

LỜI NÓI ĐẦU

Ngày nay, hầu như việc viết một ứng dụng để chạy trên máy đơn cục bộ không còn được ưa chuộng và thích hợp nữa. Các chương trình và ứng dụng hiện đại phải tích hợp và triệu gọi lẫn nhau trên mạng Intranet (mạng cục bộ), mạng Internet (mạng toàn cầu) và ngôn ngữ lập trình Java là một trong những lựa chọn tốt nhất để làm việc này. Java là một ngôn ngữ lập trình không đơn giản, ngoài sự nổi tiếng về bản thân ngôn ngữ, nền tảng Java còn hướng đến các ứng dụng mạng như: giao tiếp trên mạng theo mô hình khách/chủ (client/server) ... So với lập trình thông thường, lập trình mạng đòi hỏi người lập trình những hiểu biết và kỹ năng chuyên sâu hơn để tạo giao tiếp và trao đổi dữ liệu giữa các máy tính với nhau.

Để giúp sinh viên chuyên ngành CNTT trong Nhà trường có thể tiếp cận được với những kỹ thuật mới này, chúng tôi đã mạnh dạn soạn thảo cuốn **“Bài giảng Lập trình mạng”** để đưa vào giảng dạy cho sinh viên CNTT học năm thứ 3 trong Nhà trường. Cuốn bài giảng này được soạn thảo dựa trên nền tảng các sinh viên CNTT sau 2 năm học đầu trong trường đã được trang bị đầy đủ các kiến thức về Ngôn ngữ lập trình hướng đối tượng, Mạng máy tính, Thiết kế Web. Đây là một môn học với đặc thù là kiến thức luôn đổi mới và cập nhật, do đó yêu cầu với môn học này là sinh viên phải tự đọc thêm tài liệu, giáo viên chỉ là người hướng dẫn những vấn đề cơ bản nhất cho sinh viên.

Lập trình mạng là môn học mới được đưa vào giảng dạy, nên trong quá trình soạn thảo bài giảng không tránh khỏi bờ ngõ và thiếu sót. Chúng tôi rất mong được sự giúp đỡ, đóng góp ý kiến của các đồng nghiệp và độc giả quan tâm để lần tái bản sau cuốn sách được hoàn thiện hơn. Mọi ý kiến đóng góp có thể gửi về theo địa chỉ email: qtmcn1@yahoo.com.

Chúng tôi xin chân thành cảm ơn!

Hà Nội 12/2005

Các tác giả

MỤC LỤC

LỜI NÓI ĐẦU	1
MỤC LỤC.....	2
PHẦN 1: TỔNG QUAN VỀ LẬP TRÌNH MẠNG	5
I. <i>Các giao thức mạng.....</i>	5
I.1. Họ giao thức TCP/IP	5
I.2. Giao thức TCP và UDP	6
I.3. Dịch vụ từ phía máy chủ và khái niệm cổng (PORT)	7
II. <i>Giao tiếp trên mạng theo mô hình khách/chủ (Client/Server) và khái niệm socket</i>	8
II.1. Giao tiếp theo mô hình khách/chủ (Client/Server).....	8
II.2. Lập trình mạng thông qua Socket	8
II.3. Tìm hiểu một số lớp cần thiết của gói thư viện Java.net	8
PHẦN 2: NGÔN NGỮ LẬP TRÌNH JAVA	15
CHƯƠNG 1: TỔNG QUAN VỀ JAVA	15
I. <i>Lịch sử Java.....</i>	15
II. <i>Java là gì?</i>	16
III. <i>Cấu trúc của Java.....</i>	16
IV. <i>Các đặc tính chính của Java.....</i>	18
IV.1. An ninh	18
IV.2. Giao diện lập trình ứng dụng chuẩn - Core API	19
IV.3. Tương thích với nhiều kiểu phần cứng	19
IV.4. Đặc tính động và phân tán	19
IV.5. Hướng đối tượng.....	19
IV.6. Đa luồng (multi-threads).....	20
IV.7. Quản lý bộ nhớ và quá trình thu dọn 'rác'	20
CHƯƠNG 2: CÁC THÀNH PHẦN CƠ BẢN CỦA NGÔN NGỮ LẬP TRÌNH JAVA	22
I. <i>Ghi chú (Comment)</i>	22
II. <i>Câu lệnh và khối lệnh</i>	23
II.1. Câu lệnh	23
II.2. Khối lệnh.....	24
III. <i>Tập ký tự dùng trong JAVA.....</i>	24
IV. <i>Từ khóa và tên</i>	25
IV.1. Tên.....	25
IV.2. Từ khóa.....	25
V. <i>Kiểu dữ liệu.....</i>	25
V.1. Kiểu dữ liệu cơ bản	26
V.2. Kiểu dữ liệu dẫn xuất (Reference)	26
V.3. Giá trị mặc định	26
VI. <i>Hằng (literal)</i>	27
VII. <i>Biến.....</i>	27
VII.1. Kiểu biến.....	28
VII.1.1. Biến đối tượng.....	28
VII.1.2. Biến lớp.....	28
VII.1.3. Biến cục bộ.....	28
VII.1.4. Phạm vi của biến	29
VIII. <i>Chuyển đổi kiểu dữ liệu</i>	29
IX. <i>Biểu thức và Toán tử.....</i>	30
IX.1. Các toán tử số học.....	30
IX.2. Các phép toán tăng giảm.....	2
IX.3. Toán tử quan hệ và điều kiện	2
IX.4. Toán tử luận lý.....	2
IX.5. Các toán tử làm việc với bit.....	2
IX.6. Toán tử gán.....	33
IX.7. Một số toán tử khác	33
IX.8. Phép toán trên kiểu chuỗi (String)	34
IX.9. Độ ưu tiên các toán tử.....	34
IX.10. Biểu thức.....	35
X. <i>Các câu lệnh điều khiển.....</i>	35

X.1.	Cấu trúc rẽ nhánh.....	35
X.1.1.	Cấu trúc điều kiện rẽ nhánh if.....	35
X.1.2.	Cấu trúc điều kiện rẽ nhánh phức : switch.....	36
X.2.	Cấu trúc lặp.....	38
X.2.1.	Vòng lặp for.....	38
X.2.2.	Vòng lặp while và do:.....	39
X.3.	Ngoại lệ và câu lệnh nắm bắt ngoại lệ.....	41
CHƯƠNG 3:	APPLETS.....	43
I.	Đại cương về HTML.....	43
II.	Tổng quan về applet.....	43
II.1.	Ví dụ về Applet.....	43
II.2.	Vòng đời của một Applet.....	44
II.2.1.	Nạp một Applet.....	44
II.2.2.	Rời khỏi và quay trở về trang web chứa applet.....	45
II.2.3.	Nạp lại Applet (Reloading the Applet).....	45
II.2.4.	Thoát khỏi trình duyệt.....	45
II.2.5.	Tóm tắt.....	45
II.3.	Các phương thức cơ bản.....	46
II.3.1.	init().....	46
II.3.2.	start().....	46
II.3.3.	stop().....	46
II.3.4.	destroy().....	46
II.4.	Các phương thức vẽ và nắm bắt sự kiện.....	47
II.5.	Các phương thức cho lập trình giao diện người dùng.....	47
II.5.1.	Các thành phần UI xây dựng sẵn.....	47
II.5.2.	Các phương thức để sử dụng các thành phần UI trong các Applet.....	48
II.5.3.	Thêm một Text Field không edit được vào applet Simple.....	48
II.6.	Giới hạn của Applet.....	49
II.6.1.	Giới hạn về bảo mật.....	49
II.6.2.	Các khả năng của Applet.....	50
II.7.	Test một applet.....	50
III.	Các tính năng cao cấp của Applet API.....	51
III.1.	Tìm kiếm và nạp các file dữ liệu.....	52
III.2.	Hiển thị chuỗi tình trạng ngắn.....	52
III.3.	Hiển thị tài liệu trong trình duyệt.....	53
III.4.	Gửi thông điệp tới các applet khác.....	54
III.5.	Tìm một applet bằng tên: sử dụng phương thức getApplet.....	55
III.6.	Tìm tất cả các applet trên một trang: sử dụng phương thức getApplets.....	59
III.7.	Đan xen vào các trang Web.....	60
III.7.1.	Các thuộc tính (Attributes).....	61
III.7.2.	Các thông số của applet.....	61
CHƯƠNG 4:	CÁC GÓI & GIAO DIỆN.....	63
I.	Giới thiệu.....	63
II.	Các giao diện.....	63
II.1.	Các bước để tạo một giao diện.....	63
II.2.	Hiện thực giao diện.....	64
III.	Các gói.....	66
III.1.	Tạo một gói.....	68
III.2.	Thiết lập đường dẫn cho lớp (classpath).....	70
IV.	Gói và điều khiển truy xuất.....	72
IV.1.	Gói java.lang.....	73
IV.1.1.	Lớp String (lớp chuỗi).....	74
IV.1.2.	Chuỗi mặc định (String pool).....	75
IV.1.3.	Các phương thức của lớp String.....	76
IV.1.4.	Lớp StringBuffer.....	78
IV.1.5.	Các phương thức lớp StringBuffer.....	80
IV.1.5.	Lớp java.lang.Math.....	82
IV.1.6.	Lớp Runtime (Thời gian thực hiện chương trình).....	84
IV.1.7.	Lớp System.....	85
IV.1.8.	Lớp Class.....	87
IV.1.9.	Lớp Object.....	88
IV.2.	Gói java.util.....	89
IV.2.1.	Lớp Hashtable (bảng băm).....	90
IV.2.2.	Lớp random.....	93
IV.2.3.	Lớp Vector.....	94
IV.2.4.	Lớp StringTokenizer.....	97

PHẦN 3: LẬP TRÌNH SOCKET	102
CHƯƠNG 1: LẬP TRÌNH TCP SOCKET	102
I. Xây dựng chương trình EchoServer	102
II. Xây dựng chương trình EchoClient	103
CHƯƠNG 2: LẬP TRÌNH UDP SOCKET.....	105
I. Xây dựng chương trình ExchangeRateServer	105
II. Xây dựng chương trình ExchangeRateTable.....	106
PHẦN 4: LẬP TRÌNH TRÊN INTERNET	109
CHƯƠNG 1: JSP VÀ CÁC KHÁI NIỆM MỞ ĐẦU	110
I. Các cơ chế hoạt động của trang JSP	110
II. Xây dựng trang JSP	110
CHƯƠNG 2: CÁC CỤ PHÁP CƠ BẢN CỦA JSP	112
I. Các đối tượng mặc định của JSP	112
II. Các thẻ lệnh JSP.....	112
II.1. Thẻ bọc mã <% %>.....	112
II.2. Thẻ hiển thị kết xuất <%= %>	113
II.3. Thẻ chỉ dẫn biên dịch trang <%@ page %>.....	114
II.4. Chèn chú thích vào mã trang JSP.....	114
II.5. Khai báo phương thức và biến hằng <%! %>	116
III. Truy xuất cơ sở dữ liệu trong trang JSP.....	116
TÀI LIỆU THAM KHẢO	119

Phần 1: Tổng quan về lập trình mạng

I. Các giao thức mạng

I.1. Họ giao thức TCP/IP

Để hai hay nhiều máy có thể giao tiếp được với nhau, chúng phải dùng một ngôn ngữ chung: chẳng hạn máy này phải gửi những tín hiệu gì đến máy kia và máy kia phải gửi trả lại những tín hiệu nào để nhận biết. Trên Internet ngày nay việc hai máy có thể trao đổi được với nhau đa số đều dựa theo quy ước hay giao thức cốt lõi là TCP/IP (Transmission Control Protocol/ Internet Protocol). Theo giao thức này, mỗi máy sẽ được đặt cho một số riêng biệt gọi là địa chỉ IP (IP address) có vai trò tương tự số điện thoại, chẳng hạn máy tính có tên là www.microsoft.com sẽ có địa chỉ IP là 207.46.230.219. Các số IP này là duy nhất và không máy nào được trùng nhau (trên toàn thế giới). Khi bạn muốn máy của mình có địa chỉ IP để tham gia vào hệ thống Internet toàn cầu như là một máy chủ (host hay server) ta phải đăng ký với tổ chức quốc tế InterNIC (Internet Network Information Center) để nhận được một số IP riêng biệt.

Vậy tại sao vừa có địa chỉ IP lại vừa có tên riêng cho từng máy? Dùng cái nào để xác định liên lạc với một máy chủ (như ở trên, ta nên dùng www.microsoft.com hay số 207.46.230.219 để kết nối với một máy chủ của công ty Microsoft). Thật ra tên và địa chỉ IP là một, nhưng địa chỉ IP được ghi bằng số, còn tên của máy chủ lại được ghi bằng chữ có ý nghĩa và gần gũi hơn với con người. Với mỗi hệ thống đều có sự chuyển đổi trực tiếp từ tên vùng thành địa chỉ IP thích hợp trước khi dữ liệu được gửi đi.

Ví dụ đối với Windows có thể tham khảo hai tập tin HOSTS và LMHOSTS đây là hai tập tin văn bản (được coi như một cơ sở dữ liệu) để lưu trữ tập hợp các số IP cùng với tên tương ứng. Ta có thể tự thêm vào địa chỉ IP và tên máy chủ cách nhau bằng khoảng trắng. Khi có nhu cầu truy cập đến một máy ở xa qua giao thức mạng TCP/IP nếu gõ vào tên máy chủ thì hệ thống sẽ tự tìm địa chỉ IP tương ứng trong tập tin này:

Tập tin HOSTS

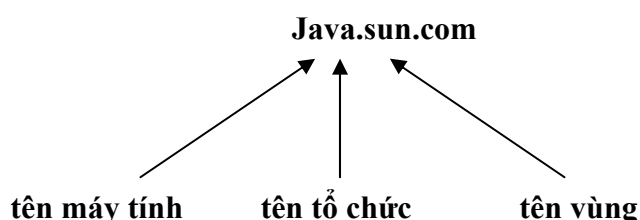
216.32.74.52 www.myyahoo.com

164.71.2.70 www.fujitsu.com

Tên của máy chủ còn được gọi là tên vùng (domain name) bởi vì chúng được đặt theo thứ tự phân cấp của tên lãnh thổ, vùng, tổ chức, hay tên cá nhân... mỗi nhóm phân cấp cách nhau bởi một dấu chấm (.). Công việc theo dõi sự thay đổi tên được phân phối qua Internet nhờ các máy chủ lớn DNS (Domain Name System) theo dõi các máy chủ khác trong vùng con của chúng.

Trước đây mỗi máy có thể tự mình lưu trữ một tập tin chứa phần lớn các tên và địa chỉ của những máy chủ thông dụng (như trong Windows là tập tin HOSTS và LMHOSTS), nhưng ngày nay ta không làm như vậy nữa mà đa số các tên vùng cũng như địa chỉ IP được lấy xuống từ các máy chủ DNS.

Khi đọc tên vùng của một máy chủ ta đi từ trái qua phải. Ví dụ:



Nói chung là theo quy ước từ phần riêng biệt nhất đến phần chung nhất. Tuy nhiên không bắt buộc là như vậy, ta vẫn có thể đặt tên theo cách khác, không nhất thiết là chỉ gồm 3 hay 4 nhóm (ví dụ: here.is.along.name.address.co.vn là hợp lệ) bởi vì cuối cùng thì tên vùng cũng được hệ thống DNS chuyển thành địa chỉ IP mà thôi.

I.2. Giao thức TCP và UDP

Quá trình chuyển dữ liệu trên mạng là khá phức tạp. Chi tiết quá trình này diễn ra tương tự như trong thực tế ta gửi thư hay bưu phẩm, trước hết phải ghi rõ địa chỉ nơi đến (trường hợp này là địa chỉ IP của máy chủ), sau đó có thể gửi thông thường hoặc gửi bảo đảm (tùy theo cách gửi mà thư hay bưu phẩm có chắc chắn đến được tay người nhận hay không), người nhận sau khi nhận được có thể hồi âm trả lời là đã nhận đủ hoặc bị mất mát gì đó trong quá trình chuyển tải. Người gửi có thể gửi tiếp những phần bị mất (hoặc không cần gửi nữa)

Cách chuyển dữ liệu bảo đảm dựa vào giao thức TCP (Transmission Control Protocol), còn cách truyền không đảm bảo dựa vào giao thức UDP (User Datagram Protocol).

Giao thức TCP gửi từng gói dữ liệu đi, nơi nhận theo giao thức này phải có trách nhiệm thông báo và kiểm tra xem dữ liệu đã đến đủ hay chưa, có lỗi hay không có lỗi. Trước khi chuyển dữ liệu bao giờ cũng có sự kết nối giữa máy gửi và máy nhận. Do phải đảm bảo dữ liệu được truyền chính xác và luôn duy trì kết nối nên sử dụng giao thức TCP cần chiếm thêm một số tài nguyên của hệ thống và cách lập trình cho giao thức này hơi khó (phải thực hiện các bước kiểm tra dữ liệu theo yêu cầu của TCP). Truyền dữ liệu theo TC thường áp dụng cho các dịch vụ như truyền tập tin, các dịch vụ trực tuyến trên Internet đòi hỏi độ tin cậy cao ...

Giao thức UDP ngược lại không đáng tin cậy lắm, không có sự kết nối trước nào giữa nơi gửi và nơi nhận, dữ liệu gửi đi mặc định rằng máy tính ở đầu nhận luôn ở trạng thái sẵn sàng để tiếp

đón dữ liệu gửi đến. Nếu dữ liệu gửi đến bị lỗi trong quá trình truyền hay không nhận được đầy đủ giao thức UDP cũng không có thông tin phản hồi gì lại cho nơi gửi. Tuy nhiên UDP không đòi hỏi sự chính xác cao như dịch vụ thông báo giờ, tỉ giá hay các dịch vụ gửi nhắn tin ...

1.3. Dịch vụ từ phía máy chủ và khái niệm cổng (PORT)

Ta có thể kết nối vào Internet thông qua dịch vụ của nhà cung cấp còn gọi là ISP (như FPT, Cnet ...) bằng đường điện thoại thông qua modem. Các nhà dịch vụ này đóng vai trò như những máy chủ (server) giúp dễ dàng truy cập dữ liệu từ những vùng khác nhau trên mạng.

Khi kết nối vào máy chủ ta có thể yêu cầu máy chủ nhiều dịch vụ khác nhau, như dịch vụ truy tìm và đọc các trang Web trên Internet, dịch vụ gửi nhận e-mail, dịch vụ dò tìm hệ thống tên vùng DNS ... Mỗi dịch vụ đều có cách gửi nhận dữ liệu theo quy ước riêng. TCP và UDP chỉ chịu trách nhiệm đưa dữ liệu từ một máy tính này đến một máy tính khác, còn dữ liệu đó được gửi cho dịch vụ nào thì phải thông qua một dịch vụ nữa gọi là cổng (hay port). Mỗi chương trình dịch vụ sẽ sử dụng một cổng khác để truy xuất thông tin. Cổng là một số nguyên dương có giá trị từ 1 đến 16383

Máy chủ (server) sẽ quy định cổng được sử dụng cho mỗi loại dịch vụ. Thông tin giữa máy khách (client) và máy chủ (server) phải sử dụng cổng tương ứng nhau thì mới trao đổi được với nhau. Tuy nhiên, hầu hết các chương trình dịch vụ nổi tiếng hiện nay đều có quy định chuẩn cổng dành riêng cho mình như:

Dịch vụ cổng (port)

FTP21

HTTP80

Telnet23

Finger79

SMTP25

Nếu tự xây dựng một ứng dụng làm dịch vụ trên máy chủ ta phải chọn cho mình một số cổng có các giá trị khác với những giá trị cổng mà những dịch vụ nổi tiếng khác đã sử dụng.

II. Giao tiếp trên mạng theo mô hình khách/chủ (Client/Server) và khái niệm socket.

II.1. Giao tiếp theo mô hình khách/chủ (Client/Server)

Có rất nhiều dịch vụ hỗ trợ trên Internet như e-mail, nhóm tin (newsgroup), chuyển tập tin (file transfer), đăng nhập từ xa (remote login), truy tìm các trang Web ... Những dịch vụ này được tổ chức và kiến trúc theo mô hình khách/chủ (mô hình Client/Server). Các chương trình ở máy khách (client) như trình duyệt (Web Browser) hay chương trình gửi nhận e-mail sẽ tạo ra kết nối (connection) với một máy chủ ở xa (server) sau đó gửi các yêu cầu đến máy chủ, các chương trình dịch vụ trên máy chủ như Web server hay Mail server ... sẽ xử lý những yêu cầu này và gửi kết quả ngược về cho máy khách (chẳng hạn Web theo địa chỉ mà máy khách đưa đến còn Mail server thì lưu giữ và gửi về cho máy khách những bức e-mail mới). Thông thường một dịch vụ trên máy chủ phục vụ cho rất nhiều khách.

II.2. Lập trình mạng thông qua Socket

Như vậy trước khi yêu cầu một dịch vụ trên máy chủ thực hiện điều gì đó, máy khách (client) phải có khả năng kết nối được với máy chủ. Quá trình kết nối này được Java thực hiện thông qua một cơ chế trừu tượng hóa gọi là Socket (tạm gọi là “cơ chế ổ cắm”). Kết nối giữa máy khách và máy chủ tương tự như việc cắm phích điện vào ổ cắm điện. Máy khách thường được coi như phích cắm điện còn máy chủ được coi như ổ cắm điện, một ổ cắm có thể cắm vào đó nhiều phích điện khác nhau cũng như một máy chủ có thể kết nối và phục vụ cho rất nhiều máy khách.

Nếu kết nối socket thành công thì máy khách và máy chủ có thể trao đổi dữ liệu với nhau thực hiện các yêu cầu về dịch vụ trên máy chủ. Việc kết nối theo cơ chế socket cần biết hai thông tin chủ yếu đó là địa chỉ của máy cần kết nối và số hiệu cổng của chương trình dịch vụ. Java cung cấp lớp Socket (thường được dùng như “phích cắm điện” cho máy khách) và lớp ServerSocket (thường được dùng như “ổ cắm điện” đặt trên máy chủ). Hai lớp này được đặt trong gói thư viện Java.net.

II.3. Tìm hiểu một số lớp cần thiết của gói thư viện Java.net

a. Lớp InetAddress

Vì địa chỉ Internet theo số IP và theo tên rất thường dùng khi kết nối vào mạng cho nên Java xây dựng hẳn một lớp InetAddress dành riêng cho việc quản lý địa chỉ theo tên và theo số. Lớp InetAddress cung cấp các phương thức static thông dụng nhất dùng để chuyển đổi và truy xuất

địa chỉ IP (không có phương thức khởi dựng cho lớp này). Thường ta sẽ quan tâm đến các phương thức sau:

- `public static InetAddress getLocalHost ()`
throws `UnknownHostException`

Trả về đối tượng `InetAddress` là địa chỉ của máy cục bộ (local host)

- `public static InetAddress getByName (String host)`
throws `UnknownHostException`

Phương thức này nhận địa chỉ của một máy bằng kiểu chuỗi `String` và trả về đối tượng kiểu `InetAddress` thay mặt cho địa chỉ máy này.

- `public static InetAddress[] getAllByName (String host)`
throws `UnknownHostException`

Phương thức này nhận địa chỉ của một máy bằng kiểu chuỗi và trả về tất cả các đối tượng `InetAddress` thay mặt cho địa chỉ máy này.

- `public byte[] getAddress()`

Trả về địa chỉ IP của đối tượng `InetAddress` dưới dạng một dãy các byte. Vị trí byte cao nhất nằm ở byte 0.

- `public String.getHostAddress()`

Trả về địa chỉ IP của đối tượng `InetAddress` dưới dạng một chuỗi được định dạng phân làm 4 nhóm `%d.%d.%d.%d` (Ví dụ: “172.16.11.12”).

Dưới đây là ví dụ cho thấy cách dùng lớp `InetAddress` để lấy về các thông tin của địa chỉ máy chủ:

Ví dụ 1-0 `AddrLookupApp.java`

```
import java.net
```

b. Lớp Socket

Lớp `Socket` dùng tạo kết nối từ phía khách với máy chủ thường được khởi dựng bằng các phương thức sau:

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

Tạo ra một socket để kết nối máy có tên theo địa chỉ host và số cổng port.

- `public Socket(InetAddress address, int port) throws IOException`

Tạo ra một socket kết nối địa chỉ là đối tượng `InetAddress` và số cổng port.

- `Public Socket(String host, int port, boolean stream)` throws `IOException`

Tạo ra một socket kết nối theo địa chỉ host và số cổng port tham số stream cuối cùng để quy định kết nối theo TCP (stream=true) hay UDP (stream=false). Tuy nhiên nếu áp dụng để tạo socket cho giao thức UDP nên sử dụng lớp thay thế là `DatagramSocket`.

Các phương thức khác hỗ trợ cho lớp `Socket` từ phía máy khách bao gồm:

- `InputStream getInputStream()` throws `IOException`

Lấy về luồng nhập để máy khách có thể nhập dữ liệu trả về từ phía máy chủ.

- `OutputStream getOutputStream()` throws `IOException`

Lấy về luồng xuất để máy khách có thể ghi dữ liệu gửi đến máy chủ.

- `InetAddress getInetAddress()`

Lấy địa chỉ kết nối socket của máy chủ

- `int getPort()`

Lấy về số cổng dùng kết nối của máy chủ.

Ví dụ đoạn mã sau sẽ thực hiện kết nối với máy chủ có địa chỉ “my.testing.server” và mở ra hai luồng xuất nhập để đọc và gửi thông tin đến máy chủ theo số cổng 1234.

```
try{
    Socket me=new Socket("my.testing.server", 1234);
    // Luồng nhập thông tin để trả về máy chủ từ phía kết nối
    DataInputStream in = new DataInputStream(me.getInputStream());
    // Luồng xuất để ghi thông tin gửi đến máy chủ
    DataOutputStream out = new DataOutputStream(me.getOutputStream());
    Catch (Exception e) {
        System.out.println(e);
    }
}
```

c. Lớp `ServerSocket`

Lớp `ServerSocket` dùng tạo kết nối từ phía máy chủ với các máy khách. Đối tượng `ServerSocket` được tạo ra trên máy chủ và lắng nghe những kết nối từ phía máy khách gửi đến theo một số cổng định trước. Đối tượng `ServerSocket` được khởi dựng từ phương thức sau:

- `public ServerSocket(int port)` throws `IOException`

port là số hiệu cổng mà đối tượng `ServerSocket` phải lắng nghe để nhận biết những kết nối từ phía máy khách gửi đến

Để chờ đợi kết nối từ các máy khách gửi đến đối tượng `ServerSocket` thường nhờ đến phương thức `accept` như sau:

- `Socket accept ()` throws `IOException`

Phương thức này thực sự dừng lại chờ đợi cho đến khi nhận được thông tin kết nối sẽ trả về đối tượng socket của máy khách nơi có yêu cầu nối vào máy chủ.

