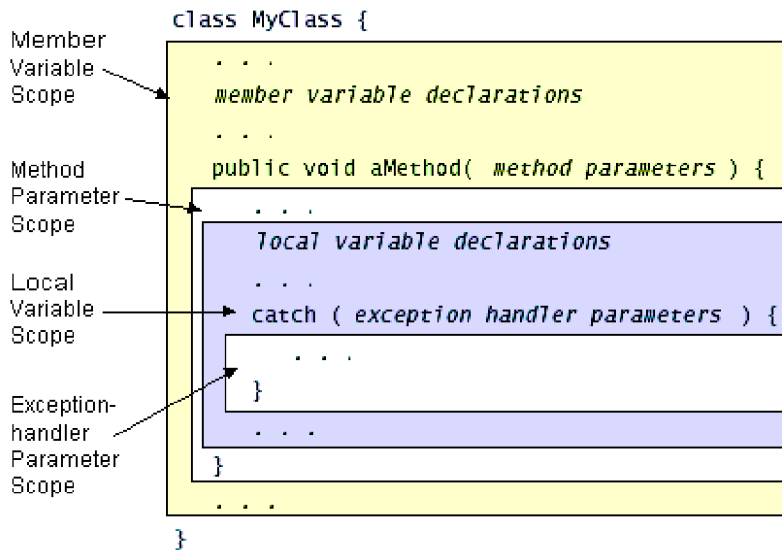
- type là kiểu dữ liệu của biến
- name là tên của biến, tên biến là một chuỗi ký tự được đặt theo quy tắc đặt tên của java
- InitValue là giá trị khởi tạo cho biến, đây là phần tùy chọn, nếu bỏ qua phần này thì giá trị ban đầu của biến được khởi tạo giá trị mặc định

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 phẩy
- 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
 - + ký tự null (mã 0) cho ký 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.



- Ø 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
- Ø 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 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 thủy.

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

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 dãy 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 với op2
-	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 bảng 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 xuống 1 đơn vị, giá trị của op được giảm xuống trước khi biểu thức chứa nó được tính
--	--op	Giảm op xuống 1 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...

$i < j = \text{true}$
 $j < i = \text{false}$
 $k < j = \text{false}$
 Less than or equal to...
 $i \leq j = \text{true}$
 $j \leq i = \text{false}$
 $k \leq j = \text{true}$
 Equal to...
 $i == j = \text{false}$
 $k == j = \text{true}$
 Not equal to...
 $i != j = \text{true}$
 $k != j = \text{false}$

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

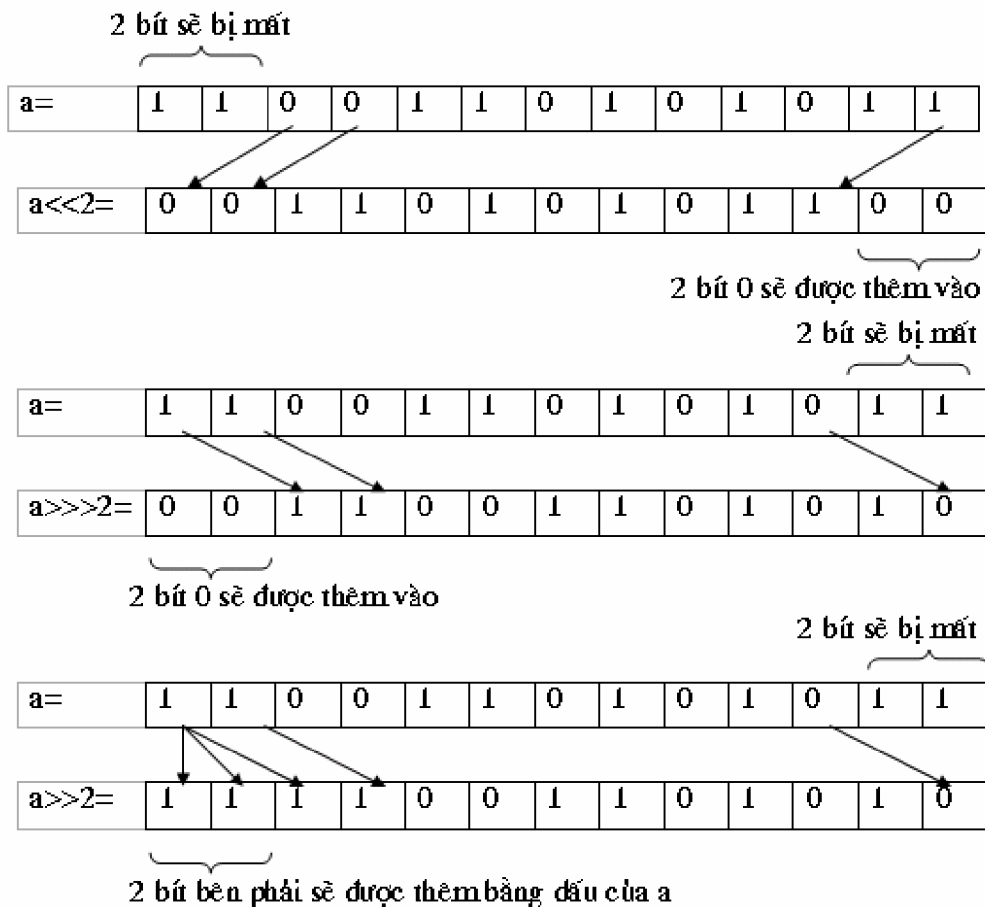
6. phép toán thao tác trên bit

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 bit 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 bit 0
>>>	op1 >>> op2	Dịch chuyển op1 sang phải op2 bit, op2 bit

Sau đây là hình minh họa phép toán dịch bit



Ví dụ:

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

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

$5 \ll 2 = 100$ 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 not các bit của op2

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

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

Phép XOR		
op1	op2	Result
0	0	0
0	1	1

Phép NOT	
op1	Result
0	1
1	0

1	0	1			
1	1	0	op1	op2	Result
			0	0	0
			0	1	1
			1	0	1
			1	1	1

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:

- Ø Các phép toán một ngôi bao giờ cũng được thực hiện trước tiên
- Ø 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
- Ø 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.

- Toán tử dãy
Không giống như C/C++, trong java chỗ duy nhất mà ta có thể đặt toán tử dãy 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 bằng 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:

- Phép toán chuyển kiểu là phép toán có độ ưu tiên cao, nên $(int)3.5*2 \neq (int)(3.4*2)$
- 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 là một biểu thức kiểu	double trong hệ

	arg	double thể hiện một cung theo radians	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 $\frac{arg1}{arg2}$	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 trong hai tham số	arg1, arg2 là một biểu thức kiểu int, long, float, hoặc double	Nhận về kiểu cùng kiểu với tham số
min (arg1,arg2)	Nhận về giá trị nhỏ trong hai tham số	arg1, arg2 là một biểu thức kiểu int, long, float, hoặc double	Nhận về kiểu cùng kiểu 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 kiểu float, nhận về kiểu 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 $arg1^{arg2}$	Cả arg1 và arg2 là các biểu thức kiểu double	double
exp(arg)	tính e^{arg}	arg là biểu thức kiểu double	double
log(arg)	tính logarithm cơ số e của arg	arg là biểu thức kiểu double	double
random()	Nhận về một số giả ngẫu nhiên nằm trong khoảng [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 LƯỠNG CHƯƠNG TRÌNH

Chương trình là một dãy 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 tùy 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

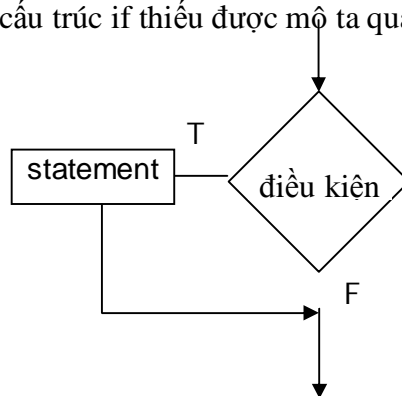
1.1. Phát biểu if

a) Dạng khuyết

Cú pháp

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

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

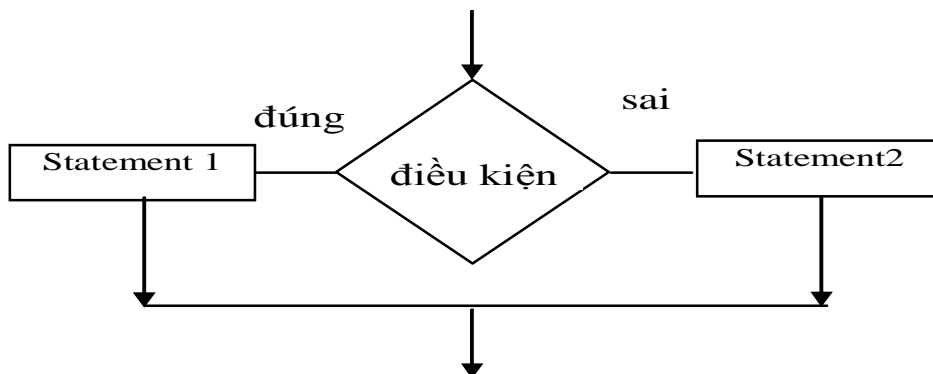


b) Dạng đủ

Cú pháp

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

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



1.2. Biểu thức điều kiện

Cú pháp:

Variable=booleanExpression? true-result-expression:

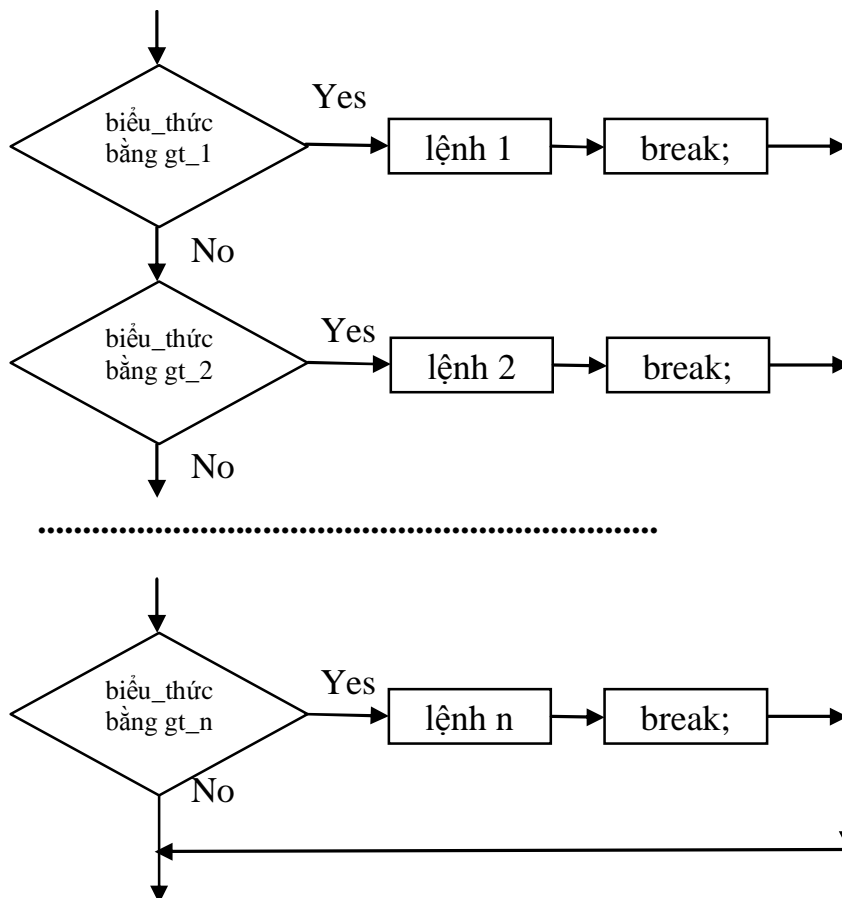
false-result-expression;

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;]  
}
```

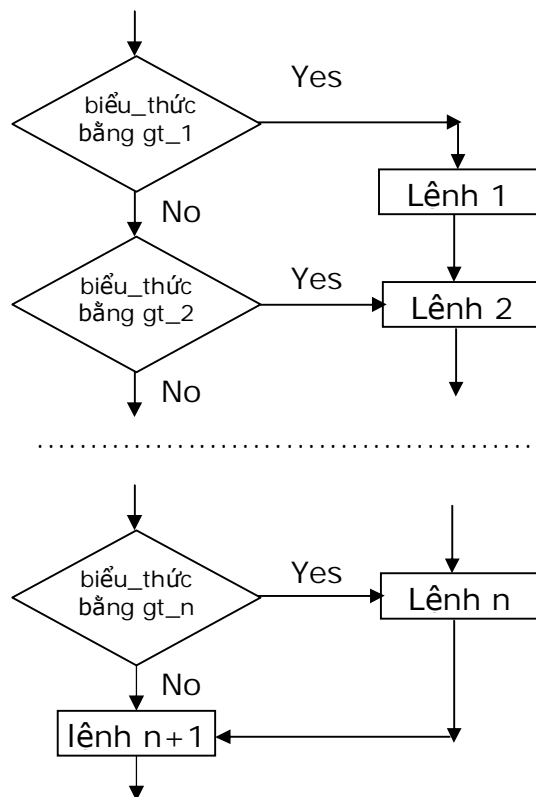
Sau đây là sơ đồ khối mô tả sự hoạt động của cấu trúc rẽ nhánh switch dạng thiếu



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 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 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

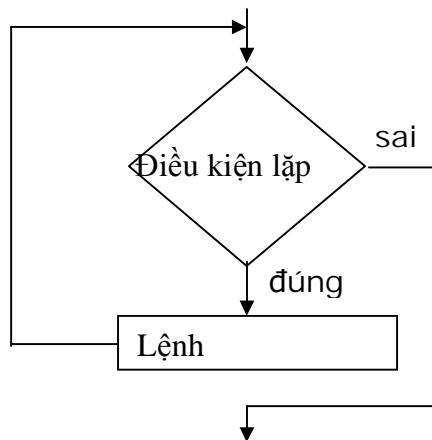
1. Lặp kiểm tra điều kiện trước

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 logic ban đầu có giá trị false)

2. Lặp kiểm tra điều kiện sau

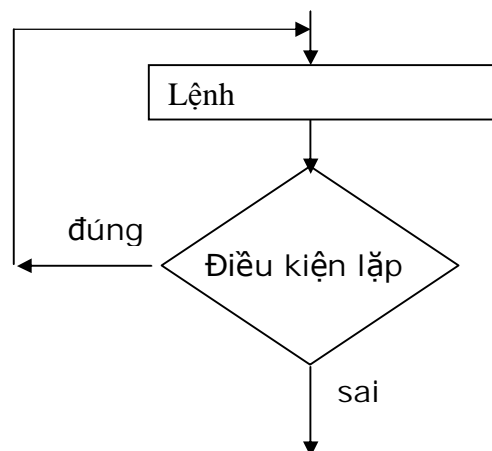
Cú pháp:

```

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

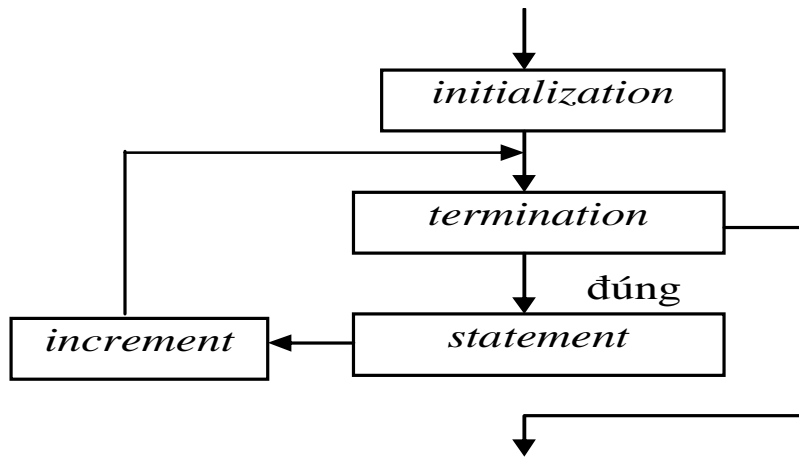
Đây là cấu trúc lặp phổ biến nhất trong các ngôn ngữ lập trình, mà nội dung của vòng lặp cần phải lặp đi lặp lại một số lần biết trước, cú pháp của nó như sau:

```
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 đây phải 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: liệt kê ra 128 các kí tự ascii 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 dãy và vòng lặp for

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

Ví dụ về toán tử dãy

```
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 phần 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 nhảy goto, tuy nhiên trong java vẫn có một vài vết tích của lệnh nhảy 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 tùy 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.

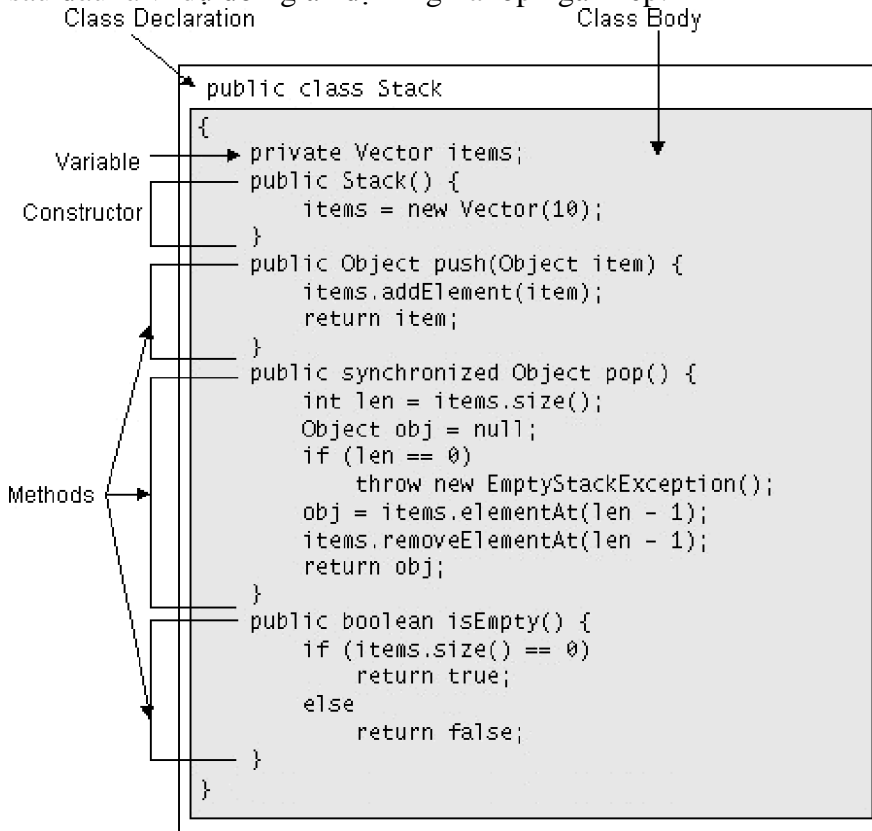
BÀI 1. ĐỊNH NGHĨA LỚP

I. Khai báo lớp

1.1. Một lớp được định nghĩa theo mẫu sau:

```
[public][final][abstract] class <tên_lớp>{  
    // khai báo các thuộc tính  
    // khai báo các phương thức  
}
```

sau đây 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:

- a) 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
- b) 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 giới 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 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.
- c) 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.
- d) Các phương thức thường khai báo là public, để chúng có thể truy cập từ bất cứ đâu.
- e) 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

1.2. Khai báo thuộc tính

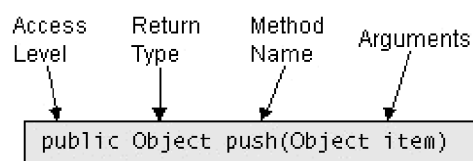
Trở lại lớp Stack

```
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:



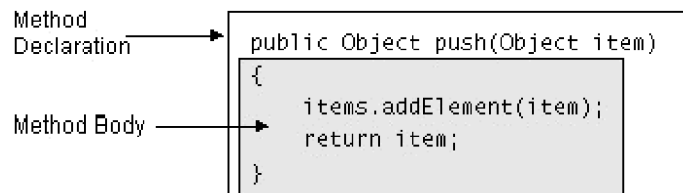
Trong đó:

- Ø *accessLevel* có thể là một trong các từ *public*, *private*, *protected* hoặc có thể bỏ trống, ý nghĩa của các từ này được mô tả ở phần trên
- Ø - *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.
- Ø - *transient* và *volatile* chưa được dùng
- Ø - *type* là một kiểu dữ liệu nào đó
- Ø *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.

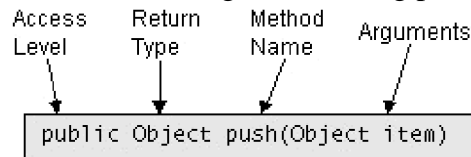
1.3. Khai báo phương thức

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	//khai báo các ngoại lệ có thể được ném ra từ phương thức
exception	

Trong đó:

- **accessLevel** có thể là một trong các từ khoá public, private, protected hoặc bỏ trống, ý nghĩa của các 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 thì 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 thủy 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 này 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 thủy, 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 sẽ 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 màu 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--]; }
    }
}
```

Ví dụ: Xây dựng lớp man

```
import java.lang.*;
import java.io.*;
import java.util.*;
public class Man
```

```

{protected
    String ten;
    int namsinh;
    public Man(String name,int birthday){
        ten=name;namsinh=birthday;
    }
    public int tuoi()
    {Calendar now=Calendar.getInstance();
    return (now.get(Calendar.YEAR) - namsinh);
    }
    public void hienthi()
    {System.out.println("Ho va ten:"+ten);
    System.out.println("Ho va ten:"+tuoi());
    }
    public void HienThiNgay()
    { Calendar now=Calendar.getInstance();
    System.out.println("Ngày hiện tại=" +now.get(Calendar.DATE));
    System.out.println("Thang hiện tại=" +now.get(Calendar.MONTH));
    System.out.println("Nam hiện tại=" +now.get(Calendar.YEAR));
    System.out.println("Gio hiện tại=" +now.get(Calendar.HOUR));
    System.out.println("Phut hiện tại=" +now.get(Calendar.SECOND));
    }
}

```

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 khai 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 thủy

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 sẽ 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

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

- 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");//
}
}

```

b) 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. Khởi khởi đầu vô danh và khởi khởi đầu tĩnh

1. Khởi 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 thuộc 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 ( "khai khai dau thu 3 ");
} // kết thúc khối vô danh
// bắt đầu khối vô danh
{
    System.out.println ( "khai khai dau thu 1 ");
} //kết thúc khối vô danh
// bắt đầu khối vô danh
{

```

```

        System.out.println ( "khôi khôi đầu 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:

```

khôi khôi đầu thu 3
khôi khôi đầu thu 1
khôi khôi đầu thu 2
Day la ham tao
khôi khôi đầu thu 3
khôi khôi đầu thu 1
khôi khôi đầu 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 thứ 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 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 hủy, sẽ có một hàm được gọi tự động, hàm này được gọi là hủy tử hay còn gọi là hàm hủy, thông thường hàm hủy mặc định là đủ là đủ để dọn dẹp, tuy nhiên trong một số trường hợp thì hàm hủy mặc định lại không thể đáp ứng được, do vậy người lập trình C++, phải viết ra hàm hủy riêng để làm việc đó, tuy nhiên java lại không có khái niệm hàm hủy 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_tượng.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:

Ø Sử dụng khởi đầu tĩnh (xem lại bài trước)

Ø 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_tượng.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)

Ø 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

Thuộc 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:

- Ø Phương thức tạo dựng
- Ø Phương thức finalize

V. Khởi đầu lớp cơ sở

Lớp dẫn xuất kế thừa mọi thành phần của lớp cơ sở, 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 tương minh của lớp cơ sở, thế thì lớp cơ sở phải được bảo đảm khởi đầu đúng, để thực hiện điều đó trong 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 bạn, 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 xuất" );  
    }  
}
```

```

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 chương 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 nhận 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.overridden MethodName();
```

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 tùy 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 là hằng, 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 chẳng 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

Ví dụ về tính kế thừa

1. Xây dựng lớp cơ sở

```

package JavaContent;
import java.util.*;
class HocSinh {
    //Khai báo du lieu thanh phan
    protected

```

```

String hoTen;
String diaChi;
int namSinh;
protected double dToan;
double dLy;
double dHoa;
//Khai bao Constructor & Destructo
public HocSinh(String name,String Address,int birthday,double d1,double d2,double
d3)
{hoTen=name;diaChi=Address;namSinh=birthday;dToan=d1;dLy=d2;dHoa=d3; }
//Khai bao cac phuong thuc cua lop Man
public int tinhTuoi()
{Calendar now=Calendar.getInstance();
return (now.get(Calendar.YEAR)-namSinh); }
public double tongDiem()
{return (dToan+dLy+dHoa);}
public void hienThi()
{System.out.println("Ho va ten nhan su="+hoTen);
System.out.println("Dia chi =" +diaChi);
System.out.println("Tuoi="+tinhTuoi());
System.out.println("Diem Toan="+dToan);
System.out.println("Diem Ly="+dLy);
System.out.println("Diem Hoa="+dHoa);
System.out.println("Tong diem cac mon="+tongDiem());
}
public static void main(String args[])
{HocSinh ng1=new HocSinh("Nguyen Tam Hung","Quang Ninh",1977,6.4,9.0,6.7);
ng1.hienThi();
}
}

```

2. Xây dựng lớp dẫn xuất

```

package JavaContent;
import java.io.*;
import java.lang.*;
public class HocSinhCD extends HocSinh
{
//-----Khai bao du lieu thanh phan cho lop CanBo
protected double dTin;
static private BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
//-----Khai bao Constructor,Destructo;
public HocSinhCD(String name, String Address, int birthday, double d1,double d2,
double d3,double d4)
{ super(name, Address, birthday, d1, d2, d3);dTin=d4;}
//-----Xay dung cac phuong thuc
public double tongDiem()
{return (dToan+dLy+dHoa+dTin);}
//-----
public double diemTBUT()
{return (dToan*2+dLy+dHoa+dTin*2)/6;}
//-----
public void hienThi1()
{System.out.println("CAC THONG SO VE CAN BO");
}
}

```

```

hienThi();
System.out.println("Diem mon Tin=" +dTin);
System.out.println("Diem TBC co uu tien=" +diemTBUT());
}
//-----
public static void main(String args[])
{String ht=" ",dc=" ";
int ns=0;
double d1=0,d2=0,d3=0,d4=0;
    System.out.print("Nhap vao ho ten:");ht=readString();
    System.out.print("Nhap vao dia chi:");dc=readString();
    System.out.print("Nhap vao nam sinh:");ns=readInt();
    System.out.print("Nhap vao diem Toan:");d1=readDouble();
    System.out.print("Nhap vao diem Ly:");d2=readDouble();
    System.out.print("Nhap vao diem Hoa:");d3=readDouble();
    System.out.print("Nhap vao diem in:");d4=readDouble();
    HocSinhCD hs1=new HocSinhCD(ht,dc,ns,d1,d2,d3,d4);
//      HocSinhCD hs1=new HocSinhCD("Nguyen Van Thanh","Ha
Tay",1980,7.0,7.0,7.0,7.0);
    hs1.hienThi1();
    HocSinhCD TK1[]=new HocSinhCD[100];//Khai bao mot mang hoc sinh trong 1 lop
    TK1[0]=hs1;
    TK1[0].hienThi();

//Goi phuong thuc Hie Thi cua lop co so hs1.hienThi();
//Doi hpuong thuc readDoube

}
//-----Xay dung cac phuong thuc nhap du lieu
public static int readInt()
{int d=0;
try{
    String str=br.readLine();
    d=Integer.parseInt(str);
catch(IOException ex){System.out.println("Loi IO "+ex.toString());}
return d;
}
//-----
public static long readLong()
{ long d=0;
try{
    String str=br.readLine();
    d=Long.parseLong(str);
catch(IOException ex){System.out.println("Loi IO "+ex.toString());}
return d;
}
//-----
public static float readFloat()
{ float d=0;
try{
    String str=br.readLine();
    d=Float.parseFloat(str);
catch(IOException ex){System.out.println("Loi IO "+ex.toString());}
}
}

```

```
    return d;
}
//-----
public static double readDouble()
{ double d=0;
  try{
    String str=br.readLine();
    d=Double.parseDouble(str);
  }catch(IOException ex){System.out.println("Loi IO "+ex.toString());}
  return d;
}
//-----
public static String readString()
{ String str=" ";
  try{
    str=br.readLine();
  }catch(IOException ex){System.out.println("Loi IO "+ex.toString());}
  return str;
}
}
```

BÀI 6. LỚP CƠ SỞ TRỪ TƯỢNG

Một lớp cơ sở trừ 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 trừ tượng

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

Chú ý:

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

- Ø 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.
- Ø 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:

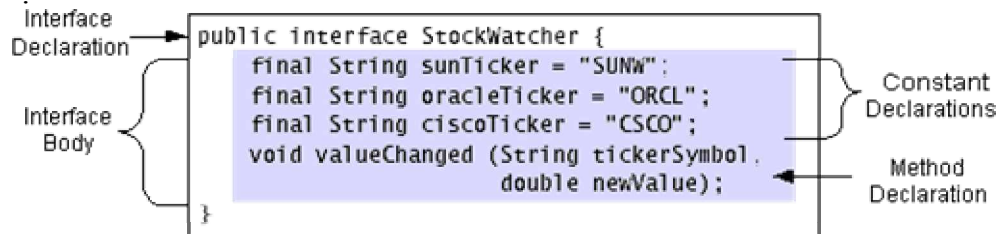
- Ø Để tạo ra một lớp cơ sở thuần ảo, một lớp không có bất cứ hàm nào được cài đặt
- Ø Thực hiện hành vi tương tự như kế thừa bội, bởi trong java không có khái niệm kế thừa bội, như của C++

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 túy”, 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 bỏ 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 bỏ 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 bỏ 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 tùy 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 bỏ từ này vào.
2. Java yêu cầu tất cả các thành phần của giao diện phải là `public`, nếu ta thêm các bỏ 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 bỏ 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 1

```
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)) {
...
}
}
}
```

Ví dụ 2

```
public interface MyInterface{
public void add(int x, int y);
public void volume(int x, int y, int z);
}
```



```

    } //Hoặc
interface Interface_Const{
    public static final double price = 100.0;
    public static final int counter = 5;
}

```

+ Sử dụng Interface trên

```

package JavaContent;
public class Use_MyInterface implements MyInterface,Interface_Const
{ public void add(int x, int y)
  {System.out.println("Tong cua 2 so "+x+" + " + y+" la:"+(x+y));}

  public void volume(int x, int y, int z)
  { System.out.println("Tich cua 3 so "+x+" * " + y+"*"+z+" la:"+ (x*y*z));
  }
  public static void main(String args[])
  {Use_MyInterface uu=new Use_MyInterface();
  uu.add(4,8);
  uu.volume(20,15,9);
  System.out.println("Gia tri hang so price="+price);
  System.out.println("Gia tri hang so counter="+counter);
  }
}

```

Chú ý:

- + Nếu một lớp triển khai nhiều giao diện thì các giao diện này được liệt kê cách nhau bởi dấu phẩy ‘,’
- + 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
- + Do giao diện cũng là một lớp trừu tượng do vậy ta không thể tạo thể hiện của giao diện
- + 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
- + 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 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:

- Ø 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.
- Ø Spt là số phần tử của mảng.

Chú ý:

- Ø Mảng trong Java là một đối tượng
- Ø Cũng như các đối tượng khác, mảng phải được tạo ra bằng toán tử **new** như sau:
Tên_mảng=new KDL[spt];
- Ø 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 ký 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", "Hung Yên");
arraySinhVien[1]=new SinhVien("sv02", "Nguyễn Thị Bình", "Bắc Giang");
```

....

```
arraySinhVien[99]=new SinhVien("sv100", "Đào Thị Mên", "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 xuất 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:

KDL [][]...[] **tên_mảng;**
 N lần

hoặc

KDL **tên_mảng** [][]...[] =new **KDL**[spt1][spt2]...[sptN];
 N lần

Trong đó:

Ø KDL là một kiểu dữ liệu bất kỳ: nguyên thủy hoặc lớp

Ø 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**[][];

Ø 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, hàng 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 được 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 rỗng.
```

```
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' };
```

Tên lớp

```
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 substring)**

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)//Chuyển 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 họa 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:

- Nếu dung lượng(NewCapacity) mới lớn hơn đôi số được truyền cho phương thức “ensureCapacity()”, thì dung lượng mới (NewCapacity) được đặt.
- Nếu dung lượng mới nhỏ hơn đôi số được truyền cho phương thức “ensureCapacity()”, thì dung lượng được đặt bằng giá trị tham số truyền vào.

Chương trình sau minh họa 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 chiều 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 tùy 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, String)	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,bbb,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 để 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.

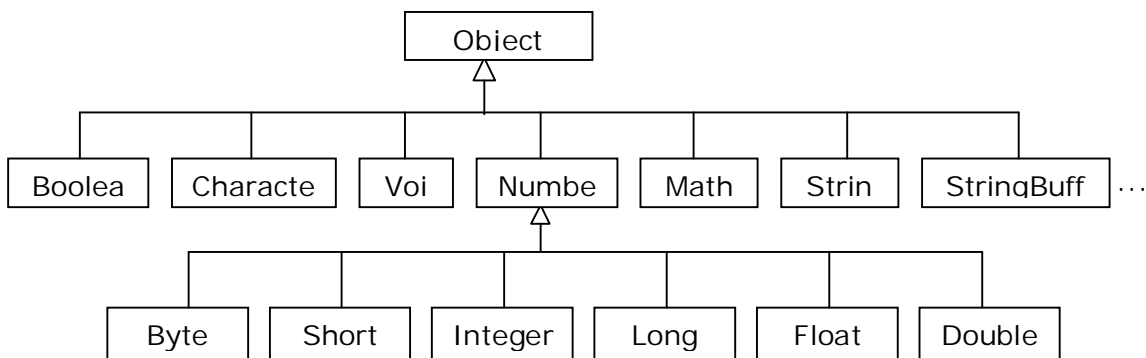
```

Command Prompt
D:\Temp>java StringTokenizerImplementer
Tokens f mathExpr:
4
*
3
+
2
/
4
Comma-delimited tokens :
field1
field2
field3
and field4
D:\Temp>

```

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 phân 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:

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);
```

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");
```

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

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");
```

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"
```

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'
```

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 nếu ch 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("Nhập vào " + n + " số nguyên");  
        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 tính 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 ($d1^{d2}$).

static double exp(double d)

+ Hàm **exp()** trả lại giá trị là lũy thừa cơ số e và số mũ d (e^d).

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()

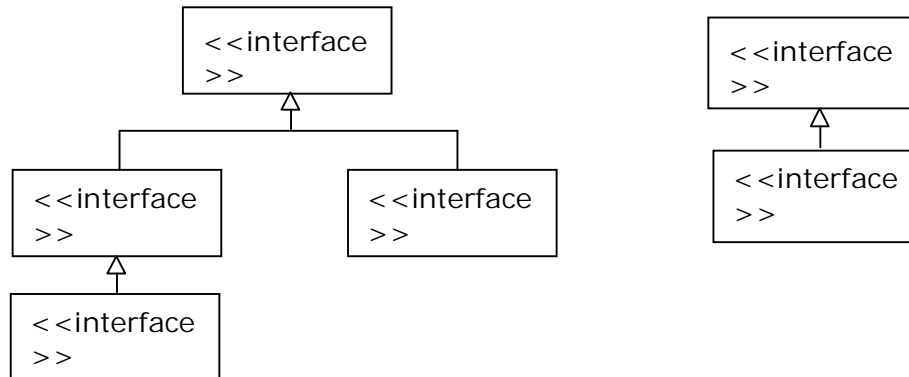
+ Hàm **random()** cho lại giá trị là số ngẫu nhiên trong khoảng từ 0.0 đến 1.0.

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 lỗi.

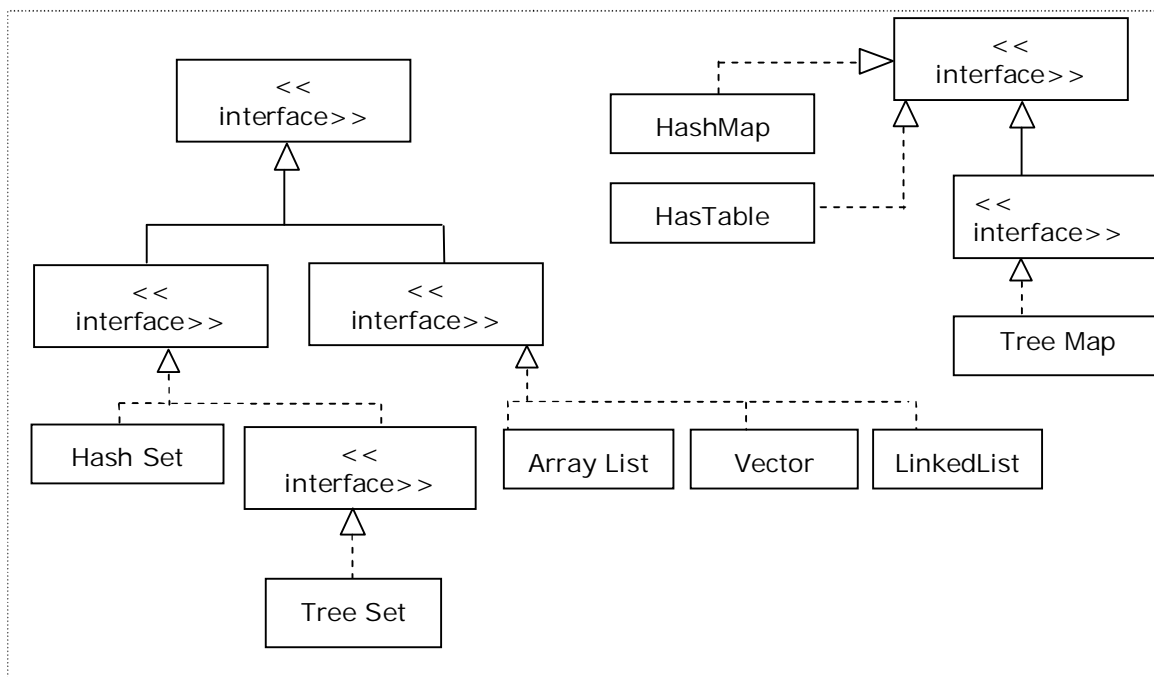


Các giao diện lỗi của cấu trúc Collection được mô tả trong bảng sau:

interface	Mô tả
Collection	<i>interface cơ sở định nghĩa tất cả các phép toán cơ bản cho các lớp cần duy trì thực hiện và cài đặt chúng</i>
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	<i>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</i>
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 lỗi 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 lỗi được xây dựng theo cấu trúc phân cấp như trong hình H6-3.



Các giao diện lồi và các lớp cài đặt chúng

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

Phân 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 containsAll(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

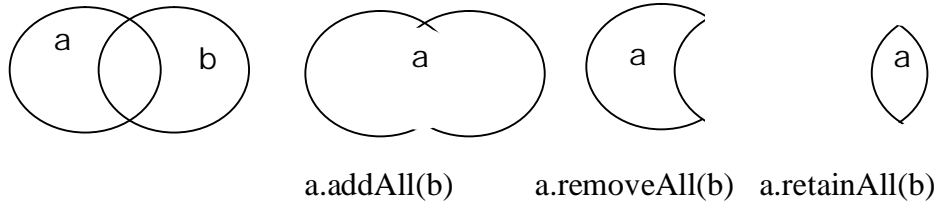
Thực hiện phép trừ hai tập hợp

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.



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()	$a = \emptyset$ (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 trùng 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 khoá phải là duy nhất (không cho phép lặp). Mỗi khoá 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.

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 NhómTrongluong {
    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 đầu (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 keylterator = keys.listIterator();
        // Tìm tần suất của các trọng lượng được nhập vào trong các nhóm
        while(keylterator.hasNext()) {
            Integer nhom = (Integer) keylterator.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:

```

          *
          *
1. **
      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:

- Lỗi cú pháp (syntac error)
 - Lỗi logic thuật toán
 - 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 nhiên 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 xảy ra JVM sẽ phát sinh một ngoại lệ, nếu một chương trình không cung cấp mã xử lý ngoại lệ có thể kết thúc không bình thường, trong bài hôm nay ta sẽ bàn về vấn đề xử 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 xử lý ngoại lệ của java

Mô hình xử 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 thiệu 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:

- Bạn phải chú ý giữa lệnh khai báo biệt lệ và lệnh ném ra ngoại lệ
- Một phương thức chỉ có thể ném ra các ngoại lệ mà nó được khai báo

IV. Bắt ngoại lệ

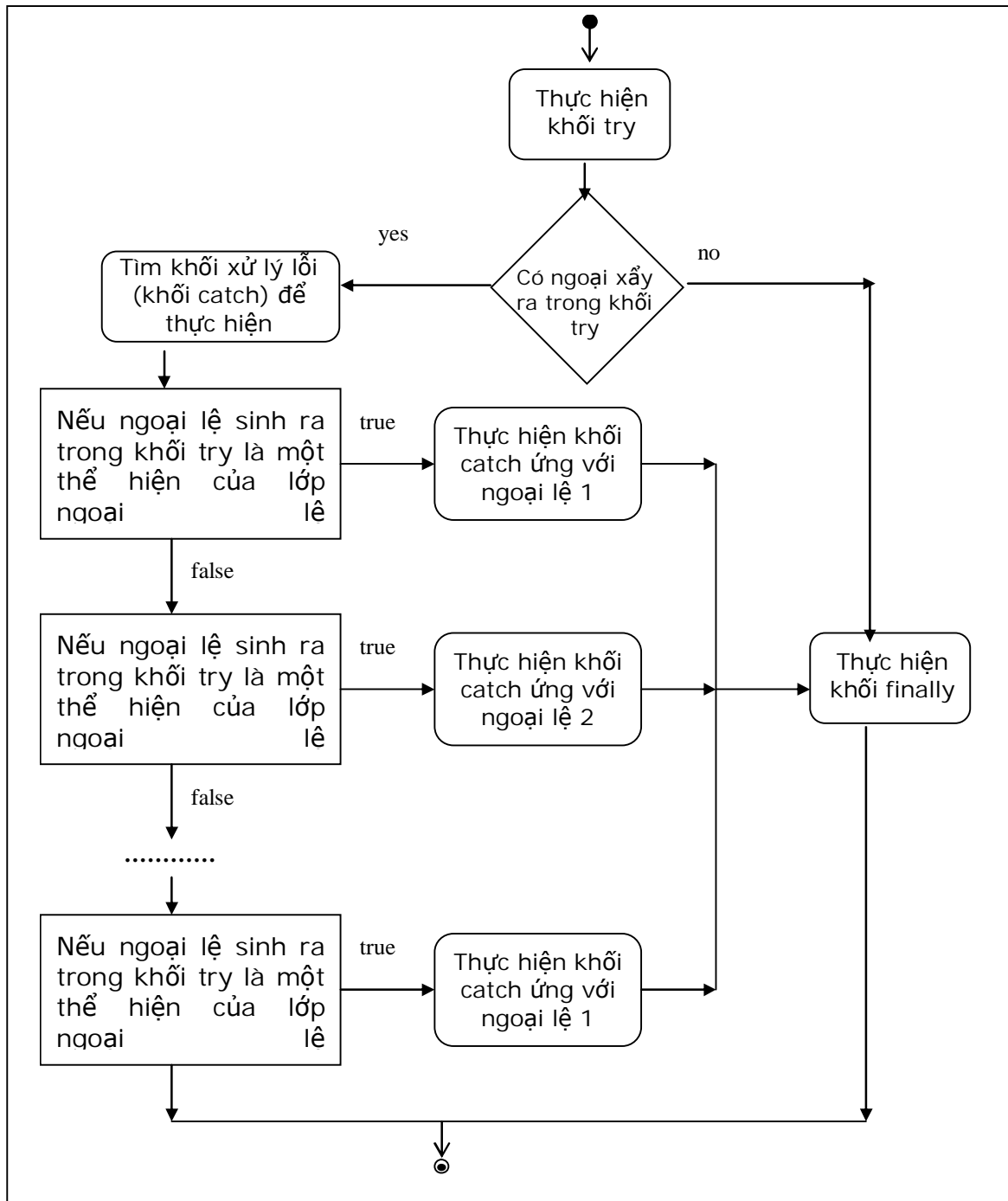
Một ngoại lệ (exception) trong chương trình Java là dấu hiệu chỉ ra rằng có sự xuất hiện một điều kiện không bình thường nào đó.

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 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 ngoại 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("Lỗi chia cho 0");}
    }
}

```

Khi chạy chương trình này ta sẽ thu được một dòng in ra màn hình như sau:
Lỗi 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 xảy 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 xảy ra, ta có thể lấy các 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,....

```

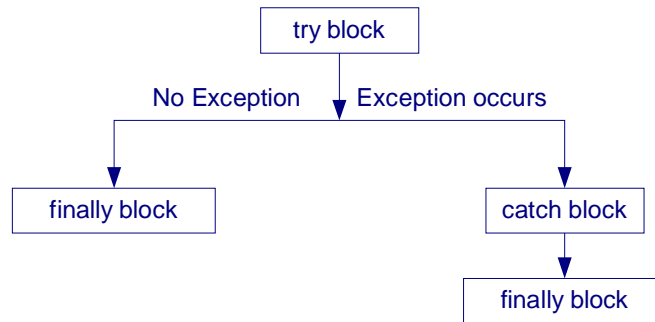
try{
    //Các lệnh có khả năng ném ra ngoại lệ
}
catch(Exception1 ex1){
    ...
}
catch(Exception2 ex2){
    ...
}
catch(Exceptionn exn){
    ...
}
finally{
    //Mã lệnh dọn dẹp
}

```

Khối 'finally' là tùy 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'.

