UBND TỈNH LÂM ĐỒNG
TRƯỜNG CAO ĐẲNG ĐÀ LẠT

GIÁO TRÌNH

MÔN HỌC: NGÔN NGỮ JAVA

NGÀNH/NGHỀ: CÔNG NGHỆ THÔNG TIN (ỨNG DỤNG PHẦN MỀM)

TRÌNH ĐỘ: CAO ĐẲNG

Ban hành kèm theo Quyết định số: /QĐ-… ngày……tháng….năm ..... …… ..... của……

LƯU HÀNH NỘI BỘ

Đà Lạt, năm 2017

TUYÊN BỐ BẢN QUYỀN

LỜI GIỚI THIỆU

Đây là tài liệu được xây dựng theo chương trình khung chương trình đào tạo nghề Công nghệ thông tin (Ứng dụng phần mềm)hệ cao đẳng nghề.

Trong giai đoạn viết giáo trình chúng tôi cũng đã có những sự điều chỉnh để giáo trình có tính thiết thực và phù hợp hơn với sự phát triển của lĩnh vực công nghệ thông tin.

Ngôn ngữ lập trình java ra đời và được các nhà nghiên cứu của Công ty Sun Microsystem giới thiệu vào năm 1995. Sau khi ra đời không lâu, ngôn ngữ lập trình này đã được sử dụng rộng rãi và phổ biến đối với các lập trình viên chuyên nghiệp cũng như các nhà phát triển phần mềm. Gần đây ngôn ngữ lập trình, công nghệ java đã được đưa vào giảng dạy ở các cơ sở đào tạo lập trình viên chuyên nghiệp. Một số trường đại học ở Việt Nam dạy môn lập trình java như một chuyên đề tự chọn cho các sinh viên công nghệ thông tin giai đoạn chuyên ngành.

Nội dung giáo trình tập trung vào những kiến thức căn bản nhất của lập trình java giúp người đọc bước đầu tiếp cập dễ dàng với công nghệ mới này, và đây cũng chính là một bước đệm để chúng ta tiếp cận việc lập trình ứng dụng trên các thiết bị di động, nó là một trong những xu hướng lập trình ứng dụng đang được phát triển mạnh mẽ trong thực tiễn đời sống hiện nay.

Để có thể đọc hiểu giáo trình này người đọc cần nắm vững các kiến thức về: nhập môn lập trình, lập trình hướng đối tượng. Trong quá trình biên soạn, mặc dù đã cố gắng tham khảo nhiều tài liệu và giáo trình khác nhưng tác giả không tránh khỏi được những thiếu sót và hạn chế. Tác giả chân thành mong đợi những nhận xét, đánh giá và góp ý để cuốn giáo trình ngày một hoàn thiện hơn. Mọi thông tin đóng góp xin gửi thư điện tử về địa chỉ ngothienhoang@cdndalat.edu.vn

Tài liệu này được thiết kế theo từng mô đun/ môn học thuộc hệ thống mô đun/môn học của một chương trình, để đào tạo hoàn chỉnh nghề Công nghệ thông tin (Ứng dụng phần mềm) ở cấp trình độ cao đẳng nghề và được dùng làm Giáo trình cho học sinh sinh viên trong các khóa đào tạo, cũng có thể được sử dụng cho đào tạo ngắn hạn hoặc cho đào tạo công nhân kỹ thuật, các nhà quản lý và lập trình viên tham khảo.

Đà Lạt, ngày 07 tháng 7 năm 2017

Tham gia biên soạn

1. Chủ biên Ngô Thiên Hoàng
2. Phạm Đình Nam
3. Trương Thị Thanh Thảo
4. Nguyễn Quỳnh Nguyên
5. Phan Ngọc Bảo

MỤC LỤC

TRANG

Tên môn học: NGÔN NGỮ JAVA

Mã môn học: MH29

I. Vị trí, tính chất của môn học:

1. Vị trí: được bố trí giảng sau môn học: Lập trình căn bản.

2. Tính chất: Là môn học tự chọn áp dụng cho trình độ Cao đẳng Công nghệ thông tin (ứng dụng phần mềm).

II. Mục tiêu môn học:

1. Về kiến thức:

- Xác định rõ đặc điểm, môi trường phát triển - hoạt động, khả năng ứng dụng của ngôn ngữ Java;
- Trình bày được qui trình biên soạn, biên dịch và thực thi một ứng dụng Java;
- Trình bày được khái niệm, tính năng, cách sử dụng của các nền tảng của Java như các kiểu dữ liệu, các cấu trúc điều khiển, kỹ thuật hướng đối tượng,... của Java;
- Liệt kê được tính năng, đặc điểm của thư viện đồ họa AWT cũng như cách xử lý các biến cố trong giao diện ứng dụng;
- Trình bày được về khái niệm, cách xử lý với các luồng vào ra dữ liệu, tập tin;
- Trình bày được cách kết nối ứng dụng Java với cơ sở dữ liệu và các thao tác tương tác dữ liệu: cập nhật, truy vấn,....

2. Về kỹ năng:

- Viết và thực thi được các chương trình ứng dụng Java xử lý một số yêu cầu đơn giản;
- Thiết kế được các ứng dụng với các giao diện đồ họa đẹp, thân thiện với các tính năng thông dụng, kết nối được và tương tác được trên các hệ cơ sở dữ liệu: thêm, xem, xóa,... dữ liệu;

3. Về năng lực tự chủ và trách nhiệm:

- Có khả năng tự nghiên cứu, tự học, tham khảo tài liệu liên quan đến môn học để vận dụng vào hoạt động hoc tập.
- Vận dụng được các kiến thức tự nghiên cứu, học tập và kiến thức, kỹ năng đã được học để hoàn thiện các kỹ năng liên quan đến môn học một cách khoa học, đúng quy định.

III. Nội dung môn học:

Chương 1: GIỚI THIỆU TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH JAVA

Mã bài : MH 29.1

Mục tiêu:

- Trình bày khái niệm về Java: bản chất, lịch sử phát triển;
- Nêu được một số ứng dụng cơ bản, kiểu chương trình Java;
- Mô tả được môi trường phát triển (JDK), công cụ soạn thảo Java;
- Cài đặt được môi trường phát triển, công cụ soạn thảo, máy ảo Java;
- Thái độ nghiêm túc, cẩn thận.

1. Giới thiệu về ngôn ngữ lập trình Java

1.1 Java là gì?

Java là ngôn ngữ lập trình hướng đối tượng (tựa C++) do Sun Microsystem đưa ra vào giữa thập niên 90.

Chương trình viết bằng ngôn ngữ lập trình java có thể chạy trên bất kỳ hệ thống nào có cài máy ảo java (Java Virtual Machine).

1.2 Lịch sử phát triển của ngôn ngữ lập trình Java

Ngôn ngữ lập trình Java do James Gosling và các công sự của Công ty Sun Microsystem phát triển.

Đầu thập niên 90, Sun Microsystem tập hợp các nhà nghiên cứu thành lập nên nhóm đặt tên là Green Team. Nhóm Green Team có trách nhiệm xây dựng công nghệ mới cho ngành điện tử tiêu dùng. Để giải quyết vấn đề này nhóm nghiên cứu phát triển đã xây dựng một ngôn ngữ lập trình mới đặt tên là Oak tương tự như C++ nhưng loại bỏ một số tính năng nguy hiểm của C++ và có khả năng chạy trên nhiều nền phần cứng khác nhau. Cùng lúc đó world wide web bắt đầu phát triển và Sun đã thấy được tiềm năng của ngôn ngữ Oak nên đã đầu tư cải tiến và phát triển. Sau đó không lâu ngôn ngữ mới với tên gọi là Java ra đời và được giới thiệu năm 1995.

Java là tên gọi của một hòn đảo ở Indonexia, Đây là nơi nhóm nghiên cứu phát triển đã chọn để đặt tên cho ngôn ngữ lập trình Java trong một chuyến đi tham quan và làm việc trên hòn đảo này. Hòn đảo Java này là nơi rất nổi tiếng với nhiều khu vườn trồng cafe, đó chính là lý do chúng ta thường thấy biểu tượng ly café trong nhiều sản phẩm phần mềm, công cụ lập trình Java của Sun cũng như một số hãng phần mềm khác đưa ra.

1.3 Một số đặc điểm nổi bậc của Java

Máy ảo Java (JVM - Java Virtual Machine).

Tất cả các chương trình muốn thực thi được thì phải được biên dịch ra mã máy. Mã máy của từng kiến trúc CPU của mỗi máy tính là khác nhau (tập lệnh mã máy của CPU Intel, CPU Solarix, CPU Macintosh … là khác nhau), vì vậy trước đây một chương trình sau khi được biên dịch xong chỉ có thể chạy được trên một kiến trúc CPU cụ thể nào đó. Đối với CPU Intel chúng ta có thể chạy các hệ điều hành như Microsoft Windows, Unix, Linux, OS/2, … Chương trình thực thi được trên Windows được biên dịch dưới dạng file có đuôi .EXE còn trên Linux thì được biên dịch dưới dạng file có đuôi .ELF, vì vậy trước đây một chương trình chạy được trên Windows muốn chạy được trên hệ điều hành khác như Linux chẳng hạn thì phải chỉnh sửa và biên dịch lại. Ngôn ngữ lập trình Java ra đời, nhờ vào máy ảo Java mà khó khăn nêu trên đã được khắc phục. Một chương trình viết bằng ngôn ngữ lập trình Java sẽ được biên dịch ra mã của máy ảo java (mã java bytecode). Sau đó máy ảo Java chịu trách nhiệm chuyển mã java bytecode thành mã máy tương ứng. Sun Microsystem chịu trách nhiệm phát triển các máy ảo Java chạy trên các hệ điều hành trên các kiến trúc CPU khác nhau.

Thông dịch:

Java là một ngôn ngữ lập trình vừa biên dịch vừa thông dịch. Chương trình nguồn viết bằng ngôn ngữ lập trình Java có đuôi *.java đầu tiên được biên dịch thành tập tin có đuôi *.class và sau đó sẽ được trình thông dịch thông dịch thành mã máy.

Độc lập nền:

Một chương trình viết bằng ngôn ngữ Java có thể chạy trên nhiều máy tính có hệ điều hành khác nhau (Windows, Unix, Linux, …) miễn sao ở đó có cài đặt máy ảo java (Java Virtual Machine). Viết một lần chạy mọi nơi (write once run anywhere).

Hướng đối tượng:

Hướng đối tượng trong Java tương tự như C++ nhưng Java là một ngôn ngữ lập trình hướng đối tượng hoàn toàn. Tất cả mọi thứ đề cập đến trong Java đều liên quan đến các đối tượng được định nghĩa trước, thậm chí hàm chính của một chương trình viết bằng Java (đó là hàm main) cũng phải đặt bên trong một lớp. Hướng đối tượng trong Java không có tính đa kế thừa (multi inheritance) như trong C++ mà thay vào đó Java đưa ra

khái niệm interface để hỗ trợ tính đa kế thừa. Vấn đề này sẽ được bàn chi tiết trong chương 3.

Đa nhiệm - đa luồng (MultiTasking - Multithreading):

Java hỗ trợ lập trình đa nhiệm, đa luồng cho phép nhiều tiến trình, tiểu trình có thể chạy song song cùng một thời điểm và tương tác với nhau.

Khả chuyển (portable):

Chương trình ứng dụng viết bằng ngôn ngữ Java chỉ cần chạy được trên máy ảo Java là có thể chạy được trên bất kỳ máy tính, hệ điều hành nào có máy ảo Java. "Viết một lần, chạy mọi nơi" (Write Once, Run Anywhere).

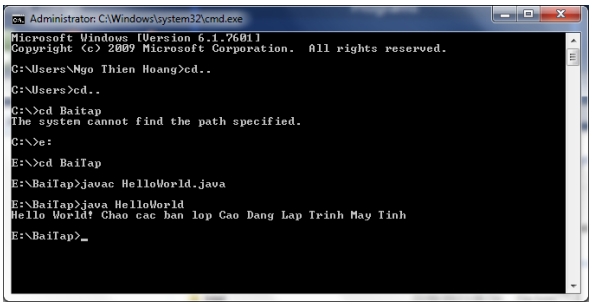
Hỗ trợ mạnh cho việc phát triển ứng dụng:

Công nghệ Java phát triển mạnh mẽ nhờ vào "đại gia Sun Microsystem" cung cấp nhiều công cụ, thư viện lập trình phong phú hỗ trợ cho việc phát triển nhiều loại hình ứng dụng khác nhau cụ thể như: J2SE (Java 2 Standard Edition) hỗ trợ phát triển những ứng dụng đơn, ứng dụng client-server; J2EE (Java 2 Enterprise Edition) hỗ trợ phát triển các ứng dụng thương mại, J2ME (Java 2 Micro Edition) hỗ trợ phát triển các ứng dụng trên các thiết bị di động, không dây, …

2. Các ứng dụng Java

2.1. Java và ứng dụng Console

Ứng dụng Console là ứng dụng nhập xuất ở chế độ văn bản tương tự như màn hình Console của hệ điều hành MS-DOS. Loại chương trình ứng dụng này thích hợp với những ai bước đầu làm quen với ngôn ngữ lập trình java. Các ứng dụng kiểu Console thường được dùng để minh họa các ví dụ cơ bản liên quan đến cú pháp ngôn ngữ, các thuật toán, và các chương trình ứng dụng không cần thiết đến giao diện người dùng đồ họa.



Hình 1.1 Ứng dụng Console

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

System.out.print("Hello World! Chao cac ban lop Cao Dang Lap Trinh May Tinh\n");

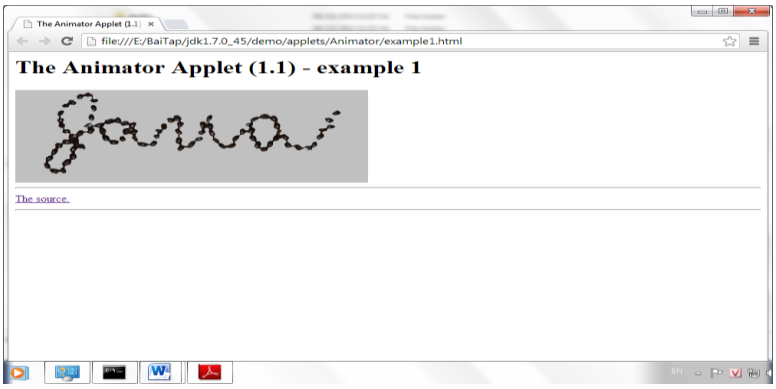
}

}

2.2 Java và ứng dụng Applet

Java Applet là loại ứng dụng có thể nhúng và chạy trong trang web của một trình duyệt web. Từ khi internet mới ra đời, Java Applet cung cấp một khả năng lập trình mạnh mẽ cho các trang web. Nhưng gần đây khi các chương trình duyệt web đã phát

triển với khả năng lập trình bằng VB Script, Java Script, HTML, DHTML, XML, … cùng với sự canh tranh khốc liệt của Microsoft và Sun đã làm cho Java Applet lu mờ. Và cho đến bây giờ gần như các lập trình viên đều không còn "mặn mà" với Java Applet nữa. (trình duyệt IE đi kèm trong phiên bản Windows 2000 đã không còn mặc nhiên hỗ trợ thực thi một ứng dụng Java Applet). Hình bên dưới minh họa một chương trình java applet thực thi trong một trang web.



Hình 1.2 Ứng dụng Applet

2.3 . Java và phát triển ứng dụng Desktop dùng AWT và JFC

Việc phát triển các chương trình ứng dụng có giao diện người dùng đồ họa trực quan giống như những chương trình được viết dùng ngôn ngữ lập trình VC++ hay Visual Basic đã được java giải quyết bằng thư viện AWT và JFC. JFC là thư viện rất phong phú và hỗ trợ mạnh mẽ hơn nhiều so với AWT. JFC giúp cho người lập trình có thể tạo ra một giao diện trực quan của bất kỳ ứng dụng nào. Liên quan đến việc phát triển các ứng dụng

có giao diện người dùng đồ họa trực quan chúng ta sẽ tìm hiểu chi tiết trong chương 4.

2.4. Java và phát triển ứng dụng Web

Java hỗ trợ mạnh mẽ đối với việc phát triển các ứng dụng Web thông qua công nghệ J2EE (Java 2 Enterprise Edition). Công nghệ J2EE hoàn toàn có thể tạo ra các ứng dụng Web một cách hiệu quả không thua kém công nghệ .NET mà Microsft đang quảng cáo.

Hiện nay có rất nhiều trang Web nổi tiếng ở Việt Nam cũng như khắp nơi trên thế giới được xây dựng và phát triển dựa trên nền công nghệ Java. Số ứng dụng Web được xây dựng dùng công nghệ Java chắc chắn không ai có thể biết được con số chính xác là bao nhiêu, nhưng chúng tôi đưa ra đây vài ví dụ để thấy rằng công nghệ Java của Sun là một "đối thủ đáng gờm" của Microsoft.

http://java.sun.com/

http://e-docs.bea.com/

http://www.macromedia.com/software/jrun/

http://tomcat.apache.org/index.html

Chắc không ít người trong chúng ta biết đến trang web thông tin nhà đất nổi tiếng ở TPHCM đó là: http://www.nhadat.com/. Ứng dụng Web này cũng được xây dựng dựa trên nền công nghệ java. Bạn có thể tìm hiểu chi tiết hơn về công nghệ J2EE tạo địa chỉ:

http://java.sun.com/j2ee/

2.5. Java và phát triển các ứng dụng nhúng

Java Sun đưa ra công nghệ J2ME (The Java 2 Platform, MicroEdition J2ME) hỗ trợ phát triển các chương trình, phần mềm nhúng. J2ME cung cấp một môi trường cho những chương trình ứng dụng có thể chạy được trên các thiết bị cá nhân như: điện thoại di động, máy tính bỏ túi PDA hay Palm, cũng như các thiết bị nhúng khác. Bạn có thể tìm hiểu chi tiết hơn về công nghệ J2ME tại địa chỉ: http://java.sun.com/j2me/

3. Dịch và thực thi một chương trình viết bằng Java

Việc xây dựng, dịch và thực thi một chương trình viết bằng ngôn ngữ lập trình java có thể tóm tắt qua các bước sau:

- Viết mã nguồn: dùng một chương trình soạn thảo nào đấy (NotePad hay Jcreator chẳng hạn) để viết mã nguồn và lưu lại với tên có đuôi ".java"

- Biên dịch ra mã máy ảo: dùng trình biên dịch javac để biên dịch mã nguồn ".java" thành mã của máy ảo (java bytecode) có đuôi ".class" và lưu lên đĩa

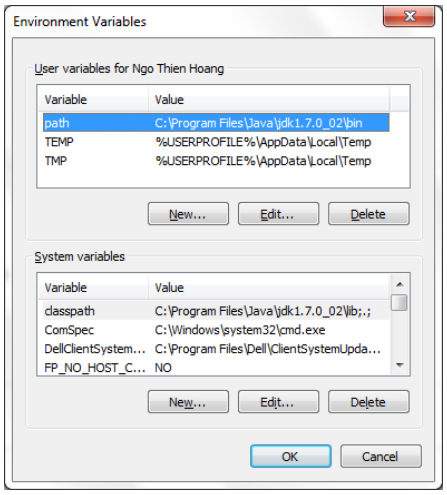
- Thông dịch và thực thi: ứng dụng được load vào bộ nhớ, thông dịch và thực thi dùng trình thông dịch Java thông qua lệnh "java".

- Đưa mã java bytecode vào bộ nhớ: đây là bước "loading". Chương trình phải được đặt vào trong bộ nhớ trước khi thực thi. "Loader" sẽ lấy các files chứa mã java bytecode có đuôi ".class" và nạp chúng vào bộ nhớ.
- Kiểm tra mã java bytecode: trước khi trình thông dịch chuyển mã bytecode thành mã máy tương ứng để thực thi thì các mã bytecode phải được kiểm tra tính hợp lệ.
- Thông dịch & thực thi: cuối cùng dưới sự điều khiển của CPU và trình thông dịch tại mỗi thời điểm sẽ có một mã bytecode được chuyển sang mã máy và thực thi.

4. Công cụ lập trình và chương trình dịch

4.1. JDK7

Download JDK phiên bản mới nhất tương ứng với hệ điều hành đang sử dụng từ địa chỉ java.sun.com và cài đặt lên máy tính (phiên bản được chúng tôi sử dụng khi viết giáo trình này là JDK 1.7.0). Sau khi cài xong, chúng ta cần cập nhật đường dẫn PATH hệ thống chỉ đến thư mục chứa chương trình dịch của ngôn ngữ java.



Hình 1.3 Cập nhật đường dẫn

4.2.Công cụ soạn thảo mã nguồn Java.

Để viết mã nguồn java chúng ta có thể sử dụng trình soạn thảo NotePad hoặc một số môi trường phát triển hỗ trợ ngôn ngữ java như: Jbuilder của hãng Borland, Visual Café của hãng Symantec, JDeveloper của hãng Oracle, Visual J++ của Microsoft, …

Trong khuôn khổ giáo trình này cũng như để hướng dẫn sinh viên thực hành chúng tôi dùng công cụ JCreator LE của hãng XINOX Software. Các bạn có

thể download

JCreator từ http://www.jcreator.com/download.htm.

5. Bài tập

Bài tập 1:.Trình bày khả năng của ngôn ngữ lập trình Java

Bài tập 2:.Hãy nêu những đặc điểm của ngôn ngữ lập trình Java

Bài tập 3:Trình bày các ứng dụng của ngôn ngữ lập trình Java

Bài tập 4: Download JDK phiên bản mới nhất tương ứng với hệ điều hành đang sử dụng từ địa chỉ java.sun.com và cài đặt lên máy tính. Thực thi chương trình HelloWorld.

CHƯƠNG 2 NỀN TẢNG CỦA JAVA

Mã bài : MH 29.2

Mục tiêu:

– Trình bày được khái niệm cơ bản về ngôn ngữ Java: tập kí tự, từ khóa, cấu trúc chương trình, các kiểu dữ liệu, các toán tử, biến, hằng,...;

– Xác định được các cấu trúc điều khiển cơ bản của ngôn ngữ lập trình Java;

– Đọc hiểu và thực thi một số chương trình đầu tiên viết bằng Java;

– Viết một số chương trình Java thực hiện các yêu cầu đơn giản;

– Nghiêm túc, cẩn thận, sáng tạo trong học lý thuyết và thực hành.

2.1 Tập ký tự, từ khóa, định danh

2.1.1 Ký hiệu cơ sở

Ngôn ngữ Java được xây dựng từ bộ ký hiệu cơ sở sau:

- Bộ 26 chữ cái La-Tinh viết thường (nhỏ): a,b,...,z.
- Bộ 26 chữ cái La-Tinh viết hoa (lớn): A,B,...,Z.
- Bộ 10 chữ số hệ thập phân : 0,1,...,9.
- Bộ dấu các toán tử số học : + - * /
- Bộ dấu các toán tử so sánh: < > =
- Ký tự gạch nối: _ (Khác dấu trừ -).
- Các ký hiệu khác: ' " ; ,.: [ ] # $ & { } % ! . . .

Đặc biệt có khoảng trắng dùng để ngăn cách các từ (phím Space). Các ký hiệu cơ sở đều có trên bàn phím.

2.1.2 Các từ

Từ trong Java được xây dựng bởi các ký hiệu cơ sở trên. Có 2 loại từ: Từ khóa và tên.

a. Từ khóa (Key Word)

Là những từ có ý nghĩa hoàn toàn xác định, chúng thường được dùng để khai báo các kiểu dữ liệu, để viết các toán tử, và các câu lệnh. Sau đây là một số từ khóa trong Java:

asm	aut	bre	cas	catc	cha
clas	con	con	def	dele	do
dou	else	enu	exte	floa	for
frie	got	if	Inli	int	lon
new	ope	priv	Prot	pub	regi

retu	sho	sign	Siz	stati	stru
swit	tem	this	thro	try	typ
uni	unsi	virt	voi	vol	whi
cde	_cs	_ds	_es	_ex	far
hug	inte	_lo	Nea	pas	_re
_sa	_se	_ss			

b. Tên hoặc danh hiệu (identifier):

Là từ do người sử dụng tự đặt để giải quyết bài toán của mình. Từ tự đặt dùng để đặt tên cho hằng, biến, hàm, tên kiểu dữ liệu mới,...

Tên được đặt theo quy tắc: phải bắt đầu bằng một chữ cái hoặc dấu gạch nối,sau đó là các chữ cái, chữ số hoặc dấu gạch nối, và không được trùng với từ khóa.