Cuối cùng máy chủ có thể cắt đứt mọi kết nối bằng cách gọi phương thức close của đối tượng ServerSocket

- `Public void close () throws IOException`

Ví dụ đoạn mã sau sẽ tạo một đối tượng ServerSocket trên máy chủ luôn lắng nghe kết nối từ phía máy khách gửi đến qua số cổng 1234

```
Try {  
    ServerSocket server = new ServerSocket (1234);  
    Socket client;  
    // Chương trình sẽ dừng lại ở đây để chờ đợi sự kết nối  
    client = server.accept ();  
    // Có một kết nối gửi đến từ phía máy khách  
    System.out.println("Accept connect");  
    // Xử lý các yêu cầu về dịch vụ  
    // ...  
    // Cắt đứt các kết nối  
    client.close ();  
    server.close ();  
    catch (Exception e) {  
        System.out.println (e);  
    }  
}
```

d. Lớp DatagramSocket

Lớp này được dùng để chuyển đi một gói dữ liệu (biểu diễn bằng đối tượng DatagramPackage) theo giao thức UDP. Dữ liệu được gửi đi không bảo đảm được nhận đầy đủ và có thể bị lỗi trên đường truyền (cơ chế dùng DatagramSocket không an toàn bằng lớp Socket). Dưới đây là một số phương thức dùng của lớp DatagramSocket:

- `public DatagramSocket () throws SocketException`
Phương thức khởi dựng để tạo kết nối UDP
- `public DatagramSocket (int port) throws SocketException`
Phương thức khởi dựng để tạo kết nối UDP với số hiệu cổng port
- `public void synchronized send (DatagramSocket p) throws IOException`
Gửi gói dữ liệu đi.
- `Public void synchronized receive (DatagramSocket p) throws IOException`

Nhận gói dữ liệu về.

- `public void synchronized close ()`

Đóng kết nối.

e. Lớp DatagramPackage

Lớp này dùng cho một gói dữ liệu gửi đi trên mạng theo kết nối DatagramSocket. Một gói có thể chứa các thông tin như dữ liệu, chiều dài gói, các địa chỉ IP và số cổng mà từ đó các gói dữ liệu được gửi đi. Dưới đây là một số phương thức hữu dụng của lớp DatagramPackage này:

- `public DatagramPackage(byte buff[], int len)`

Phương thức khởi tạo ra gói có dữ liệu chứa trong bộ đệm `buff[]` và chiều dài gói dữ liệu là `len`.

- `public DatagramPackage (byte buff[], int len, InetAddress iaddr, int port)`

Phương thức khởi tạo ra gói có dữ liệu chứa trong bộ đệm `buff[]` cùng với chiều dài vùng đệm muốn lấy, địa chỉ máy đích và số hiệu cổng.

- `public InetAddress getAddress()`

Trả về địa chỉ chứa trong gói dữ liệu.

- `public byte[] getData()`

Trả về dữ liệu thật sự chứa trong gói.

- `public int getLength()`

Trả về kích thước hay chiều dài gói dữ liệu.

- `public int getPort()`

Trả về số hiệu cổng chứa trong gói dữ liệu.

f. Lớp URL

URL(Uniform Resource Locator) là địa chỉ định vị tài nguyên trên mạng, thường một URL như đã đề cập bao gồm 3 phần: phần nghi thức(protocol), phần địa chỉ hay tên máy chủ(host name), và phần chỉ định tên tập tin hay tài liệu muốn lấy từ máy chủ về.

Java đóng gói tất cả những đặc điểm này vào một lớp URL. Đối tượng URL được tạo ra bằng một trong những phương thức khởi tạo sau:

- `public URL(String spec) throws MalformedURLException`

Tạo một đối tượng URL từ địa chỉ định vị là một chuỗi.

- `public URL(String protocol, String host, int port, String file) throws MalformedURLException`

Tạo một địa chỉ định vị tuyệt đối với đầy đủ nghi thức(protocol), máy chủ(server), cổng(port), đường dẫn(file) tới tập tin cần lấy trên máy chủ.

- *public URL(String protocol, String host, String file) throws MalformedURLException*

Tạo một địa chỉ định vị tuyệt đối với đầy đủ nghi thức(protocol), máy chủ(server), đường dẫn(file) tới tập tin cần lấy trên máy chủ(bỏ qua thành phần định vị cổng giao tiếp, nếu truy tìm trang web theo nghi thức http thì cổng chương trình web server mặc định là 80).

Các phương thức hỗ trợ khác dùng cho lớp URL là:

- *public final Object getContent() throws IOException*

lấy về nội dung mà kết nối theo địa chỉ URL có được.

- *String getFile()*

Lấy về tên tập tin hay tài liệu nằm trong chuỗi địa chỉ URL có được.

- *String getHost()*

Lấy tên máy chủ(thường là thành phần thứ 2 của chuỗi URL)

- *String getPort()*

Lấy về số hiệu cổng.

- *String getProtocol()*

Lấy về tên giao thức(thường là thành phần đầu tiên trong chuỗi URL)

- *String getRef()*

Lấy về nội dung chuỗi tham khảo thêm trong chuỗi URL(được đặt sau dấu # của chuỗi)

- *Public final InputStream openStream() throws IOException*

Mở luồng nhập để đọc thông tin trả về từ máy chủ

Ví dụ đoạn mã sau đây dùng để lấy về nội dung trang web index.htm từ máy chủ java.sun.com:

```
try{
```

```
// Mở kết nối đến trang Web theo địa chỉ định vị URL
```

```
URL o = new URL("http://java.sun.com/index.htm");
```

```
// Tạo luồng nhập để đọc nội dung trang Web trả về từ máy chủ
```

```
BufferedReader inStream = new BufferedReader(new InputStreamReader(o.openStream()));
```

```
// In nội dung trang Web index.htm ra màn hình
```

```
String line;
```

```
while((line = in.readLine()) != null) {
```

```
        System.out.println (line);  
    }  
} catch (Exception e) {  
// Quá trình mở và kết nối với trang web bị lỗi  
    System.out.println (e) ;  
}
```

Phần 2: Ngôn ngữ lập trình Java

CHƯƠNG 1:

Tổng quan về Java

I. Lịch sử Java

Chúng ta biết đến Sun Microsystems như một nhà sản xuất phần cứng với các trạm làm việc UNIX. Trên thực tế, hãng Sun cũng phát triển phần mềm, đặc biệt nổi tiếng là hệ điều hành Solaris và hệ thống tập tin mạng (Network File System - NFS). Năm 1990, Sun Microsystems bắt đầu thực hiện dự án có tên gọi Green nhằm phát triển phần mềm trong các thiết bị điện tử dân dụng. James Gosling, chuyên gia lập trình phần mềm mạng được giao trách nhiệm thực hiện dự án.

Ban đầu, Gosling sử dụng C++ để viết phần mềm điều khiển, hiển thị số cho thiết bị như VCR (Video Cassette Recorder), PDA (Personal Digital Assistant). Nhưng ngay sau đó, Gosling phát hiện ra rằng C++ không phải là ngôn ngữ thích hợp cho công việc này. Ngôn ngữ C++ đủ mềm dẻo để điều khiển hệ thống, nhưng nó lại dễ gây ra những lỗi dẫn đến treo hệ thống. Một cách chi tiết hơn, C++ xâm nhập trực tiếp đến tài nguyên hệ thống, yêu cầu người lập trình phải tự mình quản lý các tài nguyên này. Điều này tạo thành một rào cản không cho C++ trở thành một công cụ viết các phần mềm có độ tin cậy cao, tính tương thích lớn, đặc biệt trong việc điều khiển các thiết bị điện tử dân dụng.

Gosling giải quyết vấn đề này bằng cách tạo ra một ngôn ngữ lập trình mới có tên là Oak. Ngôn ngữ này có cú pháp giống như C++, nhưng bỏ qua các tính năng "nguy hiểm" của C++ như truy cập trực tiếp tài nguyên hệ thống, các phép toán với con trỏ, nạp chồng tác tử. Oak được thiết lập với mục đích tạo tính tương thích cao (chạy trên nhiều loại chip khác nhau), giúp các nhà sản xuất thiết bị có thể thay đổi kiểu phần cứng mà không phải viết lại phần mềm trước đó.

Khi ngôn ngữ Oak trưởng thành, World Wide Web cũng đang bước vào thời kỳ phát triển mạnh mẽ, và đội ngũ phát triển phần mềm của Sun thấy rằng đây cũng là ngôn ngữ đặc biệt thích hợp cho Internet. Vào năm 1994, họ đã đưa ra WebRunner, một trình duyệt Web viết bằng Oak (sau này trình duyệt này được đổi tên thành HotJava và hiện nay vẫn đang được tiếp tục phát triển).

Cuối cùng, vào năm 1995, Oak được đổi tên thành Java (do mục đích thương mại) và đưa ra trình diễn tại SunWorld 95. Từ đó đến nay, Java nhanh chóng phát triển. Thậm chí trước khi

trình dịch Java đầu tiên được đưa ra vào tháng 1 năm 1996, Java đã được coi là một chuẩn công nghiệp cho Internet.

Trong 6 tháng đầu năm 1996, nhiều nhà sản xuất phần mềm cũng như phần cứng đứng đầu thế giới đã mua bản quyền công nghệ Java từ Sun, bao gồm Adobe, Asymetrix, Borland, IBM, Macromedia, Metrowerks, Microsoft, Novel, Oracle, Spyglass và Symantec... Các hãng này sẽ kết hợp Java vào các sản phẩm của họ như: các phần mềm, hệ điều hành, công cụ phát triển.

II. Java là gì?

Java là công nghệ cho phép tạo ra các phần mềm phân tán (distributed software). Đây là những phần mềm đặt trên máy chủ (server), được nạp về qua kết nối mạng và thực hiện trên máy khách (client).

Mặc dù được tạo ra từ những năm 70, Internet chỉ thực sự quyền rũ các doanh nghiệp vào những năm 90 nhờ có sự ra đời của World Wide Web. Web cho phép người sử dụng truy cập trực tiếp các thông tin trên Internet mà không cần phải học các lệnh phức tạp, cung cấp thông tin trực tuyến về nhiều lĩnh vực với hình ảnh, âm thanh,.. Sự ra đời của Java cho phép Web tiến xa hơn nữa, biến các trang Web tĩnh thành các ứng dụng sống động, có thể tương tác với người sử dụng. Những lý do căn bản để mọi người chú ý đến Java là:

- + Cho phép viết các chương trình mạnh và tin cậy.
- + Xây dựng ứng dụng chạy được trên hầu hết các phần cứng và hệ điều hành khác nhau (multi-platform).
- + Phân phối các ứng dụng trên mạng với độ bảo mật và an toàn cao.

Như vậy, có thể nói Java đã thay đổi chức năng của Internet, cũng giống như Web đã thay đổi cách tiếp cận vào Internet của chúng ta. Nói cách khác, Java đã chuyển mạng từ chỗ chỉ đơn thuần cung cấp thông tin và chia sẻ tài nguyên sang đóng vai trò hệ điều hành. Đây chính là nền móng cho phép xây dựng các máy tính mạng (Network Computer - NC) của Oracle, IBM và Sun.

III. Cấu trúc của Java

Sức mạnh Java có được chính là nhờ cấu trúc của nó. Java được thiết kế nhằm mục đích trước hết là đơn giản hoá công việc của người lập trình. Kể đến, do nhu cầu chạy trên mạng, Java phải thật sự an toàn và ổn định, cũng như có khả năng làm việc được với nhiều kiểu phần cứng, phần mềm khác nhau. Cấu trúc ngôn ngữ Java thực sự đã đảm bảo được tất cả các tính năng trên.

Cũng như các ngôn ngữ lập trình khác, Java cần một trình biên dịch để chuyển đổi mã lệnh cho người đọc (mã nguồn) sang ứng dụng thực thi được. Các trình biên dịch thông thường như Microsoft Visual C++ cho Windows 95 sẽ biên dịch chương trình sang mã lệnh thực hiện trên một loại phần cứng nhất định nào đó (trong trường hợp này là mã lệnh cho Intel x86). Trái lại, trình biên dịch Java lại chuyển chương trình nguồn Java thành các bytecode. Các bytecode này chỉ có thể chạy được trên máy ảo Java (Java Virtual Machine -JVM).

Lưu ý: Hiện nay, máy ảo Java (JVM) mới được tạo dựng bằng phần mềm chứ không phải phần cứng. Sun Microsystems và một số công ty điện tử khác đang tiến hành nghiên cứu phát triển chip picoJava, nhằm mục đích tạo máy ảo Java bằng phần cứng. Các chip này cho phép đưa Java vào các thiết bị điện tử một cách dễ dàng hơn, đồng thời làm tăng tốc độ tối đa cho các JVM viết bằng phần mềm.

Bộ Java Developers Kit (JDK) do Sun cung cấp bao gồm một số chương trình tiện ích cho phép bạn biên dịch, bắt lỗi và tạo tài liệu cho một ứng dụng Java. Hiện nay trên thị trường đang có rất nhiều môi trường phát triển Java của hãng thứ ba rất tiện lợi (như Visual J++, Symantec Cafe,...), nhưng tất cả các chương trình này đều dựa trên nền JDK. Các trình tiện ích của JDK bao gồm:

javac: Bộ biên dịch Java: Làm nhiệm vụ chuyển mã nguồn Java sang bytecode.

java: Bộ thông dịch Java: Thực thi các ứng dụng Java trực tiếp từ tập tin lớp (class).

appletviewer: Một trình thông dịch Java thực thi các Java applet từ tập tin HTML.

javadoc: Tạo tài liệu dạng HTML từ mã nguồn cùng với các chú thích bên trong.

jdb (Java debugger): Cho phép bạn thực hiện từng dòng trong chương trình, đặt các điểm dừng (breakpoint), xem giá trị các biến.

javah: Tạo ra tập tin header của C cho phép C gọi hàm Java hoặc ngược lại.

javap: Trình dịch ngược java (disassembler): Hiển thị các hàm và dữ liệu truy cập được bên trong một tập tin lớp đã dịch. Nó cũng cho phép hiển thị nghĩa của bytecode.

Quá trình biên dịch Java như sau: mã nguồn trong các tập tin *.java, qua trình biên dịch javac được chuyển thành các bytecode. Bytecode nằm trong tập tin *.class, được gọi là tập tin lớp (bởi mỗi tập tin chứa một lớp riêng biệt của Java). Các ứng dụng Java có thể bao gồm nhiều lớp khác nhau.

Chú ý: Một lớp (class) của Java cũng giống hệt như một lớp trong C++. Lớp chính là các biến dữ liệu và thủ tục kết hợp với nhau thành một khối.

Khi thực hiện chương trình Java, máy ảo Java sử dụng trình nạp lớp (class loader) để đọc các bytecode từ đĩa hoặc kết nối mạng. Các lớp được nạp sẽ phải đi qua trình kiểm tra lớp (class verifier) để chắc chắn rằng chúng sẽ không sinh ra các lỗi ảnh hưởng đến hệ thống khi thực thi. Quá trình kiểm tra này làm tăng thời gian nạp một lớp, tuy nhiên nó chỉ phải thực hiện có một lần mà thôi.

Phần thực hiện (execution unit) trong máy ảo Java sẽ thực thi các lệnh quy định trong từng bytecode. Bộ phận thực thi đơn giản nhất là một trình thông dịch, chuyển đổi từng bytecode sang các thủ tục cần làm trên từng hệ thống. Cách này rất chậm vì trình thông dịch luôn phải tra nghĩa của bytecode mà nó thực thi. Để khắc phục nhược điểm này, người ta đưa ra trình biên dịch Just-in-time (JIT): Quá trình chuyển đổi từ bytecode sang mã lệnh riêng của từng hệ thống sẽ được làm luôn một lần ngay khi nạp chương trình, do đó tăng được tốc độ đáng kể.

Chương trình viết bằng Java có thể là ứng dụng riêng biệt (stand-alone application), hay là ứng dụng kí sinh trên Web (applet) hoặc đồng thời cả hai. Applet là chương trình được nhúng trong trang Web, được đọc và thực hiện bởi trình duyệt hỗ trợ Java (Java-enabled Web browser). Khi trình duyệt đọc tới trang Web này, applet sẽ được thực thi. Trái lại, một ứng dụng Java riêng biệt được chạy bằng dòng lệnh (java Tên-lớp-cần-thực-thi Tham-số), không cần thông qua trình duyệt Web.

Một trình duyệt Web hỗ trợ Java (Java-enabled browser) có máy ảo Java riêng. Hiện nay, các trình duyệt hỗ trợ Java như vậy khá nhiều: Netscape 2.0 trở lên, HotJava, Microsoft Internet Explorer 3.0 (bản beta 2 trở lên),... Các trang Web nhúng Java applet có chứa đường dẫn kiểu URL tới tập tin lớp chính của applet đó. Trình duyệt chỉ việc khởi động máy ảo Java và cung cấp cho trình nạp lớp đường dẫn này. Chú ý rằng mỗi lớp đều đưa ra tên của các lớp phụ nó cần, do đó trình nạp lớp phải nạp một số lớp phụ khác trước khi thực hiện chương trình.

IV. Các đặc tính chính của Java

Trong phần này, chúng ta sẽ điểm qua bảy đặc tính quan trọng khiến Java trở thành một công cụ phát triển mạnh, tin cậy.

IV.1. An ninh

An ninh là một vấn đề khó khăn hàng đầu mà người lập trình mạng gặp phải: Người sử dụng luôn e ngại hai điều: Thứ nhất, thông tin họ gửi đi trên mạng có thể bị đọc lén và thứ hai, hệ thống của họ có thể bị xâm nhập và phá hoại. Cấu trúc an ninh của Java nhằm vào giải quyết đồng thời hai vấn đề trên.

Cấu trúc an ninh của Java dựa vào ba thành phần: Trình nạp lớp, trình kiểm tra lớp và trình quản lý an ninh (SecurityManager). Chúng ta đã biết rằng trình kiểm tra lớp làm nhiệm vụ đảm bảo chắc chắn chương trình Java được biên dịch đúng đắn, khi thực hiện sẽ không gây lỗi ảnh hưởng đến hệ thống cũng như không đụng chạm đến những dữ liệu "riêng tư" trên máy khách hàng.

Bên cạnh đó, trình nạp lớp phân biệt rõ lớp nào đến từ mạng, lớp nào nằm ngay trên máy khách hàng. Điều này ngăn lớp tới từ mạng "giả dạng" một lớp trên máy khách hàng để thực hiện các tác vụ bị cấm đối với lớp này. Ngoài ra nó còn giúp tách biệt hoạt động của các lớp khác nhau tới từ các máy chủ khác nhau.

Trình quản lý an ninh kiểm soát các hoạt động một máy ảo Java được quyền làm đối các điều kiện khác nhau. Một ví dụ rất cơ bản là hoạt động đọc/ghi tập tin (file I/O) được quản lý chặt chẽ bởi trình quản lý an ninh: Các applet chỉ có quyền đọc/ghi các tập tin tại máy server chứa nó mà thôi! Tuy nhiên, trình quản lý an ninh cũng là một lớp trong Java, do đó chúng ta có thể viết một lớp an ninh riêng cho mình bằng cách tạo một lớp dẫn xuất từ lớp SecurityManager.

IV.2. Giao diện lập trình ứng dụng chuẩn - Core API

Java cung cấp cho người lập trình một thư viện các hàm chuẩn, đó là Core API. Các hàm chuẩn này được đặt trong các gói (package) - là tập hợp của các lớp có mối quan hệ với nhau (ví dụ như gói java.awt chứa các lớp Abstract Windowing Toolkit, giúp người lập trình xây dựng ứng dụng với giao diện GUI trên các platform khác nhau).

IV.3. Tương thích với nhiều kiểu phần cứng

Mã bytecode của Java có thể chạy trên hầu như mọi loại phần cứng và hệ điều hành hiện nay như: PC, Macintosh... cũng như các máy khác có chạy máy ảo Java. Một điểm nữa là thư viện các thủ tục chuẩn Java có chứa đầy đủ các hàm có thể dùng chung cho các platform khác nhau.

IV.4. Đặc tính động và phân tán

Hệ điều hành Windows cho phép các chương trình sử dụng chung và nạp tự động các thư viện liên kết động DLL. Chia sẻ tập tin DLL làm giảm dung lượng bộ nhớ cũng như đĩa cần dùng và tăng tính cấu trúc của chương trình. Java cũng có đặc tính này: các lớp được nạp tự động khi cần và nhiều chương trình có thể dùng chung một lớp. Nó còn hỗ trợ đặc tính phân tán, tức là các phần của chương trình có thể nằm trên máy chủ lẫn trên máy khách hàng.

IV.5. Hướng đối tượng

Lập trình hướng đối tượng (OOP) là phương thức viết các ứng dụng dễ bảo trì, dễ nâng cấp và đặc biệt là có thể tái sử dụng các mã lệnh. Java là một ngôn ngữ hướng đối tượng, do đó nó có

đầy đủ các đặc tính trên. Ngoài ra, thư viện lớp Java cung cấp khá đầy đủ cho người lập trình để bắt đầu một dự án mới.

IV.6. Đa luồng (multi-threads)

Các ứng dụng viết bằng Java có thể có nhiều tiến trình được xử lý cùng một lúc. Một ứng dụng đơn luồng chỉ có thể thực hiện một tác vụ tại một thời điểm: Giả sử ứng dụng đang bạn lấy từ trên mạng xuống một tập tin mất vài phút, trong thời gian này ứng dụng không thể làm các việc khác như vẽ lại màn hình... Với ứng dụng viết bằng Java, bạn có thể tạo hai tiến trình song song làm việc: một tiến trình nạp tập tin, một tiến trình khác làm nhiệm vụ cập nhật màn hình.

IV.7. Quản lý bộ nhớ và quá trình thu dọn 'rác'

Quản lý bộ nhớ là một vấn đề khá phức tạp đối với C và C++. Trong thời gian thực hiện chương trình, người lập trình chịu trách nhiệm khởi tạo các vùng nhớ, sau khi dùng xong lại giải phóng chúng. Chỉ cần một lỗi nhỏ trong đó cũng có thể làm cạn kiệt tài nguyên hay dẫn đến treo hệ thống. Java đã vứt bỏ gánh nặng này cho người lập trình: Các vùng nhớ được tạo ra trong chương trình sẽ tự động được giải phóng thông qua quá trình thu dọn "rác" (Garbage Collection). Các vùng nhớ tự động được giải phóng nếu như nó không được quy chiếu đến bởi bất cứ đối tượng đang hoạt động nào.

Các quản lý bộ nhớ của Java củng cố thêm tính an ninh của các máy ảo. Trong C và C++, người lập trình có thể truy cập đến hệ thống thông qua con trỏ (pointer). Với Java, phép toán với con trỏ bị cấm, do đó con trỏ luôn chỉ đến dữ liệu cần thiết của ứng dụng trên máy ảo, ngăn ngừa ứng dụng truy xuất đến tài nguyên bên ngoài máy ảo.

Cấu trúc và các đặc tính trên đây của Java cho thấy Java có những điểm mạnh và yếu riêng. Tuy nhiên, chúng ta có thể tin rằng, trong tương lai, Java sẽ dần dần hoàn thiện những nhược điểm, phát triển và tiếp tục làm thay đổi bộ mặt của Internet cũng như Intranet.

Kiểm tra sự tiến bộ

1. Bạn có thể viết các chương trình dạng thủ tục bằng Java. **Đúng/Sai**
2. Java là ngôn ngữ có kiểu dữ liệu chặt chẽ. **Đúng/Sai**
3.là chương trình Java chạy độc lập, và sử dụng giao diện đồ họa để người sử dụng nhập dữ liệu.
4.sử dụng JDBC API để kết nối cơ sở dữ liệu.
5.hiểu một dòng các lệnh máy tính trừu tượng.
6. Coalescing và Compaction là gì?
7. Lệnh được sử dụng để dịch file mã nguồn .java.
8. Lớp phải là lớp cha của các applet, applet là chương có thể nhúng vào trang Web hay chạy bằng Java AppletViewer.

Bài tập

1. Cài đặt Java 2
2. Gõ các lệnh sau tại dấu nhắc và liệt kê các tham số khác nhau của chúng:
 - javac
 - java

CHƯƠNG 2:

Các thành phần cơ bản của ngôn ngữ lập trình java

Văn phạm Java chỉ định cách viết như sau:

- Lời chú thích: Được thêm vào bởi người lập trình với mục đích giải thích.
- Các câu lệnh: Mỗi lệnh là dòng đơn của chương trình.
- Khối lệnh: Đặt các câu lệnh nhóm lại cùng với nhau như một khối
- Cấu trúc file: Các bộ phận cấu thành của một file Java nguồn và lệnh là được xác định rõ.
- Các từ khóa: Các từ được định nghĩa trước trong ngôn ngữ Java (không được sử dụng như danh hiệu).
- Danh hiệu: Các tên bạn đặt ra cho các lớp, các biến và các hàm. Danh hiệu không hạn chế các kí tự theo chỉ định, nhưng chưa phải được sử dụng tùy ý, có vài qui ước cho danh hiệu.
- Hằng (literals): Các giá trị hằng được viết một cách khác nhau tùy thuộc vào kiểu dữ liệu.

Ví dụ: để phân biệt các kí tự "123" với số 123.

- Biểu thức: Sự kết hợp nhiều mối quan hệ để tạo ra một giá trị dữ liệu.
- Toán tử: Các toán tử được thực hiện như phép cộng, trừ, nhân, các toán tử toán học và không toán học khác.

Mỗi khái niệm văn phạm trên sẽ được mô tả trong các phần sau:

I. Ghi chú (Comment)

Các lời chú thích có thể được thêm vào trong mã nguồn Java bằng hai cách giống như trong C++. Cách thứ nhất bắt đầu lời chú thích với dấu `/*` và kết thúc `*/` cho phép đặt nhiều dòng chú thích trong cặp dấu trên.

Ví dụ:

```
a = b + c;  
/* Here is comment which  
   extends across two lines  
*/
```

Nói chung bạn không thể đặt các chú thích trong chú thích

Ví dụ:

```
/* Here is a comment which /* a comment within a comment*/  
   extends across two line*/
```

Trong ví dụ này chú thích đầu tiên kết thúc trước bằng dấu `*/` ở dòng hai không bắt đầu mở đầu chú thích. Kết quả của câu này là lỗi trong quá trình biên dịch.

Nhiều dòng chú thích có ý nghĩa đặc biệt cho tiện ích javadoc khi kí hiệu đầu tiên bên trong dấu chú thích là một dấu `*`.

Ví dụ:

```
a = b+ c;
/* * This comment has epecial meaning for the javadoc
    unitily.It will be part of documenttation
    automaticlly generated By the javadoc program
*/
```

Cách thứ hai đánh dấu `//` và ghi lời chú thích sau đó trên cùng một dòng.

Ví dụ:

```
a = b + c;//this comment extends to the end of this line of text
```

Những chú thích này có thể đặt trong cặp dấu chú thích `/* */`.

Như vậy, ghi chú nhằm giúp cho người lập trình dễ dàng nhớ lại quá trình suy nghĩ nhất là khi cần bổ sung, sửa chữa nâng cấp chương trình

Có 3 loại ghi chú trong chương trình Java:

Bắt đầu	Kết thúc	Mục đích
<code>/*</code>	<code>*/</code>	Đoạn code bị giới hạn là phần ghi chú.
<code>//</code>		Ghi chú trên 1 dòng, trình biên dịch bỏ qua từ <code>//</code> đến cuối dòng
<code>/**</code>	<code>*/</code>	Ghi chú dành cho javadoc, trình biên dịch bỏ qua

II.Câu lệnh và khối lệnh

II.1.Câu lệnh

Một câu lệnh trong Java được kết thúc bằng 1 dấu chấm phẩy (`;`)

Thí dụ:

Dòng `a = b + c + d + e + f + g;`

Cũng giống như `a = b + c + d +`
`e + f + g;`

Khoảng trống giới hạn giữa các lệnh có thể bao gồm bất kì số kí tự trống nào. Các kí tự trống là các khoảng trống, các dấu tab, các dấu về đầu dòng, xuống dòng.