Khởi ‘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 xảy ra trong ứng dụng
RuntimeException	Lớp cơ sở cho nhiều ngoại lệ java.lang
ArithmeticException	Lỗi về số học, ví dụ như ‘chia cho 0’.
IllegalAccessException	Lớp không thể truy cập.
IllegalArgumentException	Đối số không hợp lệ.
ArrayIndexOutOfBoundsException	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 thể

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

Ø HĐH đơn nhiệm là HĐH chỉ cho phép 1 tiến trình chạy tại một thời điểm, ví dụ HĐH DOS là HĐH đơn nhiệm.

Ø - HĐH đa nhiệm cho phép nhiều tiến trình chạy tại một thời điểm, ví dụ HĐH windows, Unix, Linux là các HĐH đa nhiệm

Ø HĐH đa nhiệm ưu tiên: các tiến trình được cấp phát thời gian sử dụng CPU theo mức ưu tiên khác nhau

Ø HĐH đa nhiệm không ưu tiên: các tiến trình không có mức ưu tiên nào cả, chúng “tự giác” nhà quyền kiểm soát CPU sau khi kết thúc phần công việc

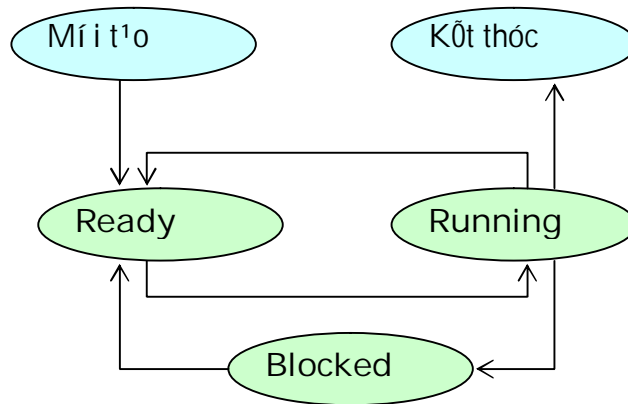
Chú ý: trong thực tế mỗi máy thường chỉ có 1 CPU, nên không thể có nhiều tiến trình chạy tại một thời điểm. Nên thông thường sự đa chương chỉ là giả lập. Chúng được giả lập bằng cách lưu trữ nhiều tiến trình trong bộ nhớ tại một thời điểm, và điều phối CPU qua lại giữa các tiến trình.

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 xảy 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:

1. Tạo mới: tiến trình đang được thành lập
2. Running: các chỉ thị của tiến trình đang được xử lý, hay nói cách khác tiến trình đang sở hữu CPU
3. Blocked: tiến trình đang chờ được cấp tài nguyên, hay chờ một sự kiện nào đó xảy ra
4. Ready: tiến trình đang chờ cấp CPU để xử lý
5. Kết thúc: tiến trình đã hoàn tất việc xử lý



Sự chuyển trạng thái của một tiến trình

5. Miền găng (Critical Section)

a) Vấn đề tranh chấp tài nguyên

Ta xét tình huống sau:

Ø giả sử A có 500\$ trong tài khoản ngân hàng

Ø A quyết định rút ra 100\$ từ tài khoản ngân hàng, thao tác của A gồm 2 bước:

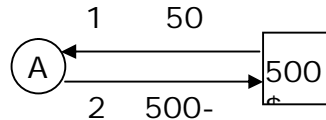
1) lấy ra 100\$

2) giảm số tài khoản đi 100\$

Ø Tình huống giữa 2 thao tác 1 và

2, B trả A 300\$, do vậy B cập

nhập 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 xảy 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 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:

Ø Mở rộng lớp Thread và Ghi đè phương thức run()

Ø 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:

Ø **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

Ø **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

Ø **public void stop()**

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

Ø **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 nhà 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()

Ø **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()

Ø **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.

Ø **public void interrupt()**

làm gián đoạn tuyến thi hành

Ø **public static boolean isInterrupt()**

kiểm tra xem tuyến có bị ngắt không

Ø **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]

Ø **public final void wait() throws InterruptedException**

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

- Ø **public final void notify ()**
đánh thức tuyến đang chờ, trên đối tượng này
- Ø **public final void notifyAll()** đánh thức tất cả các tuyến đang chờ trên đối tượng này
- Ø **isAlive()** Trả về True, nếu luồng là vẫn còn tồn tại (sống)
- Ø **getPriority()** Trả về mức ưu tiên của luồng
- Ø **join()** Đợi cho đến khi luồng kết thúc
- Ø **isDaemon()** Kiểm tra nếu luồng là luồng một luồng chạy ngầm (daemon)
- Ø **setDaemon(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”

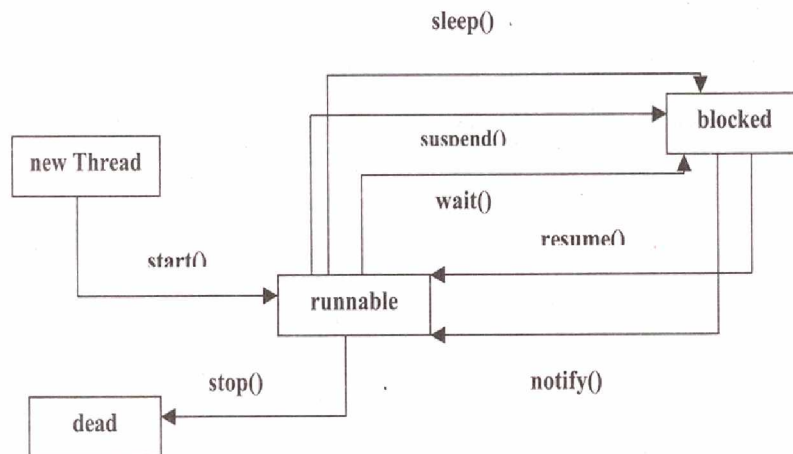
```
public class Hello{
public static void main ( String[] args ){
    new ChaoDH ().start ();
    new ChaoJV ().start ();
}
}
class ChaoDH extends Thread{
public void run (){
    for ( int i = 1; i <= 200; i++ )
        System.out.println ( "Đại học sư phạm kỹ thuật Hưng Yên" );
}
}
class ChaoJV extends 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 (daemon)

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 (daemon)

Người sử dụng tạo ra các luồng người sử dụng, trong khi các luồng daemon là các luồng chạy nền. Luồng daemon cung cấp các dịch vụ cho các luồng khác. Máy ảo Java thực hiện tiến trình thoát, khi đó chỉ còn duy nhất luồng daemon vẫn còn sống. Máy ảo Java có ít nhất một luồng daemon 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 daemon:

- **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) 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.

Ø 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

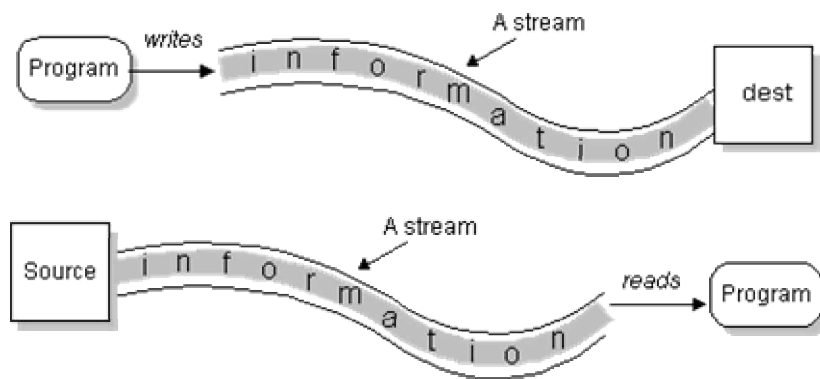
Mấu chốt của sự đồng bộ hóa là khái niệm “monitor” (giám sát), hay còn gọi “semaphore” (cờ hiệu). Một “monitor” là một đối tượng mà được khóa độc quyền. Chỉ một luồng có thể có monitor tại mỗi thời điểm. Tất cả các luồng khác cố gắng thâm nhập vào monitor sẽ bị trì hoãn, cho đến khi luồng đầu tiên thoát khỏi monitor. Các luồng khác được báo chờ đợi monitor. Một luồng có thể monitor một đối tượng nhiều lần.

Chương 5: NHẬP XUẤT (Input/Output)

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:

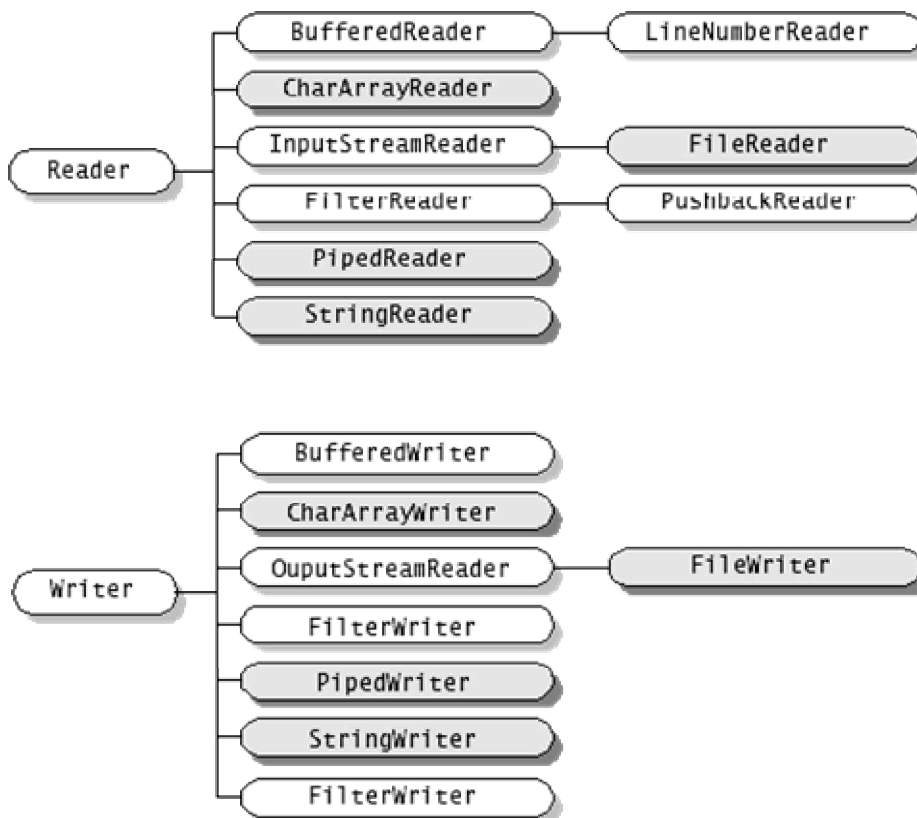
I. Lớp luồng

Java đưa ra nhiều lớp luồng, để xử lý mọi loại dữ liệu, java chia luồng ra thành 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.

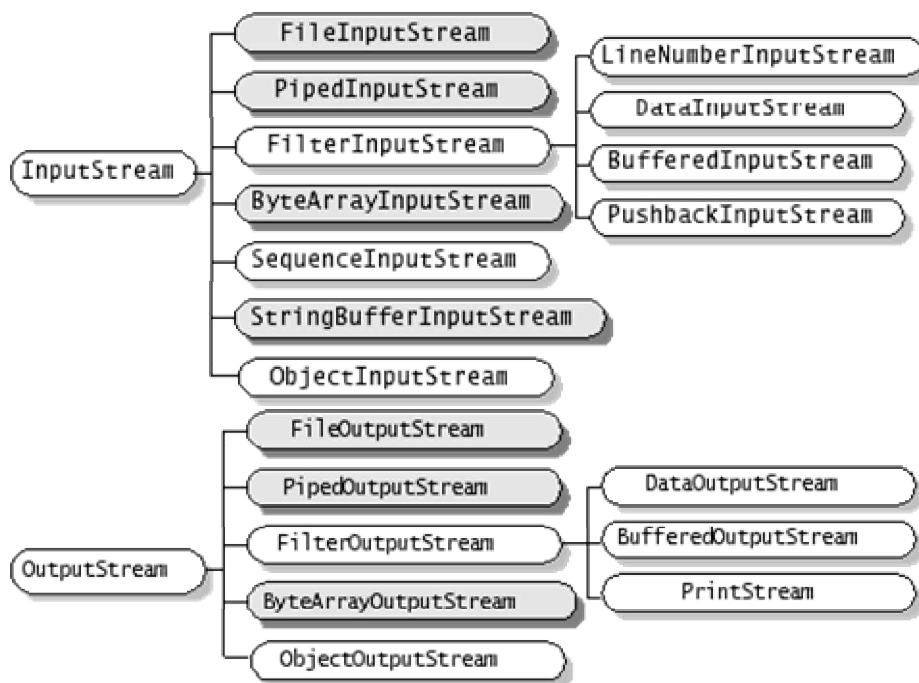
1.1. Lớp nhập, xuất hướng kí tự

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 đây là mô hình phân cấp các lớp đọc/ ghi hướng kí tự



1.2. Luồng hướng byte

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 lần 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 thủy, ta sử dụng luồng `DataInputStream` và `DataOutputStream`, lớp `DataInputStream` triển khai giao diện `DataInput`, còn lớp `DataOutputStream` triển khai giao diện `DataOutput`
 Các phương thức sau được định nghĩa trong giao diện `DataOutput`

<code>void write(byte[] b)</code>	Ghi một mảng byte ra luồng
<code>void write(byte[] b, int off, int len)</code>	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 thập của v
void writeBytes(String s)	Ghi một chuỗi ra luồng
void writeChar(int v)	Ghi một kí tự ra luồng
void writeChars(String s)	Ghi một chuỗi 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 chuỗi 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
void readFully(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 chuỗi 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
int readUnsignedShort()	đọc một số nguyên không dấu trong đoạn từ 0..65535
String readUTF()	đọc một chuỗi 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 lại:

```
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"));
out.writeInt(10); // ghi số nguyên
out.writeLong(123456789); // ghi số long
```

```

out.writeDouble(123.456789); // ghi số thực chính xác kép
out.writeFloat(123.456789f); // ghi số thực chính xác đơn
out.writeBoolean(true); // ghi giá trị logic
out.writeUTF("Day la mot xau ki tu"); // ghi một xâu
out.close();
// read it in again
DataInputStream in = new DataInputStream(new
FileInputStream("c:/TestIO.txt"));
try {
System.out.println("Gia tri nguyen " + in.readInt()); // đọc lại số nguyên
System.out.println("Gia tri long " + in.readLong()); // đọc lại số nguyên dài
System.out.println("Gia tri double " + in.readDouble()); // đọc lại số thực chính xác kép
System.out.println("Gia tri float " + in.readFloat()); // đọc lại số thực chính xác đơn
System.out.println("Gia tri boolean " + in.readBoolean()); // đọc lại giá trị logic
System.out.println("Gia tri xau " + in.readUTF()); // đọc lại một xâu unicode
}
catch (EOFException e) { System.out.println("loi");}
in.close();
}
}

```

1.3. Luồng in ẩn

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 ascii để xem được, trong java có thể sử dụng luồng in ẩn để xuất dữ liệu ra dưới dạng ascii. 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 ascii.

Một số phương thức của lớp PrintStream:

boolean checkError()	đón hết dữ liệu ra và kiểm tra lỗi luồng
void close()	đóng luồng
void flush()	đón 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 mảng 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 mảng 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 chuỗi 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 lên 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 dòng)

Một số phương thức của lớp *PrintWriter*

boolean checkError()	đón hết dữ liệu ra và kiểm tra lỗi luồng
void close()	đóng luồng
void flush()	đón 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 mảng 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 chuỗi
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 mảng 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 chuỗi 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 chuỗi
void write(String s, int off, int len)	Ghi một chuỗi 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 chuỗi kí tự, sau khi chạy chương trình bạn có thể sử dụng lệnh type của DOS để xem

```
import java.io.*;
public class DatalODemo1 {
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();
}
}
```

1.4. 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, BufferedReader, BufferedWriter, 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 *BufferedReader* (*Reader* in)

public *BufferedReader* (*Reader* in, int bufferSize)

public *BufferWriter* (*Writer* out)

public *BufferWriter* (*Writer* out, int bufferSize)

Tập 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 từ 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 chuỗi 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 họa 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:

```

C:\winnt\System32\cmd.exe
G:\Temp>java rndexam
67868
J
678.68
true
G:\Temp>

```

II. 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 chuỗi 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 chuỗi 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.

Một số ví dụ ứng dụng

Ví dụ 1: Ví dụ minh họa việc nhập dữ liệu từ bàn phím

```

package app001;
import java.lang.*;
import java.io.*;
import java.util.*;
public class InputOutput
{ public static void main(String srgs[]) throws IOException
{ BufferedReader klnput=new BufferedReader(new InputStreamReader(System.in));
  String ten,nam;
  do {
    System.out.print("Nhap ho va ten:");ten=klnput.readLine();
    System.out.print("Nhap nam sinh:");nam=klnput.readLine();

```

```

}while ((ten==null)||(nam==null));
klnput.close();
int ns=Integer.parseInt(nam);
System.out.println("Ho ten ban vua nhap =" +ten);
System.out.println("Nam sinh ban vua nhap =" +ns);
}
}

```

Ví dụ 2 Xây dựng gói nhập dữ liệu cho riêng mình

```

package app002;
import java.io.*;
import java.util.*;
public class MyInput{
    static private StringTokenizer stok;
    static private BufferedReader br =
        new BufferedReader(new InputStreamReader(System.in));
    public static int readInt(){
        int i = 0;
        try{ String str = br.readLine();
            stok = new StringTokenizer(str);
            i = new Integer(stok.nextToken()).intValue();
        }
        catch (IOException ex){
            System.out.println(ex);
        }
        return i;
    }
    public static double readDouble(){
        double d = 0;
        try{
            String str = br.readLine();
            stok = new StringTokenizer(str);
            d = new Double(stok.nextToken()).doubleValue();
        }
        catch (IOException ex){
            System.out.println(ex);
        }
        return d;
    }
    public static String readString(){
        String S=" ";
        try{ S = br.readLine();}
        catch (IOException ex){
            System.out.println(ex);
        }
        return S;
    }
}

```

Ví dụ 3 Sử dụng gói nhập dữ liệu trong chương trình Java

```
import app002.MyInput;
public class TestInput{
    public static void main(String[] args){
        System.out.print("Enter an integer: ");
        int i = MyInput.readInt();
        System.out.println("\nYou entered i = " + i);
        System.out.print("Enter a double: ");
        double d = MyInput.readDouble();
        System.out.println("\nYou entered d = " + d);
        System.out.print("Enter a String: ");
        String SS=MyInput.readString();
        System.out.println("\nYou entered S = " + SS);
    }
}
```

Ví dụ 4. Minh họa việc đọc, ghi file Text

```
package app001;
import java.lang.*;
import java.io.*;
import java.util.*;
import app001.*;
public class IOFile {
    public static void main(String args[]) throws IOException
    { BufferedReader FileInput=new BufferedReader(new FileReader("C:\\filein.txt"));
      PrintWriter FileOut=new PrintWriter(new BufferedWriter(new
FileWriter("C:\\fileout.txt")));
      String ten=" ",nam="345";
      long n=0;
      while (FileInput.ready()==true) //==While (not eof())
      { ten = FileInput.readLine();
        nam = FileInput.readLine();
        System.out.println("Ho ten=" + ten);
        System.out.println("nam sinh=" + nam);
      }
      FileInput.close();
      int ns = Integer.parseInt(nam);
      FileOut.println(ten);
      FileOut.println(nam);
      FileOut.close();
      Man ng1=new Man(ten,ns);
      ng1.hienthi();
      ng1.HienThiNgay();
    }
}
```

Ví dụ 5: Minh họa sử dụng luồng

```

package app002;
import java.io.FileOutputStream;
import java.io.FileInputStream;
import java.io.File;
import java.io.IOException;
public class FileIOExam0 {
    public static void main(String args[]) throws IOException
    {String s="Welcome to File Input Output Stream...";
        //Mo file abc.txt de doc
        FileOutputStream os=new FileOutputStream("C:\\abc.txt");
        for(int i=0;i<s.length();i++) os.write(s.charAt(i)); //Ghi tung byte
        os.close(); //Dong file abc.txt
        //Mo file abc.txt de doc
        FileInputStream is=new FileInputStream("C:\\abc.txt");
        int ibyts=is.available(); //Lay so byte cua file abc.txt
        byte ibuf[]=new byte[ibyts]; //Khai bao dung bo dem du lieu
        int byrd=is.read(ibuf,0,ibyts); //Doc ibyts du lieu ra luong
        System.out.println("They are:"+new String(ibuf)); //Du lieu trong file
        System.out.println("Input Stream has:" + ibyts+" available bytes");
        System.out.println("Number of Bytes read are:"+byrd);
        is.close();
        //Xoa file abc.txt
        // File f1=new File("c:\\abc.txt");
        // f1.delete();
    }
}

```

Ví dụ 6: Sử dụng file ngẫu nhiên

```

package welcome;
import java.lang.System;
import java.io.RandomAccessFile;
import java.io.IOException;
public class RandomAaccessFile0
{
    public static void main(String args[]) throws IOException
    { RandomAccessFile rf=new RandomAccessFile("c:\\abcd.txt", "rw");
        rf.writeBoolean(true);
        rf.writeInt(2345);
        rf.writeChars("A");
        rf.writeDouble(456.78);
// int a[10]={2,4,6,9,1,5,8,0,12,90};
        int A[]=new int[10];
        A[0]=2;A[1]=1;A[2]=9;A[3]=7;A[4]=8;A[5]=3;A[6]=4;A[7]=5;A[8]=12;A[9]=21;
        for(int i=0;i<10;i++)
            rf.writeInt(A[i]);
        rf.seek(0);
        System.out.println(rf.readBoolean());
        System.out.println(rf.readInt());
    }
}

```

```
System.out.println(rf.readChar());  
System.out.println(rf.readDouble());  
for(int i=0;i<10;i++) System.out.println(rf.readInt());  
rf.close();  
}  
}
```

Chương 6: LẬP TRÌNH ĐỒ HOẠ AWT

Một khả năng của Java là cho phép ta xây dựng các ứng dụng có giao diện đồ họa hay còn gọi là GUI (Graphical User Interface). Khi Java được phát hành, các thành phần đồ họa 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 trực 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 họa của Java

1. Giới thiệu chung

Thiết kế API cho lập trình đồ họa của Java là một ví dụ hoàn hảo về cách dùng lớp, sự kế thừa và giao diện. API cho lập trình đồ họa 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. Mỗi quan hệ kế thừa giữa các thành phần này được mô tả trong hình sau:

Component 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.

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 là `JFrame` và `JPanel`.

JComponent Là lớp cha của mọi thành phần Swing *light weight*, được vẽ trực tiếp lên canvas bằng mã lệnh Java.

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`.

JFrame là cửa sổ không lồng bên trong cửa sổ khác.

JDialog là một cửa sổ được hiển thị dưới dạng modal.

JApplet là lớp cha của mọi lớp ứng dụng applet.

JPanel là một vật chứa, lưu giữ các thành phần giao diện người dùng.

Graphics là lớp trừu tượng cung cấp ngữ cảnh đồ họa để vẽ các đối tượng đồ họa như: Đường thẳng, đường tròn, hình ảnh...

Color lớp này biểu diễn một màu sắc.

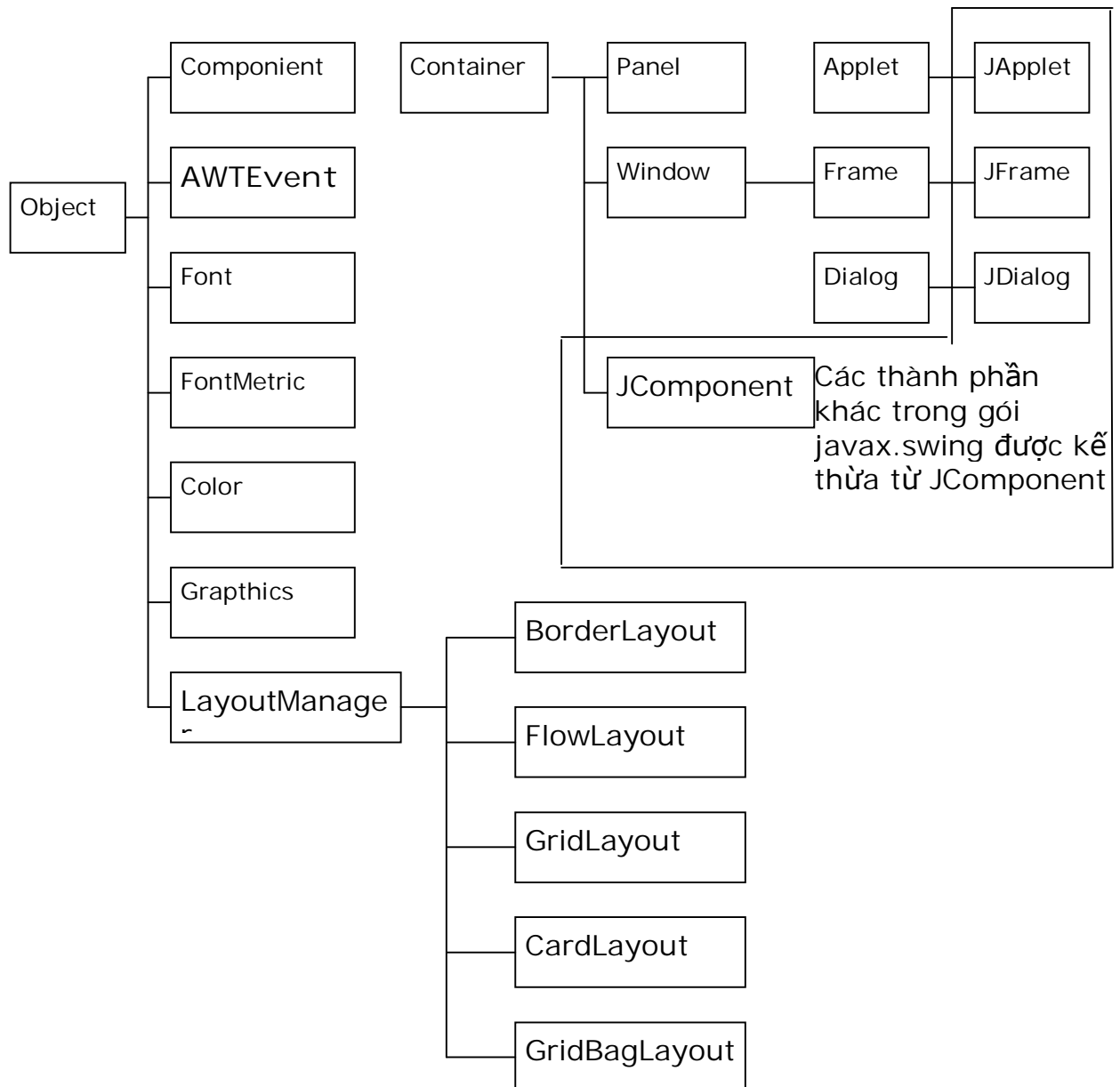
Font lớp này biểu thị cho một font đồ họa.

FontMetrics là một lớp trừu tượng dùng để xác định các thuộc tính của Font.

Tất cả các thành phần đồ họa 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 menu có tên là JMenu.

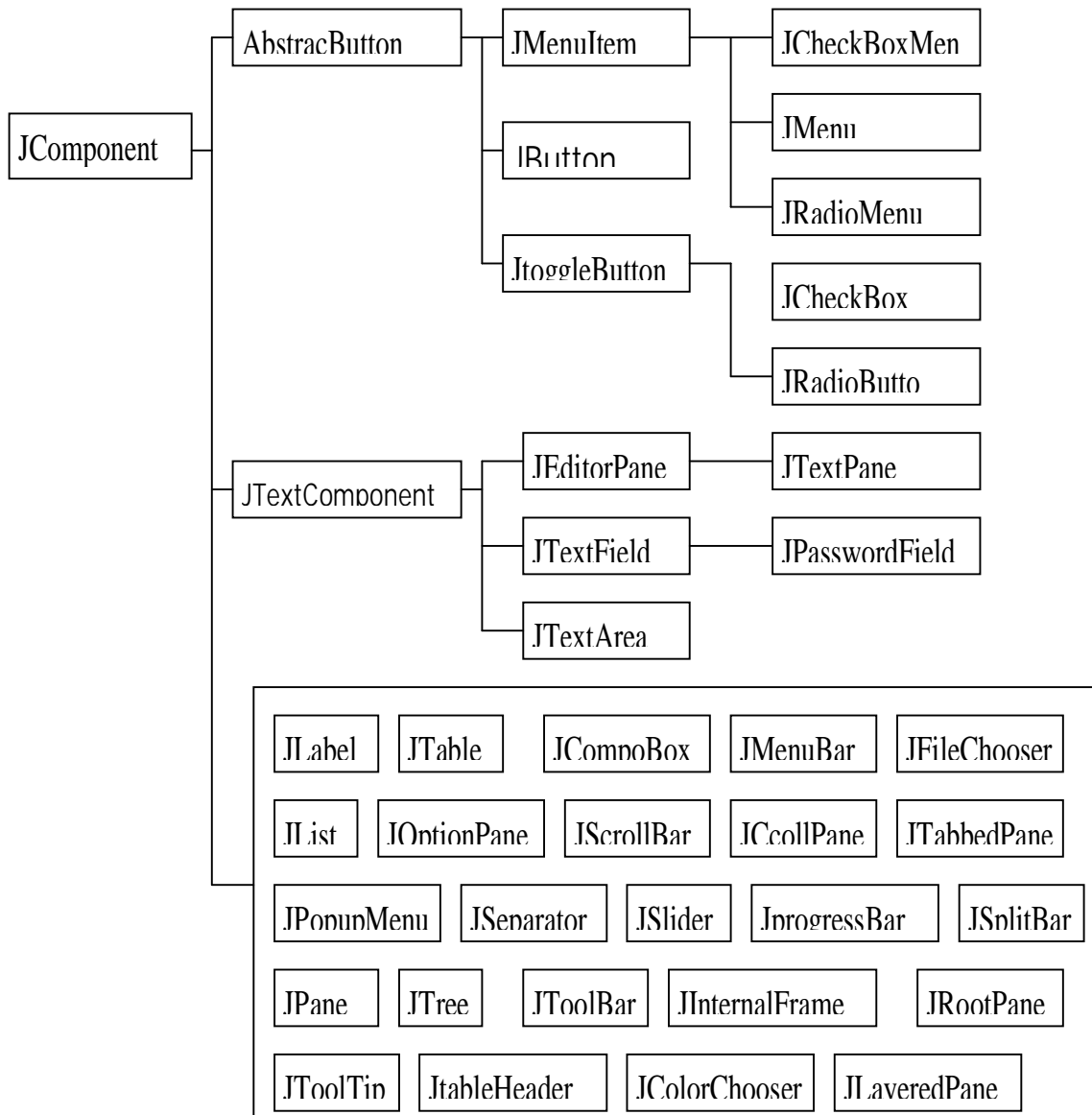
Tất cả những lớp khác được liệt kê trong hình dưới đây

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:

- Tạo ra một frame có tiêu đề gì đó, ví dụ “My Frame” :

```
JFrame myWindow= new JFrame(“My Frame”);
```

- 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

- Đặt lại kích thước cho frame sử dụng hàm setSize():
myWindow.setSize(200, 300);// Đặt lại khung frame là 200 (300
- Gói khung frame đó lại bằng hàm pack():
myWindow.pack();
- 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 sắp 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)

Component add(Component comp, Object constraints)

Component 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 alignment)

FlowLayout(int alignment, int horizontalgap, int verticalgap)

public static final int LEFT

public static final int CENTER

public static final int RIGHT

Đối số alignment xác định cách sắp theo hàng: từ trái, phải hay trung tâm, horizontalgap 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 horizontalgap, int verticalgap)

Tạo ra một lưới hình chữ nhật có rows (columns ô có khoảng cách giữa các hàng các cột là horizontalgap, 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 horizontalgap, int verticalgap)

Tạo ra một layout mặc định hoặc có khoảng cách giữa các thành phần (tính bằng pixel) là horizontalgap 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

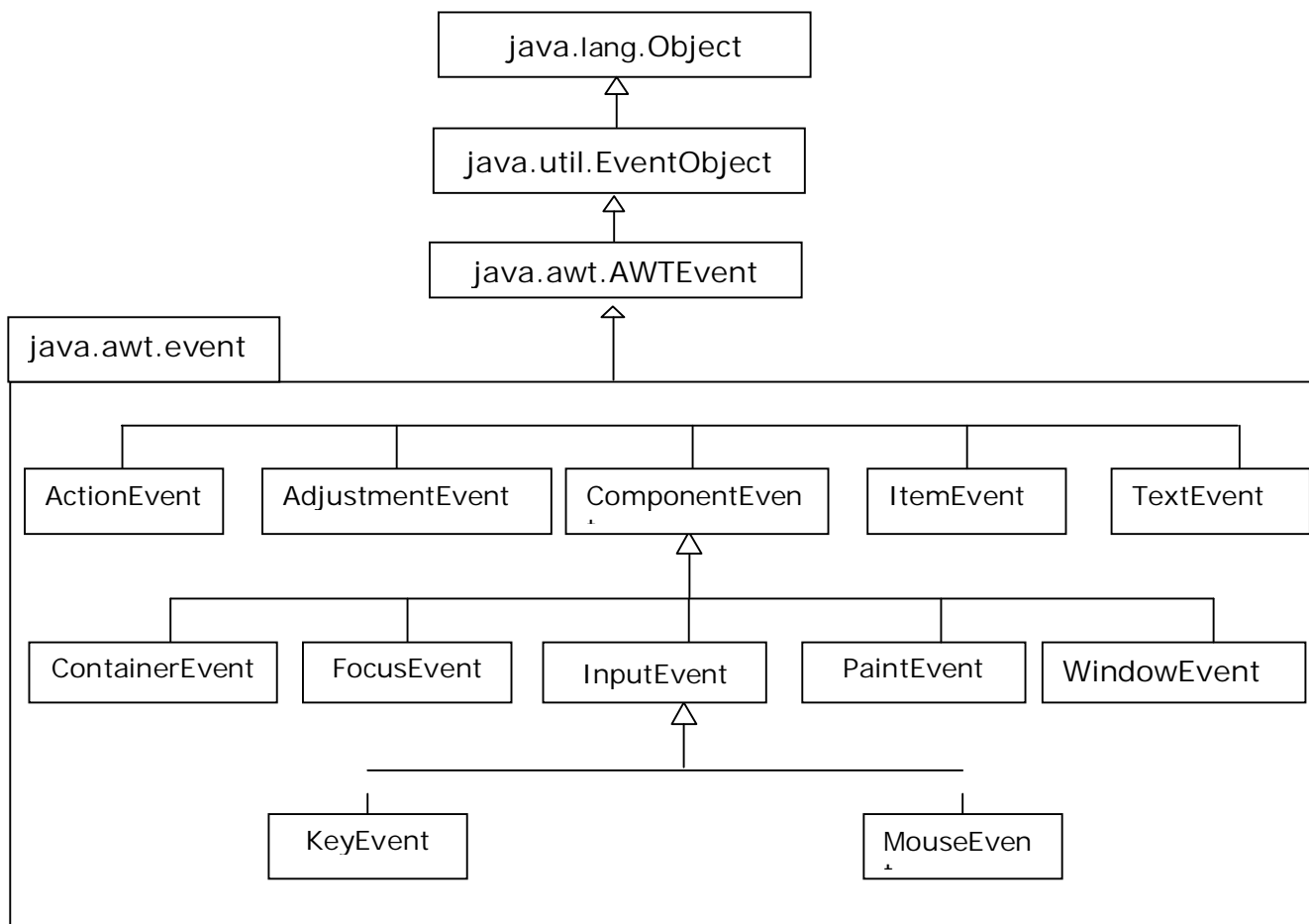
public static final String WEST

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:

Các lớp mô tả về ngữ nghĩa của các sự kiện,
 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:

- *JButton* - khi một nút button được kích hoạt,
- *JList* - khi một mục trong danh sách được kích hoạt đúp,
- *JmenuItem*, *JcheckBoxMenu*, *JradioMenu* - khi một mục trong thực đơn được chọn,
- *JTextField* - khi gõ phím ENTER trong trường văn bản (text).

b. AdjustmentEvent

Sự kiện này xảy ra khi ta điều chỉnh (adjustment) giá trị thanh cuộn (JScrollbar)

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:

- JCheckbox - khi trạng thái của hộp kiểm tra Checkbox thay đổi.
- CheckboxMenuItem - khi trạng thái của hộp kiểm tra Checkbox ứng với mục của thực đơn thay đổi.
- JRadioButton- khi trạng thái của hộp chọn (Option) thay đổi.
- JList - khi một mục trong danh sách được chọn hoặc bị loại bỏ chọn.
- 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:

- TextArea - khi kết thúc bằng nhấn nút ENTER,
- 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 xảy 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 đăng ký, gỡ bỏ đối tượng lắng nghe	Giao diện Listener lắng nghe tương ứng
ActionEvent	JButton JList TextField	addComponentListener removeActionListener	ActionListener
AdjustmentEvent	JScrollbar	addAdjustmentListener removeAdjustmentListener	AdjustmentListener
ItemEvent	JCheckbox JCheckboxMenuItem JRadioButton JList JComboBox	addItemListener removeItemListener	ItemListener
TextEvent	JTextArea JTextField JTextPane JEditorPane	addTextListener 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	MouseMotionLi

		remoMouseListener addMouseMotionListener remoMouseMotionListener	stener
WindowEvent	Window	addWindowListener removeWindowListener	WindowListener

3. Một số lớp điều hợp

Giao diện Listener lắng nghe	Lớp điều hợp tương ứng
ActionListener	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

Một số ví dụ minh họa

/* The applet enables the user to enter the interest rate, the number of years, and the loan amount. Clicking the compute button displays the monthly payment and the total payment.
*/

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.TitledBorder;
import java.applet.Applet;
import javax.swing.*;
public class StudentApp extends JFrame implements ActionListener{
    private JTextField tRate = new JTextField(10);
    private JTextField tYear = new JTextField(10);
    private JTextField tLoan = new JTextField(10);
    private JTextField tTotal = new JTextField(10);
    private JButton compute = new JButton("Compute mortgage");
    public static void main(String[] args){
        StudentApp fr = new StudentApp();
        fr.setSize(400, 200);
        fr.setTitle("Mortgage Application");
        //fr.center();
        fr.setVisible(true);
    }
}
```

```

public StudentApp(){
    tTotal.setEditable(false);
    JPanel p1 = new JPanel();
    p1.setLayout(new GridLayout(5,2));
    p1.add(new Label("Interest Rate"));
    p1.add(tRate);
    p1.add(new Label("Years"));
    p1.add(tYear);
    p1.add(new Label("Loan Amount"));
    p1.add(tLoan);
    p1.add(new Label("Total Payment"));
    p1.add(tTotal);
    p1.setBorder(new TitledBorder("Enter interest rate, year and loan amount"));
    JPanel p2 = new JPanel();
    p2.setLayout(new FlowLayout(FlowLayout.RIGHT));
    p2.add(compute);
    // Add the components to the applet
    getContentPane().add(p1, BorderLayout.CENTER);
    getContentPane().add(p2, BorderLayout.SOUTH);
    // Register listener
    compute.addActionListener(this);
}
// Handler for the compute button
public void actionPerformed(ActionEvent e){
    if(e.getSource() == compute){
        // Get values from text fields
        double interest = (Double.valueOf(tRate.getText())).doubleValue();
        int year = (Integer.valueOf(tYear.getText())).intValue();
        double loan = (Double.valueOf(tLoan.getText())).doubleValue();
        // Compute the total and displays it
        double m = loan * interest * year;
        tTotal.setText(String.valueOf(m));
    }
}
}

```

Sử dụng ListBox và nút lệnh

```

import java.awt.*;
import java.awt.event.*;
class ChoiceList extends Frame implements ActionListener{
    Label l1 = new Label("What is your favorite color");
    Choice colors = new Choice();
    Button bt = new Button("Exit");
    public ChoiceList(String title){
        super(title);
        setLayout(new FlowLayout());
        add(bt);
        add(l1);
    }
}

```

```

        colors.addItem("White");
        colors.addItem("Red");
        colors.addItem("Orange");
        colors.addItem("Green");
        colors.addItem("Yellow");
        colors.addItem("Blue");
        add(colors);
        bt.addActionListener(this);
    }
    public void actionPerformed(ActionEvent ev){
        if(ev.getSource() == bt){
            System.exit(0);
        }
    }
    // public void mouseExited(MouseEvent m){}
    public static void main(String args[]){
        ChoiceList t = new ChoiceList("Choice List");
        t.setSize(300, 200);
        t.show();
    }
}

```

```

/* ClockGroup.java
 * display a group of international clocks
 */

```

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;
import javax.swing.Timer;
import java.text.*;
public class ClockGroup extends JApplet implements ActionListener{
    private ClockPanel clockP1, clockP2, clockP3;
    private JButton btResum, btSus;
    public static void main(String[] arg){
        JFrame fr = new JFrame("Clock Group Demo");
        ClockGroup app = new ClockGroup();
        fr.getContentPane().add(app, BorderLayout.CENTER);
        app.init();
        app.start();
        fr.setSize(600, 350);
        fr.setVisible(true);
    }
    public void init(){
        // Panel p1 for holding 3 clocks
        JPanel p1 = new JPanel();
        p1.setLayout(new GridLayout(1,3));
        // Create a clock for Berlin
        p1.add(clockP1 = new ClockPanel());
        clockP1.setTitle("Berlin");
    }
}

```