Tên có thể viết bằng chữ thường hoặc chữ hoa. Trong java có phân biệt chữ thường và chữ hoa.

2.2 Cấu trúc của một chương trình Java

Java là một ngôn ngữ thuần đối tượng (pure object). Tất cả các thành phần được khai báo nhờ hằng, biến, hàm thủ tục đều phải nằm trong phạm vi của một lớp nào đó. Một ứng dụng trong Java là một tập hợp các lớp liên quan nhau, bao gồm các lớp trong thư viện do Java cung cấp và các lớp được định nghĩa bởi ngọời lập trình. Trong một ứng dụng chỉ có một Lớp thực thi được. Đây là lớp đầu tiên được xem xét đến khi chúng ta thực thi ứng dụng.

Lớp thực thi đọợc này có các đặc điểm sau:

- Có tên lớp trùng với tên tập tin chứa nó.
- Phải khai báo phạm vi là public
- Có chứa phương thức:

public static void main (String args[]){

. . .

}

là phương thức được thực thi đầu tiên.

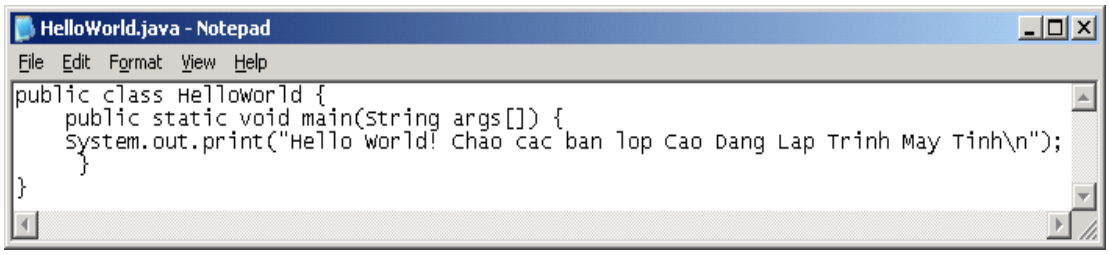
Nếu nhiều lớp được định nghĩa trong một tập tin, chỉ có một lớp được khai báo public.

2.3 Chương trình java đầu tiên

2.3.1. Chương trình HelloWorld

Trong ví dụ này, chúng ta viết một chương trình ứng dụng in ra màn hình

dòng chữ "Hello World !". Đây là ứng dụng đơn giản chỉ có một lớp thực thi được tên là HelloWorld. Lớp này được khai báo là public, có phương thức main(), chứa trong tập tin cùng tên là HelloWorld.java (phần mở rộng bắt buộc phải là .java).



Hình 2.1 Dùng Notepad biên soạn tập tin cùng tên là HelloWorld.java

Phương thức System.out.print() sẽ in tất cả các tham số trong dấu () của nó ra màn hình.

Ta có thể dùng bất kỳ chương trình sọn thảo nào để biên soạn chương trình, nhưng chú ý phải ghi lại chương trình với phần mở rộng là .java

2.3.2. Biên soạn chương trình bằng phần mềm Notepad của Ms Windows

Notepad là trình soạn thảo đơn giản có sẵn trong MS Windows mà ta có thể dùng để biên soạn chương trình HelloWorld. Hãy thực hiện các bước sau:

- Chạy chương trình Notepad:

+ Chọn menu Start \ Programs \ Accessories \ Notepad

- Nhập nội dung sau vào Notepad

```
public class HelloWorld {
   public static void main(String args[]) {
   System.out.print("Hello World! Chao cac ban lop Cao Dang Lap Trinh May Tinh\n");
    }
}
```

- Save tập tin với tên HelloWorld.java

+ Chọn menu File \ Save

+ Tại cửa sổ Save As hãy nhập vào:

- Save in: Thư mục nơi sẽ lưu tập tin
- File Name: HelloWorld.java
- Save as type: All Files

- Nhấp vào nút Save

2.3.4. Biên dịch và thực thi chương trình

- Mở cửa sổ MS-DOS: Chọn menu Start \ Programs \ Accessories \ Command Prompt.
- Chuyển vào thư mục chứa tập tin HelloWorld.java
- Dùng chương trình javac để biên dịch tập tin HelloWorld.java

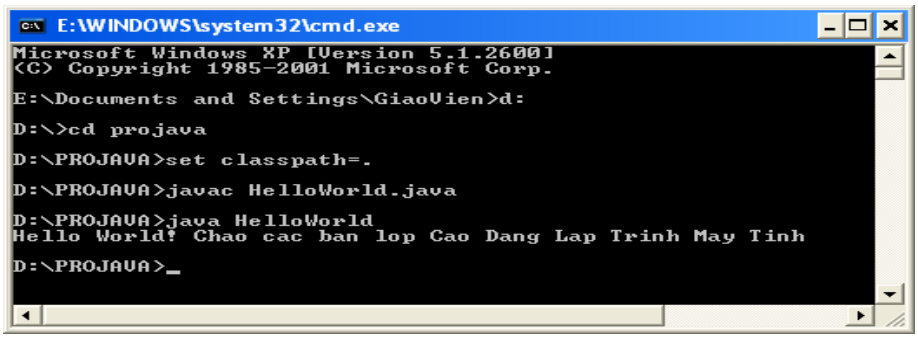
```
javac  HelloWorld.java
```

+ Nếu có lỗi, trên màn hình sẽ xuất hiện thông báo lỗi với dấu ^ chỉ vị trí lỗi.

+ Nếu không có lỗi, tập tin thực thi HelloWorld.class được tạo ra.

- Thực thi chương trình HelloWorld.class

```
java HelloWorld
```



Hình 2.2 Kết quả thực thi chương trình HelloWorld

Trên màn hình sẽ xuất hiện dòng chữ Hello World! Chao cac ban lop Cao Dang Lap Trinh May Tinh

2.4 Biến, hằng

2.4.1. Biến

- Biến là vùng nhớ dùng để lưu trữ các giá trị của chương trình. Mỗi biến gắn liền với một kiểu dữ liệu và một định danh duy nhất gọi là tên biến.

- Tên biến thông thường là một chuỗi các ký tự (Unicode), ký số.

- Tên biến phải bắt đầu bằng một chữ cái, một dấu gạch dưới hay dấu dollar.
- Tên biến không được trùng với các từ khóa (xem phụ lục các từ khóa trong java).
- Tên biến không có khoảng trắng ở giữa tên.

- Trong java, biến có thể được khai báo ở bất kỳ nơi đâu trong chương

trình.

Cách khai báo

<kiểu_dữ_liệu> <tên_biến>;

<kiểu_dữ_liệu> <tên_biến> = <giá_trị>;

Gán giá trị cho biến

<tên_biến> = <giá_trị>;

Biến công cộng (toàn cục): là biến có thể truy xuất ở khắp nơi trong chương trình, thường được khai báo dùng từ khóa public, hoặc đặt chúng trong một class.

Biến cục bộ: là biến chỉ có thể truy xuất trong khối lệnh nó khai báo.

Lưu ý: Trong ngôn ngữ lập trình java có phân biệt chữ in hoa và in thường. Vì vậy chúng ta cần lưu ý khi đặt tên cho các đối tương dữ liệu cũng như các xử lý trong chương trình.

Ví dụ:

```
Import java.lang.*;
import java.io.*;
class VariableDemo
{
static int x, y;
public static void main(String[] args)
{
x = 10;
y = 20;
int z = x+y;
System.out.println("x = " + x);
System.out.println("y = " + y);
System.out.println("z = x + y =" + z);
System.out.println("So nho hon la so:" + Math.min(x, y));
char c = 80;
System.out.println("ky tu c la: " + c);
}
}
```

2.4.2. Hằng

- Hằng là một giá trị bất biến trong chương trình

- Tên hằng được đặt theo qui ước giống như tên biến.

- Hằng số nguyên: trường hợp giá trị hằng ở dạng long ta thêm vào cuối chuỗi số chữ “l” hay “L”. (ví dụ: 1L)

- Hằng số thực: trường hợp giá trị hằng có kiểu float ta thêm tiếp vĩ ngữ “f” hay “F”, còn kiểu số double thì ta thêm tiếp vĩ ngữ “d” hay “D”.

- Hằng Boolean: java có 2 hằng boolean là true, false.

- Hằng ký tự: là một ký tự đơn nằm giữa nằm giữa 2 dấu ngoặc đơn.

o Ví dụ: ‘a’: hằng ký tự a

o Một số hằng ký tự đặc biệt

Ký tự	Ý nghĩa
\b	Xóa lùi
\t	Tab
\n	Xuống hàng
\r	Dấu enter
\”	Nháy kép
\’	Nháy đơn
\\	Số ngược
\f	Đẩy trang
\uxxxx	Ký tự unicode

- Hằng chuỗi: là tập hợp các ký tự được đặt giữa hai dấu nháy kép “”. Một hằng chuỗi không có ký tự nào là một hằng chuỗi rỗng.

- Ví dụ: “Hello Wolrd”
- Lưu ý: Hằng chuỗi không phải là một kiểu dữ liệu cơ sở nhưng vẫn được khai báo và sử dụng trong các chương trình.

2.5 Các kiểu dữ liệu cơ sở

- **Kiểu số**

- Java cung cấp 4 kiểu số nguyên khác nhau là: byte,short, int, long. Kích thước, giá trị nhỏ nhất, lớn nhất, cũng như giá trị mặc định của các kiểu dữ liệu số nguyên được mô tả chi tiết trong bảng.

- Kiểu mặc định của các số nguyên là kiểu int.

- Các số nguyên kiểu byte và short rất ít khi được dùng.

- Trong java không có kiểu số nguyên không dấu như

trong ngôn ngữ C/C++.

Tên	Kích thước
byte	1 byte
short	2 bytes
int	4 bytes
long	8 bytes
float	4 bytes
double	8 bytes

Một số lưu ý đối với các phép toán trên số nguyên:

- Nếu hai toán hạng kiểu long thì kết quả là kiểu long.

Một trong hai toán hạng không phải kiểu long sẽ được chuyển thành kiểu long trước khi thực hiện phép toán.

- Nếu hai toán hạng đầu không phải kiểu long thì phép tính sẽ thực hiện với kiểu int.

- Các toán hạng kiểu byte hay short sẽ được chuyển sang kiểu int trước khi thực hiện phép toán.

- Trong java không thể chuyển biến kiểu int và kiểu boolean như trong ngôn ngữ C/C++.

- **Kiểu dấu chấm động**

Đối với kiểu dấu chấm động hay kiểu thực, java hỗ trợ hai kiểu dữ liệu là float và double.

Kiểu float có kích thước 4 byte và giá trị mặc định là 0.0f

Kiểu double có kích thước 8 byte và giá trị mặc định là 0.0d

Số kiểu dấu chấm động không có giá trị nhỏ nhất cũng không có giá trị lớn nhất. Chúng có thể nhận các giá trị:

- Số âm

- Số dương

- Vô cực âm

- Vô cực dương

Khai báo và khởi tạo giá trị cho các biến kiểu dấu chấm động:
float x = 100.0/7;

double y = 1.56E6;

Một số lưu ý đối với các phép toán trên số dấu chấm động:

- Nếu mỗi toán hạng đều có kiểu dấn chấm động thì phép

toán chuyển thành phép toán dấu chấm động.

- Nếu có một toán hạng là double thì các toán hạng còn lại sẽ được chuyển thành kiểu double trước khi thực hiện phép toán.

- Biến kiểu float và double có thể ép chuyển sang kiểu dữ liệu khác trừ kiểu boolean.

- **Kiểu ký tự char**

Java dùng 2 bytes cho kiểu ký tự, theo chuẩn mã UNICODE (127 ký tự đầu tương thích với mã ASCII). Do đó, ta sử dụng tương tự như bảng mã ASCII.

Kiểu ký tự trong ngôn ngữ lập trình java có kích thước là 2 bytes và chỉ dùng để biểu diễn các ký tự trong bộ mã Unicode. Như vậy kiểu char trong java có thể biểu diễn tất cả 216 = 65536 ký tự khác nhau.

Giá trị mặc định cho một biến kiểu char là null.

- **Kiểu chuỗi ký tự String**

Thực chất đây là một lớp nằm trong thư viện chuẩn của Java (Core API), java.lang.String

Kiểu chuỗi là tập hợp các ký tự được đặt giữa hai dấu nháy kép “”. Một hằng chuỗi không có ký tự nào là một hằng chuỗi rỗng.

- Ví dụ: “Hello Wolrd”
- Lưu ý: Hằng chuỗi không phải là một kiểu dữ liệu cơ sở nhưng vẫn được khai báo và sử dụng trong các chương trình.

- **Kiểu luận lý boolean**

Nhận 2 giá trị là : true và false.

- Trong java kiểu boolean không thể chuyển thành kiểu nguyên và ngược lại.

- Giá trị mặc định của kiểu boolean là false.

- **Kiểu mảng**

  - Khai báo:

  + int[] a ; float[] yt; String[] names;

  + hoặc: int a[]; float yt[]; String names[];int maTran[][]; float bangDiem[][];

  - Khởi tạo:

  + a = new int[3]; yt = new float[10]; names = new String[50];

  + maTran = int[10][10];

- Sử dụng mảng:

+ int i = a[0]; float f = yt[9]; String str = names[20]; int x = matran [2][5];

2.6 Lệnh, khối lệnh trong java

Giống như trong ngôn ngữ C, các câu lệnh trong java kết thúc bằng một dấu chấm phẩy (;).

Một khối lệnh là đoạn chương trình gồm hai lệnh trở lên và được bắt đầu bằng dấu mở ngoặc nhọn ({) và kết thúc bằng dấu đóng ngoặc nhọc (}).

Bên trong một khối lệnh có thể chứa một hay nhiều lệnh hoặc chứa các khối lệnh khác.

```
{ // khối 1
    { // khối 2
      lệnh 2.1
      lệnh 2.2 …
     } // kết thúc khối lệnh 2
      lệnh 1.1
      lệnh 1.2
       …
} // kết thúc khối lệnh 1
{ // bắt đầu khối lệnh 3
// Các lệnh thuộc khối lệnh 3
// …
} // kết thúc thối lệnh 3
```

2.7 Toán tử và biểu thức

2.7.1.Toán tử số học

Toá	Ý nghĩa
+	Cộng
-	Trừ
	Nhân
/	Chia nguyên
%	Chia dư
++	Tăng 1
--	Giảm 1

2.7.2.Toán tử trên bit

Toán	Ý nghĩa
&	AND
|	OR
^	XOR
<<	Dịch trái
>>	Dịch phải
>>>	Dịch phải và điền 0 vào bit
~	Bù bit

27.3.Toán tử quan hệ & logic

Toán tử	Ý nghĩa
==	So sánh bằng
!=	So sánh khác
>	So sánh lớn hơn
<	So sánh nhỏ hơn
>=	So sánh lớn hơn hay bằng
<=	So sánh nhỏ hơn hay bằng
||	OR (biểu thức logic)
&&	AND (biểu thức logic)
!	NOT (biểu thức logic)

2.7.4.Toán tử ép kiểu

- Ép kiểu rộng (widening conversion): từ kiểu nhỏ sang kiểu lớn (không mất mát thông tin)
- Ép kiểu hẹp (narrow conversion): từ kiểu lớn sang kiểu nhỏ (có khả năng mất mát thông tin)

<tên biến> = (kiểu_dữ_liệu) <tên_biến>;

Ví dụ float fNum = 2.2;

int iCount = (int) fNum; // (iCount = 2)

2.7.5.Toán tử điều kiện

Cú pháp: <điều kiện> ? <biểu thức 1> : < biểu thức 2>

Nếu điều kiện đúng thì có giá trị, hay thực hiện <biểu thức 1>, còn ngược

lại là <biểu thức 2>.

<điều kiện>: là một biểu thức logic

<biểu thức 1>, <biểu thức 2>: có thể là hai giá trị, hai biểu thức hoặc hai hành động.

Ví dụ: int x = 10;

int y = 20;

int Z = (x<y) ? 30 : 40;

// Kết quả z = 30 do biểu thức (x < y) là đúng.

2.7.6 Thứ tự ưu tiên tính từ trái qua phải và từ trên xuống dưới

Ca			
()	[]		
++	--	~	!
	/	%	
+	-		
>>	>>> *(dịch*	<<	
>	>=	<	<
==	!=		
&			
^			
|			
&&			
||			
?:			
=	<toántử>		
Thấ			

2.8 Cấu trúc điều khiển

2.8.1. Lệnh if – else

Cú pháp:

```
if (Condition) {
// Các lệnh sẽ được thực hiện nếu giá trị của Condition là true
}
if (Condition) {
// Các lệnh sẽ được thực hiện nếu giá trị của Condition là true
} else {
// Các lệnh sẽ được thực hiện nếu giá trị của Condition là false
}
```

Ví dụ:

Lưu chương trình sau vào tập tin IfDemo.java :

```
import java.io.*;
public class IfDemo {
public static void main(String args[]) {
System.out.print("Vui long nhap mot ky tu:");
try {
int ch = System.in.read();
if (ch == 'A') {
System.out.print("Ban rat may man !");
}
else {
System.out.print("Ban khong gap may !");
}
} catch(IOException ie) { System.out.print("Error:"+ie);
}
}
}
```

Biên dich và thực thi có kết quả như sau:

Hình 2.3 Kết quả thực thi tập tin IfDemo.java

2.8.2. Phép toán ?

Cú pháp:

(condition) ? Operation1 : Operation2;

Nếu điều kiện condition có giá trị là true lệnh sẽ trả về giá trị của biểu thức

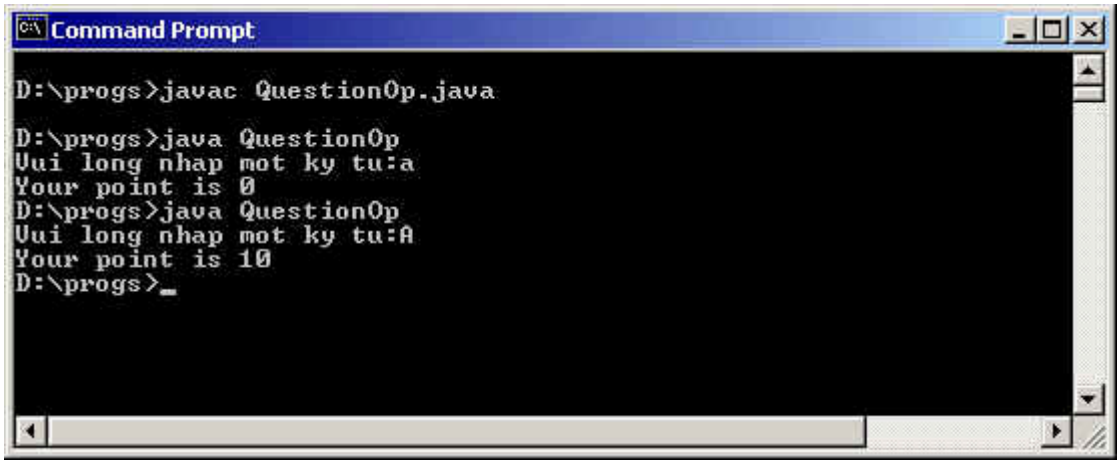
Operation1, ngược lại sẽ trả về giá trị của biểu thức Operation2.

Ví dụ:

Lưu chương trình sau vào tập tin QuestionOp.java :

```
import java.io.*;
public class QuestionOp {
public static void main(String args[]) {
System.out.print("Vui long nhap mot ky tu:");
try {
int ch = System.in.read();
int point = (ch == 'A') ? 10:0;
System.out.print("Diem cua ban la:"+point);
}
catch(IOException ie) { System.out.print("Error:"+ ie);
    }
  }
}
```

Biên dịch và thực thi được kết quả như sau:



Hình 2.4 Kết quả thực thi tập tin QuestionOp.java

2.8.3. Lệnh switch

Cú pháp

```
switch ( variable ) {
case value1 : {
Task 1;
// Các tác vụ sẽ được thực thi nếu giá trị của variable là value1 break;
}
```

case value2 : {

Task 2;

// Các tác vụ sẽ được thực thi nếu giá trị của variable là value2 break;

}

. . .

default:

Task n;

// Tác vụ sẽ được thực thi nếu giá trị của variable không là các giá trị trên

}

Ví dụ

Lưu chương trình sau vào tập tin CaseOp.java

```
import java.io.*;
public class CaseOp {
public static void main(String args[])
{
System.out.print("Enter a number character: ");
try {
int ch = System.in.read();
switch(ch) {
case '0': {
System.out.print("Zero"); break;}
case '1': { System.out.print("One"); break;}
case '2': { System.out.print("Two"); break;}
case '3': { System.out.print("Three");break;}
case '4': { System.out.print("Four"); break;}
case '5': { System.out.print("Five"); break;}
case '6': { System.out.print("Six"); break;}
case '7': { System.out.print("Seven");break;}
case '8': { System.out.print("Eight");break;}
case '9': { System.out.print("Nine"); break;}
default: { System.out.print("I don't know"); break;}
}
```

```
}
catch(IOException ie) { System.out.print("Error "+ie);
  }
 }
}
```

Biên dịch và thực thi được kết quả sau:

Hình 2.5 Kết quả thực thi tập tin CaseOp.java

2.8.4. Lệnh while

Cú pháp

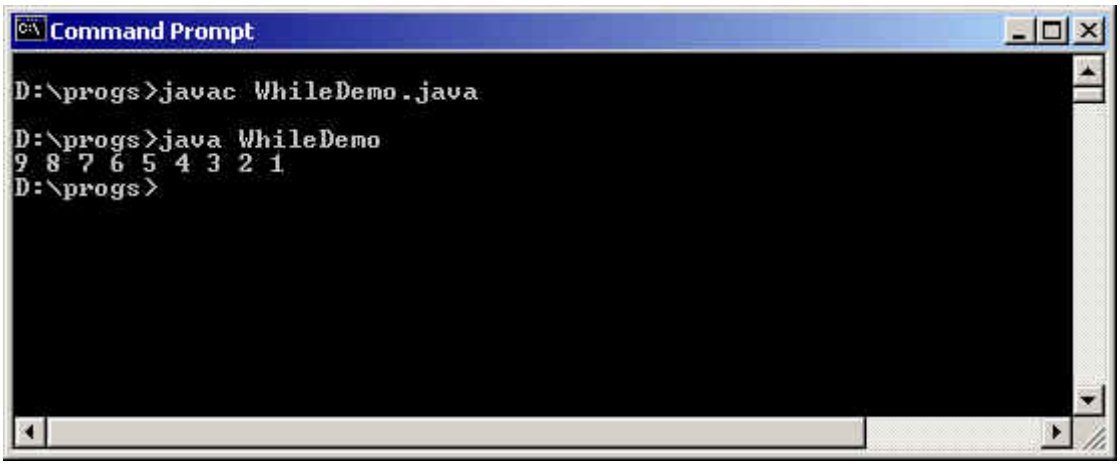
```
while (condition) {
// nếu condition có giá trị là true, thì các tác vụ ở đây sẽ được lặp lại
}
```

Ví dụ

Lưu chương trình sau vào tập tin WhileDemo.java

```
import java.io.*;
public class WhileDemo {
public static void main(String args[]) {
int num = '9';
while (num > '0') { System.out.print((char)num +" "); num--;
}
}
}
```

Biên dịch và thực thi được kết quả sau:



Hình 2.6 Kết quả thực thi tập tin WhileDemo.java

2.8.5. Lệnh do - while

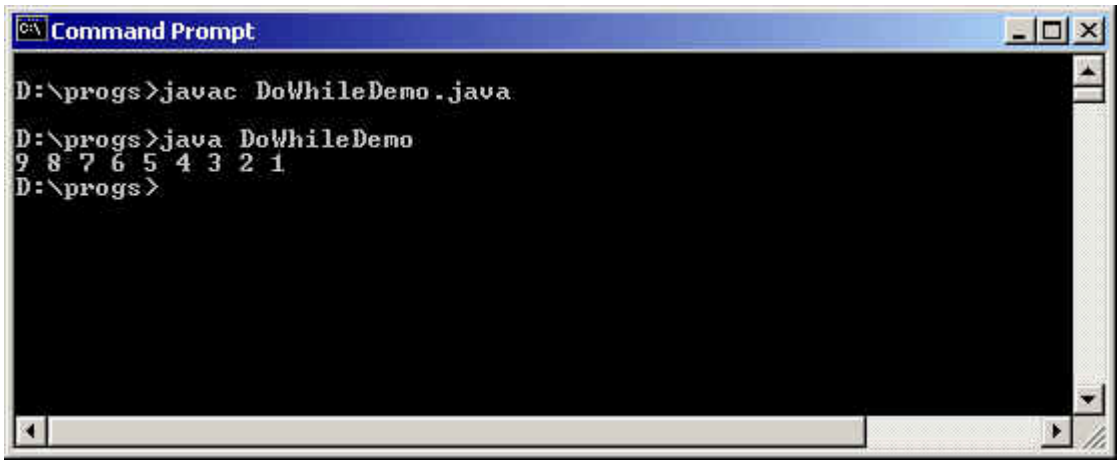
Cú pháp

```
do {
// Lặp lại các tác vụ ở đây cho đến khi điều kiện condition có giá trị là false
} while (condition)
```

