

ĐOÀN VĂN BẢN

# Lập trình **Java** nâng cao



NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT

INHA  
UNIVERSITY

DOÀN VĂN BẢN

# LẬP TRÌNH JAVA NÂNG CAO

- Lập trình đa luồng và xử lý đa tiến trình
- Lập trình triệu gọi từ xa RMI và phân tán đối tượng
- Lập trình mạng và mô hình Client/Server
- Lập trình với Swing nâng cao
- Java Bean và công nghệ thành phần
- Phát triển các dịch vụ Servlet và JSP
- Vấn đề bảo mật và anh ninh hệ thống
- Lập trình theo chuẩn quốc tế và vấn đề bản địa hóa phần mềm



NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT  
HÀ NỘI

# MỤC LỤC

---

MỤC LỤC .....	3
LỜI NÓI ĐẦU .....	9
<b>Chương 1. LẬP TRÌNH ĐA LUỒNG .....</b>	13
1.1. Tiến trình, đa nhiệm và đa luồng .....	13
1.1.1. Các luồng trong Java .....	15
1.1.2. Các trạng thái của Thread.....	20
1.2. Các mức ưu tiên của luồng.....	22
1.3. Các nhóm luồng.....	26
1.4. Đồng bộ hóa .....	27
1.4.1. Các hàm đồng bộ.....	28
1.4.2. Sự trao đổi giữa các luồng không đồng bộ.....	30
1.4.3. Đồng bộ việc truy cập vào các tài nguyên chia sẻ .....	33
1.5. Vấn đề tắc nghẽn.....	38
1.6. Đa luồng trong Applet.....	40
<b>Chương 2. LẬP TRÌNH RMI VÀ PHÂN TÁN ĐỐI TƯỢNG .....</b>	52
2.1. Triệu gọi phương thức của đối tượng từ xa RMI .....	52
2.1.1. Triệu gọi phương thức từ xa .....	53
2.1.2. Kiến trúc RMI Java .....	53
2.2. Thiết lập môi trường triệu gọi từ xa .....	56
2.2.1. Trên máy phục vụ (Server).....	56
2.2.2. Trên máy khách (Client).....	64
2.2.3. Triển khai ứng dụng Web.....	67
2.3. Truyền tham số trong các lời gọi phương thức từ xa.....	68
2.3.1. Truyền đối tượng đến trình chủ theo tham trị.....	70
2.3.2. Chuyển đổi tương đến chương trình chủ theo tham chiếu.....	70
2.3.3. Truyền gọi đối tượng từ xa .....	79

<b>2.4. Sử dụng RMI với Applet.....</b>	<b>81</b>
<b>2.5. Ngôn ngữ định nghĩa giao diện Java (IDL) và CORBA .....</b>	<b>86</b>
2.5.1. Ngôn ngữ định nghĩa giao diện IDL.....	86
2.5.2. Phát triển ứng dụng với IDL và CORBA .....	91
<b>2.6. Nhận xét về phương thức lập trình phân tán RMI.....</b>	<b>96</b>
<b>Chương 3. LẬP TRÌNH MẠNG.....</b>	<b>99</b>
<b>3.1. Kết nối với Server .....</b>	<b>99</b>
<b>3.2. Mô hình tính toán Client/Server.....</b>	<b>102</b>
<b>3.3. Phục vụ nhiều chương trình Client .....</b>	<b>106</b>
3.3.1. Xây dựng chương trình ứng dụng độc lập .....	107
3.3.2. Xây dựng ứng dụng nhúng Applet .....	109
<b>3.4. Mô hình Client/Server sử dụng dịch vụ không kết nối .....</b>	<b>118</b>
<b>3.5. Truy cập vào các trang Web .....</b>	<b>121</b>
3.5.1. Xem nội dung của trang Web .....	121
3.5.2. Tìm các tập tin ở Web Server .....	123
<b>3.6. Dịch vụ thư điện tử E-mail.....</b>	<b>125</b>
3.6.1. Định dạng thư điện tử .....	126
3.6.2. Gửi thư điện tử .....	127
3.6.3. Kết nối theo bộ định vị tài nguyên URL để tìm tin .....	131
<b>Chương 4. LẬP TRÌNH VỚI SWING NÂNG CAO .....</b>	<b>138</b>
<b>4.1. Cấu trúc cây Tree.....</b>	<b>138</b>
4.1.1. Cây đơn giản .....	139
4.1.2. Soạn thảo cây và đường đi .....	142
4.1.3. Đánh số các nút trên cây .....	147
4.1.4. Thay đổi biểu diễn các nút của cây.....	148
4.1.5. Hiển thị thông tin của các nút.....	153
<b>4.2. Các bảng dữ liệu.....</b>	<b>157</b>
4.2.1. Bảng đơn giản .....	158
4.2.2. Mô hình bảng TableModel.....	161
4.2.3. Hiển thị các record dữ liệu từ cơ sở dữ liệu .....	164
4.2.4. Sắp xếp các hàng trong bảng.....	170
4.2.5. Soạn thảo bảng .....	177
<b>4.3. Các thanh trượt và các thước đo tiến độ .....</b>	<b>183</b>
4.3.1. Thanh trượt Slider .....	183
4.3.2. Thước tiến độ JProgressBar .....	189

<b>Chương 5. JAVA BEAN.....</b>	196
<b>5.1. Giới thiệu về JavaBean.....</b>	196
<b>5.2. Tại sao phải phát triển JavaBean?.....</b>	197
<b>5.3. Bộ công cụ phát triển Bean BDK.....</b>	197
<b>5.3.1. BeanBox.....</b>	198
<b>5.3.2. Xây dựng chương trình ứng dụng bằng BeanBox .....</b>	199
<b>5.3.3. Xây dựng Applet từ BeanBox .....</b>	200
<b>5.4. Tạo lập mới thành phần Bean .....</b>	200
<b>5.4.1. Kiểm tra các thuộc tính và các sự kiện của Bean .....</b>	201
<b>5.4.2. Phát sinh báo cáo về các thuộc tính .....</b>	201
<b>5.5. Thiết lập các thuộc tính cho Bean.....</b>	202
<b>5.5.1. Các thuộc tính đơn giản Simple Properties.....</b>	202
<b>5.5.2. Các thuộc tính biên Bound Properties .....</b>	203
<b>5.5.3. Các thuộc tính bị khống chế Constrained Properties .....</b>	205
<b>5.5.4. Các thuộc tính chỉ số hóa Indexed Properties.....</b>	206
<b>5.5. Các phần tử lắng nghe các sự kiện trên các thành phần.....</b>	206
<b>5.6. Thiết lập một Bean riêng của bạn .....</b>	207
<b>5.7. Ngữ cảnh của Bean .....</b>	209
<b>Chương 6. PHÁT TRIỂN CÁC DỊCH VỤ SERVLET VÀ JSP.....</b>	219
<b>6.1. Giới thiệu về Servlet.....</b>	219
<b>6.2. Ưu điểm của Servlet .....</b>	221
<b>6.3. Môi trường thực hiện Servlet .....</b>	222
<b>6.3.1. Mô tả Servlet đơn.....</b>	223
<b>6.3.2. Mô tả Servlet gộp .....</b>	223
<b>6.3.3. Mô tả Servlet nhúng .....</b>	223
<b>6.3.4. Cài đặt trình chủ Apache Tomcat .....</b>	224
<b>6.4. Kiến trúc của Servlet .....</b>	225
<b>6.4.1. Giao diện Servlet .....</b>	225
<b>6.4.2. Lớp cơ sở HttpServlet .....</b>	226
<b>6.5. Dịch vụ cài đặt Servlet.....</b>	228
<b>6.5.1. Dịch chương trình Servlet .....</b>	228
<b>6.5.2. Thực hiện Servlet.....</b>	229
<b>6.6. Chu trình sống của các Servlet .....</b>	231
<b>6.6.1. Khởi động Servlet .....</b>	231