```

clockP1.clock.setTimeZoneID("ECT");
clockP1.clock.setLocale(Locale.GERMAN);
// Create a clock for SanFrancisco
p1.add(clockP2 = new ClockPanel());
clockP2.setTitle("San Francisco");
clockP2.clock.setTimeZoneID("PST");
clockP2.clock.setLocale(Locale.US);
// Create a clock for Taipei
p1.add(clockP3 = new ClockPanel());
clockP3.setTitle("Taipei");
clockP3.clock.setLocale(Locale.CHINESE);
clockP3.clock.setTimeZoneID("CTT");
// Panel p2 for holding 2 group buttons
JPanel p2 = new JPanel();
p2.setLayout(new FlowLayout());
p2.add(btResum = new JButton("Resume All"));
p2.add(btSus = new JButton("Suspend All"));
// Add p1, p2 into the applet
getContentPane().setLayout(new BorderLayout());
getContentPane().add(p1, BorderLayout.CENTER);
getContentPane().add(p2, BorderLayout.SOUTH);
// Register listener
btResum.addActionListener(this);
btSus.addActionListener(this);
}
public void actionPerformed(ActionEvent e){
    if(e.getSource() == btResum){
        // Start all clock
        clockP1.resume();
        clockP2.resume();
        clockP3.resume();
    }
    else if (e.getSource() == btSus){
        // Start all clock
        clockP1.suspend();
        clockP2.suspend();
        clockP3.suspend();
    }
}
}
class ClockPanel extends JPanel implements ActionListener{
    private JLabel labelT;
    protected Clock clock = null;
    private JButton btResum, btSus;
    public ClockPanel(){
        JPanel bt = new JPanel();
        bt.add(btResum = new JButton("Resume"));
        bt.add(btSus = new JButton("Suspend"));
        // Set BorderLayout for the ClockPanel

```

```

        setLayout(new BorderLayout());
        // Add title label to the north of the panel
        add(labelT = new JLabel(), BorderLayout.NORTH);
        labelT.setHorizontalAlignment(JLabel.CENTER);
        // Add the clock to the center of the panel
        add(clock = new Clock(),BorderLayout.CENTER);
        add(bt,BorderLayout.SOUTH);
        // Register
        btResum.addActionListener(this);
        btSus.addActionListener(this);
    }
    public void setTitle(String title){
        labelT.setText(title);
    }
    public void actionPerformed(ActionEvent e){
        if(e.getSource() == btResum){
            clock.resume();
        }
        else if (e.getSource() == btSus){
            clock.suspend();
        }
    }
    public void resume(){
        if(clock != null) clock.resume();
    }
    public void suspend(){
        if(clock != null) clock.suspend();
    }
}
class Clock extends StillClock implements ActionListener{
    protected Timer timer;
    public Clock(){
        this(Locale.getDefault(), TimeZone.getDefault());
    }
    public Clock(Locale l, TimeZone t){
        super(l, t);
        timer = new Timer(1000,this);
        timer.start();
    }
    public void resume(){
        timer.start();
    }
    public void suspend(){
        timer.stop();
    }
    public void actionPerformed(ActionEvent e){
        repaint();
    }
}

```

```

// Display a clock in JPanel
class StillClock extends JPanel{
    protected TimeZone timeZ;
    protected int xC, yC;
    protected int clockR;
    protected DateFormat form;
    public StillClock(){
        this(Locale.getDefault(), TimeZone.getDefault());
    }
    public StillClock(Locale l, TimeZone t){
        setLocale(l);
        this.timeZ = t;
    }
    public void setTimeZoneID(String newT){
        timeZ = TimeZone.getTimeZone(newT);
    }
    // Override the paintComponent to display a clock
    public void paintComponent(Graphics g){
        super.paintComponent(g);
        clockR = (int)(Math.min(getSize().width,getSize().height)*0.7*0.5);
        xC = (getSize().width)/2;
        yC = (getSize().height)/2;
        // Draw circle
        g.setColor(Color.black);
        g.drawOval(xC-clockR, yC - clockR, 2*clockR,2*clockR);
        g.drawString("12", xC - 5, yC - clockR);
        g.drawString("9", xC - clockR-10, yC + 3);
        g.drawString("3", xC + clockR, yC + 3);
        g.drawString("6", xC + 3, yC + clockR + 10);
        // Get current time using GregorianCalendar
        GregorianCalendar cal = new GregorianCalendar(timeZ);
        // Draw second hand
        int second = (int) cal.get(GregorianCalendar.SECOND);
        int sLen = (int)(clockR *0.9);
        int xS = (int)(xC+sLen*Math.sin(second*(2*Math.PI/60)));
        int yS = (int)(yC-sLen*Math.cos(second*(2*Math.PI/60)));
        g.setColor(Color.red);
        g.drawLine(xC, yC, xS, yS);
        // Draw minut hand
        int minute = (int) cal.get(GregorianCalendar.MINUTE);
        int mLen = (int)(clockR *0.75);
        int xM = (int)(xC+mLen*Math.sin(minute*(2*Math.PI/60)));
        int yM = (int)(yC-mLen*Math.cos(minute*(2*Math.PI/60)));
        g.setColor(Color.blue);
        g.drawLine(xC, yC, xM, yM);
        // Draw hour hand
        int hour = (int) cal.get(GregorianCalendar.HOUR_OF_DAY);
        int hLen = (int)(clockR *0.6);
        int xH = (int)(xC+hLen*Math.sin((hour+minute/60.0)*(2*Math.PI/12)));
    }
}

```

```

        int yH = (int)(yC-hLen*Math.cos((hour+minute/60.0)*(2*Math.PI/12)));
        g.setColor(Color.green);
        g.drawLine(xC, yC, xH, yH);
        // Set display form in specified style, locale, and timzone
        form = DateFormat.getDateInstance(
            DateFormat.MEDIUM,DateFormat.LONG, getLocale());
        form.setTimeZone(timeZ);
        // Display current date
        String today = form.format(cal.getTime());
        FontMetrics fm = g.getFontMetrics();
        g.drawString(today, (getSize().width-fm.stringWidth(today))/2,
yC+clockR+30);
    }
}

```

Sử dụng các điều khiển

```

package welcome;
import java.awt.*;
import java.awt.event.*;
public class ComponentTest extends Frame implements
ActionListener,MouseListener
{ TextField tf1=new TextField(30);
  TextField tf2 = new TextField(40);
  Label label1 = new Label("This is a way to use component...");
  Checkbox b1 = new Checkbox("Red", true);
  Checkbox b2 = new Checkbox("Blue", true);
  CheckboxGroup cb = new CheckboxGroup();
  Checkbox bg1 = new Checkbox("Small", cb, true);
  Checkbox bg2 = new Checkbox("Large", cb, true);
  TextArea ta = new TextArea(3, 25);
  Choice colors = new Choice();
  Button btnResult = new Button("Double is");
  Button btnExt = new Button("Exit");
  PopupMenu optionsMenu = new PopupMenu("Options");
public ComponentTest(String title) {
    super(title);
    setLayout(new FlowLayout());
    add(label1);
    add(ta);
    add(tf1);
    add(tf2);
    add(b1);
    add(b2);
    add(bg1);
    add(bg2);
    colors.addItem("While");
    colors.addItem("Red");
    colors.addItem("Oranges");

```

```

colors.addItem("Green");
colors.addItem("Yellow");
colors.addItem("Blue");
colors.addItem("Black");
add(colors);
add(btnResult);
add(btnExt);
btnResult.addActionListener(this);
btnExt.addActionListener(this);
MenuBar mbar = new MenuBar();
setMenuBar(mbar);
Menu fileMenu = new Menu("File");
mbar.add(fileMenu);
fileMenu.addActionListener(this);
MenuItem newItem = new MenuItem("New");
fileMenu.add(newItem);
MenuItem openItem = new MenuItem("Open");
fileMenu.add(openItem);
fileMenu.addSeparator();
MenuItem saveItem = new MenuItem("Save");
fileMenu.add(saveItem);
fileMenu.addSeparator();
MenuItem exitItem = new MenuItem("Exit");
fileMenu.add(exitItem);
exitItem.addActionListener(this);

Menu editMenu = new Menu("Edit");
mbar.add(editMenu);
editMenu.addActionListener(this);
MenuItem cutItem = new MenuItem("Cut");
editMenu.add(cutItem);
MenuItem copyItem = new MenuItem("Copy");
editMenu.add(copyItem);
fileMenu.addSeparator();
MenuItem pasteItem = new MenuItem("Paste");
editMenu.add(pasteItem);
editMenu.add(optionsMenu);
//PopupMenu optionsMenu=new PopupMenu("Options");
optionsMenu.addActionListener(this);
MenuItem readItem = new MenuItem("Read Only");
optionsMenu.add(readItem);
optionsMenu.addSeparator();
Menu formatMenu = new Menu("Format text");
optionsMenu.add(formatMenu);
optionsMenu.addSeparator();
CheckboxMenuItem insertItem = new CheckboxMenuItem("Insert", true);
formatMenu.add(insertItem);
formatMenu.addSeparator();
CheckboxMenuItem overtypeItem = new CheckboxMenuItem("Over type", false);

```



```

formatMenu.add(overtypeltem);
formatMenu.addSeparator();
}
public void actionPerformed(ActionEvent ae) {
    if (ae.getSource() == btnResult) {
        int num = Integer.parseInt(tf1.getText()) * 2;
        tf2.setText(String.valueOf(num));
    }
    if (ae.getSource() == btnExt) {
        System.exit(0);
    }
    if (ae.getActionCommand().equals("Exit")) {
        System.exit(0);
    }
    if (ae.getActionCommand().equals("Open")) {
        System.out.println("You pressed the menuItem Open...");
        label1.setText("Open");
    }
}
public void mouseEntered(MouseEvent m) { }
public void mouseExited(MouseEvent m) { }
public void mouseClicked(MouseEvent m) {
    optionsMenu.show(this, m.getX(), m.getY());
}
public void mouseReleased(MouseEvent m) { }
public void mousePressed(MouseEvent m) { }
public static void main(String args[]) {
    ComponentTest c = new ComponentTest("Vi du dung thu vien java.awt.*");
    c.setSize(500, 300);
    c.show();
}
}
}

```

/*

The program has 4 buttons labeled Add, Sub, Mul, Div and also create a menu to perform the same operation. The user will also create to perform the same operation. The user can choose the operation either from buttons or from menu selections. Fig. The following contains a sample run of the program.

*/

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class MenuDemo1 extends JFrame implements ActionListener{
    private JTextField num1, num2, res;
    private JButton btAdd, btSub, btMul, btDiv;
    private JMenuItem mAdd, mSub, mMul, mDiv, mClose;
    public static void main(String[] arg){
        MenuDemo1 fr = new MenuDemo1();
        fr.pack();
    }
}

```

```

        fr.setVisible(true);
    }
    public MenuDemo1(){
        setTitle("Menu Demo");
        JMenuBar mb = new JMenuBar();
        setJMenuBar(mb);
        JMenu opMenu = new JMenu("Operation");
        opMenu.setMnemonic('O');
        mb.add(opMenu);
        JMenu exitMenu = new JMenu("Exit");
        exitMenu.setMnemonic('E');
        mb.add(exitMenu);
        opMenu.add(mAdd = new JMenuItem("Add", 'A'));
        opMenu.add(mSub = new JMenuItem("Sub", 'S'));
        opMenu.add(mMul = new JMenuItem("Mul", 'M'));
        opMenu.add(mDiv = new JMenuItem("Div", 'D'));
        exitMenu.add(mClose = new JMenuItem("Close", 'C'));
        mAdd.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_A,
ActionEvent.CTRL_MASK));
        mSub.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
ActionEvent.CTRL_MASK));
        mMul.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_M,
ActionEvent.CTRL_MASK));
        mDiv.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_D,
ActionEvent.CTRL_MASK));

        JPanel p1 = new JPanel();
        p1.setLayout(new FlowLayout());
        p1.add(new JLabel("Number 1"));
        p1.add(num1 = new JTextField(4));
        p1.add(new JLabel("Number 2"));
        p1.add(num2 = new JTextField(4));
        p1.add(new JLabel("Result"));
        p1.add(res = new JTextField(4));
        res.setEditable(false);

        JPanel p2 = new JPanel();
        p2.setLayout(new FlowLayout());
        p2.add(btAdd = new JButton("Add"));
        p2.add(btSub = new JButton("Sub"));
        p2.add(btMul = new JButton("Mul"));
        p2.add(btDiv = new JButton("Div"));

        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(p1, BorderLayout.CENTER);
        getContentPane().add(p2, BorderLayout.SOUTH);

        btAdd.addActionListener(this);
        btSub.addActionListener(this);
    }
}

```

```

    btMul.addActionListener(this);
    btDiv.addActionListener(this);
    mAdd.addActionListener(this);
    mSub.addActionListener(this);
    mMul.addActionListener(this);
    mDiv.addActionListener(this);
    mClose.addActionListener(this);
}

public void actionPerformed(ActionEvent e){
    String comm = e.getActionCommand();
    if(e.getSource() instanceof JButton){
        if("Add".equals(comm))
            calculate('+');
        else if ("Sub".equals(comm))
            calculate('-');
        else if ("Mul".equals(comm))
            calculate('*');
        else if ("Div".equals(comm))
            calculate('/');
    }
    else if (e.getSource() instanceof JMenuItem){
        if("Add".equals(comm))
            calculate('+');
        else if ("Sub".equals(comm))
            calculate('-');
        else if ("Mul".equals(comm))
            calculate('*');
        else if ("Div".equals(comm))
            calculate('/');
        else if ("Close".equals(comm))
            System.exit(0);
    }
}

private void calculate(char op){
    int n1 = (Integer.parseInt(num1.getText().trim()));
    int n2 = (Integer.parseInt(num2.getText().trim()));
    int r = 0;
    switch( op){
        case '+': r = n1 + n2; break;
        case '-': r = n1 - n2; break;
        case '*': r = n1 * n2; break;
        case '/': r = n1 / n2; break;
    }
    res.setText(String.valueOf(r));
}
}
}

```

/*

The program has 4 buttons labeled Add, Sub, Mul, Div and also create a menu to perform the same operation. The user will also create to perform the same operation. The user can choose the operation either from buttons or from menu selections. Fig. The following contains a sample run of the program.

*/

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class MenuDemo1 extends JFrame implements ActionListener{
    private JTextField num1, num2, res;
    private JButton btAdd, btSub, btMul, btDiv;
    private JMenuItem mAdd, mSub, mMul, mDiv, mClose;
    public static void main(String[] arg){
        MenuDemo1 fr = new MenuDemo1();
        fr.pack();
        fr.setVisible(true);
    }
    public MenuDemo1(){
        setTitle("Menu Demo");
        JMenuBar mb = new JMenuBar();
        setJMenuBar(mb);
        JMenu opMenu = new JMenu("Operation");
        opMenu.setMnemonic('O');
        mb.add(opMenu);
        JMenu exitMenu = new JMenu("Exit");
        exitMenu.setMnemonic('E');
        mb.add(exitMenu);
        opMenu.add(mAdd = new JMenuItem("Add", 'A'));
        opMenu.add(mSub = new JMenuItem("Sub", 'S'));
        opMenu.add(mMul = new JMenuItem("Mul", 'M'));
        opMenu.add(mDiv = new JMenuItem("Div", 'D'));
        exitMenu.add(mClose = new JMenuItem("Close", 'C'));
        mAdd.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_A,
ActionEvent.CTRL_MASK));
        mSub.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
ActionEvent.CTRL_MASK));
        mMul.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_M,
ActionEvent.CTRL_MASK));
```

```

        mDiv.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_D,
ActionEvent.CTRL_MASK));
        JPanel p1 = new JPanel();
        p1.setLayout(new FlowLayout());
        p1.add(new JLabel("Number 1"));
        p1.add(num1 = new JTextField(4));
        p1.add(new JLabel("Number 2"));
        p1.add(num2 = new JTextField(4));
        p1.add(new JLabel("Result"));
        p1.add(res = new JTextField(4));
        res.setEditable(false);
        JPanel p2 = new JPanel();
        p2.setLayout(new FlowLayout());
        p2.add(btAdd = new JButton("Add"));
        p2.add(btSub = new JButton("Sub"));
        p2.add(btMul = new JButton("Mul"));
        p2.add(btDiv = new JButton("Div"));

        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(p1, BorderLayout.CENTER);
        getContentPane().add(p2, BorderLayout.SOUTH);

        btAdd.addActionListener(this);
        btSub.addActionListener(this);
        btMul.addActionListener(this);
        btDiv.addActionListener(this);
        mAdd.addActionListener(this);
        mSub.addActionListener(this);
        mMul.addActionListener(this);
        mDiv.addActionListener(this);
        mClose.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e){
        String comm = e.getActionCommand();
        if(e.getSource() instanceof JButton){
            if("Add".equals(comm))
                calculate('+');
            else if ("Sub".equals(comm))
                calculate('-');
            else if ("Mul".equals(comm))
                calculate('*');

```

```

        else if ("Div".equals(comm))
            calculate('/');
    }
    else if (e.getSource() instanceof JMenuItem){
        if("Add".equals(comm))
            calculate('+');
        else if ("Sub".equals(comm))
            calculate('-');
        else if ("Mul".equals(comm))
            calculate('*');
        else if ("Div".equals(comm))
            calculate('/');
        else if ("Close".equals(comm))
            System.exit(0);
    }
}

private void calculate(char op){

    int n1 = (Integer.parseInt(num1.getText().trim()));
    int n2 = (Integer.parseInt(num2.getText().trim()));
    int r = 0;
    switch( op){
        case '+': r = n1 + n2; break;
        case '-': r = n1 - n2; break;
        case '*': r = n1 * n2; break;
        case '/': r = n1 / n2; break;
    }
    res.setText(String.valueOf(r));

}
}

```

Chương 7 LẬP TRÌNH CƠ SỞ DỮ LIỆU

I. Tổng quan

Một hứa hẹn lớn của java là khả năng xây dựng các ứng dụng CSDL độc lập với hệ nền, công nghệ này được biết đến với cái tên JDBC (Java Database Connectivity), JDBC được thiết kế tương đối đơn giản.

Mô hình JDBC đưa ra một tập các lớp và các phương thức để giao tiếp với ngôn ngữ dữ liệu. JDBC API được tích hợp vào ngay nền java nên bạn không cần phải cài đặt gì thêm khi truy cập CSDL.

II. Các kiểu trình điều khiển JDBC

Các trình điều khiển JDBC có thể phân thành 4 kiểu sau phụ thuộc vào cách thức hoạt động, cách kết nối với CSDL.

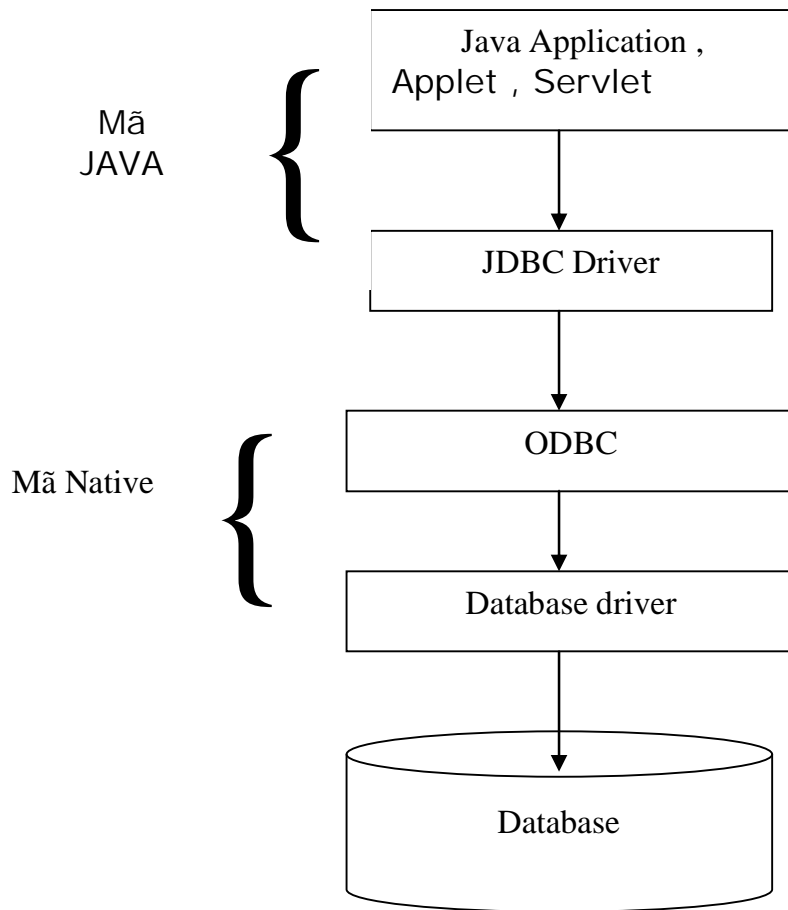
Kiểu 1 : JDBC sử dụng cầu nối ODBC (JDBC-ODBC Bridge)

ODBC là một công nghệ của Microsoft dùng để kết nối đến các hệ cơ sở dữ liệu thông qua trình điều khiển Driver của hệ cơ sở dữ liệu đó. Mỗi hệ cơ sở dữ liệu cung cấp một trình điều khiển (Driver) có khả năng làm việc (đọc, cập nhật,..) được cơ sở dữ liệu của chúng . Trình điều khiển này sẽ được đăng ký với bộ quản lý ODBC . ODBC API là một tập các hàm API để truy cập vào CSDL thông qua các trình điều khiển, người lập trình sẽ sử dụng các hàm API trong giao diện này để truy cập vào CSDL.

Cầu nối JDBC – ODBC là một trình điều khiển sử dụng mã java để gọi lại các hàm trong JDBC API. Bằng cách này bạn có thể kết nối được với nhiều hệ quản trị cơ sở dữ liệu khác nhau.

Để có thể kết nối vào một hệ CSDL cụ thể nào đó (chẳng hạn như: Access, SQL Server, MySQL...) sử dụng cầu nối JDBC-ODBC bạn cần phải có:

- Trình điều khiển (Driver) của dùng để truy cập vào hệ CSDL đó. Thông thường trình điều khiển này được cung cấp bởi chính các hãng sản xuất hệ CSDL và nó thường được cài đặt vào trong máy khi bạn cài đặt hệ quản trị CSDL.
- Cầu nối JDBC-ODBC, đây là trình điều khiển được cung cấp miễn phí bởi hãng Sun và nó được cài đặt tự động khi bạn cài đặt JDK.



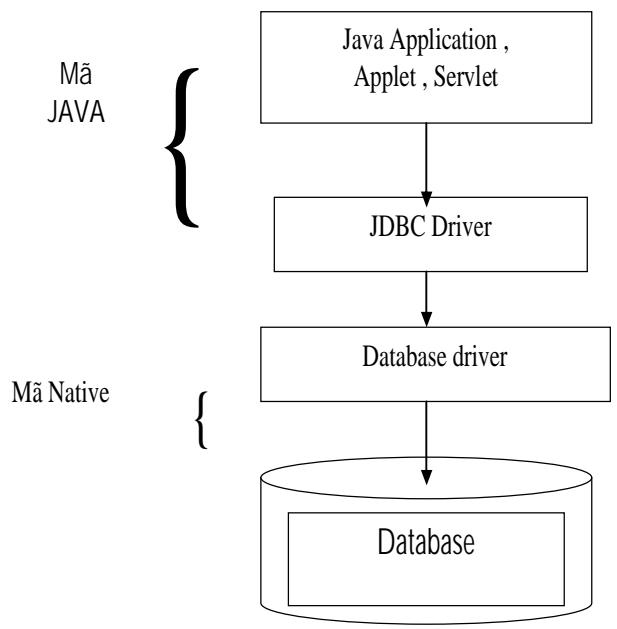
Mô hình truy cập CSDL qua cầu nối JDBC

ó thể

sử dụng nên nó chỉ thích hợp cho các ứng dụng máy đơn, hoặc được chạy trên máy chủ trong mô hình nhiều tầng (Multi - tier), không thích hợp cho các ứng dụng chạy trên máy khách như: Applet, Fat Client.

Kiểu 2 : JDBC kết nối trực tiếp với các trình điều khiển cơ sở dữ liệu .

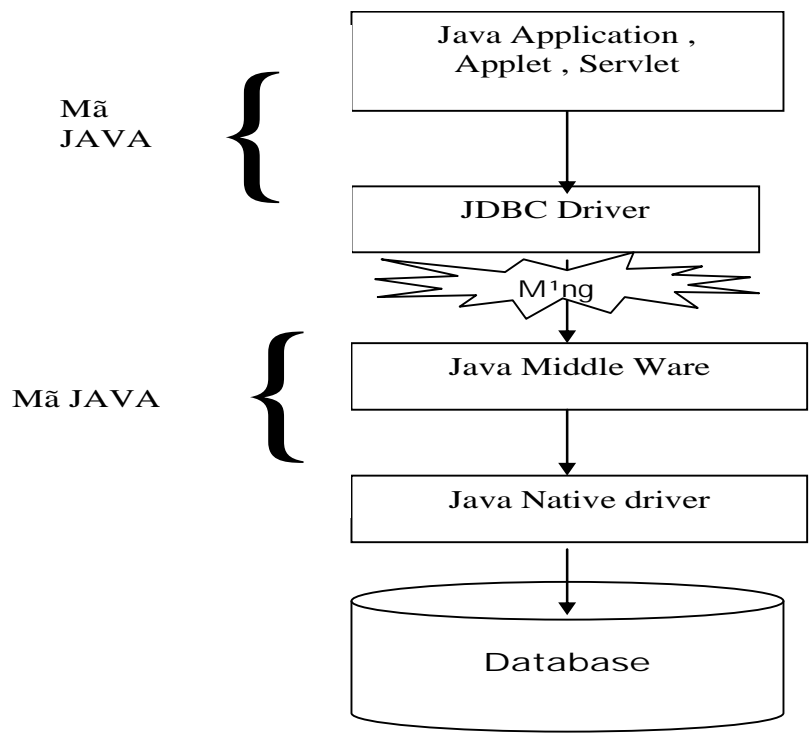
Kiểu kết nối này sử dụng mã java trực tiếp gọi các hàm trong trình điều khiển đặc thù của mỗi hệ cơ sở dữ liệu mà không phải qua trung gian ODBC . Do vậy kiểu kết nối này nhanh và hiệu quả hơn cầu nối JDBC – ODBC, nhưng phải có trình điều khiển cơ sở đặc thù do nhà phát triển cung cấp hoặc của một hãng thứ ba nào đó.



Mô hình kết nối trực tiếp

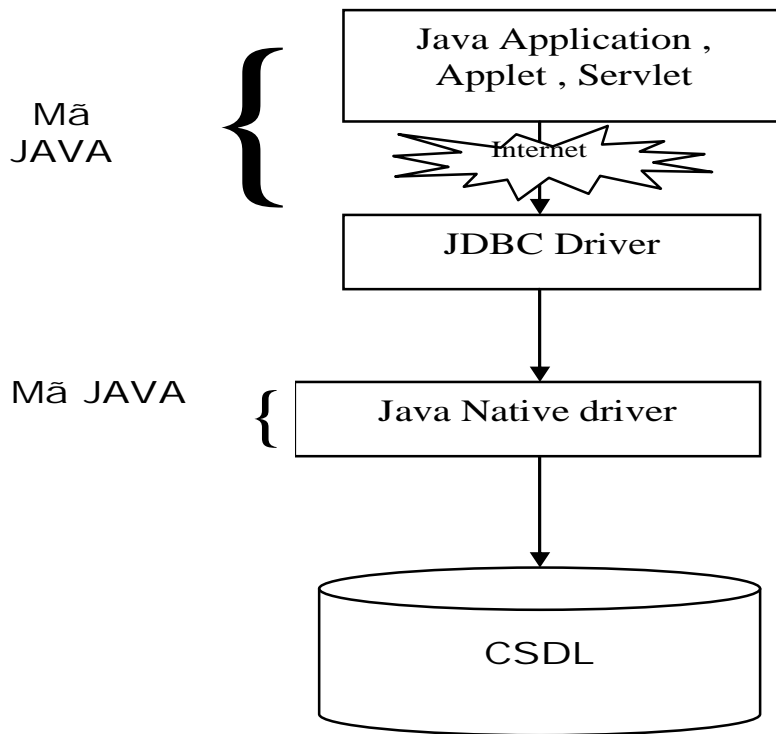
Kiểu 3 :
 Vớ
 Mã
 vớ

tiếp



Kiểu

Kiểu này cho phép máy khách sử dụng trình điều khiển gọn nhẹ nối kết trực tiếp vào trình điều khiển cơ sở dữ liệu đặc thù trên máy chủ ở xa thông qua mạng Internet .



III. Các lớp trong JDBC API dùng để truy cập CSDL

JDBC API bao gồm các lớp và các giao diện được chứa trong hai gói: java.sql và javax.sql. Gói java.sql chứa các giao diện và gói cho phép ta thực hiện các thao tác cơ bản đối với CSDL, gói javax.sql chứa các lớp và giao diện giúp ta thực hiện các tính năng cao cấp.

1. Một số giao diện và lớp trong gói java.sql

a) Một số giao diện

Tên giao diện	Mô tả ý nghĩa
CallableStatement	Giao diện chứa các phương thức cho phép ta làm việc với thủ tục lưu trữ nội
DatabaseMetaData	Cho phép ta xem các thông tin về CSDL
PreparedStatement	Giao diện này cho phép ta thực thi các câu lệnh SQL chứa tham số
ResultSetMetaData	
Connection	Thể hiện một kết nối đến CSDL
Driver	Giao diện mà mỗi trình điều khiển phải cài đặt
ResultSet	Thể hiện một tập các bản ghi lấy về từ CSDL
Statement	Giao diện cho phép ta thực hiện các phát biểu SQL

b) Mét số lớp

Tên lớp	Ý nghĩa
Date	Lớp biểu diễn kiểu DATE
DriverPropertyInfo	Chứa các thuộc tính của trình điều khiển đã nạp
Timestamp	Lớp biểu diễn cho SQL TimeTemp
DriverManager	Lớp quản lý các trình điều khiển
Time	Lớp biểu diễn kiểu DATE
Types	Lớp định nghĩa các hằng tương ứng với các kiểu dữ liệu SQL, hay còn gọi là kiểu dữ liệu JDBC

2. Một số lớp và giao diện trong gói javax.sql

Tham khảo tài liệu về JDBC

IV. Kết nối CSDL với JDBC

Để có thể làm việc với CSDL, một chương trình java (hay bất cứ chương trình viết bằng ngôn ngữ nào khác) phải tuân theo các bước sau:

Mở kết nối đến CSDL.

Thực hiện các lệnh làm việc với CSDL như: truy vấn, cập nhật...

Đóng kết nối, giải phóng tài nguyên.

Các bước để xây dựng một ứng dụng CSDL với JDBC:

1. Chuẩn bị

a) Cài đặt JDBC và trình điều khiển:

Một ứng java chỉ có thể kết nối và làm việc được với dữ liệu trong CSDL thông qua các trình điều khiển và các thư viện JDBC API.

Trước khi truy cập vào hệ CSDL nào bạn phải có trong tay trình điều khiển tương ứng với hệ CSDL đó. Bạn có thể download các trình điều khiển cho một số hệ CSDL quen thuộc từ địa chỉ java.sun.com/jdbc

b) Cài đặt CSDL

Bạn cần lựa chọn và cài đặt một hệ quản trị CSDL, để quản trị phần dữ liệu của chương trình. Đây là lựa chọn của bạn tùy theo tính chất của dữ liệu trong chương trình cũng như mức tiền bạn có để có thể mua được hệ quản trị đó.

Sau khi bạn lựa chọn được hệ quản trị CSDL phù hợp bạn tiến hành cài đặt CSDL: tạo bảng, view, thủ tục lưu trữ..

Chú ý: Để có thể truy cập vào CSDL của hệ quản trị CSDL nào thì bạn phải có trong tay trình điều khiển tương ứng.

2. Thiết lập kết nối

Đây là bước quan trọng nhất bao gồm hai bước:

a) *Nạp trình điều khiển*

JDBC sử dụng một bộ trình quản lý điều khiển (DriverManager), mỗi trình điều khiển (Driver) cho phép ta làm việc với một CSDL cụ thể, cho nên nếu bạn có 3 CSDL do 3 nhà cung cấp khác nhau cung cấp thì bạn cần phải có 3 trình điều khiển khác nhau.

Để nạp và đăng ký trình điều khiển bạn sử dụng lệnh `Class.forName(URL)`

Trong đó URL là một chuỗi mô tả các thông tin về tên của trình điều khiển dùng kết nối với cơ sở dữ liệu, chẳng hạn lệnh

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

dùng để nạp trình điều khiển JDBC-ODBC

Chú ý: Với mỗi trình điều khiển khác nhau thì URL của phương thức `Class.forName()` sẽ khác nhau. Nên để có thể nạp đúng trình điều khiển bạn nên tham khảo tài liệu đi kèm của các trình điều khiển, để biết được URL tương ứng.

b) *Tạo một kết nối*

Bước tiếp theo bạn tạo một kết nối đến CSDL thông qua trình điều khiển mà bạn vừa nạp ở bước trên. Đoạn mã dùng để tạo một kết nối thông thường như sau:

```
Connection con = DriverManager.getConnection(url, "myLogin", "myPassword");
```

Trong đó :

- url là chuỗi chứa thông tin kết nối đến CSDL, nếu bạn sử dụng cầu nối JDBC-ODBC thì url là "jdbc:odbc:DataSourceName", chẳng hạn tôi đã tạo một DNS tên là TestDB với tên truy cập là theht, mật khẩu truy cập là theht thì câu lệnh kết nối đến CSDL là

```
Connection con = DriverManager.getConnection("jdbc:odbc:TestDB","theht", "theht");
```

- myLogin là tên truy cập, nếu mật không có mật khẩu bạn có thể truyền cho nó một chuỗi rỗng

- myPassword là mật khẩu truy cập, nếu mật không có mật khẩu bạn có thể truyền cho nó một chuỗi rỗng

Chú ý: Thông thường chuỗi URL có định dạng như sau:

jdbc:subprotocol:subname

Trong đó:

subprotocol là giao thức con tương ứng với loại CSDL

subname là tên của nguồn dữ liệu (Thông thường nó là tên của CSDL)

Một vài ví dụ:

Ví dụ 1: Nạp trình điều khiển và kết nối đến CSDL của MySQL

Trình điều khiển để truy cập vào CSDL của MySQL, bạn có thể download tại địa chỉ MySQL.com, sau khi download bạn có một tệp tin duy nhất đã được nén theo chuẩn jar, đặt classpath trở đến tệp tin này

- Nạp tình điều khiển của MySQL

```
Class.forName("com.mysql.jdbc.Driver");
```

- **Kết nối đến CSDL**

```
Connection conn = DriverManager.getConnection(  
    "jdbc:mysql://ServerName/DBName?user=UserName&password=Pass");
```

- ServerName: là tên của máy chủ CSDL
- DBName: là tên của CSDL
- UserName: là tên truy cập CSDL
- Pass: là mật khẩu truy cập

Ví dụ 2: Nạp trình điều khiển và kết nối đến CSDL của SQL Server 7.0, 2000

Có rất nhiều trình điều khiển cho MS SQL Server, đa phần các trình điều khiển đều là các sản phẩm thương mại, có một trình điều khiển mà ta nên sử dụng đó là trình điều khiển do chính hãng Microsoft cung cấp, trình điều khiển này hoàn toàn Free và hỗ trợ đầy đủ các tính năng của Sql Server.

- Địa chỉ download www.microsoft.com

- Sau khi download và tiến hành cài đặt bạn sẽ có 3 tệp tin trong thư mục cài đặt:

```
install_dir/lib/msbase.jar
```

```
install_dir/lib/msutil.jar
```

```
install_dir/lib/mssqlserver.jar
```

install_dir: là thư mục cài đặt

- Đặt classpath trở đến 3 tệp tin trên

- **Nạp trình điều khiển**

```
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
```

- **Kết nối đến CSDL**

```
Connection conn = DriverManager.getConnection(  
    "jdbc:microsoft:sqlserver://ServerName:ServerPort;DatabaseName=DBName",  
    "UserName","Password");
```

hoặc

```
Connection conn = DriverManager.getConnection  
    "jdbc:microsoft:sqlserver://ServerName:ServerPort;User=UserName;Password=Pass  
word;DatabaseName=DBName");
```

- + **ServerName:** Là tên của máy chủ SQL
- + **ServerPort:** Số hiệu cổng của SQL, nếu trong quá trình cài đặt mà bạn không đặt lại thì giá trị này là 1433
- + **UserName:** Tài khoản đăng nhập vào SQL
- + **Password:** Mật khẩu tương ứng với tài khoản đăng nhập

- **Chuyển CSDL làm việc**

conn.setCatalog("DBName");

- + **DBName:** là tên CSDL

Ví dụ hoàn chỉnh

```
import java.sql.*;
class Test{
    public static void main(String[] args) {
        try {
            Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
            Connection conn = DriverManager.getConnection(
                "jdbc:microsoft:sqlserver://theht:1433;DatabaseName=AA;user=sa;password=");
            Statement st=conn.createStatement();
            ResultSet rs=st.executeQuery("Select * from t");
            while(rs.next()) System.out.println(rs.getString("a")+ rs.getString("b"));
        } catch (SQLException ex) {}
        } catch (ClassNotFoundException ex) { }
    }
}
```

Ví dụ 3: Nạp trình điều khiển và kết nối đến CSDL của Access với cầu nối JDBC-ODBC của Sun System

Để truy cập vào CSDL của Access ta không cần phải tải xuống và cài đặt vào máy driver nào cả vì nó được tích hợp vào Java, trình điều khiển này do sun cung cấp nó có tên là cầu nối JDBC-ODBC

- **Nạp trình điều khiển**

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

- *Mở kết nối bằng cách sử dụng DataSource Name*

+ Vào control panel chạy chương trình ODBC DataSource 32 bit

+ Tạo một DataSource Name có tên là MyDB

Connection conn = DriverManager.getConnection(" jdbc:odbc:MyDB");

- *Mở kết nối bằng cách sử dụng File DataSource*

Connection conn = DriverManager.getConnection("jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=C:/Path/DatabaseName.mdb");

MyDB: tên của DataSource Name mà bạn đã tạo

Path: đường dẫn đến CSDL

DatabaseName: tên CSDL

Ví dụ hoàn chỉnh

```
import java.sql.*;
import java.util.*;
class Test {
    public static void main(String a[]) {
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Properties props = new Properties(); //Charset để làm việc với Unicode
            props.put("charSet", "UTF8");
            props.put("user", "guest");
            props.put("password", "guest");
            String url =
                "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=C:/MyDB.mdb";
            Connection con = DriverManager.getConnection(url, props);
            Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_UPDATABLE);

            //Tạo bảng
            stmt.executeUpdate("Create Table tbl(a text(50), b text(50))");
            //Chèn 1 bản ghi vào CSDL
            stmt.executeUpdate("Insert Into tbl values('Xin chào','Toi là Access')");
            ResultSet rs = stmt.executeQuery("Select * from tbl");
            if (rs.next()) {
                System.out.println(rs.getString("a") + "\t" + rs.getString("b"));
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Ví dụ 4: Nạp trình điều khiển và kết nối đến CSDL của Oracle 8i, 9i

Trình điều khiển để truy cập vào CSDL của Oracle, bạn có thể tìm thấy nó trong thư mục %ORACLE_HOME%/JDBC/lib

```
Class.forName("oracle.jdbc.OracleDriver ");
```

```
// Lấy về kết nối đến CSDL
```

```
Connection conn = DriverManager.getConnection("jdbc:oracle:oci8:@theht",
"theht","abc");
```

- @theht là tên của máy chủ CSDL
- theht là tên truy cập CSDL
- abc là mật khẩu truy cập

V. Tương tác với CSDL

Tất cả các lệnh mà ta dùng để tác động lên CSDL đều có thể thực hiện thông qua một trong ba đối tượng sau:

Đối tượng	Mô tả
Statement	Dùng để thực thi các lệnh SQL không có tham số
PreparedStatement	Dùng để thực thi các lệnh SQL có chứa tham số
CallableStatement	Dùng để làm việc với thủ tục lưu trữ nội

1. Tạo ra một JDBC Statements

Một đối tượng Statements sẽ giúp bạn gửi một lệnh SQL đến hệ quản trị CSDL, sau khi tạo đối tượng Statements bạn có thể thực hiện các lệnh SQL như SELECT để lấy về dữ liệu hay UPDATE để cập nhật dữ liệu, để tạo đối tượng Statements bạn sử dụng phương thức createStatement của đối tượng connection mà bạn vừa tạo ra trong bước II.

Statement stmt = con.createStatement();

Tại thời điểm này bạn chưa có phát biểu SQL nào để truyền đến CSDL, bây giờ chúng ta sẽ sử dụng phương thức execute của đối tượng này để thực thi một lệnh SQL, chẳng hạn như đoạn lệnh sau sử dụng phương thức executeUpdate để thực thi một câu lệnh truy vấn cập nhật

```
stmt.executeUpdate("CREATE TABLE COFFEES"+"(COF_NAME VARCHAR(32),  
SUP_ID INTEGER, PRICE FLOAT, "  
"SALES INTEGER, TOTAL INTEGER)");
```

Trong java 2 bạn có thể tạo ra các tập kết quả có thể cập nhật và có thể cuộn con trỏ theo cả hai chiều, để tạo ra tập kết quả có thể cập nhật được và cho phép cuộn cả hai chiều bạn sử dụng mẫu sau:

Statement stmt = con.createStatement(ResultSetType, ConcurrencyType);

Trong đó :

Tham số ResultSetType cho biết tập kết quả (ResultSet) nhận được khi thực thi câu lệnh executeQuery() có cuộn được hay không.

Tham số ConcurrencyType tính chất của tập kết quả (ResultSet) nhận được khi thực thi câu lệnh executeQuery() có cho phép cập nhật hay không. Thực ra việc có cập nhật được hay không còn phụ thuộc vào trình điều khiển và hệ quản trị có hỗ trợ cập nhật hay không.