Ví dụ: Lưu chương trình sau vào tập tin DoWhileDemo.java

```
import java.io.*;
public class DoWhileDemo {
public static void main(String args[]) {
int num = '9';
do {
System.out.print((char)num +" ");
num--;
} while (num > '0');
}
}
```

Biên dịch và thực thi được kết quả sau:



Hình 2.7 Kết quả thực thi tập tin DoWhileDemo.java

2.8.6. Lệnh for

Cú pháp

for (operation1; condition; operation2){

// Các tác vụ được lặp lại

}

Tương đương như cấu trúc sau:

operation1;

while (condition) {

// Các tác vụ được lặp lại operation2;

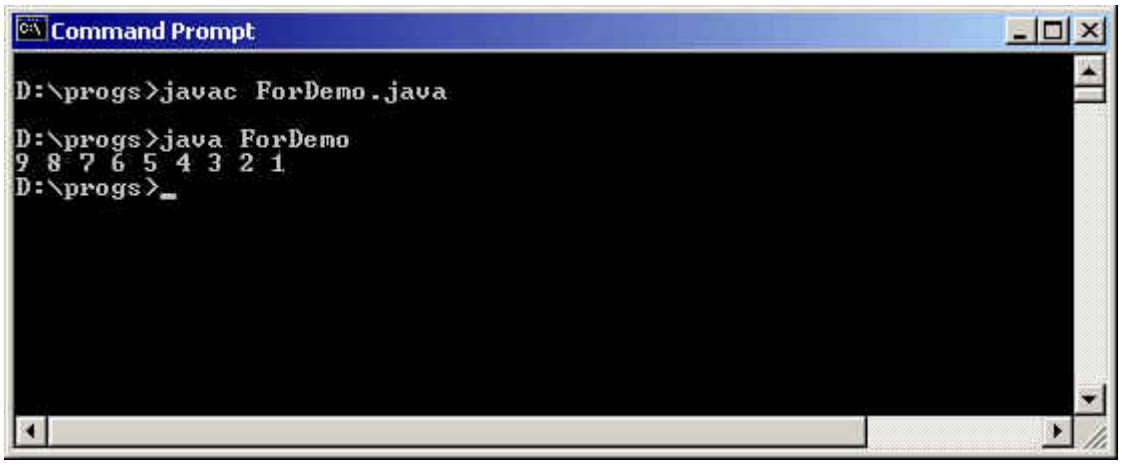
}

Ví dụ

Lưu chương trình sau vào tập tin ForDemo.java

```
import java.io.*;
public class ForDemo {
public static void main(String args[]) {
for(int num = '9'; num>'0'; num --) {
System.out.print((char)num +" ");
   }
  }
}
```

Biên dịch và thực thi được kết quả như sau:



Hình 2.8 Kết quả thực thi tập tin ForDemo.java

2.8.7. Lệnh break

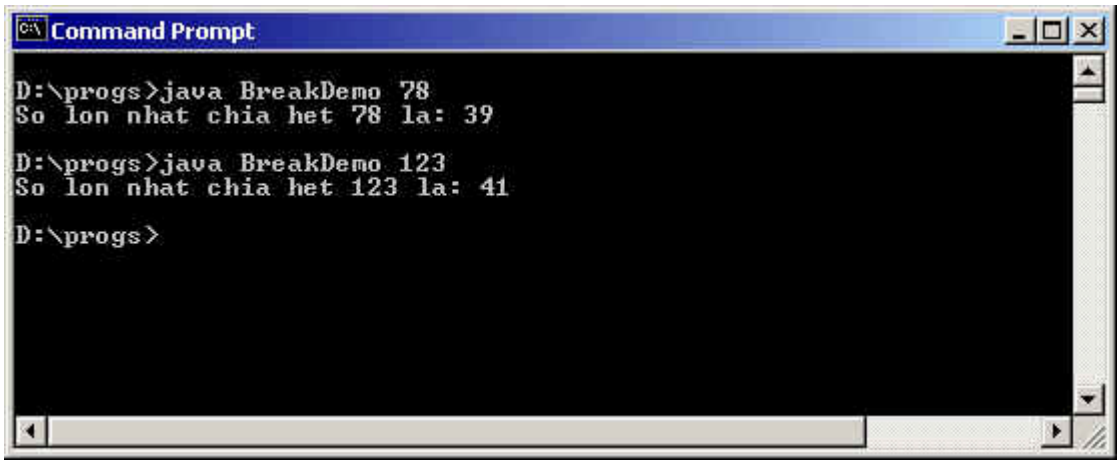
Vòng lặp của các lệnh while, do-while và for sẽ kết thúc khi lệnh break được thực hiện

Ví dụ

Lưu chương trình sau vào tập tin BreakDemo.java

```
import java.io.*;
public class BreakDemo {
public static void main(String args[]){
int num =Integer.valueOf(args[0]).intValue();
int i= num / 2;
while(true){
if (num % i ==0) break;
i--;
}
System.out.println("So lon nhat chia het "+num+ " la: "+i);
}
}
```

Biên dịch và thực thi được kết quả sau:



Hình 2.9 Kết quả thực thi tập tin BreakDemo.java

Chương trình trên đổi đối số thứ nhất của nó (lưu trong args[0]) thành số (bằng lệnh Integer.valueOf(args[0]).inValue()) và tìm số lớn nhất chia hết số này.

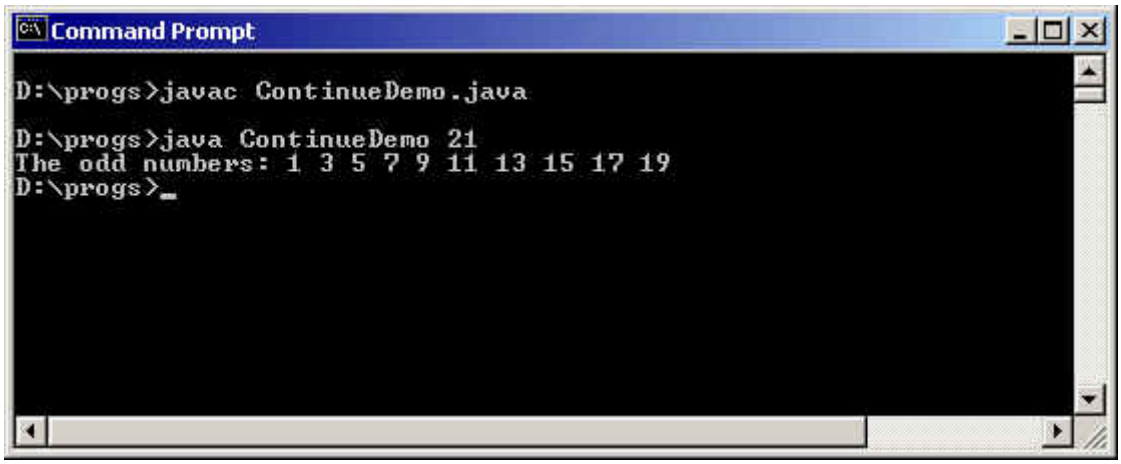
2.8.8. Lệnh continue

Trong một lần lặp nào đó của các lệnh while, do-while và for, nếu gặp lệnh continue thì lần lặp sẽ kết thúc (bỏ qua các lệnh phía sau continue) để bắt đầu lần lặp tiếp theo.

Ví dụ: Lưu chương trình sau vào tập tin ContinueDemo.java

```
import java.io.*;
public class ContinueDemo{
public static void main(String args[]){
int num =Integer.valueOf(args[0]).intValue();
System.out.print("The odd numbers: ");
for (int i =0; i< num; i++ ){ if (i % 2 ==0) continue;
System.out.print(i+ " ");
}
}
}
```

Biên dịch và thực thi được kết quả sau:



HHình 2.10 Kết quả thực thi tập tin ContinueDemo.java

Chương trình này in ra tất cả các số lẻ nhỏ hơn số đưa vào từ đối số.

2.9 Bài tập

Viết chương trình:

Bài 1: Viết chương trình khai báo 3 biến : x kiểu số thực, c kiểu ký tự, i kiểu số

nguyên.Nhập, xuất giá trị cho các biến đó.

Bài 2: Viết chương trình nhập vào 2 biến số nguyên x, y. Tính giá trị của x+y, xuất kết quả ra màn hình

Hướng dẫn:

- Khai báo 3 biến kiểu số nguyên: x,y và z (z: là biến lưu kết quả).

- Nhập giá trị cho x,y.

- Thực hiện phép gán: z = x+y.

- Xuất giá z ra màn hình.

Bài 3: Viết chương trình tính chu vi, diện tích của hình chữ nhật với chiều dài, rộng

nhập từ bàn phím.

Bài 4: Viết chương trình tính s=1+2+…+n, với n là số nguyên nhập từ bàn phím

Bài 5: Viết chương trình tính tổng các số lẽ đến n; s=1+3+5+…+n, với n lẽ là số nguyên nhập từ bàn phím.

Bài 6: Viết chương trình tính tổng các số chẵn đến n; s=1+2+4+…+n, với n chẵn là số nguyên nhập từ bàn phím.

Bài 7: Viết chương trình tính giá trị lớn nhất của 4 số nguyên a, b, c, d được nhập từ

bàn phím.

Bài 8: Viết chương trình thực hiện các phép toán số học.

Nhập vào 2 số thực x, y và ký tự K, Tính:

K = '*', ketqua = a*b

K = '+', ketqua = a+b

K = '-', ketqua = a-b

K = '/', ketqua = a/b

Hướng dẫn: sử dụng cấu trúc switch case

Bài 10: Viết chương trình giải phương trình bậc 2: ax2 + bx + c= 0, với a≠0

CHƯƠNG 3: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG JAVA

Mã bài : MH 29.3

Mục tiêu:

- Trình bày được trừu tượng hóa dữ liệu, lớp, đối tượng, gói, giao diện,...;
- Xác định được đặc điểm hướng đối tượng của Java;
- Liệt kê được các đối tượng cơ sở trong lập trình là mảng, xâu ký tự,...;
- Viết được một số chương trình hướng đối tượng thực hiện một số yêu cầu đơn giản;

3.1.Mở đầu

Thông qua chuyên đề lập trình hướng đối tượng (OOP) chúng ta đã biết OOP là một trong những tiếp cận mạnh mẽ, và rất hiệu quả để xây dựng nên những chương trình ứng dụng trên máy tính. Từ khi ra đời cho đến nay lập trình OOP đã chứng tỏ được sức mạnh, vai trò của nó trong các đề án tin học. Chương này sẽ giúp bạn đọc tìm hiểu về các kiểu dữ liệu dẫn xuất đó là lớp (class) và giao tiếp (interface), cũng như các vấn đề cơ bản về lập trình hướng đối tượng trong java thông qua việc tạo lập các lớp, các đối tượng và các tính chất của chúng.

3.2.Lớp (Class)

3.2.1.Khái niệm

Chúng ta có thể xem lớp như một khuôn mẫu (template) của đối tượng (Object). Trong đó bao gồm dữ liệu của đối tượng (fields hay properties) và các phương thức(methods) tác động lên thành phần dữ liệu đó gọi là các phương thức của lớp.

Các đối tượng được xây dựng bởi các lớp nên được gọi là các thể hiện của lớp (class instance).

3.2.2.Khai báo/định nghĩa lớp

class <ClassName>

{

<kiểu dữ liệu> <field_1>;

<kiểu dữ liệu> <field_2>;

constructor

method_1

method_2

}

class: là từ khóa của java

ClassName: là tên chúng ta đặt cho lớp

field_1, field_2: các thuộc tính, các biến, hay các thành phần dữ liệu của lớp.

constructor: là sự xây dựng, khởi tạo đối tượng lớp.

method_1, method_2: là các phương thức/hàm thể hiện các thao tác xử lý, tác động lên các thành phần dữ liệu của lớp.

3.2.3.Tạo đối tượng của lớp

ClassName objectName = new *ClassName();*

3.2.4.Thuộc tính của lớp

Vùng dữ liệu (fields) hay thuộc tính (properties) của lớp được khai báo bên trong lớp như sau:

class <ClassName>

{

// khai báo những thuộc tính của lớp

<tiền tố> <kiểu dữ liệu> field1;

// ...

}

Để xác định quyền truy xuất của các đối tượng khác đối với vùng dữ liệu của lớp người ta thường dùng 3 tiền tố sau:

- public: có thể truy xuất từ tất cả các đối tượng khác
- private: một lớp không thể truy xuất vùng private của 1 lớp khác.
- protected: vùng protected của 1 lớp chỉ cho phép bản thân lớp đó và những lớp dẫn xuất từ lớp đó truy cập đến.

Ví dụ:

public class xemay

{ public String nhasx;

public String model;

private float chiphisx;

protected int thoigiansx;

// so luong so cua xe may: 3, 4 so

protected int so;

// sobanhxe là biến tĩnh có giá trị là 2 trong tất cả // các thể hiện tạo ra từ lớp xemay

public static int sobanhxe = 2;

}

Thuộc tính *"nhasx", "model"*có thể được truy cập đến từ tất cả các đối tượng khác.

Thuộc tính *"chiphisx"* chỉ có thể truy cập được từ các đối tượng có kiểu *"xemay"*

Thuộc tính *"thoigiansx",* so có thể truy cập được từ các đối tượng có kiểu "*xemay"* và các đối tượng của các lớp con dẫn xuất từ lớp *"xemay"*

Lưu ý: Thông thường để an toàn cho vùng dữ liệu của các đối tượng người ta tránh dùng tiền tố public, mà thường chọn tiền tố private để ngăn cản quyền truy cập đến vùng dữ liệu của một lớp từ các phương thức bên ngoài lớp đó.

3.2.5.Hàm - Phương thức lớp (Method)

Hàm hay phương thức (method) trong Java là khối lệnh thực hiện các chức năng, các hành vi xử lý của lớp lên vùng dữ liệu.

Khai báo phương thức:

<Tiền tố> <kiểu trả về> <Tên phương thức> (<danh sách đối số>)

{

<khối lệnh>;

}

Để xác định quyền truy xuất của các đối tượng khác đối với các phương thức của lớp người ta thường dùng các tiền tố sau:

- public: phương thức có thể truy cập được từ bên ngoài lớp khai báo.
- protected: có thể truy cập được từ lớp khai báo và những lớp dẫn xuất từ nó.
- private: chỉ được truy cập bên trong bản thân lớp khai báo.
- static: phương thức lớp dùng chung cho tất cả các thể hiện của lớp, có nghĩa là phương thức đó có thể được thực hiện kể cả khi không có đối tượng của lớp chứa phương thức đó.
- final: phương thức có tiền tố này không được khai báo chồng ở các lớp dẫn xuất.
- abstract: phương thức không cần cài đặt (không có phần source code), sẽ được hiện thực trong các lớp dẫn xuất từ lớp này.

- synchoronized: dùng để ngăn các tác động của các đối tượng khác lên đối tượng đang xét trong khi đang đồng bộ hóa. Dùng trong lập trình miltithreads.

<kiểu trả về>: có thể là kiểu void, kiểu cơ sở hay một lớp.

<Tên phương thức>: đặt theo qui ước giống tên biến.

<danh sách thông số>: có thể rỗng

Lưu ý:

Thông thường trong một lớp các phương thức nên được khai báo dùng từ khóa public, khác với vùng dữ liệu thường là dùng tiền tố private vì mục đích an toàn.

Những biến nằm trong một phương thức của lớp là các biến cục bộ (local) và nên được khởia tạo sau khi khai báo.

Ví dụ:

```
public class xemay
{
publicString nhasx;
publicString model;
private        float   chiphisx;
protected      int     thoigiansx;
// so luong so cua xe may: 3, 4 so
protected      int     so;
// là biến tĩnh có giá trị là 2 trong tất cả
// các thể hiện tạo ra từ lớp xemay
public static    int sobanhxe = 2;
public float tinhgiaban()
{
return 1.5 * chiphisx;
}
}
```

3.2.6.Khởi tạo một đối tượng (Constructor)

Contructor thật ra là một loại phương thức đặc biệt của lớp.Constructor dùng gọi tự động khi khởi tạo một thể hiện của lớp, có thể dùng để khởi gán

những giá trị mặc định. Các constructor không có giá trị trả về, và có thể có tham số hoặc không có tham số.

Constructor phải có cùng tên với lớp và được gọi đến dùng từ khóa new.

Nếu một lớp không có constructor thì java sẽ cung cấp cho lớp một constructor mặc định (default constructor). Những thuộc tính, biến của lớp sẽ được khởi tạo bởi các giá trị mặc định (số: thường là giá trị 0, kiểu luận lý là giá trị false, kiểu đối tượng giá trị null, …)

Lưu ý: thông thường để an toàn, dễ kiểm soát và làm chủ mã nguồn chương trình chúng ta nên khai báo một constructor cho lớp.

Ví dụ:

```
public class xemay
{
// ...
public xemay()
{}
public xemay(String s_nhasx, String s_model,
f_chiphisx, int i_thoigiansx, int i_so);
{
nhasx = s_nhasx;
model = s_model;
chiphisx = f_chiphisx;
thoigiansx = i_thoigiansx;
so = i_so;
// hoặc
// this.nhasx = s_nhasx;
// this.model = s_model;
// this.chiphisx = f_chiphisx;
// this.thoigiansx = i_thoigiansx; // this.so = i_so;
}
}
```

3.2.7.Biến this

Biến this là một biến ẩn tồn tại trong tất cả các lớp trong ngông ngữ java. Một class trong Java luôn tồn tại một biến this, biến this được sử dụng trong khi chạy và tham khảo đến bản thân lớp chứa nó.

Ví dụ:

```
<tiền tố> class A
{
<tiền tố> int <field_1>;
<tiền tố> String <field_2>;
// Contructor của lớp A
public A(int par_1, String par_2)
{
this.field_1 = par_1;
this.field_2 = par_2; }
<tiền tố> <kiểu trả về> <method_1>()
{
// ...
}
<tiền tố> <kiểu trả về> <method_2>()
{
this.method_1()
// ...
}
}
```

3.2.8.Khai báo chồng phương thức (overloading method)

Việc khai báo trong một lớp nhiều phương thức có cùng tên nhưng khác tham số (khác kiểu dữ liệu, khác số lượng tham số) gọi là khai báo chồng phương thức (overloading method).

Ví dụ:

```
public class xemay
{       // khai báo fields ...
public float tinhgiaban()
{       return 2 * chiphisx;
```

}

public float tinhgiaban(float huehong)

*{return (2 * chiphisx + huehong);*

}

}

3.3.Đặc điểm hướng đối tượng trong java

Hỗ trợ những nguyên tắc cơ bản của lập trình hướng đối tượng, tất cả các ngôn ngữ lập trình kể cả java đều có ba đặc điểm chung: tính đóng gói (encapsulation), tính đa hình (polymorphism), và tính kế thừa (inheritance).

3.3.1.Đóng gói (encapsulation)

Cơ chế đóng gói trong lập trình hướng đối tượng giúp cho các đối tượng dấu đi một phần các chi tiết cài đặt, cũng như phần dữ liệu cục bộ của nó, và chỉ công bố ra ngoài những gì cần công bố để trao đổi với các đối tượng khác. Hay chúng ta có thể nói đối tượng là một thành tố hỗ trợ tính đóng gói.

Đơn vị đóng gói cơ bản của ngôn ngữ java là class. Một class định nghĩa hình thức của một đối tượng. Một class định rõ những thành phần dữ liệu và các đoạn mã cài đặt các thao tác xử lý trên các đối tượng dữ liệu đó. Java dùng class để xây dựng những đối tượng. Những đối tượng là những thể hiện (instances) của một class.

Một lớp bao gồm thành phần dữ liệu và thành phần xử lý. Thành phần dữ liệu của một lớp thường bao gồm các biến thành viên và các biến thể hiện của lớp. Thành phần xử lý là các thao tác trên các thành phần dữ liệu, thường trong java người gọi là phương thức. Phương thức là một thuật ngữ hướng đối tượng trong java, trong C/C++ người ta thường dùng thuật ngữ là hàm.

3.3.2.Tính đa hình (polymorphism):

Tính đa hình cho phép cài đặt các lớp dẫn xuất khác nhau từ một lớp nguồn. Một đối tượng có thể có nhiều kiểu khác nhau gọi là tính đa hình.

Ví dụ:

class A_Object

{

// ...

void method_1()

{

// ...

```
}
}
class B_Object extends A_Object
{
// ...
void method_1()
{
// ...
}
}
class C
{       public static void main(String[] args)
{
// Tạo một mảng 2 phần tử kiểu A
A_Object arr_Object = new A_Object[2]; B_Object var_1 = new B_Object();
// Phần tử đầu tiên của mảng arr_Object[0] tham // chiếu đến 1 đối tượng kiểu B_Object dẫn xuất // từ A_Object
arr_Object[0] = var_1; A_Object var_2;
for (int i=0; i<2; i++)
{
var_2 = arr_Object[i];
        var_2.method_1(); }
}
}
```

Vòng lặp for trong đoạn chương trình trên:

- Với i = 0 thì biến var_2 có kiểu là B_Object, và lệnh var_2.method_1() sẽ gọi thực hiện phương thức method_1 của lớp B_Object.
- Với i = 1 thì biến var_2 có kiểu là A_Object, và lệnh var_2.method_1() sẽ gọi thực hiện phương thức method_1 của lớp A_Object.

Trong ví dụ trên đối tượng var_2 có thể nhận kiểu A_Object hay B_Object. Hay nói các khác, một biến đối tượng kiểu A_Object như var_2 trong ví dụ trên có thể tham chiếu đến bất kỳ đối tượng nào của bất kỳ lớp con nào của lớp A_Object (ví dụ var_2 có thể tham chiếu đến đối tượng var_1, var_1 là đối tượng của lớp B_Object dẫn xuất từ lớp A_Object). Ngược lại một biến của lớp con không thể tham chiếu đến bất kỳ đối tượng nào của lớp cha.

3.3.3.Tính kế thừa (inheritance)

Một lớp con (subclass) có thể kế thừa tất cả những vùng dữ liệu và phương thức của một lớp khác (siêu lớp - superclass). Như vậy việc tạo một lớp mới từ một lớp đã biết sao cho các thành phần (fields và methods) của lớp cũ cũng sẽ thành các thành phần (fields và methods) của lớp mới. Khi đó ta gọi lớp mới là lớp dẫn xuất (derived class) từ lớp cũ (superclass). Có thể lớp cũ cũng là lớp được dẫn xuất từ một lớp nào đấy, nhưng đối với lớp mới vừa tạo thì lớp cũ đó là một lớp siêu lớp trực tiếp (immediate upperclass).

Dùng từ khóa extends để chỉ lớp dẫn xuất.

```
class A extends B
{
// ...
 }
```

3.3.3.1 Khái báo phương thức chồng

Tính kế thừa giúp cho các lớp con nhận được các thuộc tính/phương thức public và protected của lớp cha. Đồng thời cũng có thể thay thế các phương thức của lớp cha bằng cách khai báo chồng. Chẳng hạn phương thức *tinhgiaban()* áp dụng trong lớp xega sẽ cho kết quả gấp 2.5 lần chi phí sản xuất thay vì gấp 2 chi phí sản xuất giống như trong lớp *xemay*.

Ví dụ:

```
public class xega extends xemay
{
 public xega()
{
}
public xega(String s_nhasx, String s_model, f_chiphisx, int i_thoigiansx);
{
this.nhasx = s_nhasx;
```

this.model = s_model;

this.chiphisx = f_chiphisx;

this.thoigiansx = i_thoigiansx; this.so = 0;

}

public float tinhgiaban()

{

*return 2.5 * chiphisx;*

}

}

Java cung cấp 3 tiền tố/từ khóa để hỗ trợ tính kế thừa của lớp:

- public: lớp có thể truy cập từ các gói, chương trình khác.
- final: Lớp hằng, lớp không thể tạo dẫn xuất (không thể có con), hay đôi khi người ta gọi là lớp "vô sinh".
- abstract: Lớp trừu tượng (không có khai báo các thành phần và các phương thức trong lớp trừu tượng). Lớp dẫn xuất sẽ khai báo, cài đặt cụ thể các thuộc tính, phương thức của lớp trừu tượng.