Chú ý: Trên hệ thống UNIX và Masintosh, mỗi dòng văn bản thường kết thúc bởi ký tự xuống dòng (mã ASCII 13). Trong Windows, các dòng văn bản thường không giới hạn bởi các

ký tự xuống dòng, về đầu dòng (CR/CF hay mã ASCII là 13/10). Java biên dịch nhận ra các ký tự trên cũng như các ký tự trống và không quan tâm đến những dòng văn bản đã kết thúc.

II.2. Khối lệnh

Các câu lệnh đơn có thể nhóm lại thành các khối bởi vậy một khối lệnh có thể dễ dàng kiểm soát sự thực hiện của nhiều câu lệnh khác. Bộ lệnh của Java không giới hạn trong cặp ngoặc nhọn { và }. Bạn đã nhìn thấy các khối lệnh được dùng để tạo thành nhóm các lệnh thuộc một lớp:

```
class Flight {
    int    altitude;
    int    heading;
    int    speed;
    float  latitude;
float  longintude;
//change the flight's heading by angle degree
void  turnFlight (int angle){
    heading = (heading + angle) % 360 ;
    // make sure angle is in the range 0-359 degrees
    if (heading<0)
heading = heading + 360 ;
} //end turnFlight
} //end Flight
```

Như bạn thấy các khối lệnh có thể đặt trong các khối khác

Tổng số các khoảng trống đặt giữa các dấu ngoặc nhọn và các câu lệnh là tùy ý nhưng cũng có qui ước là dấu ngoặc nhọn bên trái phải đặt sau một dòng (hay bắt đầu một dòng tiếp theo), dấu ngoặc nhọn bên phải là thuộc chính dòng đó và chỗ thụt đầu dòng thường dùng để làm nổi bật nhóm lệnh.

III. Tập ký tự dùng trong JAVA

Mọi ngôn ngữ đều được xây dựng từ một bộ ký tự nào đó. Các ký tự được nhóm lại theo nhiều cách để tạo nên các từ. Đến lượt mình các từ lại được liên kết theo một quy tắc nào đó để tạo nên các câu lệnh, từ khóa, tên...

Ngôn ngữ JAVA được xây dựng trên bộ ký tự sau :

26 chữ cái hoa : A..Z và 26 chữ cái thường a..z.

10 chữ số : 0..9

Các ký hiệu toán học : +, -, *, /, %, =, ()...

Dấu nối : _

Các ký hiệu đặc biệt khác : ;, :, {}, [], ?, \, &, |, #, \$...

Ngoài ra ngôn ngữ JAVA còn dùng bộ ký tự Unicode. Đó là một bộ ký tự, nó không chỉ bao gồm những ký tự đã có trong bộ ký tự ASCII mà còn có vài triệu ký tự khác tương ứng với hầu hết các bảng chữ cái trên thế giới.

IV. Từ khóa và tên

IV.1. Tên

Cũng tương tự như các ngôn ngữ lập trình khác như C, C++, ngôn ngữ lập trình JAVA cũng quy định cách đặt tên như sau :

Tên có thể được bắt đầu bằng một ký tự, hoặc dấu \$, _... nhưng không thể bắt đầu bằng một số và không có dấu cách trong tên. Ngoài ra nó còn có thể dùng bộ ký tự Unicode để đặt tên. JAVA phân biệt chữ hoa và chữ thường.

IV.2. Từ khóa

Từ khóa là những từ có ý nghĩa xác định. Thường được sử dụng để khai báo các kiểu dữ liệu, để viết các toán tử và các câu lệnh.

Sau đây liệt kê một số từ khóa của JAVA

abstract	case	const	else
for	boolean	cast	continue
extends	future	break	catch
default	final	generic	byte
char	do	finally	goto
byvalue	class	double	float
implements	inteface	operator	public
super	thows	volatile	import
long	rest	outer	swicth
transient	while	inner	native
package	return	synchronized	try
instanceof	new	private	short
this	var	int	null
protected	static	throw	void

Chú ý:

- Không được dùng từ khóa để đặt tên cho hằng, biến, mảng, hàm...
- Từ khóa phải viết bằng chữ thường

V. Kiểu dữ liệu

Mỗi biến phải có một kiểu dữ liệu. Kiểu dữ liệu của biến xác định các giá trị mà biến có thể chứa đựng và các toán tử thao tác trên chúng.

Ngôn ngữ Java có 2 loại dữ liệu:

- Kiểu dữ liệu cơ bản - Primitive
- Kiểu dữ liệu dẫn xuất - Reference

Kiểu dữ liệu cơ bản là các khối dữ liệu được xác định trong ngôn ngữ

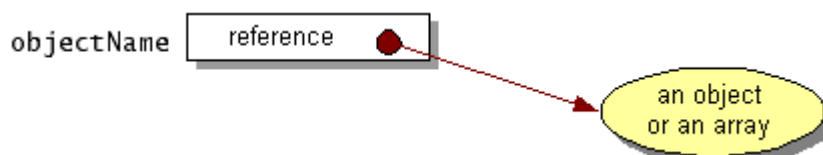
V.1.Kiểu dữ liệu cơ bản

Kiểu	Mô tả	Độ lớn	Min ÷ Max
boolean	true , false	1 bit	
char	Kí tự đơn	16 bits	\u0000 ÷ \uFFFF
byte	Byte-length	8 bits	-128 ÷ 127
short	integer	16 bits	-32768 ÷ 32767
int	Số nguyên ngắn	32 bits	-2147483648 ÷ 2147483647
long	Số nguyên Số nguyên dài	64 bits	-9223372036854775808 ÷ 9223372036854775807
float		32 bits	+/-3.40282347E+38 ÷ +/-1.40239846E-
double	Số thực Số thực	64 bits	45 +/-1.79769313486231570E ÷ +/-4.9065645841246544E-324

V.2.Kiểu dữ liệu dẫn xuất (Reference)

Array, class và interface đều là kiểu dữ liệu dẫn xuất. Giá trị của biến kiểu dẫn xuất trái ngược với kiểu dữ liệu gốc, nó tham chiếu tới (địa chỉ) một giá trị hay là tập hợp các giá trị mà biến khai báo.

Một kiểu Reference được gọi là một con trỏ hay là một địa chỉ bộ nhớ của ngôn ngữ lập trình khác. Ngôn ngữ lập trình Java không hỗ trợ rõ ràng cách sử dụng địa chỉ giống như các ngôn ngữ khác. Thay vào đó bạn có thể sử dụng tên biến.



V.3.Giá trị mặc định

Một lỗi hay gặp phải khi lập trình là sử dụng những biến chưa được khởi tạo. Điều này thường gây ra lỗi khó đoán được bởi vì nó có thể nhận bất kỳ giá trị nào trong vùng nhớ khi chương trình bắt đầu chạy. Java đã tránh được lỗi trên bằng cách khởi tạo giá trị mặc định cho các biến

Kiểu	Giá trị mặc định
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	null
boolean	false
dẫn xuất	null

VI. Hằng (literal)

Hầu hết các ngôn ngữ đều có một cách để khai báo biến là hằng số (đó là giá trị không đổi). Java không là ngoại lệ Từ khóa `final` chỉ dẫn một biến cục bộ hoặc biến thành phần không thể thay đổi. Mục đích chính sử dụng của biến `final` như là biểu thị một hằng số. Bạn có thể thích hằng số là tên, và tên đó được định nghĩa trong một chỉ định đơn trong mã nguồn của bạn. Nếu sau này bạn phải thay đổi số trong mã nguồn của bạn, bạn chỉ cần thay đổi tại biến `final` đã định nghĩa.

(Các hàm thành và các lớp có thể khai báo như là `final` . Một hàm thành phần `final` không thể chồng lên nhau, và một lớp `final` không thể là lớp con).

Cú pháp:

```
final Type ConstName = Value;
```

Ví dụ:

```
final int MAX = 10;
```

Lưu ý: Cả hai cách khai báo hằng trong C và C++ là `#define` và `const` đều không dùng trong JAVA.

VII. Biến

Một chương trình tham chiếu tới giá trị của biến bằng tên biến. Tên biến phải là duy nhất trong cùng một phạm vi, có thể có nhiều biến cùng tên nhưng khai báo ở các phạm vi khác nhau.

Mọi biến cần được khai báo trước khi sử dụng. Việc khai báo được thực hiện theo mẫu sau. Là những vị trí trong bộ nhớ mà giá trị có thể được lưu trữ ở đó chúng có tên và kiểu giá trị.

Cú pháp:

```
Type    Variable;
```

hay:

```
Type    Variable = Value;
```

VII.1.Kiểu biến

Có thể là một trong ba hình thức sau :

- Một trong 8 kiểu dữ liệu cơ bản.
- Tên của một lớp.
- Một mảng.

Để khai báo một kiểu biến mới, ta phải khai báo một lớp mới. Sau đó các biến mới được khai báo là kiểu của lớp đó.

JAVA có 3 loại biến : Biến lớp, biến đối tượng , biến cục bộ.

VII.1.1.Biến đối tượng

Thường dùng để định nghĩa thuộc tính, trạng thái cho một đối tượng nó có thể là biến toàn cục của một đối tượng. Khai báo như sau :

```
Type    Variable;
```

VII.1.2.Biến lớp

Tương tự như biến đối tượng, nhưng giá trị của nó nằm trong chính lớp đó. Nó ảnh hưởng toàn cục đến một lớp và tất cả các đối tượng trong lớp. Rất thích hợp để trao đổi thông tin giữa các đối tượng khác nhau trong cùng một lớp hay theo dõi trạng thái toàn cục của một đối tượng. Để khai báo một biến là biến lớp ta dùng từ khóa static như sau :

```
static Type    Variable;
```

VII.1.3.Biến cục bộ

Được khai báo và sử dụng trong thân phương thức. Biến cục bộ bắt buộc phải gán giá trị trước khi sử dụng. Không giống như các ngôn ngữ lập trình khác. JAVA không có biến toàn cục. Biến đối tượng hay biến lớp được dùng để truyền thông tin toàn cục giữa hai hay nhiều đối tượng.

VII.1.4. Phạm vi của biến

Giống như ngôn ngữ COBOL tất cả các biến đều là biến toàn cục. Biến toàn cục là một biến có thể truy cập bất cứ từ đâu trong một chương trình và do vậy mà biến toàn cục phải là biến có tên độc nhất. Do tất cả các biến trong COBOL là toàn cục nên mỗi biến trong chương trình COBOL là duy nhất. Từ đó, trong thực tiễn việc sử dụng một biến đơn cho nhiều mục đích khác nhau trong từng phần của chương trình. Việc kiểm soát biến toàn cục là một công việc khó khăn và làm nhiều chương trình trở nên khó chịu. Đặc biệt, một thay đổi trong một phần nhỏ của mã có thể ảnh hưởng bất lợi đến hầu hết các phần khác của chương trình.

Giải pháp cho vấn đề rắc rối của biến toàn cục là sử dụng biến cục bộ, biến này có thời gian tồn tại giới hạn và quan hệ chỉ trong phần nhỏ của mã. Bạn có thể dùng hai biến cục bộ có cùng tên định danh trong các phần khác nhau của chương trình.

Các biến được định nghĩa với một hàm thành phần là cục bộ liên quan đến hàm thành phần, vì vậy bạn có thể dùng cùng tên biến trong vài hàm thành phần, như ví dụ sau:

```
Class MyClass {
    int i;          // member variable
    int First() {
        int j;     // local variable
        // i và j đều có thể truy cập từ đây
        return 1;
    }
    int Second() {
        int j;     // local variable
        // i và j đều có thể truy cập từ đây
        return 2;
    }
}
```

Biến j định nghĩa trong hàm First() được tạo ra khi hàm này được gọi và mất đi khi hàm này thoát. Điều này cũng đúng cho biến cục bộ j trong hàm Second(). Biến j xuất hiện đồng thời trong cả hai hàm nhưng chúng không gây xung đột bởi vì hai biến cục bộ hoàn toàn độc lập với nhau. Có một cách khác để nghĩ là: tưởng tượng trình biên dịch sẽ đổi tên các biến cục bộ đó (j) để tên của mỗi biến là duy nhất (như ví dụ: j1 và j2).

VIII. Chuyển đổi kiểu dữ liệu

Trong JAVA có các kiểu dữ liệu cơ bản, kiểu đối tượng. Do đó ta có 3 dạng chuyển đổi kiểu dữ liệu.

- Chuyển đổi cho các kiểu dữ liệu cơ bản

Cú pháp : (NewType) Value;

- Chuyển đổi kiểu cho các đối tượng: Đối tượng của một lớp có thể được chuyển đổi kiểu để trở thành một đối tượng của một lớp khác. Với điều kiện các đối tượng muốn đổi kiểu phải thuộc các lớp thừa kế nhau.

Cú pháp : (NewClass) Object ;

- Chuyển đổi kiểu dữ liệu cơ bản sang đối tượng và ngược lại: Thông thường ta không thể chuyển đổi được nhưng trong gói java.lang có sẵn những lớp đặc biệt tương ứng với từng kiểu nguyên thủy. Như lớp Integer cho kiểu int, lớp Float cho kiểu float... do đó ta có thể tạo ra các đối tượng tương đương bằng cách dùng toán tử new.

ví dụ : `int Intobj = new Integer (32);`

Khi muốn lấy lại giá trị ban đầu ta dùng phương thức có sẵn trong lớp tương đương. Như intValue() cho kiểu int...

IX. Biểu thức và Toán tử

Toán tử là những ký hiệu đặc biệt mà thông thường được dùng trong biểu thức. Như +, -, *, ...

IX.1. Các toán tử số học:

Tương tự như ngôn ngữ lập trình C và C++ Java gồm các phép toán như sau : +, -, *, /, %.

Toán tử	Cách dùng	Mô tả
+	op1 + op2	Cộng hai toán hạng
-	op1 - op2	Trừ
*	op1 * op2	Nhân
/	op1 / op2	Chia
%	op1 % op2	Chia lấy dư

Ví dụ:

```
public class ArithmeticDemo {
    public static void main(String[] args) {
        //a few numbers
        int i = 37;
        int j = 42;
        double x = 27.475;
```

```
double y = 7.22;
System.out.println("Variable values...");
System.out.println("    i = " + i);
System.out.println("    j = " + j);
System.out.println("    x = " + x);
System.out.println("    y = " + y);

//adding numbers
System.out.println("Adding...");
System.out.println("    i + j = " + (i + j));
System.out.println("    x + y = " + (x + y));

//subtracting numbers
System.out.println("Subtracting...");
System.out.println("    i - j = " + (i - j));
System.out.println("    x - y = " + (x - y));

//multiplying numbers
System.out.println("Multiplying...");
System.out.println("    i * j = " + (i * j));
System.out.println("    x * y = " + (x * y));

//dividing numbers
System.out.println("Dividing...");
System.out.println("    i / j = " + (i / j));
System.out.println("    x / y = " + (x / y));

//computing the remainder resulting from dividing numbers
System.out.println("Computing the remainder...");
System.out.println("    i % j = " + (i % j));
System.out.println("    x % y = " + (x % y));

//mixing types
System.out.println("Mixing types...");
System.out.println("    j + y = " + (j + y));
System.out.println("    i * x = " + (i * x));
}
}
```

Kết quả của chương trình:

Variable values...

i = 37

j = 42

x = 27.475

y = 7.22

Adding...

i + j = 79

x + y = 34.695

Subtracting...	$i / j = 0$
$i - j = -5$	$x / y = 3.8054$
$x - y = 20.255$	Computing the remainder...
	$i \% j = 37$
Multiplying...	$x \% y = 5.815$
$i * j = 1554$	Mixing types...
$x * y = 198.37$	$j + y = 49.22$
Dividing...	$i * x = 1016.58$

IX.2.Các phép toán tăng giảm

Dùng các toán tử ++ và -- để tăng giảm giá trị của biến.

IX.3.Toán tử quan hệ và điều kiện

Toán tử	Ý nghĩa	Ví dụ
==	Bằng	$a == 10;$
!=	Không bằng	$a != b$
<	Bé hơn	$5 < 6$
>	Lớn hơn	$7 > 4$
<=	Bé hơn hoặc bằng	$a <= 6$
>=	Lớn hơn hoặc bằng	$b >= 8$

IX.4.Toán tử luận lý

Các biểu thức mà có kết quả trả về là giá trị boolean, có thể kết hợp lại với nhau bằng các toán tử luận lý như AND (& hoặc &&), OR (| hoặc ||), XOR (^), NOT (!)...

IX.5.Các toán tử làm việc với bit

Giống C và C++, những toán tử làm việc với bit được dùng để thể hiện từng bit riêng biệt đối với số nguyên. Ta có bảng những toán tử làm việc với bit như sau :

Toán tử	Ý nghĩa
&	Và
	Hoặc
^	Hoặc loại trừ
<<	Dịch trái 1 bit

>>	Dịch phải 1 bit
>>>	Dịch phải 1 bit và điền 0 vào những chỗ vừa dịch
-	Đảo bit
>>=	Dịch trái rồi gán ($x=x<<y$)
<<=	Dịch phải rồi gán ($x=x>>y$)
>>>=	Dịch phải, điền 0 vào những chỗ vừa dịch rồi gán ($x=x>>y$)
$x \&=y$	$x=x\&y$
$x =y$	$x=x y$
$x^=y$	$x=x^y$

IX.6.Toán tử gán

Toán tử gán có dạng như sau:

Variable Operator = Value.

Điều này tương đương với:

Variable = Variable Operator Value;

Ví dụ : $a += 4$; tương đương với $a = a + 4$;

Toán tử	Cách dùng	Cách dùng khác
+=	$op1 \ += \ op2$	$op1 = op1 + op2$
-=	$op1 \ -= \ op2$	$op1 = op1 - op2$
*=	$op1 \ *= \ op2$	$op1 = op1 * op2$
/=	$op1 \ /= \ op2$	$op1 = op1 / op2$
%=	$op1 \ \% = \ op2$	$op1 = op1 \% op2$
&=	$op1 \ \&= \ op2$	$op1 = op1 \& op2$
=	$op1 \ = \ op2$	$op1 = op1 op2$
^=	$op1 \ ^= \ op2$	$op1 = op1 ^ op2$
<<=	$op1 \ <<= \ op2$	$op1 = op1 << op2$
>>=	$op1 \ >>= \ op2$	$op1 = op1 >> op2$
>>>=	$op1 \ >>>= \ op2$	$op1 = op1 >>> op2$

IX.7.Một số toán tử khác

Toán tử	Cách dùng	Mô tả
?:	$op1 \ ? \ op2 \ : \ op3$	Nếu $op1$ true thì trả về $op2$. Ngược lại thì trả về $op3$.
[]	$type \ []$	Khai báo mộ mảng chưa biết độ lớn, chứa các

		phần tử có kiểu là <i>type</i>
<code>[]</code>	<code>type[op1]</code>	Tạo một mảng với <i>op1</i> phần tử. Phải được sử dụng với toán tử <i>new</i> .
<code>[]</code>	<code>op1[op2]</code>	Truy cập đến một phần tử của mảng <i>op1</i> với chỉ số <i>op2</i> .
<code>.</code>	<code>op1.op2</code>	Một sự tham khảo tới hàm thành phần <i>op2</i> của <i>op1</i> .
<code>()</code>	<code>op1 (params)</code>	Định nghĩa hoặc gọi một phương thức có tên là <i>op1</i> với các tham số là <i>params</i> . Danh sách các tham số có thể là danh sách rỗng. Các phần tử trong danh sách cách nhau bằng dấu phẩy.
<code>(type)</code>	<code>(type) op1</code>	Chuyển đổi kiểu dữ liệu của <i>op1</i> với kiểu mới là <i>type</i> . Một ngoại lệ sẽ xảy ra nếu kiểu của <i>op1</i> không tương thích với <i>type</i> .
<code>new</code>	<code>new op1</code>	Khởi tạo một đối tượng hay một mảng. <i>op1</i> là một constructor hay một mảng
<code>instanceof</code>	<code>op1 instanceof op2</code>	Trả về true nếu <i>op1</i> là một an instance của <i>op2</i>

IX.8.Phép toán trên kiểu chuỗi (String)

Một biểu thức đặc biệt trong Java là dùng toán tử "+" để tạo hay ghép nối các chuỗi.

Ví dụ :

```
System.out.println(name+ "is a" + color );
```

Nếu có phần tử con nào không phải là kiểu String thì nó tự động chuyển sang kiểu String.

Mô tả phần này các bạn tham khảo phần đối tượng và các kiểu dữ liệu đối tượng.

IX.9.Độ ưu tiên các toán tử

```
( ), []
++, --, !, -, instanceof
new (type) Biểu_thức
*, /, %
+, -
<<, >>, >>>
< >
```

```
==, !=  
&  
^  
|  
? :  
, +=, -=, *=, /=, % =, ^=  
&=, |=, <=, >=, >>=
```

IX.10. Biểu thức

Biểu thức là sự kết nối các biến, các từ khóa hay các kí hiệu để trả về một giá trị của một kiểu nào đó. Giá trị này có thể là số, chuỗi, hoặc kiểu dữ liệu khác. Bạn có thể hiểu rằng một biểu thức là một cái gì đó được viết bên phải của một lệnh gán.

Biểu thức đơn giản nhất là các biến hay các hằng

Ví dụ: 22, a, "Hello"

Các biểu thức này có thể được tìm thấy bên phải của một lệnh gán như gán một chuỗi "Hello" cho một biến s:

```
s = "Hello";
```

Như trong C, việc gán một giá trị cho chính nó, đó là giá trị bên phải của lệnh gán

Ví dụ:

```
b = a = 25;
```

Trong ví dụ này giá trị 25 được gán cho biến a và 25 cũng được gán cho biến b

Có các loại biểu thức như biểu thức logic, biểu thức số học, biểu thức gán

Ví dụ : a <= 10;

- Biểu thức gán: Variable1= Variable2=...=Value;

- Biểu thức điều kiện :

```
Expression ? Expression_when_true : Expression_when_false;
```

X. Các câu lệnh điều khiển

X.1. Cấu trúc rẽ nhánh

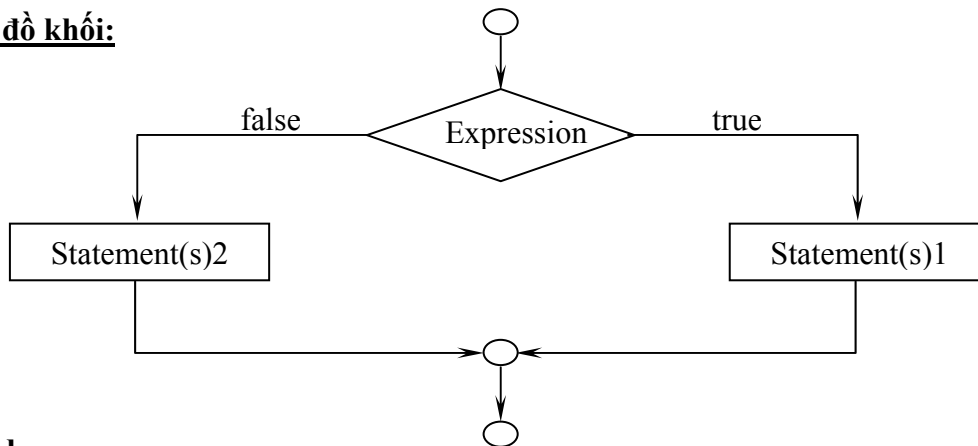
X.1.1. Cấu trúc điều kiện rẽ nhánh if

Cú pháp:

```
if (Expression) {  
    Statements1;  
}
```

```
else{  
    Statements2;  
}
```

Sơ đồ khối:



Ví dụ:

```
public class IfElseDemo {  
    public static void main(String[] args) {  
        int testscore = 76;  
        char grade;  
        if (testscore >= 90) {  
            grade = 'A';  
        } else if (testscore >= 80) {  
            grade = 'B';  
        } else if (testscore >= 70) {  
            grade = 'C';  
        } else if (testscore >= 60) {  
            grade = 'D';  
        } else {  
            grade = 'F';  
        }  
        System.out.println("Grade = " + grade);  
    }  
}
```

Kết quả chương trình là:

Grade = C

Expression phải là biểu thức logic trả về giá trị kiểu boolean (true hoặc false). Tương tự như C và C++ các câu lệnh điều kiện có thể lồng nhau tùy ý.

X.1.2.Cấu trúc điều kiện rẽ nhánh phức : switch

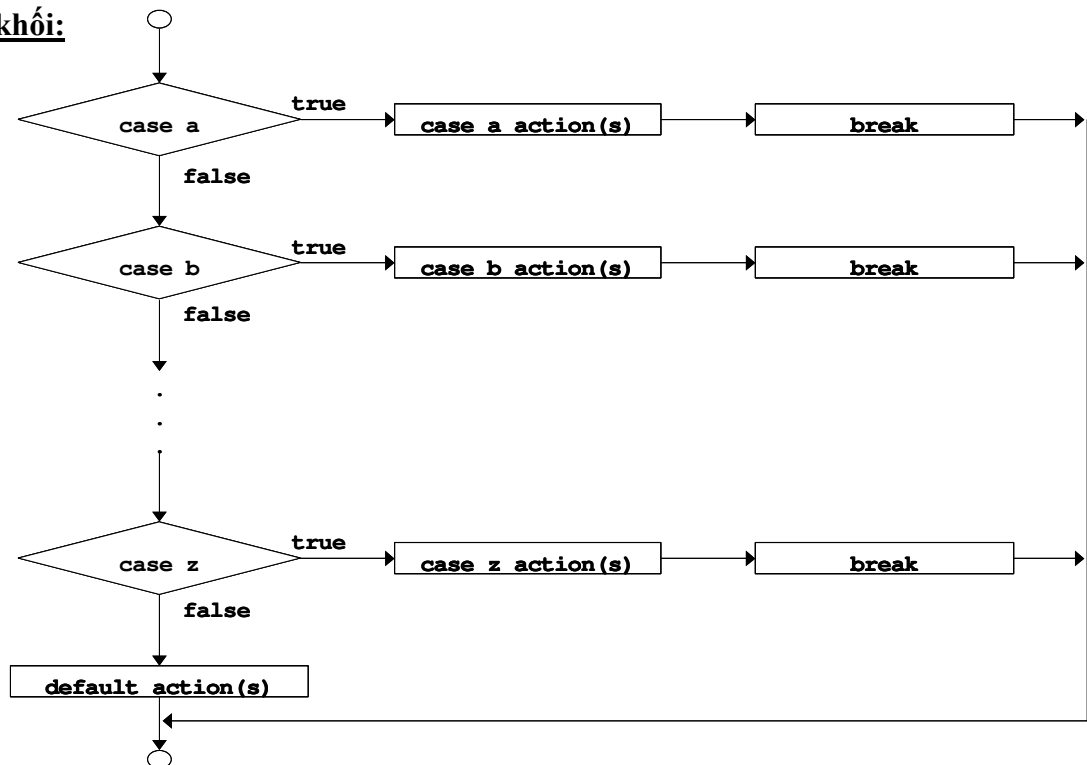
Cú pháp:

```
switch(integer expression) {  
    case integer expression:  
        statement(s)
```

```
        break;
    ...
    default:
        statement(s)
        break;
}
```

Expression : Biểu thức điều kiện phụ thuộc vào kiểu dữ liệu cơ bản (byte, int ...)