Sau đây là bảng các giá trị mà tham số ResultSetType có thể nhận

Kiểu ResultSet	
TYPE_FORWARD_ONLY	Cho biết ResultSet nhận được chỉ có thể duyệt theo một chiều từ BOF đến EOF
TYPE_SCROLL_INSENSITIVE	Cho phép duyệt theo cả hai chiều nhưng không thấy được sự thay đổi dữ liệu của người dùng khác
TYPE_SCROLL_SENSITIVE	Cho phép duyệt theo cả hai chiều nhưng thấy được sự thay đổi dữ liệu của người dùng khác

Sau đây là bảng các giá trị mà tham số ConcurrencyType có thể nhận

Kiểu ConcurrencyType	
CONCUR_READ_ONLY	Cho biết ResultSet không thể cập nhật được, hay chỉ đọc
CONCUR_UPDATEABLE	Cho biết ResultSet có thể cập nhật được

a) Thực hiện các lệnh DDL (Data Definition Language)

Để thực thi các lệnh DDL bạn sử dụng phương thức `executeUpdate` của đối tượng Statement theo mẫu:

```
stmt.executeUpdate(DDL_SQL_STRING);
```

Trong đó:

stmt là một đối tượng thuộc lớp Statement

DDL_SQL_STRING là một câu lệnh DDL

Chú ý: Phương thức `executeUpdate` trả về giá trị 0 nếu câu lệnh mà nó thực thi là một câu lệnh DDL

b) Thực thi các lệnh cập nhật CSDL

Để thực thi các lệnh cập nhật như: INSERT, UPDATE, DELETE bạn sử dụng phương thức `executeUpdate()` của đối tượng Statement theo mẫu:

```
stmt.executeUpdate(DML_SQL_STRING);
```

Trong đó:

stmt: là một đối tượng thuộc lớp Statement

DML_SQL_STRING là một câu lệnh cập nhật

Chú ý: Phương thức `executeUpdate` trả về một số nguyên thể hiện số bản ghi được ảnh hưởng sau lệnh này, để biết được số bản ghi được ảnh hưởng bạn có thể thực hiện theo mẫu sau:

```
[int recordEffect=] stmt.executeUpdate(DML_SQL_STRING);
```

Ví dụ1: Thêm một bản ghi vào bảng `sinhvien`

```
stmt.executeUpdate("INSERT INTO sinhvien(masv, tensv) values ('01', 'Nguyễn Văn An');
```

Ví dụ2: Xóa các bản ghi của bảng `sinhvien` có `masv=01`

```
stmt.executeUpdate("DELETE sinhvien WHERE masv=01");
```

Ví dụ 3: Sửa lại tên của sinh viên có masv=01 thành Nguyễn Văn B

```
stmt.executeUpdate("UPDATE sinhvien SET tensv='Nguyễn Văn B' WHERE  
masv=01");
```

Cập nhật lại dữ liệu

Để cập nhật lại dữ liệu bạn sử dụng mệnh đề UPDATE SQL, thực thi chúng thông qua phương thức executeUpdate

Ví dụ 4: bạn muốn cộng thêm phụ cấp cho tất cả các người có trình độ đại học lên 10 đồng bạn làm

```
stmt.executeUpdate( "UPDATE COFFEES " + "SET CP=PC+10 WHERE  
td='dh'");
```

Thêm bản ghi vào bảng CSDL

Để chèn dữ liệu vào bảng bạn sử dụng phương thức executeUpdate của đối tượng Statement, ví dụ bạn muốn chèn thêm một bản ghi vào bảng COFFEES bạn làm sau:

```
Statement stmt = con.createStatement();  
stmt.executeUpdate("INSERT INTO COFFEES VALUES ('Colombian', 101, 7.99, 0,  
0)");
```

Xoá bản ghi trong CSDL

Để xoá dữ liệu từ bảng CSDL bạn sử dụng phương thức executeUpdate của đối tượng Statement. Ví dụ bạn muốn tất cả các bản ghi của bảng COFFEES bạn làm sau:

```
Statement stmt = con.createStatement();  
stmt.executeUpdate( "DELETE FROM COFFEES");
```

c) Thực thi câu lệnh truy vấn (Query Language)

Để thực thi câu lệnh truy vấn bạn sử dụng phương thức executeQuery() của đối tượng Statement theo mẫu sau:

```
ResultSet rs=stmt.executeQuery(QUERY_SQL_STRING);
```

Trong đó:

stmt: là một đối tượng thuộc lớp Statement

QUERY_SQL_STRING là một câu lệnh truy vấn dữ liệu

rs là một đối tượng của lớp ResultSet

Chú ý: Phương thức executeQuery trả về một tập kết quả từ CSDL được chứa trong đối tượng ResultSet, nên thông thường bạn sẽ lấy về tham chiếu đến kết tập quả bằng cách:

```
ResultSet rs=stmt.executeQuery(QUERY_SQL_STRING);
```

Ví dụ sau lấy về tất cả các bản ghi của bảng SinhVien và hiển thị ra màn hình

```
ResultSet rs=stmt.executeQuery("SELECT * FROM SinhVien");
While(rs.next())
{
    //.. Xử lý tập kết quả
}
```

d) Xử lý tập kết quả ResultSet

Các phương thức dùng để di chuyển con trỏ

Các phương thức sau dùng để di chuyển con trỏ:

Phương thức	Ý nghĩa
next	Di chuyển con trỏ sang bản ghi kế trong tập bản ghi, phương thức trả về true nếu việc di chuyển là thành công ngược lại cho false
previous	Di chuyển con trỏ về bản ghi trước bản ghi hiện tại, phương thức trả về true nếu việc di chuyển là thành công ngược lại cho false
last	Di chuyển con trỏ về bản ghi cuối cùng trong tập bản ghi, phương thức trả về true nếu việc di chuyển là thành công ngược lại cho false
first	Di chuyển con trỏ về bản ghi đầu tiên trong tập bản ghi, phương thức trả về true nếu việc di chuyển là thành công ngược lại cho false
afterLast	Di chuyển con trỏ về trước bản ghi đầu tiên trong tập bản ghi
beforeFirst	Di chuyển con trỏ về sau bản ghi cuối cùng trong tập bản ghi
absolute(int pos)	Di chuyển con trỏ về bản ghi thứ pos tính từ bản ghi đầu tiên nếu pos là số dương, hoặc di chuyển về bản ghi thứ pos tính từ bản ghi cuối cùng nếu pos là số âm
relative(int pos)	Di chuyển con trỏ về trước bản ghi hiện tại pos bản ghi nếu pos là số âm, hoặc di chuyển về phía sau pos bản ghi so với bản ghi hiện tại nếu pos là số dương

Sử dụng các phương thức getXXX

Ta sử dụng phương thức getXXX để nhận về giá trị hiện tại của một cột tương ứng, tùy theo kiểu dữ liệu của cột mà bạn sử dụng phương thức getXXX tương ứng, ví dụ nếu cột có kiểu là VARCHAR thì bạn sử dụng phương thức getString

Ví dụ: liệt kê tất cả các bản ghi của bảng COFFEES

```
String query = "SELECT COF_NAME, PRICE FROM COFFEES";
ResultSet rs = stmt.executeQuery(query);
while (rs.next())
{
    // sử dụng phương thức getString vì cột //COF_NAME có kiểu VARCHAR
    String s = rs.getString("COF_NAME");
    float n = rs.getFloat("PRICE");
    // Sử dụng phương thức getFloat vì cột PRICE có kiểu float
}
```

```
System.out.println(s + " " + n);
}
```

Chú ý: Bạn có thể sử dụng thứ tự cột khi lấy về dữ liệu thay cho tên cột

Chẳng hạn trong ví dụ trên bạn có thể thay

```
String s = rs.getString("COF_NAME"); bởi
```

```
String s = rs.getString(1);
```

```
Và float n = rs.getFloat("PRICE"); bởi
```

```
float n = rs.getFloat(2)
```

Sau đây là bảng các phương thức tương ứng với kiểu dữ liệu SQL

	T	S	I	B	R	F	D	D	N	B	C	V	L	B	V	L	D	T	T
	I	M	N	I	E	L	O	E	U	I	H	A	O	I	A	O	A	I	I
	N	A	T	G	A	O	U	C	M	T	A	R	N	N	R	N	T	M	M
	Y	L	E	I	L	A	B	I	E		R	C	G	A	B	G	E	E	E
	I	L	G	N		T	L	M	R		H	V	R	I	V				S
	N	I	E	T			E	A	I		A	A	Y	N	A				T
	T	N	R					L	C		R	R		A	R				A
		T											C		R	B			M
													H			Y			P
													A			N			
													R			A			
													R			Y			
getBytes														X	X	x			
getDate											x	x	x				X		x
getString	x	x	x	x	x	x	x	x	x	x	X	X	x	x	x	x	x	x	x
getBoolean	x	x	x	x	x	x	x	x	x	X	x	x	x						
getBigDecimal	x	x	x	x	x	x	x	X	X	x	x	x	x						
getDouble	x	x	x	x	x	X	X	x	x	x	x	x	x						
getFloat	x	x	x	x	X	x	x	x	x	x	x	x	x						
getLong	x	x	x	X	x	x	x	x	x	x	x	x	x						
getInt	x	x	X	x	x	x	x	x	x	x	x	x	x						
getShort	x	X	x	x	x	x	x	x	x	x	x	x	x						
getBytes																			
getDate																			

getTime												x	x	x					X	x
getTimestamp												x	x	x				x	x	X
getAsciiStream												x	x	X	x	x	x			
getUnicodeStream												x	x	X	x	x	x			
getBinaryStream															x	x	X			
getObject	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Trong bảng trên vị trí được đánh “x” trên hàng của phương thức getXXX, thì ta có thể sử dụng phương thức này để nhận về giá trị của cột có kiểu dữ liệu tương ứng, vị trí đánh “X” thì phương thức đó là phương thức sử dụng thích hợp nhất

Cập nhật tập ResultSet

Trong phần trước ta dùng các lệnh SQL- DML để cập nhật trực tiếp vào bảng của CSDL. Trong phần này ta sử dụng các phương thức trong JDBC 2.0 để thực hiện cũng công việc đó.

Điều bạn cần phải chú ý là ta cập nhật dữ liệu của bảng thông qua các ResultSet, thế nên các ResultSet này cần phải là các ResultSet cập nhật được.

Ta có các phương thức updateXXX() khác nhau ứng với các kiểu dữ liệu mà JDBC hỗ trợ. Tương tự như phương thức getXXX() khi ta nhận thông tin từ bảng. Như updateFloat(),updateInt(),updateString(), ...

Sau khi gán giá trị cho các trường bằng các phương thức updateXXX() tương ứng, bạn xác nhận việc cập nhật bằng phương thức updateRow(). Phương thức này hoạt động tương tự như phương thức commit(), phương thức updateRow() có nhiệm vụ ghi lại sự thay đổi đối với hàng hiện tại (đang sửa) vào trong bảng CSDL trên đĩa.

Như vậy khác với các lệnh SQL-DML, cơ chế cập nhật này chỉ có tác dụng đối với bản ghi hiện hành. Thế nên để sửa lại bản ghi nào bạn phải di chuyển con trỏ bản ghi vào bản ghi cần sửa bằng một trong các phương thức đã liệt trong phần trước.

Ta cũng có một phương thức tương tự như phương thức rollback() là phương thức cancelRowUpdate(). Khi gọi phương thức này các thay đổi do gọi phương thức updateXXX() kể từ sau câu lệnh updateXXX() lần trước sẽ bị vô hiệu hoá.

Ví dụ: Cập nhật ResultSet

```
rs.first(); // Nhảy đến bản ghi đầu tiên
rs.updateString("tensv", "Nguyễn Van Hung"); // gán giá trị mới cho cột tensv
rs.updateRow(); // Xác nhận sự thay đổi
rs.last(); // Nhảy đến bản ghi cuối cùng
rs.updateString("tensv", "Nguyễn Van Đạt"); // gán giá trị mới cho cột tensv
rs.cancelRowUpdate(); // Huỷ bỏ việc cập nhật trên
```

Thêm bản ghi mới

Ngoài việc sử dụng các lệnh SQL-DML để thêm các bản ghi trong bảng, ta còn có thể thêm chúng bằng cách thông qua các phương thức của đối tượng ResultSet cập nhật được.

Một ResultSet cập nhật được luôn tồn tại một bản ghi đặc biệt gọi là insertrow để dùng cho việc thêm một bản ghi mới. Bản ghi này không là một thành phần của ResultSet. Thực chất nó chỉ là một vùng đệm để thêm một bản ghi mới.

Các bước để thêm một bản ghi mới vào tập ResultSet

Di chuyển con trỏ đến bản ghi tạm insertrow bằng phương thức **moveToInsertRow()**;
Thực hiện việc điền các giá trị vào các cột tương ứng bằng các phương thức **updateXXX()**;

Sau khi chắc chắn rằng các giá trị thêm vào là đúng. Ta gọi phương thức **insertRow()** để xác nhận việc thêm mới bản ghi vào bảng và ResultSet.

Ví dụ: Thêm bản ghi vào ResultSet

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); // Nạp trình điều khiển
Connection cn=DriverManager.getConnection("jdbc:odbc:t"); // Lấy về kết nối
Statement st=cn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
    ResultSet.CONCUR_UPDATABLE); //Tập ResultSet có thể cập nhật được
ResultSet rs=st.executeQuery("SELECT * FROM sinhvien"); //Lấy về tập ResultSet
rs.moveToCurrentRow(); //Bước 1: Di chuyển con trỏ đến bản ghi tạm để thêm mới
rs.updateString("masv","005"); //Bước 2: Gán giá trị cho các cột bằng phương
    //updateXXX() tương ứng
rs.updateString("tensv","Nguyễn Tự Hào");
rs.updateDate("namsinh",new Date(2004,5,30));
rs.insertRow(); //Bước 3: Xác nhận sự thêm mới
```

Chú ý:

+ Nếu khi gán giá trị cho các cột mà ta bỏ sót một số cột không gán giá trị khi đó các cột đó sẽ nhận các giá trị NULL. Khi đó nếu cột đó có thuộc tính không cho phép nhận giá trị NULL thì bạn sẽ nhận được một ngoại lệ SQLException.

+ Không có phương thức để ta huỷ bỏ việc thêm mới bản ghi. Tuy nhiên để huỷ bỏ việc thêm mới bạn chỉ cần di chuyển con trỏ bản ghi ra khỏi bản ghi tạm bằng một trong các phương thức di chuyển chuyên con trỏ.

Xoá bản ghi

Để xoá một bản ghi bạn có thể sử dụng các lệnh SQL-DML, bạn có thể thực hiện việc này bằng các phương thức của đối tượng ResultSet.

Các bước để xoá một bản ghi khỏi ResultSet

Di chuyển con trỏ đến bản ghi cần xoá

Gọi phương thức deleteRow();

Chú ý: Tuỳ vào trình điều khiển mà việc xoá được thực hiện ngay hay chỉ là đánh dấu xoá. để biết chắc chắn bạn nên tham khảo tài liệu đi kèm với Driver bạn sử dụng.

Ví dụ: Xoá bản ghi

```
rs.first();           // Xoá bản ghi đầu tiên
rs.deleteRow();
rs.absolute(3);      // Xoá bản ghi thứ 3
rs.deleteRow();
rs.last();           // Xoá bản ghi cuối cùng
rs.deleteRow();
```

2. Sử dụng đối tượng Prepared Statements

Khi nào sử dụng đối tượng Prepared Statements?, câu trả lời được phân tích sau:

- Mỗi khi thực thi một lệnh SQL thì DBMS lại phải phân tích lại cú pháp của lệnh, sau đó tối ưu lệnh SQL, công việc này mất rất nhiều thời gian, nếu bạn cần chèn 100 dòng vào CSDL, thế thì bạn phải sử dụng 100 lệnh *executeUpdate để chèn 100 dòng, do vậy phải mất 100 lần* phân tích lệnh, 100 lần tối ưu hoá lệnh, bạn sẽ đặt câu hỏi có các nào mà chỉ cần phân tích lệnh, tối ưu lệnh chỉ một lần mà thôi, vâng cách đó chính là đối tượng Prepared Statements.

- Đôi khi bạn muốn thực hiện một lệnh SQL mà các thông số cụ thể lại chưa biết, trong trường hợp này bắt buộc bạn phải sử dụng truy vấn tham số hoá, tuy nhiên đối tượng Statement lại không hỗ trợ truy vấn tham số hoá

Ví dụ: Sau cập nhật một số dòng của bảng COFFEES sử dụng đối tượng Prepared Statements

```
PreparedStatement updateSales;
String updateString = "update COFFEES set SALES = ? where COF_NAME like ?";
updateSales = con.prepareStatement(updateString);
int [] salesForWeek = {175, 150, 60, 155, 90};
String [] coffees = {"Colombian", "French_Roast", "Espresso",
                    "Colombian_Decaf", "French_Roast_Decaf"};
int len = coffees.length;
for(int i = 0; i < len; i++) {
    updateSales.setInt(1, salesForWeek[i]);
    updateSales.setString(2, coffees[i]);
    updateSales.executeUpdate();
}
```

```
}
```

Như bạn thấy qua ví dụ trên đây:

- + Lấy về đối tượng PreparedStatement bạn sử dụng phương thức ***prepareStatement*** của đối tượng ***Connection*** và truyền vào một câu lệnh SQL chứa tham số
- + Gán giá trị cho các tham số bạn sử dụng phương thức ***setXXX(index, value)***
- + Thực thi truy vấn với các tham số vừa cung cấp bạn gọi phương thức ***executeUpdate()*** của đối tượng ***prepareStatement***.

3. Gọi thủ tục lưu trữ bằng JDBC

JDBC cho phép bạn gọi một thủ tục lưu trữ trong CSDL, từ trong ứng dụng viết bằng java, để có thể gọi một thủ tục lưu trữ trước tiên bạn tạo ra một đối tượng CallableStatement đối tượng này có phương thức prepareCall giúp bạn có thể gọi thủ tục lưu trữ nội

- Các bước:
1. **CallableStatement** cs = con.prepareCall("{call tên_thủ_tục_lưu_trữ}");
 2. Gọi thủ tục **executeXXX()** tương ứng để thực thi thủ tục lưu trữ.

VI. Quản lý giao dịch

Khi một kết nối được tạo, nó được đặt trong trạng thái auto-commit, tức là nó tự động commit sau khi thực thi thành công một lệnh đơn, tuy nhiên bạn có thể đặt lại chế độ auto-commit để bạn có thể quyết định commit hay rollback lại một nhóm lệnh. Như đoạn mã sau:

```
con.setAutoCommit(false);
```

Ví dụ:

```
// Tắt chế độ auto-commit
con.setAutoCommit(false);
// tác động lên CSDL
PreparedStatement updateSales = con.prepareStatement(
    "UPDATE COFFEES SET SALES = ? WHERE COF_NAME LIKE ?");
updateSales.setInt(1, 50);
updateSales.setString(2, "Colombian");
updateSales.executeUpdate();
PreparedStatement updateTotal = con.prepareStatement(
    "UPDATE COFFEES SET TOTAL = TOTAL + ? WHERE COF_NAME LIKE ?");
updateTotal.setInt(1, 50);
updateTotal.setString(2, "Colombian");
updateTotal.executeUpdate();
// Xác nhận commit
con.commit();
// đặt lại chế độ auto-commit
con.setAutoCommit(true);
```

Ví dụ: Lớp truy cập AccessDriver


```

import java.sql.*;
import java.util.*;
import java.io.*;
public class AccessDriver {
    public static void main(String[] args) throws SQLException,IOException{
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //Nap trinh dieu khien Access
        }
        catch (ClassNotFoundException ex) {
            System.out.print("Error: " + ex.getMessage());
        }
        try {
            Properties pro = new Properties();
            pro.put("charSet", "UTF8"); //ma unicode
            String urlldb = "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=c:/data.mdb";
            Connection conn = DriverManager.getConnection(urlldb, pro);
            PreparedStatement pst = conn.prepareStatement("Insert into test(a,b) values (?,?)");
            pst.setString(1, "0021");
            pst.setString(2, "Tran Thi Hong Ha");
            boolean yes = pst.execute(); //PreparedStatement interface thuc thi sql co tham so
            dau vao
            System.out.println(yes);
            Statement st =
            conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
                //st.executeUpdate("Create Table Test (a text(10),b text(255))");
            st.executeUpdate("Insert Into Test(a,b) values ('1234','Nguyen Xuan Truong)");
            // st.executeUpdate("update test set b='dhrhrtr' where a='0021'");
            st.executeUpdate("Delete from test where a='12'");
            String sql = "Select * from test";
            ResultSet rs = st.executeQuery(sql); //lay ve tap ban ghi
            //cap nhat du lieu bang ResultSet
            // rs.moveToCurrentRow();
            rs.moveToInsertRow();
            rs.updateString("a", "rs.update");
            rs.updateString("b", "Cap nhat du lieu bang 455454ResultSet");
            //rs.updateRow();
            rs.insertRow();
            //Xoa du lieu bang ResultSet
            rs.absolute(3); //System.out.println("Rs.absolute(1) " + rs.getString(1));
            rs.deleteRow();
            //Updatable test
            ResultSetMetaData rd = rs.getMetaData();
            if (rs.getConcurrency() == ResultSet.CONCUR_UPDATABLE)
                System.out.println("Rs is Updatable");
            else System.out.println("Rs is ReadOnly");
            int cols = rd.getColumnCount();
            System.out.println("Start----->rd.getColumnLabel");
            for (int i = 1; i <= cols; i++) //return feildName
                System.out.print(rd.getColumnLabel(i) + ( (i == cols) ? "\n" : "\t"));

```

```

System.out.println("End---->rd.getColumnLabel");
while (rs.next()) { //hien thi thong tin, in xuai
    System.out.println(rs.getString("a") + " " + rs.getString("b"));
}
        System.out.println("-----");
rs.last(); //in nguoc lai
while (rs.previous()) {
    System.out.println(rs.getString("a") + " " + rs.getString("b"));
}
rs.close();
conn.close();
}
catch (SQLException ex) {
    System.out.print("Error: maxlenght 10 " + ex.getMessage());
}
}
}
}

```

Ví dụ: Lớp truy cập SQLDriver

```

import java.sql.*;
public class SQLDriver {
    public static void main(String []a){
        try{
            //Nap trinh dieu khien SQLServer
            Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
        }
        catch(ClassNotFoundException ex){
            System.out.println("Can not load driver " +ex.getMessage());
            System.exit(0);
        }
        //jdbc:microsoft:sqlserver://server1:1433;user=test;password=secret
        String
url="jdbc:microsoft:sqlserver://xuantruong:1433;DatabaseName=BOOK;user=sa;password=";
        try{
            Connection conn=DriverManager.getConnection(url);
            Statement st=conn.createStatement();
            ResultSet rs=st.executeQuery("Select * from tblSach");
            while(rs.next()){
                System.out.println(rs.getString(1)+"\t\t"+rs.getString(2));
            }
            rs.close();
            conn.close();
        }
        catch(SQLException ex){
            System.out.println("Can not connection to database " + ex.getMessage());
            System.exit(0);
        }
        System.out.println("OK");
    }
}
}

```

Ví dụ:Lớp truy cập JdbcOdbcDriver

```

package com.borland.samples.welcome;
import java.sql.*;
import javax.sql.*;
public class Connect {
public static void main(String args[]) throws ClassNotFoundException,SQLException {
    System.out.println("Ket noi CSDL");
try{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    String url="jdbc:odbc:myDatabase";
    Connection conn=DriverManager.getConnection(url,"login","password");
    Statement stmt=conn.createStatement();
        //Them record vao bang
    String sql0="INSERT INTO KhachHang(Id,TenKH,DiaChi,Luong) VALUES('8','Hoang
        Anh','HP','900')";

    stmt.executeUpdate(sql0);
        //Cap nhat du lieu (Tang luong cho cac cong nhan)
    String sql2="UPDATE Khachhang SET Luong=Luong+luong*0.1";
    int n=stmt.executeUpdate(sql2);
    if (n < 1) System.out.println("Khong co ban ghi nao duong cap nhat");
    else System.out.println("Co "+ n +" ban ghi nao duong cap nhat");
        //Doc Record ra mh
    String sql="SELECT Id,TenKH,DiaChi,Luong FROM KhachHang";
    ResultSet rs=stmt.executeQuery(sql);
    while (rs.next())
    { int id=rs.getInt("Id");
      double l=rs.getDouble("Luong");
      String s=rs.getString("TenKH");
      String d=rs.getString("DiaChi");
      System.out.println("ID=" +id + " " + s+ " " + d + " Luong=" + l) ;
    }
} catch(SQLException e) {System.out.println("Loi thao tac CSDL");}
}
}

```

Chương 8: LẬP TRÌNH SOCKET

I. Các kiến thức liên quan

Trước khi bắt đầu học bài này các bạn cần đọc lại các kiến thức liên quan sau:

- Giao thức, yêu cầu tối thiểu các bạn phải nắm được 3 giao thức : IP, TCP, UDP
- Cách đánh địa chỉ IP, địa chỉ dạng tên miền, giao thức chuyển đổi địa chỉ IP thành địa chỉ tên miền, và ngược lại
- Một số địa chỉ đặc biệt: địa chỉ lặp, địa chỉ broadcast, multicast...
- Cổng giao thức
- Phân biệt được sự khác nhau, giống nhau giữa 2 giao thức TCP và UDP

1. Giới thiệu Socket

Socket là một giao diện lập trình ứng (API - Application Program Interface) dụng mạng, thông qua giao diện này ta có thể lập trình điều khiển việc truyền thông giữa 2 máy sử dụng các giao thức mức thấp như TCP, UDP..., Socket là một sự trừu tượng hoá ở mức cao, có thể tương tự, nó như là một thiết bị truyền thông 2 chiều tương tự như tệp tin, chúng ta gửi/ nhận dữ liệu giữa 2 máy, tương tự như việc đọc/ ghi trên tệp tin.

Để liên lạc thông qua Socket, ta cần tiến hành các thao tác:

- Tạo lập hay mở một Socket

- + Gắn một Socket với một địa chỉ, địa chỉ này chính là địa chỉ của máy mà nó cần liên lạc
- + Thực hiện việc liên lạc, có 2 kiểu liên lạc tùy thuộc vào chế độ kết nối:

a) Liên lạc trong chế độ không kết nối:

Hai tiến trình liên lạc với nhau không kết nối trực tiếp mỗi thông điệp gửi đi phải kèm theo địa chỉ của người nhận
Hình thức liên lạc này có đặc điểm:
người gửi không chắc chắn thông điệp của họ có đến tay người nhận không
một thông điệp có thể gửi nhiều lần
thông điệp gửi sau có thể đến đích trước thông điệp gửi trước đó

b) Liên lạc trong chế độ kết nối:

Có một đường kết nối “ảo” được thành lập giữa 2 tiến trình, trước khi một kết nối được thành lập thì một trong 2 tiến trình phải đợi tiến trình kia yêu cầu kết nối, có thể sử dụng Socket để liên lạc theo mô hình Client/Server. Trong mô hình này server sử dụng lời gọi *listen* và *accept* để lắng nghe và chấp nhận một yêu cầu kết nối

2. Lập trình Socket trong java

Java cung cấp một số lớp cho phép các ứng dụng mạng có thể trao đổi với nhau qua cơ chế Socket, cụ thể lớp Socket cung cấp cho ta cơ chế liên lạc trong chế độ kết nối (sử dụng giao thức TCP) và lớp DatagramSocket cho phép các ứng dụng mạng liên lạc với nhau trong chế độ không kết nối (sử dụng giao thức UDP), tất cả các lớp liên quan đến việc lập trình Socket được java nhóm lại và để trong gói *java.net*

II. Khảo sát một số lớp trong gói java.net

1. Lớp InetAddress

Mỗi một máy khi tham gia truyền thông cần phải có một định danh, định danh này phải là duy nhất, định danh này được thể hiện bằng địa chỉ IP hoặc địa chỉ dưới dạng

tên miền. Lớp `InetAddress` biểu thị cho một địa chỉ của một máy nào đó, khi ta muốn liên lạc với một máy ở xa, ta phải biết được địa chỉ IP của máy ở xa, tuy nhiên địa chỉ IP thì rất khó nhớ, đôi khi ta không thể biết chính xác địa chỉ IP của máy đó, bởi vì nhiều nguyên nhân khác nhau như: máy đó khởi động từ xa hoặc được nối vào nhà cung cấp dịch vụ Internet, do vậy mỗi lần kết nối vào nhà cung cấp dịch vụ ISP ta lại có 1 địa chỉ IP khác nhau. Vậy thế thì làm thế nào để ta có thể liên lạc với máy ở xa khi ta chỉ biết địa chỉ máy đó dưới dạng tên miền?, câu trả lời là lớp `InetAddress` đã làm điều đó cho ta, lớp này tự động chuyển địa chỉ dạng tên miền thành địa chỉ IP và ngược lại.

Lớp `InetAddress` cung cấp một số phương thức tĩnh (*static*) dùng để chuyển đổi địa chỉ dưới dạng tên miền thành địa chỉ IP và ngược lại. Có một số phương thức sau mà bạn cần quan tâm:

Phương pháp	Mô tả
<code>Public void equals(Object obj)</code>	So sánh 2 đối tượng
<code>Public byte[] getAddress()</code>	Lấy về địa chỉ IP dưới dạng mảng byte
<code>public static InetAddress[] getAllByName(String host)</code> <i>throws UnknownHostException</i>	Trả về mảng đối tượng <code>InetAddress</code> , vì một máy có thể có nhiều địa chỉ IP (do có nhiều card mạng), nên phương thức này trả về một mảng chứa tất cả các địa chỉ tương ứng với tên miền.
<code>public static InetAddress getByName(String host)</code> <i>throws UnknownHostException</i>	Trả lại đối tượng <code>InetAddress</code> có tên được chỉ ra, tên này là một chuỗi ký tự dưới dạng tên miền hoặc địa chỉ IP
<code>public String getHostAddress()</code>	Trả về địa chỉ IP của máy chủ
<code>public String getHostName()</code>	Trả về tên của máy chủ
<code>public static InetAddress getLocalHost()</code> <i>throws UnknownHostException</i>	Trả về đối tượng <code>InetAddress</code> kết hợp với chính máy đó
<code>public boolean isMulticastAddress()</code>	Kiểm tra xem địa chỉ này có phải là địa chỉ Multicast không

Chú ý:

Trong gói `java.net` còn lớp `Inet4Address` và lớp `Inet6Address` hai lớp này thể hiện cho các địa chỉ IP version 4 và IP version 6, nó gồm tất cả các thành phần của lớp `InetAddress`

Ta cần thêm mệnh đề `import` lớp `java.net.InetAddress` trước khi có thể sử dụng nó.

Phương thức *getByName* sẽ cố gắng phân giải tên miền thành địa chỉ IP tương ứng bằng cách: Trước tiên nó đi tìm trong cache, nếu không tìm thấy nó tìm tiếp trong tệp host, nếu vẫn không tìm thấy nó sẽ cố gắng kết nối đến máy chủ DNS để yêu cầu phân giải tên này thành địa chỉ IP, nếu không thể phân giải được tên này thì nó sẽ sinh ra một ngoại lệ *UnknownHostException*, thế nên bạn cần đặt chúng vào một khối try **..catch**.

Ví dụ 1: Minh họa cách sử dụng các phương thức *getByName* để tạo ra một *InetAddress*

```
import java.net.*;
class InetAddress1 {
    public static void main(String[] args) {
        try {
            InetAddress address = InetAddress.getByName("www.theht.edu.vn");
            System.out.println(address);
        }
        catch (UnknownHostException ex) {
            System.out.println("Could not find www.theht.edu.vn");
        }
    }
}
```

Kết quả chạy chương trình như sau:

www.theht.edu.vn/127.0.0.1 Ngoài cách truyền vào phương thức *getByName* một chuỗi ký tự thể hiện tên máy bạn ta thể truyền vào một chuỗi thể hiện địa chỉ IP của máy như sau:

```
InetAddress address = InetAddress.getByName("192.168.101.1");
```

Ví dụ 2: Tạo ra một *InetAddress* tương ứng với máy cục bộ:

```
import java.net.*;
class MyAddress {
    public static void main(String[] args) {
        try {
            InetAddress address = InetAddress.getLocalHost();
            System.out.println(address);
        }
        catch (UnknownHostException ex) {
            System.out.println("Could not find this computer's address.");
        }
    }
}
```

Kết quả chạy chương trình như sau:

theht1/192.168.101.1 Ví dụ 3: Nếu máy bạn có cài nhiều card mạng bạn có thể lấy về một mảng các InetAddress tương ứng với địa chỉ IP cho từng card mạng đó:

```
import java.net.*;
class AllAddressesOfTheht {
    public static void main(String[] args) {
        try {
            InetAddress[] addresses =
                InetAddress.getAllByName("www.theht.edu.vn");
            for (int i = 0; i < addresses.length; i++) {
                System.out.println(addresses[i]);
            }
        }
        catch (UnknownHostException ex) {
            System.out.println("Could not find www.theht.edu.vn");
        }
    }
}
```

Kết quả chạy chương trình như sau:

www.theht.edu.vn /192.168.101.1 www.theht.edu.vn /10.0.0.2 www.theht.edu.vn /162.163.10.5 www.theht.edu.vn /3.152.90.25
sở dĩ cho kết quả như trên là do máy của tôi có cài 4 card mạng, tương ứng với mỗi card mạng tôi đặt một địa chỉ IP cho nó.

Nếu máy của bạn có nối mạng Internet bạn có thể kiểm tra xem máy chủ www.microsoft.com của Microsoft được cài bao nhiêu bộ giao tiếp mạng bằng cách thay `InetAddress.getAllByName("www.theht.edu.vn");` bởi

```
InetAddress.getAllByName("www.microsoft.com");
```

Ví dụ 4: Lấy về tên máy khi biết địa chỉ IP của nó, để lấy về tên của máy ta sử dụng phương thức `getHostName` như sau:

```
import java.net.*;
class ReverseTest {
    public static void main(String[] args) {
        try {
            InetAddress ia = InetAddress.getByName("162.163.10.5");
            System.out.println(ia.getHostName());
        } catch (Exception ex) {
            System.err.println(ex);
        }
    }
}
```

Kết quả ra như sau:

theht

Ví dụ 5: Lấy về địa chỉ IP của máy khi biết tên của máy, để lấy về địa chỉ IP của máy ta sử dụng phương thức *getHostAddress* như sau:

```
import java.net.*;
class GetHostAddress {
    public static void main(String[] args) {
        try {
            InetAddress me = InetAddress.getLocalHost();
            String dottedQuad = me.getHostAddress();
            System.out.println("My address is " + dottedQuad);
        }
        catch (UnknownHostException ex) {
            System.out.println("I'm sorry. I don't know my own address.");
        }
    }
}
```

Kết quả in ra như sau:

My address is 192.168.101.1 Ta có thể lấy về địa chỉ IP tương ứng với một tên miền bất kỳ không nhất thiết là máy cục bộ như trên, chẳng hạn bạn có thể lấy về địa chỉ IP của máy www.theht.edu.vn hoặc www.microsoft.com như sau:

```
import java.net.*;
class GetHostAddress1 {
    public static void main(String[] args) {
        try {
            InetAddress me = InetAddress.getByName("www.theht.edu.vn");
            String dottedQuad = me.getHostAddress();
            System.out.println("Address is " + dottedQuad);
        }
        catch (UnknownHostException ex) {
            System.out.println("I'm sorry. I don't know my own address.");
        }
    }
}
```

Kết quả in ra như sau:

Address is 192.168.101.1

Ví dụ 6: Kiểm tra xem hai địa chỉ tên miền có cùng một địa chỉ IP hay không. để kiểm tra điều này ta sử dụng phương thức *equals* như sau:

```
import java.net.*;
class Equal {
    public static void main(String args[]) {
        try {
            InetAddress add1 = InetAddress.getByName("www.theht.edu.vn");
            InetAddress add2 = InetAddress.getByName("www.theht.com.vn");
            if (add1.equals(add2)) {
                System.out.println("Hai địa chỉ này có cùng IP");
            }
            else {
                System.out.println("Hai địa chỉ này khác IP");
            }
        }
        catch (UnknownHostException ex) {
            System.out.println("Không thể tìm thấy host.");
        }
    }
}
```

Kết quả cho như sau:

Hai địa chỉ này khác nhau

Ví dụ 7: Xây dựng chương trình HostLookup tương tự như chương trình NSLookup của Windows, chương trình này có nhiệm vụ khi bạn gõ vào địa chỉ IP thì nó sẽ trả về địa chỉ tên miền và ngược lại:

```
import java.net.*;
import java.io.*;
public class HostLookup {
    public static void main(String[] args) {
        if (args.length > 0) {
            // Sử dụng tham số dòng lệnh
            for (int i = 0; i < args.length; i++) {
                System.out.println(lookup(args[i]));
            }
        }
        else {
            BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
```

```

System.out.println(
    "Enter names and IP addresses.Enter \"exit\" or \"quit\" to quit.");
try {
    while (true) {
        String host = in.readLine();
        if (host.equalsIgnoreCase("exit") ||
            host.equalsIgnoreCase("quit")) {
            break;
        }
        System.out.println(lookup(host));
    }
}
catch (IOException ex) {
    System.err.println(ex);
}
}

private static String lookup(String host) {
    InetAddress node;
    try {
        node = InetAddress.getByName(host);
    }
    catch (UnknownHostException ex) {
        return "Cannot find host " + host;
    }
    if (isHostname(host)) {
        //Dia chi nay duoi dang ten mien
        return node.getHostAddress();
    }
    else {
        // Dia chi nay duoi dang IP
        return node.getHostName();
    }
}

//Hàm kiểm tra xem địa chỉ host dưới dạng tên miền hay địa chỉ IP
private static boolean isHostname(String host) {
    char[] ca = host.toCharArray();
    for (int i = 0; i < ca.length; i++) {

```

```

    if (!Character.isDigit(ca[i])) {
        if (ca[i] != '.')return true;
    }
}
return false;
}
}

```

2. Lớp URL và URI

Lớp URL là một lớp rất đơn giản giúp bạn trong việc định vị và lấy về dữ liệu từ mạng, mà bạn không cần phải quan tâm đến giao thức được sử dụng, định dạng của dữ liệu hoặc không cần quan tâm đến cách giao tiếp với máy chủ.

Tạo ra một URL

Không giống như đối tượng InetAddress bạn có thể sử dụng hàm tạo của lớp URL để tạo ra một đối tượng URL mới. Có sáu hàm tạo khác nhau của lớp URL cho phép bạn tạo ra các URL với các yêu cầu khác nhau. Tất cả các hàm tạo này đều ném ra ngoại lệ

MalformedURLException nếu bạn tạo ra một URL từ một giao thức không được hỗ trợ hoặc các thông tin cung cấp trong URL không chính xác thì bạn sẽ nhận được một ngoại lệ **MalformedURLException**

a) Tạo ra một URL từ một chuỗi

public URL(String url) **throws** MalformedURLException

Giống như các hàm tạo khác bạn chỉ cần dùng toán tử **new** và cũng giống các hàm tạo khác của lớp nó ném ra ngoại lệ **MalformedURLException**.

Ví dụ 1: Tạo ra một URL từ một chuỗi và bắt ngoại lệ sinh ra

```

try {
    URL u = new URL("http://www.utehy.edu.vn/");
}
catch (MalformedURLException ex) {
    System.err.println(ex);
}

```

Ví dụ 2: Ví dụ này tạo ra một số URL và kiểm tra xem giao thức tương ứng với các URL có được hỗ trợ trong virtual machine của bạn hay không

/ Which protocols does a virtual machine support? */*

```

import java.net.*;
class ProtocolTester {
    public static void main(String[] args) {
        testProtocol("http://www.adc.org");           // hypertext transfer protocol
        testProtocol("https://www.amazon.com/exec/obidos/order2/"); // secure http
                                                // file transfer protocol
        testProtocol("ftp://metalab.unc.edu/pub/languages/java/javafaq");
        testProtocol("mailto:elharo@metalab.unc.edu"); // Simple Mail Transfer Protocol
        testProtocol("telnet://dibner.poly.edu/");     // telnet
        testProtocol("file:///etc/passwd");           // local file access
        testProtocol("gopher://gopher.anc.org.za/");  // gopher
        testProtocol(                                // Lightweight Directory Access Protocol

```

```

"ldap://ldap.itd.umich.edu/o=University%20of%20Michigan,c=US?postalAddress");
testProtocol(                                     // JAR
"jar:http://cafeaulait.org/books/javaio/ioexamples/javaio.jar!"
+ "/com/macfaq/io/StreamCopier.class");
testProtocol("nfs://utopia.poly.edu/usr/tmp/");    // NFS, Network File System
testProtocol("jdbc:mysql://luna.metalab.unc.edu:3306/NEWS");
                                                // a custom protocol for JDBC

// rmi, a custom protocol for remote method invocation
testProtocol("rmi://metalab.unc.edu/RenderEngine");
testProtocol("doc:/UsersGuide/release.html");     // custom protocols for HotJava
testProtocol("netdoc:/UsersGuide/release.html");
testProtocol("systemresource://www.adc.org/+/index.html");
testProtocol("verbatim:http://www.adc.org/");
}
private static void testProtocol(String url) {
try {
URL u = new URL(url);
System.out.println(u.getProtocol() + " is supported");
}
catch (MalformedURLException ex) {
String protocol = url.substring(0, url.indexOf(':'));
System.out.println(protocol + " is not supported");
}
}
}
}
}

```

Kết quả chạy trên máy của tôi như sau:

```

http is supported
https is supported
ftp is supported
mailto is supported

```

b) Tạo ra một URL từ các phần riêng lẻ

Hàm tạo này cho gồm ba phần riêng biệt, chúng cho biết: Giao thức, tên máy chủ, tệp tin mà URL này sử dụng.