6.6.2. Tương tác với các Client.....	232
6.6.3. Hủy bỏ Servlet .....	232
<b>6.7. Xử lý các yêu cầu.....</b>	<b>233</b>
6.7.1. Truy tìm thông tin .....	233
6.7.2. Gửi thông tin.....	234
6.7.3. Xử lý các dữ liệu mâu .....	238
<b>6.8. Các phiên làm việc Session .....</b>	<b>242</b>
<b>6.9. Sự truyền thông giữa các Servlet .....</b>	<b>244</b>
<b>6.10. Servlet kết nối các cơ sở dữ liệu .....</b>	<b>245</b>
6.10.1. Vai trò của Servlet trong mô hình kết nối CSDL .....	245
6.10.2. Xử lý giao dịch với JDBC .....	246
<b>6.11. JSP .....</b>	<b>249</b>
6.11.1. Kiến trúc của JSP.....	249
6.11.2. Các mô hình truy cập của JSP.....	251
6.11.3. Các phạm vi đối tượng.....	252
6.11.4. Cơ sở cú pháp của JSP .....	253
6.11.5. Các hành động của JSP.....	257
<b>CHƯƠNG 7. VẤN ĐỀ BẢO MẬT VÀ AN NINH THÔNG TIN .....</b>	<b>266</b>
<b>7.1. Giới thiệu về vấn đề bảo mật, an toàn hệ thống thông tin .....</b>	<b>266</b>
<b>7.2. Bộ nạp lớp và kiểm tra byte code .....</b>	<b>269</b>
7.2.1. Viết bộ nạp lớp ClassLoader riêng .....	269
7.2.2. Kiểm tra byte code.....	274
<b>7.3. Lớp SecurityManager và Permission.....</b>	<b>277</b>
7.3.1. Vấn đề bảo mật trong Java 2 Platform .....	278
7.3.2. Các tệp chính sách <i>bảo mật</i> .....	280
7.3.3. Bộ quản lý <i>bảo mật</i> tùy biến.....	288
<b>7.4. Vấn đề <i>bảo mật</i> trong gói java.security.....</b>	<b>293</b>
7.4.1. Dấu vết thông điệp .....	293
7.4.2. Chữ ký số .....	298
<b>7.5. Vấn đề chứng thực .....</b>	<b>305</b>
7.5.1. Giấy chứng nhận Certificate X.509 .....	307
7.5.2. Lớp giấy chứng nhận .....	308
7.5.3. Ký giấy chứng nhận.....	310
7.5.4. Ký nhận các tệp JAR .....	317

<b>CHƯƠNG 8. LẬP TRÌNH THEO CHUẨN QUỐC TẾ VÀ BẢN ĐỊA HÓA PHẦN MỀM .....</b>	323
<b>8.1. Văn đề biểu diễn dữ liệu phụ thuộc vào văn hóa         của từng dân tộc.....</b>	323
<b>8.2. Lớp Locale .....</b>	324
<b>8.3. Các dữ liệu số và đơn vị tiền tệ .....</b>	327
<b>8.4. Ngày tháng và thời gian.....</b>	333
<b>8.5. Văn bản Text .....</b>	340
<b>8.5.1. Sắp xếp văn bản.....</b>	340
<b>8.5.2. Ranh giới của các văn bản Text.....</b>	347
<b>8.6. Bọc tài nguyên .....</b>	352
<b>8.6.1. Bản địa hóa tài nguyên.....</b>	352
<b>8.6.2. Đặt tài nguyên vào các bọc .....</b>	353
<b>8.7. Bản địa hóa các giao diện đồ họa .....</b>	363
<b>DANH SÁCH CÁC THUẬT NGỮ ANH - VIỆT VÀ TỪ VIẾT TẮT .....</b>	381
<b>TÀI LIỆU THAM KHẢO .....</b>	386
<b>CHỈ MỤC .....</b>	387

# Lời nói đầu

Một đặc tính rất quan trọng của ngôn ngữ lập trình hướng đối tượng Java là tạo ra môi trường thực hiện độc lập với các nền, nó cho phép bạn sử dụng cùng một phần mềm trên các nền khác nhau như Windows, Solaris, Linux, Macintosh, v.v. Điều này thực sự cần thiết khi rất nhiều chương trình được nạp xuống từ Internet để chạy được ngay trên những nền khác nhau. Nghĩa là, Java hỗ trợ để ta thực hiện được ý tưởng “*Viết chương trình một lần và thực thi được ở mọi nơi*”.

Java là ngôn ngữ hướng đối tượng thực sự, mọi thứ trong chương trình đều là đối tượng, ngoại trừ các kiểu dữ liệu nguyên thuỷ như các kiểu giá trị số. Một chương trình viết bằng Java, đúng theo tư tưởng của cách tiếp cận hướng đối tượng, là một tập các đối tượng trao đổi với nhau bằng cách gửi và nhận thông điệp. Như vậy, Java hỗ trợ để tạo ra những phần mềm có chất lượng cao, đáp ứng các yêu cầu của người sử dụng phân tán trên mạng, có tính mở cao, tùy biến theo khách hàng, an toàn, và tin cậy với nhiều cơ chế phát triển như: lập trình đa luồng, lập trình phân tán đối tượng với phương pháp triệu gọi từ xa (RMI), lập trình mạng, phát triển các dịch vụ Servlet trên các Server, các giải pháp bảo mật thông tin như chữ ký, chứng chỉ số, hay các vấn đề quốc tế hoá, bản địa hoá phần mềm, v.v.

Tiếp theo cuốn “*Lập trình hướng đối tượng với Java*” [1] (tái bản năm 2005), cuốn sách này giới thiệu tiếp phần lập trình hướng đối tượng nâng cao với Java. Nội dung của cuốn sách được trình bày trong tám chương.

Chương I giới thiệu về *cơ chế xử lý đa luồng*, một đặc tính quan trọng của Java. Nó hỗ trợ để tạo ra các chương trình thực thi bởi nhiều luồng đồng thời về mặt logic. Trên các máy tính đơn bộ xử lý như của chúng ta hiện nay, thực chất cơ chế đa luồng thực hiện theo nguyên lý *chia sẻ thời gian*, nhưng trên những máy nhiều bộ xử lý, hay trên mạng các máy tính, chúng có thể thực hiện đồng thời về mặt vật lý, nghĩa là thực hiện song song nhằm tăng hiệu quả của hệ thống máy tính và tăng tốc độ xử lý.

Ngày nay, cộng đồng những người lập trình hướng đối tượng bắt đầu nghĩ rằng “đối tượng có ở mọi nơi”, đặc biệt có nhiều trong mã nguồn mở. Những đối tượng này, tất nhiên được hỗ trợ để trao đổi được với nhau theo những giao thức chuẩn trên mạng. Tận dụng các đối tượng đó chính là ý tưởng của Java-RMI (*triệu gọi phương thức từ xa*), được trình bày ở chương II. *Lập trình phân tán đối tượng bằng cách triệu gọi*

*phương thức từ xa RMI* là cách hợp tác giữa các đối tượng có những mã lệnh cài đặt (bao gồm các phương thức và thuộc tính) nằm trên các máy khác nhau (chính xác là nằm trên các JVM – máy ảo Java khác nhau), có thể triệu gọi lẫn nhau để trao đổi tin nhằm tận dụng những đối tượng với hàng trăm triệu dòng mã lệnh đã được thiết kế và xây dựng sẵn, phân tán ở trên mạng, đang hoạt động rất hiệu quả.

Khi nói tới *lập trình mạng*, ta thường nghĩ đến cách trao đổi giữa một chương trình phục vụ (Server) với một hay nhiều chương trình khách (Client). Chương trình khách gửi một yêu cầu tới cho chương trình phục vụ, và chương trình này xử lý dữ liệu để trả lời cho chương trình khách. Như vậy, khi chương trình khách muốn gửi đi các yêu cầu thì trước hết phải tìm cách kết nối với Server. Server có thể chấp nhận hay từ chối sự kết nối này. Một khi sự kết nối đã được thiết lập thì Client và Server trao đổi với nhau thông qua các Socket. Chương III đề cập đến việc sử dụng các lớp trong gói `java.net` cùng với các phương thức kết nối mạng và trao đổi tin giữa các máy với nhau, chủ yếu theo mô hình Client/Server.

Người lập trình thường xuyên phải hiển thị, tổ chức thông tin và các thành phần của một hệ thống phần mềm dưới dạng cấu trúc cây hay cấu trúc bảng. Chương IV trình bày cách sử dụng các lớp của gói `javax.swing` phát triển để tạo lập các *cấu trúc cây* và *các bảng dữ liệu*. Nhóm xây dựng Swing đã có nhiều cố gắng để tạo ra các cấu trúc điều khiển, cây và bảng giúp người lập trình sử dụng chúng dễ dàng hơn. Gói `javax.swing` hỗ trợ để chúng ta tổ chức, biểu diễn tin dưới dạng đồ họa rất trực quan và tiện lợi, hoàn toàn đáp ứng nguyên lý “thấy và cảm nhận”, cách mà chương trình biểu diễn để người dùng thấy, theo dõi được và cách mà họ có thể tương tác (cảm nhận được), cũng như cách thực hiện với hệ thống giống như chúng ta mong muốn.