Sơ đồ khối:



Ví dụ:

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

        int month = 2;
        int year = 2000;
        int numDays = 0;

        switch (month) {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                numDays = 31;
                break;
        }
    }
}
```

```
        case 4:
        case 6:
        case 9:
        case 11:
            numDays = 30;
            break;
        case 2:
            if ( ((year % 4 == 0) && !(year % 100 == 0))
                || (year % 400 == 0) )
                numDays = 29;
            else
                numDays = 28;
            break;
    }
    System.out.println("Number of Days = " + numDays);
}
}
```

Kết quả của chương trình:

Number of Days = 29

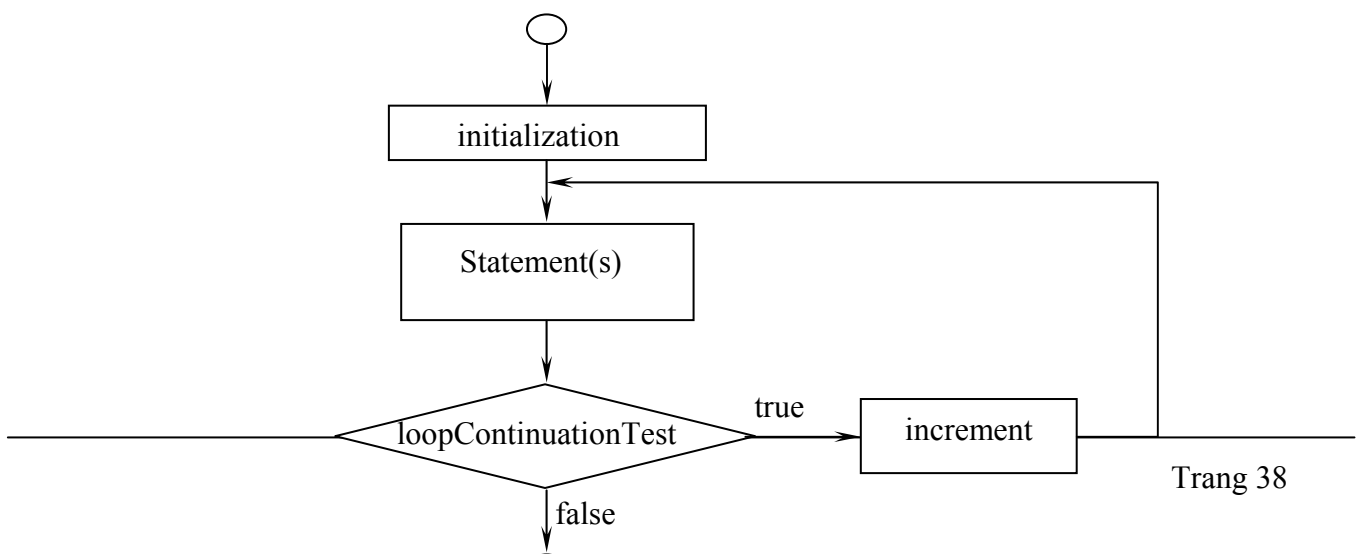
X.2.Cấu trúc lặp

X.2.1.Vòng lặp for

Cú pháp:

```
for ( initialization; loopContinuationTest; increment ) {
    Statements;
}
```

Sơ đồ khối:



Ví dụ:

```
public class ForDemo {  
    public static void main(String[] args) {  
        int[] arrayOfInts = { 32, 87, 3, 589, 12,  
                               1076, 2000, 8, 622, 127 };  
        for (int i = 0; i < arrayOfInts.length; i++) {  
            System.out.print(arrayOfInts[i] + " ");  
        }  
        System.out.println();  
    }  
}
```

Kết quả chương trình: 32 87 3 589 12 1076 2000 8 622 127.

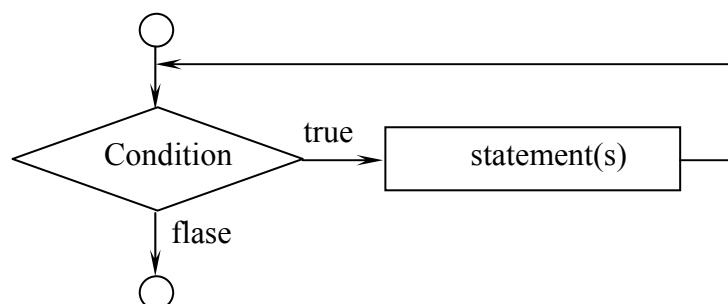
X.2.2.Vòng lặp while và do:

Tương tự cấu trúc của C và C++

Cú pháp:

```
while (Condition) {  
    Statement(s)  
}
```

Sơ đồ khối:



Ví dụ:

```
public class WhileDemo {  
    public static void main(String[] args) {  
  
        String copyFromMe = "Copy this string until you " +  
                             "encounter the letter 'g'.";  
        StringBuffer copyToMe = new StringBuffer();  
  
        int i = 0;
```

```
char c = copyFromMe.charAt(i);

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

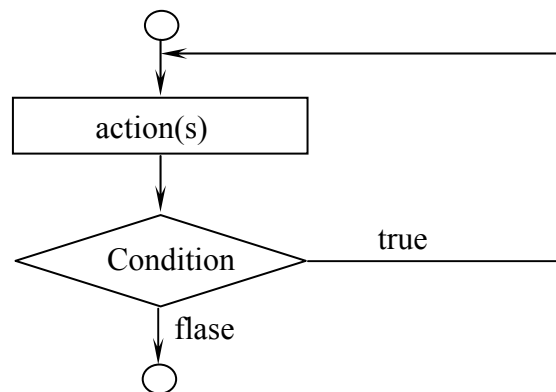
System.out.println(copyToMe);
}
```

Java cung cấp một cấu trúc khác tương tự như câu lệnh while.

Cú pháp:

```
do {
    Statement(s);
while (Condition);
```

Sơ đồ khối:



Ví dụ:

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

        String copyFromMe = "Copy this string until you " +
                            "encounter the letter 'g'.";
        StringBuffer copyToMe = new StringBuffer();
        int i = 0;
        char c = copyFromMe.charAt(i);
        do {
            copyToMe.append(c);
            c = copyFromMe.charAt(++i);
        } while (c != 'g');
        System.out.println(copyToMe);
    }
}
```


- Câu lệnh **break** : Dùng để thoát tức thời ra khỏi vòng lặp chứa nó.
- Câu lệnh **continue** : Dùng để quay trở về đầu vòng lặp chứa nó.

X.3.Ngoại lệ và câu lệnh nắm bắt ngoại lệ

Sử dụng các lệnh `try`, `catch` và `finally` để bắt giữ các ngoại lệ.

Cú pháp:

```
try {  
    statement1(s)  
} catch (exceptiontype name) {  
    statement(s)  
} catch (exceptiontype name) {  
    statement2(s)  
} finally {  
    statement3(s)  
}
```

`try`: định nghĩa một khối lệnh (*statement1(s)*) mà ngoại lệ có thể xảy ra.

`catch`: đi kèm với `try`, để bắt ngoại lệ, khi có ngoại lệ xảy ra thì *statement2(s)* thực hiện

statement3(s) luôn luôn thực hiện cho dù có hay không ngoại lệ

Kiểm tra sự tiến bộ

1. Trong Java, kiểu dữ liệu dạng byte nằm trong giới hạn từ.....đến.....
2. Hãy chỉ các danh định hợp lệ:

- a. `Tel_num`
- b. `Emp1`
- c. `8678`
- d. `batch.no`

3. Cho biết kết quả thu được khi thực hiện đoạn chương trình sau?

```
class me  
{  
    public static void main(String srgs[ ])   
    {  
        int sales=820;  
        int profit=200;  
        System.out.println((sale +profit)/10*5);  
    }  
}
```

4.là sự cài đặt của các hành động của đối tượng
5. Phương thức **public** có thể truy cập phương thức **private** trong cùng một lớp.

Đúng/Sai

6. ‘static’ hàm ý rằng phương thức không có mã và được bổ sung trong các lớp con **Đúng/Sai** Khi bạn không định nghĩa một hàm khởi tạo cho một lớp, JVM sẽ cung cấp một hàm mặc định hoặc một hàm khởi tạo ẩn (implicit). **Đúng/Sai** Vòng lặp while thực thi ít nhất một lần thậm chí nếu điều kiện được xác định là False **Đúng/Sai**

Bài tập

1. Hãy viết một đoạn chương trình để in ra dòng chữ ” Welcome to the world of Java”
2. Hãy viết hai phương thức khởi tạo tường minh cho một lớp dùng để tính diện tích hình chữ nhật. Khi một giá trị được truyền vào phương thức khởi tạo, nó cho rằng độ dài và chiều rộng bằng nhau và bằng giá trị truyền vào. Lúc đó, nó sẽ tính diện tích tương ứng. Khi hai giá trị được truyền vào, nó sẽ tính diện tích hình chữ nhật.
3. Viết một chương trình thực hiện những việc sau đây:
 - a. Khai báo và gán giá trị đầu cho các biến m và n là 100 và 200 tương ứng.
 - b. Theo các điều kiện: nếu m bằng 0, hiển thị kết quả tương ứng.
 - c. Nếu m lớn hơn n , hiển thị kết quả tương ứng.
 - d. Kiểm tra giá trị n là chẵn hay lẻ.
4. Viết một chương trình hiển thị tổng các bội số của 7 nằm giữa 1 và 100.
5. Viết chương trình để cộng bảy số hạng của dãy sau:
1!+2!+3!.....

CHƯƠNG 3:

APPLETS

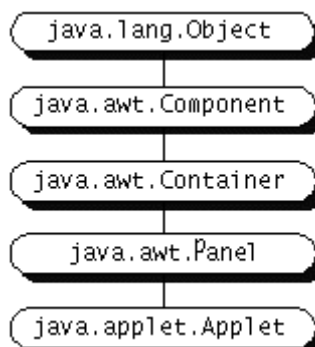
I.Đại cương về HTML

"Ngôn ngữ đánh dấu siêu văn bản" (Hypertext Markup Language - HTML) được dùng để khởi tạo các "siêu văn bản" trên WEB. Nó không phải là một ngôn ngữ lập trình phức tạp như C, C++, Java... Ngôn ngữ HTML chỉ một tập hợp những quy tắc chỉ định bởi những TAGS (Thẻ) và MARK (Đánh dấu) trên những thành phần của tài liệu.

Một tài liệu HTML chỉ là những văn bản thông thường, bằng những quy tắc đơn giản ta có thể đưa vào những hình ảnh, những "siêu liên kết", "siêu phương tiện"... Ngoài ra ta có thể bổ sung những hình ảnh, âm thanh sống động ...vào các trang Web của mình bằng cách tăng cường Java. Đó chính là các APPLETS.

II.Tổng quan về applet

Các lớp trong chương trình Java luôn là dẫn xuất từ một lớp chuẩn nào đó của JDK. Mỗi applet là dẫn xuất của lớp Applet theo sơ đồ sau:



II.1. Ví dụ về Applet

Sau đây là mã nguồn của một applet gọi là *Simple*

```
import java.applet.Applet;
import java.awt.Graphics;

public class Simple extends Applet {

    StringBuffer buffer;

    public void init() {
```

```
        buffer = new StringBuffer();
        addItem("initializing... ");
    }

    public void start() {
        addItem("starting... ");
    }

    public void stop() {
        addItem("stopping... ");
    }

    public void destroy() {
        addItem("preparing for unloading...");
    }

    void addItem(String newWord) {
        System.out.println(newWord);
        buffer.append(newWord);
        repaint();
    }

    public void paint(Graphics g) {
        //Draw a Rectangle around the applet's display
        area.
            g.drawRect(0, 0, size().width - 1,
            size().height - 1);

        //Draw the current string inside the rectangle.
        g.drawString(buffer.toString(), 5, 15);
    }
}
```

II.2.Vòng đời của một Applet

Có thể sử dụng ví dụ trên để minh họa cho vòng đời của một applet

II.2.1.Nạp một Applet

Bạn có thể thấy "initializing... starting..." ở trên, đó là kết quả trả về khi một applet đang nạp. Khi một applet nạp sẽ xảy ra các việc sau:

- Một instance của lớp con applet được tạo ra.
- Đối tượng applet này tự khởi tạo mình, bằng phương thức `init()`.

- Đối tượng applet này bắt đầu chạy, phương thức `start()` được gọi, tiếp theo là phương thức `run()`.

II.2.2. Rời khỏi và quay trở về trang web chứa applet

Khi người dùng rời khỏi trang web chứa applet sang một trang khác, khi đó phương thức `stop()` của applet sẽ được gọi. Khi người dùng quay về lại trang đó thì applet tự động kích hoạt lại phương thức `start()`. Quá trình này cũng xảy ra khi người dùng thay đổi của sổ chứa applet (*minaturize*, *minimize*, và *close*.)

Chú ý: Một vài trình duyệt có thể sẽ nạp lại applet khi ta trở lại trang chứa applet đó như thể có thể xảy ra một lỗi nếu một applet tự khởi tạo mà không cần nạp lại.

II.2.3. Nạp lại Applet (Reloading the Applet)

Một số trình duyệt cho phép người dùng nạp lại applet, nghĩa là người dùng đã thoát applet và nạp lại một lần nữa. Vì mỗi khi thoát khỏi Applet phương thức `stop()` sẽ tự động thực hiện, xảy ra quá trình dọn rác bằng phương thức `destroy()`. Như vậy, applet đã giải phóng các tài nguyên mà nó đã dùng và applet chính thức không còn trong bộ nhớ. Và khi được nạp lại, quá trình nạp sẽ xảy ra giống như đã nói ở trên.

II.2.4. Thoát khỏi trình duyệt

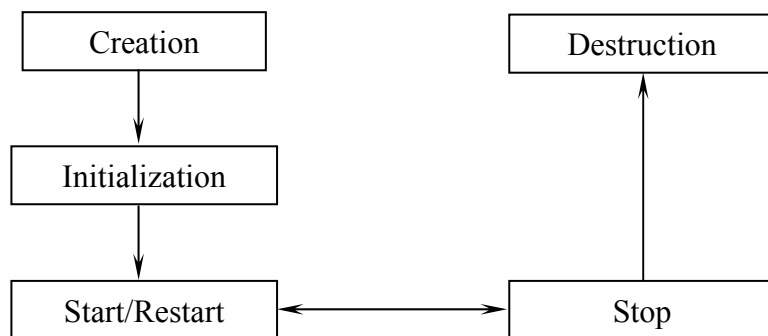
Khi người dùng thoát khỏi trình duyệt (hay ứng dụng chứa applet) thì applet tự động gọi phương thức `stop()` để dọn dẹp tài nguyên hệ thống trước khi trình duyệt thoát.

II.2.5. Tóm tắt

Một applet có thể trải qua các sự kiện sau:

- Nó có thể tự khởi tạo (*initialize*).
- Nó có thể tự động khởi động (*start*).
- Nó có thể tự dừng (*stop*).
- Tự dọn dẹp tài nguyên hệ thống trước khi thoát.

Sơ đồ:



II.3.Các phương thức cơ bản

```
public class Simple extends Applet {  
    . . .  
    public void init() { . . . }  
    public void start() { . . . }  
    public void stop() { . . . }  
    public void destroy() { . . . }  
    . . .  
}
```

Trong applet Simple cũng như tất cả các applet khác đều là lớp con của lớp Applet. Lớp Simple đã override lại 4 phương thức của Applet vì vậy nó có thể trả lời các sự kiện: `init`, `start`, `stop`, `destroy`.

II.3.1.init()

Dùng để khởi tạo một lần, những hoạt động không kéo dài. Phương thức này thường chứa các mã lệnh mà ta thường dùng trong các constructor.

II.3.2.start()

Applet sau khi khởi tạo cần làm công việc gì đó thì phải cài đặt phương thức `start()`, trừ khi phải trả lời trực tiếp tác động của người dùng. Phương thức này có thể thực hiện một số việc của applet hay khởi tạo các tuyến đoạn.

II.3.3.stop()

Đa số các applet khi cài phương thức `start()` đều có cài `stop()`. Phương thức này dùng để tạm ngưng các hoạt động của applet nhưng không giải phóng tài nguyên hệ thống.

II.3.4.destroy()

Phương thức này thường dùng để giải phóng tài nguyên hệ thống.

II.4.Các phương thức vẽ và nắm bắt sự kiện

Simple applet định lại phương thức paint override phương thức paint của applet:

```
class Simple extends Applet {  
    . . .  
    public void paint(Graphics g) { . . . }  
    . . .  
}
```

Phương thức paint là một trong hai phương thức mà một applet có thể override:

- paint: Là phương thức vẽ cơ bản. Nhiều applet cài đặt phương thức paint để vẽ những trình bày của nó bên trong trình duyệt.
- update: Là phương thức mà bạn có thể sử dụng sau khi đã thực hiện *paint* để tăng thêm hiệu quả vẽ.

Applet thừa hưởng các phương thức này từ lớp AWT. Applet còn thừa hưởng các phương thức về sự kiện như nhấp chuột, rê chuột,... trong lớp Component của AWT.

II.5.Các phương thức cho lập trình giao diện người dùng

II.5.1.Các thành phần UI xây dựng sẵn

Thư viện AWT hỗ trợ các thành phần cho lập trình giao diện (UI components) (Các lớp dùng để cài đặt mỗi thành phần được đặt trong dấu ngoặc đơn):

- Buttons (`java.awt.Button`)
- Checkboxes (`java.awt.Checkbox`)
- Text fields (`java.awt.TextField`)
- Text areas (`java.awt.TextArea`)
- Labels (`java.awt.Label`)
- Lists (`java.awt.List`)
- Choices (`java.awt.Choice`)
- Sliders and scrollbars (`java.awt.Scrollbar`)
- Drawing areas (`java.awt.Canvas`)
- Menus (`java.awt.Menu`, `java.awt.MenuItem`, `java.awt.CheckboxMenuItem`)

- Containers (`java.awt.Panel`, `java.awt.Window` and its subclasses)

II.5.2. Các phương thức để sử dụng các thành phần UI trong các Applet

Bởi vì lớp Applet thừa kế từ lớp AWT `Container` nên dễ dàng thêm các thành phần vào các applet và sử dụng các trình quản lý lớp (layout managers) để điều khiển vị trí của các thành phần trên màn hình. Đây là một số phương thức của `Container` mà các applet có thể sử dụng:

`add` : Thêm một Component.
`remove` : Xoá một Component.
`setLayout` : Thiết lập trình quản lý layout.

II.5.3. Thêm một Text Field không edit được vào applet `Simple`

Để làm cho đơn giản applet sử dụng lớp `TextField` để khai báo một text field chỉ được cuộn, không được sửa chữa

Sự thay đổi cho thấy sau đây:

```
//Không cần Import java.awt.Graphics
//khi applet này không cài đặt
//phương thức paint.
. . .
import java.awt.TextField;
public class ScrollingSimple extends Applet {
    //Instead of using a StringBuffer,
    //use a TextField:
    TextField field;

    public void init() {
        // Tạo một text field và làm cho nó không thể sửa chữa.
        field = new TextField();
        field.setEditable(false);
        //Set the layout manager so that the text field will be
        //as wide as possible.
        setLayout(new java.awt.GridLayout(1,0));
        //Đặt text field vào Applet.
        add(field);
        validate();
        addItem("initializing... ");
    }
    . . .
```




```
void addItem(String newWord) {  
    //Nối chuỗi vào TextField.  
    String t = field.getText();  
    System.out.println(newWord);  
    field.setText(t + newWord);  
    repaint();  
}  
  
//Phương thức paint không cần thiết,  
//khi TextField tự động repaints.
```

Xem lại phương thức init tạo ra một text field không edit được. Nó thiết lập trình quản lý cách trình bày vào applet làm cho text field có thể rộng ra được sau đó đặt text field vào applet

Sau cùng phương thức init gọi một phương thức hợp lệ (phương thức mà Applet thừa hưởng từ Component). Điều này làm cho có hiệu lực đối với một hoặc nhiều thành phần thêm vào applet và bảo đảm rằng các thành phần này đã tự vẽ ra trên màn hình.

Sau đây là kết quả của applet:



initializing... starting...

II.6. Giới hạn của Applet

Mục này cho thấy một cách tổng quan về mặt hạn chế của applet và các khả năng mà chúng có thể có được.

II.6.1. Giới hạn về bảo mật

Tất cả các trình duyệt có cài đặt chế độ bảo mật đều giữ nó lại trong hệ thống bảo mật. Mục này mô tả chế độ bảo mật mà các trình duyệt hiện thời kèm vào. Tuy nhiên việc cài đặt cơ chế bảo mật giữa các trình duyệt là khác nhau. Ngoài ra cơ chế bảo mật thường thay đổi. Thí dụ như nếu một trình duyệt được phát triển để chỉ sử dụng trong môi trường tin cậy thì cơ chế bảo mật có thể lỏng lẻo hơn những mô tả ở đây.

Các trình duyệt hiện thời thường giới hạn hoạt động của applet khi nạp các applet từ mạng về như sau:

- Một applet không thể nạp các thư viện hay các phương thức sử dụng mã nguồn.
- Không được đọc hay ghi bình thường lên các tập tin ở máy chủ thực thi chúng.
- Không được tạo các kết nối mạng ngoại trừ kết nối với máy đang chạy nó.

- Không thể khởi động bất cứ một chương trình nào trên máy chủ chạy nó.
- Không được đọc bất cứ tính chất nào của hệ thống.
- Cửa sổ mà một applet mở ra sẽ khác với các cửa sổ của các ứng dụng khác mở.

Mỗi trình duyệt có một đối tượng `SecurityManager` được cài đặt để thực hiện vấn đề bảo mật. Khi `SecurityManager` dò tìm ra sự vi phạm thì nó ném ra một ngoại lệ `SecurityException`. Applet của bạn có thể bắt ngoại lệ `SecurityException` đó và tác động trở lại một cách thích hợp.

II.6.2.Các khả năng của Applet

Gói `java.applet` cung cấp một API cho các applet một số khả năng mà ứng dụng đơn (application) không có được. Thí dụ như applet có thể tạo âm thanh cái mà các chương trình khác không thực hiện được.

Sau đây là một số cái mà các trình duyệt hiện hành và các applet viewer cho phép applet thực hiện:

- Applet có thể thường xuyên tạo kết nối tới máy đang chạy nó.
- Khi applet chạy trong trình duyệt thì có thể dùng tài liệu HTML để hiển thị.
- Applet có thể gọi tới một phương thức công cộng của một applet khác trong cùng trang Web.
- Các applet được nạp từ tập tin cục bộ sẽ không bị giới hạn giống các applet được nạp từ mạng xuống.
- Hầu hết các applet ngừng thực thi khi rời khỏi trang web chứa nó, nhưng không nhất thiết như vậy.

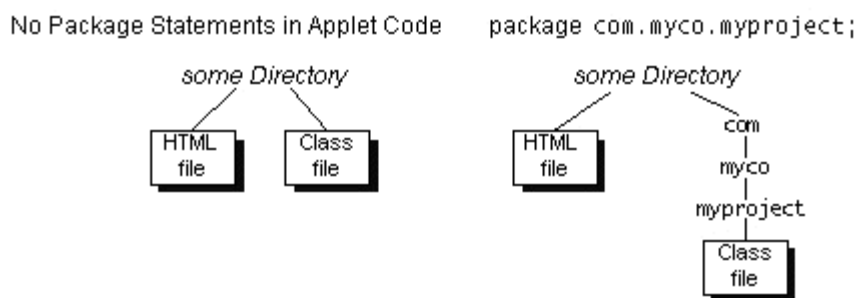
II.7.Test một applet

Một khi đã viết vài đoạn mã cho applet của bạn, bạn sẽ muốn chạy applet để kiểm tra nó. Để chạy applet trước hết bạn phải đặt applet vào trong một trang HTML bằng cách sử dụng thẻ (tag) `<APPLET>`. Sau đó đánh địa chỉ URL của trang HTML đó trong một trình duyệt có hỗ trợ Java.

Đây là một thẻ `<APPLET>` đơn giản:

```
<APPLET CODE=AppletSubclass.class WIDTH=anInt HEIGHT=anInt>
</APPLET>
```

Thẻ này cho trình duyệt biết cần nạp một applet có tên là *AppletSubclass*. Hình sau cho thấy nơi để đặt file applet, quan hệ với tài liệu HTML chứa thẻ `<APPLET>`. Như hình vẽ thì trừ khi applet được đặt trong một gói (package), applet phải đặt trong cùng thư mục với tệp HTML chứa thẻ `<APPLET>`.



Khi một trình duyệt hỗ trợ Java đựng một thẻ `<APPLET>`, thì nó dự trữ một vùng hiển thị với chiều rộng (width) và chiều cao (hight) cho applet, nạp mã bytecode của applet, tạo một instance của lớp đó, sau đó gọi phương thức `init` và `start`.

Thẻ `<APPLET>` có rất nhiều tùy chọn do đó bạn có thể tùy biến cho phù hợp với sự thực thi của applet mà bạn tạo ra.

III. Các tính năng cao cấp của Applet API

Applet API cung cấp cho bạn các tính năng cao cấp về mối quan hệ gần gũi giữa các applet và các trình duyệt Web. API được cung cấp bởi gói `java.applet` -- chủ yếu là lớp `Applet` và giao tiếp (interface) `AppletContext`. Nhờ vào applet API mà các applet có thể thực hiện được các việc sau:

- Đưa thông báo cơ bản từ trình duyệt.
- Tải các files dữ liệu xác định bởi URL của applet hay trang HTML mà applet được nhúng.
- Hiển thị các chuỗi ngắn về trạng thái.
- Làm trình duyệt hiển thị tài liệu.
- Tìm các applet khác trên cùng trang.
- Tạo âm thanh.

Lấy tham số được chỉ ra bởi người dùng trong thẻ `<APPLET>`.

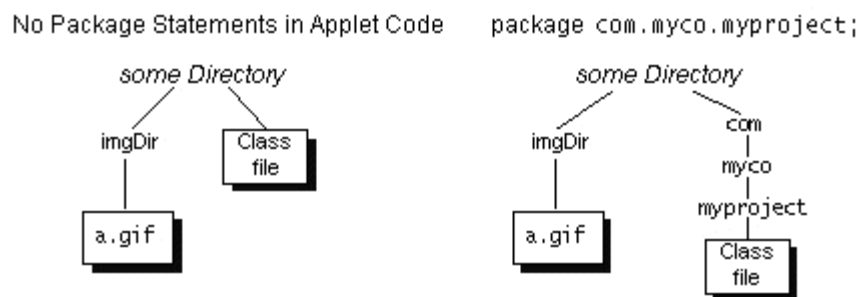
III.1. Tìm kiếm và nạp các file dữ liệu

Bất cứ lúc nào applet cần để nạp vài dữ liệu từ tệp được xác định bởi URL (một URL mà không chỉ rõ đầy đủ vị trí của file), applet thường sử dụng hoặc là code base hoặc là document base để hoàn chỉnh địa chỉ URL. Code base được trả về bởi phương thức `getCodeBase`, nó chỉ ra địa chỉ URL xác định thư mục chứa lớp applet đã được nạp. Document base được trả về bởi phương thức `getDocumentBase`, nó chỉ ra thư mục chứa trang HTML mà applet đó được nhúng.