```

public URL(String protocol, String hostname, String file)
throws MalformedURLException

```

Hàm tạo này sẽ tự động đặt port là giá trị mặc định, tùy theo giao thức mà bạn chỉ định nó sẽ sử dụng mà giá trị mặc định này là khác nhau: Ví dụ như : http thì port=80, ftp=23...

Ví dụ:

```

try {
URL u = new URL("http", "www.microsoft.com", "/index.html");
}
catch (MalformedURLException ex) {
// All VMs should recognize http
}
}

```

Đoạn chương trình trên tạo ra một URL chỉ đến ***http://www.microsoft.com/index.html***, cổng mặc định là 80

Để tạo ra một URL với một cổng chỉ định bạn sử dụng hàm tạo sau:
**public URL(String protocol, String host, int port, String file) throws
MalformedURLException**

Ví dụ: Tạo ra một URL gắn với máy localhost trên cổng 8080

```
try {  
    URL u = new URL("http", "localhost", 8080, "/index.html");  
}  
catch (MalformedURLException ex) {  
    System.err.println(ex);  
}
```

Ví dụ này tạo ra một URL trỏ đến **http://localhost: 8080/index.html**

c) Tạo ra một URL từ một URL khác

Hàm tạo này cho phép bạn tạo ra một URL từ một URL khác, cú pháp như sau:

public URL(URL base, String relative) throws MalformedURLException

Ví dụ:

```
try {  
    URL u1 = new URL("http://www.ibiblio.org/javafaq/index.html");  
    URL u2 = new URL(u1, "mailinglists.html");  
}  
catch (MalformedURLException ex) {  
    System.err.println(ex);  
}
```

Trong ví dụ này thì u1 đang trỏ đến <http://www.ibiblio.org/javafaq/index.html> còn u2 thì đang trỏ đến <http://www.ibiblio.org/javafaq/ mailinglists.html>.

Một vài phương thức của lớp URL

- **public String getHost()** trả về tên của máy chủ
- **public int getPort()** trả về cổng được sử dụng
- **public int getDefaultPort()** trả về cổng mặc định của giao thức được sử dụng. Ví dụ như :
http=80, email=25, ftp=23...

- **public String getFile()** trả về tên tệp tin

- **public String getPath()** // Java 1.3 trả về đường dẫn đến file, đường dẫn này thường là một query string.

- **public String getRef()**, trong ví dụ sau thì getRef sẽ trả về xtocid1902914

```
URL u = new URL(  
    "http://www.ibiblio.org/javafaq/javafaq.html#xtocid1902914");
```

```
System.out.println("The fragment ID of " + u + " is " + u.getRef( ));
```

- **public String getQuery()** // Java 1.3 trả về một query string trong URL

Ví dụ: Trong ví dụ này thì getQuery sẽ trả về category=Piano

```
URL u = new URL(  
    "http://www.ibiblio.org/nywc/compositions.phtml?category=Piano");
```

```
System.out.println("The query string of " + u + " is " + u.getQuery( ));
```

- **public String getUserInfo()** // Java 1.3 trả về user name

- **public String getAuthority()** // Java 1.3

Ví dụ sau minh họa các phương thức trên

```
import java.net.*;
class URLSplitter {
    public static void main(String args[]) {
        try {
            URL u = new URL("http://java.sun.com");
            System.out.println("The URL is " + u);
            System.out.println("The scheme is " + u.getProtocol());
            System.out.println("The user info is " + u.getUserInfo());
            String host = u.getHost();
            if (host != null) {
                int atSign = host.indexOf('@');
                if (atSign != -1) host = host.substring(atSign + 1);
                System.out.println("The host is " + host);
            }
            else {
                System.out.println("The host is null.");
            }
            System.out.println("The port is " + u.getPort());
            System.out.println("The path is " + u.getPath());
            System.out.println("The ref is " + u.getRef());
            System.out.println("The query string is " + u.getQuery());
        } // end try
        catch (MalformedURLException ex) {
            System.err.println("is not a URL I understand.");
        }
        System.out.println();
    } // end main
} // end URLSplitter
```

Kết quả chạy chương trình như sau:

The URL is http://java.sun.comThe scheme is httpThe user info is nullThe host is java.sun.comThe port is -1The path is The ref is nullThe query string is null)
Nhận về dữ liệu từ URL

Bạn có thể nhận về dữ liệu được chỉ định trong URL bằng cách sử dụng các phương thức sau:

```
public InputStream openStream( ) throws IOException
public URLConnection openConnection( ) throws IOException
public URLConnection openConnection(Proxy proxy) throws IOException // 1.5
public Object getContent( ) throws IOException
public Object getContent(Class[] classes) throws IOException // 1.3
public final InputStream openStream( ) throws IOException
```

Phương thức này sẽ kết nối đến máy chủ và yêu cầu lấy về một InputStream gắn với tài nguyên được chỉ định trong URL.

Ví dụ sau lấy về nội dung của trang web được chỉ định và in ra màn hình

```
import java.net.*;
import java.io.*;
class SourceViewer {
    public static void main(String[] args) {
        try {
            //Open the URL for reading
            URL u = new URL("http://localhost:80/index.html");
            InputStream in = u.openStream(); // buffer the input to increase performance
            in = new BufferedInputStream(in); // chain the InputStream to a Reader
            Reader r = new InputStreamReader(in);
            int c;
            while ( (c = r.read()) != -1) {
                System.out.print( (char) c);
            }
        }
        catch (MalformedURLException ex) {
            System.err.println("is not a parseable URL");
        }
        catch (IOException ex) {
            System.err.println(ex);
        }
    } // end main
} // end SourceViewer
```

Kết quả chạy như sau:

```
<html><head><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><title> Web Posting Information </title></head><body><p><font face=".VnTime">Xin chào các bạn</font></p></body></html>
```

- **Phương thức** `public URLConnection openConnection() throws IOException`

Phương thức này mở một socket kết nối đến máy chủ. Nếu phương thức này thất bại nó ném ra ngoại lệ `IOException`.

Ví dụ:

```
try {
    URL u = new URL("http://www.jennicam.org/");
    try {
        URLConnection uc = u.openConnection( );
        InputStream in = uc.getInputStream( );
        // read from the connection...
    } // end try
    catch (IOException ex) {
        System.err.println(ex);
    }
} // end try
catch (MalformedURLException ex) {
    System.err.println(ex);
}
```

Trong java 1.5 còn một phương thức được nạp chồng như sau:

public URLConnection openConnection(Proxy proxy) **throws** IOException

Phương thức này rất hữu dụng khi trong mạng của bạn có sử dụng proxy

- **Phương thức public final** Object getContent() **throws** IOException

Ví dụ:

```
import java.net.*;
import java.io.*;
class ContentGetter {
    public static void main(String[] args) {
        //Open the URL for reading
        try {
            URL u = new URL("http://localhost:80/index.html");
            try {
                Object o = u.getContent();
                System.out.println("I got a " + o.getClass().getName());
            } // end try
            catch (IOException ex) {
                System.err.println(ex);
            }
        } // end try
        catch (MalformedURLException ex) {
            System.err.println("is not a parseable URL");
        }
    } // end main
} // end ContentGetter
```

Kết quả chạy như sau:

I got a sun.net.www.protocol.http.HttpURLConnection\$HttpInputStream Nếu thay URL

u = new URL("http://localhost:80/index.html"); bởi URL u = new

URL("http://localhost:80/print.gif"); kết quả thu được như sau:

I got a sun.awt.image.URLImageSource

3. Lớp Socket

Lớp này được dùng cho cả máy chủ và máy khách. Đối với máy khách nó thường dùng để kết nối đến máy chủ, còn đối với máy chủ nó thường dùng để đáp lại kết nối từ máy khách. Thông thường ta tạo ra một Socket bằng cách sử dụng hàm tạo của lớp Socket. Sau đây là một số hàm tạo của lớp Socket.

3.1 Một số hàm tạo của lớp Socket

a) public Socket(String host, int port) throws UnknownHostException, IOException

Ta thường sử dụng hàm tạo này để kết nối đến máy chủ. Trong hàm tạo này tên của máy là một chuỗi ký tự, tên của máy chủ có thể là tên miền hoặc địa chỉ IP. Nếu như không tồn tại máy này thì nó sẽ ném ra một ngoại lệ **UnknownHostException**. Nếu Socket không thể mở vì bất cứ lý do nào thì nó ném ra ngoại lệ **IOException**.

Ví dụ: Kết nối đến web server `www.theht.edu.vn`

```
try {
    Socket toTheht= new Socket("www.theht.edu.vn", 80);
    //~ Socket toTheht= new Socket("162.163.10.5", 80);
    // send and receive data...
}
catch (UnknownHostException ex) {
    System.err.println(ex);
}
catch (IOException ex) {
    System.err.println(ex);
}
```

- **public** Socket(InetAddress host, int port) **throws** IOException

Ta thường sử dụng hàm tạo này để kết nối đến máy chủ khi biết một *InetAddress* gắn với nó.

Ví dụ sau kết cũng kết nối đến máy chủ *www.theht.edu.vn* trên cổng 80

```
try {
    InetAddress theht = InetAddress.getByName("www.theht.edu.vn");
    //~ InetAddress theht = InetAddress.getByName("162.135.10.5");
    Socket thehtSocket = new Socket(theht , 80);
    // send and receive data...
}
catch (UnknownHostException ex) {
    System.err.println(ex);
}
catch (IOException ex) {
    System.err.println(ex);
}
```

- **public** Socket(String host, int port, InetAddress interface, int localPort) **throws** IOException, UnknownHostException

Nếu máy của bạn có nhiều bộ giao tiếp mạng thì khi kết nối đến máy chủ ta cần chỉ ra kết nối được thiết lập thông qua giao diện mạng nào.

Nếu tham số port nhận giá trị 0 thì java runtime sẽ chọn ngẫu nhiên một cổng nào đó chưa được sử dụng trong khoảng 1024 à 65535.

Ví dụ:

```
try {
    InetAddress inward = InetAddress.getByName("theht");
    Socket socket = new Socket("www.theht.edu.vn", 80, inward, 0);
    // work with the sockets...
}
catch (UnknownHostException ex) { System.err.println(ex);
}
catch (IOException ex) {
    System.err.println(ex);
}
```

- **public** Socket(InetAddress host, int port, InetAddress interface, int localPort) **throws** IOException

Hàm tạo này cũng tương tự như hàm tạo trên.

Ví dụ:

```
try {
    InetAddress inward = InetAddress.getByName("theht");
    InetAddress http= InetAddress.getByName("www.theht.edu.vn");
    Socket socket = new Socket(http, 80, inward, 0);
        // work with the sockets...
}
catch (UnknownHostException ex) {
    System.err.println(ex);
}
catch (IOException ex) {
    System.err.println(ex);
}
```

3.2. Lấy về thông tin gắn với Socket

- Phương thức **public** InetAddress getInetAddress() dùng để lấy về đối tượng InetAddress tương ứng với máy remote.

```
try {
    Socket theSocket = new Socket("java.sun.com", 80);
    InetAddress host = theSocket.getInetAddress( );
    System.out.println("Connected to remote host " + host);
} // end try
catch (UnknownHostException ex) {
    System.err.println(ex);
}
catch (IOException ex) {
    System.err.println(ex);
}
```

- Phương thức **public int** getPort() dùng để lấy về cổng của máy remote.

Ví dụ:

```
try {
    Socket theSocket = new Socket("java.sun.com", 80);
    int port = theSocket.getPort( );
    System.out.println("Connected on remote port " + port);
} // end try
catch (UnknownHostException ex) {
    System.err.println(ex);
}
catch (IOException ex) {
    System.err.println(ex);
}
```

- Phương thức **public int getLocalPort()** dùng để lấy về cổng của máy cục bộ dùng để giao tiếp với máy ở xa.

Ví dụ:

```
try {
    Socket theSocket = new Socket("java.sun.com", 80);
    int localPort = theSocket.getLocalPort( );
    System.out.println("Connecting from local port " + localPort);
} // end try
catch (UnknownHostException ex) {
    System.err.println(ex);
}
catch (IOException ex) {
    System.err.println(ex);
}
```

- Phương thức **public InetAddress getLocalAddress()** dùng để lấy về đối tượng `InetAddress` gắn với card mạng dùng để giao tiếp.

Ví dụ:

```
try {
    Socket theSocket = new Socket(hostname, 80);
    InetAddress localAddress = theSocket.getLocalAddress( );
    System.out.println("Connecting from local address " + localAddress);
} // end try
catch (UnknownHostException ex) {
    System.err.println(ex);
}
catch (IOException ex) {
    System.err.println(ex);
}
```

Ví dụ sau in ra màn hình một số thông tin gắn với `Socket`

```
import java.net.*;
import java.io.*;

public class SocketInfo {
    public static void main(String[] args) {
        try {
            Socket theSocket = new Socket("www.theht.edu.vn", 80);
            System.out.println("Connected to " + theSocket.getInetAddress()
                + " on port " + theSocket.getPort() + " from port "
                + theSocket.getLocalPort() + " of "
                + theSocket.getLocalAddress());
        } // end try
        catch (UnknownHostException ex) {
            System.err.println("I can't find host");
        }
    }
}
```

```

catch (SocketException ex) {
    System.err.println("Could not connect to host");
}
catch (IOException ex) {
    System.err.println(ex);
}
} // end main
} // end SocketInfo

```

- Phương thức **public** `OutputStream getOutputStream()` **throws** `IOException` dùng để lấy về một `OutputStream`. Việc gửi dữ liệu đến máy remote được thông qua `OutputStream` này.

- Phương thức **public** `InputStream getInputStream()` **throws** `IOException` phương thức này dùng để nhận về một `InputStream`. Việc nhận dữ liệu từ máy remote được thực hiện thông qua `InputStream` này.

3.3. Đóng Socket

Khi kết nối không còn được sử dụng nữa bạn cần gọi phương thức `close()` một cách tường minh để đóng Socket lại. Khi gọi đến phương thức `close` java sẽ tự động đóng hai luồng nhập xuất gắn với nó. Việc đóng Socket lại sẽ giải phóng một số tài nguyên của hệ thống.

public void `close()` **throws** `IOException`

Một Socket sẽ tự động đóng lại nếu xảy ra một trong các tính huống sau:

- Cả hai luồng nhập xuất gắn với Socket đều được đóng lại.
- Khi chương trình kết thúc.
- Hoặc khi bộ thu rác của java được tiến hành.

Tuy các Socket cũng tự động đóng lại khi cần thiết, nhưng một thói quen thực hành tốt là nên gọi đến phương thức `close()` khi không còn dùng đến nó nữa.

Để đảm bảo Socket luôn được đóng lại cho dù có ngoại lệ xảy ra hay không. thì chỗ hợp lý nhất để đóng Socket lại là trong khối `finally`.

Ví dụ sau thể hiện điều đó:

```

Socket connection = null;
try {
    connection = new Socket("www.theht.edu.vn", 80);
    // interact with the socket...
} // end try
catch (UnknownHostException ex) {
    System.err.println(ex);
}
catch (IOException ex) {
    System.err.println(ex);
}
finally {
    // Đóng Socket lại
    if (connection != null) connection.close( );
}

```

- Phương thức **public boolean isClosed()** (chỉ có trong java 1.4 trở lên) phương thức này trả về true nếu như Socket đã được đóng lại, nhận về false trong trường hợp ngược lại. Thế nên để đảm bảo các thao tác nhập xuất không xảy ra lỗi bạn nên sử dụng mẫu chương trình sau:

```
if (socket.isClosed() ) {
    // do something...
}
else {
    // do something else...
}
```

Tuy nhiên nếu Socket chưa bao giờ được mở thì phương thức này trả về false.

- Phương thức **public boolean isConnected()** (Chỉ có trong java 1.4 trở lên) phương thức này trả về true nếu bạn **đã từng** kết nối đến máy ở xa hay chưa và nhận về false trong trường hợp ngược lại.

Thế nên để kiểm tra xem Socket có đang mở hay không bạn cần sử dụng đoạn chương trình sau:

```
boolean connected = socket.isConnected() && ! socket.isClosed();
```

- Phương thức **public boolean isBound()** (chỉ có trong java 1.4 trở lên) phương thức này trả về true nếu như Socket được liên kết với một cổng cục bộ nào đó.

3.4. Truyền trong chế độ haft duplex

Kể từ phiên bản java 1.3 có thêm hai phương thức:

```
public void shutdownInput() throws IOException
```

```
public void shutdownOutput() throws IOException
```

Hai phương thức này cho phép bạn đóng luồng nhập hoặc xuất gắn với Socket lại. khi này bạn chỉ có thể truyền hoặc nhận một chiều (haft duplex).

Ví dụ như khi bạn kết nối đến web server để yêu cầu một trang bất kỳ sau khi yêu cầu xong thì luồng xuất là không còn cần thiết nữa. do vậy ta nên đóng lại để giải phóng tài nguyên.

```
Socket connection = null;
try {
    connection = new Socket("www.theht.edu.vn", 80);
    Writer out = new OutputStreamWriter(connection.getOutputStream());
    out.write("GET / HTTP 1.0\r\n\r\n");
    out.flush();
    /*
    Đóng luồng xuất lại. Sau khi đóng luồng xuất bạn vẫn có thể sử dụng luồng nhập một cách
    bình thường
    */
    connection.shutdownOutput();
    // read the response...
}
catch (IOException ex) {
}
finally {
    try {
```

```
    if (connection != null) connection.close( );
    }
    catch (IOException ex) {}
}
```

4. Lớp ServerSocket

Lớp này chỉ dùng cho phía máy chủ. Nó thường được sử dụng để lắng nghe một kết nối đến.

4.1. Một số hàm tạo của lớp ServerSocket:

```
public ServerSocket(int port) throws BindException, IOException
```

```
public ServerSocket(int port, int queueLength)
```

```
    throws BindException, IOException
```

```
public ServerSocket(int port, int queueLength, InetAddress bindAddress)
```

```
    throws IOException
```

```
public ServerSocket( ) throws IOException // Java 1.4
```

Các hàm tạo này xác định cổng, độ dài của hàng đợi để lưu trữ các yêu cầu kết nối đến và giao diện mạng được sử dụng. Bây giờ ta xét cụ thể từng hàm tạo:

- Hàm tạo **public ServerSocket(int port) throws BindException, IOException**

Hàm tạo này sẽ tạo ra một ServerSocket lắng nghe trên cổng port. Nếu như port =0 thì hệ thống sẽ chọn một giá trị cổng còn rỗi tiếp theo. Tuy nhiên trong trường hợp này máy khách khó có thể kết nối đến nó vì không biết rõ cổng. Tuy nhiên trong một số trường hợp nó rất hữu dụng.

Ví dụ: Tạo một ServerSocket lắng nghe trên cổng 80.

```
try {
    ServerSocket httpd = new ServerSocket(80);
}
catch (IOException ex) {
    System.err.println(ex);
}
```

Chú ý:

- Nếu không thể tạo ra một ServerSocket thì nó sẽ ném ra ngoại lệ IOException
- Nếu port đã được sử dụng cho một chương trình khác rồi thì ngoại lệ **BindException** sẽ được ném ra.

Ví dụ tìm xem các cổng trên máy của bạn đã được sử dụng hay chưa

```
package inetaddress;
```

```
import java.net.*;
import java.io.*;
class LocalPortScanner {
    public static void main(String[] args) {
        for (int port = 1; port <= 65535; port++) {
            try {
                // the next line will fail and drop into the catch block if
                // there is already a server running on the port
                ServerSocket server = new ServerSocket(port);
            }
        }
    }
}
```

```

    }
    catch (IOException ex) {
        System.out.println("There is a server on port " + port + ".");
    } // end catch
} // end for
}
}

```

- **Hàm tạo** `public ServerSocket(int port, int queueLength) throws IOException, BindException`

Hàm tạo này sẽ tạo ra một `ServerSocket` lắng nghe trên cổng *port* và tạo ra một hàng đợi cho phép *queueLength* kết nối nằm trong hàng chờ.

Ví dụ: Tạo ra một `ServerSocket` lắng nghe trên cổng 3456 và cho phép tối đa 100 yêu cầu kết nối đồng thời.

```

try {
    ServerSocket httpd = new ServerSocket(3456, 100);
}
catch (IOException ex) {
    System.err.println(ex);
}

```

- **Hàm tạo** `public ServerSocket(int port, int queueLength, InetAddress bindAddress) throws BindException, IOException`

Hàm tạo này có nhiệm vụ tương tự hàm tạo trước đó. Tuy nhiên `ServerSocket` chỉ lắng nghe trên bộ giao tiếp mạng được chỉ ra trong tham số thứ 3. Nếu máy của bạn có nhiều bộ giao tiếp mạng thì bạn cần phải chỉ ra `ServerSocket` này sẽ lắng nghe trên bộ giao tiếp nào.

Ví dụ: Giả sử rằng máy của bạn có hai bộ giao tiếp mạng, một bộ giao tiếp mạng có địa chỉ 162.163.10.5 và một bộ giao tiếp mạng khác có địa chỉ 192.168.101.1. Để tạo ra một `ServerSocket` lắng nghe trên cổng 3456, bộ giao tiếp mạng 162.163.10.5 (chứ không phải là 192.168.101.1) và cho phép 100 yêu cầu kết nối.

```

try {
    ServerSocket httpd = new ServerSocket(3456, 100,
        InetAddress.getBy_name("162.163.10.5"));
}
catch (IOException ex) {
    System.err.println(ex);
}

```

- **Hàm tạo** `public ServerSocket() throws IOException // Java 1.4`

Hàm tạo này tạo ra một `ServerSocket`, tuy nhiên nó chưa thực sự lắng nghe trên cổng nào cả. Thế nên bạn cũng chưa thể gọi phương thức `accept` để chấp nhận các kết nối. Để thực sự liên kết với một cổng nào đó bạn cần gọi phương thức *bind()*, đây là phương thức chỉ có trong java 1.4 trở lên.

public void bind(SocketAddress endpoint) throws IOException // Java 1.4

public void bind(SocketAddress endpoint, int queueLength) // Java 1.4
throws IOException

Hàm tạo này cũng rất hữu dụng khi bạn cần đặt một số thuộc tính cho ServerSocket trước khi nó thực sự chấp nhận các kết nối.

Thông thường bạn sử dụng mẫu sau:

```
//Tạo ra một ServerSocket chưa thực sự liên kết với một cổng nào đó.  
ServerSocket ss = new ServerSocket( );  
    // Thiết đặt một số thuộc tính cho Socket...  
    //Tạo ra một đối tượng SocketAddress để liên kết  
SocketAddress http = new InetSocketAddress(80);  
    //Liên kết ServerSocket với cổng 80  
ss.bind(http);
```

4.2. Chấp nhận và đóng kết nối

a) Chấp nhận kết nối

Việc tạo ra một ServerSocket chưa thực sự chấp nhận các kết nối đến. Để chấp nhận kết nối đến ta cần gọi đến phương thức *accept()* phương thức này sẽ tạm ngưng tuyến thì hành hiện tại cho đến khi có một yêu cầu kết nối đến. Khi có yêu cầu kết nối đến thì tuyến được tiếp tục và hàm này trả về cho ta một đối tượng Socket. Việc giao tiếp với máy khách được tiến hành thông qua Socket này.

Chú ý: Phương thức *accept()* chỉ chấp nhận một kết nối đến mà thôi. do vậy để có thể chấp nhận nhiều kết nối đến bạn cần phải lặp lại việc gọi đến phương thức *accept()* bằng cách đưa chúng vào một vòng lặp như sau:

Ví dụ:

Chương trình trên máy chủ

```
//Server.java  
import java.net.*;  
import java.io.*;  
  
class Server {  
    public static void main(String[] args) throws IOException {  
        ServerSocket server = new ServerSocket(3456);  
        while (true) {  
            Socket connection = server.accept();  
            OutputStreamWriter out  
                = new OutputStreamWriter(connection.getOutputStream());  
            out.write("You've connected to this server. Bye-bye now.\n");  
            out.flush();  
            connection.close();  
        }  
    }  
}
```

Chương trình trên máy khách

```
//Client.java  
import java.net.*;  
import java.io.*;  
class Client {  
    public static void main(String[] args) throws IOException {  
        Socket sk = new Socket();
```



```

SocketAddress sv = new InetSocketAddress("theht1", 3456);
sk.connect(sv);
InputStream in = sk.getInputStream();
int c;
do {
    c = in.read();
    if (c == -1)break;
    System.out.print( (char) c);
}
while (true);
}
}

```

Với chương trình trên bạn chạy Server một lần sau đó bạn có thể chạy nhiều chương trình máy khách.

b) Đóng kết nối

Khi kết nối không còn được sử dụng đến nữa thì ta cần đóng lại để giải phóng các tài nguyên. Kết nối cũng tự động được đóng lại trong một số trường hợp như: Chương trình kết thúc, bộ thu rác được tiến hành..., Tuy nhiên một phong cách lập trình tốt là luôn đóng lại một cách tường minh khi không còn sử dụng đến nó nữa.

Ví dụ:

```
ServerSocket server = new ServerSocket(port);
```

```
//Làm một số việc
```

```
server.close();
```

Trong java 1.4 bổ sung thêm hai phương thức:

- **public** boolean isClosed() //Nếu kết nối đang mở thì trả về true và ngược lại, tuy nhiên nếu kết nối chưa được mở thì nó cũng trả về false

- **public** boolean isBound() // Trả về true nếu nó đã từng được liên kết với một cổng nào đó.

Thế nên để kiểm tra xem ServerSocket có đang mở hay không ta phải làm như sau:

```
public static boolean isOpen(ServerSocket ss) {
    return ss.isBound( ) && ! ss.isClosed( );
}

```

5. Lớp DatagramSocket

Lớp này dùng để tạo ra một Socket để liên lạc bằng giao thức UDP.

6. Lớp DatagramPacket

Lớp này thể hiện cho một gói dữ liệu được gửi đi bằng giao thức UPD. Nó được sử dụng với lớp *DatagramSocket*.

III. TCP Socket

- Liên lạc trong chế độ kết nối sẽ có một đường kết nối “ảo” được thành lập giữa 2 tiến trình, trước khi một kết nối được thành lập thì một trong 2 tiến trình phải đợi tiến trình kia yêu cầu kết nối. Trong mô hình này server phải lắng nghe và chấp nhận một yêu cầu kết nối đến từ Client

- Để viết các chương trình liên lạc với nhau trong chế độ kết nối ta cần phải viết 2 chương trình: một cho server, một cho client, chương trình server có nhiệm vụ lắng nghe và phản hồi kết nối mới, còn chương trình phía client thì khởi xướng yêu cầu kết nối
- Liên lạc trong chế độ kết nối sẽ sử dụng giao thức TCP, để gửi và nhận dữ liệu

Viết chương trình phía server

Để thiết lập một server ta theo các bước sau:

Bước 1) tạo ra một đối tượng ServerSocket và liên kết nó với một cổng, ta có thể làm việc này thông qua hàm tạo hoặc khởi gán trực tiếp cho các trường ví dụ:

```
ServerSocket s=new ServerSocket(port [, queueLength]);
```

Hoặc

```
ServerSocket s=new ServerSocket();
s.bind(InetAddress addr, port);
```

Bước 2) chấp nhận kết nối bằng cách gọi phương thức *accept()* của đối tượng ServerSocket, sau khi gọi phương thức, tuyến này sẽ bị phong tỏa cho đến khi có một kết nối đến, phương thức này trả về một đối tượng Socket, ta sẽ sử dụng đối tượng này để “giao tiếp” với máy khách

```
Socket connectToClient=s.accept();
```

Bước 3) nhận về các đối tượng outputStream và inputStream từ đối tượng Socket nhận về từ phương thức accept ở bước 2, các đối tượng này sẽ giúp ta gửi/nhận dữ liệu từ máy client, đối tượng outputStream cho phép ta gửi dữ liệu đến client, để nhận được các đối tượng này ta gọi phương thức *getOutputStream()* trên đối tượng Socket, để nhận tham chiếu đến đối tượng outputStream vốn liên kết với đối tượng này, và gọi phương thức *getInputStream()* để nhận một tham chiếu đến đối tượng inputStream vốn liên kết với đối tượng Socket này.

Chúng ta có thể sử dụng đối tượng inputStream và outputStream để nhận/gửi dữ liệu dưới dạng thô, hoặc ta có thể nối nó với một luồng nhập/xuất khác để nhận về dữ liệu mong muốn.

Bước 4) Xử lý dữ liệu gửi đến từ client và gửi trả dữ liệu.

Viết chương trình phía máy khách

Việc viết chương trình cho client gồm một số bước cơ bản sau:

Bước 1) tạo ra một Socket để kết nối đến máy chủ, nối kết đến máy chủ được thực hiện thông qua hàm tạo của lớp Socket, thông thường ta sử dụng hàm tạo 2 đối như sau:

```
Socket connectToServer=new Socket(serverName,port);
```

Hàm tạo này có 2 đối: đối thứ nhất cho biết tên của máy chủ(tên này có thể là tên miền hoặc địa chỉ IP hoặc có thể là một đối tượng InetAddress kết hợp với máy chủ) còn đối thứ 2 cho biết số hiệu cổng mà server đang lắng nghe

Bước 2) Nhận về các đối tượng inputStream và outputStream kết hợp với đối tượng Socket *connectToServer*, bước này tương tự như bước 3 trong phần viết chương trình cho server

Bước 3) xử lý dữ liệu nhận về/ gửi đi thông qua 2 đối tượng outputStream và inputStream

Bước 4) đóng kết nối để giải phóng tài nguyên, thông thường việc đóng kết nối được khởi xướng từ phía máy khách

Ví dụ 1: Nhận về trang index.html từ máy chủ web www.theht.edu.vn và in ra màn hình

```
import java.net.*;
import java.io.*;
class SocketClient {
    public static void main(String[] args) {
        PrintWriter out = null;
        try {
            Socket http = new Socket("www.theht.edu.vn", 80);
            OutputStream raw = http.getOutputStream();
            OutputStream buffered = new BufferedOutputStream(raw);
            out = new PrintWriter(buffered);
            out.write("GET /index.html HTTP/1.0\r\n\r\n");
            out.flush();
            InputStream in = http.getInputStream();
            int c;
            do {
                c = in.read();
                if (c == -1)break;
                System.out.print( (char) c);
            }
            while (true);
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
        finally {
            try {
                out.close();
            }
            catch (Exception ex) {}
        }
    }
}
```

Ví dụ 2: Xây dựng chương trình EchoClient và EchoServer.

Xây dựng chương trình phía máy chủ

```
import java.net.*;
import java.io.*;
public class EchoServer {
    public static void main(String args[]){
        try{
            ServerSocket server = new ServerSocket(3456);
            int localPort = server.getLocalPort();
            System.out.println("Echo Server is listening on port"+localPort+".");
            Socket client = server.accept();
            String destName =client.getInetAddress().getHostName();
            int destPort = client.getPort();
            System.out.println("Accepted connection to "+destName
```

```

        +" on port "+destPort+".");
    BufferedReader inStream = new BufferedReader(
        new InputStreamReader(client.getInputStream()));
    DataOutputStream outStream = new
        DataOutputStream(client.getOutputStream());
    boolean finished = false;
    do {
        String inLine = inStream.readLine();
        System.out.println("Received: "+inLine);
        if(inLine.equalsIgnoreCase("quit")) finished=true;
        String outLine=new StringBuffer(inLine.trim()).reverse().toString();
        outStream.write(outLine.getBytes());
        outStream.write(13);
        outStream.write(10);
        outStream.flush();
        System.out.println("Sent: "+outLine);
    } while(!finished);
    inStream.close();
    outStream.close();
    client.close();
    server.close();
} catch (IOException e){
    System.out.println(e);
}
}
}
}

```

Xây dựng chương trình phía máy khách

```

import java.net.*;
import java.io.*;
public class EchoClient {
    public static void main(String args[]) {
        ClientConnect client = new ClientConnect("www.theht.edu.vn", 3456);
        client.requestServer();
        client.shutdown();
    }
}
class ClientConnect {
    Socket connection;
    DataOutputStream out;
    BufferedReader in;
    public ClientConnect(String destination, int port) {
        try {
            connection = new Socket(destination, port);
            in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
            out = new DataOutputStream(connection.getOutputStream());
            System.out.println("Connected to server at port 3456.");
        }
        catch (Exception e) { System.out.println(e); }
    }
}

```

```

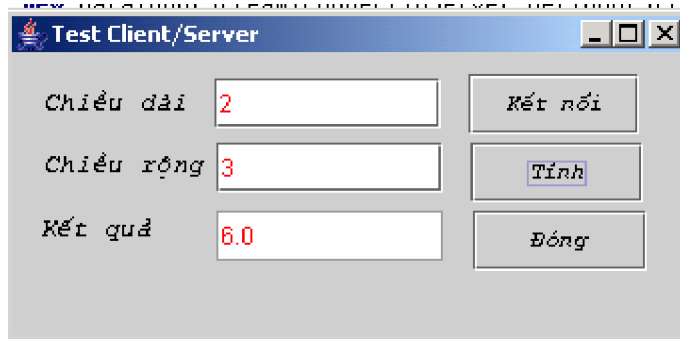
}
public void requestServer() {
    BufferedReader keyboardInput = new BufferedReader(
                                                new InputStreamReader(System.in));

    boolean finished = false;
    do {
        try {
            System.out.print("Send, receive, or quit (S/R/Q): ");
            System.out.flush();
            String line = keyboardInput.readLine();
            if (line.length() > 0) {
                line = line.toUpperCase();
                switch (line.charAt(0)) {
                    case 'S':
                        String sendLine = keyboardInput.readLine();
                        out.writeBytes(sendLine);
                        out.write(13);
                        out.write(10);
                        out.flush();
                        break;
                    case 'R':
                        int inByte;
                        System.out.print(">>>>");
                        while ( (inByte = in.read()) != '\n')
                            System.out.write(inByte);
                        System.out.println(); break;
                    case 'Q':
                        finished = true; break;
                    default: break;
                }
            }
        }
        catch (Exception e) {
            System.out.println(e);
        }
    }
    while (!finished);
}
public void shutdown() {
    try {
        connection.close();
    }
    catch (IOException ex) {
        System.out.println("IO error closing socket");
    }
}
}
}

```

Ví dụ: ta viết một ví dụ đơn giản như sau:

* chương trình phía client có giao diện sau:



khi ta kích chuột vào nút kết nối chương trình của ta sẽ kết nối với máy chủ sau khi nhập chiều dài, chiều rộng và kích chuột vào nút tính, dữ liệu gồm có chiều dài và chiều rộng sẽ được gửi đến máy chủ, máy chủ thực hiện việc tính diện tích và gửi kết quả trả lại cho máy khách, dữ liệu nhận về từ máy chủ được hiển thị trong ô kết quả khi ta kích chuột nút đóng, chương trình của ta sẽ đóng kết nối và kết thúc chương trình

* chương trình trên máy chủ chỉ có nhiệm vụ, lắng nghe kết nối và tính toán diện tích của hình nhận được

Mã nguồn phía máy chủ:

```
package net.theht;
import java.io.*;
import java.net.*;
public class Server
{
    public Server()
    {
        double a = 0, b = 0, kq = 0;
        try
        {
            ServerSocket s = new ServerSocket(2004); // nam Giap Than
            Socket connectToClient = s.accept(); // chấp nhận kết nối
            // lấy về đối tượng nhập/xuất
            DataInputStream in = new DataInputStream(connectToClient.getInputStream());
            DataOutputStream out = new DataOutputStream(connectToClient.getOutputStream());
            //Vòng lặp vô tận để phục vụ yêu cầu
            while (true) {
                a = in.readDouble();// lấy chiều dài
                b = in.readDouble();// lấy chiều rộng
                kq = a * b;// tính kết quả
                out.writeDouble(kq);// trả về kết quả cho máy khách
                out.flush();// dôn hết DL ra
            }
        }
        catch (IOException ex) { }
    }
    public static void main(String[] args)
    {

```

```
    Server server1 = new Server();  
  }  
}
```

Mã nguồn phía máy khách:

```
package net.theht;  
import javax.swing.*;  
import java.awt.*;  
import java.io.*;  
import java.net.*;  
import java.awt.event.*;  
public class Frame1 extends JFrame{  
    JLabel jLabel1 = new JLabel();  
    JLabel jLabel2 = new JLabel();  
    JTextField jtfDai = new JTextField();  
    JTextField jtfRong = new JTextField();  
    JLabel jLabel3 = new JLabel();  
    JTextField jtfKetQua = new JTextField();  
    JButton jtfConnect = new JButton();  
    JButton jtfCompute = new JButton();  
    JButton jtfClose = new JButton();  
    Socket connectToServer;  
    DataInputStream in; // luồng nhập  
    DataOutputStream out; //luồng xuất  
    boolean isConnect = false; //biến kiểm tra xem đã kết nối chưa  
    public Frame1() {  
        try { jblnit(); }  
        catch (Exception ex) { ex.printStackTrace(); }  
    }  
    void jblnit() throws Exception {  
        jLabel1.setFont(new java.awt.Font(".VnCourier New", 0, 13));  
        jLabel1.setToolTipText("");  
        jLabel1.setText("Chiều dài");  
        jLabel1.setBounds(new Rectangle(16, 15, 77, 21));  
        this.getContentPane().setLayout(null);  
        jLabel2.setBounds(new Rectangle(16, 46, 86, 21));  
        jLabel2.setText("Chiều rộng");  
        jLabel2.setFont(new java.awt.Font(".VnCourier New", 0, 13));  
        jtfDai.setForeground(Color.red);  
        jtfDai.setBounds(new Rectangle(101, 12, 113, 25));  
        jtfRong.setBounds(new Rectangle(102, 44, 113, 25));  
        jtfRong.setForeground(Color.red);  
        jLabel3.setFont(new java.awt.Font(".VnCourier New", 0, 13));  
        jLabel3.setText("Kết quả");  
        jLabel3.setBounds(new Rectangle(14, 78, 57, 21));  
        jtfKetQua.setBackground(Color.white);  
        jtfKetQua.setForeground(Color.red);  
        jtfKetQua.setDisabledTextColor(Color.red);  
        jtfKetQua.setEditable(false);  
        jtfKetQua.setText("");  
    }  
}
```

```

jtfKetQua.setBounds(new Rectangle(102, 78, 113, 25));
jtfConnect.setBounds(new Rectangle(228, 10, 85, 30));
jtfConnect.setFont(new java.awt.Font(".VnCourier New", 0, 12));
jtfConnect.setHorizontalAlignment(SwingConstants.CENTER);
jtfConnect.setHorizontalTextPosition(SwingConstants.CENTER);
jtfConnect.setText("Kết nối");
jtfConnect.setVerticalAlignment(javax.swing.SwingConstants.CENTER);
jtfConnect.setVerticalTextPosition(javax.swing.SwingConstants.CENTER);
jtfConnect.addActionListener(new Frame1_jtfConnect_actionAdapter(this));
jtfCompute.setText("Tính");
jtfCompute.addActionListener(new Frame1_jtfCompute_actionAdapter(this));
jtfCompute.setFont(new java.awt.Font(".VnCourier New", 0, 12));
jtfCompute.setBounds(new Rectangle(229, 44, 86, 30));
jtfClose.setText("Đóng");
jtfClose.addActionListener(new Frame1_jtfClose_actionAdapter(this));
jtfClose.setFont(new java.awt.Font(".VnCourier New", 0, 12));
jtfClose.setBounds(new Rectangle(230, 78, 87, 30));
this.setDefaultCloseOperation(EXIT_ON_CLOSE);
this.setTitle("Test Client/Server");
this.getContentPane().add(jLabel1, null);
this.getContentPane().add(jtfKetQua, null);
this.getContentPane().add(jtfRong, null);
this.getContentPane().add(jtfDai, null);
this.getContentPane().add(jLabel2, null);
this.getContentPane().add(jLabel3, null);
this.getContentPane().add(jtfCompute, null);
this.getContentPane().add(jtfConnect, null);
this.getContentPane().add(jtfClose, null);
}

public static void main(String[] args) {
    Frame1 frame1 = new Frame1();
    frame1.setBounds(100, 100, 350, 130);
    frame1.show();
}

void jtfConnect_actionPerformed(ActionEvent e) {
    if (isConnect == false) // nếu chưa kết nối thì kết nối
    {
        try {
            // lắng nghe kết nối đến máy chủ trên cổng 2004
            connectToServer = new Socket("localhost", 2004);
            isConnect = true;
            in = new DataInputStream(connectToServer.getInputStream());
            out = new DataOutputStream(connectToServer.getOutputStream());
        }
        catch (UnknownHostException ex)
        {
            JOptionPane.showMessageDialog(this, "khong tim thay may chu", "loi",
                JOptionPane.ERROR_MESSAGE);
        }
    }
}

```



```

    catch (IOException ex) {
        JOptionPane.showMessageDialog(this, "co loi ve nhap xuat", "loi",
            JOptionPane.ERROR_MESSAGE);
    }
}

void jtfCompute_actionPerformed(ActionEvent e) {
    if ( (in == null) || (out == null)) // nếu chưa kết nối thì báo lỗi
        JOptionPane.showMessageDialog(this, "Ban phai ket noi truooc khi tinh",
            "loi", JOptionPane.ERROR_MESSAGE);
    try {
        out.writeDouble(Double.parseDouble(jtfDai.getText())); // gửi chiều dài
        out.writeDouble(Double.parseDouble(jtfRong.getText())); // gửi chiều rộng
        out.flush(); // đẩy hết dữ liệu ra
        jtfKetQua.setText(String.valueOf(in.readDouble())); // nhận về kết quả và
        // hiển thị trong ô kết quả
    }
    catch (NumberFormatException ex) {
        // nếu nhập sai thì báo lỗi
        JOptionPane.showMessageDialog(this, "Ban nhap sai, Ban phai nhap so",
            "loi", JOptionPane.ERROR_MESSAGE);
    }
    catch (IOException ex) {
        JOptionPane.showMessageDialog(this,
            "khong the guu/nhan du lieu\nCo the do ban chua ket noi",
            "loi",
            JOptionPane.ERROR_MESSAGE);
    }
}

// thủ tục đáp ứng biến cố khi kích chuột vào nút đóng
void jtfClose_actionPerformed(ActionEvent e) {
    try { connectToServer.close(); // đóng kết nối }
    catch (IOException ex) { }
    System.exit(0); // thoát khỏi chương trình
}

// tạo ra lớp điều hợp, để đáp ứng sự kiện kích chuột vào nút kết nối
class Frame1_jtfConnect_actionAdapter implements java.awt.event.ActionListener{
    Frame1 adaptee;
    Frame1_jtfConnect_actionAdapter(Frame1 adaptee)
    { this.adaptee = adaptee; }
    public void actionPerformed(ActionEvent e)
    { adaptee.jtfConnect_actionPerformed(e); }
}

// tạo ra lớp điều hợp, để đáp ứng sự kiện kích chuột vào nút tính
class Frame1_jtfCompute_actionAdapter implements java.awt.event.ActionListener
{ Frame1 adaptee;

```

```

Frame1_jtfCompute_actionAdapter(Frame1 adaptee)
{
    this.adaptee = adaptee;
}
public void actionPerformed(ActionEvent e) {
    adaptee.jtfCompute_actionPerformed(e);
}
}

// tạo ra lớp điều hợp, để đáp ứng sự kiện kích chuột vào nút đóng
class Frame1_jtfClose_actionAdapter implements java.awt.event.ActionListener{
    Frame1 adaptee;
    Frame1_jtfClose_actionAdapter(Frame1 adaptee) {
        this.adaptee = adaptee;
    }
    public void actionPerformed(ActionEvent e) {
        adaptee.jtfClose_actionPerformed(e);
    }
}

```

Vấn đề một nguồn nhiều khách

Trong ví dụ trên ta nhận thấy, máy chủ chỉ có thể phục vụ duy nhất một khách. Trong thực tế đối với một số bài toán yêu cầu một máy chủ phải phục vụ nhiều máy khách tại một thời điểm, ví dụ như Web server, Database server,... các máy chủ này cho phép nhiều máy nhiều máy khách kết nối đồng thời, ta có thể làm được việc này bằng kỹ thuật đa luồng. Thật đơn giản, bạn chỉ cần tạo ra một luồng phục vụ cho mỗi khách, và đưa thêm vào đoạn mã tựa như sau:

```

ServerSocket s = new ServerSocket(port);
while (true) {
    Socket connectToClient = s.accept();
    new HandleClient(connectToClient);
}

```

Ta sửa lại ví dụ trên để có thể phục vụ được nhiều máy khách, chương trình phía máy khách vẫn giữ nguyên, ta chỉ thay đổi chương trình trên máy chủ như sau:

```
package net.theht;
```

```

import java.io.*;
import java.net.*;
public class Server{
    public static void main(String[] args)
    { try { //lắng nghe kết nối trên cổng 2004
        ServerSocket s = new ServerSocket(2004);
        while (true) {
            Socket connectToClient = s.accept(); // chấp nhận kết nối
            new HandleClient(connectToClient);//chạy một luồng phục vụ
        }
    }
    catch (IOException ex) { }
}