*Mô hình phần mềm thành phần* gồm: các thành phần được xem như là hạt nhân; các Container (bộ chứa), nơi các đối tượng được lắp ghép lại để tạo thành chương trình và người lập trình viết thêm những dòng lệnh thực hiện sự tương tác giữa các thành phần theo một kịch bản để hoàn thành nhiệm vụ của bài toán đặt ra. Một Container là nơi các thành phần có thể tự đăng ký và tạo ra các giao diện cho các thành phần khác biết cách tương tác với nhau. *Java Bean* là một *mô hình thành phần hoàn chỉnh*, hỗ trợ các tính năng chung cho kiến trúc thành phần, các thuộc tính, các sự kiện, v.v., được đề cập ở chương trình V.

Chương VI giới thiệu xu hướng rất quan trọng đang được tập trung phát triển ứng dụng hiện nay, đó là cách xây dựng các chương trình dịch vụ Java ở phía máy chủ (Server). *Servlet* là *thành phần chính được sử dụng để phát triển các chương trình dịch vụ Java ở phía máy chủ*. Servlet là sự phát triển mở rộng của CGI để đảm bảo Server thực hiện được các chức năng của mình. Ta có thể sử dụng Servlet của Java để tùy chỉnh lại một dịch vụ bất kỳ, như Web Server, Mail Server, v.v. Phần cuối của

chương VI được dành để giới thiệu khái quát về JSP, một ngôn ngữ không chỉ hỗ trợ để tạo ra những trang Web độc lập với các nền, độc lập với các Server, mà còn là công nghệ Java rất hiệu quả để thể hiện nguyên lý WYSIWYG (*Những gì bạn nhìn thấy là bạn có được chúng*).

*Vấn đề bảo mật và đảm bảo an ninh hệ thống phần mềm* được trình bày ở chương VII. Bảo mật và đảm bảo an toàn hệ thống là vấn đề thời sự đang được nhiều người tập trung nghiên cứu và triển khai ứng dụng. Để đảm bảo an toàn cho hệ thống, ta nên sử dụng cả những bộ nạp lớp ClassLoader chung của hệ thống (mặc định) và những bộ nạp lớp riêng, kết hợp với lớp quản trị an ninh SecurityManager để kiểm soát sự hoạt động của các đoạn mã lệnh. Nói chung, ta nên xây dựng những lớp quản trị an ninh riêng cho từng ứng dụng với những chính sách cấp quyền thực hiện cho phù hợp. Hơn nữa, có thể dễ dàng thực hiện bằng cách sử dụng các thuật toán mật mã được cài đặt trong các lớp ở gói `java.security` để ký nhận và xác thực tệp tin hay chương trình, đặc biệt là các *chữ ký điện tử và các chứng chỉ số*.

Trong thời đại hội nhập kinh tế thế giới hiện nay, sự trao đổi giữa các công ty dù lớn hay nhỏ với nhau, phần lớn đều sử dụng những ngôn ngữ khác nhau. Vấn đề trao đổi thông tin giữa các dân tộc, giữa nhiều cộng đồng trên thế giới với nhau luôn gặp phải những khó khăn, trở ngại do việc qui định cách viết và hiểu về các thông điệp, số liệu hay các đại lượng như đơn vị tiền tệ, thời gian là rất khác nhau. Thực tế cho thấy, có nhiều người trên thế giới có thể đọc và hiểu được tiếng Anh. Song, người sử dụng sẽ cảm thấy thoải mái, tự tin hơn khi sử dụng những Applet hay chương trình ứng dụng hiển thị các thông tin được viết bằng tiếng của dân tộc họ và biểu diễn dữ liệu theo những hình dạng mà họ quen thuộc. Java 2 Platform cung cấp các đặc trưng cơ sở để *thực hiện việc quốc tế hóa*, cho phép tách các mục dữ liệu phụ thuộc vào các nền văn hóa từ những phần mềm và làm cho thích ứng với những nhu cầu của người dùng (vấn đề địa phương hóa hay còn gọi là *bản địa hóa*). Nhiều người lập trình tin rằng, họ cần quốc tế hóa những phần mềm của mình bằng cách sử dụng Unicode và dịch các thông báo sang ngôn ngữ giao diện của người sử dụng. Song, như ở chương VIII chúng ta sẽ thấy, còn nhiều vấn đề liên quan đến việc quốc tế hóa trong lập trình hơn là sự hỗ trợ của Unicode. Ngày tháng, đơn vị thời gian, tiền tệ lưu hành và các số liệu được qui định biểu diễn thường khác nhau theo những vùng khác nhau trên thế giới. Do vậy, trong mỗi chương trình chúng ta cần sử dụng một cách nào đó để định dạng lại các mục trong thực đơn, các nhãn, các thông điệp, các thông báo của hệ thống, v.v., theo những ngôn ngữ, hình dạng khác nhau nhằm đáp ứng tốt hơn những yêu cầu của khách hàng. Chương VIII được dành để giải quyết những vấn đề nêu trên.

Cuốn sách này được biên soạn theo yêu cầu của giáo trình môn học *lập trình nâng cao để giảng dạy*, học tập cho giáo viên và sinh viên, học viên cao học ngành công nghệ

thông tin. Nó có thể được sử dụng như là tài liệu tham khảo cho cán các bộ nghiên cứu và những người tham gia các dự án phát triển phần mềm ứng dụng để giải quyết những bài toán của thực tế đặt ra.

Tác giả xin bày tỏ lòng biết ơn chân thành tới các bạn đồng nghiệp trong Phòng Các hệ thống phần mềm tích hợp, Viện Công nghệ thông tin, Viện Khoa học & Công nghệ Việt Nam. *Xin cảm ơn Nhà xuất bản Khoa học và Kỹ thuật đã hỗ trợ và tạo điều kiện để cuốn sách được xuất bản.*

Mặc dù rất cố gắng, nhưng tài liệu này chắc chắn không tránh khỏi những sai sót. Chúng tôi mong nhận được các ý kiến đóng góp của bạn đọc để có thể chỉnh lý kịp thời.

Thư góp ý xin gửi về: Nhà xuất bản Khoa học và Kỹ thuật 70 Trần Hưng Đạo, Hà Nội.

*Hà Nội tháng 2 năm 2006*

**Tác giả**

# Chương 1

## LẬP TRÌNH ĐA LUỒNG

Chương I giới thiệu:

- ✓ Đa nhiệm, tiến trình và luồng,
- ✓ Xử lý đa luồng trong Java,
- ✓ Mức ưu tiên của luồng,
- ✓ Vấn đề đồng bộ hoá và bài toán tắc nghẽn.

### 1.1. TIẾN TRÌNH, ĐA NHIỆM VÀ ĐA LUỒNG

Chúng ta hầu như đã quen với khái niệm đa nhiệm: ở cùng một thời điểm có nhiều hơn một chương trình thực hiện đồng thời trên cùng một máy tính. Ví dụ, bạn có thể cùng lúc vừa in một tài liệu đồng thời soạn thảo một văn bản khác. Nói chung, có hai kỹ thuật đa nhiệm:

- Đa nhiệm dựa trên các *tiến trình*
- Đa nhiệm dựa trên các *luồng*

Mỗi tác vụ được thực hiện theo một tiến trình, được xem như một chương trình đơn. Ở mức thô, đa nhiệm dựa trên các tiến trình cho phép nhiều tiến trình (nhiều chương trình đơn) thực hiện đồng thời trên một máy tính. Ví dụ, trong khi soạn thảo văn bản (chạy chương trình xử lý văn bản, như Microsoft Word), ta đồng thời chạy chương trình bảng tính điện tử. Ở mức tinh hơn, đa nhiệm dựa trên các luồng cho phép các phần của cùng một chương trình thực thi đồng thời trên cùng một máy tính. Tương tự, một chương trình xử lý văn bản có thể thực hiện đồng thời việc in ra máy in, đồng thời thực hiện tạo khuôn dạng cho một văn bản. Điều này thực hiện được khi hai tác vụ đó được thực thi theo những đường thực hiện độc lập nhau. Hai tác vụ được xác định tương ứng với hai phần (bộ phận) thực hiện đồng thời của một chương trình. Mỗi phần như thế của chương trình định nghĩa một đường thực hiện riêng được gọi là *luồng* (*thực hiện*).

Như vậy, một tiến trình có thể bao gồm nhiều luồng. Các tiến trình có các tập riêng các biến dữ liệu. Các luồng của một tiến trình có thể chia sẻ với nhau về không gian địa chỉ chương trình, các đoạn dữ liệu và môi trường xử lý, đồng thời cũng có vùng dữ liệu riêng để thao tác ([3], [6]).