3.3.3.2 Lớp nội

Lớp nội là lớp được khai báo bên trong 1 lớp khác. Lớp nội thể hiện tính đóng gói cao và có thể truy xuất trực tiếp biến của lớp cha.

Ví dụ:

public class A

{

// ...

int <field_1>

static class B

{

// ...

int <field_2>

public B(int par_1)

{

```
        field_2 = par_1 + field_1;
        }
        }
}
```

Trong ví dụ trên thì chương trình dịch sẽ tạo ra hai lớp với hai files khác nhau: A.class và B.class

3.3.3.3 Lớp vô sinh

Lớp không thể có lớp dẫn xuất từ nó (không có lớp con) gọi là lớp "vô sinh", hay nói cách khác không thể kế thừa được từ một lớp "vô sinh". Lớp "vô sinh" dùng để hạn chế, ngăn ngừa các lớp khác dẫn xuất từ nó.

Để khai báo một lớp là lớp "vô sinh", chúng ta dùng từ khóa final class.

Tất cả các phương thức của lớp vô sinh đều vô sinh, nhưng các thuộc tính của lớp vô sinh thì có thể không vô sinh.

Ví dụ:

```
public final class A
{
public final int x;
private int y;
public final void method_1()
{
// ...
}
public final void method_2()
{
// ...
}
}
```

3.3.3.4 Lớp trừu tượng

Lớp trừu tượng là lớp không có khai báo các thuộc tính thành phần và các phương thức. Các lớp dẫn xuất của nó sẽ khai báo thuộc tính, cài đặt cụ thể các phương thức của lớp trừu tượng.

Ví dụ:

```
abstract class A
{
abstract void method_1();
}
public class B extends A
{
 public void method_1()
    {
// cài đặt chi tiết cho phương thức method_1 // trong lớp con B.
// ...
}
}
public class C extends A
 {public void method_1()
{
// cài đặt chi tiết cho phương thức method_1 // trong lớp con C.
// ...
}
}
```

Lưu ý: Các phương thức được khai báo dùng các tiền tố private và static thì không được khai báo là trừu tượng abstract. Tiền tố private thì không thể truy xuất từ các lớp dẫn xuất, còn tiền tố static thì chỉ dùng riêng cho lớp khai báo mà thôi.

3.3.3.5 Phương thức finalize()

Trong java không có kiểu dữ liệu con trỏ như trong C, người lập trình không cần phải quá bận tâm về việc cấp phát và giải phóng vùng nhớ, sẽ có một trình dọn dẹp hệ thống đảm trách việc này. Trình dọn dẹp hệ thống sẽ dọn dẹp vùng nhớ cấp phát cho các đối tượng trước khi hủy một đối tượng.

Phương thức finalize() là một phương thức đặc biệt được cài đặt sẵn cho các lớp. Trình dọn dẹp hệ thống sẽ gọi phương thức này trước khi hủy một đối tượng. Vì vậy việc cài đặt một số thao tác giải phóng, dọn dẹp vùng nhớ đã cấp phát cho các đối

tượng dữ liệu trong phương thức finalize() sẽ giúp cho người lập trình chủ động kiểm soát tốt quá trình hủy đối tượng thay vị giao cho trình dọn dẹp hệ thống tự động. Đồng thời việc cài đặt trong phương thức finalize() sẽ giúp cho bộ nhớ được giải phóng tốt hơn, góp phần cải tiến tốc độ chương trình.

Ví dụ:

```
class A
{
// Khai báo các thuộc tính
public void method_1()
{
// ...
}
protected void finalize()
{
// Có thể dùng để đóng tất cả các kết nối
// vào cơ sở dữ liệu trước khi hủy đối tượng.
// ...
}
}
```

3.4.Gói (packages)

Việc đóng gói các lớp lại tạo thành một thư viện dùng chung gọi là package.

Một package có thể chứa một hay nhiều lớp bên trong, đồng thời cũng có thể chứa một package khác bên trong.

Để khai báo một lớp thuộc một gói nào đấy ta phải dùng từ khóa package.

Dòng khai báo gói phải là dòng đầu tiên trong tập tin khai báo lớp.

Các tập tin khai báo lớp trong cùng một gói phải được lưu trong cùng một thư mục.

Lưu ý: Việc khai báo import tất cả các lớp trong gói sẽ làm tốn bộ nhớ. Thông thường chúng ta chỉ nên import những lớp cần dùng trong chương trình.

Ví dụ:

```
package phuongtiengiaothong;
```

class xemay

{

// ….

}

class xega extends xemay

{

// ...

}

Khi đó muốn sử dụng lớp xemay vào chương trình ta sẽ khai báo như sau:

import phuongtiengiaothong.xemay;

3.5.Giao diện (interface)

3.5.1.Khái niệm interface:

Như chúng ta đã biết một lớp trong java chỉ có một siêu lớp trực tiếp hay một cha duy nhất (đơn thừa kế). Để tránh đi tính phức tạp của đa thừa kế (multi-inheritance) trong lập trình hướng đối tượng, Java thay thế bằng giao tiếp (interface). Một lớp có thể có nhiều giao tiếp (interface) với các lớp khác để

thừa hưởng thêm vùng dữ liệu và phương thức của các giao tiếp này.

3.5.2.Khai báo interface:

Interface được khai báo như một lớp. Nhưng các thuộc tính của interface là các hằng (khai báo dùng từ khóa final) và các phương thức của giao tiếp là trừu tượng (mặc dù không có từ khóa abstract).

Trong các lớp có cài đặt các interface ta phải tiến hành cài đặt cụ thể các phương thức này.

Ví dụ:

public interface sanpham

{ static final String nhasx = "Honda VN";

static final String dienthoai = "08-8123456";

public int gia(String s_model);

}

// khai báo 1 lớp có cài đặt interface

```
public class xemay implements sanpham
{       // cài đặt lại phương thức của giao diện trong lớp
public int     gia(String s_model)
{
if (s_model.equals("2005"))
        return (2000);
else
return (1500);
}

public String chobietnhasx()
{
return (nhasx);
}
```

Có một vấn đề khác với lớp là một giao diện (interface) không chỉ có một giao diện cha trực tiếp mà có thể dẫn xuất cùng lúc nhiều giao diện khác (hay có nhiều giao diện cha). Khi đó nó sẽ kế thừa tất cả các giá trị hằng và các phương thức của các giao diện cha. Các giao diện cha được liệt kê thành chuỗi và cách nhau bởi dấu phẩy ",". Khai báo như sau:

```
public interface InterfaceName extends interface1, interface2,
interface3
{
// ...
}
```

3.6. Mảng, xâu ký tự

3.6.1 Mảng

- Khái niệm mảng

Mảng là tập hợp nhiều phần tử có cùng tên, cùng kiểu dữ liệu và mỗi phần tử trong mảng được truy xuất thông qua chỉ số của nó trong mảng.

- Khai báo mảng

<kiểu dữ liệu> <tên mảng>[];

hoặc

<kiểu dữ liệu>[] <tên mảng>;

Ví dụ cách cách khai báo mảng:

int arrInt1[];

int[] arrInt2;

int[] arrInt3, arrInt4, arrInt5;

- Cấp phát bộ nhớ cho mảng

Để cấp phát bộ nhớ cho mảng trong Java ta cần dùng từ khóa new. (Tất cả trong

java đều thông qua các đối tượng). Chẳng hạn để cấp phát vùng nhớ cho mảng trong Java ta làm như sau:

int arrInt = new int[100];

- Truy cập mảng

Chỉ số mảng trong Java bắt đầu từ 0. Vì vậy phần tử đầu tiên có chỉ số là 0, và phần tử thứ n có chỉ số là n - 1. Các phần tử của mảng được truy xuất thông qua chỉ số của nó đặt giữa cặp dấu ngoặc vuông ([]).

Ví dụ:

int arrInt[] = {1, 2, 3};

int x = arrInt[0]; // x sẽ có giá trị là 1.

int y = arrInt[1]; // y sẽ có giá trị là 2.

int z = arrInt[2]; // z sẽ có giá trị là 3.

Lưu ý: Trong những ngôn ngữ lập trình khác (C chẳng hạn), một chuỗi được xem như một mảng các ký tự. Trong java thì khác, java cung cấp một lớp String để làm việc với đối tượng dữ liệu chuỗi cùng khác thao tác trên đối tượng dữ liệu này.

Ví dụ:

```
public class ViDuArray {
  public static void main(String[] args) {
    int arrInt[];
    arrInt = new int[4];
    arrInt[0] = 9;
    arrInt[1] = 17;
    arrInt[2] = 13;
    arrInt[3] = 14;
     String arrString[] = {"Vu Van A", "Nguyen Van B", "Nguyen Van C"};
```

```
        System.out.println("Mảng số nguyên: ");
        for (int i = 0; i < 4; i++) {
            System.out.print(arrInt[i] + " ");
        }
        System.out.println("\nMảng các chuỗi: ");

        for (int i = 0; i < 3; i++) {
            System.out.println(arrString[i] + " ");
        }
        System.out.println("");
    }
}
```

3.6.2 Xâu ký tự

- Lớp String

Chuỗi là một dãy các ký tự. Lớp String cung cấp các phương thức để thao tác với các

chuỗi. Nó cung cấp các phương thức khởi tạo (constructor) khác nhau:

```
String str1 = new String( );
//str1 chứa một chuỗi rống.
String str2 = new String(“Hello World”);
//str2 chứa “Hello World”
char ch[] = {‘A’,’B’,’C’,’D’,’E’};
String str3 = new String(ch);
//str3 chứa “ABCDE”
String str4 = new String(ch,0,2);
```

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

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

Ví dụ:

String strFirst, strSecond, strFinal;

StrFirst = “Charlie”;

StrSecond = “Chaplin”;

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

StrFinal = strFirst.concat(strSecond);

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

Các phương thức của lớp String

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

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

Ví dụ:

String name = new String(“Java Language”);

char ch = name.charAt(5);

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

– boolean startsWith(String s) Phương thức này trả về giá trị kiểu logic (Boolean), phụ

thuộc vào chuỗi có bắt đầu với một chuỗi con cụ thể nào đó không.

Ví dụ:

String strname = “Java Language”;

boolean flag = strname.startsWith(“Java”);

Biến “flag” chứa giá trịtrue.

– boolean endsWith(String s) Phương thức này trả về một giá trị kiểu logic (boolean),

hụ thuộc vào chuỗi kết thúc bằng một chuỗi con nào đó không.

Ví dụ:

String strname = “Java Language”;

boolean flag = strname.endsWith(“Java”);

Biến “flag” chứa giá trị false.

– String copyValueOf( )

Phương thức này trả về một chuỗi được rút ra từ một mảng ký tự được truyền như một đối số. Phương thức này cũng lấy hai tham số nguyên. Tham số

đầu tiên chỉ định vị trí từ nơi các ký tự phải được rút ra, và tham số thứ hai chỉ định số ký tự được rút ra từ mảng.

Ví dụ:

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

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

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

3.7 Bài tập

Bài tập 1: Minh họa tính đa hình (polymorphism) trong phân cấp kế thừa thông qua việc mô tả và xử lý một số thao tác cơ bản trên các đối tượng hình học.

*// Định nghĩa lớp trừu tượng cơ sở tên **Shape** trong*

// tập tin Shape.java

public abstract class Shape extends Object

{

// trả về diện tích của một đối tượng hình học shape

public double area()

{

return 0.0;

}

// trả về thể tích của một đối tượng hình học shape

public double volume()

{

return 0.0;

}

// Phương thức trừu tượng cần phải được hiện thực

// trong những lớp con để trả về tên đối tượng

// hình học shape thích hợp public abstract String getName(); } // end class Shape

// Định nghĩa lớp Point trong tập tin Point.java public class Point extends Shape

{ protected int x, y; // Tọa độ x, y của 1 điểm

// constructor không tham số. public Point()

{ setPoint(0, 0);

}

```
// constructor có tham số. public Point(int xCoordinate, int yCoordinate)
{ setPoint( xCoordinate, yCoordinate );
}
// gán tọa độ x, y cho 1 điểm
public void setPoint( int xCoordinate, int yCoordinate )
{
x = xCoordinate; y = yCoordinate;
}
// lấy tọa độ x của 1 điểm public int getX()
{ return x;
}
// lấy tọa độ y của 1 điểm public int getY()
{ return y;
}
// Thể hiện tọa độ của 1 điểm dưới dạng chuỗi
public String toString()
{ return "[" + x + ", " + y + "]";
}
// trả về tên của đối tượng shape public String getName()
{ return "Point";
}
} // end class Point
```

Định nghĩa một lớp cha **Shape** là một lớp trừu tượng dẫn xuất từ Object và có 3 phương thức khai báo dùng tiền tố public. Phương thức **getName()** khai báo trừu tượng vì vậy nó phải được hiện thực trong các lớp con. Phương thức **area()** (tính diện tích) và phương thức **volume()** (tính thể tích) được định nghĩa và trả về 0.0. Những phương thức này sẽ được khai báo chồng trong các lớp con để thực hiện chức năng tính diện tích cũng như thể tích phù hợp với những đối tượng hình học tương ứng (đường tròn, hình trụ, …)

Lớp **Point:** dẫn xuất từ lớp **Shape**. Một điểm thì có diện tích và thể tích là 0.0, vì vậy những phương thức **area**() và **volume**() của lớp cha không cần khai báo chồng trong lớp **Point,** chúng được thừa kế như đã định nghĩa trong lớp trừu tượng **Shape**. Những phương thức khác như **setPoint**(…) để gán tọa độ x, y cho một điểm, còn phương thức **getX()**, **getY()** trả về tọa độ x, y của một điểm.

Phương thức **getName()** là hiện thực cho phương thức trừu tượng trong lớp cha, nếu như phương thức **getName()** mà không được định nghĩa thì lớp **Point** là một lớp trừu tượng.

// Định nghĩa lớp Circle trong tập tin Circle.java public class Circle extends Point

{ // Dẫn xuất từ lớpPoint protected double radius;

// constructor không tham số public Circle()

{

// ngầm gọi đến constructor của lớp cha

setRadius(0);

}

// constructor có tham số

public Circle(double circleRadius, int xCoordinate, int yCoordinate) {

// gọi constructorcủa lớp cha super(xCoordinate, yCoordinate);

setRadius(circleRadius);

}

// Gán bán kính của đường tròn public void setRadius(double circleRadius)

{ radius = (circleRadius >= 0 ? circleRadius:0);

}

// Lấy bán kính của đường tròn

public double getRadius()

{ return radius;

}

// Tính diện tích đường tròn Circle public double area()

*{ return Math.PI * radius * radius;*

}

// Biểu diễn đường tròn bằng một chuỗi

public String toString()

{ return "Center = " + super.toString() +

"; Radius = " + radius;

}

// trả về tên của shape public String getName()

```
{
return "Circle";
}
} // end class Circle
```

Lớp **Circle** dẫn xuất từ lớp **Point**, một đường tròn có thể tích là 0.0, vì vậy phương thức **volume()** của lớp cha không khai báo chồng, nó sẽ thừa kế từ lớp **Point**, mà lớp **Point** thì thừa kế từ lớp **Shape**. Diện tích đường tròn khác với một điểm, vì vậy phương thức tính diện tích **area()** được khai báo chồng. Phương thức **getName()** hiện thực phương thức trừu tượng đã khai báo trong lớp cha, nếu phương thức **getName()** không khai báo trong lớp **Circle** thì nó sẽ kế thừa từ lớp **Point**. Phương thức **setRadius** dùng để gán một bán kính (radius) mới cho một đối tượng đường tròn, còn phương thức **getRadius** trả về bán kính của một đối tượng đường tròn.

```
// Định nghĩa lớp hình trụ Cylinder  // trong tập tin Cylinder.java.
public class Cylinder extends Circle
{
        // chiều cao của Cylinder  protected double height;
 // constructor không có tham số public Cylinder()
{
// ngầm gọi đến constructor của lớp cha
setHeight( 0 );
}
 // constructor có tham số public Cylinder( double cylinderHeight, double cylinderRadius, int xCoordinate, int yCoordinate )
{
// Gọi constructor của lớp cha super( cylinderRadius, xCoordinate, yCoordinate );
 setHeight( cylinderHeight );
}
 // Gán chiều cao cho Cylinder
public void setHeight( double cylinderHeight )
{
height = ( cylinderHeight >= 0 ? cylinderHeight
:0 ); }
```

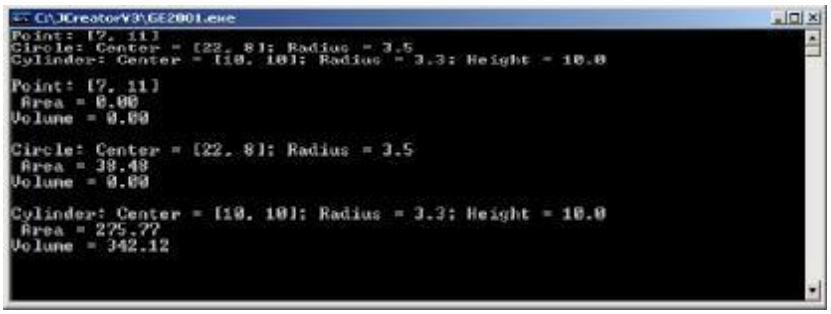
```
 // Lấy chiều cao của Cylinder public double getHeight()
{ return height;
}
 // Tính diện tích xung quanh của Cylinder  public double area()
{ return 2 * super.area() + 2 * Math.PI * radius * height;
}
 // Tính thể tích của Cylinder
public double volume()
{ return super.area() * height;
}
 // Biểu diễn Cylinder bằng một chuỗi  public String toString()
{ return super.toString() + "; Height = " + height; }
 // trả về tên của shape  public String getName()
{ return "Cylinder";
}
 } // end class Cylinder
```

Lớp **Cylinder** dẫn xuất từ lớp Circle. Một Cylinder (hình trụ) có diện tích và thể tích khác với một Circle (hình tròn), vì vậy cả hai phương thức **area()** và **volume()** cần phải khai báo chồng. Phương thức getName() là hiện thực phương thức trừu tượng trong lớp cha, nếu phương thức **getName()** không khai báo trong lớp **Cylinder** thì nó sẽ kế thừa từ lớp **Circle**. Phương thức **setHeight** dùng để gán chiều cao mới cho một đối tượng hình trụ, còn phương thức **getHeight** trả về chiều cao của một đối tượng hình trụ.

```
// Test.java
// Kiểm tra tính kế thừa của Point, Circle, Cylinder với  // lớp trừu tượng Shape. // Khai báo thư viện import java.text.DecimalFormat;
public class Test
{
 // Kiểm tra tính kế thừa của các đối tượng hình học
 public static void main( String args[] )
        {
                // Tạo ra các đối tượng hìnhhọc Point point = new Point( 7, 11
);
```

```
Circle circle = new Circle( 3.5, 22, 8 );
Cylinder cylinder = new Cylinder( 10, 3.3, 10, 10 );
 // Tạo một mảng các đối tượng hình học
Shape arrayOfShapes[] = new Shape[ 3 ];
 // arrayOfShapes[ 0 ] là một đối tượng Point  arrayOfShapes[ 0 ] = point;
// arrayOfShapes[ 1 ] là một đối tượng Circle arrayOfShapes[ 1 ] = circle;
// arrayOfShapes[ 2 ] là một đối tượng cylinder arrayOfShapes[ 2 ] = cylinder;
 // Lấy tên và biểu diễn của mỗi đối tượng hình học
String output =
point.getName() + ": " + point.toString() + "\n" + circle.getName() + ": " + circle.toString() + "\n" +
cylinder.getName() + ": " + cylinder.toString();
 DecimalFormat precision2 = new DecimalFormat(
"0.00" );
 // duyệt mảng arrayOfShapes lấy tên, diện tích, thể tích // của mỗi đối tượng hình học trong mảng. for ( int i = 0; i < arrayOfShapes.length; i++ )
{
output += "\n\n" + arrayOfShapes[ i ].getName() +
": " + arrayOfShapes[ i].toString() +
"\n Area = " +
precision2.format( arrayOfShapes[ i ].area() ) +
"\nVolume = " +
precision2.format( arrayOfShapes[ i ].volume() );
}
        System.out.println(output);
System.exit( 0 );
        }
} // end class Test
```

Kết quả thực thi chương trình:



Hình 3.1 *Kiểm tra tính kế thừa của các đối tượng hình học - Circle*

Bài tập 2: Tương tự bài tập 1 nhưng trong bài tập 2 chúng ta dùng interface để định nghĩa cho **Shape** thay vì một lớp trừu tượng. Vì vậy tất cả các phương thức trong interface **Shape** phải được hiện thực trong lớp **Point** là lớp cài đặt trực tiếp interface **Shape**.

// Định nghĩa một interface Shape trong tập tin shape.java public interface Shape

{

// Tính diện tích public abstract double area();

// Tính thể tích

public abstract double volume();

// trả về tên của shape public abstract String getName();

}

Lớp *Point* cài đặt/hiện thực *interface* tên *shape*. *// Định nghĩa lớp Point trong tập tin Point.java public class Point extends Object implements Shape*

{ protected int x, y; // Tọa độ x, y của 1 điểm

// constructor không tham số. public Point()

{ setPoint(0, 0);

}

// constructor có tham số. public Point(int xCoordinate, int yCoordinate)

{ setPoint(xCoordinate, yCoordinate);

}

// gán tọa độ x, y cho 1 điểm

public void setPoint(int xCoordinate, int yCoordinate)

{

x = xCoordinate; y = yCoordinate;

}

```
// lấy tọa độ x của 1 điểm public int getX()
{ return x;
}
// lấy tọa độ y của 1 điểm public int getY()
{ return y;
}
// Thể hiện tọa độ của 1 điểm dưới dạng chuỗi
public String toString()
{ return "[" + x + ", " + y + "]";
}
// Tính diện tích
public double area()
{ return 0.0;
}
// Tính thể tích public double volume() { return 0.0;
}
// trả về tên của đối tượng shape
public String getName()
{ return "Point";
}
} // end class Point
```

Lớp *Circle* là lớp con của lớp *Point*, và cài đặt/hiện thực gián tiếp *interface* tên *shape*.

```
// Định nghĩa lớp Circle trong tập tin Circle.java public class Circle extends Point
{       // Dẫn xuất từ lớpPoint protected double radius;
// constructor không tham số public Circle() {
// ngầm gọi đến constructor của lớp cha
setRadius( 0 );
}
// constructor có tham số
public Circle( double circleRadius, int xCoordinate, int yCoordinate ) {
// gọi constructorcủa lớp cha super( xCoordinate, yCoordinate );
```

```
 setRadius( circleRadius );
}
 // Gán bán kính của đường tròn public void setRadius( double circleRadius
)
{ radius = ( circleRadius >= 0 ? circleRadius:0 );
}
 // Lấy bán kính của đường tròn public double getRadius()
{ return radius;
}
 // Tính diện tích đường tròn Circle public double area()
{ return Math.PI * radius * radius;
}
 // Biểu diễn đường tròn bằng một chuỗi
public String toString()
{ return "Center = " + super.toString() +
"; Radius = " + radius;
}
 // trả về tên của shape public String getName()
 {
return "Circle";
}
} // end class Circle
 // Định nghĩa lớp hình trụ Cylinder  // trong tập tin Cylinder.java.
public class Cylinder extends Circle
{
        // chiều cao của Cylinder  protected double height;
 // constructor không có tham số public Cylinder()
{
// ngầm gọi đến constructor của lớp cha
setHeight( 0 );
}
 // constructor có tham số public Cylinder( double cylinderHeight, double
cylinderRadius, int xCoordinate, int yCoordinate ) {
```

```
// Gọi constructor của lớp cha super( cylinderRadius, xCoordinate, yCoordinate );
 setHeight( cylinderHeight );
}
 // Gán chiều cao cho Cylinder
public void setHeight( double cylinderHeight )
{
height = ( cylinderHeight >= 0 ? cylinderHeight
:0 );
}
  // Lấy chiều cao của Cylinder
public double getHeight()
{ return height;
}
 // Tính diện tích xung quanh của Cylinder  public double area()
{ return 2 * super.area() + 2 * Math.PI * radius * height;
}
 // Tính thể tích của Cylinder public double volume()
{ return super.area() * height;
}
 // Biểu diễn Cylinder bằng một chuỗi  public String toString()
{ return super.toString() + "; Height = " + height; }
 // trả về tên của shape
public String getName()
{ return "Cylinder";
}
 } // end class Cylinder
 // Test.java
// Kiểm tra tính kế thừa của Point, Circle, Cylinder với  // interface Shape.
 // Khai báo thư viện import java.text.DecimalFormat; public class Test
{
 // Kiểm tra tính kế thừa của các đối tượng hình học
 public static void main( String args[] )
```

```
        {
 // Tạo ra các đối tượng hìnhhọc Point point = new Point( 7, 11 );
Circle circle = new Circle( 3.5, 22, 8 );
Cylinder cylinder = new Cylinder( 10, 3.3, 10, 10 );
 // Tạo một mảng các đối tượng hình học
Shape arrayOfShapes[] = new Shape[ 3 ];
 // arrayOfShapes[ 0 ] là một đối tượng Point  arrayOfShapes[ 0 ] = point;
// arrayOfShapes[ 1 ] là một đối tượng Circle arrayOfShapes[ 1 ] = circle;
// arrayOfShapes[ 2 ] là một đối tượng cylinder arrayOfShapes[ 2 ] = cylinder;
 // Lấy tên và biểu diễn của mỗi đối tượng hình học
String output =
point.getName() + ": " + point.toString() + "\n" + circle.getName() + ": " + circle.toString() + "\n" +
cylinder.getName() + ": " + cylinder.toString();
 DecimalFormat precision2 = new DecimalFormat(
"0.00" );
 // duyệt mảng arrayOfShapes lấy tên, diện tích, thể tích // của mỗi đối tượng hình học trong mảng. for ( int i = 0; i < arrayOfShapes.length; i++ )
{
output += "\n\n" + arrayOfShapes[ i ].getName() +
": " + arrayOfShapes[ i].toString() +
"\n Area = " +
precision2.format( arrayOfShapes[ i ].area() ) +
"\nVolume = " +
precision2.format( arrayOfShapes[ i ].volume() );
}

 System.out.println(output);
System.exit( 0 );
        }
 } // end class Test
```

Kết quả thực thi chương trình:

Hình 3.2 *Kiểm tra tính kế thừa của các đối tượng hình học*

CHƯƠNG 4: THIẾT KẾ GIAO DIỆN NGƯỜI DÙNG

Mã bài : MH 29.4

Mục tiêu:

– Xác định được tính năng giao diện người dùng trong các ứng dụng Java;

– Vận dụng được cấu trúc và cách sử dụng thư viện AWT;

– Trình bày được công dụng, thuộc tính, cách sử dụng các Component, Container, Layout Manager;

– Xử lý được sự kiện với các Component;

– Thiết kế được các giao diện GUI cho các ứng dụng, viết được một số mô đun đơn giản xử lý các sự kiện cho giao diện và các thiết bị chuột, bàn phím,...;

– Rèn luyện tính nghiêm túc, cẩn thận, sáng tạo trong học lý thuyết và thực hành.