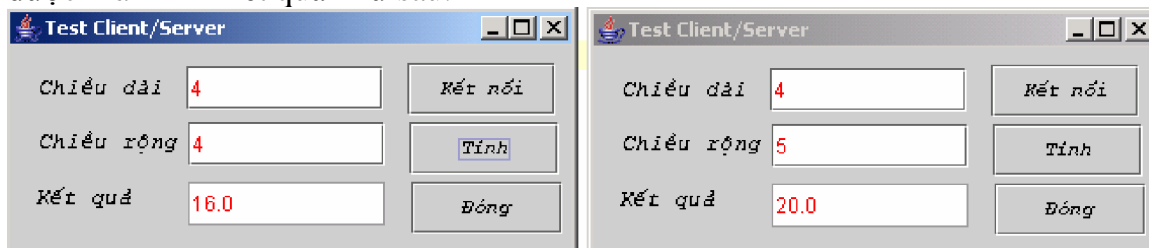
```

```

}
class HandleClient extends Thread{
    private Socket s;
    private double a = 0, b = 0, kq;
    public HandleClient(Socket s)
    {   this.s = s;
        this.start();
    }
    public void run()
    {   DataInputStream in = null;
        DataOutputStream out = null;
        try
        {   in = new DataInputStream(s.getInputStream());
            out = new DataOutputStream(s.getOutputStream());
        }
        catch (IOException ex) { }
        //Vòng lặp vô tận để phục vụ yêu cầu
        while (true)
        {   try {
            a = in.readDouble(); // lấy chiều dài
            b = in.readDouble(); // lấy chiều rộng
            kq = a * b; // tính kết quả
            out.writeDouble(kq); // trả về kết quả cho máy khách
            out.flush(); // dôn hết DL ra
        }
        catch (IOException ex1) { }
        }
    }
}
}

```

sau khi sửa lại bạn cho khởi động server, sau đó cho chạy 2 tiến trình client, bạn sẽ thu được màn hình kết quả như sau:



Liên lạc trong chế độ phi kết nối

Việc liên lạc trong chế độ không kết nối, không yêu cầu phải có 1 server luôn lắng nghe kết nối, khi liên lạc trong chế độ không kết nối java sử dụng giao thức UDP trong việc điều khiển truyền và nhận dữ liệu. Do vậy không có xác nhận về dữ liệu gửi, không thể biết được dữ liệu gửi đi có đến được đích không, không thể biết được dữ liệu nhận về có bị trùng lặp không... Bạn là người lập trình, bạn phải giải quyết điều đó, chứ không phải lớp giao thức mạng hay java, Trong thực tế họ mặc kệ cho lỗi xảy ra, bởi họ không đủ năng lực

để làm điều đó, thế nên giải pháp an toàn cho bạn là Socket dòng, bởi lẽ Socket dòng sử dụng giao thức TCP, nên những điều lo lắng đó không bao giờ xảy ra.

Để liên lạc trong chế độ không kết nối ta sử dụng lớp *DatagramSocket* và lớp *DatagramPacket*, lớp *DatagramSocket* cho phép ta gửi/ nhận về các gói dữ liệu từ máy ở xa, lớp *DatagramPacket* cho cung cấp cho ta công cụ đóng gói dữ liệu UDP, để đối tượng *DatagramSocket* gửi đi hoặc nhận về.

Ta giải thích thông qua chương trình chat, chương trình gồm có 2 chương trình, hai chương trình này là ngang hàng không có chương trình nào là chủ cả, các lệnh cần thiết để liên lạc được trong chế độ không kết nối được in đậm

Mã chương trình thứ nhất:

```
package net.theht;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.io.*;
import java.net.*;
public class Frame2 extends JFrame{
    private JTextArea jtaSend = new JTextArea();
    private JButton jbtSend = new JButton();
    private JButton jbtExit = new JButton();
    private JTextArea jtaRev = new JTextArea();
// khai báo các biến cần thiết
    private DatagramSocket s;
    private DatagramPacket sen,rev;
    private boolean isFirstLine=true;
    private JSplitPane jSplitPane1 = new JSplitPane();
    private JScrollPane jScrollPane1 = new JScrollPane();
    private JScrollPane jScrollPane2 = new JScrollPane();
    public Frame2()
    { try {
        jblnit();
    }
    catch (Exception ex) { ex.printStackTrace(); }
    }
    void jblnit() throws Exception {
        this.getContentPane().setLayout(null);
        jtaSend.setText("");
        jbtSend.setBounds(new Rectangle(315, 205, 69, 24));
        jbtSend.setFont(new java.awt.Font(".VnCourier New", 0, 12));
        jbtGửiid.seGửiixt(Gửiiu");
        jbtSend.addActionListener(new Frame2_jbtSend_actionAdapter(this));
        jbtExit.setText("Thoat");
        jbtExit.addActionListener(new Frame2_jbtExit_actionAdapter(this));
        jbtExit.setFont(new java.awt.Font(".VnCourier New", 0, 12));
        jbtExit.setBounds(new Rectangle(314, 174, 71, 25));
        jtaRev.setDisabledTextColor(Color.gray);
        jtaRev.setEditable(false);
        jtaRev.setText("");
    }
}
```

```

jSplitPane1.setOrientation(JSplitPane.VERTICAL_SPLIT);
jSplitPane1.setBounds(new Rectangle(6, 7, 300, 224));
this.setDefaultCloseOperation(HIDE_ON_CLOSE);
this.setTitle("Chương trình chat sử dụng giao thức UDP");
this.addWindowListener(new Frame2_this_windowAdapter(this));
this.getContentPane().add(jSplitPane1, null);
jSplitPane1.add(jScrollPane1, JSplitPane.TOP);
jScrollPane1.getViewport().add(jtaRev, null);
jSplitPane1.add(jScrollPane2, JSplitPane.BOTTOM);
this.getContentPane().add(jbtSend, null);
this.getContentPane().add(jbtExit, null);
jScrollPane2.getViewport().add(jtaSend, null);
jSplitPane1.setDividerLocation(150);
//user code
try {
// tạo ra đối tượng DatagramSocket đối tượng này sẽ nhận dữ //liệu từ cổng 2004
    s = new DatagramSocket(2004);
}
catch (SocketException ex) { }
}

private void nhan() {
    while (true) {
// vùng đệm để nhận dữ liệu
        byte[] dat = new byte[100];
// tạo ra đối tượng DatagramPacket để nhận gói dữ liệu
        rev = new DatagramPacket(dat, dat.length);
        try {
            s.receive(rev);// nhận về dữ liệu
            if (isFirstLine == true) {
                jtaRev.append(new String(rev.getData(), 0, rev.getLength()));
                isFirstLine = false;
            }
            else
                jtaRev.append("\n" + new String(rev.getData(), 0, rev.getLength()));
        }
        catch (IOException ex) { }
    }
}

private void dong() {
    try {
        s.close();// đóng kết nối
    }
    catch (Exception ex) { }
    System.exit(0);
}

public static void main(String[] args) {
    Frame2 frame2 = new Frame2();
}

```

```

frame2.setBounds(100, 100, 400, 280);
frame2.show();
frame2.nhan();
}
void jbtSend_actionPerformed(ActionEvent e) {
    byte dat[] = jtaSend.getText().getBytes();
    InetAddress local = null;
    try {
        local = InetAddress.getLocalHost();
    }
    catch (UnknownHostException ex) { }
        // Tạo ra đối tượng DatagramPacket để gửi gói dữ liệu
    sen = new DatagramPacket(dat, dat.length, local, 4002);
    try {
        s.send(sen);           // Gửi dữ liệu đi
    }
    catch (IOException ex1) {
        JOptionPane.showMessageDialog(this, "khong the gui duoc");
    }
}

void jbtExit_actionPerformed(ActionEvent e) {
    dong();
    System.exit(0);
}

void this_windowClosing(WindowEvent e) {
    dong();
}

class Frame2_jbtSend_actionAdapter implements java.awt.event.ActionListener{
    private Frame2 adaptee;
    Frame2_jbtSend_actionAdapter(Frame2 adaptee) {
        this.adaptee = adaptee;
    }
    public void actionPerformed(ActionEvent e) {
        adaptee.jbtSend_actionPerformed(e);
    }
}

class Frame2_jbtExit_actionAdapter implements java.awt.event.ActionListener{
    private Frame2 adaptee;
    Frame2_jbtExit_actionAdapter(Frame2 adaptee) {
        this.adaptee = adaptee;
    }
    public void actionPerformed(ActionEvent e) {
        adaptee.jbtExit_actionPerformed(e);
    }
}

```

```

}

class Frame2_this_windowAdapter extends java.awt.event.WindowAdapter{
    private Frame2 adaptee;
    Frame2_this_windowAdapter(Frame2 adaptee) {
        this.adaptee = adaptee;
    }
    public void windowClosing(WindowEvent e) {
        adaptee.this_windowClosing(e);
    }
}

```

Chương trình thứ hai:

package net.theht;

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
public class Frame3 extends JFrame{
    private JTextArea jtaRev = new JTextArea();
    private JScrollPane jScrollPane1 = new JScrollPane();
    private JButton jbtSend = new JButton();
    private JScrollPane jScrollPane2 = new JScrollPane();
    private JButton jbtExit = new JButton();
    private JTextArea jtaSend = new JTextArea();
    private DatagramSocket s;
    private DatagramPacket sen, rev;
    private boolean isFirstLine = true;
    private JSplitPane jSplitPane1 = new JSplitPane();
    public Frame3() {
        try {
            jblnit();
        }
        catch (Exception ex) {    ex.printStackTrace(); }
    }

    void jblnit() throws Exception {
        jtaRev.setEditable(false);
        jtaRev.setText("");
        this.getContentPane().setLayout(null);
        jbtSend.setBounds(new Rectangle(315, 183, 69, 24));
        jbtSend.setFont(new java.awt.Font(".VnCourier New", 0, 12));
        jbtGửiid.seGửiixt(Gửiiu");
        jbtSend.addActionListener(new Frame3_jbtSend_actionAdapter(this));
        jbtSend.addActionListener(new Frame3_jbtSend_actionAdapter(this));
        jbtExit.setText("Thoat");
        jbtExit.addActionListener(new Frame3_jbtExit_actionAdapter(this));
    }
}

```

```

jbtExit.setFont(new java.awt.Font(".VnCourier New", 0, 12));
jbtExit.setBounds(new Rectangle(314, 156, 71, 24));
jSplitPane1.setOrientation(JSplitPane.VERTICAL_SPLIT);
jSplitPane1.setBounds(new Rectangle(5, 7, 296, 219));
this.setDefaultCloseOperation(HIDE_ON_CLOSE);
this.setTitle("Chuong trinh chat su dung giao thuc UDP");
this.addWindowListener(new Frame3_this_windowAdapter(this));
this.getContentPane().add(jSplitPane1, null);
jSplitPane1.add(jScrollPane1, JSplitPane.TOP);
jScrollPane1.getViewport().add(jtaRev, null);
jScrollPane1.getViewport().add(jtaRev, null);
jSplitPane1.add(jScrollPane2, JSplitPane.BOTTOM);
this.getContentPane().add(jbtExit, null);
this.getContentPane().add(jbtSend, null);
jSplitPane1.setDividerLocation(150);
//user code
try {
    s = new DatagramSocket(4002);
}
catch (SocketException ex) { }
jScrollPane2.getViewport().add(jtaSend, null);
}

private void nhan() {
    while (true) {
        byte[] dat = new byte[100];
        rev = new DatagramPacket(dat, dat.length);
        try {
            s.receive(rev);
            if (isFirstLine == true) {
                jtaRev.append(new String(rev.getData(), 0, rev.getLength()));
                isFirstLine = false;
            }
            else
                jtaRev.append("\n" + new String(rev.getData(), 0, rev.getLength()));
        }
        catch (IOException ex) { }
    }
}

private void dong() {
    try {
        s.close();
    }
    catch (Exception ex) { }
    System.exit(0);
}

public static void main(String[] args) {

```



```

Frame3 frame3 = new Frame3();
frame3.setBounds(100, 100, 400, 280);
frame3.show();
frame3.nhan();
}
void jbtSend_actionPerformed(ActionEvent e) {
    byte dat[] = jtaSend.getText().getBytes();
    InetAddress local = null;
    try {
        local = InetAddress.getLocalHost();
    }
    catch (UnknownHostException ex) { }
    sen = new DatagramPacket(dat, dat.length, local, 2004);
    try {
        s.send(sen);
    }
    catch (IOException ex1) {
        JOptionPane.showMessageDialog(this, "khong the guu duoc");
    }
}

void jbtExit_actionPerformed(ActionEvent e) {
    dong();
}
void this_windowClosing(WindowEvent e) {
    dong();
}
}
class Frame3_jbtSend_actionAdapter implements java.awt.event.ActionListener{
    private Frame3 adaptee;
    Frame3_jbtSend_actionAdapter(Frame3 adaptee) {
        this.adaptee = adaptee;
    }
    public void actionPerformed(ActionEvent e) {
        adaptee.jbtSend_actionPerformed(e);
    }
}
class Frame3_jbtExit_actionAdapter implements java.awt.event.ActionListener{
    private Frame3 adaptee;
    Frame3_jbtExit_actionAdapter(Frame3 adaptee) {
        this.adaptee = adaptee;
    }
    public void actionPerformed(ActionEvent e) {
        adaptee.jbtExit_actionPerformed(e);
    }
}
class Frame3_this_windowAdapter extends java.awt.event.WindowAdapter{
    private Frame3 adaptee;
    Frame3_this_windowAdapter(Frame3 adaptee) {

```

```

this.adaptee = adaptee;
}
public void windowClosing(WindowEvent e) {
    adaptee.this_windowClosing(e);
}
}

```

Tóm lại để liên lạc trong chế độ không kết nối ta cần viết trên mỗi máy các lệnh cần thiết sau:

tạo ra đối tượng *DatagramSocket* để nhận và gửi dữ liệu, bằng hàm tạo của nó như sau:
s = **new** DatagramSocket(port);

Hàm tạo này sẽ tạo ra đối tượng *DatagramSocket* hoạt động trên cổng được chỉ ra.

tạo các đối tượng *DatagramPacket* dùng để làm gói dữ liệu trong việc gửi/nhận dữ liệu, bằng cách sử dụng hàm tạo của lớp *DatagramPacket*

+ DatagramPacket(byte[] buf, int length) thường dùng hàm tạo này khi nhận dữ liệu

+length.DatagramPacket(byte[] buf, int length, InetAddress address, int port) hàm tạo này thường dùng khi đọc dữ liệu từ máy tính và cổng được chỉ ra

+ DatagramPacket(byte[] buf, int offset, int length) hàm tạo này thường dùng trong việc nhận dữ liệu có độ dài length , bắt đầu từ điểm offset

+ DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port) nhận về dữ liệu trên máy, cổng được chỉ ra, và cất vào trong vùng đệm buf

-gửi dữ liệu bằng phương thức *send*, của lớp *DatagramSocket*

datagramSocket.send(datagramSocket)

- Nhận dữ liệu bằng phương thức *receive*, của lớp *DatagramSocket*

datagramSocket.receive(datagramSocket)

Ví dụ ChatApp.java

```

import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
public class ChatApp extends Frame {
    TextField msgText = new TextField();
    TextArea replyText = new TextArea();
    Button sendButt = new Button();
    Label label1 = new Label();
    TextField addrText = new TextField();
    Sender sender=new Sender();
public ChatApp(){
    this.setLayout(null);
    this.setSize(new Dimension(297, 214));
    this.setTitle("Java Chat");
    msgText.setBounds(new Rectangle(15, 34, 277, 21));
    replyText.setBounds(new Rectangle(16, 93, 274, 89));
    label1.setBounds(new Rectangle(17, 183, 55, 23));
    label1.setText("talking with");

    addrText.setBounds(new Rectangle(84, 187, 208, 19));

```

```

addrText.setText("127.0.0.1");

sendButt.setBounds(new Rectangle(15, 61, 94, 28));
sendButt.setLabel("Send");
sendButt.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        sendButt_actionPerformed(e);
    }
});
this.add(msgText, null);
this.add(replyText, null);
this.add(sendButt, null);
this.add(label1, null);
this.add(addrText, null);
}
void sendButt_actionPerformed(ActionEvent e) {
    sender.sendData(addrText.getText(),msgText.getText());
}
public static void main(String args[]){
    ChatApp app = new ChatApp();
    ListenerThread listener=new ListenerThread(app.replyText);
    listener.start();
    app.addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent event){
            System.exit(0);
        }
    });
    app.show();
}
}
class Sender{
    DatagramSocket listenerSocket;
    InetAddress serverAddress;
    String localhost;
    int bufferSize = 256;
    byte outBuffer[];
    byte inBuffer[] = new byte[bufferLength];
    DatagramPacket outDatagram;
    DatagramPacket inDatagram;
    public void sendData(String DestAddr, String data){
        try{
            serverAddress = InetAddress.getByName(DestAddr);
            outBuffer = new byte[bufferLength];
            outBuffer = data.getBytes();
            outDatagram = new
                DatagramPacket(outBuffer,outBuffer.length,serverAddress,2345);
            listenerSocket.send(outDatagram);
        } catch (Exception e){
            System.out.println(e);
        }
    }
}

```

```

public Sender(){
    try{
        listenerSocket    = new DatagramSocket();
        inDatagram    = new DatagramPacket(inBuffer,inBuffer.length);
    } catch (Exception e){        System.out.println("connect error!");    }
}
}
}
class ListenerThread extends Thread{
    TextArea text;
    DatagramSocket socket;
    InetAddress serverAddress;
    String localhost;
    int bufferSize = 256;
    byte outBuffer[];
    byte inBuffer[] = new byte[bufferLength];
    DatagramPacket outDatagram;
    DatagramPacket inDatagram;
    public ListenerThread(TextArea text){
        this.text=text;
        try{
            socket    = new DatagramSocket();
            inDatagram    = new DatagramPacket(inBuffer,inBuffer.length);
        } catch (Exception e){
            System.out.println("connect error!");
        }
    }
}
public void run() {
    try{
        DatagramSocket socket = new DatagramSocket(2345);

        String localAddress = InetAddress.getLocalHost().getHostName().trim();
        int localPort = socket.getLocalPort();
        int bufferSize = 256;
        byte outBuffer[];
        byte inBuffer[] = new byte[bufferLength];
        DatagramPacket outDatagram;
        DatagramPacket inDatagram = new DatagramPacket(inBuffer,inBuffer.length);
        while (true) {
            socket.receive(inDatagram);
            InetAddress destAddress = inDatagram.getAddress();
            String destHost = destAddress.getHostName().trim();
            int destPort = inDatagram.getPort();
            String data = new String(inDatagram.getData()).trim();
            text.append(destHost+" say: "+data+"\n");
        }
    } catch (IOException ex){        System.out.println("IOException occurred.");    }
}
}
}

```

Chương 9: JAVA MAIL

I. Các khái niệm MAIL

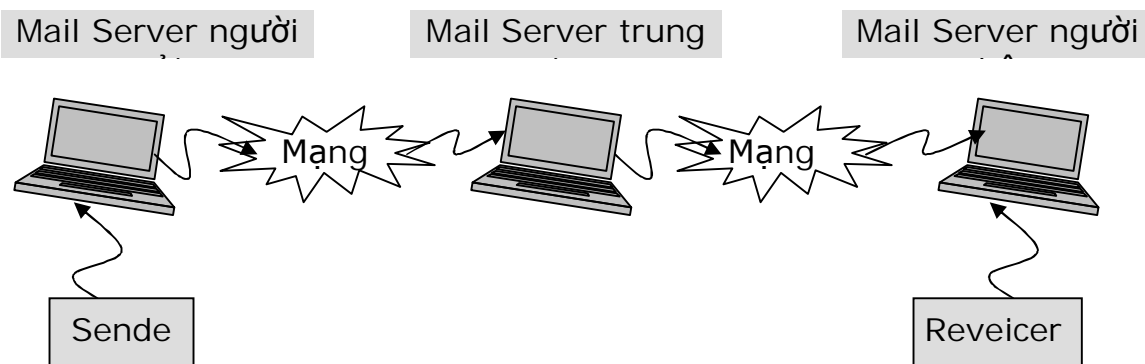
Ngày nay e-mail hay thư điện tử đã trở nên phổ biến và hầu như không thể thiếu trong thế giới Internet. E-mail cho phép gửi thông điệp từ nơi này đến nơi khác tương tự như cách gửi thư truyền thống thông qua bưu điện. Trong cách thông thường bạn phải cung cấp địa chỉ người gửi (sender), địa chỉ người nhận (recipient) sau đó người gửi mang thư đến bưu điện nhờ bưu điện chuyển thư đi. Bưu điện sẽ chuyển thư đến hộp thư của người nhận.

Đối với thư điện tử e-mail gửi qua hệ thống Internet hay mạng thì cơ chế gửi/nhận cũng hoàn toàn tương tự. Các máy chủ (mail server) cung cấp cho bạn địa chỉ mail (còn gọi là e-mail address). Bạn ghi địa chỉ mail của bạn và của người nhận, sau đó nhập vào nội dung cần gửi. Mail server sẽ gửi thư của bạn đến một máy chủ hay mail server của địa chỉ người nhận. Mail server nơi nhận sẽ cất thư vào một nơi thích hợp (mail box hay còn gọi là hộp thư) trên máy chủ. Người nhận sẽ tiến hành đăng nhập (hay còn gọi là login) vào hộp thư của máy chủ để đọc nội dung e-mail gửi cho mình.

Để thống nhất giữa các máy chủ mail trên toàn thế giới về cách thức gửi nhận mail đòi hỏi hình thành các chuẩn về mail. Điều này giúp cho việc gửi nhận các thông điệp được đảm bảo, làm cho những người ở các nơi khác nhau có thể trao đổi thông tin với nhau.

Có 2 chuẩn về Mail thường được các máy chủ mail hỗ trợ là X.400 và SMTP (Simple Mail Transfer Protocol). SMTP thường đi kèm với chuẩn POP3 và do hạn chế của SMTP mà ngày nay người ta dùng chuẩn mở rộng của nó là ESMTP (Extended SMTP). Mục đích chính của X.400 là cho phép các mail có thể được truyền nhận thông qua các loại mạng khác nhau bất chấp cấu hình phần cứng, hệ điều hành mạng, giao thức truyền dẫn được dùng. Còn mục đích của chuẩn SMTP miêu tả cách điều khiển các thông điệp trên mạng Internet. Điều quan trọng của chuẩn SMTP là giả định máy nhận phải dùng giao thức SMTP gửi Mail cho một Server luôn luôn hoạt động. Sau đó, người nhận sẽ đến lấy Mail của họ từ Server khi nào họ muốn dùng giao thức POP (Post Office Protocol), ngày nay POP được cải tiến thành POP3 (Post Office Protocol version 3). Các giao thức Mail thông dụng : chuẩn X.400, chuẩn MAIP, **SMTP** (ESMTP), POP3. Trong chương này ta sẽ sử dụng hai giao thức :POP3 và SMTP để gửi nhận mail.

Mô hình gửi nhận mail



Trước khi đi vào phần chính chúng ta cùng ôn lại một số khái niệm cơ sở sau:

Mail server: là một chương trình chạy dưới dạng một dịch vụ. dịch vụ này có nhiệm vụ nhận mail từ máy khách gửi đến, phân phối mail đến máy chủ khác, cho phép máy khách truy cập vào để lấy mail, lưu trữ mail... Chính vì vậy trước khi bạn gửi nhận mail bạn phải biết địa chỉ của máy chủ mail (máy đang chạy dịch vụ mail). Địa chỉ này thường được gọi là mail host.

Giao thức gửi mail (Mail transport protocol): Để gửi mail đến máy chủ, trình khách phải sử dụng một giao thức trò chuyện với mail server. Tương tự như trình duyệt sử dụng giao thức HTTP để trò chuyện với Web server. Các máy khách muốn bắt tay với trình chủ và gửi mail lên máy chủ sẽ sử dụng giao thức SMTP (Simple Mail Transport Protocol). SMTP khá nổi tiếng và được hầu các máy chủ mail trên thế giới hỗ trợ. Địa chỉ của máy chủ nhận mail gửi đi được gọi là *outgoing mail address*. Trình chủ tiếp nhận mail theo giao thức SMTP được gọi là SMTP Server. Trình khách sử dụng giao thức SMTP để gửi thư đi được gọi là SMTP Client.

Giao thức lưu trữ và nhận mail (mail store protocol): Khi trình chủ mail tiếp nhận được mail. Nó tiến hành lưu trữ mail theo một cách thức nào đó để trình khác có thể dễ dàng truy cập vào và nhận mail về. Hiện nay POP3 (Post Office Protocol) và IMAP (Internal Message Access Protocol) là hai giao thức lưu trữ và lấy mail từ hộp thư (Inbox) được nhiều mail server sử dụng. Địa chỉ của máy chủ lưu trữ mail cho phép máy khách truy cập vào để nhận mail về được gọi là *incoming mail address*. Các chương trình mail Client thường sử dụng giao thức POP3 để nhận mail từ trình chủ. Trình chủ tiếp nhận yêu cầu của trình khách theo giao thức POP3 được gọi là POP Server.

SMTP Relay: Các mail server nếu nhận được mail không thuộc địa chỉ do mình quản lý, khi đó nó có trách nhiệm gửi mail đến máy chủ khác với cùng giao thức SMTP. Trong trường hợp này mail server (nơi chuyển tiếp thư đi) đóng vai trò như một trình khác (mail client) đối với máy chủ nơi nhận. Cứ thế. Internet mail là một hệ thống chuyển mail liên tục giữa các máy chủ mail với nhau cho đến khi nào mail đến được máy chủ đích. Các mail server đa phần hỗ trợ tính năng Relay.

POP before SMTP (Chứng thực quyền truy cập theo giao thức POP trước khi sử dụng SMTP): Để tránh tình trạng máy chủ mail được lợi dụng để gửi mail một cách ồ ạt. Cơ chế POP before SMTP yêu cầu máy khách muốn sử dụng dịch vụ mail của máy chủ phải trước hết phải đăng nhập (login) vào tài khoản (account) theo giao thức POP. Nếu quá trình đăng nhập thành công, cơ chế gửi mail bằng SMTP mới có thể thực hiện được.

Mail client, Web mail: là chương trình dùng để nhận mail về từ trình chủ và cho phép gửi mail lên trình chủ. Nếu mail client được viết dưới giao diện Web thì nó được gọi là Web mail.

II. Gửi mail với thư viện có sẵn của java

Java cung cấp cho bạn một thư viện nhỏ gọn giúp bạn có thể gửi mail một cách dễ dàng, tuy nhiên với thư viện này bạn không có được các tính năng cao cấp như gửi file đính kèm..., Để sử dụng các tính năng cao cấp bạn xử thư viện Mail API (giới thiệu trong phần sau). Thư viện này có lớp SmtplibClient dùng để gửi mail.

Để gửi mail bạn cung cấp địa chỉ của người gửi (from), địa chỉ của người nhận (to), địa chỉ của mail server (máy chủ tiếp nhận mail do bạn gửi lên) cùng với nội dung mà bạn muốn gửi. Dưới đây là một chương trình đơn giản dùng để gửi mail đến địa

chỉ thek44a2@yahoo.com nhờ vào máy chủ mail utehy.edu.vn (máy chủ mail của trường DHSP KT Hưng Yên). Địa chỉ người gửi có thể đặt tùy ý. Nhưng thường là địa chỉ của bạn để người nhận sau khi đọc thư xong có thể hồi âm (Reply) lại cho bạn.

//SendMailApp.java

```
import sun.net.smtp.*;
import java.io.*;
public class SendMailApp {
    public static void main(String[] args) throws Exception {
        String host = "utehy.edu.vn";
        String from = "theth@utehy.edu.vn";
        String to = "thek44a2@yahoo.com";
        String subject = "Test Message";
        String body = "Hello this is test message for mail";
        /* Tạo đối tượng gửi mail theo giao thức SMTP, host là địa chỉ của máy chủ nhận mail do ta gửi lên */
        SmtplibClient mailer = new SmtplibClient(host);
        // Thiết lập địa chỉ của người gửi và người nhận
        mailer.from(from);
        mailer.to(to);
        //Lấy về luồng xuất gắn với mail
        PrintStream ps = mailer.startMessage();
        //Điền các thông tin khác
        ps.println("From:" + from);
        ps.println("To:" + to);
        ps.println("Subject:" + subject);
        ps.println();// Thêm CRLF vào trước phần nội dung
        ps.println(body);
        //Kết thúc quá trình gửi mail
        mailer.closeServer();
        System.out.println("Mail has been sent");
    }
}
```

Chú ý: Hãy thay địa chỉ From, To bằng địa chỉ e-mail mà bạn muốn gửi.

III. Giới thiệu java mail API

Lớp SmtplibClient của java mà chúng ta sử dụng trên đây chỉ giúp bạn gửi mail ở mức đơn giản nhất với nội dung mail ở dạng thuần văn bản. Một ứng dụng mail phía máy khách thường có các chức năng phức tạp hơn như : Gửi file đính kèm, dữ liệu được nhúng cả hình ảnh, âm thanh... Để thực hiện những chức năng này java đưa ra một thư viện riêng chuyên xử lý các thao tác gửi nhận mail.

1. Cài đặt thư viện java mail

Mặc định các thư viện java mail API không được cài đặt khi bạn cài JDK. Thư viện này gồm ba phần:

Phần thứ nhất là hạt nhân của các tác vụ xử lý mail, nó được đóng gói trong tệp mail.jar. Bạn có thể download từ địa chỉ <http://java.sun.com/javamail/index.html>

Phần thứ hai là JavaBean Activation Framework cần cho mọi phiên bản của java mail. Thư viện này mang tên activation.jar. với thư viện này bạn có thể tải xuống từ <http://java.sun.com/products/javabean/glasgow/jar.html>
Phần thứ ba là các lớp đọc hay nhận mail từ trình chủ. Trong java 1.4 thư viện này có tên mail.jar

Bạn đặt biến môi trường classpath=%CLASSPATH%; C:\JBuilderX\lib\mail.jar;
C:\JBuilderX\lib\activation.jar;

Chú ý: Khi bạn cài JBuilder hai thư viện này cũng được tự động cài đặt vào máy, bạn có thể tìm thấy nó trong thư mục Jbuilder_Home/lib

2. Các bước gửi nhận mail

Các bước để nhận mail:

Tạo đối tượng mail *Session*.

Tạo đối tượng *store* chuẩn bị lấy mail về. Để tạo đối tượng Store bạn yêu cầu người dùng nhập vào tên và mật khẩu chứng thực quyền truy cập hợp pháp.

Sử dụng đối tượng Store để lấy về đối tượng Folder. Mỗi Folder sẽ chứa một loại mà bạn muốn lấy. Như Folder Inbox chứa các thư viện gửi đến. Folder trash chứa các thư vừa bị xoá. Folder sent chứa các thư vừa gửi đi...

Duyệt đối tượng Folder để lấy về danh sách các thư.

Các bước để gửi mail

Yêu cầu người gửi chọn giao thức để gửi (SMTP) và nhận (POP3 hay IMAP...). Lưu các thông tin này vào đối tượng Session.

Tạo đối tượng Message. Đối tượng này sẽ chứa các thông tin về mail cần gửi đi như địa chỉ người gửi, địa chỉ người nhận, tiêu đề mail, nội dung mail...

Gọi phương thức Transport.send() hoặc phương thức sendMessage của đối tượng Transport để gửi thư đi.

3. Các lớp cơ bản xử lý thao tác gửi nhận mail

Ta tìm hiểu các gói trong thư viện java mail. Hầu hết các ứng dụng mail client cần **import** các thư viện javax.mail, javax.mail.internet, javax.mail.activation.

3.1. Các lớp liên quan đến việc gửi mail

3.1.1. Lớp Session

Lớp này thể hiện một phiên kết nối tới mail server. Trước khi truy cập (gửi/nhận) bạn cần phải tạo ra một đối tượng Session thể hiện cho một phiên làm việc. ví dụ:

```
//Tạo đối tượng Properties chứa thông tin khởi tạo nếu có
```

```
Properties props=new Properties();
```

```
/*gán giá trị cho các thuộc tính bằng phương thức  
props.setProperty(PropertyName, PropertyValue) */
```

```
//Tạo đối tượng Session
```

```
Session session =Session.getDefaultInstance(props,null);
```

Hoặc bạn có thể gọi phương thức getInstance() của lớp Session thay cho getDefaultInstance() để tạo ra thể hiện của đối tượng Session như sau:

```
//Tạo đối tượng Properties chứa thông tin khởi tạo nếu có
```

```
Properties props=new Properties();
```



```
/*gán giá trị cho các thuộc tính bằng phương thức  
props.setProperty(PropertyName, PropertyValue)*/  
//Tạo đối tượng Session
```

```
Session session =Session.getInstance(props,null);
```

Trong cả hai phương thức khởi tạo đối tượng Session nêu trên. các thuộc tính cấu hình được đặt trong đối tượng Properties nên trước khi bạn gửi nhận mail bạn cần thiết đặt các thuộc tính này. ví dụ:

```
props.setProperty("mail.host","utehy.edu.vn");  
props.setProperty("mail.from","theht@utehy.edu.vn");...
```

Ở đây thường thì bạn truyền null cho tham số thứ hai với ý nghĩa sử dụng cơ chế chứng thực (authentication) mặc định của mail server.

Sự khác nhau giữa hai phương thức này là getInstance() thì luôn tạo ra một Session mới còn getDefaultInstance() thì không tạo ra một thể hiện mới của Session nếu như nó đã được tạo ra trong cùng java virtual machine.

3.1.2. Lớp Message

Sau khi có đối tượng Session bạn, để gửi mail bạn cần tạo ra đối tượng Message. Tuy nhiên ít khi bạn làm việc trực tiếp với đối tượng này mà bạn thường làm việc với các lớp con của nó, lớp MimeMessage. ví dụ, bạn tạo thông điệp mail để chuẩn bị gửi đi như sau:

```
MimeMessage message =new MimeMessage(mailsession);
```

Nếu nội dung của thông điệp là thuần văn bản, bạn có thể thiết đặt nội dung thư gửi bằng lệnh:

```
message.setText(content);
```

Nếu bạn chỉ định Mime cho nội dung cần gửi nếu nó không phải là thuần văn bản như sau:

```
message.setContent(content,"text/html");
```

Nếu nội dung không phải thuần văn bản thì bạn phải chỉ định Mime cho nó, để nơi nhận mail sẽ căn cứ vào Mime để hiển thị đúng nội dung của thông điệp nhận được.

Một thói quen tốt nên có là đặt cho mail một tiêu đề (subject). Người nhận căn cứ vào tiêu đề thư để biết được nội dung vắn tắt mô tả trong thư. Phương thức setSubject() sẽ giúp ta làm việc đó. Ví dụ:

```
Session.setSubject("Xin chào");
```

3.1.3. Lớp Address

Sau khi bạn tạo ra đối tượng Session và Message, tiếp đến ta sử dụng lớp Address để tạo ra địa chỉ của người gửi và người nhận. Lớp Address là lớp trừu tượng, bạn sử dụng lớp con của nó là InternetAddress. Ví dụ bạn tạo địa chỉ mail từ lớp này như sau:

```
Address address =new InternetAddresss("theht@utehy.edu.vn");
```

Hoặc

```
InternetAddresss address =new InternetAddresss("theht@utehy.edu.vn");
```

Nếu bạn muốn địa chỉ e-mail có thêm thông tin diễn giải bạn thêm sử dụng cú pháp như sau:

```
Address address =new InternetAddresss("theht@utehy.edu.vn", "Hoang Trong The");
```

Thông tin diễn giải này sẽ được hiển thị thay khi người nhận đọc thư thay cho địa chỉ mail from.

Sau khi tạo xong địa chỉ của người gửi, địa chỉ của người nhận bạn cần gắn nó vào thông điệp cần gửi đi. Java cung cấp cho bạn ba loại địa chỉ phổ biến:

Kiểu địa chỉ	ý nghĩa
Message.RecipientType.TO	Địa chỉ nơi đến
Message.RecipientType.CC	Địa chỉ người gửi cần lưu ý
Message.RecipientType.BCC	Địa chỉ người gửi cần tham khảo

Bạn gọi phương thức addRecipient() của đối tượng Message để gắn địa chỉ cần gửi.

Ví dụ:

```
//Tạo địa chỉ
```

```
Address toAddress=new InternetAddresss("thek44a2@yahoo.com");
```

```
Address ccAddress=new InternetAddresss("theht@walla.com");
```

```
//Gán địa chỉ cho người nhận
```

```
message.addRecipient(Message.RecipientType.TO, toAddress);
```

```
message.addRecipient(Message.RecipientType.CC, ccAddress);
```

Để đặt địa chỉ của người gửi bạn gọi phương thức setFrom(). Ví dụ:

```
Address myAddress=new InternetAddresss("theht@utehy.edu.vn");
```

```
message.setFrom(myAddress);
```

Địa chỉ của người gửi không quan trọng, bạn có thể đặt cho nó một địa chỉ e-mail không có thực, nếu bạn đặt địa chỉ của người gửi không có thực thì người nhận không thể phản hồi (reply) lại cho bạn.

Ta có thể đặt địa chỉ của người gửi là một danh sách (mảng) để người nhận có thể phản hồi lại cho nhiều người cùng một lúc. Bạn gọi phương thức addFrom() như sau:

```
Address address[]=...
```

```
Message.addFrom(address);
```

Chú ý: java mail không cung cấp cơ chế kiểm tra xem địa chỉ của người gửi có hợp lệ hay không. Để kiểm tra một địa chỉ mail có hợp lệ hay không không phải là dễ, thông thường đơn giản nhất bạn kiểm tra xem trong địa chỉ đó có kí tự @ hay không?

3.1.4. Lớp Transport

Lớp Transport cung cấp các phương thức gửi mail đến mail server. Đơn giản bạn gọi phương thức tĩnh send() của lớp Transport để gửi đối tượng thông điệp đi. Ví dụ:

```
Transport.send(message);
```

Hoặc bạn có thể chọn giao thức gửi mail từ đối tượng Session, tạo kết nối đến trình chủ mail và gọi phương thức sendMessage() để gửi mail đi:

```
Transport transport;
```

```
transport =mailSession.getStransport("smtp");
```

```
transport.connect(host, user, password);
```

```
// hoặc bạn có thể gọi phương thức transport.connect();
```

```
transport.sendMessage(message, message.getAllRecipients());
```

```
transport.close();
```

Cách thứ hai này nhanh hơn cách thứ nhất trong trường hợp bạn muốn gửi một số lượng lớn các mail, kết nối vẫn được duy trì cho đến khi bạn gọi phương thức `transport.close()`, trong khi kết nối vẫn còn mở bạn có thể gửi bao nhiêu thư tùy ý mà không phải kết nối lại với mail server. Trong khi phương thức `transport.send()` phải kết nối, đóng kết nối mỗi khi có một thông điệp được gửi đi, do vậy mất thời gian.

3.2. Các lớp liên quan đến việc nhận mail

Lớp Store và lớp Folder: Là hai lớp này phục vụ chủ yếu cho việc nhận và đọc mail chứa trong hộp thư của mail server. Để sử dụng hai lớp này trước tiên bạn cần chỉ định giao thức nhận mail. Có hai giao thức nhận mail chính là POP3 và IMAP.

Ví dụ nếu sử dụng giao thức POP3 thì bạn khai báo gói `com.sun.mail.pop3` như sau:

```
import com.sun.mail.pop3.*;
```

Còn nếu bạn sử dụng giao thức IMAP thì bạn khai báo gói `com.sun.mail.imap` như sau:

```
import com.sun.mail.imap.*;
```

Tương tự như lớp `Transport` trong thao tác gửi mail, bạn dùng đối tượng `Session`, gọi phương thức `getStore()` để lấy về đối tượng `Store` phục vụ cho mục đích kết nối với mail server để nhận mail.

```
Store store = mailSession.getStore("pop3");
```

Để kết nối vào mail server bạn gọi phương thức `connect()` như sau:

```
store.connect(host, user, password);
```

Trong đó:

`host` là địa chỉ của mail server mà bạn muốn lấy thư

`user/password` là tài khoản đăng nhập vào hộp thư của bạn.

Nếu kết nối thành công bạn lấy về đối tượng `Folder` để duyệt mail.

```
Folder folder = store.getFolder("INBOX");
```

```
folder.open(Folder.READ_ONLY);
```

```
Message message[]=folder.getMessages();
```

Các thông điệp mail được lưu trữ trên mail theo một cấu trúc cây, tương tự như cấu trúc cây thư mục của hệ điều hành. INBOX ám chỉ thư mục chứa thư gửi đến.

Chú ý: Với giao thức POP3 bạn chỉ có thể lấy mail từ INBOX với giao thức IMAP (mở rộng của giao thức POP3) bạn có thể nhận mail trong các thư mục TRASH, SENT...

Phương thức `getMessages()` trả về một mảng các đối tượng `Message`. Bạn gọi phương thức `getContent()` để lấy về nội dung của mail như sau:

```
for(int i=0;i<message.lenth();i++)
```

```
System.out.println(((MimeMessage)message[i]).getContent());
```

Một đã đọc xong thư bạn đóng hộp thư và ngắt kết nối:

```
folder.close(true);
```

```
store.close();
```

Đổi số `true` cho biết có cập nhật các lệnh xóa mail hay không (sẽ đề cập trong phần sau).

III. Gửi mail

1. Gửi mail đơn giản

Chương trình SendMailAPI dưới đây sẽ gửi mail đi theo giao thức SMTP, SendMailAPI là chương trình java thuần túy sử dụng thư viện java mail. Nó tương tự chương trình SenMailApp chúng ta đã tìm hiểu trong phần trước. Mặc dù vậy thay vì sử dụng lớp Smtplib bạn sử dụng các lớp Session, Transport của java mail API.

//SendMailAPI.java

```
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;
import java.util.*;
public class SendMailAPI {
    public static void main(String args[]) throws Exception {
        String smtpHost = "utehy.edu.vn";
        int smtpPort = 25;
        String from = "theht@utehy.edu.vn";
        String to = "thek44a2@yahoo.com";
        String subject = "Xin chào";
        String content = "";
        // Create a mail session
        Properties props = new Properties();
        props.put("mail.smtp.host", smtpHost);
        props.put("mail.smtp.port", "" + smtpPort);
        Session session = Session.getDefaultInstance(props, null);
        // Construct the message
        Message msg = new MimeMessage(session);
        msg.setFrom(new InternetAddress(from));
        msg.setRecipient(Message.RecipientType.TO, new InternetAddress(to));
        msg.setSubject(subject);
        msg.setText(content);
        // Send the message
        Transport.send(msg);
    }
}
```

Trước tiên bạn tạo ra một đối tượng Properties để chứa các thông tin: tên máy chủ mail, cổng của mail server.. bằng câu lệnh

```
Properties props = new Properties();
props.put("mail.smtp.host", smtpHost);
props.put("mail.smtp.port", "" + smtpPort);
```

mặc định cổng của mail sẽ là 25, nếu mail hoạt động trên một cổng không phải là 25 thì bạn phải chỉ rõ cổng của mail server bằng dòng lệnh:

```
props.put("mail.smtp.port", "" + smtpPort);
```

Tiếp theo bạn lấy về một phiên kết nối bằng dòng lệnh:

```
Session session = Session.getDefaultInstance(props, null);
```

Tiếp theo bạn tạo ra đối tượng thông điệp và điền các thông tin cần thiết vào:

```
Message msg = new MimeMessage(session);
msg.setFrom(new InternetAddress(from));
msg.setRecipient(Message.RecipientType.TO, new InternetAddress(to));
msg.setSubject(subject);
msg.setText(content);
```

Cuối cùng để gửi thư đi bạn gọi phương thức:

```
Transport.send(msg);
```

Bạn có thể thay dòng lệnh này bằng đoạn lệnh sau:

```
Transport tp= session.getTransport("smtp");
tp.connect();
tp.sendMessage(msg,msg.getAllRecipients());
tp.close();
```

Nếu bạn chưa gán địa chỉ mail server và cổng của nó thì bạn phải gọi phương thức:

```
tp.connect(host,user,password);
```

Do bạn đang gửi mail nên các thông tin đăng nhập *user,password* không cần thiết, bạn có thể cung cấp cho nó một chuỗi rỗng như sau:

```
tp.connect(host,"","");
```

2. Gửi thư với nội dung có định dạng HTML

Nếu nội dung thư là thuần văn bản bạn gọi phương thức

Message.setText(StringMsg) để gán nội dung cho đối tượng thông điệp cần gửi đi. Nếu nội dung thư không phải là dạng thuần văn bản bạn cần gọi phương thức

Message.setContent(StringMsg, MimeType) với *MimeType* là một chuỗi cho biết kiểu nội dung được chứa trong mail.