Để dễ hiểu hơn về mô hình này, chúng ta có thể hình dung *tiến trình như một xí nghiệp và các luồng như là những công nhân làm việc trong xí nghiệp đó*. Các công nhân của xí nghiệp *cùng chia sẻ* với nhau diện tích mặt bằng và các tài nguyên của cả xí nghiệp. Song, mỗi công nhân lại có *chỗ làm việc* được xem như là *chỗ riêng* của họ và những người khác không truy cập được.

*Việc tạo ra một công nhân (tuyên dụng lao động) dễ hơn nhiều việc lập ra một xí nghiệp*, vì muốn có một xí nghiệp thì phải có ít nhất một số công nhân và phải đáp ứng một số tiêu chuẩn nào đó theo qui định.

Tương tự, chúng ta có thể quan sát mối quan hệ giữa tiến trình và luồng về phương diện thông tin. Các công nhân trong xí nghiệp, theo mặc định, được quyền biết về mọi sự thay đổi, mọi việc xảy ra trong xí nghiệp. Nhưng nói chung, những biến động của một xí nghiệp thì bình thường những xí nghiệp khác không biết được, trừ những xí nghiệp được thông báo trực tiếp.

Trong môi trường đơn luồng, ở mỗi thời điểm chỉ cho phép một tác vụ thực thi. Điều này thường dẫn đến lãng phí vì tốc độ xử lý của CPU là rất lớn mà không được sử dụng hết công suất, ví dụ như chương trình phải chờ người sử dụng nhập dữ liệu vào. Kỹ thuật đa nhiệm cho phép tận dụng được những thời gian rỗi của CPU để thực hiện những tác vụ khác.

Có thể tưởng tượng, mỗi luồng như đang thực hiện trong một ngũ cành độc lập, như thế nó sở hữu CPU với thanh ghi, bộ nhớ và mã chương trình riêng biệt. Thực ra, đối với hệ thống chỉ có một CPU, thì mỗi thời điểm chỉ có một luồng thi hành. CPU sẽ nhanh chóng được chuyển đổi giữa các luồng để tạo ra ảo giác là các luồng được thi hành cùng một lúc. Nhưng thực chất chúng được thực hiện theo chế độ *chia sẻ thời gian*. Hệ thống một CPU hỗ trợ tính *đồng thời logic*, chứ không phải tính đồng thời vật lý. Tính đồng thời logic được thực thi khi nhiều luồng thực hiện với những dòng điều khiển độc lập và riêng rẽ. Trong hệ thống nhiều CPU, thực tế là nhiều luồng thực hiện đồng thời cùng một lúc, đạt được tính đồng thời về mặt vật lý. Một đặc tính quan trọng của Java là *xử lý đa luồng*, nghĩa là nó hỗ trợ tính đồng thời logic, dù có hay không tính đồng thời vật lý.

Kỹ thuật đa nhiệm dựa trên các luồng có một số ưu điểm hơn dựa trên các tiến trình:

- Các luồng có thể chia sẻ cùng một không gian địa chỉ
- Việc dịch chuyển ngũ cành thực hiện giữa các luồng yêu cầu chi phí ít hơn
- Sự truyền thông giữa các luồng thường yêu cầu chi phí thấp hơn.

Đa nhiệm có thể thực hiện được theo hai cách:

- Phụ thuộc vào hệ điều hành, nó có thể cho tạm ngừng chương trình mà không cần tham khảo các chương trình đó.
- Các chương trình chỉ bị dừng lại khi chúng tự nguyện nhường điều khiển cho chương trình khác.

Cách thứ nhất còn được gọi là hệ thống *đa nhiệm theo quyền ưu tiên* còn cách thứ hai được gọi là hệ thống *đa nhiệm cộng tác*. Ví dụ, Window 3.1 là hệ thống đa nhiệm cộng tác, Window NT (và Window 95 32 bit) là hệ thống đa nhiệm theo quyền ưu tiên.

Nhiều hệ điều hành hiện nay đã hỗ trợ đa luồng như: SUN Solaris, Window NT, Window 95, OS/2, v.v., Java hỗ trợ đa nhiệm dựa trên các luồng và cung cấp các đặc tính ở mức cao cho lập trình đa luồng.

### 1.1.1. Các luồng trong Java

Như trên đã nêu, *một luồng là một mạch thực hiện trong một chương trình* (một tiến trình) và nó có thể thực hiện riêng biệt. Ở thời điểm thực hiện, các luồng của một chương trình có thể cùng sử dụng chung một không gian bộ nhớ, vì vậy có thể chia sẻ với nhau về dữ liệu và mã lệnh. Chúng cũng có thể cùng chia sẻ với nhau trong một tiến trình để thực thi một chương trình.

Các luồng trong Java được phép thực thi đồng thời mà không cần đồng bộ, nghĩa là nhiều tác vụ khác nhau có thể thực hiện đồng thời. Vì phần lớn các máy tính là đơn bộ xử lý CPU, nên máy ảo Java (JVM) sử dụng cơ chế chia sẻ thời gian và cho phép mỗi luồng có cơ hội thực hiện một khoảng thời gian ngắn sau đó nhường quyền điều khiển cho những luồng khác.

Để tận dụng được những khả năng trong mẫu hình lập trình đa luồng của Java, cần phải hiểu rõ các phương diện sau:

- Tạo lập các luồng,
- Bởi vì các luồng có thể chia sẻ với nhau cùng một không gian bộ nhớ, nên việc đồng bộ hóa để truy nhập vào dữ liệu, mã chung là rất quan trọng,
- Vì các luồng có thể ở những trạng thái khác nhau, do vậy, cần phải hiểu rõ sự chuyển đổi giữa các trạng thái như thế nào.

#### Tạo lập các luồng

Java sử dụng một cơ chế trong đó mỗi luồng có một cơ hội để chạy trong một thời khoảng tương đối nhỏ và ngay sau đó lại kích hoạt luồng khác thực hiện.

Java cung cấp hai giải pháp tạo lập luồng:

1. Thiết lập lớp con của Thread
2. Cài đặt lớp xử lý luồng từ giao diện Runnable.

(i) **Cách thứ nhất:** Trong Java có một lớp được xây dựng sẵn là Thread, lớp cơ sở để xây dựng những lớp mới kế thừa nhằm tạo các luồng thực hiện theo yêu cầu.

Ví dụ, lớp MyClass được mở rộng dựa trên cơ sở kế thừa lớp Thread nhằm tạo ra các luồng thực hiện. Các tiến trình trong hệ thống bắt đầu thực hiện ở một địa chỉ đặc biệt được xác định bởi hàm có tên là main(). Tương tự khi một luồng của lớp MyClass được tạo ra

thì nó gọi hàm run() để thực hiện. Hàm này được viết để để thực thi những công việc yêu cầu trong mỗi luồng được tạo ra.

```
class MyClass extends Thread{
    // Một số thuộc tính
    public void run(){
        // Các lệnh cần thực hiện theo luồng
    }
    // Một số hàm khác được viết đè hay được bổ sung
}
```

Khi chương trình chạy nó sẽ gọi một hàm đặc biệt đã được khai báo trong Thread, đó là start() để bắt đầu một luồng đã được tạo ra.

**Ví dụ 1.1.** Tạo ra hai luồng thực hiện đồng thời để hiển thị các từ trong dãy ("Hôm nay", "báo cáo", "bài tập", "lớn,", "môn học", "lập trình Java.") lên màn hình.

Chương trình này tạo ra hai luồng: thread1 và thread2 từ lớp MyThread. Sau đó nó khởi động cả hai luồng và thực hiện một chu trình lặp do để đợi cho đến khi các luồng kết thúc hoặc “chết”. Hai luồng này hiển thị lần lượt những dòng chữ “Hôm nay”, “báo cáo”, “bài tập”, “lớn,”, “môn học”, “lập trình Java.” sau khi chờ một khoảng thời gian ngắn ngẫu nhiên giữa các lần thực hiện. Bởi vì cả hai luồng cùng hiển thị lên màn hình nên chương trình phải xác định luồng nào có thể được hiển thị thông tin trên màn hình tại những thời điểm khác nhau trong khi chương trình thực hiện.

```
import java.lang.Thread;
import java.lang.System;
import java.lang.InterruptedException;
class ThreadTest1 {
    public static void main(String args[]){
        Mythread thread1 = new Mythread("Thread 1:");
        Mythread thread2 = new Mythread("Thread 2:");
        thread1.start();
        thread2.start();
        boolean thread1IsAlive = true;
        boolean thread2IsAlive = true;
        do{
            if(thread1IsAlive && !thread1.isAlive()){
                thread1IsAlive = false;
                System.out.println("Thread 1 is dead.");
            }
            if(thread2IsAlive && !thread2.isAlive()){
                thread2IsAlive = false;
                System.out.println("Thread 2 is dead.");
            }
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        } while(thread1IsAlive || thread2IsAlive);
    }
}
```

```

        if(thread2IsAlive && !thread2.isAlive())){
            thread2IsAlive = false;
            System.out.println("Thread 2 is dead.");
        }
    } while (thread1IsAlive || thread2IsAlive);
}
}

class Mythread extends Thread {
    static String message[] = {"Hôm nay", "báo cáo", "bài
                                tập", "lớn,", "môn học", "lập trình Java."};
    public Mythread (String id){
        super(id);
    }
    public void run(){
        String name = getName();
        for(int i=0; i < message.length; ++i){
            randomWait();
            System.out.println(name + message[i]);
        }
    }
    void randomWait(){
        try{
            sleep((long)(1000*Math.random()));
        }catch (InterruptedException x){
            System.out.println("Interruped");
        }
    }
}