Trừ khi thẻ `<APPLET>` chỉ rõ code base, cả code base và document base đều được đặt cùng thư mục trên cùng một server.

Dữ liệu mà applet luôn cần hoặc những nhu cầu tin cậy cần sao lưu, thông thường được chỉ rõ tới code base. Dữ liệu mà người dùng applet chỉ rõ, thường dùng các tham số, thông thường được chỉ tới document base.

Lớp Applet định nghĩa các hình thức thuận lợi để nạp hình ảnh và âm thanh, mà hình ảnh và âm thanh được chỉ rõ bởi URL. Ví dụ, lấy một applet và đặt nó vào trong một cấu trúc thư mục như hình sau:



Để tạo một đối tượng `Image` sử dụng một file hình ảnh `a.gif` trong thư mục `imgDir` thì applet có thể sử dụng đoạn mã sau:

```
Image image = getImage(getCodeBase(), "imgDir/a.gif");
```

III.2. Hiện thị chuỗi tình trạng ngắn

Tất cả những trình xem applet, từ Applet Viewer đến trình duyệt có hỗ trợ Java đều cho phép các applet hiện thị một chuỗi tình trạng ngắn. Trong sự thi hành hiện thời chuỗi này xuất hiện trên hàng tình trạng ở đáy của sổ duyệt applet. Trong các trình duyệt tất cả applet trên cùng trang, cũng như chính các trình duyệt nói chung là chia sẻ hàng tình trạng đó.

Không nên đặt các thông tin cốt yếu vào dòng trạng thái. Nếu nhiều người dùng cần thông tin, thay vào đó ta nên hiển thị trong vùng applet. Nếu chỉ một ít thông tin phức tạp mà người dùng cần thì nên xuất thông tin ra ở thiết bị đầu ra chuẩn.

Dòng tình trạng không phải là nổi bật thông thường và nó có thể bị ghi đè lên bởi một applet hay một trình duyệt khác. Từ những lý do trên, tốt nhất sử dụng để hiển thị thông tin phụ, nhất thời mà thôi. Thí dụ như applet nạp các file ảnh và hiển thị tên của file ảnh trong lúc đang nạp chúng.

Ví dụ:

```
showStatus("MyApplet: Loading image file " + file);
```

III.3. Hiển thị tài liệu trong trình duyệt

Bạn có bao giờ muốn applet hiển thị một tài liệu văn bản định dạng HTML không? Ở đây có cách để làm điều đó: nói cho trình duyệt hiển thị văn bản cho bạn.

Với phương thức `showDocument` trong `AppletContext` một applet có thể chỉ cho trình duyệt URL để hiển thị trong cửa sổ trình duyệt. (Trong trường hợp này bộ JDK với trình `AppletViewer` bỏ qua phương thức này, nó không thể hiển thị tài liệu). `showDocument` có 2 dạng :

```
public void showDocument(java.net.URL url)
public void showDocument(java.net.URL url, String targetWindow)
```

Dạng một tham số của `showDocument` chỉ cho trình duyệt biết địa chỉ URL của tài liệu cần hiển thị, không chỉ rõ cửa sổ để hiển thị.

Dạng hai tham số của `showDocument` chỉ cho bạn biết vị trí cửa sổ hay frame HTML để hiển thị tài liệu trong đó. Tham số thứ hai có thể có các giá trị sau:

`"_blank"`: Hiển thị tài liệu trong một cửa sổ mới, không có tên.

`"windowName"`: Hiển thị tài liệu trong một cửa sổ với tên là `windowName`. Cửa sổ này được tạo ra nếu cần thiết.

`"_self"`: Hiển thị tài liệu trong cửa sổ và frame chứa applet.

`"_parent"`: Hiển thị tài liệu trong cửa sổ applet nhưng trong frame cha của frame applet. Nếu frame applet không có frame cha thì giống như `"_self"`.

`"_top"`: Hiển thị tài liệu trong cửa sổ của applet ở frame mức đỉnh. Nếu frame của applet là mức đỉnh rồi thì giống như `"_self"`.

Ví dụ:

```
...//In an Applet subclass:
    urlWindow = new URLWindow(getAppletContext());
. . .

class URLWindow extends Frame {
    . . .
    public URLWindow(AppletContext appletContext){
    . . .
        this.appletContext = appletContext;
    . . .
    }
    . . .
    public boolean action(Event event, Object o) {
    . . .
        String urlString = /* user-entered string */;
        URL url = null;
        try {
            url = new URL(urlString);
        } catch (MalformedURLException e) {
            ...//Inform the user and return...
        }
        if (url != null) {
            if (/* user doesn't want to specify the window */) {
                appletContext.showDocument(url);
            } else {
                appletContext.showDocument(url,
                    /* user-specified window */);
            }
        }
    }
    . . .
}
```

III.4. Gửi thông điệp tới các applet khác

Các applet có thể tìm ra các applet khác và gửi thông điệp đến chúng với những giới hạn về bảo mật như sau:

- Nhiều trình duyệt đòi hỏi các applet khởi đầu từ một server.
- Nhiều trình duyệt đòi hỏi các applet phải cùng thư mục trên cùng một server (cùng code base)
- Java API đòi hỏi các applet phải chạy trên cùng một trang, trong cùng cửa sổ trình duyệt.

Chú ý: Một vài trình duyệt cho phép applet gọi các phương thức của một applet khác thậm chí trên các trang khác trong cùng trình duyệt miễn là cùng code base (cùng thư mục). Phương thức quan hệ giữa các applet này không được Java API hỗ trợ nên nhiều trình duyệt không cho phép applet gọi các applet khác trên các trang khác nhau.

Một applet có thể tìm một applet khác bằng cách tìm theo tên (sử dụng phương thức `getApplet` của `AppletContext`) hay tất cả các applet khác trên cùng trang (bằng cách sử dụng phương thức `getApplets` của `AppletContext`). Cả hai phương thức nếu thành công sẽ cho applet gọi một hay nhiều đối tượng `Applet`. Một khi applet gọi tìm được một đối tượng `Applet` thì nó có thể gọi các phương thức trên đối tượng đó.

III.5. Tìm một applet bằng tên: sử dụng phương thức `getApplet`

Phương thức `getApplet` tìm kiếm tất cả các applet khác trên trang hiện hành để thấy một trong số các applet đó bằng một tên đặc biệt. Nếu tìm thấy, `getApplet` sẽ trả về một đối tượng `Applet`.

Mặc định thì một applet không có tên. Để đặt cho applet một cái tên, phải đặt trong đoạn mã của trang HTML mà applet đan xen vào. Bạn có thể chỉ rõ tên của một applet bằng hai cách:

- Bằng cách chỉ rõ một thuộc tính `NAME` trong thẻ `<APPLET>`. Thí dụ như:

```
<APPLET CODEBASE=example/ CODE=Sender.class
      WIDTH=450
      HEIGHT=200
      NAME="buddy">
  . . .
</applet>
```

- Bằng cách chỉ rõ tham số `NAME` với thẻ `<PARAM>`. Thí dụ như:

```
<APPLET CODEBASE=example/ CODE=Receiver.class
      WIDTH=450
      HEIGHT=35>
  <PARAM NAME="name" value="old pal">
  . . .
</applet>
```

Dưới đây là hai applet minh họa cho việc tìm bằng tên. Applet đầu tiên là `Sender` tìm applet thứ hai là `Receiver`, `Sender` gửi một thông điệp tới `Receiver` bằng cách gọi các phương thức của `Receiver`. `Receiver` đáp ứng lại "Received message from *sender-name*!"

Đây là chương trình đầy đủ của Sender:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Enumeration;

public class Sender extends Applet
    implements ActionListener {
    private String myName;
    private TextField nameField;
    private TextArea status;
    private String newline;

    public void init() {
        GridBagLayout gridBag = new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();

        setLayout(gridBag);

        Label receiverLabel = new Label("Receiver name:",
                                         Label.RIGHT);
        gridBag.setConstraints(receiverLabel, c);
        add(receiverLabel);

        nameField = new TextField(getParameter("RECEIVERNAME"),
                                   10);
        c.fill = GridBagConstraints.HORIZONTAL;
        gridBag.setConstraints(nameField, c);
        add(nameField);
        nameField.addActionListener(this);

        Button button = new Button("Send message");
        c.gridwidth = GridBagConstraints.REMAINDER; //end row
        c.anchor = GridBagConstraints.WEST; //stick to the
                                         //text field
        c.fill = GridBagConstraints.NONE; //keep the button
                                         //small
        gridBag.setConstraints(button, c);
        add(button);
        button.addActionListener(this);

        status = new TextArea(5, 60);
        status.setEditable(false);
        c.anchor = GridBagConstraints.CENTER; //reset to the default
```



```
c.fill = GridBagConstraints.BOTH; //make this big
c.weightx = 1.0;
c.weighty = 1.0;
gridBag.setConstraints(status, c);
add(status);

myName = getParameter("NAME");
Label senderLabel = new Label("My name is " + myName + "."),
                               Label.CENTER);

c.weightx = 0.0;
c.weighty = 0.0;
gridBag.setConstraints(senderLabel, c);
add(senderLabel);

newline = System.getProperty("line.separator");
}

public void actionPerformed(ActionEvent event) {
    Applet receiver = null;
    String receiverName = nameField.getText(); //Get name to
                                                //search for.
    receiver = getAppletContext().getApplet(receiverName);
    if (receiver != null) {
        //Use the instanceof operator to make sure the applet
        //we found is a Receiver object.
        if (!(receiver instanceof Receiver)) {
            status.append("Found applet named "
                          + receiverName + ", "
                          + "but it's not a Receiver object."
                          + newline);
        } else {
            status.append("Found applet named "
                          + receiverName + newline
                          + "  Sending message to it."
                          + newline);
            //Cast the receiver to be a Receiver object
            //(instead of just an Applet object) so that the
            //compiler will let us call a Receiver method.
            ((Receiver)receiver).processRequestFrom(myName);
        }
    } else {
        status.append("Couldn't find any applet named "
                      + receiverName + "." + newline);
    }
}
```

```
}

public Insets getInsets() {
    return new Insets(3,3,3,3);
}

public void paint(Graphics g) {
    g.drawRect(0, 0,
               getSize().width - 1, getSize().height - 1);
}

public String getAppletInfo() {
    return "Sender by Kathy Walrath";
}
}
```

Dưới đây là chương trình Reciever:

```
import java.applet.*;
import java.awt.*;

public class Receiver extends Applet {
    private final String waitingMessage="Waiting for a message...";
    private Label label = new Label(waitingMessage, Label.RIGHT);

    public void init() {
        add(label);
        add(new Button("Clear"));
        add(new Label("My name is " + getParameter("name") + "."),
              Label.LEFT));
        validate();
    }

    public boolean action(Event event, Object o) {
        label.setText(waitingMessage);
        repaint();
        return false;
    }

    public void processRequestFrom(String senderName) {
        label.setText("Received message from " + senderName + "!");
        repaint();
    }
}
```

```
public void paint(Graphics g) {
    g.drawRect(0, 0, size().width - 1, size().height - 1);
}

public String getAppletInfo() {
    return "Receiver (named " + getParameter("name") + ") by Kathy
Walrath";
}
}
```

III.6. Tìm tắt cả các applet trên một trang: sử dụng phương thức getApplets

Phương thức `getApplets` trả về một danh sách của tất cả các applet trên cùng trang. Với lý do bảo mật, nhiều trình duyệt và applet viewer thực hiện `getApplets` sao cho nó chỉ trả về những applet bắt nguồn từ cùng một máy chủ.

Dưới đây là chương trình:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Enumeration;
public class GetApplets extends Applet
    implements ActionListener {
    private TextArea textArea;
    private String newline;
    public void init() {
        Button b = new Button("Click to call getApplets()");
        b.addActionListener(this);

        setLayout(new BorderLayout());
        add("North", b);

        textArea = new TextArea(5, 40);
        textArea.setEditable(false);
        add("Center", textArea);

        newline = System.getProperty("line.separator");
    }

    public void actionPerformed(ActionEvent event) {
        printApplets();
    }
}
```

```
public String getAppletInfo() {
    return "GetApplets by Kathy Walrath";
}

public void printApplets() {
    //Enumeration will contain all applets on this page
    //(including this one) that we can send messages to.
    Enumeration e = getAppletContext().getApplets();

    textArea.append("Results of getApplets(): " + newline);

    while (e.hasMoreElements()) {
        Applet applet = (Applet)e.nextElement();
        String info = ((Applet)applet).getAppletInfo();
        if (info != null) {
            textArea.append("- " + info + newline);
        } else {
            textArea.append("- "
                + applet.getClass().getName()
                + newline);
        }
    }
    textArea.append("_____ "
        + newline + newline);
}
}
```

III.7.Đan xen vào các trang Web

Sau khi biên dịch chương trình thành dạng mã ByteCode lúc đó ta bắt đầu tiến hành đan xen vào các trang Web bằng các thẻ và mở và đóng <APPLET>, </APPLET>.

Cũng như những thẻ khác thẻ <APPLET> cũng bao gồm nhiều phần. Tuy nhiên chỉ có một vài phần là cần thiết. Một số Applets nhất thiết phải có đủ các thành phần trong thẻ; một số khác thì chỉ cần những phần tối thiểu.

Đoạn mã sau cho ta hình ảnh đơn giản của các phần chính của một thẻ Applets :

```
<APPLET>
    <APPLET attributes> (Những thuộc tính)
    Applet-parameters (Những tham số )
    Alternate-HTML (HTML thay thế với những trình duyệt không hiểu Java)
</APPLET>
```

III.7.1.Các thuộc tính (Attributes)

Trong giới hạn tối thiểu, tất cả các thẻ `<APPLET>` phải chứa ba thuộc tính trong bảng sau:

Thuộc tính	Mô tả
CODE	Xác định tên tập tin applet
HEIGHT	Xác định chiều cao của applet theo pixel
WIDTH	Xác định chiều rộng của applet theo pixel

Ba thuộc tính trên được gọi là những thuộc tính cần. Tức là ta không thể đưa các Applets và các trang web mà không dùng chúng. Ngoài ra ta có thể sử dụng một số thuộc tính khác như trình bày trong bảng sau:

Thuộc tính	Mô tả
ALIGN	Xác định vị trí đặt applet trên trang để văn bản bao quanh nó.
ALT	Xác định văn bản thay thế
CODEBASE	Xác định URL cơ sở cho applet, mặc định là coi như applet nằm cùng thư mục với trang Web.
HSPACE	Xác định số dấu cách theo chiều ngang bao quanh applet
NAME	Xác định tên của applet. Dùng để liên lạc với các applet khác trong cùng trang
VSPACE	Xác định số dấu cách bao quanh applet theo chiều đứng

III.7.2.Các thông số của applet

Là nơi mà ta có thể tùy biến applet. Để làm cho applet tác động như ý muốn, ta dùng thẻ `<PARAM>` với hai thuộc tính sở hữu riêng là Name và Value. Ta có thể cung cấp nhiều thẻ `<PARAM>` theo quy cách :

`<PARAM NAME= " Parameter name" VALUE= "Parameter Value"> .`

Kiểm tra sự tiến bộ

1. Phương thứcdùng để tạm dừng applet.
2. Phương thứcdùng để cập nhật (update) cửa sổ.
3. Một chương trình Java có thể vừa là applet, vừa là application.

Đúng/Sai

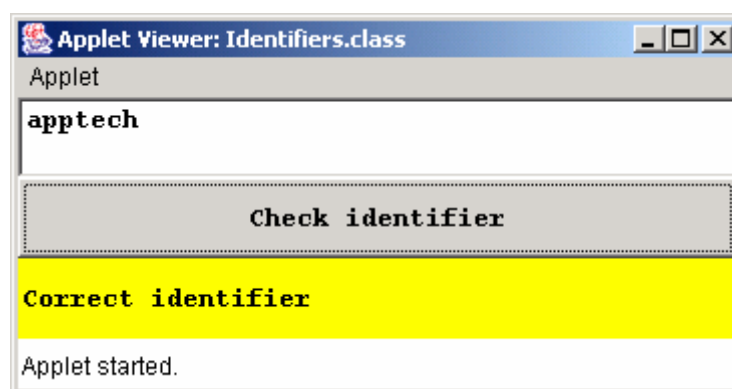
4. Java applet không thể đọc, ghi file trên máy người sử dụng.

Đúng/Sai

5. Phương thứcdùng để tạo tham chiếu đến đối tượng nền đồ hoạ (Graphics).
6. Trong java, điều khiển màu sắc được thực hiện thông qua hai màu cơ bản là trắng và đen. **Đúng/Sai**
7. Phương thứcdùng để lấy tất cả các font chữ mà hệ thống hỗ trợ.
8. Trong lớp FontMetrics, 'ascent' là khoảng cách từ 'baseline' đến đáy của ký tự.
Đúng/Sai

Bài tập

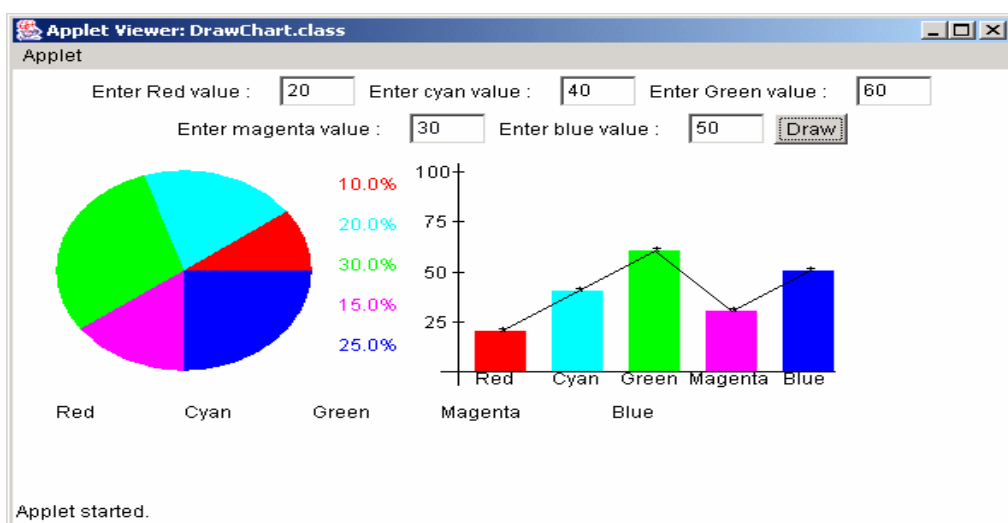
1. Viết applet như sau:



Khi người sử dụng nhập vào ô văn bản và click chuột vào nút 'Check identifier' applet phải kiểm tra xem từ có trong ô văn bản có phải là một từ khoá có trong java không.

Ghi chú: Sử dụng các font chữ khác nhau cho ô văn bản, nút lệnh và nhãn.

2. Viết applet như sau:



Người sử dụng được phép nhập vào giá trị màu trong các ô tương ứng. Khi click nút 'Draw' các biểu đồ dạng đường, bar, pie được hiện ra.

CHƯƠNG 4:

CÁC GÓI & GIAO DIỆN

I. Giới thiệu

Gói và giao diện là hai thành phần chính của chương trình Java. Các gói được lưu trữ theo kiểu phân cấp, và được nhập (import) một cách tường minh vào những lớp mới được định nghĩa. Các giao diện có thể được sử dụng để chỉ định một tập các phương thức. Các phương thức này có thể được hiện thực bởi một hay nhiều lớp.

Một tập tin nguồn Java có thể chứa một hoặc tất cả bốn phần sau đây:

- Một câu lệnh khai báo gói (package).
- Những câu lệnh nhập thêm các gói hoặc các lớp khác vào chương trình (import).
- Một khai báo lớp công cộng (public)
- Một số các lớp dạng riêng tư (private) của gói.

Một tập tin nguồn Java sẽ có khai báo lớp public đơn. Tất cả những phát biểu khác tùy chọn. Chương trình nên được viết theo thứ tự: đặt tên gói (package), lệnh nhập các gói (import), và định nghĩa lớp (class).

II. Các giao diện

Giao diện là một trong những khái niệm quan trọng nhất của ngôn ngữ Java. Nó cho phép một lớp có nhiều lớp cha (superclass). Các chương trình Java có thể thừa kế chỉ một lớp tại một thời điểm, nhưng có thể hiện thực hàng loạt giao diện. Giao diện được sử dụng để thay thế một lớp trừu tượng, không có một sự kế thừa mã thực thi nào. Giao diện tương tự như các lớp trừu tượng. Sự khác nhau ở chỗ một lớp trừu tượng có thể có những hành vi cụ thể, nhưng một giao diện thì không thể có một phương thức cụ thể nào có hành vi của riêng mình. Các giao diện cần được hiện thực. Một lớp trừu tượng có thể được thừa kế, nhưng không thể tạo ra được thể hiện (đối tượng).

II.1. Các bước để tạo một giao diện

- Định nghĩa giao diện: Một giao diện được định nghĩa như sau:

Chương trình 4.1

//Giao diện với các phương thức

public interface myinterface

{

public void add(int x,int y);

public void volume(int x,int y,int z);

}

//Giao diện để định nghĩa các hằng số

public interface myconstants

{

public static final double price=1450.00;

public static final int counter=5;

}

- Chương trình trên được dịch như sau:

javac myinterface.java

- Một giao diện được hiện thực với từ khoá “implement”. Trong trường hợp trên, giao diện cho phép ứng dụng mối quan hệ “is a”. Ví dụ:

class demo implements myinterface

- Nếu nhiều hơn một giao diện được thực thi, các tên sẽ được ngăn cách với nhau bởi một dấu phẩy. Điều này được trình bày như sau:

class Demo implements MyCalc, Mycount

Hãy ghi nhớ các lưu ý sau trong khi tạo một giao diện:

- Tất cả các phương thức trong các giao diện này phải là kiểu public.
- Các phương thức được định nghĩa trong một lớp mà lớp này hiện thực giao diện.

II.2.Hiện thực giao diện

Các giao diện không thể thừa kế (extends) các lớp, nhưng chúng có thể thừa kế các giao diện khác. Nếu khi bạn hiện thực một giao diện mà thừa kế các giao diện khác, bạn định nghĩa đè (**override**) các phương thức trong giao diện mới giao diện đã thừa kế. Trong ví dụ trên, các phương thức chỉ được khai báo, mà không được định nghĩa. Các phương thức phải được định

nghĩa trong một lớp mà lớp đó hiện thực giao diện này. Nói một cách khác, bạn cần chỉ ra hành vi của phương thức. Tất cả các phương thức trong các giao diện phải là kiểu **public**. Bạn không được sử dụng các bộ ngữ (modifiers) chuẩn khác như protected, private,..khi khai báo các phương thức trong giao diện.

Đoạn mã Chương trình 4.2 biểu diễn một giao diện được cài đặt như thế nào:

Chương trình 4.2

```
import java.io.*;

class Demo implements myinterface
{
    public void add(int x,int y)
    {
        System.out.println(" +(x+y));
        //Giả sử phương thức add được khai báo trong giao diện
    }

    public void volume(int x,int y,int z)
    {
        System.out.println(" +(x*y*z));
        //Giả sử phương thức volume được khai báo trong giao diện
    }

    public static void main(String args[])
    {
        Demo d=new Demo();
        d.add(10,20);
        d.volume(10,10,10);
    }
}
```

Khi bạn định nghĩa một giao diện mới, có nghĩa là bạn đang định nghĩa một kiểu dữ liệu tham chiếu mới. Bạn có thể sử dụng các tên giao diện ở bất cứ nơi đâu như bất kỳ kiểu dữ liệu

khác. Chỉ có một thể hiện (instance) của lớp mà lớp đó thực thi giao diện có thể được gán cho một biến tham chiếu. Kiểu của biến tham chiếu đó là tên của giao diện.

III. Các gói

Gói được coi như các thư mục, đó là nơi bạn tổ chức các lớp và các giao diện của bạn. Các chương trình Java được tổ chức như những tập của các gói. Mỗi gói gồm có nhiều lớp, và/hoặc các giao diện được coi như là các thành viên của nó. Đó là một phương án thuận lợi để lưu trữ các nhóm của những lớp có liên quan với nhau dưới một cái tên cụ thể. Khi bạn đang làm việc với một chương trình ứng dụng, bạn tạo ra một số lớp. Các lớp đó cần được tổ chức một cách hợp lý. Điều đó trở nên dễ dàng khi ta tổ chức các tập tin lớp thành các gói khác nhau. Hãy tưởng tượng rằng mỗi gói giống như một thư mục con. Tất cả các điều mà bạn cần làm là đặt các lớp và các giao diện có liên quan với nhau vào các thư mục riêng, với một cái tên phản ánh được mục đích của các lớp.

Nói tóm lại, các gói có ích cho các mục đích sau:

- Chúng cho phép bạn tổ chức các lớp thành các đơn vị nhỏ hơn (như là các thư mục), và làm cho việc xác định vị trí trở nên dễ dàng và sử dụng các tập tin của lớp một cách phù hợp.
- Giúp đỡ để tránh cho việc đặt tên bị xung đột (trùng lặp tên). Khi bạn làm việc với một số các lớp bạn sẽ cảm thấy khó để quyết định đặt tên cho các lớp và các phương thức. Đôi lúc bạn muốn sử dụng tên giống nhau mà tên đó liên quan đến lớp khác. Các gói giấu các lớp để tránh việc đặt tên bị xung đột.
- Các gói cho phép bạn bảo vệ các lớp, dữ liệu và phương thức ở mức rộng hơn trên một nền tảng class-to-class.
- Các tên của gói có thể được sử dụng để nhận dạng các lớp.

Các gói cũng có thể chứa các gói khác.

Để tạo ra một lớp là thành viên của gói, bạn cần bắt đầu mã nguồn của bạn với một khai báo gói, như sau:

package mypackage;

Hãy ghi nhớ các điểm sau trong khi tạo gói:

- Đoạn mã phải bắt đầu với một phát biểu “package”. Điều này nói lên rằng lớp được định nghĩa trong tập tin là một phần của gói xác định.
- Mã nguồn phải nằm trong cùng một thư mục, mà thư mục đó lại là tên gói của bạn.
- Quy ước rằng, các tên gói sẽ bắt đầu bằng một chữ thường để phân biệt giữa lớp và gói.
- Các phát biểu khác có thể xuất hiện sau khai báo gói là các câu lệnh nhập, sau chúng bạn có thể bắt đầu định nghĩa lớp của bạn.
- Tương tự tất cả các tập tin khác, mỗi lớp trong một gói cần được biên dịch.
- Để cho chương trình Java của bạn có khả năng sử dụng các gói đó, hãy nhập (import) chúng vào mã nguồn của bạn.
- Sự khai báo sau đây là hợp lệ và không hợp lệ :

Hợp lệ

```
package mypackage;  
  
import java.io.*;
```

Không hợp lệ

```
import java.io.*;  
  
package mypackage;
```

Bạn có các tùy chọn sau trong khi nhập vào một gói:

- Bạn có thể nhập vào một tập tin cụ thể từ gói:

```
import java.mypackage.calculate
```

- Bạn có thể nhập (import) toàn bộ gói:

```
import java.mypackage.*;
```

Máy ảo Java (JVM) sẽ quản lý các thành phần nằm trong các gói đã được nhập vào (import).