Ví dụ sau minh họa cách gửi thư với nội dung có định dạng HTML:

// SendMailAPIFormat.java

```
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;
import java.util.*;
public class SendMailAPIFormat{
    public static void main(String args[]) throws Exception {
        String smtpHost = "utehy.edu.vn";
        int smtpPort = 25;
        String from = "theht@utehy.edu.vn";
        String to = "thek44a2@yahoo.com";
        String subject = "Xin chao";
        //Gán nội dung là một chuỗi có định dạng HTML
        String content = "<p><b><font face=\\\".\\nArialH\\\" size=\\\"8\\\">Xin chào các
        bạn</font></b></p><p><font face=\\\".\\nTimeH\\\">Chúc một ngày vui vẻ</font></p>";
        // Create a mail session
        Properties props = new Properties();
        props.put("mail.smtp.host", smtpHost);
        props.put("mail.smtp.port", "" + smtpPort);
        Session session = Session.getDefaultInstance(props, null);
        // Construct the message
```

```

Message msg = new MimeMessage(session);
msg.setFrom(new InternetAddress(from));
msg.setRecipient(Message.RecipientType.TO, new InternetAddress(to));
msg.setSubject(subject);
    //Gán kiểu nội dung MIMEType là text/html
msg.setContent(content, "text/html");
    // Send the message
Transport tp = session.getTransport("smtp");
tp.connect("utehy.edu.vn", "", "");
tp.sendMessage(msg, msg.getAllRecipients());
tp.close();
}
}

```

Trong ví dụ trên bạn gán nội dung là một chuỗi có định dạng HTML, để báo cho người nhận biết nội dung bên trong thư có định dạng HTML bạn gọi phương thức *msg.setContent(content, "text/html");*

3. Gửi thư với file đính kèm

Thư viện java mail API cho phép bạn gửi mail với các file đính kèm. Những file này thường được nhúng vào phần nội dung mail gửi đi ở dạng mã hoá từng phân đoạn (Multi – part MIME). Một Message được xem là gồm nhiều phân đoạn. Mỗi phân đoạn được gọi là một BodyPart, Tất cả các BodyPart được gọi là MultiPart. Dựa vào hai lớp BodyPart và MultiPart ta có thể tạo ra một mail phức tạp có thể phân tích bởi nơi nhận. Chương trình SendMailAttachAPI.java dưới đây sẽ cho thấy kỹ thuật gửi mail với file đính kèm.

//SendMailAttachAPI.java

```

import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;
import java.util.*;
public class SendMailAttachAPI {
    public static void main(String args[]) throws Exception {
        String smtpHost = "utehy.edu.vn";
        int smtpPort = 25;
        String from = "theht@utehy.edu.vn";
        String to = "thek44a2@yahoo.com";
        String subject = "Xin chao";
        String content = "Chuc mot buoi sang tot lanh";
        //Tên của file đính kèm
        String fileName = "C:/test.txt";
            //tạo đối tượng session
        Properties props = new Properties();
        props.put("mail.smtp.host", smtpHost);
        props.put("mail.smtp.port", "" + smtpPort);
        Session session = Session.getDefaultInstance(props, null);
            //Tạo đối tượng thông điệp

```

```

Message msg = new MimeMessage(session);
msg.setFrom(new InternetAddress(from));
msg.setRecipient(Message.RecipientType.TO, new InternetAddress(to));
msg.setSubject(subject);
/* Tạo đối tượng Multipart, đối tượng này dùng để chứa các BodyPart */
Multipart multiPart = new MimeMultipart();
    //Tạo ra đối tượng BodyPart để chứa nội dung mail
BodyPart msgBodyPart = new MimeBodyPart();
msgBodyPart.setText(content);
    //Thêm đối tượng msgBodyPart vào multiPart
multiPart.addBodyPart(msgBodyPart);
/*Tạo đối tượng BodyPart thứ hai, đối tượng này dùng để chứa phần file đính kèm */
msgBodyPart = new MimeBodyPart();
DataSource source = new FileDataSource(fileName);
msgBodyPart.setDataHandler(new DataHandler(source));
    //Gán tên tệp tin cho BodyPart
msgBodyPart.setFileName(fileName);
    //Cho biết đây là phần chứa file đính kèm
msgBodyPart.setDisposition("attachment");
    //Thêm đối tượng msgBodyPart vào multiPart
multiPart.addBodyPart(msgBodyPart);
    //Gán multiPart cho đối tượng thông điệp
msg.setContent(multiPart);
    //Gửi thông điệp đi
Transport tp = session.getTransport("smtp");
tp.connect("utehy.edu.vn", "", "");
tp.sendMessage(msg, msg.getAllRecipients());
tp.close();
}
}

```

Chú ý: Bạn có thể đính kèm bao nhiêu file tùy ý, bằng cách tạo ra nhiều BodyPart khác nhau.

IV. Nhận mail

1. Nhận mail đơn giản

Để nhận mail, bạn kết nối vào mail server để lấy về các Message trong thư mục INBOX. Ví dụ GetMailAPI sau minh họa cách nhận mail từ trình chủ và hiển thị ra màn hình.

//GetMailAPI.java

```

import java.util.*;
import javax.mail.*;
import com.sun.mail.pop3.*;
public class GetMailAPI {
    public static void main(String[] args) throws Exception {
        String smtpHost = "utehy.edu.vn";
        int smtpPort = 25;
        //Tạo đối tượng Session

```

```

Properties props = new Properties();
props.put("mail.smtp.host", smtpHost);
props.put("mail.smtp.port", "" + smtpPort);
Session session = Session.getDefaultInstance(props, null);
    //Tạo đối tượng Store
Store store = session.getStore("pop3");
store.connect(smtpHost, "thek44a2", "abc");
    //Lấy về thư mục INBOX
Folder folder = store.getFolder("INBOX");
if (folder == null) {
    System.out.println("No folder");
    System.exit(1);
}
    //Mở thư mục INBOX với chế độ chỉ đọc
folder.open(Folder.READ_ONLY);
    //Lấy về các đối tượng Message và hiển thị ra màn hình
Message[] messages = folder.getMessages();
for (int i = 0; i < messages.length; i++) {
    printMessage(messages[i]);
}
/*Đóng kết nối nhưng không xoá các Message từ mail server */
folder.close(false);
store.close();
}
public static void printMessage(Message m) throws Exception {
    //In phần tiêu đề thư
    //Lấy về các địa chỉ gửi
Address[] from = m.getFrom();
if (from != null) {
    for (int i = 0; i < from.length; i++) {
        System.out.println("From: " + from[i]);
    }
}
    //Lấy về các địa chỉ nhận TO
Address[] to = m.getRecipients(Message.RecipientType.TO);
if (to != null) {
    for (int i = 0; i < to.length; i++) {
        System.out.println("To: " + to[i]);
    }
}
    //Lấy về tiêu đề thư
String subject = m.getSubject();
if (subject != null) {
    System.out.println("Subject: " + subject);
}
    //Lấy về ngày thư được gửi
Date d = m.getSentDate();
if (d != null) {
    System.out.println("Date: " + d);
}

```



```

}
//Thêm một hàng xuống dòng để phân cách phần tiêu đề và phần chính của thư
System.out.println();
    //In nội dung của thư
    Object content = m.getContent();
    if (content instanceof String) {
        System.out.println(content);
    }
    else if (content instanceof InputStream) {
        InputStream in = (InputStream) content;
        int c;
        while ( (c = in.read()) != -1) System.out.write(c);
    }
    else {
        //Định dạng này sẽ được giới thiệu trong phần sau
        System.out.println("Unrecognized Content Type");
    }
}
}
}
}

```

Chú ý: Bạn có thể không cần dùng đối tượng Store, bạn có thể thay đoạn chương trình sau:

```

Store store = session.getStore("pop3");
store.connect(smtpHost, "thek44a2", "abc");
    //Lấy về thư mục INBOX

```

```

Folder folder = store.getFolder("INBOX");

```

Bởi đoạn chương trình

```

URLName urlName=new
URLName("pop3","utehy.edu.vn",110,"INBOX","thek44a2","abc");
Folder folder = session.getFolder(urlName);

```

Bằng cách này bạn không cần phải tạo ra đối tượng Store

2. Lấy về các thông tin đi kèm với mail

Bạn có thể lấy về các thông tin đi kèm với thư như: ngày gửi, tiêu đề... phương thức printInfo sau sẽ in ra màn hình các thông tin đi kèm với thư.

```

public static void printInfo(Message m)throws Exception
{
    //Lấy về địa chỉ của người gửi
    String from = InternetAddress.toString(m.getFrom());
    if (from != null) System.out.println("From: " + from);
        //Hoặc
    Address[] from = m.getFrom();
    if (from != null) {
        for (int i = 0; i < from.length; i++) {
            System.out.println("From: " + from[i]);
        }
    }

    //Lấy về các địa chỉ đã phản hồi
    String replyTo = InternetAddress.toString(m.getReplyTo());

```

```

if (replyTo != null)
    System.out.println("Reply-to: " + replyTo);
    //Hoặc
    Address[] reply = m.getReplyTo();
    if (reply != null) {
        for (int i = 0; i < reply.length; i++) {
            System.out.println("Reply: " + reply[i]);
        }
    }
    //Lấy về các địa chỉ TO
String to = InternetAddress.toString(m.getRecipients(Message.RecipientType.TO));
if (to != null) System.out.println("To: " + to);
    //Lấy về các địa chỉ CC
String cc = InternetAddress.toString(m.getRecipients(Message.RecipientType.CC));
if (cc != null) System.out.println("Cc: " + cc);
    //Lấy về các địa chỉ BCC
String bcc = InternetAddress.toString(m.getRecipients(Message.RecipientType.BCC));
if (bcc != null) System.out.println("Bcc: " + to);
    //Lấy về tiêu đề của thư
String subject = m.getSubject();
if (subject != null) System.out.println("Subject: " + subject);
    //Lấy về ngày thư được gửi
Date sent = m.getSentDate();
if (sent != null) System.out.println("Sent: " + sent);
    //Lấy về ngày thư được nhận
Date received = m.getReceivedDate();
if (received != null) System.out.println("Received: " + received);
}

```

3. Nhận mail với các file đính kèm

Nếu nội dung thư có chứa file đính kèm bạn có thể tách phần đính kèm, phần thư ra. Ví dụ GetMailAttach.java minh họa cách kiểm tra mail có file đính kèm hay không?, nếu có thì ghi nội dung của file đính kèm xuống thư mục C:\data.

//GetMailAttach.java

```

import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;
import com.sun.mail.pop3.*;
import java.net.*;
import java.io.*;
public class GetMailAttach {
    public static void main(String[] args) throws Exception {
        String smtpHost = "utehy.edu.vn";
        int smtpPort = 25;
        //tạo ra một mail Session
        Properties props = new Properties();
        props.put("mail.smtp.host", smtpHost);
        props.put("mail.smtp.port", "" + smtpPort);
    }
}

```

```

Session session = Session.getDefaultInstance(props, null);
    //Lấy về Store
    Store store = session.getStore("pop3");
    //Kết nối với mail server
    store.connect(smtpHost, "thek44a2", "abc");
    //Lấy về thư mục INBOX
    Folder folder = store.getFolder("INBOX");
    //Nếu không có thư mục này thì thoát
    if (folder == null) {
        System.out.println("No folder");
        System.exit(1);
    } //Mở thư mục để lấy thư theo chế độ chỉ đọc
    folder.open(Folder.READ_ONLY);
    //Lấy về các thư trong Mail Box
    Message[] messages = folder.getMessages();
    //Xử lý từng thư một
    for (int i = 0; i < messages.length; i++) {
        //Lấy về nội dung của thư
        Object body = messages[i].getContent();
        //Kiểm tra xem nội dung thư có nhiều phần hay không?
        if (body instanceof Multipart) {
            //Nội dung thư gồm nhiều phần
            processMultipart( (Multipart) body);
        }
        else { //Nội dung thư chỉ có một phần
            processPart(messages[i]);
        }
    }
    // Đóng kết nối, nhưng không xoá các thư
    folder.close(false);
    store.close();
}

//Phương thức xử lý thư gồm nhiều Part
public static void processMultipart(Multipart mp) throws MessagingException {
    for (int i = 0; i < mp.getCount(); i++) {
        processPart(mp.getBodyPart(i));
    }
}

//Phương thức xử lý một Part
public static void processPart(Part p) {
    try {
        //Thử lấy về tên file của Part
        String fileName = p.getFileName();
        /* ấy về dấu hiệu cho biết Part có phải là phần file đính kèm không */
        String disposition = p.getDisposition();
        /* Kiểm tra xem Part có phải là file đính kèm không?
        Chú ý không nên sử dụng disposition.equals(Part.ATTACHMENT), vì disposition có thể
        nhận giá trị null khi đó phương thức equals sẽ nhận một ngoại lệ
        */

```


V. Xoá mail

1. Xoá mail

Để xoá mail bạn mở hộp thư với quyền đọc/ghi như sau:

```
Folder folder = store.getFolder("INBOX");  
folder.open(Folder.READ_ONLY);  
Message[] messages = folder.getMessages();
```

Tiếp đến đặt cờ xoá:

```
Messages[i].setFlag(Flags.Flag.DELETE,true);
```

Tiếp đến đóng kết nối, mail server sẽ tự động xoá mail
folder.close(true); //Chú ý bắt buộc phải là tham số true

2. Kiểm tra trạng thái cờ của mail

// Kiểm tra trạng thái của các cờ :

//Đã đánh dấu xoá

```
if (messages[i].isSet(Flags.Flag.DELETED)) {  
    System.out.println("Deleted");  
}  
    //Đã trả lời  
if (messages[i].isSet(Flags.Flag.ANSWERED)) {  
    System.out.println("Answered");  
}  
    //Đã vào thùng rác  
if (messages[i].isSet(Flags.Flag.DRAFT)) {  
    System.out.println("Draft");  
}  
//Đã được gán cờ  
if (messages[i].isSet(Flags.Flag.FLAGGED)) {  
    System.out.println("Marked");  
}  
if (messages[i].isSet(Flags.Flag.RECENT)) {  
    System.out.println("Recent");  
}  
    //Đã đọc  
if (messages[i].isSet(Flags.Flag.SEEN)) {  
    System.out.println("Read");  
}  
if (messages[i].isSet(Flags.Flag.USER)) {  
    // We don't know what the user flags might be in advance  
    // so they're returned as an array of strings  
    String[] userFlags = messages[i].getFlags().getUserFlags();  
    for (int j = 0; j < userFlags.length; j++) {  
        System.out.println("User flag: " + userFlags[j]);  
    }  
}
```

Chương 10: LẬP TRÌNH PHÂN TÁN VỚI JAVA RMI (REMOTE METHOD INVOKE)

Lập trình phân tán với RMI là một trong những vấn đề nóng bỏng của nền công nghiệp phần mềm ngày nay. Để giúp các bạn có các kiến thức cơ bản về lập trình phân tán và công nghệ lập trình phân tán RMI. Chương này ta thảo luận về các khía cạnh đó.

Sau khi học xong chương này người học có thể:

Hiểu được khái niệm về lập trình phân tán và công nghệ RMI

ứng dụng công nghệ RMI để xây dựng các ứng dụng phân tán

Chuyên tham số cho phương thức triệu gọi từ xa và nhận kết quả trả về từ phương thức triệu gọi từ xa

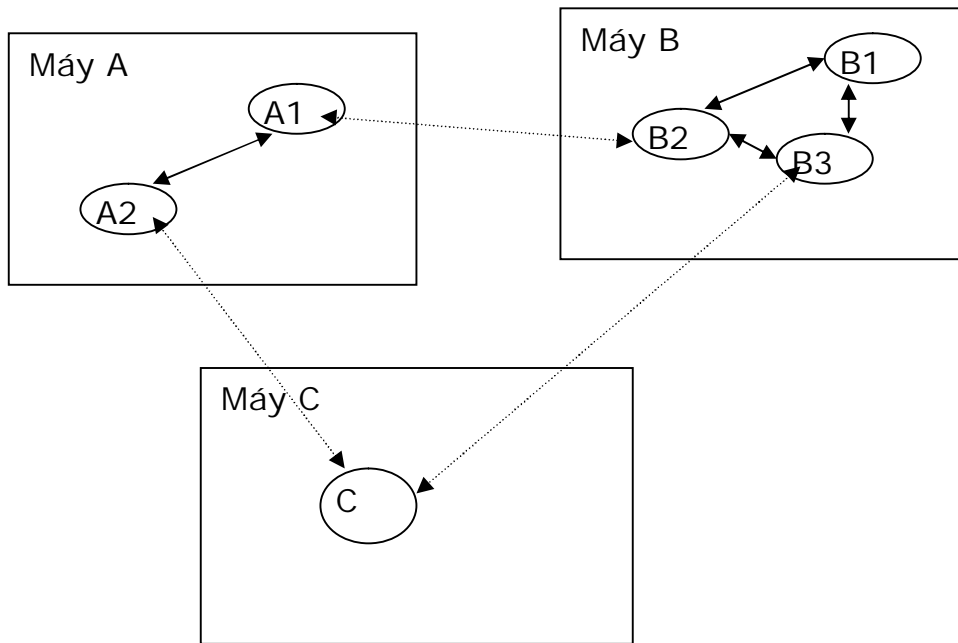
I. RMI và lập trình phân tán đối tượng

Thông thường mã lệnh của một chương trình khi thực thi được tập trung trên cùng một máy, đây là cách lập trình truyền thống. Sự phát triển như vũ bão của mạng máy tính đặc biệt là mạng Internet toàn cầu, đã khiến các chương trình truyền thống này không còn đáp ứng được yêu cầu nữa. Các chương trình bây giờ yêu cầu phải có sự hợp tác xử lý, tức là mã lệnh của nó đã không tập trung trên một máy mà được phân tán trên nhiều máy. Khi một ứng dụng có mã lệnh thực thi được phân tán trên nhiều máy thì chương trình đó được gọi là chương trình phân tán và việc lập trình để tạo ra các chương trình này được gọi là lập trình phân tán. Có rất nhiều công nghệ lập trình phân tán như: DCOM, CORBA, RMI, EJB.. trong đó RMI là công nghệ thuần Java và dễ lập trình nhất.

Thông thường nếu các đối tượng được tập trung trên cùng một máy thì bạn có thể triệu gọi các phương thức của nó bằng cách gửi cho nó một thông báo, câu hỏi đặt ra là làm thế nào để có thể triệu gọi các phương thức của một đối tượng nằm trên một máy khác. đây chính là nội dung của lập trình phân tán mã lệnh RMI (Remote Method Invoke – tạm dịch là triệu gọi phương thức từ xa). RMI là cách thức giao tiếp giữa các đối tượng Java có mã lệnh cài đặt (bao gồm cả phương thức và thuộc tính) nằm trên các máy khác nhau có thể triệu gọi lẫn nhau.

Hình sau mô hình triệu gọi đối tượng phân tán. Trên máy A các đối tượng A1, A2 gọi các phương thức của nhau được gọi là triệu gọi phương thức cục bộ (local method invoke) đây là cách lập trình hướng đối tượng truyền thống vẫn sử dụng, tương tự các đối tượng B1, B2, B3 là các đối tượng cục bộ.

Tuy nhiên các đối tượng Java có thể triệu gọi phương thức của một đối tượng nằm trên một máy khác dựa vào giao thức triệu gọi từ xa RMI. Trong mô hình dưới đây thì lời triệu gọi phương thức của đối tượng B2 (nằm trên máy B) từ đối tượng A1 (nằm trên máy A) là lời gọi phương thức từ xa.



Hình 1: Mô hình triệu gọi các phương thức từ xa

II. Gọi phương thức từ xa và các vấn đề phát sinh

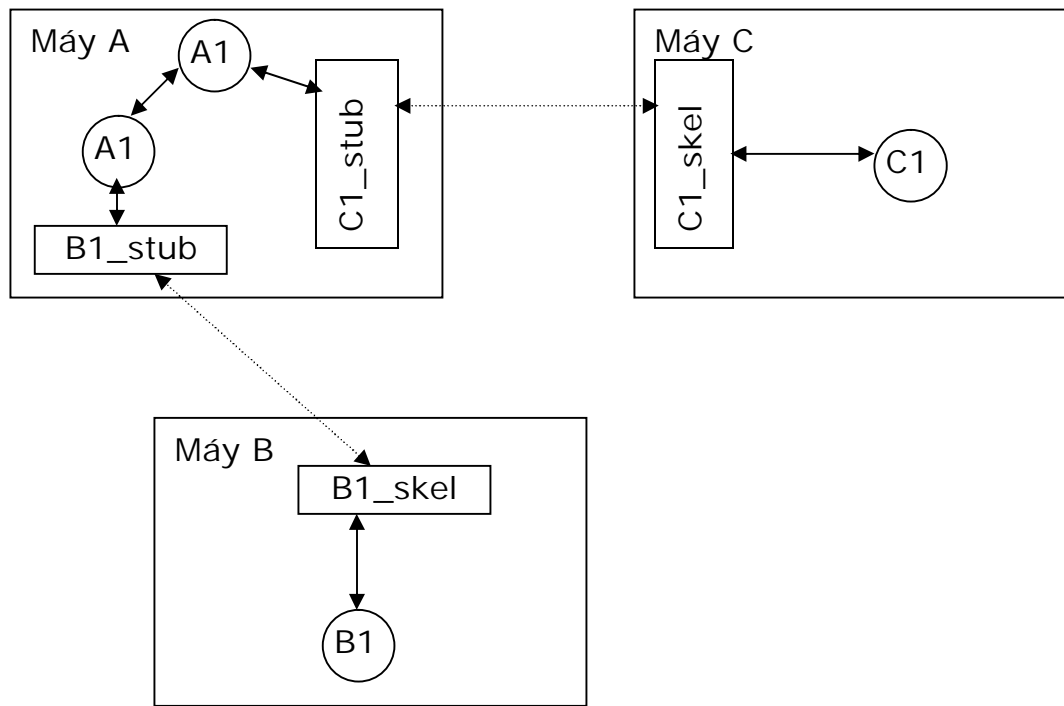
Việc triệu gọi một phương thức từ xa thoạt nhìn có vẻ đơn giản nhưng thực tế lại phức tạp hơn triệu gọi phương thức cục bộ. Các đối tượng trên hai máy khác nhau hoạt động trên hai tiến trình khác nhau (có hai không gian địa chỉ khác nhau) nên việc tham chiếu biến địa chỉ đối tượng là khác nhau. Ví dụ khi bạn truyền một đối tượng cho một phương thức triệu gọi từ xa thì thực sự bạn truyền một tham chiếu đối tượng đến phương thức từ xa, tuy nhiên vùng nhớ thực sự của đối tượng lại nằm trên một máy khác. Lời gọi phương thức cục bộ luôn trả về kết quả thông qua ngăn xếp trong khi lời gọi phương thức từ xa kết quả trả về phải thông qua kết nối mạng chính vì vậy các sự cố về truyền thông luôn có thể xảy ra, như vậy việc bắt và kiểm soát lỗi trong các ứng dụng phân tán là rất đắt.

III. Vai trò của các lớp trung gian

Đối với lời gọi phương thức cục bộ, các tham số truyền vào phương thức cũng như kết quả trả về từ phương thức được thực hiện thông qua ngăn xếp (stack) trong khi lời gọi phương thức từ xa phải được đóng gói và chuyển qua mạng.

Để đơn giản và trong suốt đối với người lập trình đối tượng Java trên hai máy khác nhau không triệu gọi phương thức của nhau một cách trực tiếp mà thông qua lớp trung gian. Lớp trung gian tồn tại ở cả hai phía: phía máy khách (nơi gọi phương thức của đối tượng ở xa) và máy chủ (nơi đối tượng thực sự được cài đặt). Phía máy khác lớp trung gian này được gọi là stub (lớp móc), phía máy chủ lớp trung gian được gọi là skeleton (lớp nối).

Ta có thể hình dung lớp trung gian stub và skel là hai người trung gian giúp các đối tượng ở xa có thể giao dịch được với nhau.



Hình 2: Gọi phương thức của đối tượng thông qua lớp trung gian

Trong hình trên bạn có đối tượng C1 được cài đặt trên máy C. trình biên dịch Java giúp ta tạo ra hai lớp trung gian C1_stub và C1_skel. Lớp C1_stub được mang về máy A. Khi A1 trên máy A triệu gọi phương thức của đối tượng C1 nó sẽ chuyển lời gọi phương thức cho lớp trung gian C1_stub. Lớp trung gian C1_stub có trách nhiệm đóng gói các tham số chuyển tham số qua mạng đến phương thức được triệu gọi của đối tượng C1. Trên máy C lớp trung gian C1_Skel có nhiệm vụ nhận các tham số và chuyển vào vùng địa chỉ thích hợp sau đó gọi phương thức tương ứng. Kết quả trả về (nếu có) của phương thức do C1 trả về sẽ được lớp C1_Skel đóng gói và chuyển ngược về cho trình khách. Trên máy khách lớp trung gian C1_Stub chuyển giao kết quả cuối cùng lại cho A1. Bằng cơ chế này A1 luôn nghĩ rằng đối tượng C1 đang tồn tại ngay trên cùng máy với nó như các đối tượng cục bộ khác. Hơn nữa nhờ có lớp trung gian C1_Stub mà khi kết nối mạng gặp sự cố thì lớp trung gian stub sẽ luôn biết cách thông báo lỗi đến đối tượng A1.

Thực tế làm cách nào A1 tham chiếu được đến C1, một khi không có đối tượng C1 được cài đặt trên máy A? C1_Stub trên máy A chỉ thực hiện việc chuyển tham số, nhận kết quả trả về nếu có và thực hiện các giao thức mạng, nó không phải là hình ảnh của C1. Để làm được điều này đối tượng C1 cần cung cấp một giao diện tương ứng với các phương thức mà các đối tượng trên máy khác có thể triệu gọi. Chúng ta tiếp tục làm sáng tỏ điều này khi nói về giao diện từ xa.

IV. Cài đặt đối tượng phân tán

1. Giao diện từ xa

Khi bạn muốn tạo ra một sự vật từ xa thì bạn che mặt nạ cài đặt nền bằng cách truyền qua một giao diện. Vậy khi khách thu được một tham chiếu đến đối tượng từ xa thì thực chất đó là một giao diện.

Khi tạo ra một giao diện từ xa, thì bạn phải tuân theo các hướng dẫn sau:
 Giao diện từ xa phải là một giao diện public, tức là khi tạo ra một giao diện từ xa ta phải thêm từ khoá **public** vào trước định nghĩa giao diện. Bằng không, khi bạn tham chiếu đến đối tượng từ xa bạn sẽ thu được một ngoại lệ
 Giao diện từ xa phải là giao diện được kế thừa từ giao diện Remote
 Mỗi phương pháp trong giao diện từ xa phải khai báo RemoteException trong mệnh đề **throws** bên bất cứ ngoại lệ nào khác, tức là tất cả các phương thức trong giao diện từ xa đều phải ném ra ngoại lệ RemoteException
 Nếu tham số truyền cho phương thức hoặc giá trị nhận về từ thức triệu gọi từ xa là một đối tượng thì đối tượng đó phải triển khai giao diện **Remote** hoặc giao diện **Serializable**
 Thông thường bạn thường thấy một giao diện từ xa có cấu trúc như sau:

```
import java.rmi.*;
public interface RemoteInterface extends Remote{
    [public] ReturnDataType method1([DataType arg1],[ DataType arg2,...] ) throws
    RemoteException;
    [public] ReturnDataType method2() throws RemoteException;
}
```

Ví dụ 1-1: Sau đây là một giao diện từ xa đơn giản của ứng dụng HelloRMI

HelloRMI.java

```
import java.rmi.*;
public interface HelloRMI extends Remote{
    public String sayHello() throws RemoteException;
}
```

Nhìn vào giao diện này ta thấy nó giống bất kỳ giao diện nào khác ngoại trừ nó được mở rộng từ giao diện Remote và tất cả các phương thức trong giao diện này đều phải ném ra ngoại lệ *RemoteException*, bạn hãy nhớ rằng tất cả các phương thức được khai báo trong giao diện sẽ tự động là public, thế nên trong giao diện trên bạn có thể bỏ từ khoá **public** khi khai báo phương thức sayHello.

Bạn tiến hành biên dịch javac HelloRMI.java bạn sẽ thu được tệp tin HelloRMI.class

2. Triển khai giao diện từ xa

Sau khi bạn tạo ra giao diện từ xa, công việc tiếp theo mà bạn cần phải làm là triển khai tất cả các phương thức trong giao diện từ xa.

Ví dụ 1-2: Sau đây là cài đặt của giao diện từ xa HelloRMI

HelloRMIImpl.java

```
import java.rmi.*;
public class HelloRMIImpl implements HelloRMI {
    public String sayHello() throws RemoteException
    {
        return "Hello RMI";
    }
}
```

Vấn đề đặt ra là làm thế nào để cài đặt đối tượng HelloRMI lên một máy (máy 2) và gọi phương thức sayHello() của HelloRMI từ máy khác (máy 1)? Như đã nêu ở trên ta không gọi được phương thức sayHello của HelloRMI một cách trực tiếp mà cần có thêm hai lớp trung gian là HelloRMIImpl_Stub và HelloRMIImpl_Skel. Dựa vào lớp

HelloRMIIImpl.class, trình biên dịch *rmic.exe* của Java sẽ giúp ta tạo ra hai lớp trung gian Stub và Skel. Bạn mở cửa sổ DOS – Prompt và gõ vào dòng lệnh

Rmic HelloRMIIImpl.class

Kết quả bạn sẽ thu được hai tệp ***HelloRMIIImpl_Stub.class*** và ***HelloRMIIImpl_Skel.class***

3. Cài đặt, đăng kí đối tượng từ xa

Bước tiếp theo sau khi bạn cài đặt giao diện từ xa là công việc đăng ký nó với trình *rmiregistry* theo mẫu sau:

Ví dụ 1-4: Ví dụ sau là chương trình đăng ký và cài đặt đối tượng HelloRMIIImpl với trình chủ *rmiregistry*.

Setup.java

```
import java.rmi.server.*;
import java.rmi.*;
import java.net.*;
public class Setup {
    public static void main(String[] args) {
// tạo ra một thể hiện của đối tượng từ xa
        HelloRMI h=new HelloRMIIImpl();
        try {
// Khai báo đối tượng có khả năng triệu gọi từ xa
            UnicastRemoteObject.exportObject(h);
// đăng ký nó với trình rmiregistry
            Naming.bind("//localhost/hello", h);
// có thể thay phương thức bind bởi phương thức rebind như sau:
// Naming.rebind("//localhost/hello", h);
        }
        catch (MalformedURLException ex) {
        }
        catch (AlreadyBoundException ex) {
        }
        catch (RemoteException ex) {
        }
    }
}
```

Công việc đầu tiên bạn phải làm khi cài đặt đối tượng từ xa là tạo ra một thể hiện của đối tượng từ xa.

HelloRMI h=new HelloRMIIImpl(); bạn có thể thay bằng dòng lệnh

HelloRMIIImpl h=new HelloRMIIImpl();

Tiếp theo bạn gọi phương thức tĩnh *exportObject* của lớp *UnicastRemoteObject* để máy ảo java biết đối tượng *h* (HelloRMI) là đối tượng có khả năng truy xuất từ xa.

UnicastRemoteObject.exportObject(h);

Lưu ý để sử dụng được lớp ***UnicastRemoteObject*** bạn phải khai báo ***import java.rmi.server.*;*** ở đầu chương trình

Bước cuối cùng là bạn đặt cho đối tượng h một cái tên gọi nhớ và đăng ký tên này với bộ đăng ký rmiregistry. Phương thức tĩnh của lớp Naming sẽ thực hiện điều này

Naming.bind("[rmi://hostname[:port]/name", h);

Phương thức bind có hai tham số: tham số thứ nhất là một chuỗi định vị URL, đối số thứ hai là bản thân đối tượng cần đăng ký

Chuỗi định vị URL có định dạng như sau:

[rmi://hostname[:port]/name

Trong đó:

rmi là tên giao thức, đây là phần tùy chọn, mặc định giao thức là rmi nên bạn có thể bỏ qua.

Hostname là tên của máy chủ hoặc địa chỉ IP của máy chủ nơi đối tượng phân tán đang tồn tại.

Port là số hiệu cổng của chương trình rmiregistry, đây là tham số tùy chọn, nếu bỏ qua tham số này thì cổng mặc định là 1009

Name là tên gọi nhớ của đối tượng phân tán

Phương thức bind sẽ đi vào vòng lặp vô hạn chờ kết nối từ máy khách.

Các chương trình phía máy khách sẽ dựa vào chuỗi định vị URL mà ta đăng ký với trình rmiregistry để truy tìm tham chiếu đến đối cần dùng.

Bạn có thể khởi động bộ đăng ký rmiregistry này ở đâu? Bạn mở cửa sổ DOS-Prompt và cho chạy chương trình rmiregistry này từ dòng lệnh như sau:

C:\JDK1.4\bin\rmiregistry.exe

Nếu trong WintNT thì bạn có thể chạy nó như một dịch vụ bằng dòng lệnh ***start***

C:\JDK1.4\bin\rmiregistry.exe

Sau khi khởi động chương trình rmiregistry bằng một trong hai cách bạn không thấy phản hồi gì cả bởi vì đây là chương trình chạy dưới dạng dịch vụ (service).

Mặc định rmiregistry lắng nghe các kết nối trên cổng 1009. Bạn có thể chỉ định cổng khác cho nó. Ví dụ lệnh sau sẽ khởi động rmiregistry trên cổng 2004

C:\jdk1.4\bin\rmiregistry.exe 2004

Chú ý:

Trong java 1 hostname không thể là localhost, nghĩa là nếu bạn muốn thử nghiệm trên máy cục bộ thì bạn phải truyền cho nó một địa chỉ IP chẳng hạn như 127.0.0.1 hoặc tên của máy (để biết tên của máy của mình bạn vào Control Panel\Net Work chọn thẻ Identification và bạn sẽ thấy tên máy của mình trong phần Computer Name)

RMI sẽ không làm việc chừng nào bạn chưa cài họ giao thức TCP/IP.

Nếu bạn đang kiểm thử trên máy cục bộ thì bạn có thể đăng ký với trình rmiregistry đơn giản như sau:

Naming.bind("hello",h);

Nếu bạn đăng ký một đối tượng mới với một cái tên gọi nhớ đã được sử dụng cho một đối tượng khác bạn sẽ nhận được một ngoại lệ ***AlreadyBoundException***. Để ngăn cản điều này bạn nên dùng phương thức Naming.rebind() thay cho phương thức Naming.bind() vì Naming.rebind() bổ sung một cái tên mới nếu nó chưa có và thay thế một cái tên đã tồn tại với một đối tượng mới.

Cho dù phương thức main() kết thúc thì đối tượng từ xa bạn tạo ra và đăng ký cho nó một cái tên vẫn còn đó chừng nào mà rmiregistry vẫn còn đang chạy và bạn không gọi

phương thức Naming.unbind() thì sự vật này vẫn còn đó. Bởi lý do này, khi bạn phát triển ứng dụng của mình thì bạn cần phải tắt rmiregistry và cho chạy lại nó mỗi khi bạn biên dịch lại đối tượng từ xa.

Bạn không nhất thiết phải khởi động rmiregistry như một tiến trình ngoài. Nếu bạn biết rằng ứng dụng của mình là ứng dụng duy nhất sẽ dùng số đăng ký. Thì bạn có thể bắt đầu nó ở bên trong chương trình của mình bằng dòng lệnh:

```
LocateRegistry.createRegistry(Port);
```

ở đây Port là số hiệu cổng. Điều này là tương đương với việc chạy rmiregistry Port từ dấu nhắc của DOS-Prompt

4. Viết trình khách triệu gọi phương thức của đối tượng cài đặt từ xa

Trình khách triệu gọi đối tượng phân tán RMI có thể là một trong các kiểu ứng dụng sau:

- ứng dụng console
- ứng dụng có giao diện đồ hoạ
- ứng dụng Servlet
- trang JSP

...

Điều duy nhất bạn phải làm trong chương trình khách là tra cứu và lấy về giao diện từ xa từ trình chủ. Từ đó trở đi việc triệu gọi phương thức từ xa giống như là triệu gọi phương thức của bất kỳ đối tượng cục bộ nào khác.

Sau đây là mẫu chương trình phía trình khách triệu gọi đối tượng phân tán RMI

```
import java.rmi.*;
import java.rmi.registry*;
public Client
{
    public static void main(String args[]) throws RemoteException
    {
        // lấy về tham chiếu đối tượng phân tán bằng phương thức Naming.lookup()
        RemoteInterface r=(RemoteInterface)Naming.lookup("[rmi:]/hostname[:port]/name");
        // triệu gọi các phương thức của đối tượng từ xa
    }
}
```

Ví dụ 1-5: Sau đây là trình khách triệu gọi đối tượng HelloRMI

```
Client.java
import java.rmi.*;
import java.rmi.registry*;
public Client
{
    public static void main(String args[]) throws RemoteException
    {
        { // lấy về tham chiếu đối tượng phân tán bằng phương thức Naming.lookup()
            HelloRMI r=( HelloRMI)Naming.lookup("//localhost/hello");
            // triệu gọi phương thức sayHello của đối tượng từ xa
        }
        System.out.println(HelloRMI.sayHello());
    }
}
```

Để truy tìm đối tượng ở xa, chương trình máy khách gọi phương thức `Naming.lookup()`; Phương thức này chỉ yêu cầu đối số là chuỗi cho biết địa chỉ của máy chủ và tên đăng ký của đối tượng.

`RemoteInterface=(RemoteInterface)Naming.lookup(“[rmi://hostname[:port]/name”]);`

Mặc định phương thức `Naming.lookup()` trả về một tham chiếu Object nên bạn cần phải ép kiểu sang giao diện từ xa.

Bạn có thể hình dung phương thức `Naming.lookup()` đóng vai trò như một dịch vụ tìm kiếm đối tượng trên máy chủ, nó liên lạc với trình `rmiregistry` trên máy chủ và yêu cầu trả về tham chiếu đối tượng cho bạn sử dụng. Còn `Naming.bind()` đóng vai trò như dịch vụ đăng ký.

Biên dịch và chạy trình khách

`Javac Client.java`

Tất cả đã sẵn sàng, bạn hãy chạy chương trình khách như sau:

Java Client

Kết quả thu được là lời chào “*Hello RMI*”

Hình 3 diễn đạt cơ chế gọi phương thức từ xa của các đối tượng RMI một cách tổng quát:

Đăng ký đối tượng với trình `rmiregistry` bằng cách gọi phương thức `Naming.bind()` hoặc `Naming.rebind()`.

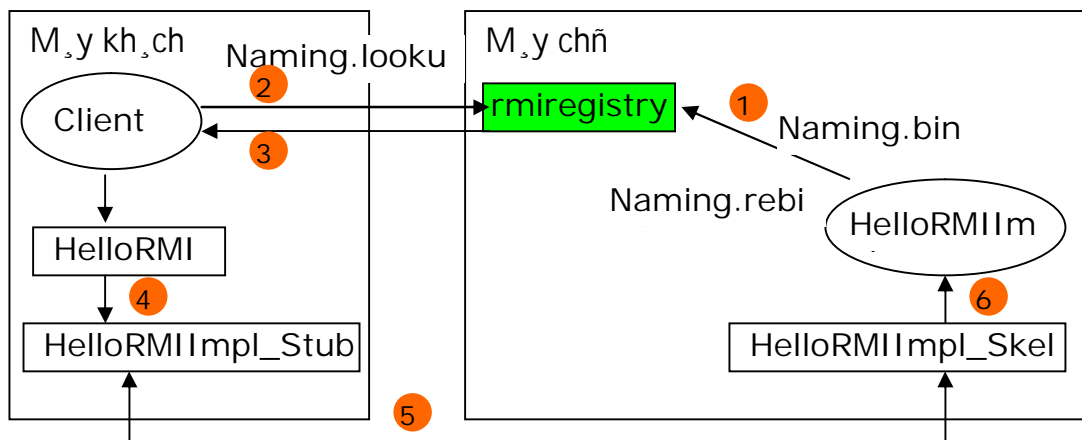
Máy khách muốn gọi phương thức của đối tượng trên máy chủ trước hết cần phải gọi phương thức `Naming.lookup()` để truy tìm tham chiếu đến đối tượng ở xa theo tên.

Bộ đăng ký sẽ trả về tham chiếu của đối tượng ở xa thông qua giao diện từ xa.

Dựa vào giao diện từ xa mà trình khách có thể gọi các phương thức của đối tượng ở xa.

Khi một phương thức được gọi, lời gọi sẽ được chuyển đến lớp trung gian `_Stub`. Lớp trung gian này có nhiệm vụ đóng gói các tham số và chuyển đến lớp trung gian `_Skel` trên trình chủ.

Lớp trung gian trên máy chủ sẽ tiếp nhận các tham số gửi đến từ trình khách và trực tiếp yêu cầu đối tượng thực thi phương thức và chuyển kết quả (nếu có) về cho trình khách.