```

Chương trình trên mỗi khi thực thi sẽ cho kết quả khác nhau. Sở dĩ như vậy là vì chương trình sử dụng bộ đếm số ngẫu nhiên để xác định một luồng sẽ đợi trong thời gian ngẫu nhiên trước khi hiển thị thông báo lên màn hình. Sau đây là kết quả trong một lần chạy thử :

```

Thread 1: Hôm nay
Thread 2: Hôm nay

```

```

Thread 1: báo cáo
Thread 2: báo cáo
Thread 2: bài tập
Thread 1: lớn,
Thread 1: môn học
Thread 2: lớn,
Thread 2: môn học
Thread 1: Lập trình Java.
Thread 1: is dead.
Thread 2: Lập trình Java.
Thread 2: is dead.

```

Kết quả trên cho thấy luồng thread1 thực hiện trước và hiển thị dòng chữ “Hôm nay” lên màn hình, sau đó đợi để được thi hành tiếp trong khi thread2 đang hiển thị dòng chữ “Hôm nay”. Khi đến lượt, luồng thread1 hiển thị tiếp dòng chữ “báo cáo”, rồi lại nhường cho thread2 để hiển thị tiếp dòng chữ “báo cáo” và sau đó là “bài tập”. Luân phiên thực hiện như vậy cho đến khi các luồng kết thúc.

Lớp ThreadTest1 có một hàm main(). Hàm này tạo ra hai đối tượng mới là thread1 và thread2 của lớp Mythread. Sau đó hàm main() khởi động cả hai luồng trên bằng hàm start().

*Lưu ý:* Java không hỗ trợ đa kế thừa. Do vậy, nếu người lập trình muốn tạo ra một lớp kế thừa từ một lớp cơ sở và để thực hiện được theo luồng thì nó cũng đồng thời phải kế thừa từ lớp Thread. Điều này không thực hiện được. Do vậy, ta phải thực hiện theo cách thứ hai.

(ii) *Cách thứ hai:* Java giải quyết hạn chế trên bằng cách xây dựng lớp trên cơ sở cài đặt giao diện luồng Runnable để tạo ra các luồng thực hiện. Người lập trình thiết kế các lớp thực hiện theo luồng bằng cách cài đặt theo giao diện Runnable như sau.

```

class MyClass implements Runnable{
    // Các thuộc tính
    // Nạp chồng hay viết đè một số hàm
    // Viết đè hàm run()
    .
    .
}

```

**Ví dụ 1.2.** Ta cũng có thể tạo một chương trình tương tự chương trình ở ví dụ 1.1, nhưng tạo ra luồng là đối tượng của lớp MyClass cài đặt giao diện Runnable. Các đối tượng của lớp MyClass sẽ được thực hiện dưới dạng luồng bằng cách truyền chúng như các đối số cho hàm tạo dựng của lớp Thread.

```

import java.lang.Thread;
import java.lang.System;
import java.lang.InterruptedException;

```

```
import java.lang.Runnable;
class ThreadTest2 {
    public static void main(String args[]) {
        Thread thread1 = new Thread(new MyClass("Thread 1:"));
        Thread thread2 = new Thread(new MyClass("Thread 2:"))
        thread1.start();
        thread2.start();
        boolean thread1IsAlive = true;
        boolean thread2IsAlive = true;
        do {
            if(thread1IsAlive && ! thread1.isAlive()) {
                thread1IsAlive = false;
                System.out.println("Thread 1 is dead.");
            }
            if(thread2IsAlive && ! thread2.isAlive()) {
                thread2IsAlive = false;
                System.out.println("Thread 2 kết thúc.");
            }
        } while (thread1IsAlive || thread2IsAlive);
    }
}
class MyClass implements Runnable {
    static String message[] = {"Hôm nay", "báo cáo", "bài tập",
        "lớn,", "môn học", "lập trình Java."};
    String name;
    public MyClass (String id) {
        name = id;
    }
    public void run() {
        for(int i=0; i < message.length; ++i) {
            randomWait();
            System.out.println(name+message[i]);
        }
    }
    void randomWait() {
        try{
            Thread.currentThread().sleep((long)(3000*Math.random()));
        }
```

```

        } catch (InterruptedException x) {
            System.out.println("Interruped!");
        }
    }
}

```

Chạy chương trình trên cho ta một trong các kết quả như sau :

```

Thread 1: Hôm nay
Thread 1: báo cáo
Thread 2: Hôm nay
Thread 2: báo cáo
Thread 1: bài tập
Thread 1: lớn,
Thread 2: bài tập
Thread 2: lớn,
Thread 2: môn học
Thread 1: môn học
Thread 1: lập trình Java.
Thread 1: kết thúc.
Thread 2: lập trình Java.
Thread 2: kết thúc.

```

Hàm main() của lớp ThreadTest2 khác với hàm main() của ThreadTest1 ở chỗ tạo ra thread1 và thread2. Lớp ThreadTest1 tạo ra luồng là một thực thể mới của lớp Mythread. Còn ThreadTest2 thì không tạo ra luồng một cách trực tiếp bởi vì lớp MyClass không phải là lớp con của lớp Thread. Vì vậy, trước tiên lớp ThreadTest2 tạo ra hai đối tượng của lớp MyClass và sau đó chuyển chúng cho Thread() để tạo lập các luồng của lớp Thread. Hàm tạo dựng Thread() được lớp ThreadTest2 sử dụng với đối số là đối tượng của bất kỳ lớp nào cài đặt giao diện Runnable. Phần còn lại của hàm main() trong lớp ThreadTest2 tương tự như ThreadTest1.

Hàm run() của hai lớp ThreadTest1 và ThreadTest2 gần như giống nhau, chỉ khác ở tên gọi.

### 1.1.2. Các trạng thái của Thread

Một luồng có thể ở một trong các trạng thái sau: ([3], [6])

- New: Khi một luồng mới được tạo ra với toán tử new() và sẵn sàng hoạt động.

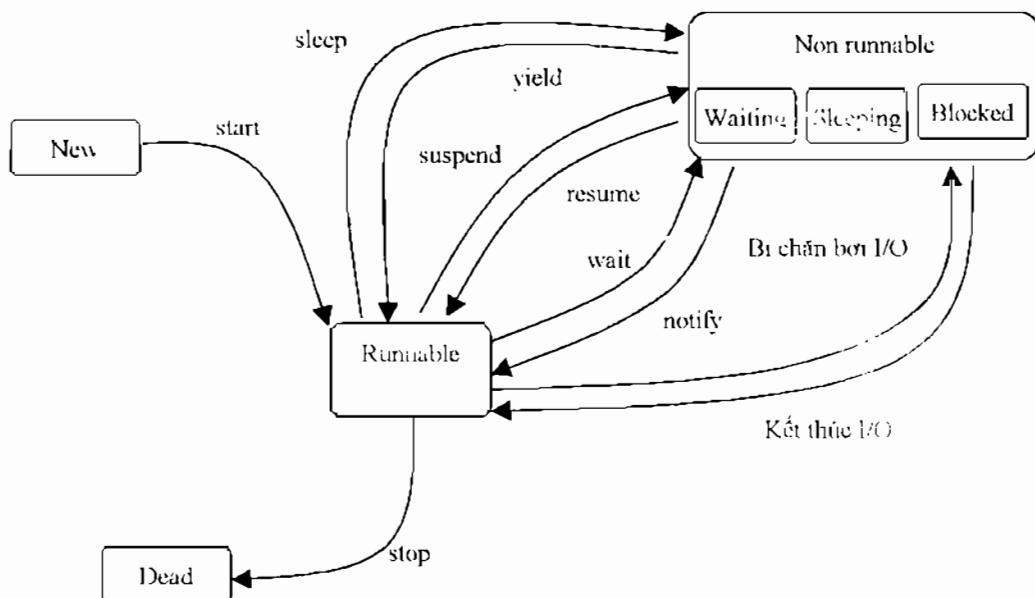
- Runnable: Trạng thái mà luồng đang chiếm CPU để thực hiện, khi bắt đầu thì nó gọi hàm start(). Bộ lập lịch phân luồng của hệ điều hành sẽ quyết định luồng nào sẽ được chuyển về trạng thái Runnable và hoạt động. Cũng cần lưu ý rằng ở một thời điểm, một luồng ở trạng thái Runnable có thể hoặc không thể thực hiện.
- Non runnable (blocked): Từ trạng thái Runnable chuyển sang trạng thái ngừng thực hiện ("bị chặn") khi gọi một trong các hàm: sleep(), suspend(), wait(), hay bị chặn lại ở Input/output. Trong trạng thái bị chặn có ba trạng thái con:
  - Waiting: khi ở trạng thái Runnable, một luồng thực hiện hàm wait() thì nó sẽ chuyển sang trạng thái chờ đợi (Waiting).
  - Sleeping: khi ở trạng thái Runnable, một luồng thực hiện hàm sleep() thì nó sẽ chuyển sang trạng thái ngủ (Sleeping).
  - Blocked: khi ở trạng thái Runnable, một luồng bị chặn lại bởi những yêu cầu về tài nguyên, như yêu cầu vào/ra (I/O), thì nó sẽ chuyển sang trạng bị chặn (Blocked).