Bạn đã sẵn sàng làm việc với một lệnh nhập import -java.io.*. Bản thân Java đã được cài đặt sẵn một tập các gói, bảng dưới đây đề cập đến một vài gói có sẵn của Java:

Gói	Mô tả
-----	-------

java.lang	Không cần phải khai báo nhập. Gói này luôn được nhập cho bạn.
java.io	Bao gồm các lớp để trợ giúp cho bạn tất cả các thao tác vào ra.
java.applet	Bao gồm các lớp để bạn cần thực thi một applet trong trình duyệt.
java.awt	Các thành phần để xây dựng giao diện đồ họa (GUI).
java.util	Cung cấp nhiều lớp và nhiều giao diện tiện ích khác nhau, như là các cấu trúc dữ liệu, lịch, ngày tháng, v.v..
java.net	Cung cấp các lớp và các giao diện cho việc lập trình mạng TCP/IP.

Bảng 4.1 Các gói trong Java.

Bên cạnh đó, Java còn cung cấp thêm nhiều gói để phát triển ứng dụng và applet. Nếu bạn không khai báo các gói trong đoạn mã của bạn, thì các lớp và các giao diện của bạn sau khi kết thúc sẽ nằm trong một gói mặc định mà không có tên. Thông thường, gói mặc định này chỉ có ý nghĩa cho các ứng dụng nhỏ hoặc các ứng dụng tạm thời. Khi bạn bắt đầu việc phát triển cho một ứng dụng lớn, bạn có khuynh hướng phát triển một số các lớp. Bạn cần tổ chức các lớp đó trong các thư mục khác nhau để dễ dàng truy cập. Để làm được điều này, bạn phải đặt chúng vào các gói.

Ý nghĩa lớn nhất của gói là bạn có khả năng sử dụng các tên lớp giống nhau, nhưng bạn phải đặt chúng vào các gói khác nhau.

III.1.Tạo một gói

Gói là một phương thức hữu dụng để nhóm các lớp mà tránh được các tên trùng nhau. Các lớp với những tên giống nhau có thể đặt vào các gói khác nhau. Các lớp được định nghĩa bởi người sử dụng cũng có thể được nhóm lại trong các gói.

Các bước sau đây cho phép tạo nên một gói do người dùng định nghĩa:

- Khai báo gói bằng cách sử dụng cú pháp thích hợp. Đoạn mã phải bắt đầu với khai báo gói. Điều này chỉ ra rằng lớp được định nghĩa trong tập tin là một phần của gói xác định.

package mypackage;

- Sử dụng phát biểu import để nhập các gói chuẩn theo yêu cầu.

import java.util.*;

- Khai báo và định nghĩa các lớp sẽ nằm trong gói đó. Tất cả các thành phần của gói sẽ là public, để có thể được truy cập từ bên ngoài. Máy ảo Java (JVM) quản lý tất cả các phần tử nằm trong gói đó.

```
package mypackage; //khai báo gói

import java.util.*;

public class Calculate //định nghĩa một lớp
{
    int var;

    Calculate(int n)
    {
        ...

        var = n;

        //các phương thức

        //...

        public class Display //định nghĩa một lớp
        {
            ...//Các phương thức
        }
    }
}
```

- Lưu các định nghĩa trên trong một tập tin với phần mở rộng .java, và dịch các lớp được định nghĩa trong gói. Việc dịch có thể thực hiện với tham số “-d”. Chức năng này tạo một thư mục trùng với tên gói, và đặt tập tin .class vào thư mục được chỉ rõ.

javac -d d:\temp Calculate.java

Nếu khai báo gói không có trong chương trình, lớp hoặc giao diện đó sẽ nằm trong gói mặc định mà không có tên. Nói chung, gói mặc định này thì chỉ có nghĩa cho các ứng dụng nhỏ hoặc tạm thời.

Hãy ghi nhớ các điểm sau đây khi bạn khai thác các gói do người dùng định nghĩa trong các chương trình khác:

- Mã nguồn của các chương trình đó phải tồn tại trong cùng một thư mục với gói được định nghĩa bởi người sử dụng.
- Để cho các chương trình Java khác sử dụng được các gói đó, hãy khai báo chúng vào đoạn mã nguồn.
- Để nhập một lớp ta dùng:

import java.mypackage.Calculate;

- Để nhập toàn bộ một gói, ta làm như sau:

import java.mypackage.*;

- Tạo một tham chiếu đến các thành phần của gói. Ta dùng đoạn mã đơn giản sau:

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

```
import mypackage.Calculate;
```

```
class PackageDemo{
```

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

```
Calculate calc = new Calculate();
```

```
}
```

```
}
```

Nếu phát biểu import cho gói đó không được sử dụng, thì khi sử dụng lớp đó phải chỉ ra lớp đó ở gói nào. Cú pháp như sau:

mypackage.Calculate calc = new mypackage.Calculate();

III.2.Thiết lập đường dẫn cho lớp (classpath)

Chương trình dịch và chương trình thông dịch tìm kiếm các lớp trong thư mục hiện hành, và tập tin nén (zip) chứa các lớp của JDK. Điều này có nghĩa các tập tin nén chứa các lớp của JDK và thư mục hiện hành chứa mã nguồn tự động được đặt vào **classpath**. Tuy nhiên, trong một vài trường hợp, bạn cần phải tự thiết lập classpath.

Classpath là một danh sách các thư mục, danh sách này trợ giúp để tìm kiếm các tập tin .class tương ứng. Thông thường, ta không nên thiết lập môi trường classpath lâu dài. Nó chỉ thích hợp khi thiết lập CLASSPATH để chạy chương trình, chỉ thiết lập đường dẫn cho việc thực thi hiện thời.

javac -classpath c:\temp Packagedemo.java

Thứ tự của các mục trong classpath rất quan trọng. Khi bạn thực thi đoạn mã của bạn, máy ảo Java sẽ tìm kiếm các mục trong classpath theo thứ tự các thư mục trong classpath, cho đến khi nó tìm thấy lớp cần tìm.

Ví dụ của một gói

Chương trình 4.3

```
package mypackage;

public class calculate
{
    public double volume(double height, double width, double depth) {
        return (height*width*depth);
    }

    public int add(int x, int y) {
        return (x+y);
    }

    public int divide(int x, int y) {
        return (x/y);
    }
}
```

Để sử dụng gói này, bạn cần phải:

- Nhập lớp được sử dụng.
- Nhập toàn bộ gói.
- Sử dụng các thành phần của gói.

Bạn cần dịch tập tin này. Nó có thể được dịch với tùy chọn `-d`, nhờ đó `javac` nó tạo một thư mục với tên của gói và đặt tập tin `.class` vào thư mục này.

javac -d c:\temp calculate.java

Chương trình biên dịch tạo một thư mục được gọi là “mypackage” trong thư mục temp, và lưu trữ tập tin `calculate.class` vào thư mục này.

Ví dụ sau biểu diễn cách sử dụng một gói:

Chương trình 4.4

```
import java.io.*;

import mypackage.calculate;

class PackageDemo{

    public static void main(String args[]){

        calculate calc = new calculate();

        int sum = calc.add(10,20);

        double vol = calc.volume(10.3f,13.2f,32.32f);

        int div = calc.divide(20,4);

        System.out.println("The addition is: "+sum);

        System.out.println("The Volume is: "+vol);

        System.out.println("The division is: "+sum);

    }

}
```

Nếu bạn sử dụng một lớp từ một gói khác, mà không sử dụng khai báo import cho gói đó, thì khi đó, bạn cần phải sử dụng tên lớp với tên gói.

mypackage.calculate calc = new mypackage.calculate();

IV.Gói và điều khiển truy xuất

Các gói chứa các lớp và các gói con. Các lớp chứa dữ liệu và đoạn mã. Java cung cấp nhiều mức độ truy cập thông qua các lớp, các gói và các chỉ định truy cập. Bảng sau đây sẽ tóm tắt quyền truy cập các thành phần của lớp:

	public	protected	No modifier	private
Cùng lớp	Yes	Yes	Yes	Yes
Cùng gói- lớp thừa kế (Subclass)	Yes	Yes	Yes	No
Cùng gói-không thừa kế (non-	Yes	Yes	Yes	No

Subclass)				
Khác gói-lớp thừa kế (subclass)	Yes	Yes	No	No
Khác gói-không thừa kế (non-Subclass)	Yes	No	No	No

Bảng 4.2: Truy cập đến các thành phần của lớp.

IV.1. Gói java.lang

Mặc định, mỗi chương trình java đều nhập gói java.lang. Vì thế, không cần lệnh nhập gói java.lang này trong chương trình.

Lớp bao bọc (wrapper class)

Các kiểu dữ liệu nguyên thủy thì không phải là các đối tượng. Vì thế, chúng không thể tạo ra hay truy cập bằng phương thức. Để tạo và thao tác kiểu dữ liệu nguyên thủy, ta sử dụng “wrapper class” tương ứng với. Bảng sau liệt kê các lớp trình bao bọc (wrapper). Các phương thức của mỗi lớp này có trong phần phụ lục.

Kiểu dữ liệu	Lớp trình bao bọc
boolean	Boolean
byte	Byte
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

Bảng 4.3: Các lớp trình bao bọc cho các kiểu dữ liệu nguyên thủy.

Ví dụ một vài phương thức của lớp wrapper:

```
Boolean wrapBool = new Boolean("false");
```

```
Integer num1 = new Integer("31");
```

```
Integer num2 = new Integer("3");
```

```
Int sum = num1.intValue()*num2.intValue();
```

//intValue() là một hàm của lớp trình bao bọc Integer.

Chương trình sau đây minh họa cách sử dụng lớp wrapper cho kiểu dữ liệu int

Chương trình 4.5

```
class CmdArg
{
    public static void main(String args[])
    {
        int sum = 0;
        for(int i = 0; i < args.length; i++)
            sum += Integer.parseInt(args[i]);
        System.out.println("Tổng là: " + sum);
    }
}
```

Vòng lặp for được sử dụng để tìm tổng của các số được truyền vào từ dòng lệnh. Các số đó được lưu trữ trong mảng String args[]. Thuộc tính “length” xác định số các phần tử trong mảng args[]. Mảng args[] là kiểu String. Vì thế, các phần tử phải được đổi sang kiểu dữ liệu int trước khi cộng chúng. Quá trình chuyển đổi được thực hiện với sự giúp đỡ của lớp trình bao bọc “Integer”. Phương thức “parseInt()” trong lớp “Integer” thực hiện quá trình chuyển đổi của kiểu dữ liệu chuỗi sang kiểu dữ liệu số nguyên.

Tất cả các lớp trình bao bọc, ngoại trừ lớp “Character” có một phương thức tĩnh “valueOf()” nhận một chuỗi, và trả về một giá trị số nguyên được. Các lớp bao bọc của byte, int, long, và short cung cấp các hằng số MIN_VALUE và MAX_VALUE. Các lớp bao bọc của double và long cũng cung cấp các hằng POSITIVE_INFINITY và NEGATIVE_INFINITY.

IV.1.1.Lớp String (lớp chuỗi)

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

```
String str1 = new String();
```

//str1 chứa một chuỗi rỗng.

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

//str2 chứa “Hello World”

```
char ch[] = {'A','B','C','D','E'};
```

```
String str3 = new String(ch);
```

//str3 chứa “ABCDE”

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

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

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

```
String strFirst, strSecond, strFinal;
```

```
StrFirst = “Charlie”;
```

```
StrSecond = “Chaplin”;
```

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

```
StrFinal = strFirst.concat(strSecond);
```

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

IV.1.2.Chuỗi mặc định (String pool)

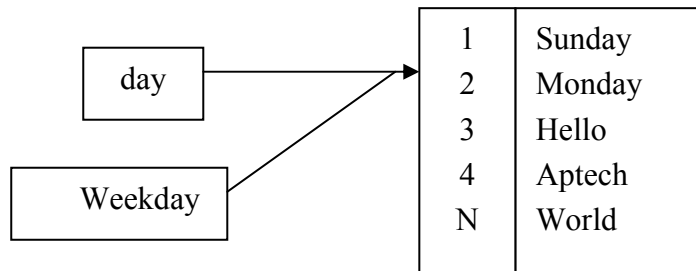
Một chương trình Java có thể chứa nhiều chuỗi. “String Pool” đại diện cho tất cả các chữ được tạo trong chương trình. Mỗi khi một chuỗi được tạo, String Pool tìm kiếm trong nó, nếu tìm thấy nếu chuỗi đã tồn tại thì không tạo thể hiện mà chỉ gán thể tìm thấy cho chuỗi mới. Việc này tiết kiệm rất nhiều không gian bộ nhớ. Ví dụ:

```
String day = “Monday”;
```

```
String weekday = “Monday”;
```

Ở đây, một thể hiện cho biến “day”, biến đó có giá trị là “Monday”, được tạo trong String Pool. Khi chuỗi bằng chữ “weekday” được tạo, có giá trị giống như của biến “day”, một thể hiện đang tồn tại được gán đến biến “weekday”. Vì cả hai biến “day” và “weekday” cũng

đều nhằm chỉ vào chuỗi giống hệt nhau trong String Pool. Hình ảnh sau minh họa khái niệm của “String Pool”.



Hình 4.1 Khái niệm của String Pool.

IV.1.3.Các phương thức của lớp String

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

➤ *charAt()*

Phương thức này trả về một ký tự tại một vị trí 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.

➤ *startsWith()*

Phương thức này trả về giá trị kiểu logic (Boolean), phụ thuộc vào chuỗi có bắt đầu với một chuỗi con cụ thể nào đó không. Ví dụ:

```
String strname = "Java Language";
```

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

Biến “flag” chứa giá trị true.

➤ *endsWith()*

Phương thức này trả về một giá trị kiểu logic (boolean), phụ thuộc vào chuỗi kết thúc bằng một chuỗi con nào đó không, Ví dụ:

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

Biến “flag” chứa giá trị false.

➤ **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”.

➤ **toCharArray()**

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

```
String text = new String("Hello World");  
char textArray[] = text.toCharArray();
```

➤ **indexOf()**

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

```
String day = new String("Sunday");  
int index1 = day.indexOf('n');  
//chứa 2  
int index2 = day.indexOf('z', 2);  
//chứa -1 nếu “z” không tìm thấy tại vị trí 2.  
int index3 = day.indexOf("Sun");  
//chứa mục 0
```

➤ **toUpperCase()**

Phương thức này trả về chữ hoa của chuỗi.

```
String lower = new String("good morning");  
System.out.println("Uppercase: "+lower.toUpperCase());
```

➤ ***toLowerCase()***

Phương thức này trả về chữ thường của chuỗi.

```
String upper = new String("APTECH");  
System.out.println("Lowercase: "+upper.toLowerCase());
```

➤ ***trim()***

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

```
String space = new String("    Spaces    ");  
System.out.println(space);  
System.out.println(space.trim()); //Sau khi cắt bỏ khoảng trắng
```

➤ ***equals()***

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

```
String name1 = "Aptech", name2 = "APTECH";  
boolean flag = name1.equals(name2);
```

Biến "flag" chứa giá trị false.

IV.1.4.Lớp StringBuffer

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

Chương trình 4.6

```
class StringBufferCons  
{  
    public static void main(String args[])  
    {  
        StringBuffer s1 = new StringBuffer();  
        StringBuffer s2 = new StringBuffer(20);
```

```
StringBuffer s3 = new StringBuffer("StringBuffer");

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

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

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

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

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

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

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

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

$$NewCapacity = OldCapacity * 2 + 2$$

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

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

Chương trình 4.7 minh hoạ dung lượng được tính toán và được đặt như thế nào.

Chương trình 4.7

```
class test{

    public static void main(String args[]){

        StringBuffer s1 = new StringBuffer(5);

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

```
s1.ensureCapacity(8);

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

s1.ensureCapacity(30);

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

}

}
```

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

s1.ensureCapacity(8);

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

s1.ensureCapacity(30);

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

IV.1.5.Các phương thức lớp StringBuffer

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

➤ ***append()***

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

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

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

Giá trị trong s1 bây giờ là “goodevening”.

➤ ***insert()***

Phương thức này có hai tham số. Tham số đầu tiên là vị trí chèn. Tham số thứ hai có thể là một chuỗi, một ký tự (char), một giá trị nguyên (int), hay một giá trị số thực (float) được

chèn vào. Vị trí chèn sẽ lớn hơn hay bằng 0, và nhỏ hơn hay bằng chiều dài của đối tượng StringBuffer. Bất kỳ đối số nào, trừ ký tự hoặc chuỗi, được chuyển sang chuỗi và sau đó mới được chèn vào. Ví dụ:

```
StringBuffer str = new StringBuffer("Java sion");  
str.insert(1, 'b');
```

Biến "str" chứa chuỗi "Jbava sion".

➤ **charAt()**

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

```
StringBuffer str = new StringBuffer("James Gosling");  
char letter = str.charAt(6); //chứa "G"
```

➤ **setCharAt()**

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

```
StringBuffer name = new StringBuffer("Jawa");  
name.setCharAt(2, 'v');
```

Biến "name" chứa "Java".

➤ **setLength()**

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

```
StringBuffer str = new StringBuffer(10);  
str.setLength(str.length() + 10);
```

➤ **getChars()**

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

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

Chỉ số kết thúc: vị trí kết thúc

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

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

Ví dụ:

```
StringBuffer str = new StringBuffer("Leopard");  
char ch[] = new char[10];  
str.getChars(3,6,ch,0);
```

Bây giờ biến "ch" chứa "par"

➤ **reverse()**

Phương thức này đảo ngược nội dung của một đối tượng StringBuffer, và trả về một đối tượng StringBuffer khác. Ví dụ:

```
StringBuffer str = new StringBuffer("devil");  
StringBuffer strrev = str.reverse();
```

Biến "strrev" chứa "lived".

IV.1.5.Lớp java.lang.Math

Lớp này chứa các phương thức tĩnh (static) để thực hiện các thao tác toán học. Chúng được mô tả như sau:

Cú pháp là Math.<tên hàm>

➤ **abs()**

Phương thức này trả về giá trị tuyệt đối của một số. Đối số được truyền đến nó có thể là kiểu int, float, double, hoặc long. Kiểu dữ liệu byte và short được chuyển thành kiểu int nếu chúng được truyền tới như là một đối số. Ví dụ:

```
int num = -1;  
Math.abs(num) //trả về 1.
```

➤ **ceil()**

Phương thức này tìm thấy số nguyên nhỏ nhất lớn hơn hoặc bằng đối số được truyền vào.

➤ **floor()**

Phương thức này trả về số nguyên lớn nhất nhỏ hơn hoặc bằng đối số được truyền vào.

```
System.out.println(Math.ceil(8.02)); //trả về 9.0
```

```
System.out.println(Math.ceil(-1.3)); //trả về -1.0
```

```
System.out.println(Math.ceil(100)); //trả về 100.0
```

```
System.out.println(Math.floor(-5.6)); //trả về -6.0
```

```
System.out.println(Math.floor(201.1)); //trả về 201
```

```
System.out.println(Math.floor(100)); //trả về 100
```

➤ **max()**

Phương thức này tìm giá trị lớn nhất trong hai giá trị được truyền vào. Các đối số được truyền vào có thể là kiểu int, long, double, và float.

➤ **min()**

Phương thức này tìm giá trị nhỏ nhất trong hai giá trị được truyền vào. Các đối số được truyền vào có thể là kiểu int, long, double và float.

➤ **round()**

Phương thức này làm tròn đối số có dấu phẩy động. Ví dụ, câu lệnh `Math.round(34.5)` trả về 35.

➤ **random()**

Phương thức này trả về một số ngẫu nhiên kiểu double giữa 0.0 và 1.0.

➤ **sqrt()**

Phương thức này trả về căn bậc hai của một số. Ví dụ, câu lệnh `Math.sqrt(144)` trả về 12.0.

➤ **sin()**

Phương thức này trả về sine của một số, nếu góc được truyền đến bằng radian. Ví dụ: **Math.sin(Math.PI/2)** trả về 1.0, giá trị của $\sin 45^\circ$.

PI/2 radian = 90 độ. Giá trị của “PI” được định nghĩa trong lớp Math (Math.PI).

➤ **cos()**

Phương thức này trả về cosine của một góc tính bằng radian.

➤ **tan()**

Phương thức này trả về tan của một góc tính bằng radian.

IV.1.6.Lớp Runtime (Thời gian thực hiện chương trình)

Lớp Runtime chứa thông tin về môi trường thực thi. Lớp này được sử dụng cho việc quản lý bộ nhớ, và việc thực thi của các quá trình xử lý khác. Mỗi chương trình Java có một thể hiện của lớp này, để cho phép ứng dụng giao tiếp với môi trường. Nó không thể được khởi tạo, một ứng dụng không thể tạo ra một thể hiện của thuộc lớp này. Tuy nhiên, chúng ta có thể tham chiếu thể hiện trong lúc thực hiện chương trình từ việc dùng phương thức `getRuntime()`.

Bây giờ, chúng ta biết rằng việc thu gom các dữ liệu không thích hợp trong Java là một tiến trình tự động, và chạy một cách định kỳ. Để kích hoạt một cách thủ công bộ thu thập dữ liệu không còn được sử dụng ta gọi phương thức `gc()` trên đối tượng Runtime hiện thời. Để xem chi tiết việc cấp phát bộ nhớ, sử dụng các phương thức `totalMemory()` và `freeMemory()`.

```
Runtime r = Runtime.getRuntime();
```

```
.....
```

```
.....
```

```
long freemem = r.freeMemory();
```

```
long totalmem = r.totalMemory();
```

```
r.gc();
```

Bảng sau trình bày một vài phương thức của lớp này:

Phương thức	Ý nghĩa
<code>exit(int)</code>	Dừng việc thực thi, và trả về giá trị của chương trình cho hệ điều hành. Nếu thoát bình thường thì trả về 0; giá trị khác 0 cho thoát không bình thường.
<code>freeMemory()</code>	Trả về kích thước bộ nhớ chưa sử dụng tính bằng byte
<code>getRuntime()</code>	Trả về thể hiện Runtime
<code>gc()</code>	Gọi bộ phận thu thập rác.
<code>totalMemory()</code>	Trả về kích thước bộ nhớ tính bằng byte.

Exec(String)	Chạy chương trình ở môi trường bên ngoài
--------------	--

Bảng 4.4 Lớp Runtime

Chương trình 4.7

```
class RuntimeDemo
{
    public static void main(String args[])
    {
        Runtime r = Runtime.getRuntime();
        Process p = null;
        try {
            p = r.exec("calc.exe");
        }
        catch(Exception e)
        {
            System.out.println("Error executing calculator");
        }
    }
}
```

Bạn có thể tham chiếu đến Runtime hiện hành thông qua phương thức Runtime.getRuntime().

Sau đó, bạn có thể chạy chương trình calc.exe và tham chiếu đến calc.exe trong đối tượng Process.

IV.1.7.Lớp System

Lớp System cung cấp các tiện ích như là, dòng vào, dòng ra chuẩn và dòng lỗi. Nó cũng cung cấp phương thức để truy cập các thuộc tính liên quan đến hệ thống Runtime của Java, và các thuộc tính môi trường khác nhau như là, phiên bản (version), đường dẫn, hay các dịch vụ, v.v..Các trường của lớp này là **in**, **out**, và **err**, các trường này tiêu biểu cho dòng vào, ra và lỗi chuẩn tương ứng.

Bảng sau mô tả các phương thức của lớp này:

Phương thức	Mục đích
exit(int)	Dừng việc thực thi, và trả về giá trị của đoạn mã. 0 cho biết có thể thoát ra một cách bình thường.
gc()	Gọi bộ phận thu thập rác.
getProperties()	Trả về thuộc tính của hệ thống thời gian chạy Java.

setProperty()	Thiết lập các thuộc tính hệ thống hiện hành.
currentTimeMillis()	Trả về thời gian hiện tại bằng mili giây (ms), được tính từ lúc 0 giờ ngày 01 tháng 01 năm 1970.
arrayCopy(Object, int, Object, int, int)	Sao chép mảng.

Bảng 4.5 Lớp System.

Lớp System không thể tạo thể hiện (instance) được.

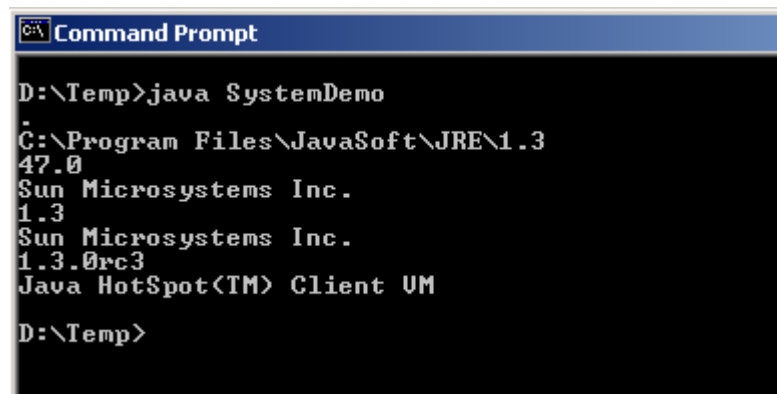
Đoạn mã trong chương trình sau đọc và hiển thị một vài các thuộc tính môi trường Java.

Chương trình 4.9

```
class SystemDemo
{
    public static void main(String args[])
    {
        System.out.println(System.getProperty("java.class.path"));
        System.out.println(System.getProperty("java.home"));
        System.out.println(System.getProperty("java.class.version"));
        System.out.println(System.getProperty("java.specification.vendor"));
        System.out.println(System.getProperty("java.specification.version"));
        System.out.println(System.getProperty("java.vendor"));
        System.out.println(System.getProperty("java.vendor.url"));
        System.out.println(System.getProperty("java.version"));
        System.out.println(System.getProperty("java.vm.name"));
    }
}
```

Mỗi thuộc tính cần in ra cần được cung cấp như một tham số (dạng chuỗi) đến phương thức System.getProperty(). Phương thức này sẽ trả về thông tin tương ứng và phương thức System.out.println() in ra màn hình.

Kết quả chương trình trên như sau:



```
Command Prompt
D:\Temp>java SystemDemo
C:\Program Files\JavaSoft\JRE\1.3
47.0
Sun Microsystems Inc.
1.3
Sun Microsystems Inc.
1.3.0rc3
Java HotSpot(TM) Client VM
D:\Temp>
```

Hình 4.2 Ví dụ về lớp System

IV.1.8.Lớp Class

Các thể hiện của lớp này chứa trạng thái thời gian thực hiện của một đối tượng trong ứng dụng Java đang chạy. Điều này cho phép chúng ta truy cập thông tin về đối tượng trong thời gian chạy.

Chúng ta có thể lấy một đối tượng của lớp này, hoặc một thể hiện bằng một trong ba cách sau:

Sử dụng phương thức getClass() của đối tượng.