4.1.Mở đầu

Chương này cung cấp cho sinh viên những kiến thức cơ bản để xây dựng giao diện (Graphic User Interface - GUI) của chương trình ứng dụng bằng ngôn ngữ java:

- Những nguyên tắc thiết kế giao diện.
- Những thư viện, gói xây dựng giao diện: gồm những lớp (class), những giao tiếp (interface) quản lý sự kiện và những thành phần (components) xây dựng nên giao diện người dùng.
- Bộ quản lý trình bày (layout managers)
- Xử lý sự kiện

Trong khuôn khổ giáo trình lập trình java căn bản này chúng tôi trình bày việc thiết kế GUI dùng thư viện awt (abstract windows toolkit). Việc thiết kết GUI sẽ trực quan, uyển chuyển hơn khi chúng ta sử dụng thư viện JFC (Java Foundation Class) sẽ giới được giới thiệu trong chuyên đề java nâng cao.

4.2. Giới thiệu thư viện awt

Thư viện awt là bộ thư viện dùng để xây dựng giao diện người dùng cho một chương trình ứng dụng có đầy đủ các thành phần cơ bản như: Label, Button, Checkbox, Radiobutton, Choice, List, Text Field, Text Area, Scrollbar, Menu, Frame…

Giống như các API của Windows, java cung cấp cho người lập trình thư viện awt. Nhưng khác với các hàm API, thư viện

awt không phụ thuộc hệ điều hành. Thư viện awt là nền tảng, cơ sở giúp cho chúng ta tiếp cận với thư viện mở rộng JFC hiệu quả hơn.

Cấu trúc cây phân cấp của tất cả những lớp trong thư viện awt chúng ta có thể xem chi tiết trong tài liệu kèm theo bộ công cụ j2se (phần API Specification)

4. 3. Các khái niệm cơ bản

4.3.1.Component

Component là một đối tượng có biểu diễn đồ họa được hiển thị trên màn hình mà người dùng có thể tương tác được. Chẳng hạn như những nút nhấn (button), những checkbox, những scrollbar,… Lớp **Component** là một lớp trừu tượng.

java.lang.Object

└java.awt.Component

4.3.2.Container

Container là đối tượng vật chứa hay những đối tượng có khả năng quản lý và nhóm các đối tượng khác lại. Những đối tượng con thuộc thành phần awt như: button, checkbox, radio button, scrollbar, list,… chỉ sử dụng được khi ta đưa nó vào khung chứa (container).

Một số đối tượng container trong Java:

Panel: Đối tượng khung chứa đơn giản nhất, dùng để nhóm các đối tượng, thành phần con lại. Một Panel có thể chứa bên trong một Panel khác.

java.lang.Object +--java.awt.Component

+--java.awt.Container

+--java.awt.Panel

Frame: khung chứa Frame là một cửa số window hẳn hoi ở mức trên cùng bao gồm một tiêu đều và một đường biên (border) như các ứng dụng windows thông thường khác. Khung chứa Frame thường được sử dụng để tạo ra cửa sổ chính của các ứng dụng.

java.lang.Object +--java.awt.Component

+--java.awt.Container

+--java.awt.Window

+--java.awt.Frame

Dialogs: đây là một cửa sổ dạng hộp hội thoại (cửa sổ dạng này còn được gọi là pop-up window), cửa sổ dạng này thường được dùng đểđưa ra thông báo,

hay dùng để lấy dữ liệu nhập từ ngoài vào thông qua các đối tượng, thành phần trên dialog như TextField chẳng hạn. Dialog cũng là một cửa sổ nhưng không đầy đủ chức năng nhưđối tượng khung chứa Frame. java.lang.Object +-- java.awt.Component

+--java.awt.Container

+--java.awt.Window

+--java.awt.Dialog

ScrollPanes: là một khung chứa tương tự khung chứa Panel, nhưng có thêm 2 thanh trượt giúp ta tổ chức và xem được các đối tượng lớn choán nhiều chỗ trên màn hình như những hình ảnh hay văn bản nhiều dòng.

java.lang.Object +--java.awt.Component

+--java.awt.Container

+--java.awt.ScrollPane

4.3.3.Layout Manager

Khung chứa container nhận các đối tượng từ bên ngoài đưa vào và nó phải biết làm thế nào để tổ chức sắp xếp "chỗở" cho các đối tượng đó. Mỗi đối tượng khung chứa đều có một bộ quản lý chịu trách nhiệm thực hiện công việc đấy đó là bộ quản lý trình bày (Layout Manager). Các bộ quản lý trình bày mà thư viện AWT cung cấp cho ta bao gồm:

FlowLayout: Sắp xếp các đối tượng từ trái qua phải và từ trên xuống dưới. Các đối tượng đều giữ nguyên kích thước của mình.

BorderLayout: Các đối tượng được đặt theo các đường viền của khung chứa theo các cạnh West, East, South, North và Center tức Đông, Tây, Nam, Bắc và Trung tâm hay Trái, Phải, Trên, Dưới và Giữa tùy theo cách nhìn của chúng ta.

GridLayout: Tạo một khung lưới vô hình với các ô bằng nhau. Các đối tượng sẽđặt vừa kích thước với từng ô đó. Thứ tự sắp xếp cũng từ trái qua phải và từ trên xuống dưới.

GridBagLayout: Tương tự như GridLayout, các đối tượng khung chứa cũng được đưa vào một lưới vô hình. Tuy nhiên kích thước các đối tượng không nhất thiết phải vừa với 1 ô mà có thể là 2, 3 ô hay nhiều hơn tùy theo các ràng buộc mà ta chỉđịnh thông qua đối tượng GridBagConstraint.

Null Layout: Cách trình bày tự do. Đối với cách trình bày này người lập trình phải tựđộng làm tất cả từ việc định kích thước của các đối tượng, cũng như xác định vị trí của nó trên màn hình. Ta không phụ thuộc vào những ràng buộc đông, tây , nam, bắc gì cả.

4. 4. Thiết kế GUI cho chương trình

4.4.1.Tạo khung chứa cửa sổ chương trình

Thông thường để tạo cửa sổ chính cho chương trình ứng dụng ta tiến hành các bước:

- Tạo đối tượng Frame
- Xác định kích thước của Frame
- Thể hiện Frame trên màn hình **Ví dụ**:

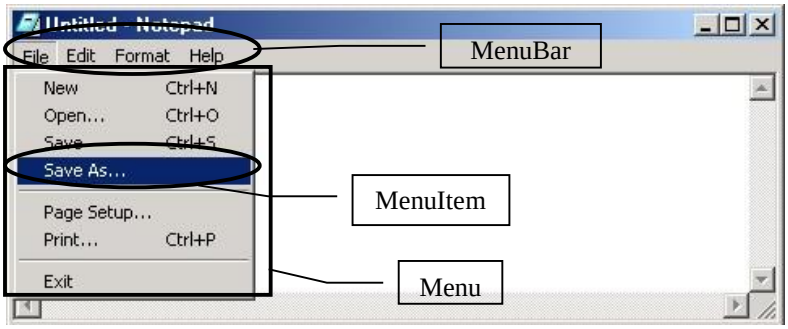
```
import java.awt.*; class FrameDemo
        {
public static void main(String args[])
                {
// Tạo đối tượng khung chứaFrame
 Frame fr = new Frame("My First Window") ;   // Xác định kích thước, vị trí của Frame   fr.setBounds(0, 0, 640, 480);
                // Hiển thị Frame
                        fr.setVisible(true);
                }
}
```

Kết quả thực thi chương trình:

Hình 4.1 Tạo khung chứa cửa sổ chương trình

4.4.2.Tạo hệ thống thực đơn

Đối với thư viện awt, để xây dựng hệ thống thực đơn cho chương trình ứng dụng chúng ta có thể dùng các lớp MenuBar, Menu, MenuItem, MenuShortcut.



Hình 4.2 Tạo hệ thống thực đơn

Ví dụ: Tạo hệ thống thực đơn cho chương trình Calculator import java.awt.*; *import java.awt.event.*;*

```
class Calculator
{
        public static void main(String[] args)
        {
createMenu();
        }
                private static void createMenu()
        {
// Tao Frame ung dung    final Frame fr = new Frame();
        fr.setLayout(new BorderLayout());
        // Tao cac menu bar
        MenuBar menu = new MenuBar();
        Menu menuFile = new Menu("Edit");
MenuItem copyItem = new MenuItem("Copy Ctrl+C");
MenuItem pasteItem = new MenuItem("Paste Ctrl+V");
menuFile.add(copyItem);
        menuFile.add(pasteItem);
        Menu menuHelp = new Menu("Help");
MenuItem hTopicItem = new MenuItem("Help Topics");
MenuItem hAboutItem = new MenuItem("About
Calculator");
        menuHelp.add(hTopicItem);      menuHelp.addSeparator();
menuHelp.add(hAboutItem);      menu.add(menuFile);
```

```
        menu.add(menuHelp);

    fr.setMenuBar(menu);
        fr.setBounds(100, 100, 300, 200);
    fr.setTitle("Calculator"); //fr.setResizable(false);
    fr.setVisible(true);

        // xử lý biến sự kiện đóng cửa số ứng dụng.
        fr.addWindowListener(
                new WindowAdapter()
                {
            public void windowClosing(WindowEvent e)
                    {
                        System.exit(0);
            }
                });
        }
    }
```

Kết quả thực thi chương trình:

Hình 4.3 Tạo hệ thống thực đơn

4.4.3.Gắn Component vào khung chứa

Để gắn một thành phần, một đối tượng component vào một cửa số (khung chứa) chúng ta dùng phương thức add của đối tượng khung chứa container.

Ví dụ:

```
import java.awt.*; class AddDemo
{
```

```
        public static void main(String args[])
        {
            // Tạo đối tượng khung chứaFrame
            Frame fr = new Frame("AddDemo App");
            // Tạo đối tượng Component
        Button buttOk = new Button("OK");
// Gắn đối tượng nút nhấn vào khung chứa        fr.add(buttOk);
        // Xác định kích thước, vị trí của Frame
fr.setSize(100, 100);
    // Hiển thị Frame        fr.setVisible(true);
        }
    }
```

Kết quả thực thi chương trình:

Hình 4.4 Gắn Component vào khung chứa

4.4.4.Trình bày các Component trong khung chứa

Như chúng ta đã biết khung chứa container nhận các đối tượng từ bên ngoài đưa vào và nó phải biết làm thế nào để tổ chức sắp xếp "chỗ ở" cho các đối tượng đó. Mỗi đối tượng khung chứa đều có một bộ quản lý chịu trách nhiệm thực hiện công việc đấy đó là bộ quản lý trình bày (Layout Manager). Chúng ta sẽ tìm hiểu chi tiết về các kiểu trình bày của thư viện

AWT.

Interface LayoutManager định nghĩa giao tiếp cho những lớp biết được làm thế nào để trình bày những trong những containers

4.4.4.1 FlowLayout

public class **FlowLayout** extends Object implements LayoutManager, Serializable

Đối với một container trình bày theo kiểu FlowLayout thì:

- Các component gắn vào được sắp xếp theo thứ tự từ trái sang phải và từ trên xuống dưới.
- Các component có kích thước như mong muốn.
- Nếu chiều rộng của Container không đủ chỗ cho các component thì chúng tựđộng tạo ra một dòng mới.
- FlowLayout thường được dùng đểđể sắp xếp các button trong 1 panel.
- Chúng ta có thểđiều chỉnh khoảng cách giữa các component.

Ví dụ: import java.awt.*; import java.lang.Integer;

```
class FlowLayoutDemo
{
        public static void main(String args[])
        {
 Frame fr = new Frame("FlowLayout Demo");
fr.setLayout(new FlowLayout());
fr.add(new Button("Red"));  fr.add(new Button("Green"));
        fr.add(new Button("Blue"));

        List li = new List();          for (int i=0; i<5; i++)
        {
                li.add(Integer.toString(i));
        }

        fr.add(li);
        fr.add(new Checkbox("Pick me", true));
        fr.add(new Label("Enter your name:"));
fr.add(new TextField(20));
        // phương thức pack() được gọi sẽ làm cho cửa sổ
// hiện hành sẽ có kích thước vừa với kích thước  // trình bày bố trí những thành phần con của nó.
        fr.pack();
        fr.setVisible(true);
        }
```

}

Kết quả thực thi chương trình:

Hình 4.5 FlowLayout

4.4.4.2 BorderLayout

public class BorderLayout extends Object implements LayoutManager2, Serializable

Đối với một container trình bày theo kiểu BorderLayout thì:

Bộ trình bày khung chứa được chia làm 4 vùng: NORTH, SOUTH, WEST, EAST và CENTER. (Đông, Tây, Nam, Bắc và trung tâm). Bộ trình bày loại này cho phép sắp xếp và thay đổi kích thước của những components chứa trong nó sao cho vừa với 5 vùng ĐÔNG, TÂY, NAM, BẮC, TRUNG TÂM.

Không cần phải gắn component vào cho tất cả các vùng.

Các component ở vùng NORTH và SOUTH có chiều cao tùy ý nhưng có chiều rộng đúng bằng chiều rộng vùng chứa.

Các component ở vùng EAST và WEST có chiều rộng tùy ý nhưng có chiều cao đúng bằng chiều cao vùng chứa.

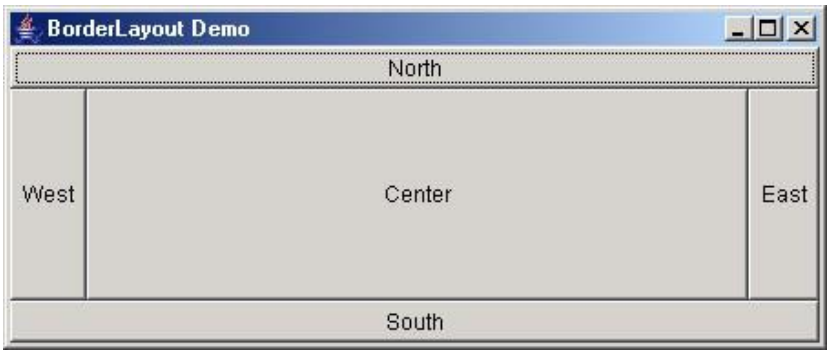
Các component ở vùng CENTER có chiều cao và chiều rộng phụ thuộc vào các vùng xung quanh.

Ví dụ:

```
import java.awt.*;
class BorderLayoutDemo extends Frame
{
        private Button north, south, east, west, center;
        public BorderLayoutDemo(String sTitle)
        {
        super(sTitle);
north = new Button("North");
south = new Button("South");
```

```
east    = new Button("East");
west    = new Button("West");
center = new Button("Center");
this.add(north, BorderLayout.NORTH);
        this.add(south, BorderLayout.SOUTH);
        this.add(east, BorderLayout.EAST);
this.add(west, BorderLayout.WEST);
        this.add(center, BorderLayout.CENTER);
        }
        public static void main(String args[])
        {
        Frame fr = new BorderLayoutDemo ("BorderLayout
Demo");      fr.pack();
        fr.setVisible(true);
        }
}
```

Kết quả thực thi chương trình:



Hình 4.6 BorderLayout

4.4.4.3 GridLayout

public class GridLayout extends Objectimplements LayoutManager Đối với một container trình bày theo kiểu GridLayout thì:

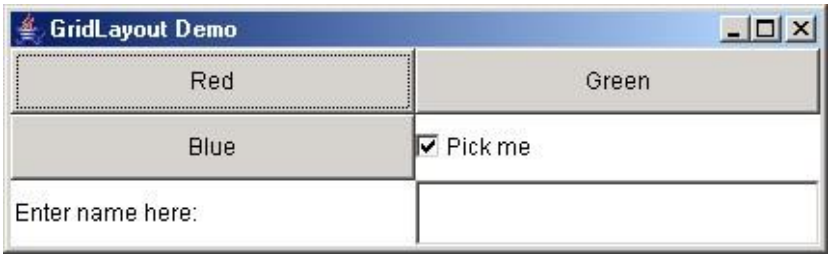
Bộ trình bày tạo một khung lưới vô hình với các ô bằng nhau.

Các đối tượng sẽđặt vừa kích thước với từng ô đó. Thứ tự sắp xếp từ trái qua phải và từ trên xuống dưới.

Ví dụ:

```
import java.awt.*; public class GridLayoutDemo
{
        public static void main(String arg[])
        {
Frame f = new Frame("GridLayout Demo");
f.setLayout(new GridLayout(3,2));
                f.add(new Button("Red"));
                f.add(new Button("Green"));
                f.add(new Button("Blue"));
                f.add(new Checkbox("Pick me", true));
                f.add(new Label("Enter name here:"));
                f.add(new TextField());

                f.pack();
                f.setVisible(true);
        }
}
```

Kết quả thực thi chương trình:



Hình 4.7 GridLayout

4.4.4.4 GridBagLayout

public class GridBagLayout extends Object implements LayoutManager2 (public interface LayoutManager2 extends LayoutManager)

Đối với một container trình bày theo kiểu GridBagLayout thì:

Các componets khi được đưa vào khung chứa sẽđược trình bày trên 1 khung lưới vô hình tương tự như GridLayout. Tuy nhiên khác với GridLayout kích thước các đối tượng không nhất thiết phải vừa với 1 ô trên khung lưới mà có thể là 2, 3 ô hay nhiều hơn tùy theo các ràng buộc mà ta chỉđịnh thông qua đối tượng GridBagConstraints.

Lớp GridBagConstraints dẫn xuất từ lớp Object. Lớp GridBagConstraints dùng để chỉđịnh ràng buộc cho những components trình bày trong khung chứa container theo kiểu GridBagLayout. o gridx, gridy: vị trí ô của khung lưới vô hình mà ta sẽđưa đối tượng con vào

gridwidth, gridheight: kích thước hay vùng trình bày cho đối tượng con.

Insets: là một biến đối tượng thuộc lớp Inset dùng để qui định khoảng cách biên phân cách theo 4 chiều (trên, dưới, trái, phải).

weightx, weighty: chỉđịnh khoảng cách lớn ra tương đối của các đối tượng con với nhau Ví dụ:

```
import java.awt.*; public class GridBagLayoutDemo
{
        public static void main(String arg[])
        {
        Frame f = new Frame("GridBagLayout Demo");

        // Thiet lap layout manager
// Tao doi tuong rang buoc cho cach trinh bay  // GridBagLayout.
        GridBagLayout layout = new GridBagLayout();
GridBagConstraints constraints = new
GridBagConstraints();
        f.setLayout(layout);

        // Tao ra 9 nut nhan
String[] buttName = {"Mot", "Hai", "Ba", "Bon",
"Nam", "Sau", "Bay", "Tam", "Chin"};
        Button[] buttons = new Button[9];

        for(int i=0;i<9;i++)
        {
        buttons[i] = new Button (buttName[i]);
        }

        // Rang buoc cac nut nhan cach nhau 2 pixel
```

```
constraints.insets = new Insets(2,2,2,2);

        // Qui dinh cac nut nhan se thay doi kich thuoc
        // theo ca 2 chieu
        constraints.fill = GridBagConstraints.BOTH;

        // Rang buoc cho nut nhan thu 1
constraints.gridx = 1;
constraints.gridy = 1;
constraints.gridheight = 2;
        constraints.gridwidth = 1;
        layout.setConstraints(buttons[0], constraints);

        // Rang buoc cho nut nhan thu 2          constraints.gridx  =  2;
constraints.gridy = 1;
constraints.gridheight = 1;
        constraints.gridwidth = 2;
        layout.setConstraints(buttons[1], constraints);

        // Rang buoc cho nut nhan thu 3
constraints.gridx = 2;
constraints.gridy = 2;
constraints.gridheight = 1;
        constraints.gridwidth = 1;
        layout.setConstraints(buttons[2], constraints);

        // Rang buoc cho nut nhan thu 4
        constraints.gridx = 1;
constraints.gridy = 3;
constraints.gridheight = 1;
        constraints.gridwidth = 2;
        layout.setConstraints(buttons[3], constraints);
```

```
        // Rang buoc cho nut nhan thu 5
constraints.gridx = 3;
constraints.gridy = 2;
constraints.gridheight = 2;
        constraints.gridwidth = 1;
        layout.setConstraints(buttons[4], constraints);
                // Rang buoc cho nut nhan thu 6
constraints.gridx = 4;
constraints.gridy = 1;
constraints.gridheight = 3;
constraints.gridwidth = 1;
        layout.setConstraints(buttons[5], constraints);

        // Tu nut thu 7 tro di khong can rang buoc
// thay vi doi kich thuoc
        constraints.fill = GridBagConstraints.NONE;

        // Rang buoc cho nut nhan thu 7
        constraints.gridx = 1;
        constraints.gridy = 4;
        constraints.gridheight = 1;
        constraints.gridwidth = 1;
        constraints.weightx = 1.0;
        layout.setConstraints(buttons[6], constraints);

        // Rang buoc cho nut nhan thu 8
        constraints.gridx = 2;
        constraints.gridy = 5;
        constraints.gridheight = 1;
        constraints.gridwidth = 1;
        constraints.weightx = 2.0;
        layout.setConstraints(buttons[7], constraints);
```

```
        // Rang buoc cho nut nhan thu 9
        constraints.gridx = 3;
        constraints.gridy = 6;
        constraints.gridheight = 1;
        constraints.gridwidth = 1;
        constraints.weightx = 3.0;
        layout.setConstraints(buttons[8], constraints);

        // Dua cac nut nhan khung chua chuong trinh
        for (int i=0;i<9;i++)
        f.add(buttons[i]);
        f.pack();
        f.setVisible(true);
        }
    }
```

Kết quả thực thi chương trình:

Hình 4.8 GridBagLayout

4.4.4.5 Null Layout

Một khung chứa được trình bày theo kiểu Null Layout có nghĩa là người lập trình phải tự làm tất cả từ việc qui định kích thước của khung chứa, cũng như kích thước và vị trí của từng đối tượng component trong khung chứa.