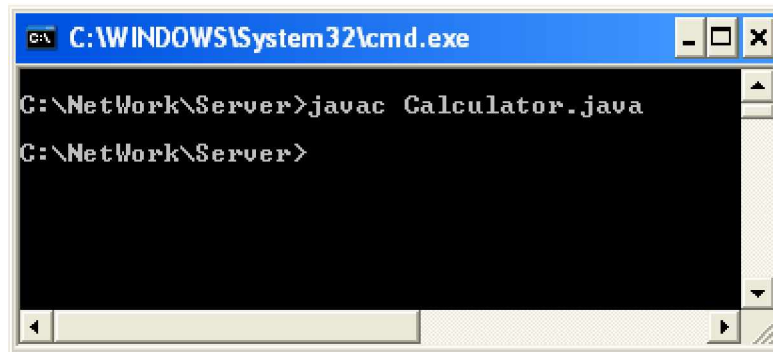
Ví dụ 2: Viết lớp Calculator, lớp này có các phương thức `add`, `less`, `mul`, `div` dùng để thực hiện các phép toán tương ứng cộng, trừ, nhân và chia

Bước 1: tạo ra giao diện từ xa

- Mở một trình soạn thảo bất kỳ gõ vào nội dung như sau:

```
//Calculator.java
import java.rmi.*;
public interface Calculator extends Remote {
    float add(float a, float b) throws RemoteException;
    float less(float a, float b) throws RemoteException;
    float mul(float a, float b) throws RemoteException;
    float div(float a, float b) throws RemoteException;
}
```

- Ghi lại vào thư mục C:\NetWork\Server
- Mở cửa sổ DOS-Prompt và gõ vào dòng lệnh biên dịch như sau:



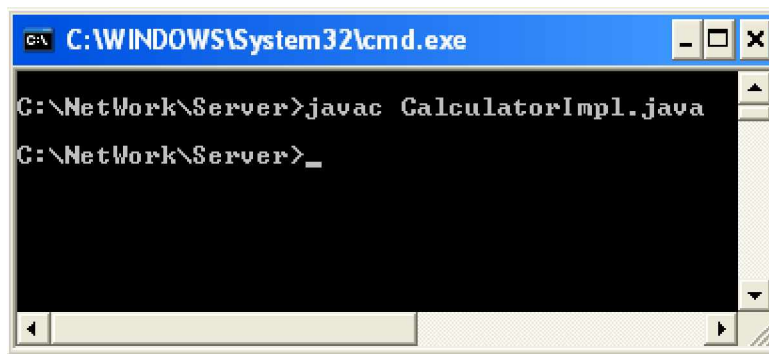
Bước 2: Cài đặt giao diện từ xa

- Mở trình soạn thảo bất kỳ gõ vào nội dung sau:

```
// CalculatorImpl.java
```

```
import java.rmi.RemoteException;
public class CalculatorImpl implements Calculator {
    public float add(float a, float b) throws RemoteException {
        return a+b;
    }
    public float less(float a, float b) throws RemoteException {
        return a-b;
    }
    public float mul(float a, float b) throws RemoteException {
        return a*b;
    }
    public float div(float a, float b) throws RemoteException {
        return a/b;
    }
}
```

- Ghi lại với cái tên CalculatorImpl.java
- Mở cửa sổ DOS_Prompt và gõ vào dòng lệnh sau để biên dịch

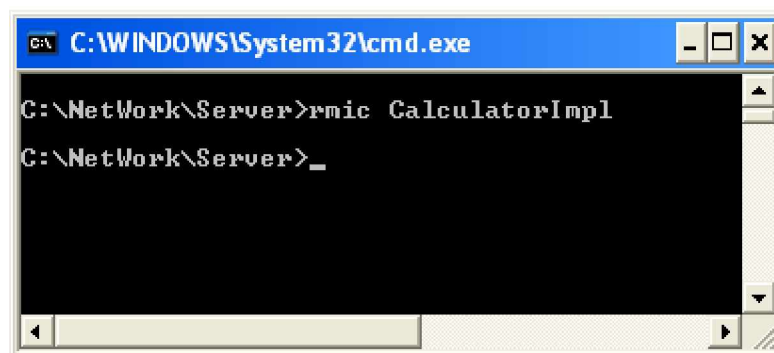


```
C:\WINDOWS\System32\cmd.exe
C:\Network\Server>javac CalculatorImpl.java
C:\Network\Server>_
```

sau khi biên dịch bạn thu được tệp tin CalculatorImpl.class

Bước 3 Tạo ra các lớp trung gian _Stub và _Skel

Bạn mở cửa sổ DOS_Prompt và gõ vào dòng lệnh



```
C:\WINDOWS\System32\cmd.exe
C:\Network\Server>rmic CalculatorImpl
C:\Network\Server>_
```

Sau khi thực hiện xong lệnh này bạn sẽ thu được hai lớp trung gian là CalculatorImpl_Stub và CalculatorImpl_Skel trong cùng thư mục C:\network\server

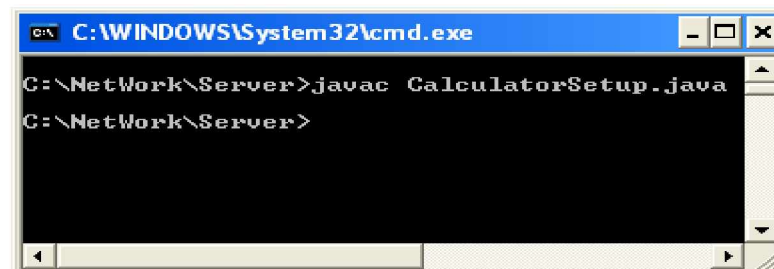
Bước 4: Đăng ký đối tượng từ xa với trình rmiregistry

- Bạn mở trình soạn thảo bất kỳ và gõ vào nội dung như sau:

```
// CalculatorSetup.java
import java.rmi.server.*;
import java.rmi.*;
public class CalculatorSetup {
    public static void main(String[] args) throws Exception{
        Calculator c=new CalculatorImpl();
        UnicastRemoteObject.exportObject(c);
        Naming.rebind("rmi://localhost/Cal",c);
    }
}
```

- Ghi lại với cái tên CalculatorSetup.java trong vào thư mục c:\network\server

- Mở cửa sổ DOS_Prompt gõ vào dòng lệnh sau:



```
C:\WINDOWS\System32\cmd.exe
C:\Network\Server>javac CalculatorSetup.java
C:\Network\Server>
```

Bước 5: Khởi động dịch vụ đăng ký tên rmiregistry

Bạn mở cửa sổ DOS-Prompt, chuyển vào thư mục C:\NetWork\Server và gõ vào dòng lệnh sau:



```
C:\WINDOWS\System32\cmd.exe - rmiregistry
C:\>cd network
C:\Network>cd server
C:\Network\Server>rmiregistry
```

Chương trình đăng ký đã được khởi động

Bước 6: Khởi động trình setup

Bạn mở cửa sổ DOS-Prompt khác gõ vào dòng lệnh

Chú ý: Nếu bạn không chuyển vào thư mục C:\NetWork\Server trước khi khởi động dịch vụ đăng ký thì bạn sẽ nhận được một lỗi

Chương 11: JAVA JSP (JAVA SERVLET)

Chương 12: EJB (ENTERPRISE JAVA BEANS)

Chương này ta sẽ nghiên cứu và xây dựng một trong những thành phần thú vị của java đó là JavaBean một mô hình quan trọng trong kiến trúc của Java.

Các vấn đề chính sẽ được đề cập trong chương này

JavaBean và công nghệ phần mềm

Tìm hiểu các thành phần của JavaBean

Tìm hiểu bộ phân tích Introspector của java

Xây dựng và sử dụng JavaBean

Tích hợp JavaBean với các công nghệ thành phần khác

I. Tổng quan về JavaBean

JavaBean là một công nghệ lập trình hướng thành phần của Sun dựa trên nền tảng của ngôn ngữ Java. JavaBean là một giao diện lập trình dùng để tạo ra những khối mã xây dựng sẵn và có thể dùng lại được. Những thành phần JavaBean có thể được “nhúng” vào một ứng dụng, liên kết với thành phần JavaBean khác hay được dùng để kết nối với các ứng dụng. Bằng cách sử dụng JavaBean các nhà phát triển ứng dụng có thể giảm khối lượng mã mà họ phải viết. Họ có thể tự viết ra chúng hoặc có thể mua lại của một hãng phát triển thứ ba khác (third party).

JavaBean thực chất là một lớp Java như các lớp Java thông thường khác, nhưng nó được viết theo một quy tắc quy định trước, Bạn không cần phải viết thêm bất cứ mã phụ nào hay sử dụng những mở rộng ngôn ngữ đặc biệt nào để tạo ra một Bean. Điều duy nhất bạn phải làm hơi sửa cách đặt tên cho các phương thức của mình. Chính tên phương thức mới báo cho công cụ xây dựng biết đây có là một thuộc tính hay một sự kiện hay chỉ là một phương thức thông thường.

II. Xây dựng thành phần JavaBean

Như đã giới thiệu ở trên JavaBean thực chất là một lớp java thông thường. Như vậy một JavaBean cũng có các thuộc tính và các phương thức. Một câu hỏi bạn đặt ra là làm thế nào để người dùng Bean của bạn có thể biết được trong Bean đó có những thuộc tính, phương thức nào, kiểu của chúng ra làm sao? Điều này đã được các nhà thiết kế java giải quyết một cách rất đơn giản. Để biết được các thuộc tính và các phương thức của một Bean bạn chỉ cần tuân thủ theo quy tắc đặt tên sau:

Với một thuộc tính có tên là **xxx**, thì bạn tạo ra hai phương thức: **getXxx()** và **setXxx()**. Lưu ý rằng ký tự thứ nhất sau “get” hay “set” phải viết hoa. Kiểu trả về của phương thức “get” cũng chính là kiểu của đối phương thức “set” và thường thì chúng cũng là kiểu của thuộc tính xxx. Bạn cũng chú ý cho là tên của thuộc tính và kiểu cho “get” và “set” là không liên quan.

Với một thuộc tính **boolean**, bạn có thể dùng cách tiếp cận “get” và “set” ở trên, nhưng bạn cũng có thể dùng “is” thay cho “get”.

Các phương thức thông thường của Bean không tuân theo quy tắc trên, nhưng chúng phải là phương thức **public**.

Nếu Bean cho phép đăng ký thính giả thì với một thính giả (listener) XxxListener bạn phải có hai phương thức tương ứng đi kèm là **addXxxListener(XxxListener)** và

removeXxxListener(XxxListener). Phần lớn các biến cố và thính giả là có sẵn bạn có thể tự tạo ra các biến cố và thính giả riêng của mình.

1. Thuộc tính của Bean

Thuộc tính thực chất là một biến nằm bên trong một lớp, thông thường các thuộc tính thường được đặt mức truy cập là **private**. để truy cập đến các thuộc tính **private** này ta thường dùng hai phương thức: “get” mang nghĩa là lấy giá trị của thuộc tính và “set” hàm nghĩa gán giá trị cho thuộc tính. Ví dụ với một Bean đơn giản có tên là xBean như sau:

```
public class xBean
{
    private String myName;
    public String getName() { return myName;}
    public void setName(String newName) { myName=newName;}
}
```

Thuộc tính myName trong lớp xBean được lấy và gán giá trị thông qua phương thức getName và setName. Như bạn thấy giữa tên thuộc tính và tên phương thức “get” và “set” là không liên quan đến nhau. Khi Bộ phân tích (Instrospector) của java thấy trong Bean có hai phương thức tên là getName và setName nó sẽ hiểu rằng bên trong Bean này có một thuộc tính tên là name (vì có hai phương thức tên là getName và setName).

Ví dụ: Trình bày cách tạo ra các thuộc tính và cách xây dựng các phương thức “get” và “set”

```
public class Bean1 {
    private String str; // thuộc tính kiểu String
    private int i;      // thuộc tính kiểu int
    private int[] arr; // thuộc tính kiểu mảng int
    private boolean b; // thuộc tính kiểu logic
    private float f;   // thuộc tính kiểu float
    private java.io.DataInputStream is; // thuộc tính kiểu đối tượng
    private java.io.DataOutputStream os;
    public String getStr() { //Phương thức get cho thuộc tính str
        return str;
    }
    public void setStr(String str) { //Phương thức set cho thuộc tính str
        this.str = str;
    }
    public int getI() { //Phương thức get cho thuộc tính i
        return i;
    }
    public void setI(int i) { //phương thức set cho thuộc tính i
        this.i = i;
    }
    //Phương thức get cho thuộc tính arr thuộc tính này là một //mảng nên giá trị trả về là một mảng
    public int[] getArr() { return arr; }
    //Phương thức set cho thuộc tính arr, thuộc tính này là một //mảng nên tham số là một mảng
    public void setArr(int[] arr) { this.arr = arr; }
```

```
//Phương thức get cho thuộc tính b, do thuộc tính này kiểu //logic nên bạn có thể sử dụng
is thay cho get
public boolean isB() { return b; }
public void setB(boolean b) { this.b = b; }
public float getF() { return f; }
public void setF(float f) { this.f = f; }
//nếu thuộc tính là một đối tượng thì phương thức get trả //về một đối tượng kiểu đối
tượng
public java.io.DataInputStream getIs() { return is; }
public void setIs(java.io.DataInputStream is) { this.is = is; }
public java.io.DataOutputStream getOs() { return os; }
public void setOs(java.io.DataOutputStream os) { this.os = os; }
}
```

a) Các thuộc tính thể hiện sự thay đổi (Bound Property)

JavaBean là các thành phần riêng biệt nhưng nó vẫn có thể tương tác với các đối tượng khác thông qua cơ chế truyền thông điệp (message). Khi một thuộc tính của một Bean A thay đổi nó có thể thông báo cho các đối tượng khác biết là thuộc tính của A đã thay đổi để đối tượng này thực hiện công việc gì đó chẳng hạn như cập nhật lại giá trị của nó theo thuộc tính của A.

Cơ chế cài đặt để một thuộc tính có thể thông báo cho các đối tượng khác biết khi nó thay đổi rất đơn giản chỉ cần cài đặt hai phương thức dùng để đăng ký và loại bỏ một đối tượng listener như sau:

```
public void add<PropertyName>Listener(PropertyChangeListener lst)
public void remove<PropertyName>Listener(PropertyChangeListener lst)
```

Ví dụ: bạn có một JavaBean A có một thuộc tính size cho biết kích cỡ gì đó. Khi thuộc tính size thay đổi để các đối tượng khác đều biết sự thay đổi này ta cài đặt như sau:

```
public class BeanA
{float size;
  

private PropertyChangeSupport changeSize=new PropertyChangeSupport(this);
public void addSizeListener(PropertyChangeListener lst){...}
public void removeSizeListener(PropertyChangeListener lst){...}
}
```

Các đối tượng muốn nhận được thông báo từ JavaBean này đều phải cài đặt giao diện **PropertyChangeListener** , sau đó đăng ký với thành phần BeanA nhờ các phương thức mà bạn xây dựng ở bước trên. Phương thức duy nhất mà bạn phải ghi đè khi cài đặt giao diện **PropertyChangeListener** là **public void propertyChange(PropertyChangeEvent evt) {...}**

Các thủ tục trên đây chỉ thực hiện nhiệm vụ đăng ký và tạo đối tượng Listener mà thôi. Để các đối tượng Listener này có thể nhận biết được sự thay đổi của thuộc tính size ta phải gọi phương thức **firePropertyChange** của đối tượng **PropertyChangeSupport** từ thành phần BeanA như sau:

```
public void firePropertyChange (String propertyName, Object oldValue, Object
newValue)
```

Phương thức này sẽ thông báo đến mọi đối tượng listener (bằng cách gọi phương thức **propertyChange** của listener) đã đăng ký với thành phần JavaBean. Sự thay đổi bao

gồm giá trị trước khi thay đổi (oldValue) và giá trị sau khi thay đổi (newValue) và tên thuộc tính được thay đổi.

Dưới đây là cài đặt lớp BeanA và một lớp để thử nghiệm thuộc tính nhận biết sự thay đổi size

//BeanA.java

```
import java.beans.*;
public class BeanA {
    //Khai báo thuộc tính size
    private float size;
    // changeSize có nhiệm vụ quản lý các đối tượng listener đã đăng //ký và gửi thông báo
    //đến chúng mỗi khi thuộc tính size thay đổi
    PropertyChangeSupport changeSize = new PropertyChangeSupport(this);
    public float getSize() { return size; }
    //Phương thức gán giá trị cho thuộc tính size
    public void setSize(float newSize) {
        float oldSize = size; size = newSize;
    //Nếu có thay đổi về kích thước thì thông báo đến tất cả các đối //tượng đã đăng ký lắng
    //nghe
        if(oldSize!=newSize)
            changeSize.firePropertyChange("size", new Float(oldSize), new Float(newSize));
    }
    public void addSizeListener(PropertyChangeListener lst) {
        changeSize.addPropertyChangeListener(lst);
    }
    public void removeSizeListener(PropertyChangeListener lst) {
        changeSize.removePropertyChangeListener(lst);
    }
}
```

Ví dụ sau thử nghiệm BeanA vừa cài đặt ở trên

// TestBeanA.java

```
import java.beans.*;
public class TestBeanA {
    public static void main(String[] args) {
    // tạo ra một đối tượng BeanA
        BeanA ba = new BeanA();
    //Đăng ký đối tượng lắng nghe với đối tượng ba
        ba.addSizeListener(new MyListener());
    //thay đổi giá trị của thuộc tính size
        ba.setSize(333);
    } }
    //Tạo ra đối tượng có khả năng lắng nghe sự thay đổi của thuộc tính
class MyListener implements PropertyChangeListener {
    //Phương thức duy nhất cần phải cài đặt, phương thức này //được triệu gọi mỗi khi
    //thuộc tính thay đổi
    public void propertyChange(PropertyChangeEvent evt) {
        float oldSise = ( (Float) evt.getOldValue()).floatValue();
        float newdSise = ( (Float) evt.getNewValue()).floatValue();
        String propertyName = evt.getPropertyName();
    }
}
```

```

System.out.println("Change " + propertyName + " old value " + oldSise +
    " new value " + newdSise);
}
}

```

Kết quả khi chạy chương trình như sau:

Change size old value 1.0 **new** value 333.0

Chú ý:

Trong ví dụ trên tên của phương thức dùng để đăng ký là ***addSizeListener*** và tên của phương thức dùng để gỡ bỏ đăng ký là ***removeSizeListener*** tuy nhiên bạn có thể đặt cho nó một cái tên tùy thích chẳng hạn như *dangKyLangNghe* để đăng ký đối tượng lắng nghe hay *huyBoLangNghe* để huỷ bỏ đối tượng lắng nghe.

Các đối tượng sẽ không biết đến sự thay đổi của thuộc tính size cho đến khi bạn gọi phương thức ***firePropertyChange*** của đối tượng ***PropertyChangeSupport***.

b) Các thuộc tính ràng buộc (Constrain Property)

Hạn chế của các thuộc tính bound ở trên là các đối tượng chỉ phản ứng lại khi giá trị của thuộc tính đã bị thay đổi. Java cung cấp một khả năng khác mạnh hơn đó là tạo ra các thuộc tính bị ràng buộc (constrain property). Tương tự như thuộc tính bound mỗi khi thuộc tính constrain định thay đổi giá trị (chỉ dự định chứ chưa thay đổi) nó sẽ thông báo đến tất cả các đối tượng đang lắng nghe. Đối tượng lắng nghe có thể từ chối sự thay đổi và như vậy giá trị của thuộc tính vẫn giữ nguyên giá trị ban đầu. Ngược lại nếu tất cả các đối tượng lắng nghe đều không phản đối sự thay đổi thì lúc đó thuộc tính này mới nhận được giá trị mới.

Để một thuộc tính có thể thông báo cho các đối tượng khác biết khi nó sắp thay đổi giá trị rất đơn giản bạn chỉ cần cài đặt hai phương thức dùng để đăng ký và loại bỏ một đối tượng listener như sau:

```
public void add<PropertyName>Listener(VetoableChangeListener lst)
```

```
public void remove<PropertyName>Listener(VetoableChangeListener lst)
```

Các đối tượng muốn nhận được thông báo từ JavaBean này đều phải cài đặt giao diện ***VetoableChangeListener***, sau đó đăng ký, loại với thành phần BeanA nhờ các phương thức mà bạn xây dựng ở bước trên. Phương thức duy nhất mà bạn phải ghi đề khi cài đặt giao diện ***VetoableChangeListener*** là

```
public void vetoableChange(PropertyChangeEvent evt) throws PropertyVetoException
```

```
{.. }
```

Đối tượng lắng nghe không đồng ý cho thay đổi thuộc tính nó chỉ cần ném ra một ngoại lệ ***PropertyVetoException***, thành phần JavaBean có thuộc tính constrain sẽ bắt lấy và huỷ bỏ việc thay đổi giá trị cho thuộc tính.

Cũng như thuộc tính bound, khi thuộc tính constrain có dự định thay đổi giá trị trong phương thức “set” nó cần thông báo đến các đối tượng đã đăng ký lắng nghe (listener) với nó bằng phương thức ***fireVetoableChange*** của đối tượng ***VetoableChangeSupport*** từ thành phần JavaBean như sau:

```
public void fireVetoableChange (String propertyName, Object oldValue, Object newValue)
```

Phương thức này sẽ thông báo đến mọi đối tượng listener (bằng cách gọi phương thức *vetoableChange* của listener) đã đăng ký với thành phần JavaBean.

Nhận xét: Đối tượng *VetoableChangeSupport* cũng tương tự như đối tượng *PropertyChangeSupport* dùng để quản lý các đối tượng listener. *PropertyChangeSupport* được dùng với các thuộc tính bound còn *VetoableChangeSupport* dùng cho thuộc tính constrain.

Dưới đây là một cài đặt cho một JavaBean B có thuộc tính size. Mỗi khi size thay đổi nó sẽ thông báo cho các thành phần khác biết thông qua đối tượng listener. Đối tượng listener này sẽ từ chối không cho thay đổi nếu size >100.

// BeanB.java

```
import java.beans.*;
public class BeanB {
    //Khai báo thuộc tính size
    private int size = 0;
    // vetoAgent có nhiệm vụ quản lý tất cả các đối tượng listener //đã đăng ký
    private VetoableChangeSupport vetoAgent = new VetoableChangeSupport(this);
    //Phương thức lấy về giá trị của thuộc tính size
    public int getSize() { return size; }
    //Phương thức dùng để thay đổi giá trị của thuộc tính size
    public void setSize(int newSize) {
        int oldSize = size;
        try { // Thông báo đến tất cả các đối tượng đã đăng ký lắng nghe
            vetoAgent.fireVetoableChange("size", oldSize, newSize);
        }
        catch (PropertyVetoException ex) {
            // Nếu có một listener nào đó phản đối không cho phép thay đổi //giá trị của thuộc
            // tính size thì kết thúc phương thức
            return;
        }
        //Nếu tất cả các listener đều không phản đối sự thay đổi thì // tiến hành gán giá trị mới
        // cho thuộc tính size
        size = newSize;
    }
    //Phương thức dùng để đăng ký đối tượng lắng nghe
    public void addSizeListener(VetoableChangeListener lst) {
        vetoAgent.addVetoableChangeListener(lst);
    }
    //Phương thức dùng để gỡ bỏ đối tượng lắng nghe
    public void removeSizeListener(VetoableChangeListener lst) {
        vetoAgent.removeVetoableChangeListener(lst);
    }
}
```

Ví dụ sau dùng để test thử BeanB vừa tạo

// TestBeanB.java

```
import java.beans.*;
public class TestBeanB {
    public static void main(String[] args) {
```

```

        //Tạo ra một đối tượng BeanB
        BeanB b = new BeanB();
        //Đăng ký đối tượng lắng nghe
        b.addSizeListener(new MyListener());
        //Gán giá trị 10 (10<100) cho thuộc tính size
        b.setSize(10);
        System.out.println("size=10");
        System.out.println(">Sau khi gán thuộc tính size= "+b.getSize());
        //Gán giá trị 123 (123>100) cho thuộc tính size
        b.setSize(123);
        System.out.println("size=123");
        System.out.println(">Sau khi gán thuộc tính size= "+b.getSize());

        //Gán giá trị 234 (234>100) cho thuộc tính size
        b.setSize(234);
        System.out.println("size=234");
        System.out.println(">Sau khi gán thuộc tính size= "+b.getSize());
        //Gán giá trị 56 (56>100) cho thuộc tính size
        b.setSize(56);
        System.out.println("size=56");
        System.out.println(">Sau khi gán thuộc tính size= "+b.getSize());
    }
}

//Cài đặt lớp lắng nghe biến cố VetoableChangeListener
class MyListener implements VetoableChangeListener {
    //Ghi đè phương thức vetoableChange
    public void vetoableChange(PropertyChangeEvent evt) throws
        PropertyVetoException {
        int oldSize = ( (Integer) evt.getOldValue()).intValue();
        int newSize = ( (Integer) evt.getNewValue()).intValue();
        String propName = evt.getPropertyName();
        //Nếu newSize>100 thì không cho thay đổi bằng cách ném ra một //ngoại lệ
        PropertyVetoException
        if (newSize > 100)
            throw new PropertyVetoException(
            "Giá trị gán cho Size lớn hơn 100", evt);
        }
    }
}

```

Kết quả khi chạy chương trình

```

size=10
>Sau khi gán thuộc tính size= 10
size=123
>Sau khi gán thuộc tính size= 10
size=234
>Sau khi gán thuộc tính size= 10
size=56
>Sau khi gán thuộc tính size= 56

```


2. Xây dựng các phương thức cho JavaBean

Các phương thức của JavaBean được xây dựng như các phương thức thông thường.

3. Tạo ra JavaBean có khả năng lắng nghe biến cố

Một JavaBean có thể được thiết kế để lắng nghe một số biến cố do các đối tượng khác gửi đến. Để có thể lắng nghe và xử lý các biến cố do các đối tượng khác gửi đến JavaBean của bạn chỉ cần triển khai các giao diện tương ứng và cài đặt các phương thức cần thiết trong các giao diện đó, sau đó đăng ký nó với các đối tượng nguồn.

Ví dụ: Để JavaBean của bạn có thể xử lý các biến cố về chuột do các đối tượng khác gửi đến, JavaBean của bạn chỉ cần cài đặt giao diện MouseListener như sau:

```
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
public class Bean3
    implements MouseListener {
    public void mouseClicked(MouseEvent e) {
    }
    public void mouseEntered(MouseEvent e) {
    }
    public void mouseExited(MouseEvent e) {
    }
    public void mousePressed(MouseEvent e) {
    }
    public void mouseReleased(MouseEvent e) {
    }
}
```

Nhận xét: Như bạn thấy việc tạo ra một JavaBean có thể lắng nghe các biến cố hoàn toàn tương tự như việc tạo ra các đối tượng lắng nghe biến cố khác.

4. Tạo ra JavaBean có thể đăng ký đối tượng lắng nghe biến cố

Ngoài việc bạn có thể tạo ra các JavaBean có thể lắng nghe biến cố do các đối tượng khác gửi đến bạn còn có thể tạo ra các JavaBean có khả năng đăng ký và gửi đi các biến cố đến các đối tượng lắng nghe khác. Lúc này JavaBean của bạn được coi là đối tượng nguồn (đối tượng phát sinh biến cố- Event Source). Các đối tượng khác muốn lắng nghe và xử lý biến cố do JavaBean của bạn gửi đến nó phải cài đặt giao diện lắng nghe tương ứng và phải đăng ký nó với JavaBean của bạn.

4.1. Đăng ký đối tượng listener với thành phần JavaBean

Để các thành phần khác có thể đăng ký được đối tượng listener với thành phần JavaBean thì bản thân thành phần JavaBean của bạn phải cài đặt hai phương thức theo khuôn mẫu (pattern) sau:

```
public synchronized void add<EventListener>(<EventListener> lst)
public synchronized void remove<EventListener>(<EventListener> lst)
```

Ví dụ: Để các đối tượng khác có thể đăng ký đối tượng listener cho tình huống nhấn chuột thì JavaBean của bạn phải cài đặt hai phương thức đăng ký như sau:

```
public synchronized void addMouseListener(MouseListener l)  
public synchronized void removeMouseListener(MouseListener l)
```

Thành phần JavaBean cần lưu lại danh sách các listener đã đăng ký. Khi muốn chuyển giao tình huống. JavaBean sẽ gọi đến từng phương thức tương ứng của các listener này. Nếu có nhiều listener cùng đăng ký thì JavaBean của bạn thường lưu lại danh sách các đối tượng listener đã đăng ký bằng một Vector như sau:

```
//Tạo ra một Vector để lưu lại danh sách các listener đã đăng ký
```

```
private transient Vector mouseListeners=new Vector();  
//Phương thức đăng ký đối tượng listener
```

```
public synchronized void addMouseListener(MouseListener l) {  
//Nếu listener đăng ký chưa có trong danh sách thì thêm nó vào  
    if (!mouseListeners.contains(l)) {  
        mouseListeners.addElement(l); // Thêm vào danh sách  
    }  
}
```

```
//Phương thức dùng để loại bỏ một listener
```

```
public synchronized void removeMouseListener(MouseListener l) {  
    if (mouseListeners != null && mouseListeners.contains(l)) {  
        mouseListeners.removeElement(l);  
    }  
}
```

Hai phương thức trên chỉ là hình thức dùng để đăng ký và loại bỏ các đối tượng lắng nghe mà thôi. Thực chất nó chưa gửi đi bất cứ thông báo nào đến đối tượng lắng nghe. Khi một JavaBean muốn thông báo đến các đối tượng lắng nghe nó chỉ cần duyệt trong danh sách các đối tượng đã đăng ký lắng nghe biến cố và gọi phương thức tương ứng mà đối tượng listener đã cài đặt.

Dưới đây là một ví dụ minh họa, ta xây dựng thành phần JavaBean có khả năng tiếp nhận các đối tượng listener dùng để xử lý tình huống chuột.

```
// MouseBean.java
```

```
import java.awt.*;  
import javax.swing.*;  
import java.awt.event.*;  
import java.util.*;  
public class MouseBean  
    extends JPanel {  
// Một Vector lưu trữ các đối tượng listener  
    private transient Vector mouseListeners = new Vector();  
// Tạo ra một nút lệnh  
    JButton jButton1 = new JButton();  
    public MouseBean() {  
        jButton1.setText("Mouse JavaBean");  
        jButton1.addMouseListener(new MyListener(this));  
        this.setLayout(new BorderLayout());  
        this.add(jButton1, BorderLayout.CENTER);  
    }  
//Phương thức dùng để loại bỏ đối tượng lắng nghe biến cố
```

```

public synchronized void removeMouseListener(MouseListener l) {
    if (mouseListeners != null && mouseListeners.contains(l)) {
        mouseListeners.removeElement(l);
    }
}
//Phương thức dùng để đăng ký đối tượng lắng nghe biến cố
public synchronized void addMouseListener(MouseListener l) {
    if (!mouseListeners.contains(l)) {
        mouseListeners.addElement(l);
    }
}
/*Phương thức này dùng để thông báo đến tất cả các đối tượng lắng nghe biến cố khi có
sự kiện mouseClicked xảy ra trên đối tượng jButton1*/
protected void fireMouseClicked(MouseEvent e) {
    if (mouseListeners != null) {
        int count = mouseListeners.size();
        for (int i = 0; i < count; i++) {
            ((MouseListener) listeners.elementAt(i)).mouseClicked(e);
        }
    }
}
/*Khi chuột được kích trên nút lệnh thì thông báo đến tất cả các đối tượng đã đăng ký
lắng nghe bằng cách gọi phương thức fireMouseClicked */
void jButton1_mouseClicked(MouseEvent e) {
    fireMouseClicked(e);
}
}
//Tạo ra lớp lắng nghe sự kiện chuột của nút lệnh
class MyListener
    extends java.awt.event.MouseAdapter {
    MouseBean adaptee;
    MyListener(MouseBean adaptee) {
        this.adaptee = adaptee;
    }
    public void mouseClicked(MouseEvent e) {
        adaptee.jButton1_mouseClicked(e);
    }
}
}

```

Sau đây là ứng dụng để test thử thành phần MouseBean. Có một thính giả lắng nghe biến cố click trên MouseBean. Khi bạn kích chuột trên MouseBean thì thính giả này sẽ xuất hiện một hộp thông báo (message).

// **Frame1.java**

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Frame1
    extends JFrame {
//tạo ra đối tượng MouseBean

```

```

MouseBean mouseBean1 = new MouseBean();
public Frame1() {
    this.setDefaultCloseOperation(EXIT_ON_CLOSE);
    this.setLocale(java.util.Locale.getDefault());
    this.getContentPane().setLayout(null);
    mouseBean1.setBounds(new Rectangle(61, 34, 137, 48));
    //Đăng ký đối tượng lắng nghe với đối tượng MouseBean
    mouseBean1.addMouseListener(new MouseBeanListener(this));
    this.getContentPane().add(mouseBean1, null);
}
public static void main(String[] args) {
    Frame1 f = new Frame1();
    f.setBounds(100, 100, 270, 150);
    f.setVisible(true);
}
//Khi kích chuột vào MouseBean thì hiển thị hộp thông báo
void mouseBean1_mouseClicked(MouseEvent e) {
    JOptionPane.showMessageDialog(this, "Ban vua kich chuot vao MouseBean","Test
MouseBean",JOptionPane.INFORMATION_MESSAGE);
}
}
//Tạo ra đối tượng lắng nghe
class MouseBeanListener
    extends java.awt.event.MouseAdapter {
    Frame1 adaptee;
    MouseBeanListener(Frame1 adaptee) {
        this.adaptee = adaptee;
    }
    public void mouseClicked(MouseEvent e) {
        adaptee.mouseBean1_mouseClicked(e);
    }
}
}

```

Kết quả chạy chương trình sau khi bạn kích chuột vào MouseBean



4.2. Khả năng đăng ký một listener (unicast) và nhiều listener (multicast) của thành phần JavaBean

Thông thường một JavaBean cho phép đăng ký nhiều đối tượng lắng nghe biến cố. Cơ chế cho phép nhiều listener cùng được đăng ký để xử lý cùng một tình huống được gọi là multicast. Tuy nhiên, với cơ chế này đôi lúc sẽ dẫn đến trường hợp xâm phạm lẫn nhau, chẳng hạn listener này thì cho phép thay đổi giá trị của thuộc tính trong khi listener khác lại ngăn cản. Chính vì vậy một vài thành phần JavaBean chỉ cho phép đăng ký một listener mà thôi (unicast), nếu có nhiều hơn một listener được đăng ký thì phương thức đăng ký của JavaBean cần được thiết kế để ném ra một ngoại lệ là: *TooManyListenerException* như sau:

```
public void add<EventListenerType>(< EventListenerType> lst) throws  
TooManyListenerException
```

Đoạn chương trình sau tạo ra hai phương thức add và remove chỉ cho phép đăng ký một listener mà thôi:

```
MouseListener saveListener=null;
```

```
/* Phương thức đăng ký đối tượng listener*/
```

```
public void addMouseListener(MouseListener lst) throws TooManyListenerException
```

```
{ if(saveListener==null){
```

```
saveListener=lst;
```

```
} else{
```

```
/*
```

```
Nếu đã có một listener nào đó được đăng ký thì ném ra một ngoại lệ
```

```
*/
```

```
throw new TooManyListenerException();
```

```
}
```

```
//Phương thức này dùng để loại bỏ đối tượng listener
```

```
public void removeMouseListener(MouseListener lst)
```

```
{
```

```
saveListener=null;
```

```
}
```

Để hiểu rõ hơn, dưới đây ta sẽ viết một ví dụ minh họa cho sự tranh chấp khi có nhiều đối tượng listener cùng đăng ký xử lý một tình huống. Chương trình này dùng lại thành phần **BeanB** đã thiết kế ở phần đầu (xem ví dụ về thuộc tính ràng buộc –constrain property). Thuộc tính size của thành phần BeanB là thuộc tính ràng buộc, trước khi thay đổi giá trị của thuộc tính nó cần hỏi ý kiến của tất cả listener. Để minh họa ta sẽ tạo ra hai đối tượng listener là MyListener21 chỉ cho phép thay đổi thuộc tính size khi giá trị mới lớn hơn 50 còn MyListener22 chỉ cho phép thay đổi thuộc tính size khi giá trị mới lớn hơn 100. Chương trình sử dụng hai đối tượng listener cùng xử lý một tình huống.

```
//UseBean2.java
```

```
import java.beans.VetoableChangeListener;  
import java.beans.PropertyChangeEvent;  
import java.beans.PropertyVetoException;  
public class UseBean2 {  
public static void main(String[] args) {
```

```

BeanB b = new BeanB();
b.addSizeListener(new myListener21());
b.addSizeListener(new myListener22());
System.out.println("old size:" + b.getSize() + " set new size 30");
b.setSize(30);
System.out.println("size after setting:" + b.getSize());
System.out.println("old size:" + b.getSize() + " set new size 90");
b.setSize(90);
System.out.println("size after setting:" + b.getSize());
}
}
class myListener21
    implements VetoableChangeListener {
    public void vetoableChange(PropertyChangeEvent evt) throws
        PropertyVetoException {
        int newSize = (Integer) evt.getNewValue().intValue();
        if (newSize > 50)
            throw new PropertyVetoException("Gia tri lon hon 50 Listener 1 reject",
                evt);
        }
    }
}
class myListener22
    implements VetoableChangeListener {
    public void vetoableChange(PropertyChangeEvent evt) throws
        PropertyVetoException {
        int newSize = (Integer) evt.getNewValue().intValue();
        if (newSize > 100)
            throw new PropertyVetoException("Gia tri lon hon 100 Listener 2 reject",
                evt);
        }
    }
}

```

Kết quả chạy chương trình:

old size:0 set new size 30

size after setting:30

old size:30 set new size 90

size after setting:30

Quan sát kết quả chạy chương trình ta thấy tất cả giá trị lớn hơn 50 đều không thể gán được cho thuộc tính size, mặc dù listener thứ hai là myListener22 cho phép gán giá trị >50 và <100 nhưng listener thứ nhất là myListener21 lại không cho phép, như vậy trong trường hợp listener thứ hai myListener22 cài đặt mà không biết rằng myListener21 đã được cài đặt trước đó thì người dùng cuối sẽ bị nhầm lẫn.

Chính vì vậy java phân biệt hai cách đăng ký listener là unicast và multicast. Tuy nhiên theo khuyến khích thì các thành phần JavaBean nên được xây dựng để hỗ trợ cơ chế đăng ký nhiều đối tượng listener (multicast).

III. Tìm hiểu Instrospector

1. Instrospector là gì?

Một câu hỏi bạn đặt ra là khi bạn kéo một JavaBean từ thanh công cụ và thả vào Form (trong một môi trường phát triển trực quan nào đó chẳng hạn như Jbuilder chẳng hạn), làm thế nào để môi trường phát triển trực quan này có thể biết được trong JavaBean của ta có những thuộc tính, phương thức, sự kiện nào...?

Vấn đề này đã được giải quyết bằng tính phản xạ (*reflect*), cơ chế phản xạ giúp ta có thể phát hiện ra các thuộc tính, phương thức, sự kiện của một lớp bất kỳ (kể cả lớp vô danh). Điều này là hoàn hảo để giải quyết vấn đề Bean mà không yêu cầu bạn phải làm thêm bất cứ thủ thuật ngôn ngữ nào như vẫn bị yêu cầu trong các ngôn ngữ lập trình trực quan khác.

Trong thực tế, một trong những lý do chính mà phản xạ được thêm vào cho java là để hỗ trợ JavaBean (mặc dù nó còn được dùng trong tuần tự hoá và gọi các phương thức động từ xa). Nhưng những người thiết kế java lại muốn cung cấp một công cụ chuẩn, không những làm cho JavaBean đơn giản hơn trong sử dụng mà còn cung cấp một ngõ cửa chuẩn để tạo ra các JavaBean phức tạp hơn. Công cụ này chính là lớp **Instrospector**, phương pháp quan trọng nhất của lớp này là static `getBeanInfo()`. Bạn truyền tham chiếu **Class** cho lớp này và nó hoàn toàn đi hỏi lớp đó và cho lại một đối tượng **BeanInfo**, dựa vào **BeanInfo** này bạn có thể mổ xẻ để tìm ra các thuộc tính, phương thức, sự kiện của JavaBean.

2. Cách mà Instrospector làm việc

Ví dụ khi ra có hai phương thức như sau:

```
public int getSize()  
public void setSize(int newSize)
```

Theo cách này java hiểu rằng bạn muốn tạo ra một thuộc tính có tên là *size* có kiểu *int*. Hay khi tạo ra các thuộc tính ràng buộc (constrain property) ta viết:

```
public addSizeListener(VetoableChangeListener lst)  
public removeSizeListener(VetoableChangeListener lst)
```

java sẽ hiểu là có một thuộc tính size cần gọi đến hai phương thức này mỗi khi muốn đăng ký hay loại bỏ một đối tượng listener dùng để quan sát sự thay đổi của thuộc tính.

Tóm lại nếu các phương thức được thiết kế theo một mẫu (pattern) nào đó thì nó sẽ được đem dùng vào mục đích riêng. Thường thì ta dùng các mẫu sau khi xây dựng các phương thức dành cho thuộc tính và sự kiện.

Thiết kế các phương thức mẫu dùng cho thuộc tính

Đối với mỗi thuộc tính bạn cần xây dựng hai phương thức “get” và “set” theo mẫu sau đây:

```
public <propertyType> get<PropertyName>()  
public void set<PropertyName>(<propertyType> newValue)
```

Trong đó:

propertyType là bất cứ kiểu dữ liệu nào nguyên thủy hay đối tượng

PropertyName là tên của thuộc tính

Chú ý:

Nếu thuộc tính có kiểu boolean thì phương thức “get” có thể thay bằng phương thức “is”

Java chỉ hiểu là bạn định tạo ra một thuộc tính nếu như có đủ cả hai phương thức “get” và “set”

Kiểu trả về của phương thức “get” thường là kiểu của đối phương thức “set”

Thiết kế các phương thức mẫu dùng cho sự kiện

Để xây dựng một sự kiện nào đó cho một thành phần JavaBean ta cần xây dựng hai phương thức “add” và “remove” theo mẫu sau đây:

```
public void add<EventListenerType> (<EventListenerType> evt)
```

```
public void remove<EventListenerType> (<EventListenerType> evt)
```

Trong đó EventListenerType là đối tượng tình huống mà JavaBean muốn các thành phần khác tương tác được với mình. Ví dụ để thành phần JavaBean có thể đăng ký được các tình huống *Action* ta viết

```
public void addActionListener (ActionListener lst)
```

```
public void removeActionListener (ActionListener lst)
```

3. Tìm hiểu lớp BeanInfo

Khi bạn tạo ra BeanB (xem lại lớp này trong phần thuộc tính ràng buộc) bạn có thể tạo ra một lớp phụ khác là *BeanBBeanInfo.java* lớp này được dẫn xuất từ lớp *SimpleBeanInfo* có nhiệm vụ cung cấp các thông tin về thành phần JavaBean mà ta xây dựng:

```
import java.beans.*;  
public class BeanBBeanInfo extends SimpleBeanInfo {  
    Class beanClass = BeanB.class;  
    String iconColor16x16Filename;  
    String iconColor32x32Filename;  
    String iconMono16x16Filename;  
    String iconMono32x32Filename;  
    public BeanBBeanInfo() { }  
    public PropertyDescriptor[] getPropertyDescriptors() {  
        try {  
            PropertyDescriptor _size = new PropertyDescriptor("size", beanClass, "getSize",  
                                                                "setSize");  
  
            PropertyDescriptor[] pds = new PropertyDescriptor[] { _size};  
            return pds;  
        }  
        catch(IntrospectionException ex) {    ex.printStackTrace();    return null; }  
    }  
    public java.awt.Image getIcon(int iconKind) {  
        switch (iconKind) {  
            case BeanInfo.ICON_COLOR_16x16:  
                return iconColor16x16Filename != null ? loadImage(iconColor16x16Filename) : null;  
            case BeanInfo.ICON_COLOR_32x32:  
                return iconColor32x32Filename != null ? loadImage(iconColor32x32Filename) : null;  
            case BeanInfo.ICON_MONO_16x16:  
                return iconMono16x16Filename != null ? loadImage(iconMono16x16Filename) : null;  
            case BeanInfo.ICON_MONO_32x32:  
                return iconMono32x32Filename != null ? loadImage(iconMono32x32Filename) : null;  
        }  
        return null;  
    }  
}
```


Các phương thức quan trọng mà bạn cần quan tâm là:

Phương thức *getIcon()* phương thức này được gọi khi các môi trường phát triển trực quan muốn lấy về biểu tượng cho JavaBean.

Phương thức *getPropertyDescriptors()* phương thức này giúp cho Introspector dễ dàng lấy về các thuộc tính của JavaBean.

Phương thức *getMethodDescriptors()* phương thức này giúp cho Introspector dễ dàng lấy về các phương thức của JavaBean...

Bằng cách xây dựng lớp BeanInfo đi kèm với JavaBean của bạn. Trình biên dịch dễ dàng tìm ra các thuộc tính, phương thức, sự kiện. Trước khi muốn biết JavaBean của bạn có những thành phần nào trong Bean của bạn trước tiên nó đi tìm trong lớp BeanInfo tương ứng, nếu không tìm thấy BeanInfo đi kèm với lớp lúc đó nó mảy sử dụng cơ chế phản xạ hay đi tìm các phương thức được thiết kế theo mẫu.

IV. Đóng gói JavaBean

Trước khi xuất bản JavaBean bạn thường đóng gói nó vào trong một tệp .jar chuẩn. Để các công cụ xây dựng trực quan biết trong tệp đóng gói này là một JavaBean bạn cần tạo tệp mô tả đóng gói bằng cách sau:

Sử dụng một trình soạn thảo bất kỳ để tạo ra tệp tin BeanB.mf như sau:

Manifest-Version: 1.0

Name BeansDir/BeanB.class

Name BeansDir/BeanBInfo.class

Java-Bean: True

Ghi lại với cái tên gì đó chẳng hạn BeanB.mf

Dòng thứ nhất cho biết phiên bản, từ dòng thứ hai trở đi cho biết trong tên của các lớp có trong tệp đóng gói, dòng cuối cùng cho biết đơn vị đóng gói này là một JavaBean.

Bạn tiến hành đóng gói bằng lệnh jar hoàn toàn tương tự như bạn đóng gói các lớp khác.