Mỗi luồng phải thoát ra khỏi trạng thái Blocked để quay về trạng thái Runnable, khi

- Nếu một luồng đã được cho đi "ngủ" (gọi sleep(n)) sau khoảng thời gian n micro giây.
- Nếu một luồng bị chặn lại vì vào/ra và quá trình này đã kết thúc.
- Nếu luồng bị chặn lại khi gọi hàm wait(), sau đó được thông báo tiếp tục bằng cách gọi hàm notify() hoặc notifyAll().
- Nếu một luồng bị chặn lại để chờ monitor của đối tượng đang bị chiếm giữ bởi luồng khác, khi monitor đó được giải phóng thì luồng bị chặn này có thể tiếp tục thực hiện (khái niệm monitor được đề cập ở phần sau).
- Nếu một luồng bị chặn lại bởi lời gọi hàm suspend(), muốn thực hiện thì trước đó phải gọi hàm resume().

Nếu ta gọi các hàm không phù hợp đối với các luồng thì JVM sẽ phát sinh ra ngoại lệ IllegalThreadStateException.

- Dead: Luồng chuyển sang trạng thái "chết" khi nó kết thúc hoạt động bình thường, hoặc gặp phải ngoại lệ không thực hiện tiếp được. Trong trường hợp đặc biệt, bạn có thể gọi hàm stop() để kết thúc ("giết chết") một luồng.



Hình 1.1. Sơ đồ chuyển trạng của Thread

## 1.2. CÁC MỨC ƯU TIÊN CỦA LUỒNG

Trong Java, mỗi luồng có một mức ưu tiên thực hiện nhất định. Khi chương trình chính thực hiện sẽ tạo ra luồng chính (luồng cha). Luồng này sẽ tạo ra các luồng con, và cứ thế tiếp tục. Theo mặc định, một luồng con sẽ kế thừa mức ưu tiên của luồng cha trực tiếp của nó. Bạn có thể tăng hay giảm mức ưu tiên của luồng bằng cách sử dụng hàm `setPriority()`. Mức ưu tiên của các luồng có thể đặt lại trong khoảng từ `MIN_PRIORITY` (Trong lớp `Thread` được mặc định bằng 1) và `MAX_PRIORITY` (mặc định bằng 10), hoặc `NORM_PRIORITY` (mặc định là 5).

Luồng có mức ưu tiên cao nhất trong số các luồng đang chiếm dụng tài nguyên sẽ tiếp tục thực hiện cho đến khi:

- Nó nhường quyền điều khiển cho luồng khác bằng cách gọi hàm `yield()`
- Nó dừng thực hiện (bi “chết” hoặc chuyển sang trạng thái bị chặn)
- Có một luồng với mức ưu tiên cao hơn vào trạng thái `Runnable`.

Khi đó bộ lập lịch sẽ chọn luồng mới có mức ưu tiên cao nhất trong số những luồng ở trạng thái `Runnable` để thực hiện.

Vấn đề này sinh là chọn luồng nào để thực hiện khi có nhiều hơn một luồng sẵn sàng thực hiện và có cùng một mức ưu tiên cao nhất? Nói chung, một số cơ sở sử dụng bộ lập lịch lựa chọn ngẫu nhiên, hoặc lựa chọn chúng để thực hiện theo thứ tự xuất hiện.

**Ví dụ 1.3.** Chúng ta hãy xét chương trình hiển thị các quả bóng màu xanh hoặc đỏ này (chuyển) theo những đường nhất định. Mỗi khi nhấn nút “Blue ball” thì có 5 luồng được

tạo ra với mức ưu tiên thông thường (mức 5) để hiển thị và di chuyển các quả bóng xanh. Khi nhấn nút “Red ball” thì cũng có 5 luồng được tạo ra với mức ưu tiên (mức 7) cao hơn mức thông thường để hiển thị và di chuyển các quả bóng đỏ. Để kết thúc trò chơi bạn nhấn nút “Close”.

```
//Bounce.java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Bounce{
    public static void main(String arg[]){
        JFrame fr = new BounceFrame();
        fr.show();
    }
}
class BounceFrame extends JFrame{
    public BounceFrame(){
        setSize(300, 200);
        setTitle("Bong chuyen");
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
        Container contentPane = getContentPane();
        canvas = new JPanel();
        contentPane.add(canvas, "Center");
        JPanel p = new JPanel();
        addButton(p, "Blue ball", new ActionListener(){
            public void actionPerformed(ActionEvent evt){
                for(int i = 0; i < 5; i++){
                    Ball b = new Ball(canvas, Color.blue);
                    b.setPriority(Thread.NORM_PRIORITY);
                    b.start();
                }
            }
        });
    }
}
```

```
 addButton(p, "Red ball", new ActionListener(){
    public void actionPerformed(ActionEvent evt){
        for(int i = 0; i < 5; i++){
            Ball b = new Ball(canvas, Color.red);
            b.setPriority(Thread.NORM_PRIORITY + 2);
            b.start();
        }
    }
});

addButton(p, "Close", new ActionListener(){
    public void actionPerformed(ActionEvent evt){
        canvas.setVisible(false);
        System.exit(0);
    }
});
contentPane.add(p, "South");
}

public void addButton(Container c, String title, ActionListener a){
    JButton b = new JButton(title);
    c.add(b);
    b.addActionListener(a);
}

private JPanel canvas;
}

class Ball extends Thread{
    public Ball(JPanel b, Color c){
        box = b; color = c;
    }
    public void draw(){
        Graphics g = box.getGraphics();
        g.setColor(color);
        g.fillOval(x, y, XSIZE, YSIZE);
        g.dispose();
    }
}
```

```
}

public void move(){
    if(!box.isVisible()) return;
    Graphics g = box.getGraphics();
    g.setXORMode(box.getBackground());
    g.setColor(color);
    g.fillOval(x, y, XSIZE, YSIZE);
    x += dx;
    y += dy;
    Dimension d = box.getSize();
    if(x < 0){
        x = 0; dx = -dx;
    }
    if(x + XSIZE >= d.width){
        x = d.width - XSIZE; dx = -dx;
    }
    if(y < 0){
        y = 0; dy = -dy;
    }
    if(y + YSIZE >= d.height){
        y = d.height - YSIZE; dy = -dy;
    }
    g.fillOval(x, y, XSIZE, YSIZE);
    g.dispose();
}

public void run(){
    try{
        for(int i = 1; i <= 1000; i++){
            move();
            sleep(5);
        }
    }catch(InterruptedException e){
    }
}
```

```

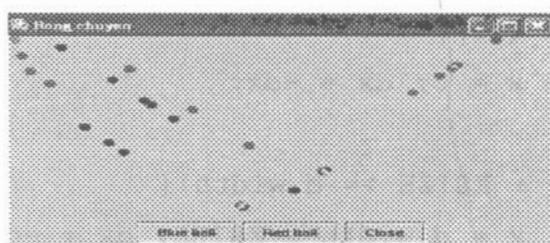
    }

    private JPanel box;

    private static final int XSIZE = 10;
    private static final int YSIZE = 10;
    private int x = 0;
    private int y = 0;
    private int dx = 2;
    private int dy = 2;
    private Color color;
}

}

```



**Hình 1.2. Các mức ưu tiên của luồng**

Chạy chương trình trên chúng ta nhận thấy hình như những quả bóng đỏ nhảy nhanh hơn vì các luồng thực hiện chúng có mức ưu tiên cao hơn.