Để thiết lập cách trình bày là Null Layout cho một container ta chỉ việc gọi phương thức setLayout(null) với tham số là null.

Một số phương thức của lớp trừu tượng Component dùng đểđịnh vị và qui định kích thước của component khi đưa chúng vào khung chứa trình bày theo kiểu kiểu tự do:

Public void setLocation(Point p) o Public void setSize(Dimension p) o Public void setBounds(Rectangle r)

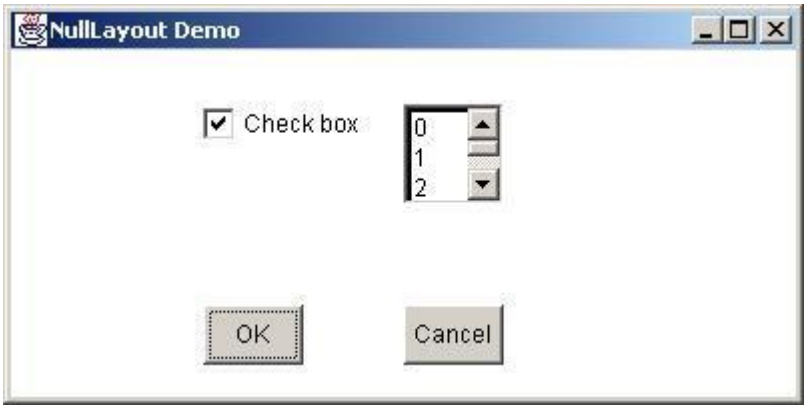
Ví dụ:

MyButton.setSize(new Dimension(20, 10)); o MyButton.setLocation(new Point(10, 10)); o MyButton.setBounds(10, 10, 20, 10);

```
import java.awt.*;
class NullLayoutDemo
{
    public static void main(String args[])
    {
        Frame fr = new Frame("NullLayout Demo");
fr.setLayout(null);
Button buttOk = new Button("OK");  buttOk.setBounds(100, 150, 50, 30);
Button buttCancel = new Button("Cancel");
buttCancel.setBounds(200, 150, 50, 30); Checkbox checkBut = new Checkbox("Check box", true);
        checkBut.setBounds(100, 50, 100, 20);

        List li = new List();
        for (int i=0; i<5; i++)
        {
            li.add(Integer.toString(i));
        }
        li.setBounds(200, 50, 50, 50);
        fr.add(buttOk);
        fr.add(buttCancel);
fr.add(checkBut);
        fr.add(li);
        fr.setBounds(10, 10, 400, 200);
fr.setVisible(true);
```

```
        }
}
```

Kết quả thực thi chương trình:



Hình 4.9 Null Layout

4.4.5.Các đối tượng khung chứa Container

Như chúng ta đã biết container là đối tượng khung chứa có khả năng quản lý và chứa các đối tượng (components) khác trong nó.

Các components chỉ có thể sử dụng được khi đưa nó vào 1 đối tượng khung chứa là container.

Mỗi container thường gắn với một LayoutManager

(FlowLayout, BorderLayout, GridLayout, GridBagLayout, Null Layout) qui định cách trình bày và bố trí các components trong một container.

Các loại container trong java: Frame, Panel, Dialog, ScrollPanes.

4.4.5.1 Khung chứa Frame

java.lang.Object +--java.awt.Component

+--java.awt.Container

+--java.awt.Window

+--java.awt.Frame

Khung chứa Frame là một cửa số window hẳn hoi ở mức trên cùng bao gồm một tiêu đều và một đường biên (border) như các ứng dụng windows thông thường khác. Khung chứa Frame thường được sử dụng để tạo ra cửa sổ chính của các ứng dụng.

Khung chứa Panel có bộ quản lý trình bày (LayoutManager) mặc định là FlowLayout.

4.4.5.2 Khung chứa Panel

java.lang.Object +--java.awt.Component

+--java.awt.Container

+--java.awt.Panel

Khung chứa Panel có bộ quản lý trình bày (LayoutManager) mặc định là FlowLayout.

Đối với khung chứa Panel thì các Panel có thể lồng vào nhau, vì vậy khung chứa Panel thường được dùng để bố trí các nhóm components bên trong một khung chứa khác.

Ví dụ: import java.awt.*;

public class PanelDemo extends Frame

{

private Button next, prev, first; private List li;

public PanelDemo(String sTitle)

{

super(sTitle); next = new Button("Next >>"); prev = new Button("<< Prev"); first = new Button("First");

Panel southPanel = new Panel(); southPanel.add(next); southPanel.add(prev); southPanel.add(first); // BorderLayout.SOUTH: vùng dưới

this.add(southPanel, BorderLayout.SOUTH);

Panel northPanel = new Panel(); northPanel.add(new Label("Make a Selection"));

// BorderLayout.NORTH: vùng trên this.add(northPanel, BorderLayout.NORTH);

li = new List();

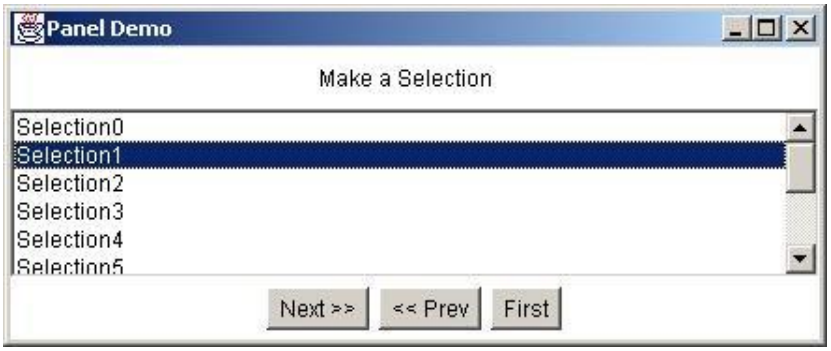
for(int i=0;i<10;i++)

{

li.add("Selection" + i);

```
        }
        this.add(li, BorderLayout.CENTER);
        }
        public static void main(String arg[])
        {
                Container f = new PanelDemo("Panel Demo");
f.setSize(300, 200);
                f.setVisible(true);
        }
}
```

Kết quả thực thi chương trình:



Hình 4.10 Khung chứa Panel

4.4.5.2 Khung chứa Dialog

java.lang.Object +--java.awt.Component

+--java.awt.Container

+--java.awt.Window

+--java.awt.Dialog

Dialog là một lớp khung chứa tựa Frame và còn được gọi là popup window. Có hai loại dialog phổ biến:

Modal Dialog: sẽ khóa tất cả các cửa số khác của ứng dụng khi dialog dạng này còn hiển thị.

Non-Modal Dialog: vẫn có thể đến các cửa số khác của ứng dụng khi dialog dạng này hiển thị.

Một cửa sổ dạng Dialog luôn luôn phải gắn với một cửa sổứng dụng (Frame).

Để tạo một đối tượng khung chứa Dialog ta có thể dùng một trong các constructor của nó:

public Dialog (Frame parentWindow, boolean isModal) public Dialog (Frame parentWindow, String title, boolean isModal) parentWindow: cửa sổ cha title: tiêu đề của Dialog isModal: true -> là Dialog dạng modal isModal: false -> là Dialog không phải dạng modal

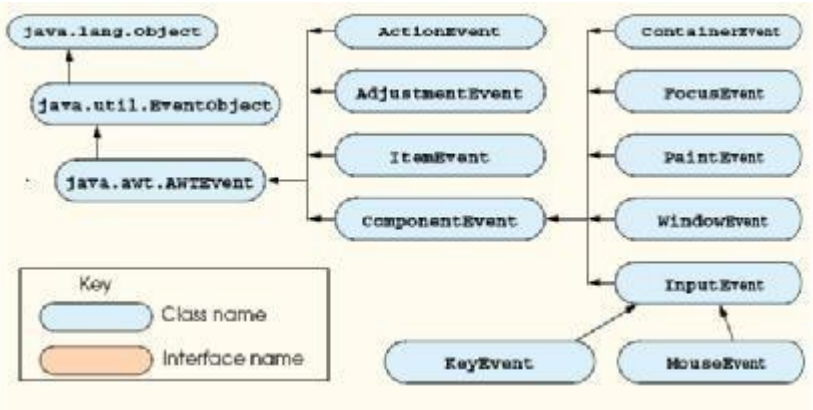
(hay non-modal)

4.5.Xử lý biến cố/sự kiện

4.5.1.Mô hình xử lý sự kiện (Event-Handling Model)

Ở trên chúng ta chỉđề cập đến vấn đề thiết kế giao diện chương trình ứng dụng mà chưa đề cập đến vấn đề xử lý sự kiện. Những sự kiện được phát sinh khi người dùng tương tác với giao diện chương trình (GUI). Những tương tác thường gặp như: di chuyển, nhấn chuột, nhấn một nút nhấn, chọn một MenuItem trong hệ thống thực đơn, nhập dữ liệu trong một ô văn bản, đóng cửa sổ ứng dụng, … Khi có một tương tác xảy ra thì một sự kiện được gởi đến chương trình. Thông tin về sự kiện thường được lưu trữ trong một đối tượng dẫn xuất từ lớp AWTEvent. Những kiểu sự kiện trong gói java.awt.event có thể dùng cho cả những component AWT và JFC. Đối với thư viện JFC thì có thêm những kiểu sự kiện mới trong gói java.swing.event.

Những lớp sự kiện của gói java.awt.event



Hình 4.11 lớp sự kiện của gói java.awt.event

Có 3 yếu tố quan trọng trong mô hình xử lý sự kiện:

- Nguồn phát sinh sự kiện (event source)
- Sự kiện (event object)

- Bộ lắng nghe sự kiện (event listener)

Nguồn phát sinh sự kiện: là thành phần của giao diện mà người dùng tác động.

Sự kiện: Tóm tắt thông tin về xử kiện xảy ra, bao gồm tham chiếu đến nguồn gốc phát sinh sự kiện và thông tin sự kiện sẽ gởi đến cho bộ lắng nghe xử lý.

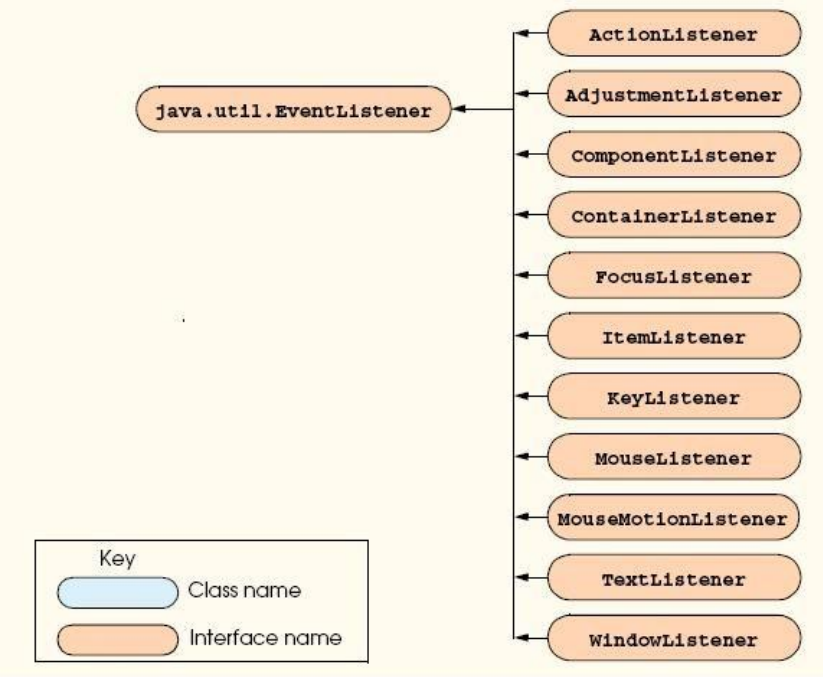
Bộ lắng nghe: Một bộ lắng nghe là một đối tượng của một lớp hiện thực một hay nhiều interface của gói java.awt.event hay java.swing.event (đối với những component trong thư viện JFC). Khi được thông báo, bộ lắng nghe nhận sự kiện và xử lý. Nguồn phát sinh sự kiện phải cung cấp những phương thức đểđăng ký hoặc hủy bỏ một bộ lắng nghe. Nguồn phát sinh sự kiện luôn phải gắn với một bộ lắng nghe, và nó sẽ thông báo với bộ lắng nghe đó khi có sự kiện phát sinh đó.

Như vậy người lập trình cần làm hai việc:

Tạo và đăng ký một bộ lắng nghe cho một component trên GUI.

Cài đặt các phương thức quản lý và xử lý sự kiện

Những interfaces lắng nghe của gói java.awt.event



Hình 4.12 Những interfaces lắng nghe của gói java.awt.event

Một đối tượng Event-Listener lắng nghe những sự kiện khác nhau phát sinh từ các components của giao diện chương trình. Với mỗi sự kiện khác nhau phát sinh thì phương thức tương ứng trong những Event-Listener sẽđược gọi thực hiện.

Mỗi interface Event-Listener gồm một hay nhiều các phương thức mà chúng cần cài đặt trong các lớp hiện thực (implements) interface đó. Những phương thức trong các interface là trừu tượng vì vậy lớp (bộ lắng nghe) nào hiện thực các interface thì phải cài đặt tất cả những phương thức đó. Nếu không thì các bộ lắng nghe sẽ trở thành các lớp trừu tượng.

4.5.2.Xử lý sự kiện chuột

Java cung cấp hai intefaces lắng nghe (bộ lắng nghe sự kiện chuột) là MouseListener và MouseMotionListener để quản lý và xử lý các sự kiện liên quan đến thiết bị chuột. Những sự kiện chuột có thể "bẫy" cho bất kỳ component nào trên GUI mà dẫn xuất từ java.awt.component.

Các phương thức của interface MouseListener:

public void mousePressed(MouseEvent event): được gọi khi một nút chuột được nhấnvà con trỏ chuột ở trên component.

public void mouseClicked(MouseEvent event): được gọi khi một nút chuột được nhấn và nhả trên component mà không di chuyển chuột.

public void mouseReleased(MouseEvent event): được gọi khi một nút chuột nhả sa khi kéo rê.

public void mouseEntered(MouseEvent event): được gọi khi con trỏ chuột vào trong đường biên của một component.

public void mouseExited(MouseEvent event): được gọi khi con trỏ chuột ra khỏi đường biên của một component.

Các phương thức của interface MouseMotionListener:

public void mouseDragged(MouseEvent even): phương thức này được gọi khi người dùng nhấn một nút chuột và kéo trên một component.

public void mouseMoved(MouseEvent event): phương thức này được gọi khi di chuyển chuột trên component.

Mỗi phương thức xử lý sự kiện chuột có một tham số MouseEvent chứa thông tin về sự kiện chuột phát sinh chẳng hạn như: tọa độ x, y nơi sự kiện chuột xảy ra. Những phương thức tương ứng trong các interfaces sẽ tựđộng được gọi khi chuột tương tác với một component.

Để biết được người dùng đã nhấn nút chuột nào, chúng ta dùng những phương thức, những hằng số của lớp InputEvent (là lớp cha của lớp MouseEvent).

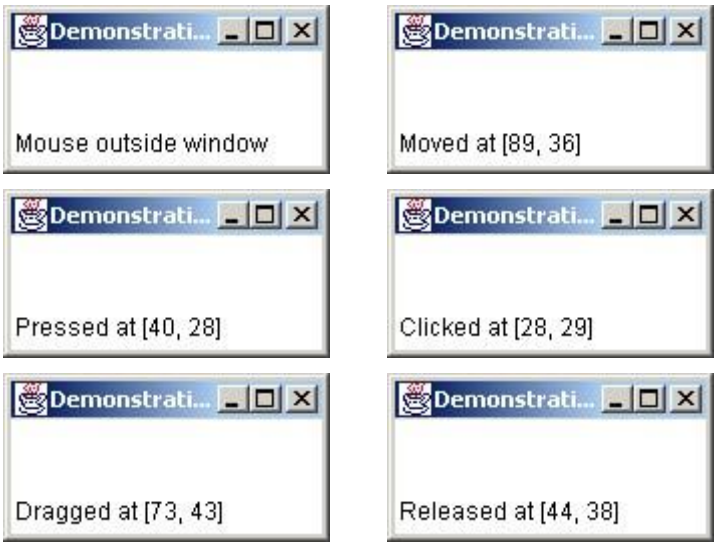
Ví dụ: Chương trình tên MouseTracker bên dưới minh họa việc dùng những phương thức của các interfaces MouseListener và MouseMotionListener để "bẫy" và xử lý các sự kiện chuột tương ứng.

```
import java.awt.*; import java.awt.event.*; public class MouseTracker
extends Frame       implements MouseListener, MouseMotionListener
{
        private Label statusBar;
        // set up GUI and register mouse event handlers     public
MouseTracker()
        {       super( "Demonstrating Mouse Events" );              statusBar
= new Label();              this.add( statusBar, BorderLayout.SOUTH );
    // application listens to its own mouse events
                addMouseListener( this );
    addMouseMotionListener( this );              setSize( 275, 100 );
                setVisible( true );
        }

        // MouseListener event handlers
        // handle event when mouse released immediately
        // after press
        public void mouseClicked( MouseEvent event )
        {
                statusBar.setText( "Clicked at [" + event.getX() +
                        ", " + event.getY() + "]" );
        }

        // handle event when mouse pressed    public    void    mousePressed(
MouseEvent event )
        {
                statusBar.setText( "Pressed at [" + event.getX() +
                        ", " + event.getY() + "]" );
        }

        // handle event when mouse released after dragging
        public void mouseReleased( MouseEvent event )
        {
```

```
            statusBar.setText( "Released at [" + event.getX() +
                ", " + event.getY() + "]" );
        }
        // handle event when mouse enters area
        public void mouseEntered( MouseEvent event )
        {
            statusBar.setText( "Mouse in window" );
    }

        // handle event when mouse exits area public void
mouseExited( MouseEvent event ) { statusBar.setText( "Mouse outside
window" );
        }
            // MouseMotionListener event handlers
    // handle event when user drags mouse with button pressed
        public void mouseDragged( MouseEvent event )
        {
            statusBar.setText( "Dragged at [" + event.getX() +
                ", " + event.getY() + "]" );
        }

        // handle event when user moves mouse public void
mouseMoved( MouseEvent event )
        {
            statusBar.setText( "Moved at [" + event.getX() +
                ", " + event.getY() + "]" );
        }

        // execute application
        public static void main( String args[] )
        {
            MouseTracker application = new MouseTracker();
        }
```

} // end class MouseTracker

Kết quả thực thi chương trình:



Hình 4.13 Xử lý sự kiện chuột

4.5.3.Xử lý sự kiện bàn phím

Để xử lý sự kiện bàn phím java hỗ trợ một bộ lắng nghe sự kiện đó là interface KeyListener. Một sự kiện bàn phím được phát sinh khi người dùng nhấn và nhả một phím trên bàn phím. Một lớp hiện thực KeyListener phải cài đặt các phương thức keyPressed, keyReleased và keyTyped. Mỗi phương thức này có một tham số là một đối tượng kiểu KeyEvent. KeyEvent là lớp con của lớp InputEvent.

Các phương thức của interface KeyListener

Phương thức keyPressedđược gọi khi một phím bất kỳđược nhấn.

Phương thức keyTyped được gọi thực hiện khi người dùng nhấn một phím không phải "phím hành động" (như phím mũi tên, phím Home, End, Page Up, Page Down, các phím chức năng như: Num Lock, Print Screen, Scroll Lock, Caps Lock, Pause).

Phương thức keyReleasedđược gọi thực hiện khi nhả phím nhấn sau khi sự kiện keyPressed hoặc keyTyped.

Ví dụ: minh họa việc xử lý sự kiện chuột thông qua các phương thức của interface KeyListener. Lớp KeyDemo bên dưới hiện thực interface KeyListener, vì vậy tất cả 3 phương thức trong KeyListener phải được cài đặt trong chương trình.

```
// KeyDemo.java
// Demonstrating keystroke events. // Java core packages import java.awt.*; import java.awt.event.*;
public class KeyDemo extends Frame implements KeyListener
{
    private String line1 = "", line2 = ""; private String line3 = "";
    private TextArea textArea;

    // set up GUI public KeyDemo() {
        super( "Demonstrating Keystroke Events" );

        // set up TextArea textArea = new TextArea( 10, 15 ); textArea.setText( "Press any key on the keyboard..." );
        textArea.setEnabled( false );
        this.add( textArea );
    // allow frame to process Key events addKeyListener( this );

        setSize( 350, 100 );
        setVisible( true );
    }
    // handle press of any key public void keyPressed( KeyEvent event )
    {
        line1 = "Key pressed: " + event.getKeyText( event.getKeyCode() ); setLines2and3( event );
    }
```

```
        // handle release of any key      public      void      keyReleased(
KeyEvent event )
        {
                line1 = "Key released: " +              event.getKeyText(
event.getKeyCode() );          setLines2and3( event );
        }
                // handle press of an action key       public          void
keyTyped( KeyEvent event )
        {
                line1 = "Key typed: " + event.getKeyChar();
    setLines2and3( event );
        }

        // set second and third lines of output
        private void setLines2and3( KeyEvent event )
        {
    line2 = "This key is " + ( event.isActionKey() ? "" : "not
    " ) + "an action key";

        String      temp      =      event.getKeyModifiersText(
    event.getModifiers() );
    line3 = "Modifier keys pressed: " + ( temp.equals( "" ) ?
    "none" : temp );

                textArea.setText(line1+"\n"+line2+"\n"+ line3+"\n" );
        }

        // execute application
        public static void main( String args[] )
        {
                KeyDemo application = new KeyDemo();
        }
```

} // end class KeyDemo

Kết quả thực thi chương trình:

Hình 4.14 Xử lý sự kiện bàn phím

4.6. Bài tập

Bài tập 1: Tạo bộ lắng nghe biến cố cho đối tượng khung chứa Frame, và xử lý biến cốđóng cửa sổ.

import java.awt.; import java.awt.event.*; public class*

```
WindowClosingDemo
    {
        public static void main(String args[])
        {
            Frame f = new Frame ("WindowClosing Demo");
    WindowCloser closer = new WindowCloser();
    f.addWindowListener(closer);
            f.setBounds(10, 10, 300, 200);
        f.setVisible(true);
        }
    }

    import java.awt.event.*;
    class WindowCloser implements WindowListener
    {
        public void windowClosing(WindowEvent e)
        {
            System.out.println("windowClosing..");
            System.exit(0);
        }
        public void windowActivated(WindowEvent e)
        {
            System.out.println("windowActivated...");
        }
        public void windowClosed(WindowEvent e)
        {
            System.out.println("windowClosed...");
        }
        public void windowDeactivated(WindowEvent e)
        {
            System.out.println("windowDeactivated...");
        }
        public void windowDeiconified(WindowEvent e)
```

```
        {
            System.out.println("windowDeiconified...");
        }
        public void windowIconified(WindowEvent e)
        {
            System.out.println("windowIconified...");
        }
        public void windowOpened(WindowEvent e)
        {   System.out.println("windowOpened...");
        }
}
```

Có thể dùng lớp trừu tượng WindowAdapter để tạo ra bộ lắng nghe.

public abstract class WindowAdapter extends Object implements WindowListener

(WindowAdapter hiện thực interface WindowListener nên lớp ảo này cũng có 7 phương thức giống như giao diện WindowListener)

```
import java.awt.event.*; class WindowCloser extends WindowAdapter {
public void windowClosing(WindowEvent e)
        {   System.out.println("windowClosing..");
            System.exit(0);
        }
}
```

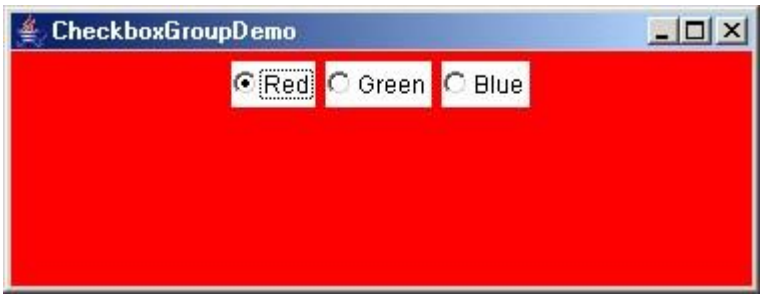
Bài tập 2:

```
CheckboxGroup Demo import java.awt.*;
public class CheckboxGroupDemo extends Frame
{
    private Checkbox red, green, blue;
private CheckboxGroup checkGroup;
public CheckboxGroupDemo(String title)
    {
super(title);
checkGroup = new CheckboxGroup();
```

```
        red = new Checkbox("Red", checkGroup, false);  green = new Checkbox("Green", checkGroup, false);  blue = new Checkbox("Blue", checkGroup, false);  //add the checkboxes to the frame
        Panel north = new Panel();
    north.add(red);
        north.add(green);
        north.add(blue);
            this.add(north, BorderLayout.NORTH);
            //register the event listener
            SetColor listener = new SetColor(this);
        red.addItemListener(listener);
    green.addItemListener(listener);
    blue.addItemListener(listener);
        }
        public static void main(String [] args)
        {
            Frame f = new
    CheckboxGroupDemo("CheckboxGroupDemo");
            f.setSize(300,300);
            f.setVisible(true);
        }
    } // end of class
        import java.awt.*; import java.awt.event.*;
        public class SetColor implements ItemListener
        {
            private Frame pallette;    private Color c;
            public SetColor(Frame c)
            {
                pallette = c;
            }

            public void itemStateChanged(ItemEvent e)
```

```
        {
            String item = (String) e.getItem();          int
        state = e.getStateChange();
            if (item.equalsIgnoreCase("red"))
                c = new Color(255, 0, 0);
            if (item.equalsIgnoreCase("green"))
                c = new Color(0, 255, 0);
      if (item.equalsIgnoreCase("blue"))
    c = new Color(0, 0, 255);
    pallette.setBackground(c);
        }
    } // end of class
```

Kết quả thực thi chương trình:



Hình 4.15 Tạo bộ lắng nghe biến cố cho đối tượng khung chứa Frame, và xử lý biến cốđóng cửa sổ.