- Sử dụng phương thức tĩnh forName() của lớp để lấy một thể hiện của lớp thông qua tên của lớp đó.
- Sử dụng một đối tượng ClassLoader để nạp một lớp mới.

Lớp Class không có phương thức xây dựng (constructor).

Các chương trình sau minh họa cách sử dụng phương thức của một lớp để truy cập thông tin của lớp đó:

Chương trình 4.10

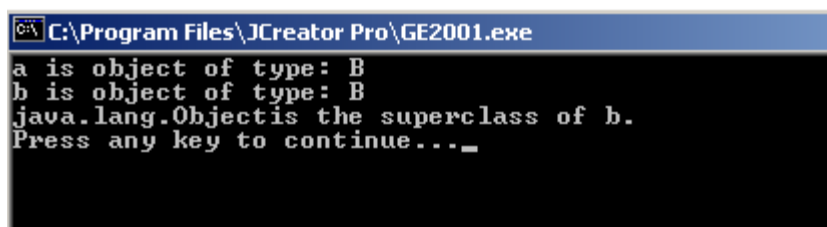
interface A

```
{
    final int id = 1;
    final String name = "Diana";
}
```

```
class B implements A
{
    int deptno;
}

class ClassDemo
{
    public static void main(String args[])
    {
        A a = new B();
        B b = new B();
        Class x;
        x = a.getClass();
        System.out.println("a is object of type: "+x.getName());
        x= b.getClass();
        System.out.println("b is object of type: "+x.getName());
        x=x.getSuperclass();
        System.out.println(x.getName()+ "is the superclass of b.");
    }
}
```

Kết quả chạy chương trình được mô tả như hình dưới đây:



Hình 4.3 Quá trình xuất ra các kết quả của lớp Class.

IV.1.9.Lớp Object

Lớp Object là một lớp cha của tất cả các lớp. Dù là một lớp do người dùng định nghĩa không thừa kế lại bất kỳ một lớp nào khác, theo mặc định nó thừa kế lớp Object.

Một vài các phương thức của lớp Object được biểu diễn bên dưới:

Phương thức	Mục đích
<code>equals(Object)</code>	So sánh đối tượng hiện tại với đối tượng khác.
<code>finalize()</code>	Phương thức cuối cùng. Thông thường bị định nghĩa đè ở lớp con.
<code>notify()</code>	Thông báo cho Thread (luồng) mà hiện thời trong trạng thái đang chờ.
<code>notifyAll()</code>	Thông báo tất cả các Thread (luồng) hiện hành trong trạng thái chờ.
<code>toString()</code>	Trả về một chuỗi đại diện cho đối tượng.
<code>wait()</code>	Đưa Thread (luồng) vào trạng thái chờ.

Bảng 4.6 Lớp Object.

Trong chương trình sau, chúng ta không khai báo bất kỳ lớp hoặc gói nào. Bây giờ, chúng ta có thể tạo bằng cách sử dụng phương thức `equals()`. Bởi vì, theo mặc định lớp `ObjectDemo` mở rộng lớp `Object`.

Chương trình 4.11

Class ObjectDemo

```
{  
  
    public static void main(String args[])  
    {  
  
        if (args[0].equals("Aptech"))  
  
            System.out.println("Yes, Aptech is the right choice!");  
  
    }  
}
```

IV.2. Gói java.util

Gói `Java.util` cung cấp một số lớp tiện ích Java, thường xuyên trong tất cả các loại chương trình ứng dụng. Nó bao gồm một số lớp sau:

➤ `Hashtable`

- Random
- Vector
- StringTokenizer

IV.2.1.Lớp Hashtable (bảng băm)

Lớp Hashtable mở rộng lớp trừu tượng Dictionary, lớp này cũng được định nghĩa trong gói java.util. *Hashtable* được sử dụng để ánh xạ khoá (key) đến giá trị (value). Ví dụ, nó có thể được sử dụng để ánh xạ các tên đến tuổi, những người lập trình đến những dự án, chức danh công việc đến lương, và cứ như vậy.

Hashtable mở rộng kích thước khi các phần tử được thêm vào. Khi một Hashtable mới, bạn có thể chỉ định dung lượng ban đầu và yếu tố nạp (load factor). Điều này sẽ làm cho *hashtable* tăng kích thước lên, bất cứ lúc nào việc thêm vào một phần tử mới làm vượt qua giới hạn hiện hành của *Hashtable*. Giới hạn của Hashtable là dung lượng nhân lên bởi các yếu tố được nạp. Ví dụ: một bảng băm với dung lượng 100, và một yếu tố nạp là 0.75 sẽ có một giới hạn là 75 phần tử. Các phương thức xây dựng cho bảng băm được biểu diễn trong bảng sau:

Constructor	Purpose
Hashtable(int)	Xây dựng một bảng mới với dung lượng ban đầu được chỉ định.
Hashtable(int, float)	Xây dựng một lớp mới với dung lượng ban đầu được chỉ định và yếu tố nạp.
Hashtable()	Xây dựng một lớp mới bằng cách sử dụng giá trị mặc định cho dung lượng ban đầu và yếu tố nạp.

Bảng 4.7 Các phương thức xây dựng Hashtable.

Hashtable hash1 = new Hashtable(500,0.80);

Trong trường hợp này, Bảng băm “hash1” sẽ lưu trữ 500 phần tử. Khi bảng băm lưu trữ vừa đầy 80% (một yếu tố nạp vào của 0.80), kích thước tối đa của nó sẽ được tăng lên.

Mỗi phần tử trong một hashtable bao gồm một khoá và một giá trị. Các phần tử được thêm vào bảng băm bằng cách sử dụng phương thức put(), và được truy lục bằng cách sử dụng phương thức get(). Các phần tử có thể được xoá từ một bảng băm với phương thức remove(). Các phương thức contains() và containsKey() có thể được sử dụng để tra cứu một giá trị hoặc một khoá trong bảng băm. Một vài phương thức của Hashtable được tóm tắt trong bảng sau:

Phương thức	Mục đích
clear()	Xoá tất cả các phần tử từ bảng băm.
clone()	Tạo một bản sao của Hashtable.
contains(Object)	Trả về True nếu bảng băm chứa các đối tượng được chỉ định.
containsKey(Object)	Trả về True nếu bảng băm chứa khoá được chỉ định.
elements()	Trả về một tập hợp phần tử của bảng băm.
get(Object key)	Trả về đối tượng có khoá được chỉ định.
isEmpty()	Trả về true nếu bảng băm rỗng.
keys()	Trả về tập hợp các khoá trong bảng băm.
put(Object, Object)	Thêm một phần tử mới vào bảng băm (Object, Object) là khoá và giá trị.
rehash()	Thay đổi bảng băm thành một bảng băm lớn hơn.
remove(Object key)	Xoá một đối tượng được cho bởi khoá được chỉ định.
size()	Trả về số phần tử trong bảng băm.
toString()	Trả về đại diện chuỗi được định dạng cho bảng băm.

Bảng 4.8 Các phương thức lớp Hashtable.

Chương trình sau sử dụng lớp Hashtable. Trong chương trình này, tên của các tập ảnh là các khoá, và các năm là các giá trị.

“contains” được sử dụng để tra cứu phần tử nguyên 1969, để thấy có danh sách chứa bất kỳ các tập ảnh từ 1969.

“containsKey” được sử dụng để tìm kiếm cho khoá “Animals”, để tìm tập ảnh đó trong bảng băm.

Phương thức “get()” được sử dụng để tìm tập ảnh “Wish You Were Here” có trong bảng băm không. Phương thức get() trả về phần tử cùng với khoá (tên và năm).

Chương trình 4.12

```
import java.util.*;

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

```
//tạo một bảng băm mới
Hashtable ht = new Hashtable();

//thêm các tập ảnh tốt nhất của Pink Floyd
ht.put("Pulse", new Integer(1995));
ht.put("Dark Side of the Moon", new Integer(1973));
ht.put("Wish You Were Here", new Integer(1975));
ht.put("Animals", new Integer(1997));
ht.put("Ummagumma", new Integer(1969));

//Hiển thị bảng băm
System.out.println("Initailly: "+ht.toString());

//kiểm tra cho bất kỳ tập ảnh nào từ 1969
if(ht.contains(new Integer(1969)))
System.out.println("An album from 1969 exists");

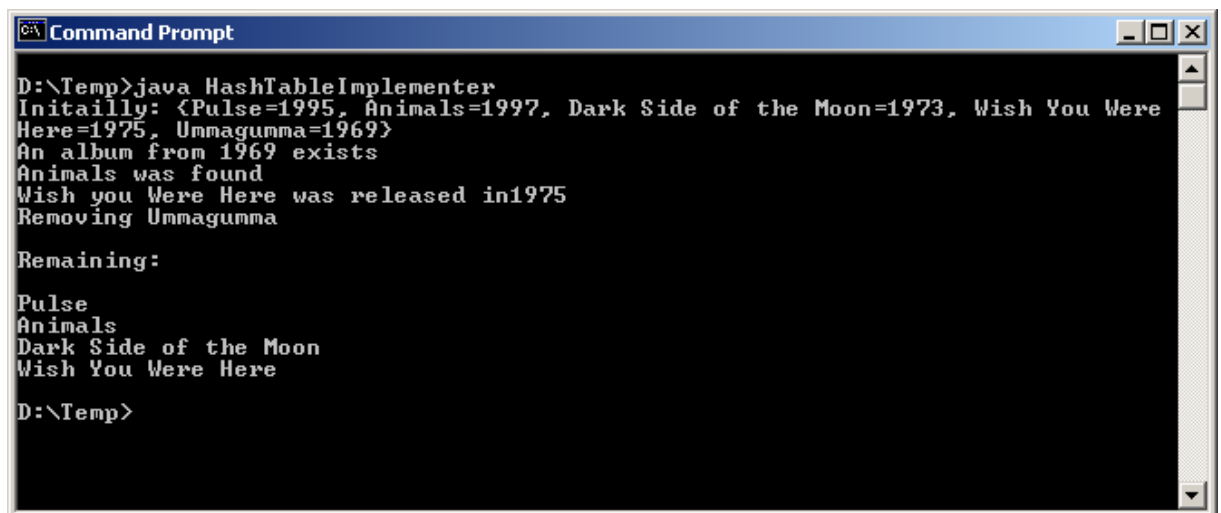
//kiểm tra cho tập ảnh các con thú
if(ht.containsKey("Animals"))
    System.out.println("Animals was found");

//Tìm ra
Integer year = (Integer)ht.get("Wish You Were Here");
System.out.println("Wish you Were Here was released in "+year.toString());

//Xoá một tập ảnh
System.out.println("Removing Ummagumma\r\n");
ht.remove("Ummagumma");

//Duyệt qua tất cả các khoá trong bảng.
System.out.println("Remaining:\r\n");
for(Enumeration enum = ht.keys(); enum.hasMoreElements();)
    System.out.println((String)enum.nextElement());
}
}
```

Kết quả sẽ chạy chương trình như sau:



```
Command Prompt
D:\Temp>java HashTableImplementer
Initailly: <Pulse=1995, Animals=1997, Dark Side of the Moon=1973, Wish You Were
Here=1975, Ummagumma=1969>
An album from 1969 exists
Animals was found
Wish you Were Here was released in1975
Removing Ummagumma

Remaining:

Pulse
Animals
Dark Side of the Moon
Wish You Were Here

D:\Temp>
```

Hình 4.4 Kết quả của HashTableImplementer

IV.2.2.Lớp random

Lớp này là một bộ tạo số giả ngẫu nhiên (pseudo-random). Có hai phương thức xây dựng được định nghĩa. Một trong những phương thức xây dựng này lấy giá trị khởi đầu (seed) như một tham số. Phương thức xây dựng khác không có tham số, và sử dụng thời gian hiện tại như một giá trị khởi đầu. Việc xây dựng một bộ tạo số ngẫu nhiên với một giá trị khởi đầu là một ý tưởng hay, trừ khi bạn muốn bộ tạo số ngẫu nhiên luôn tạo ra một tập các giá trị giống nhau. Mặt khác, thỉnh thoảng nó rất hữu ích để tạo ra trình tự giống nhau của các số random. Điều này có ý nghĩa trong việc gỡ rối chương trình. Một khi bộ tạo số ngẫu nhiên được tạo ra, bạn có thể sử dụng bất kỳ các phương thức sau đây để cập một giá trị từ nó:

- nextDouble()
- nextFloat()
- nextGaussian()
- nextInt()
- nextLong()

Các phương thức xây dựng và các phương thức của lớp Random được tóm tắt trong bảng sau:

Phương thức	Mục đích
random()	tạo ra một bộ tạo số ngẫu nhiên mới
random(long)	Tạo ra một bộ tạo số ngẫu nhiên mới dựa trên giá trị khởi tạo được chỉ định.
nextDouble()	Trả về một giá trị kiểu double kế tiếp giữa 0.0 đến 1.0 từ bộ tạo số

	ngẫu nhiên.
nextFloat()	Trả về một giá trị kiểu float kế tiếp giữa 0.0F và 1.0F từ bộ tạo số ngẫu nhiên.
nextGaussian()	Trả về giá trị kiểu double được phân phối Gaussian kế tiếp từ bộ tạo số ngẫu nhiên. Tạo ra các giá trị Gaussian sẽ có một giá trị trung bình của 0, và một độ lệch tiêu chuẩn của 1.0.
nextInt()	Trả về giá trị kiểu Integer kế tiếp từ một bộ tạo số ngẫu nhiên.
nextLong()	Trả về giá trị kiểu long kế tiếp từ một bộ tạo số ngẫu nhiên.
setSeed(long)	Thiết lập giá trị khởi tạo từ bộ tạo số ngẫu nhiên.

Bảng 4.9 Các phương thức lớp Random.

IV.2.3.Lớp Vector

Một trong các vấn đề với một mảng là chúng ta phải biết nó lớn như thế nào khi chúng ta tạo nó. Trong thực tế có nhiều trường hợp không thể xác định kích thước của mảng trước khi tạo nó.

Lớp Vector của Java giải quyết vấn đề này. Nó cung cấp một dạng mảng với kích thước ban đầu, mảng này có thể tăng thêm khi nhiều phần tử được thêm vào. Một lớp Vector lưu trữ các mục là kiểu Object, nó có thể dùng để lưu trữ các thể hiện của bất kỳ lớp Java nào. Một lớp Vector có thể lưu trữ các phần tử khác nhau, các phần tử khác nhau này là thể hiện của các lớp khác nhau.

Tại bất kỳ thời điểm, một lớp Vector có dung lượng để lưu trữ một số lượng nào đó các phần tử. Khi một lớp Vector dùng hết dung lượng của nó, thì dung lượng của nó được gia tăng bởi một số lượng riêng cho Vector đó. Lớp Vector cung cấp ba phương thức xây dựng khác nhau mà có thể chúng ta chỉ định dung lượng khởi tạo, và tăng số lượng của một Vector, khi nó được tạo ra. Các phương thức xây dựng này được tóm tắt trong bảng sau:

Phương thức xây dựng	Mục đích
Vector(int)	Tạo ra một lớp Vector mới với dung lượng ban đầu được chỉ định.
Vector(int, int)	Tạo ra một lớp Vector mới với dung lượng ban đầu được chỉ định, và lượng tăng.
Vector()	Tạo ra một lớp Vector mới với dung lượng khởi tạo mặc định, và lượng tăng mặc định.

Bảng 4.10 các phương thức xây dựng của lớp Vector.

Một phần được thêm vào một lớp Vector bằng cách sử dụng phương thức `addElement()`. Tương tự, một phần tử có thể được thay thế bằng cách sử dụng phương thức `setElementAt()`. Một lớp Vector có thể tìm kiếm bằng cách sử dụng phương thức `contains()`, phương thức này đơn giản chỉ tìm sự xuất hiện của một đối tượng trong Vector. Phương thức `elements()` trả về một tập hợp các đối tượng được lưu trữ trong lớp Vector. Các phương thức này và các phương thức thành viên khác của lớp Vector được tóm tắt trong bảng dưới đây:

Phương thức	Mục đích
<code>addElement(Object)</code>	Thêm phần tử được chỉ định vào lớp Vector.
<code>capacity()</code>	Trả về dung lượng hiện thời của lớp Vector.
<code>clone()</code>	sao chép lớp vector, nhưng không phải là các phần tử của nó.
<code>contains(Object)</code>	Trả về True nếu lớp Vector chứa đối tượng được chỉ định.
<code>copyInto(Object [])</code>	Sao chép các phần tử của lớp Vector vào mảng được chỉ định.
<code>elementAt(int)</code>	Lấy phần tử vị trí được chỉ định.
<code>elements()</code>	Trả về một bảng liệt kê của các phần tử trong lớp Vector.
<code>ensureCapacity(int)</code>	Đảm bảo rằng lớp Vector có thể lưu trữ ít nhất dung lượng tối thiểu được chỉ định.
<code>firstElement()</code>	Trả về phần tử đầu tiên trong lớp Vector.
<code>indexOf(Object)</code>	Tìm kiếm lớp Vector, và trả về chỉ mục đầu tiên tìm thấy đối tượng.
<code>indexOf(Object, int)</code>	Tìm kiếm lớp Vector bắt đầu từ vị trí chỉ định, trả về vị trí đầu tiên tìm thấy.
<code>insertElementAt(Object, int)</code>	Chèn đối tượng được chỉ định tại vị trí được chỉ định.
<code>isEmpty()</code>	Trả về True nếu lớp Vector không có phần tử.
<code>lastElement()</code>	Trả về phần tử cuối cùng trong lớp Vector.
<code>lastIndexOf(Object)</code>	Tìm kiếm lớp Vector, và trả về chỉ mục của đối tượng tìm thấy cuối cùng.
<code>lastIndexOf(Object, int)</code>	Tìm kiếm lớp Vector bắt đầu tại số chỉ mục được chỉ định, và trả về chỉ mục của phần tử cuối cùng tìm thấy.
<code>removeAllElements()</code>	Xoá tất cả các phần tử từ lớp Vector.
<code>removeElement(Object)</code>	Xoá đối tượng được chỉ định từ lớp Vector.
<code>removeElementAt(int)</code>	Xoá đối tượng tại chỉ mục được chỉ định.
<code>setElementAt(Object, int)</code>	Thay thế đối tượng tại chỉ mục được chỉ định với đối tượng được chỉ định.
<code>setSize(int)</code>	Thiết lập kích thước của lớp Vector thành kích thước mới được chỉ định.
<code>Size()</code>	Trả về số của các phần tử hiện thời trong lớp Vector.
<code>toString()</code>	Trả về một chuỗi chứa nội dung của lớp Vector.

trimToSize()	Định lại kích thước của lớp Vector để di chuyển dung lượng thừa trong nó.
--------------	---

Bảng 4.11 Các phương thức lớp Vector

Chương trình sau tạo ra một lớp Vector vect. Nó chứa 6 phần tử: “Numbers In Words”, “One”, “Two”, “Three”, “Four”, “Five”. Phương thức removeElement() được sử dụng để xóa các phần tử từ vect.

Chương trình 4.13

```
import java.util.*;

public class VectorImplementation
{
    public static void main(String args[])
    {
        Vector vect = new Vector();
        vect.addElement("One");
        vect.addElement("Two");
        vect.addElement("Three");
        vect.addElement("Four");
        vect.addElement("Five");
        vect.insertElementAt("Numbers In Words",0);
        vect.insertElementAt("Four",4);
        System.out.println("Size: "+vect.size());
        System.out.println("Vector ");
        for(int i = 0; i<vect.size(); i++)
        {
            System.out.println(vect.elementAt(i)+" ", "");
        }

        vect.removeElement("Five");
        System.out.println("");
    }
}
```



```
System.out.println("Size: "+vect.size());

System.out.println("Vector ");

for(int i = 0;i<vect.size();i++)

{

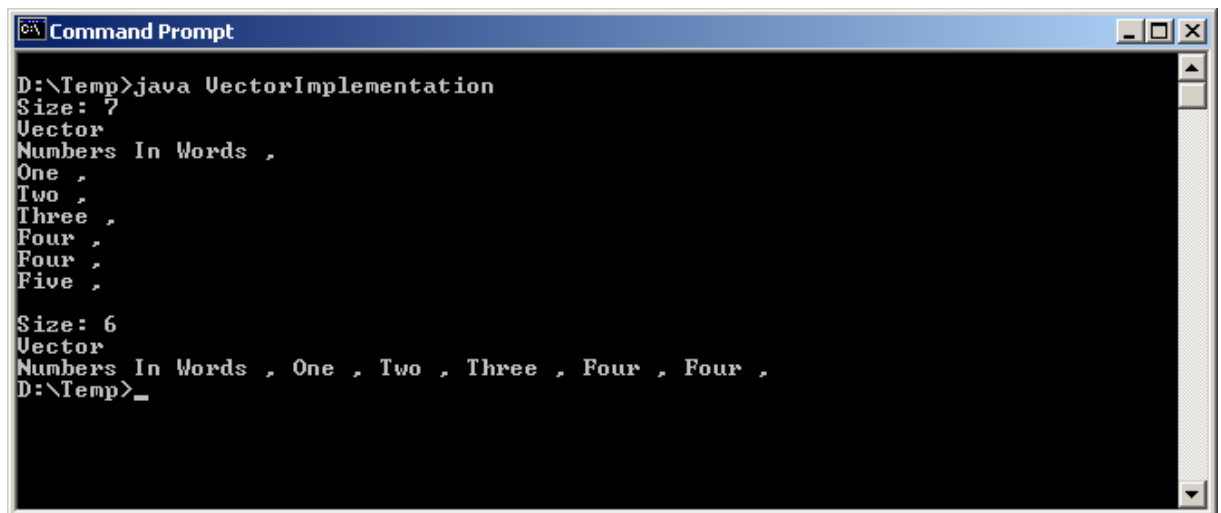
    System.out.print(vect.elementAt(i)+ " , ");

}

}

}
```

Quá trình hiển thị kết quả sẽ được mô tả như hình dưới.



```
Command Prompt
D:\Temp>java VectorImplementation
Size: 7
Vector
Numbers In Words ,
One ,
Two ,
Three ,
Four ,
Four ,
Five ,

Size: 6
Vector
Numbers In Words , One , Two , Three , Four , Four ,
D:\Temp>
```

Hình 4.5 Kết quả của chương trình minh họa lớp Vector.

IV.2.4.Lớp StringTokenizer

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

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

Phương thức xây dựng	Ý nghĩa
StringTokenizer(String)	Tạo ra một đối tượng StringTokenizer mới dựa trên chuỗi được chỉ định.

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

Bảng 4.12 Các phương thức xây dựng của lớp StringTokenizer.

Các phương thức xây dựng ở trên được sử dụng trong các ví dụ sau:

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

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

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

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

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

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

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

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

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

Bảng 4.13 Các phương thức lớp StringTokenizer.

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

Chương trình 4.13

```
import java.util.*;

public class StringTokenizerImplementer
{
    public static void main(String args[])
    {
        // đặt một biểu thức toán học và tạo một tokenizer cho chuỗi đó.
        String mathExpr = "4*3+2/4";

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

        //trong khi vẫn còn các token, hiển thị System.out.println("Tokens of mathExpr: ");
        while(st1.hasMoreTokens())

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

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

        String commas = "field1,field2,field3,and field4";

        StringTokenizer st2 = new StringTokenizer(commas, ",", false);

        //trong khi vẫn còn token, hiển thị.

        System.out.println("Comma-delimited tokens : ");

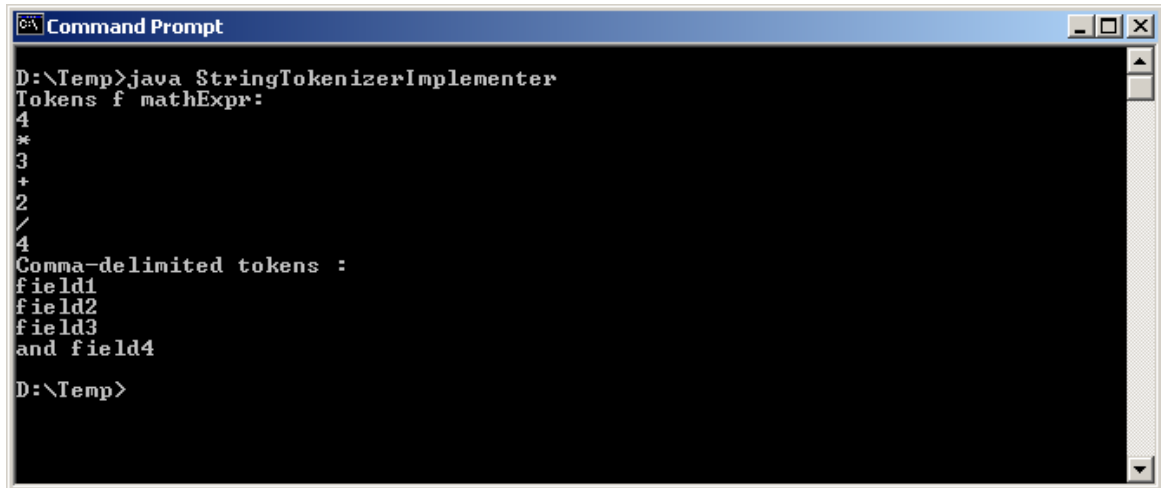
        while (st2.hasMoreTokens())
```

```
System.out.println(st2.nextToken());
```

```
}
```

```
}
```

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



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

Hình 4.6 Kết quả chạy chương trình minh hoạ lớp StringTokenizer.

Kiểm tra sự tiến bộ

1.luôn là lệnh đầu tiên trước các lệnh: import, class trong chương trình Java.
2. Một giao diện có thể chứa nhiều các phương thức. **Đúng/Sai**
3. Trong khi tạo gói, thì mã nguồn phải nằm trong thư mục có tên như tên gói. **Đúng/Sai**
4.là một danh sách của các thư mục, mà JVM sẽ tìm kiếm các tập tin lớp.
5. Lớp bao bọc (wrapper) cho các kiểu dữ liệu double và long cung cấp hai hằng số làvà.....
6.phương thức được sử dụng để thay thế một ký tự trong lớp StringBuffer bằng một ký tự khác tại vị trí được chỉ định.
7. Phương thức..... của lớp StringTokenizer trả về số token còn lại.

Bài tập

1. Tạo một giao diện và sử dụng nó trong một chương trình của Java để hiển thị bình phương và lũy thừa 3 của một số.
2. Tạo một gói và viết một hàm, hàm đó trả về giai thừa của một đối số được truyền vào trong một chương trình.
3. Viết một chương trình bằng cách sử dụng các hàm của lớp Math để hiển thị bình phương của các số lớn nhất và nhỏ nhất của một tập các số được nhập vào bởi người sử dụng tại dòng lệnh.

4. Hãy tạo ra sổ ghi nhớ của chính bạn, nơi mà những con số được nhập vào như sau:

Joy	34543
Jack	56765
Tina	34567

Chương trình phải làm như sau:

- Kiểm tra xem số 3443 có tồn tại trong sổ ghi nhớ của bạn hay không.
 - Kiểm tra xem mẫu tin của Jack có hiện hữu trong sổ ghi nhớ của bạn hay không.
 - Hiện thị số điện thoại của Tina.
 - Xoá số điện thoại của Joy.
 - Hiện thị các mẫu tin còn lại.
5. Viết một chương trình mà nhập vào một số điện thoại tại dòng lệnh, như một chuỗi có dạng (091) 022-6758080. Chương trình sẽ hiện thị mã quốc gia (091), mã vùng (022), và số điện thoại (6758080) (Sử dụng lớp StringTokenizer).

Phần 3: **Lập trình Socket**

CHƯƠNG 1:

Lập trình TCP Socket

Kết nối và gửi dữ liệu theo giao thức TCP được dùng rất nhiều trong các ứng dụng do tính an toàn và bảo đảm của nó.

Dưới đây ta sẽ xây dựng một mô hình ứng dụng khách/chủ (client/server) sử dụng lớp Socket và ServerSocket để kết nối và trao đổi thông tin giữa máy khách và máy chủ. Máy khách (client) sẽ gửi một chuỗi đến máy chủ (server), máy chủ sẽ đảo ngược thứ tự của chuỗi và gửi trả về cho máy khách.

Để mô hình này hoạt động ta phải xây dựng hai chương trình: một chương trình là EchoServer chạy trên máy chủ dùng để lắng nghe kết nối từ phía máy khách và xử lý yêu cầu do máy khách đưa đến, một chương trình khác là EchoClient chạy trên máy khách có nhiệm vụ nhận dữ liệu nhập vào từ bàn phím sau đó gửi đến cho máy chủ xử lý. Khi máy chủ xử lý xong chương trình EchoClient sẽ nhận về và đưa kết quả ra màn hình.

I. Xây dựng chương trình EchoServer

EchoServer chạy trên máy chủ và đón nhận dữ liệu do máy khách gửi đến cổng 8008. Khi nhận được dữ liệu máy chủ sẽ tạo ra đối tượng EchoString để thực hiện việc đảo ngược chuỗi sau đó trả dữ liệu về cho máy khách.

Ví dụ 1.1: EchoServer.java

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

class EchoServer {

    public static void main(String[] args) {
        System.out.println("EchoServer started.");
        try {
            ServerSocket s = new ServerSocket(8008);
            Socket incoming = s.accept();

            System.out.println("Connected to: " + incoming.getInetAddress() +
                               " at port: " + incoming.getLocalPort());

            BufferedReader in
                = new BufferedReader(new InputStreamReader(incoming.getInputStream()));
            PrintWriter out
```

```
        = new PrintWriter(new OutputStreamWriter(incoming.getOutputStream()));

out.println("Hello! This is Java EchoServer. Enter BYE to exit.");
out.flush();

for (;;) {
    String str = in.readLine();
    if (str == null) {
        break;
    } else {
        out.println("Echo: " + str);
        out.flush();
        System.out.println("Received: " + str);

        if (str.trim().equals("BYE"))
            break;
    }
}
incoming.close();
} catch (Exception e) {
    System.out.println("Error: " + e);
}
System.out.println("EchoServer stopped.");
}
}
```

Biên dịch chương trình: *javac EchoServer.java*

rồi chạy trên máy chủ Server sau khi xây dựng xong chương trình EchoClient.