*Lưu ý:* Các luồng có mức ưu tiên thấp hơn sẽ không có cơ hội thực hiện nếu những luồng cao hơn không nhường, hoặc nhường bằng hàm `yield()`. Vì vậy, ở hàm `run()` chúng ta sử dụng hàm `sleep(5)` để mỗi luồng sau một khoảng thời gian thực hiện thì đi “ngủ” 5 micro giây và dành quyền điều khiển cho luồng khác thực hiện. Hàm `static void yield()` được sử dụng để luồng đang thực hiện nhường quyền cho luồng khác thực hiện. Nếu có những luồng đang ở trạng thái `Runnable` mà có mức ưu tiên ít nhất bằng mức ưu tiên của luồng vừa nhường thì một trong số chúng được xếp lịch để thực hiện.

- Bộ lập lịch thường xuyên tính lại mức ưu tiên của các luồng đang thực hiện
- Tìm luồng có mức ưu tiên cao nhất để thực hiện.

### 1.3. CÁC NHÓM LUỒNG

Các chương trình thường thực hiện với một số luồng nhất định. Sẽ hiệu quả hơn nếu có thể phân loại chúng theo chức năng. Chúng ta hãy xét trình duyệt IE (Internet Explore). Nếu có nhiều luồng đang yêu cầu nạp các bức ảnh về từ máy chủ (Server) và người sử dụng nhấn nút “Stop” để dừng nạp ở trang hiện thời thì nó sẽ dừng đồng thời tất cả các luồng. Java hỗ trợ để bạn có thể tạo ra các nhóm luồng để có thể làm việc đồng thời với chúng như sau.

```

String grName = "ThreadGroup";
ThreadGroup g = new ThreadGroup(grName);
    
```

Sau đó bạn có thể tạo ra các luồng và đưa chúng vào nhóm đã được thiết lập.

```

Thread t = new Thread(g, threadName);
    
```

Để kiểm tra xem trong nhóm g có một luồng nào đang hoạt động thì sử dụng hàm `activeCount()`.

```

if(g.activeCount() == 0) {
    
```

// Tất cả các luồng trong g đã kết thúc

```

}
    
```

## java.lang.ThreadGroup

## API

- `ThreadGroup(String name)` tạo ra một nhóm mới có tên là `name`.
- `ThreadGroup(ThreadGroup parent, String name)` tạo ra một nhóm `name` là con của nhóm `parent`.
- `int activeCount()` trả lại số các luồng đang hoạt động trong nhóm.
- `int enumerate(Thread[] list)` đặt tham chiếu tới từng luồng hoạt động trong nhóm. Bạn có thể sử dụng `activeCount()` để biết được cỡ của `list`.
- `ThreadGroup getParent()` xác định cha của nhóm hiện thời.
- `void interrupt()` dừng tất cả các luồng trong nhóm và tất cả các nhóm con của nó.

## java.lang.Thread

## API

- `Thread(ThreadGroup g, String name)` tạo ra luồng `name` và đặt vào nhóm `g`.
- `ThreadGroup getThreadGroup()` xác định nhóm chứa luồng hiện thời.

## 1.4. ĐỒNG BỘ HÓA

Các luồng chia sẻ với nhau cùng một không gian bộ nhớ, nghĩa là chúng có thể chia sẻ với nhau các tài nguyên. Khi có nhiều hơn một luồng cùng muốn sử dụng một tài nguyên sẽ xuất hiện tình trạng căng thẳng, ở đó chỉ cho phép một luồng được quyền truy cập. Ví dụ, một luồng muốn truy cập vào một biến để đọc dữ liệu, trong khi một luồng khác lại muốn thay đổi biến dữ liệu ở cùng một thời điểm. Để cho các luồng chia sẻ với nhau được các tài nguyên và hoạt động hiệu quả, luôn đảm bảo nhất quán dữ liệu thì phải có cơ chế đồng bộ chúng. Java cung cấp cơ chế đồng bộ ở mức cao để điều khiển truy cập của các luồng vào những tài nguyên dùng chung.

### 1.4.1. Các hàm đồng bộ

Như chúng ta đã biết, khái niệm semaphore (monitor được Tony Hoare đề xuất) thường được sử dụng để điều khiển đồng bộ các hoạt động truy cập vào những tài nguyên dùng chung. Một luồng muốn truy cập vào một tài nguyên dùng chung (như biến dữ liệu) thì trước tiên nó phải yêu cầu để có được monitor riêng. Khi có được monitor thì luồng như có được “chìa khóa” để “mở cửa” vào miền “tranh chấp” (tài nguyên dùng chung) để sử dụng những tài nguyên đó.

Cơ chế monitor thực hiện hai nguyên tắc đồng bộ chính:

- Không một luồng nào khác được phân monitor khi có một luồng đã yêu cầu và đang chiếm giữ nó. Những luồng khác yêu cầu monitor sẽ phải chờ cho đến khi monitor được giải phóng.
- Khi có một luồng giải phóng (ra khỏi) monitor, một trong số các luồng đang chờ monitor có thể truy cập vào tài nguyên dùng chung tương ứng với monitor đó.

Mỗi đối tượng trong Java đều có monitor, mỗi đối tượng có thể được sử dụng như một khóa loại trừ nhau và cung cấp khả năng đồng bộ truy cập vào những tài nguyên chia sẻ.

Trong lập trình có hai cách để thực hiện việc đồng bộ:

- Các hàm (phương thức) được đồng bộ
- Các khối được đồng bộ.

#### (i) Các hàm đồng bộ

Hàm của một lớp chỉ cho phép một luồng được thực hiện ở một thời điểm thì nó phải khai báo synchronized, được gọi là *hàm đồng bộ*. Một luồng muốn gọi để thực hiện một hàm đồng bộ thì nó phải chờ để có được monitor của đối tượng có hàm đó. Trong khi một luồng đang thực hiện hàm đồng bộ thì tất cả các luồng khác muốn thực hiện hàm này của cùng một đối tượng, đều phải chờ cho đến khi luồng đó thực hiện xong và được giải phóng. Bằng cách đó, những hàm được đồng bộ sẽ không bao giờ bị tắc nghẽn. Những hàm không được đồng bộ của đối tượng có thể được gọi thực hiện mọi lúc bởi bất kỳ đối tượng nào.

**Ví dụ 1.4.** Chương trình sau sử dụng Thread để tính tổng các phần tử của một mảng.

```
public class Adder{
    public int[] array;
    private int sum = 0;
    private int index = 0;
    private int noOfThread = 10;
    private int threadQuit;
    public Adder(){
        threadQuit = 0;
        array = new int[1000];
        initializeArray();
    }
}
```

```

        startThread();
    }
    public synchronized int getNextIndex(){
        if(index < 1000) return(index++);
        else return (-1);
    }
    public synchronized void addPartSum(int s){
        sum += s;
        if(++threadQuit == noOfThread)
            System.out.println(The sum is: "+sum);
    }
    private void initializeArray(){
        int i;
        for(i=0; i < 1000; i++) array[i] = i;
    }
    public void startThread(){
        int i;
        for(i = 0; i < 10; i++){
            AdderThread at = new AdderThread(this, i);
            at.start();
        }
    }
    public static void main(String args){
        Adder a = new Adder();
    }
}
class AdderThread extends Thread{
    int s=0;
    Adder parent;
    int num;
    public AdderThread(Adder parent, int num){
        this.parent = parent;
        this.num = num;
    }
    public void run(){
        int index = 0;
        while(index != -1){
            s += parent.array[index];
            index = parent.getNextIndex();
        }
        System.out.println("Tổng bộ phận tính theo luồng
                           số: " + num + " là: " + s);
    }
}

```

Khi chạy, chương trình sẽ thi hành tuân tự các lệnh cho đến khi kết thúc chương trình.

Trong Java, hàm đồng bộ có thể khai báo static. Các lớp cũng có thể có các monitor tương tự như đối với các đối tượng. Một luồng yêu cầu monitor của lớp trước khi nó có thể thực hiện với một hàm được đồng bộ tĩnh (static) nào đó trong lớp, đồng thời các luồng khác muốn thực hiện những hàm như thế của cùng một lớp thì bị chặn lại.