Bài tập 3: *TextComponent import java.awt.*;*

```
class TextComponentDemo extends Frame
{
    private TextField textField;      private TextArea textArea;
    private Button enter, clear;

    public TextComponentDemo (String title)
    {
        super(title);
textArea = new TextArea("", 0, 0,
TextArea.SCROLLBARS_VERTICAL_ONLY);
```

```
textArea.setEditable(false);

                textField = new TextField();          enter = new
Button("Enter");          clear = new Button("Clear");

                //layout the GUI
                this.add(textArea, BorderLayout.CENTER);

        Panel southEast = new Panel(new BorderLayout());
southEast.add(enter, BorderLayout.EAST);
                southEast.add(clear, BorderLayout.WEST);
                Panel south = new Panel(new BorderLayout());
        south.add(textField, BorderLayout.CENTER);
                south.add(southEast, BorderLayout.EAST);
                this.add(south, BorderLayout.SOUTH);
        //setup the event handling
        CreateList listener = new CreateList(textField, textArea);
        enter.addActionListener(listener);
        clear.addActionListener(listener);
                textField.addActionListener(listener);
        }
        public TextField getTextField()
        {
                return textField;
        }
        public static void main(String [] args)
        {
                TextComponentDemo f = new TextComponentDemo
        ("TextComponentDemo ");
                f.setSize(300,200);
                f.setVisible(true);
                f.getTextField().requestFocus();
        }
```

```
}
import java.awt.*; import java.awt.event.*;
public class CreateList implements ActionListener
{
        private int counter = 0;   private TextField source;
        private TextArea destination;
        public CreateList(TextField s, TextArea d)
        {       source = s;          destination = d;
        }

        public void actionPerformed(ActionEvent e)
        {
    String      action      =      e.getActionCommand();          if
(action.equalsIgnoreCase("Enter"))
                {
                        String text = source.getText();
                        counter++;
                        destination.append(counter + "." + text + "\n");
                        source.setText("");
                }
                else
                        if (action.equalsIgnoreCase("Clear"))
                        {
                                destination.setText("");
                                counter = 0;
                        }
        }
}
```

Kết quả thực thi chương trình:

Hình 4.16 *TextComponentDemo*

Bài tập 4: *ListDemo import java.awt.*;*

public class ListDemo extends Frame

{ private List li; private Label selected;

public ListDemo(String title)

{

super(title);

li = new List();

li.add("Monday");

li.add("Tuesday");

li.add("Wednesday");

li.add("Thursday");

li.add("Friday");

li.add("Saturday");

li.add("Sunday");

selected = new Label("Double click a day:", Label.CENTER);

this.setLayout(new BorderLayout());

this.add(selected , BorderLayout.NORTH);

this.add(li, BorderLayout.CENTER);

// Tao listener cho List ShowSelectionListener listener = new ShowSelectionListener(selected); li.addActionListener(listener);

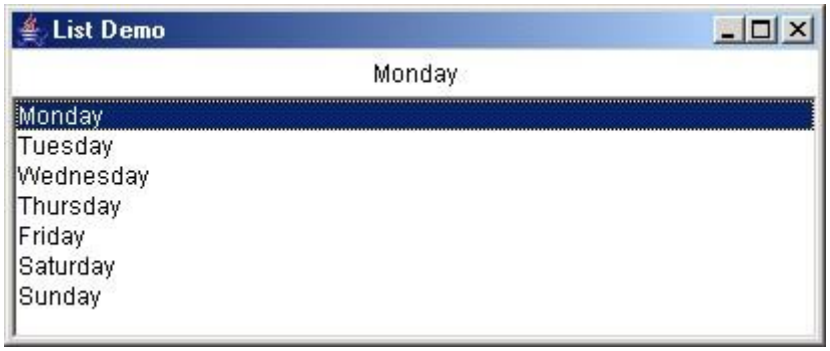
}

public static void main(String args[]) { ListDemo f = new

```
ListDemo("List Demo");                f.setBounds(10, 10, 300, 200);
                f.setVisible(true);
        }
    }

    import java.awt.*; import java.awt.event.*;
    class ShowSelectionListener implements ActionListener
    {   private Label lab;
        public ShowSelectionListener(Label label_sel)
        {
            lab = label_sel;
        }

        public void actionPerformed(ActionEvent e)
        {   // Tra ve Object ma Event da xuat hien // getSource la phuong
thuc ke thua tu
    // java.util.EventObject
            Object source = e.getSource();

      // Nguon goc phat sinh bien co khong phai la List   if (!(source instanceof
List))



            {   return;
            }
            else
            {
                List li = (List) source;
                String selected = li.getSelectedItem();
        lab.setText(selected);
            }
        }
```

}

Kết quả thực thi chương trình:



Hình 4.17 *ListDemo*

Bài tập 5: Xây dựng 1 lớp khung chứa Dialog dùng để hiển thị message giống như hàm *MessageBox trên Windows.*

```
import java.awt.*; import java.awt.event.*;
class DialogDemo
{
        public static void main(String[] args)
        {
                createMenu();
        }
                private static void createMenu()
        {
 // Tao Frame ung dung   final Frame fr = new Frame();   fr.setLayout(new
BorderLayout());
        // Tao cac menu bar
                MenuBar menubar = new MenuBar();
                Menu mTest = new Menu("Test");
                MenuItem testDlg = new MenuItem("Test Dialog");
testDlg.addActionListener(
                new ActionListener()
                {
                        public void actionPerformed(ActionEvent e)
                        {
MessageBox msgBox = new
```

```
MessageBox(fr, "Here it is", "T/bao
Dialog");
                        msgBox.show();
                }
        }
        );

        mTest.add(testDlg);          menubar.add(mTest);
fr.setMenuBar(menubar);          fr.setBounds(100, 100, 300, 200);
    fr.setVisible(true);

        fr.addWindowListener(
                new WindowAdapter()
        {
                public void windowClosing(WindowEvent e)
                        {
                                System.exit(0);
                        }
                }
        );
    }// end of createmenu()
} // end of class
import java.awt.*; import java.awt.event.*;
public class MessageBox
{
        Dialog msgBox;
        /* ------------------------------------------------------------  //
Contructor cua lop MessageBox
        // parentWindow: cua so cha
        // title: Tieu de cua Dialog
        // msg: chuoi thong bao
        ------------------------------------------------------------*/      public
MessageBox(Frame parentWindow, String msg,
```

```
String title)
    {
        if (parentWindow == null)
        {
            Frame emptyWin = new Frame();
            // Tao Modal Dialog (tham so thu 3:true)
            msgBox = new Dialog(emptyWin, title, true);
        }
        else
        {
            msgBox = new Dialog(parentWindow, title, true);
        }
        // Doi tuong nhan dung de trinh bay cau thong bao
Label Message = new Label(msg);
        // Thiet lap che do trinh bay layout cho cac doi tuong.
msgBox.setLayout(new FlowLayout());  // Dua nhan thong bao Label vao khung
chua Dialog  msgBox.add(Message);
        // Dua nut nhan OK vao trong khung chua Dialog
Button okButton = new Button("OK");          msgBox.add(okButton);
 // Khai bao kich thuoc cua cua so thong bao  msgBox.setSize(200, 100);

        // Xu ly tinh huong khi nguoi dung nhan nut OK
        okButton.addActionListener(
            new ActionListener()
    {
                public void actionPerformed(ActionEvent evt)
                {
                    msgBox.setVisible(false);
                }
            }
        );
    }
        public void show()
```

```
    {
        msgBox.show();
    }
} // end of class MessageBox
```

Kết quả thực thi chương trình:

Hình 4.18 DialogDemo

CHƯƠNG 5: LUỒNG VÀ TẬP TIN

Mã bài : MH 29.5

Mục tiêu:

– Sinh viên hiểu được các khái niệm và công dụng của luồng, tập tin trong lập trình Java;

– Biết cách phân loại, cách sử dụng các loại luồng byte, luồng ký tự,...;

– Hiểu rõ các khái niệm, tác dụng, thuộc tính, phương thức của lớp InputStream và OutputStream, lớp RandomAccesFile;

– Viết và thực thi được các chương trình trao đổi dữ liệu với các loại tập tin sử dụng luồng byte, luồng ký tự.

5.1.Mở đầu

Việc lưu trữ dữ liệu trong các biến chương trình, các mảng có tính chất tạm thời và dữ liệu sẽ mất đi khi biến ra khỏi tầm ảnh hưởng của nó hoặc khi chương trình kết thúc. Files giúp cho các chương trình có thể lưu trữ một lượng lớn dữ liệu, cũng như có thể lưu trữ dữ liệu trong một thời gian dài ngay cả khi chương trình kết thúc. Trong chương này chúng ta sẽ tìm hiểu làm thế nào các chương trình java có thể tạo, đọc, ghi và xử lý các files tuần tự và các file truy cập ngẫu nhiên thông qua một số ví dụ minh họa.

Xử lý files là một vấn đề hết sức cơ bản, quan trọng mà bất kỳ một ngôn ngữ lập trình nào cũng phải hỗ trợ những thư viện, hàm để xử lý một số thao tác cơ bản nhất đối với kiểu dữ liệu file.

Xử lý files là một phần của công việc xử lý các luồng, giúp cho một chương trình có thểđọc, ghi dữ liệu trong bộ nhớ, trên files và trao đổ dữ liệu thông qua các kết nối trên mạng.

Chương này sẽ cung cấp cho chúng ta những kiến thức cơ bản về luồng (streams) và files:

Thư viện các lớp về luồng trong java: luồng byte, luồng ký tự.

Xuất nhập Console dùng luồng byte, luồng ký tự.

Xuất nhập files dùng luồng ký tự và luồng byte.

Vấn đề xử lý files truy cập ngẫu nhiên dùng lớp RandomAccessFile.

Xử lý file và thư mục dùng lớp File.

5.2.Luồng (Streams)

5.2.1.Khái niệm luồng

Tất cả những hoạt động nhập/xuất dữ liệu (nhập dữ liệu từ bàn phím, lấy dữ liệu từ mạng về, ghi dữ liệu ra đĩa, xuất dữ liệu ra màn hình, máy in, …) đều được quy về một khái niệm gọi là luồng (stream). Luồng là nơi có thể "sản xuất" và "tiêu thụ" thông tin. Luồng thường được hệ thống xuất nhập trong java gắn kết với một thiết bị vật lý. Tất cả các luồng đều có chung một nguyên tắc hoạt độngngay cả khi chúng được gắn kết với các thiết bị vật lý khác nhau. Vì vậy cùng một lớp, phương thức xuất nhập có thể dùng chung cho các thiết bị vật lý khác nhau. Chẳng hạn cùng một phương thức có thể dùng để ghi dữ liệu ra console, đồng thời cũng có thể dùng để ghi dữ liệu xuống một file trên đĩa. Java hiện thực luồng bằng tập hợp các lớp phân cấp trong gói java.io.

Java định nghĩa hai kiểu luồng: byte và ký tự (phiên bản gốc chỉđịnh nghĩa kiểu luồng byte, và sau đó luồng ký tựđược thêm vào trong các phiên bản về sau).

Luồng byte (hay luồng dựa trên byte) hỗ trợ việc xuất nhập dữ liệu trên byte, thường được dùng khi đọc ghi dữ liệu nhị phân.

Luồng ký tựđược thiết kế hỗ trợ việc xuất nhập dữ liệu kiểu ký tự (Unicode). Trong một vài trường hợp luồng ký tự sử dụng hiệu quả hơn luồng byte, nhưng ở mức hệ thống thì tất cả những xuất nhập đều phải qui về byte. Luồng ký tự hỗ trợ hiệu quả chỉđối với việc quản lý, xử lý các ký tự.

5.2.2.Luồng byte (Byte Streams)

Các luồng byte được định nghĩa dùng hai lớp phân cấp.

Mức trên cùng là hai lớp trừu tượng InputStream và OutputStream. InputStreamđịnh nghĩa những đặc điểm chung cho những luồng nhập byte. OutputStream mô tả cách xử lý của các luồng xuất byte.

Các lớp con dẫn xuất từ hai lớp InputStream và

OutputStream sẽ hỗ trợ chi tiết tương ứng với việc đọc ghi dữ liệu trên những thiết bị khác nhau. Đừng choáng ngợp với hàng loạt rất nhiều các lớp khác nhau. Đừng quá lo lắng, mỗi khi bạn nắm vững, sử dụng thành thạo một luồng byte nào đó thì bạn dễ dàng làm việc với những luồng còn lại.

Lớp luồng byte	Ý nghĩa
BufferedInputStream	Buffered input stream
BufferedOutputStream	Buffered output stream
ByteArrayInputStream	Input stream đọc dữ liệu

	từ một mảng byte
ByteArrayOutputStream	Output stream ghi dữ liệu đến một mảng byte
DataInputStream	Luồng nhập có những phương thức đọc những kiểu dữ liệu chuẩn trong java
DataOutputStrem	Luồng xuất có những phương thức ghi những kiểu dữ liệu chuẩn trong java
FileInputStream	Luồng nhập cho phép đọc dữ liệu từ file
FileOutputStrem	Luồng xuất cho phép ghi dữ liệu xuống file
FilterInputStrem	Hiện thực lớp trừu tượng InputStream
FilterOutputStrem	Hiện thực lớp trừu tượng OutputStream
InputStream	Lớp trừu tượng, là lớp cha của tất cả các lớp luồng nhập kiểu Byte
OutputStream	Lớp trừu tượng, là lớp cha của tất cả các lớp xuất nhập kiểu Byte
PipedInputStream	Luồng nhập byte kiểu ống (piped) thường phải được gắn với một luồng xuất kiểu ống.
PipedOutputStream	Luồng nhập byte kiểu ống (piped) thường phải được gắn với một luồng nhập kiểu ống để tạo nên một kết nối trao đổi dữ liệu kiểu ống.

PrintStream	Luồng xuất có chứa phương thức print() và println()
PushbackInputStream	Là một luồng nhập kiểu Byte mà hỗ trợ thao tác trả lại (push back) và phục hồi thao tác đọc một byte (unread)
RandomAccessFile	Hỗ trợ các thao tác đọc, ghi đối với file truy cập ngẫu nhiên.
SequenceInputStream	Là một luồng nhập được tạo nên bằng cách nối kết logic các luồng nhập khác.

5.2.3.Luồng ký tự (Character Streams)

Các luồng ký tựđược định nghĩa dùng hai lớp phân cấp.

Mức trên cùng là hai lớp trừu tượng Reader và Writer. Lớp Reader dùng cho việc nhập dữ liệu của luồng, lớp Writer dùng cho việc xuất dữ liệu cua luồng. Những lớp dẫn xuất từReader và Writer thao tác trên các luồng ký tự Unicode.

Lớp luồng ký tự	Ý nghĩa
BufferedReader	Luồng nhập ký tựđọc dữ liệu vào một vùng đệm.
BufferedWriter	Luồng xuất ký tự ghi dữ liệu tới một vùng đệm.
CharArrayReader	Luồng nhập đọc dữ liệu từ một mảng ký tự
CharArrayWrite	Luồng xuất ghi dữ liệu tời một

r	mảng ký tự
FileReader	Luồng nhập ký tựđọc dữ liệu từ file
FileWriter	Luồng xuất ký tự ghi dữ liệu đến file
FilterReader	Lớp đọc dữ liệu trung gian (lớp trừu tượng)
FilterWriter	Lớp xuất trung gian trừu tượng
InputStreamReader	Luồng nhập chuyển bytes thành các ký tự
LineNumberReader	Luồng nhập đếm dòng
OutputStreamWriter	Luồng xuất chuyển những ký tự thành các bytes
PipedReader	Luồng đọc dữ liệu bằng cơ chếđường ống
PipedWriter	Luồng ghi dữ liệu bằng cơ chếđường ống
PrintWriter	Luồng ghi văn bản ra thiết bị xuất (chứa phương thức print() và println())
PushbackReader	Luồng nhập cho phép đọc và khôi phục lại dữ liệu
Reader	Lớp nhập dữ liệu trừu tượng
StringReader	Luồng nhập đọc dữ liệu từ chuỗi
StringWriter	Luồng xuất ghi dữ liệu ra

	chuỗi
Writer	Lớp ghi dữ liệu trừu tượng

5.2.4.Những luồng được định nghĩa trước (The Predefined Streams)

Tất cả các chương trình viết bằng java luôn tựđộng import gói java.lang. Gói này có định nghĩa lớp System, bao gồm một sốđặc điểm của môi trường run-time, nó có ba biến luồng được định nghĩa trước là in, out và err, các biến này là các fields được khai báo static trong lớp System.

System.out: luồng xuất chuẩn, mặc định là console. System.out là một đối tượng kiểu PrintStream.

System.in: luồng nhập chuẩn, mặc định là bàn phím. System.in là một đối tượng kiểu InputStream.

System.err: luồng lỗi chuẩn, mặc định cũng là console. System.out cũng là một đối tượng kiểu PrintStream giống System.out.

5.3.Sử dụng luồng Byte

Như chúng ta đã biết hai lớp InputStream và OutputStream là hai siêu lớp (cha) đối với tất cả những lớp luồng xuất nhập kiểu byte. Những phương thức trong hai siêu lớp này ném ra các lỗi kiểu IOException. Những phương thức định nghĩa trong hai siêu lớp này là có thể dùng trong các lớp con của chúng. Vì vậy tập các phương thức đó là tập tối tiểu các chức năng nhập xuất mà những luồng nhập xuất kiểu byte có thể sử dụng.

Những phương thức định nghĩa trong lớp

InputStream và OutputStream

Phương thức	Ý nghĩa
InputStream	
int available( )	Trả về số lượng bytes có thểđọc được từ luồng nhập
void close( )	Đóng luồng nhập và giải phóng tài nguyên hệ thống gắn với luồng. Không thành công sẽ ném ra một lỗi IOException

void mark(int numBytes)	Đánh dấu ở vị trí hiện tại trong luồng nhập
boolean markSupported( )	Kiểm tra xem luồng nhập có hỗ trợ phương thức mark() và reset() không.
int read( )	Đọc byte tiếp theo từ luồng nhập
int read(byte buffer[ ])	Đọc buffer.length bytes và lưu vào trong vùng nhớ buffer. Kết quả trả về số bytes thật sựđọc được
int read(byte buffer[ ], int offset, int numBytes)	Đọc numBytes bytes bắt đầu từđịa chỉ offset và lưu vào trong vùng nhớ buffer. Kết quả trả về số bytes thật sựđọc được
void reset( )	Nhảy con trỏđến vị trí được xác định bởi việc gọi hàm mark() lần sau cùng.
long skip(long numBytes)	Nhảy qua numBytes dữ liệu từ luồng nhập
OutputStream	
void close( )	Đóng luồng xuất và giải phóng tài nguyên hệ thống gắn với luồng. Không thành công sẽ ném ra một lỗi IOException
void flush( )	Ép dữ liệu từ bộđệm phải ghi ngay xuống luồng (nếu có)
void write(int b)	Ghi byte dữ liệu chỉđịnh xuống luồng

void write(byte buffer[ ])	Ghi buffer.length bytes dữ liệu từ mảng chỉđịnh xuống luồng
void write(byte buffer[ ], int offset, int numBytes)	Ghi numBytes bytes dữ liệu từ vị trí offset của mảng chỉđịnh buffer xuống luồng

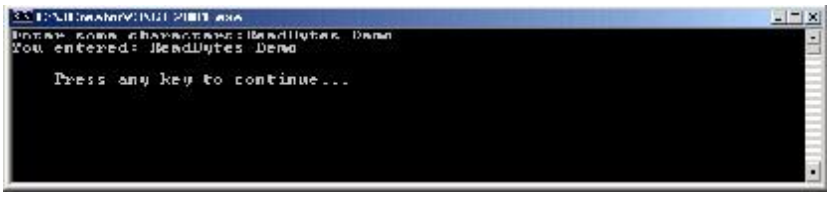
5.3.1.Đọc dữ liệu từ Console

Trước đây, khi Java mới ra đời để thực hiện việc nhập dữ liệu từ Console người ta chỉ dùng luồng nhập byte. Về sau thì chúng ta có thể dùng cả luồng byte và luồng ký tự, nhưng trong một số trường hợp thực tếđểđọc dữ liệu từ Console người ta thích dùng luồng kiểu ký tự hơn, vì lý do đơn giản và dễ bảo trì chương trình. Ởđây với mục đích minh họa chúng ta dùng luồng byte thực hiện việc nhập xuất Console.

Ví dụ: chương trình minh họa việc đọc một mảng bytes từ System.in

```
Import java.io.*;
class ReadBytes
{ public static void main(String args[]) throws IOException
{
byte data[] = new byte[100]; System.out.print("Enter some characters.");
System.in.read(data); System.out.print("You entered: "); for(int i=0; i < data.length; i++)
System.out.print((char) data[i]); }
}
```

Kết quả thực thi chương trình:



Hình 5.1 Đọc dữ liệu từ Console

5.3.2.Xuất dữ liệu ra Console

Tương tự như nhập dữ liệu từ Console, với phiên bản đầu tiên của java để xuất dữ liệu ra Console tả chỉ có thể sử dụng luồng byte. Kể từ phiên bản 1.1 (có thêm luồng ký tự), để xuất dữ liệu ra Console có thể sử dụng cả luồng ký tự và luồng byte. Tuy nhiên, cho đến nay để xuất dữ liệu ra Console thường người ta vẫn dùng luồng byte.

Chúng ta đã khá quen thuộc với phương thức print() và println(), dùng để xuất dữ liệu ra Console. Bên cạnh đ1o chúng ta cũng có thể dùng phương thức write().

Ví dụ: minh họa sử dụng phương thức System.out.write() để xuất ký tự 'X' ra *Console import java.io.*; class WriteDemo*

```
{
        public static void main(String args[])
        {
           int b;
b = 'X';
           System.out.write(b);
           System.out.write('\n');
        }
}
```

Kết quả thực thi chương trình:

Hình 5.2 Xuất dữ liệu ra Console

5.3.3.Đọc và ghi file dùng luồng Byte

Tạo một luồng Byte gắn với file chỉđịnh dùng

FileInputStream và FileOutputStream. Để mở một file, đơn giản chỉ cần tạo một đối tượng của những lớp này, tên file cần mở là thông số trong constructor. Khi file mở, việc đọc và ghi dữ liệu trên file được thực hiện một cách bình thường thông qua các phương thức cung cấp trong luồng.

5.3.3.1 Đọc dữ liệu từ file

Mở một file đểđọc dữ liệu

FileInputStream(String fileName) throws

FileNotFoundException

Nếu file không tồn tại: thì ném ra

FileNotFoundException

Đọc dữ liệu: dùng phương thức read()int read( ) throws IOException: đọc từng byte từ file và trả về giá trị của byte đọc được. Trả về -1 khi hết file, và ném ra IOException khi có lỗi đọc.

Đóng file: dùng phương thức close()void close( ) throws IOException: sau khi làm việc xong cần đóng file để giải phóng tài nguyên hệ thống đã cấp phát cho file.