II. Xây dựng chương trình EchoClient

EchoClient chạy trên máy khách nhận dữ liệu nhập vào từ bàn phím và gửi đến máy chủ để xử lý. Cổng kết nối mà chương trình EchoClient sử dụng quy định là 8008. Máy khách sau đó sẽ nhận dữ liệu trả về từ máy chủ và in ra màn hình

Ví dụ 1.2: EchoClient.java

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

class EchoClient {

    public static void main(String[] args) {
        try {
            String host;
            if (args.length > 0 && args[0] != null) {
                host = args[0];
            } else {
                host = "localhost";
            }
        }
    }
}
```

```
}
Socket t = new Socket(host, 8008);
BufferedReader in
    = new BufferedReader(new InputStreamReader(t.getInputStream()));
PrintWriter out
    = new PrintWriter(new OutputStreamWriter(t.getOutputStream()));

for (int i = 1; i <= 10; i++) {
    System.out.println("Sending: line " + i);
    out.println("line " + i);
    out.flush();
}
out.println("BYE");
out.flush();

for (;;) {
    String str = in.readLine();
    if (str == null) {
        break;
    } else {
        System.out.println(str);
    }
}
} catch (Exception e) {
    System.out.println("Error: " + e);
}
}
```

Biên dịch chương trình: *javac EchoClient.java*

Hãy chạy chương trình EchoServer trên một máy và EchoClient trên máy khác nếu hệ thống có mạng. Nếu thử nghiệm trên một máy đơn ta có thể mô phỏng mô hình khách/chủ bằng cách mở hai cửa sổ dòng lệnh DOS. EchoClient sẽ chạy trên một cửa sổ còn EchoServer chạy trên cửa sổ khác, quan sát kết quả trả về để thấy được mô hình khách/chủ làm việc (Chú ý là chạy EchoServer trước rồi mới đến EchoClient).

CHƯƠNG 2:

Lập trình UDP Socket

Trên đây là cách kết nối và truyền dữ liệu trên mạng theo giao thức TCP. Như đã đề cập, mặc dù truyền dữ liệu theo giao thức UDP không chính xác và đảm bảo bằng TCP nhưng bù lại nó không đòi hỏi nhiều tài nguyên của hệ thống, quá trình xử lý và tiếp nhận dữ liệu cũng đơn giản hơn nhiều. Dữ liệu gửi đi theo giao thức UDP thường được đóng thành một gói (data package) bao gồm địa chỉ IP của máy nhận, số cổng cùng với dữ liệu thật sự. UDP được dùng trong các ứng dụng mang tính chất thông báo như về giá cả, thời tiết ...

Dưới đây ta sẽ xây dựng một mô hình ứng dụng khách/chủ (client/server) sử dụng lớp DatagramSocket và DatagramPackage để kết nối và trao đổi thông tin giữa máy khách và máy chủ bằng giao thức UDP.

I. Xây dựng chương trình ExchangeRateServer

ExchangeRateServer chạy trên máy chủ và đón nhận những dữ liệu do máy khách gửi đến cổng 2345. Khi nhận được yêu cầu máy chủ sẽ gửi trả các thông báo về tỉ giá kèm theo ngày giờ mới nhất về cho máy khách. Ta sử dụng hàm random của lớp Math để lấy về tỷ giá mang tính chất ngẫu nhiên của 3 thị trường là Tokyo, NewYork và HongKong. Dữ liệu được nhận và gửi theo từng gói dựa vào lớp DatagramPackage

Chương trình chạy trên máy chủ cung cấp tỷ giá của các thị trường chứng khoán.

Ví dụ 2.1: ExchangeRateServer.java

```
import java.io.*;
import java.net.*;
import java.util.*;
public class ExchangeRateServer {
    public static void main (String args[]) {
        try {
            DatagramSocket socket = new DatagramSocket (2345);
            String localAddress = InetAddress.getLocalHost().getHostName().trim();
            int localPort = socket.getLocalPort();
            System.out.println (localAddress+ ": " );
            System.out.println ("Exchange Rate Server is listening on port " +localPort +".");
            int bufferLength = 256;
            byte outBuffer[];
            byte inBuffer[] = new byte[bufferLength];
```

```
DatagramPackage inDatagram = new DatagramPackage (inBuffer, inBuffer.length);
boolean finished = false;
do {
    socket.receive (inDatagram);
    InetAddress destAddress = inDatagram.getAddress();
    String destHost = destAddress.getHostName().trim();
    int destPort = inDatagram.getPort();
    System.out.println ("\n Received a datagram from "+destHost" at port "+destPort+".");
    if (data.equalsIgnoreCase("quit")) finished = true;
    String s = new Date().toString();
    s=s+"\n NewYork :"+getNewYorkRate();
    s=s+"\n Tokyo :"+getTokyoRate();
    s=s+"\n HongKong :"+getHongKongRate();
    outBuffer = s.getBytes();
    outDatagram = new DatagramPackage (outBuffer, outBuffer.length, destAddress, destPort);
    socket.send (outDatagram);
    System.out.println ("Sent "+s+" to "+destHost+" at port "+destPort+".");
} while (!finished);
} catch (IOException ex){
    System.out.println ("IOException occurred.");
}
}

private static String getNewYorkRate () {
    return Double.toString (Math.random() * 135);
}

private static String getTokyoRate () {
    return Double.toString (Math.random() * 135);
}

private static String getHongKongRate () {
    return Double.toString (Math.random() * 135);
}
}
```

Biên dịch chương trình: *javac ExchangeRateServer.java*

Chương trình ExchangeRateServer sẽ được sử dụng khi ta thiết kế xong chương trình ExchangeRateTable dùng cho máy khách

II. Xây dựng chương trình ExchangeRateTable

ExchangeRateTable chạy trên máy khách, nó chịu trách nhiệm mỗi giây gửi yêu cầu đến máy chủ để cập nhật thông tin về tỉ giá của thị trường chứng khoán. Để đạt được mục tiêu này chương trình của ta phải xây dựng 3 lớp:

- Lớp ExchangeRateTable

Là lớp chương trình dùng tạo một cửa sổ trình bày tỷ giá của các thị trường chứng khoán trong một vùng căn bản TextArea.

- Lớp ExchangeThread

Đây là một phân tuyến chạy song song với chương trình chịu trách nhiệm mỗi giây sẽ gửi yêu cầu đến máy chủ và lấy số liệu về cập nhật.

- Lớp ExchangeData

Là lớp thực hiện kết nối với máy chủ, lớp này thực sự cài đặt phương thức getRates dùng để gửi yêu cầu và nhận dữ liệu trả về từ máy chủ.

Ví dụ 2.2: ExchangeRateTable.java

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

public class ExchangeRateTable {
    public static void main (String args[]) {
        Frame myWindow = new Frame("Stock Exchange Application");
        TextArea rateTable = new TextArea("Wait ...");
        Label rateLabel = new Label("Exchange Rate Table");
        rateTable.setBounds(new Rectangle(16,33,240,100));
        rateLabel.setBounds(new Rectangle(16,6,158,21));
        myWindow.setLayout(null);
        myWindow.add(rateTable, null);
        myWindow.add(rateLabel,null);

        myWindow.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent event){
                System.exit(0);
            }
        });

        myWindow.setSize(new Dimension(300,150));
        myWindow.show();

        ExchangeThread exRate = new ExchangeThread(rateTable);
        exRate.start();
    }
}
```

```
class ExchangeThread extends Thread {
    TextArea rateTable;
    ExchangeData rate = new ExchangeData();

    public ExchangeThread(TextArea rateTable){
        this.rateTable = rateTable;
    }
    public void run(){
        while (true)
        {
            String data = rate.getRates();
            rateTable.setText(data);
            delay(1000);
        }
    }
    private void delay(int miliSeconds){
        try
        {
            this.sleep(miliSeconds);
        }
        catch (Exception e)
        {
            System.out.println("Sleep error!");
        }
    }
}

class ExchangeData
{
    DatagramSocket socket;
    InetAddress serverAddress;
    String localhost;
    int bufferSize = 256;
    byte inBuffer[] = new byte[bufferLength];
    byte outBuffer[];
    DatagramPacket outDatagram;
    DatagramPacket inDatagram;

    public ExchangeData(){
        try
        {
            socket = new DatagramSocket();
            inDatagram = new DatagramPacket(inBuffer, inBuffer.length);
        }
    }
}
```

```
serverAddress = InetAddress.getByName("my.testing.server");
}
catch (Exception e)
{
    System.out.println("Connect error!");
}
}
public String getRates(){
    String data="";
    try
    {
        outBuffer = new byte[bufferLength];
        outBuffer = "rate".getBytes();
        outDatagram = new DatagramPacket(outBuffer, outBuffer.length, serverAddress, 2345);

        socket.send(outDatagram);
        socket.receive(inDatagram);

        InetAddress destAddress = inDatagram.getAddress();
        String destHost = destAddress.getHostName().trim();
        int destPort = inDatagram.getPort();

        data = new String(inDatagram.getData());
        data = data.trim();
    }
    catch (IOException ex)
    {
        System.out.println("IOException occurred.");
    }
    return data;
}
}
```

Biên dịch chương trình: *javac ExchangeRateTable.java*

Chạy chương trình trên tương tự khi chạy 2 chương trình EchoServer và EchoClient.

Phần 4: Lập trình trên Internet

CHƯƠNG 1:

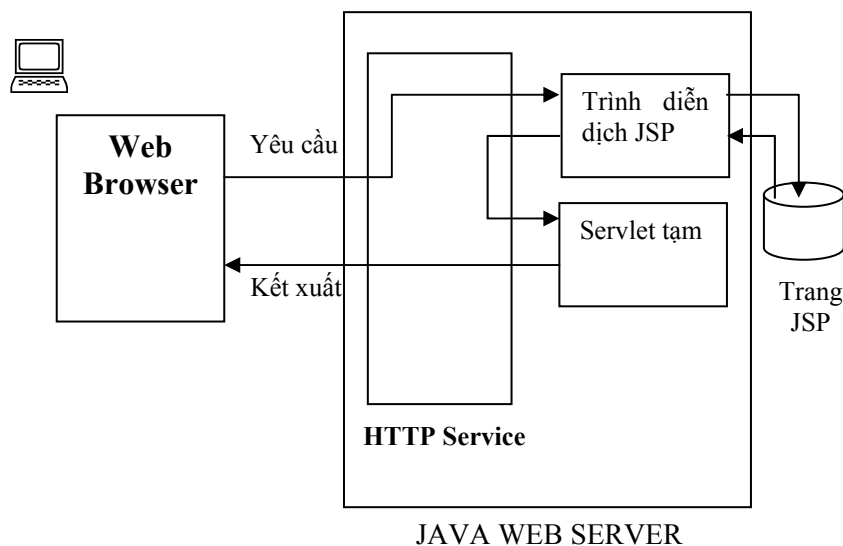
JSP và các khái niệm mở đầu

I. Các cơ chế hoạt động của trang JSP

Ở các chương trước chúng ta đã học về Servlet. Servlet là một cách thay thế cho các chương trình CGI cổ điển bằng ngôn ngữ lập trình và công nghệ Java. Tuy nhiên, servlet ở một mặt nào đó đòi hỏi bạn phải có kiến thức lập trình Java, sử dụng thành thạo trình biên dịch javac để biên dịch servlet ra tập tin .class. Đăng ký servlet với trình chủ Web trước khi sử dụng.

JSP (Java Server Page) là một cách đơn giản hơn nữa để người dùng (nhất là lập trình viên Java không chuyên) tiếp cận được hướng lập trình Web phía máy chủ hiệu quả và nhanh hơn. Nếu bạn chưa biết tường tận Java hay servlet bạn vẫn có thể thiết kế được ứng dụng Web đáp ứng nhu cầu người dùng phía trình khách thông qua các trang JSP.

Servlet đưa mã HTML vào lệnh Java trong khi ngược lại JSP đưa lệnh Java vào các mã (hay thẻ) HTML. Tương tự các trang SHTML chứa thẻ <servlet> được trình chủ thông dịch trước khi trả kết xuất về cho máy khách. Các trang JSP chứa các thẻ đặc biệt quy định gần giống thẻ của ngôn ngữ HTML. Khi bạn yêu cầu một trang JSP, trình chủ sẽ đọc trang JSP từ đĩa cứng, bộ diễn dịch JSP (JSP Page Compiler) sẽ diễn dịch mã lệnh Java chứa trong trang JSP thành một servlet. Sau đó trình chủ Java Web Server sẽ triệu gọi servlet trả kết xuất thuần HTML về cho trình khách. Cơ chế hoạt động của trang JSP được minh họa như hình dưới đây



Cơ chế triệu gọi trang JSP

II. Xây dựng trang JSP

Dưới đây là minh họa cách viết mã lệnh Java trong trang JSP

Ví dụ 1-1: FirstPage.jsp

```
<HTML>
```

```
<H1>
```

```
    Hello JSP!
```

```
</H1> <P>
```

```
<%
```

```
    out.println("Bay gio la: "+new java.util.Date()+"<br>");
```

```
    for(int i=6;i>0;i--){
```

```
        out.println("<FONT Size="+i+">Hello</FONT>");
```

```
    }
```

```
%>
```

```
</HTML>
```

Chúng ta có thể sử dụng trình soạn thảo Notepad của Windows, hoặc EditPlus, UltraEdit ... để tạo ra file FirstPage.jsp ở trên.

Để chạy được file FirstPage.jsp chúng ta cần đến một trình chủ Web server hiệu và diễn dịch được JSP/Servlet. Hiện có rất nhiều trình chủ có khả năng này như: TomCat, Java Web Server, Jrun, WebLogic ... Tuy nhiên cách cài đặt và cấu hình mỗi trình chủ rất khác biệt, ở đây chúng ta sử dụng trình chủ Jrun 3.1. Đây là trình chủ Web server hỗ trợ rất nhiều công nghệ Java, có cấu hình đơn giản, chạy được hầu hết trên các hệ điều hành Unix, Linux, Windows. Tốc độ thông dịch trang JSP của trình chủ JSP khá nhanh và hiệu quả, giao diện đẹp. Với Jrun, gọi thư mục cài đặt trình chủ là [JRUN_HOME] (thường là thư mục C:/Program Files/Allaire/JRun), chép file FirstPage.jsp vào thư mục [JRUN_HOME]/servers/default/default-app. Sau khi khởi tạo dịch vụ Web của trình chủ Jrun hoạt động trên cổng 8100 (khởi tạo trong quá trình cài đặt JRun), trang FirstPage.jsp được triệu gọi từ trình duyệt phía máy khách theo địa chỉ URL như sau: <http://localhost:8100/FirstPage.jsp>, kết quả sẽ được hiện thị trên trình duyệt.

CHƯƠNG 2:

Các cú pháp cơ bản của JSP

I. Các đối tượng mặc định của JSP

Trình diễn dịch JSP cho phép bạn sử dụng một số đối tượng đã khai báo trước. Điều này sẽ giúp bạn viết mã lệnh trong trang JSP nhanh hơn servlet.

- Đối tượng **out**: xuất phát từ lớp `PrintWriter`. Bạn có thể sử dụng đối tượng này để định dạng kết xuất gửi về máy khách. Ví dụ:

```
<% out.println("Result "+7*3); %>
```

- Đối tượng **request**: xuất phát từ lớp `HttpServletRequest`, đối tượng này giúp bạn lấy về các tham số hay dữ liệu do trình khách chuyển lên.
- Đối tượng **response**: tương tự đối tượng **out**, đối tượng **response** dùng để đưa kết xuất trả về trình khách. Tuy nhiên đối tượng **out** được dùng thường xuyên hơn
- Đối tượng **session**: xuất phát từ lớp `HttpSession`, sử dụng đối tượng **session** để theo dõi kết nối và lưu vết một phiên làm việc giữa trình khách và trình chủ.

II. Các thẻ lệnh JSP

II.1. Thẻ bọc mã <% %>

Ưu điểm của JSP là khả năng nhúng mã Java giữa các thẻ định dạng HTML, tương tự như thẻ HTML thẻ lệnh JSP cũng bao gồm thẻ mở và thẻ đóng, mỗi thẻ có các thuộc tính quy định cách sử dụng thẻ đặc trưng. Trong ví dụ 1-1 ở trên mã Java được đặt trong cặp dấu <% %>.

Mặt khác bạn có thể sử dụng các lệnh điều khiển `if..else` của mã Java để quy định kết xuất HTML thích hợp theo cách sau:

Ví dụ 2-1:

```
<HTML>
```

```
<%
```

```
    java.util.Calendar curTime = new java.util.GregorianCalendar();
```

```
    if(curTime.get(curTime.HOUR_OF_DAY)<12){
```

```
%>
```

```
        <b> Morning </b>
```

```
<%}
```



```
else{  
%>  
    <b> Afternoon </b>  
%> }  
else {  
%>  
    <b> Evening </b>  
%> } %>  
</HTML>
```

II.2.Thể hiển thị kết xuất <%= %>

Thay vì sử dụng cú pháp <% %> để diễn đạt một khối gồm nhiều lệnh bạn có thể sử dụng cú pháp <%= %> chỉ để hiển thị kết xuất của một giá trị biến hay hàm nào đó.

Ví dụ 2-2:

```
<html>  
    Welcome<%=username%>  
    You have <%=getNewMail()%> mail  
    Go to <a href=getMail.jsp> Inbox </a>  
</html>
```

Trong ví dụ trên username là biến chứa tên người dùng đăng nhập (login) vào Website của chúng ta, getNewMail() là một hàm trả về số int cho biết số mail hiện có trong hộp thư của người dùng.

Chú ý: không có dấu (;) ở cuối các biến hoặc biểu thức gọi hàm trong cú pháp <%= %>. Bởi vì nội dung của biểu thức nằm trong <%= %> sẽ được chuyển thành lệnh kết xuất out.println() tương đương như sau:

```
out.println("<html>");  
out.println("Welcome "+username);  
out.println("You have "+getNewMail()+"mail.");  
out.println("Go to <a href=getMail.jsp> Inbox </a>");  
out.println("</html>");
```

Cú pháp này được thường xuyên sử dụng đối với các lập trình viên JSP bởi chúng ngắn gọn và kết hợp với các thẻ HTML hiệu quả hơn các lệnh kết xuất `out.println()` trong cú pháp `<% %>`.

II.3. Thẻ chỉ dẫn biên dịch trang `<%@ page %>`

Thẻ `<%@page %>` chỉ dẫn một số tính chất biên dịch áp dụng cho toàn trang jsp. Bạn có thể sử dụng thẻ này để khai báo các thư viện import của java, chỉ định tùy chọn trang jsp có cần giữ trên cache bộ nhớ của trình chủ để tăng tốc hay không ... ví dụ để khai báo sử dụng các thư viện java

Ví dụ 2-3:

```
<HTML>

<%@page import="java.sql.*"%>

<%

    Connection con;

    Statement stmt;

    ResultSet rs;

    try{

        ...

    }

%>

</HTML>
```

Hay ví dụ dưới đây sẽ chỉ thị chuyển đến trang `error.html` nếu trang jsp hiện tại gặp lỗi trong quá trình thực thi.

```
<%@page errorPage="error.html"%>
```

Nếu muốn trang jsp tắt điều khiển session bạn có thể thực hiện chỉ dẫn

```
<%@page session="false"%>
```

II.4. Chèn chú thích vào mã trang JSP

Cũng như Java, JSP cho phép bạn dùng cú pháp `//` để chú thích một dòng mã lệnh trong khi cú pháp `/* */` áp dụng cho nhiều dòng. Các dòng chú thích sẽ được bỏ qua khi trình chủ diễn dịch trang JSP.

Ví dụ 2-4:

```
<HTML>
```

```
<%  
    // lấy biến dữ liệu mang tên user từ đối tượng session  
    String username=session.getAttribute("user");  
    %>  
    Welcome <%= username%>  
    <%  
        /*  
            Tạo đối tượng Date  
            Hiển thị đối tượng Date cho biết ngày giờ cập nhật của trang Web  
        */  
        java.util.Date date= new java.util.Date();  
        %>  
        This page last update on <%=date%>  
</HTML>
```

JSP còn cung cấp thêm cho bạn cú pháp chú thích `<%-- --%>`. Tất cả các khối lệnh Java và HTML nằm giữa hai dấu chú thích này sẽ được trình biên dịch trang bỏ qua không quan tâm đến. Ví dụ:

```
<%--  
    out.println("You will never see this line");  
--%>
```

Dấu chú thích này rất hiệu quả. Nó giúp bạn tạm thời cô lập hoặc che bỏ tác dụng của một đoạn mã Java nào đó đang bị lỗi trong trang JSP. Chúng ta chỉ tạm thời làm mất tác dụng của chúng chứ không cần xóa bỏ. Ví dụ:

```
<HTML>  
    <%--  
        <%  
            String username=session.getAttribute("user");  
            %>  
            Welcome <%= username%>  
        --%>
```

```
<%  
    java.util.Date date= new java.util.Date();  
%>  
  
This page last update on <%=date%>  
  
</HTML>
```

II.5.Khai báo phương thức và biến hằng <%! %>

Cú pháp này cho phép bạn định nghĩa một hoặc nhiều phương thức và biến. Phương thức và biến sau đó có thể được triệu gọi bất kỳ nơi đâu trong trang JSP.

Ví dụ 2-5: nethodCall.jsp

```
<HTML>  
  
    This is line 1 <br>  
  
<% out.println("This is line 2 <br>");  
  
<%= getNextLine("This is line 3 <br>")%>  
  
<%! public String getNextLine(String line)  
    {  
        line="Method called <br>"+line;  
        return line;  
    }  
%>  
  
%>  
  
</HTML>
```

III.Truy xuất cơ sở dữ liệu trong trang JSP

Thực sự trang JSP không khác gì servlet. Bạn có thể dễ dàng dùng trình JDBC để truy xuất cơ sở dữ liệu của các hệ như Access, SQL server ... Dưới đây là ví dụ truy vấn bằng dữ liệu Employee trong cơ sở dữ liệu Access MyData.mdb (thông qua ODBC)

Ví dụ 3-1: EmployeeList.jsp

```
<HTML>  
  
<H1> Employee List Query</H1><br>  
  
<%@ page import="java.sql.*"%>
```

<%!

```
private ResultSet query(String driverName, String connectionURL, String query)
{
    Connection con = null;
    Statement stmt = null;
    ResultSet rs = null;
    try {
        Class.forName(driverName).newInstance();
        con = DriverManager.getConnection(connectionURL);
        stmt = con.createStatement();
        rs = stmt.executeQuery(query);
        return rs;
    }
    catch(Exception es){
        out.println("Connection error!");
    }
    return rs;
}
```

%>

<%

```
ResultSet rs = query("sun.jdbc.odbc.JdbcOdbcDriver",
"jdbc:odbc:MyAccessDataSource",
"SELECT Empno, Name, Position FROM Employee");
if (rs==null){
    out.println("<center><table border>");
    out.println("<tr>");
    for (int i=0; i<columnCount;i++){
        out.println("<th>" + rmsd.getColumnLabel(i+1) + "</th>");
    }
    out.println("</tr>");
}
```

```
while (rs.next()){  
    out.println("<tr>");  
    for (int i=0; i<columnCount;i++){  
        out.println("<td>" + rs.getString(i+1) + "</td>");  
    }  
    out.println("</table></center>");  
}  
%>  
</HTML>
```

Chạy thử chương trình, gõ địa chỉ URL: <http://localhost:8100/EmployeeList.jsp>

TÀI LIỆU THAM KHẢO

- [1] Nguyễn Phương Lan, Hoàng Đức Hải - 2001
Java lập trình mạng – Nhà xuất bản Giáo dục
- [2] Nguyễn Phương Lan, Hoàng Đức Hải - 2003
Lập trình ứng dụng Web với JSP/Servlet – Nhà xuất bản Lao động – Xã hội
- [3] Nguyễn Phương Lan - 2003
JAVA (tập1, 2) – Nhà xuất bản Lao động – Xã hội
- [4] Hoàng Ngọc Giao - 1998
Lập trình Java thế nào (tập 1, 2) – Nhà xuất bản Thống kê
- [5] PTS. Lê Khắc Bình, Bùi Xuân Toại - 1995
Mạng máy tính cho mọi người - Nhà xuất bản Văn hóa
- [6] www.java.sun.com
- [7] www.javavietnam.org