Ví dụ:

```
/*
Hiển thị nội dung của một file tên test.txt lưu tạiD:\test.txt
*/import java.io.*; class ShowFile
 {
public static void main(String args[]) throws IOException
                {
int i;
FileInputStream fin; try
                {
                        fin = new FileInputStream(“D:\\test.txt”);
                }
 catch(FileNotFoundException exc)
{
                        System.out.println("File Not Found");
                        return;
}
```

```
catch(ArrayIndexOutOfBoundsException exc)
{
                System.out.println("Usage: ShowFile File");
                return;
}

        // read bytes until EOF is encountered do
{
                i = fin.read();
                if(i != -1) System.out.print((char) i);
} while(i != -1); fin.close();
        }
        }
```

Kết quả thực thi chương trình:

Hình 5.3 Đọc và ghi file dùng luồng Byte

5.3.3.2 Ghi dữ liệu xuống file

Mở một file để ghi dữ liệu

FileOutputStream(String fileName) throws

FileNotFoundException

Nếu file không tạo được: thì ném ra

FileNotFoundException

Ghi dữ liệu xuống: dùng phương thức write()void write(int byteval) throws IOException: ghi một byte xác định bởi tham sốbyteval xuống file, và ném ra IOException khi có lỗi ghi.

Đóng file: dùng phương thức close()

void close( ) throws IOException:sau khi làm việc xong cần đóng file để giải phóng tài nguyên hệ thống đã cấp phát cho file.

Ví dụ: copy nội dung một file text đến một file text khác.

/* Copy nội dung của một file text*/ import java.io.*; class CopyFile

```
{ public static void main(String args[])throws IOException
{ int i;
FileInputStream fin;
FileOutputStream fout;
        try
        {
            // open input file
            try
            {
            fin = new FileInputStream("D:\\source.txt");
            }
            catch(FileNotFoundException exc)
            {
            System.out.println("Input File Not Found");
            return;
            }

            // open output file
            try
            {
            fout = new FileOutputStream("D:\\dest.txt");
            }
            catch(FileNotFoundException exc)
            {
System.out.println("Error Opening Output
File");      return;
            }
        }
         catch(ArrayIndexOutOfBoundsException exc)
```

```
        {
        System.out.println("Usage: CopyFile From To");
    return;   }

    // Copy File     try
    {          do
               {
                    i = fin.read();
                    if(i != -1)
                     fout.write(i);
               } while(i != -1);
        }
        catch(IOException exc)
        {
               System.out.println("File Error");
        }
    fin.close();       fout.close();
}
}
```

Kết quả thực thi chương trình: chương trình sẽ copy nội dung của file D:\source.txt và ghi vào một file mới D:\dest.txt.

5.3.4.Đọc và ghi dữ liệu nhị phân

Phần trên chúng ta đã đọc và ghi các bytes dữ liệu là các ký tự mã ASCII. Đểđọc và ghi những giá trị nhị phân của các kiểu dữ liệu trong java, chúng ta sử dụng DataInputStream và DataOutputStream.

DataOutputStream: hiện thực interface DataOuput. Interface DataOutput có các phương thức cho phép ghi tất cả những kiểu dữ liệu cơ sở của java đến luồng (theo định dạng nhị phân).

Phương thức	Ý nghĩa
void writeBoolean (boolean val)	Ghi xuống luồng một giá trị boolean được xác định bởi val.
void	Ghi xuống luồng một

writeByte (int val)	byte được xác định bởi val.
void writeChar (int val)	Ghi xuống luồng một Char được xác định bởi val.
void writeDouble (double val)	Ghi xuống luồng một giá trị Double được xác định bởi val.
void writeFloat (float val)	Ghi xuống luồng một giá trị float được xác định bởi val.
void writeInt (int val)	Ghi xuống luồng một giá trị int được xác định bởi val.
void writeLong (long val)	Ghi xuống luồng một giá trị long được xác định bởi val.
void writeShort (int val)	Ghi xuống luồng một giá trị short được xác định bởi val.

Contructor**:** *DataOutputStream(OutputStream outputStream)*

OutputStream: là luồng xuất dữ liệu. Để ghi dữ liệu ra file thì đối tượng outputStream có thể là FileOutputStream.

DataInputStream: hiện thực interface DataInput. Interface DataInput có các phương thức cho phép đọc tất cả những kiểu dữ liệu cơ sở của java (theo định dạng nhị phân).

Phương thức	Ý nghĩa
boolean readBoolean( )	Đọc một giá trị boolean
Byte	Đọc một byte

readByte( )	
char readChar( )	Đọc một Char
double readDouble( )	Đọc một giá trị Double
float readFloat( )	Đọc một giá trị float
int readInt( )	Đọc một giá trị int
Long readLong( )	Đọc một giá trị long
short readShort( )	Đọc một giá trị short

Contructor: DataInputStream(InputStream inputStream)
InputStream: là luồng nhập dữ liệu. Đểđọ dữ liệu từ file thì đối tượng *InputStream* có thể là *FileInputStream.*

Ví dụ: dùng DataOutputStream và DataInputStream để ghi và đọc những kiểu dữ liệu khác nhau trên file.*import java.io.*;*

```
class RWData
{
        public static void main(String args[]) throws IOException
        {
           DataOutputStream dataOut;          DataInputStream
dataIn;          int i = 10;          double d = 1023.56;
           boolean b = true;          try
           {
dataOut = new DataOutputStream(
                   new FileOutputStream("D:\\testdata"));
           }
           catch(IOException exc)
           {
                   System.out.println("Cannot open file.");
                   return;
```

```
}

try
{
        System.out.println("Writing " + i);
        dataOut.writeInt(i);
        System.out.println("Writing " + d);
        dataOut.writeDouble(d);
        System.out.println("Writing " + b);
        dataOut.writeBoolean(b);
        System.out.println("Writing " + 12.2 * 7.4);
                dataOut.writeDouble(12.2 * 7.4);
}
catch(IOException exc)
{
        System.out.println("Write error.");
}

dataOut.close();            System.out.println();

// Now, read them back.          try
{
        dataIn = new DataInputStream(
                new FileInputStream("D:\\testdata"));
}
catch(IOException exc)
{
        System.out.println("Cannot open file.");
        return;
}
try
{
```

```
            i = dataIn.readInt();
        System.out.println("Reading " + i);
        d = dataIn.readDouble();
        System.out.println("Reading " + d);
           b = dataIn.readBoolean();
        System.out.println("Reading " + b);
        d = dataIn.readDouble();
        System.out.println("Reading " + d);
           }
           catch(IOException exc)
        {  System.out.println("Read error.");
           }
           dataIn.close();
        }
}
```

Kết quả thực thi chương trình:

Dữ liệu ghi xuống file D:\\testdata

Hình 5.4 Dữ liệu ghi xuống file D:\\testdata

Kết quả đọc và xuất ra Console:

Hình 5.5 Kết quả đọc và xuất ra Console

5.4.File truy cập ngẫu nhiên (Random Access Files)

Bên cạnh việc xử lý xuất nhập trên file theo kiểu tuần tự thông qua các luồng, java cũng hỗ trợ truy cập ngẫu nhiên nội dung của một file nào đó dùng RandomAccessFile. RandomAccessFile không dẫn xuất từ InputStream hay

OutputStream mà nó hiện thực các interface DataInput, DataOutput (có định nghĩa các phương thức I/O cơ bản). RandomAccessFile hỗ trợ vấn đềđịnh vị con trỏ file bên trong một file dùng phương thức seek(long newPos).

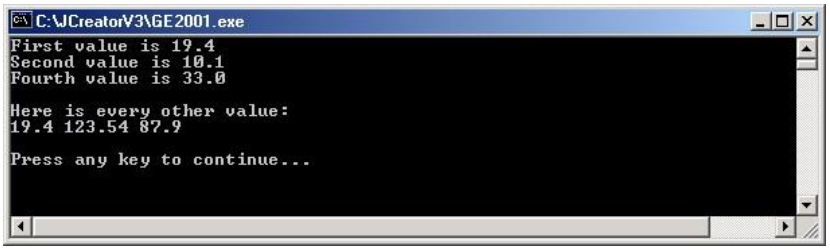
Ví dụ: minh họa việc truy cập ngẫu nhiên trên file. Chương trình ghi 6 số kiểu double xuống file, rồi đọc lên theo thứ tự ngẫu nhiên.

```
import java.io.*;
class RandomAccessDemo
{
        public static void main(String args[]) throws IOException
        {
  double data[] = {19.4, 10.1, 123.54, 33.0, 87.9, 74.25};   double d;
            RandomAccessFile raf;

            try
            {
raf = new RandomAccessFile("D:\\random.dat",
"rw");
            }
            catch(FileNotFoundException exc)
            {
                System.out.println("Cannot open file.");
                return ;
            }

            // Write values to the file.
            for(int i=0; i < data.length; i++)
            {                   try
                {
                        raf.writeDouble(data[i]);
```

```
            }
            catch(IOException exc)
            {
                System.out.println("Error writing to file.");
                return ;
            }
        }

        try
        {
            // Now, read back specific values
        raf.seek(0); // seek to first double d =
raf.readDouble();
            System.out.println("First value is " + d);
        raf.seek(8); // seek to second double
        d = raf.readDouble();
            System.out.println("Second value is " + d);
        raf.seek(8 * 3); // seek to fourth double
        d = raf.readDouble();
            System.out.println("Fourth value is " + d);
            System.out.println();

            // Now, read every other value.
            System.out.println("Here is every other value: ");
        for(int i=0; i < data.length; i+=2) {
        raf.seek(8 * i); // seek to ith double
        d = raf.readDouble();
        System.out.print(d + " ");
            }
            System.out.println("\n");
        }
        catch(IOException exc)
        {
```

```
                System.out.println("Error seeking or reading.");
            }

            raf.close();
        }
}
```

Kết quả thực thi chương trình:



HÌnh 5.6 ghi 6 số kiểu double xuống file, rồi đọc lên theo thứ tự ngẫu nhiên

5.5.Sử dụng luồng ký tự

Chúng ta đã tìm hiểu và sử dụng luồng byte để xuất/nhập dữ liệu. Tuy có thể nhưng trong một số trường hợp luồng byte không phải là cách "lý tưởng" để quản lý xuất nhập dữ liệu kiểu character, vì vậy java đã đưa ra kiểu luồng character phục vụ cho việc xuất nhập dữ liệu kiểu character trên luồng.

Mức trên cùng là hai lớp trừu tượng Reader và Writer. Lớp Reader dùng cho việc nhập dữ liệu của luồng, lớp Writer dùng cho việc xuất dữ liệu của luồng. Những lớp dẫn xuất từReader và Writer thao tác trên các luồng ký tự Unicode.

Những phương thức định nghĩa trong lớp trừu tượng Reader và Writer

Phương thức	Ý nghĩa

Reader	
abstract void close( )	Đóng luồng
void mark(int numChars)	Đánh dấu vị trí hiện tại trên luồng
boolean markSupported( )	Kiểm tra xem luồng có hỗ trợ thao tác đánh dấu

	mark() không?
int read( )	Đọc một ký tự
int read(char buffer[ ])	Đọc buffer.length ký tự cho vào buffer
abstract int read(char buffer[ ], int offset, int numChars)	Đọc numChars ký tự cho vào vùng đệm buffer tại vị trí buffer[offset]
boolean ready( )	Kiểm tra xem luồng có đọc được không?
void reset( )	Dời con trỏ nhập đến vị trí đánh dấu trước đó
long skip(long numChars)	Bỏ qua numChars của luồng nhập
Writer	
abstract void close( )	Đóng luồng xuất. Có lỗi ném ra IOException
abstract void flush( )	Dọn dẹp luồng (buffer xuất)
void write(int ch)	Ghi một ký tự
void write(byte buffer[ ])	Ghi một mảng các ký tự
abstract void write(char buffer[ ], int offset, int numChars)	Ghi một phần của mảng ký tự
void write(String	Ghi một chuỗi

str)	
void write(String str, int offset, int numChars)	Ghi một phần của một chuỗi ký tự

5.5.1.Nhập Console dùng luồng ký tự

Thường thì việc nhập dữ liệu từ Console dùng luồng ký tự thì thuận lợi hơn dùng luồng byte. Lớp tốt nhất đểđọc dữ liệu nhập từ Console là lớp BufferedReader. Tuy nhiên chúng ta không thể xây dựng một lớp BufferedReader trực tiếp từSystem.in. Thay vào đó chúng ta phải chuyển nó thành một luồng ký tự. Để làm điều này chúng ta dùng InputStreamReader chuyển bytes thành ký tự.

Để có được một đối tượng InputStreamReader gắn với System.in ta dùng constructor của InputStreamReader. InputStreamReader(InputStream inputStream)

Tiếp theo dùng đối tượng InputStreamReaderđã tạo ra để tạo ra một BufferedReader dùng constructor BufferedReader. BufferedReader(Reader inputReader)

Ví dụ: Tạo một BufferedReader gắn với Keyboard

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

Sau khi thực hiện câu lệnh trên, br là một luồng ký tự gắn với Console thông qua System.in.

Ví dụ: Dùng BufferedReader đọc từng ký tự từ Console. Việc đọc kết thúc khi gặp dấu chấm (dấu chấm để kết thúc chương trình).

```
import java.io.*; class ReadChars
{
        public static void main(String args[]) throws IOException
        {
           char c;
           BufferedReader br = newBufferedReader(
                        new InputStreamReader(System.in));

           System.out.println("Nhap chuoi ky tu,
                                gioi han dau cham.");
```

```
        // read characters
        do
        {
                c = (char) br.read();
    System.out.println(c);
        } while(c != '.');
    }
}
```

Kết quả thực thi chương trình:

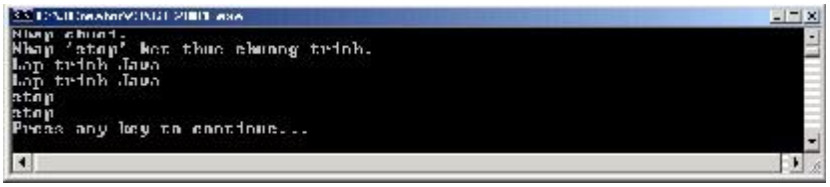
Hình 5.7 Nhập Console dùng luồng ký tự

Ví dụ: Dùng BufferedReader đọc chuỗi ký tự từ Console. Chương trình kết thúc khi gặp chuỗi đọc là chuỗi “stop”

```
import java.io.*; class ReadLines
{
    public static void main(String args[]) throws IOException
    {
        // create a BufferedReader using System.in
        BufferedReader br = new BufferedReader(new
                            InputStreamReader(System.in));
        String str;
        System.out.println("Nhap chuoi.");
System.out.println("Nhap 'stop' ket thuc chuong trinh.");
        do
        {
                str = br.readLine();
    System.out.println(str);
        } while(!str.equals("stop"));
```

```
        }
}
```

Kết quả thực thi chương trình:



Hình 5.8 Dùng BufferedReader đọc chuỗi ký tự từ Console

5.5.2.Xuất Console dùng luồng ký tự

Trong ngôn ngữ java, bên cạnh việc dùng System.outđể xuất dữ liệu ra Console (thường dùng để debug chương trình), chúng ta có thể dùng luồng PrintWriterđối với các chương trình "chuyên nghiệp". PrintWriter là một trong những lớp luồng ký tự. Việc dùng các lớp luồng ký tựđể xuất dữ liệu ra Console thường được "ưa chuộng" hơn.

Để xuất dữ liệu ra Console dùng PrintWriter cần thiết phải chỉđịnh System.out cho luồng xuất.

Ví dụ: Tạo đối tượng PrintWriter để xuất dữ liệu ra Console

PrintWriter pw = new PrintWriter(System.out, true);

Ví dụ: minh họa dùng PrintWriterđể xuất dữ liệu ra Console import java.io.*; *public class PrintWriterDemo*

```
{
        public static void main(String args[])
        {
 PrintWriter pw = new PrintWriter(System.out, true);   int i = 10;
            double d = 123.67;
            double r = i+d

            pw.println("Using a PrintWriter.");
            pw.println(i);              pw.println(d);
            pw.println(i + " + " + d + " = " + r);
        }
}
```

Kết quả thực thi chương trình:

Hình 5.9 Xuất Console dùng luồng ký tự

5.5.3.Đọc/ghi File dùng luồng ký tự

Thông thường đểđọc/ghi file người ta thường dùng luồng byte, nhưng đối với luồng ký tự chúng ta cũng có thể thực hiện được. Ưu điểm của việc dùng luồng ký tự là chúng thao tác trực tiếp trên các ký tự Unicode. Vì vậy luồng ký tự là chọn lựa tốt nhất khi cần lưu những văn bản Unicode.

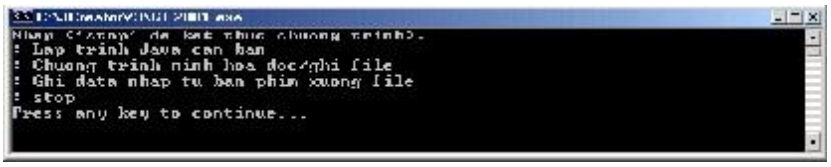
Hai lớp luồng thường dùng cho việc đọc/ghi dữ liệu ký tự xuống file là FileReader và FileWriter.

Ví dụ: Đọc những dòng văn bản nhập từ bàn phím và ghi chúng xuống file tên là “test.txt”. Việc đọc và ghi kết thúc khi người dùng nhập vào chuỗi “stop”.

```
import java.io.*; class KtoD
{
    public static void main(String args[]) throws IOException
    {
        String str;
        FileWriter fw;
        BufferedReader br = new BufferedReader(
                        new InputStreamReader(System.in));

        try
        {
                fw = new FileWriter("D:\\test.txt");
        }
        catch(IOException exc)
        {
                System.out.println("Khong the mo file.");
                return ;
        }
```

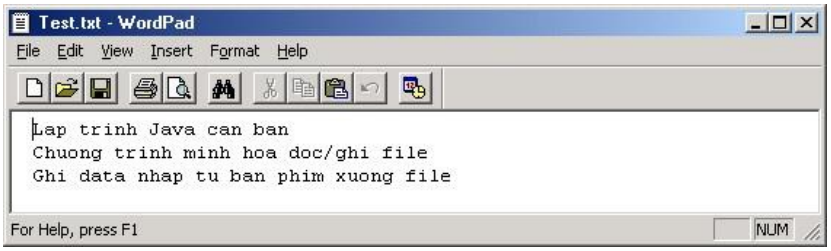
```
System.out.println("Nhap ('stop' de ket thuc chuong trinh).");

        do
        {
            System.out.print(": ");  str = br.readLine();  if(str.compareTo("stop") == 0) break;
            str = str + "\r\n";  fw.write(str);
        } while(str.compareTo("stop") != 0);

        fw.close();
    }
}
```

Kết quả thực thi chương trình Dữ liệu nhập từ Console:



Hình 5.11 Đọc File dùng luồng ký tự

Dữ liệu ghi xuống file:



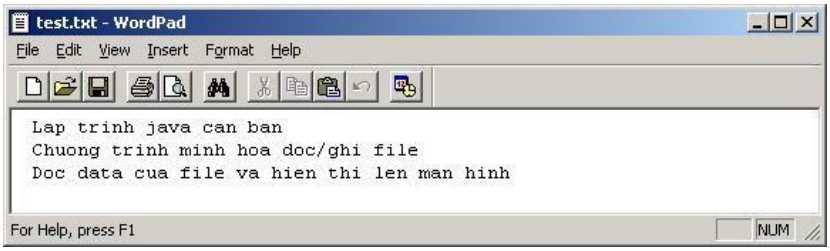
Hình 5.12 Ghi File dùng luồng ký tự

Ví dụ: đọc và hiển thị nội dung của file “test.txt” lên màn hình.

```
import java.io.*; class DtoS
{
    public static void main(String args[]) throws Exception
    {
        FileReader fr = new FileReader("D:\\test.txt");
```

```
            BufferedReader br = new BufferedReader(fr);
            String s;
            while((s = br.readLine()) != null)
            {
                  System.out.println(s);
            }

            fr.close();
        }
}
```

Kết quả thực thi chương trình Nội dung của file test.txt:



Hình 5.13 đọc và hiển thị nội dung của file "test.txt" lên màn hình

Kết quả đọc file và hiển thị ra Console:

Hình 5.14 đọc file và hiển thị ra Console

5.6.Lớp File

Lớp File không phục vụ cho việc nhập/xuất dữ liệu trên luồng. Lớp File thường được dùng để biết được các thông tin chi tiết về tập tin cũng như thư mục (tên, ngày giờ tạo, kích thước, …)

java.lang.Object +--java.io.File

Các Constructor:

Tạo đối tượng File từđường dẫn tuyệt đối public File(String pathname) ví dụ: File f = new File("C:\\Java\\vd1.java");

Tạo đối tượng File từ tên đường dẫn và tên tập tin tách biệt public File(String parent, String child) ví dụ: File f = new File("C:\\Java", "vd1.java");

Tạo đối tượng File từ một đối tượng File khác *public* ***File****(File parent, String child)*

ví dụ: *File dir = new File ("C:\\Java");*

File f = new File(dir, "vd1.java");

Một số phương thức thường gặp của lớp File (chi tiết về các phương thức đọc thêm trong tài liệu J2SE API Specification)

public StringgetName()	Lấy tên của đối tượng File
public StringgetPath()	Lấy đường dẫn của tập tin
public boolean isDirectory()	Kiểm tra xem tập tin có phải là thư mục không?
public boolean isFile()	Kiểm tra xem tập tn có phải là một file không?
…	
public String[] list()	Lấy danh sách tên các tập tin và thư mục con của đối tượng File đang xét và trả về trong một mảng.

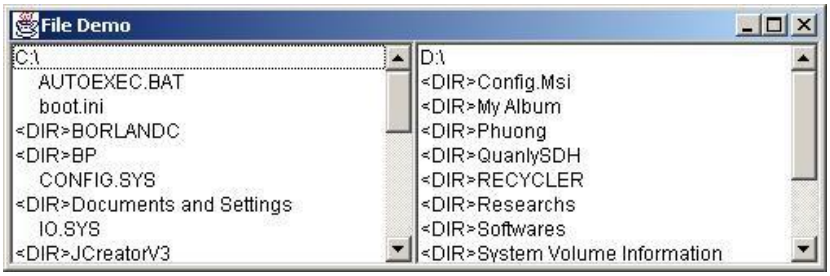
Ví dụ:

```
import java.awt.*; import java.io.*;
public class FileDemo
{
    public static void main(String args[])
    {
  Frame fr = new Frame ("File Demo");   fr.setBounds(10, 10, 300, 200);
```

```
        fr.setLayout(new BorderLayout());

        Panel p = new Panel(new GridLayout(1,2));

        List list_C = new List();
        list_C.add("C:\\");
        File driver_C = new File ("C:\\");
        String[] dirs_C = driver_C.list();

        for (int i=0;i<dirs_C.length;i++)
    {
                File f = new File ("C:\\" + dirs_C[i]);
                if (f.isDirectory())
                        list_C.add("<DIR>" + dirs_C[i]);
                else
                        list_C.add("     " + dirs_C[i]);
        }

        List list_D = new List();
list_D.add("D:\\");
File driver_D = new File ("D:\\");
String[] dirs_D = driver_D.list();
for (int i=0;i<dirs_D.length;i++)
{
                File f = new File ("D:\\" + dirs_D[i]);
                if (f.isDirectory())
                list_D.add("<DIR>" + dirs_D[i]);
                else
                        list_D.add("     " + dirs_D[i]);
        }

        p.add(list_C);
```

```
        p.add(list_D);
        fr.add(p, BorderLayout.CENTER);
        fr.setVisible(true);
    }
    }
```

Kết quả thực thi chương trình:



Hình 5.15 *FileDemo*

Tài liệu tham khảo

[1] java.sun.com

[2] Herbert Schildt. **Java 2. A Beginner's Guide**. Second Edition. McGraw-Hill - 2003.

[3] Dr. Harvey M. Deitel - Paul J. Deitel. **Java How to Program**, 4th Ed (Deitel). Prentice Hall - 2002

[4] Simon Roberts – Philip Heller – Michael Ernest. **Complete Java 2 Certification – study guide**. BPB Publications – 2000.

[5] Cay S. Horstmann – Gary Cornell. **Core Java Volum 1 - Fundamentals.** The Sun Microsystems press. 1997

[6] Cay S. Horstmann – Gary Cornell. **Core Java Volum 2 – Advanced Features.** The Sun Microsystems press. 1997