### *(ii) Các khối đồng bộ*

Như trên đã nêu, các hàm đồng bộ của một lớp được đồng bộ dựa vào monitor của đối tượng trong lớp đó. Mặt khác, đồng bộ khối lại cho phép một khối chương trình (một đoạn mã lệnh) được đồng bộ dựa vào monitor của một đối tượng bất kỳ. Đồng bộ khối có dạng như sau:

```
synchronized(<Tham chiếu đối tượng>) {<khối mã lệnh>}
```

Khi một luồng muốn vào khối mã lệnh đồng bộ để thực hiện thì nó phải yêu cầu để có được monitor ở đối tượng tham chiếu, những luồng khác sẽ phải chờ cho đến khi monitor được giải phóng. Ví dụ,

```
class Client{
    BankAccount account;
    //...
    public void updateTransaction() {
        synchronized(account) {           // (1) Khối đồng bộ
            account.update();           // (2)
        }
    }
}
```

Câu lệnh (2) được đồng bộ trong khối đồng bộ (1) theo đối tượng account của lớp BankAccount. Khi có một số luồng đồng thời muốn thực hiện updateTransaction() đối với một đối tượng của lớp Client thì ở mỗi thời điểm, câu lệnh (2) chỉ thực hiện được ở một luồng.

## 1.4.2. Sự trao đổi giữa các luồng không đồng bộ

Để loại bỏ được sự truy cập đồng thời của nhiều luồng vào những đối tượng dùng chung, chúng ta phải biết cách để đồng bộ chúng.

**Ví dụ 1.5.** Chúng ta hãy xây dựng hệ thống ngân hàng có 10 tài khoản, trong đó có các giao dịch chuyển tiền giữa các tài khoản với nhau một cách ngẫu nhiên. Chương trình tạo ra 10 luồng cho 10 tài khoản. Mỗi giao dịch được một luồng phục vụ sẽ chuyển một lượng tiền ngẫu nhiên từ một tài khoản sang tài khoản khác.

Chúng ta xây dựng lớp Bank có hàm transfer() để chuyển tiền từ một tài khoản sang tài khoản khác nếu số tiền ở tài khoản gốc còn nhiều tiền hơn số tiền cần chuyển.

```

        startThread();
    }
    public synchronized int getNextIndex(){
        if(index < 1000) return(index++);
        else return (-1);
    }
    public synchronized void addPartSum(int s){
        sum += s;
        if(++threadQuit == noOfThread)
            System.out.println(The sum is: "+sum);
    }
    private void initializeArray(){
        int i;
        for(i=0; i < 1000; i++) array[i] = i;
    }
    public void startThread(){
        int i;
        for(i = 0; i < 10; i++){
            AdderThread at = new AdderThread(this, i);
            at.start();
        }
    }
    public static void main(String args){
        Adder a = new Adder();
    }
}
class AdderThread extends Thread{
    int s=0;
    Adder parent;
    int num;
    public AdderThread(Adder parent, int num){
        this.parent = parent;
        this.num = num;
    }
    public void run(){
        int index = 0;
        while(index != -1){
            s += parent.array[index];
            index = parent.getNextIndex();
        }
        System.out.println("Tổng bộ phân tinh theo luồng
                           số: " + num + " là: " + s);
    }
}

```

Khi chạy, chương trình sẽ thi hành tuần tự các lệnh cho đến khi kết thúc chương trình.

```

public void transfer(int from, int to, int amount){
    if(accounts[from] < amount) return;
    accounts[from] -= amount;
    accounts[to] += amount;
    numTransacts++;
    if(numTransacts % NTEST == 0) test();
}

```

Sau đó xây dựng lớp TransferThread, kế thừa lớp Thread và nạp chồng hàm run() để thực hiện việc chuyển một lượng tiền ngẫu nhiên sang một tài khoản khác cũng ngẫu nhiên.

```

class TransferThread extends Thread{
    // Các thuộc tính
    public void run(){
        try{
            while(!interrupted()){
                int toAcc = (int)(bank.size() * Math.random());
                int amount = (int)(maxAmount * Math.random());
                bank.transfer(fromAcc, toAcc, amount);
                sleep(1);
            }
        } catch(InterruptedException e){
        }
    }
}

```

Chương trình thực hiện 10000 giao dịch, mỗi giao dịch transfer() sẽ gọi hàm test() để tính lại tổng số tiền và in ra màn hình.

Chương trình sẽ chạy liên tục và không dừng. Muốn dừng, bạn hãy nhấn CTRL + C để kết thúc chương trình.

```

// AsynBankTransfer.java

public class AsynBankTransfer{
    public static void main(String arg[]){
        Bank b = new Bank(NACCOUNTS,INI_BALANCE);
        for(int i = 0; i < NACCOUNTS; i++){
            TransferThread t = new TransferThread(b, i, INI_BALANCE);

```

```
        t.setPriority(Thread.NORM_PRIORITY + i % 2);
        t.start();
    }
}

public static final int NACCOUNTS = 10;
public static final intINI_BALANCE = 10000;
}

class Bank{
    public static final int NTEST = 1000;
    private int[] accounts;
    private long numTransacts = 0;
    public Bank(int n, int initBalance){
        accounts = new int[n];
        for(int i = 0; i < accounts.length; i++)
            accounts[i] = initBalance;
        numTransacts = 0;
    }
    public void transfer(int from, int to, int amount){
        if(accounts[from] < amount) return;
        accounts[from] -= amount;
        accounts[to] += amount;
        numTransacts++;
        if(numTransacts % NTEST == 0) test();
    }
    void test(){
        int sum = 0;
        for(int i = 0; i < accounts.length; i++)
            sum += accounts[i];
        System.out.println("Giao dich: " + numTransacts
                           + " tong so: " + sum);
    }
    public int size(){
        return accounts.length;
```

```

    }

}

class TransferThread extends Thread{
    private Bank bank;
    private int fromAcc;
    private int maxAmount;
    public TransferThread(Bank b, int from, int max){
        bank = b;
        fromAcc = from;
        maxAmount = max;
    }
    public void run(){
        try{
            while(!interrupted()){
                int toAcc = (int)(bank.size() * Math.random());
                int amount = (int)(maxAmount * Math.random());
                bank.transfer(fromAcc, toAcc, amount);
                sleep(1);
            }
        }catch(InterruptedException e){
        }
    }
}

```

#### 1.4.3. Đồng bộ việc truy cập vào các tài nguyên chia sẻ

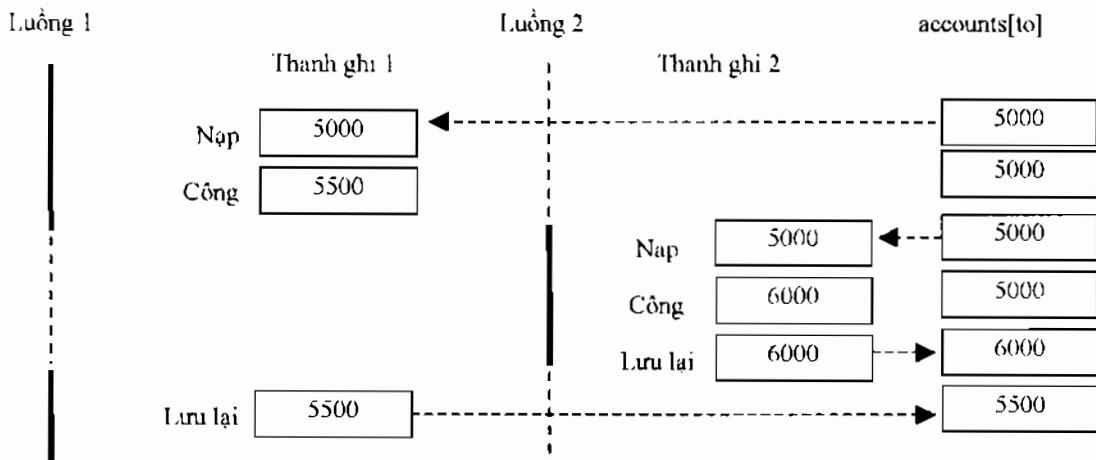
Chương trình trên thực hiện với 10 luồng hoạt động không đồng bộ để chuyển tiền giữa các tài khoản trong một ngân hàng. Vấn đề sẽ nảy sinh khi có hai luồng đồng thời muốn chuyển tiền vào cùng một tài khoản. Giả sử hai luồng cùng thực hiện:

```
accounts[to] += amount;
```

Đây không phải là thao tác nguyên tố. Câu lệnh này được thực hiện như sau:

1. Nạp accounts[to] vào thanh ghi
2. Cộng số tiền trong tài khoản accounts với amount
3. Lưu lại kết quả cho accounts[to].

Giả sử có hai tài khoản cùng chuyển tiền vào cùng một tài khoản. Chúng ta có thể giả thiết luồng thứ nhất thực hiện bước 1 và 2 với amount = 500, sau đó nó bị ngắt. Luồng thứ hai có thể thực hiện trọn vẹn cả ba bước trên với amount = 1000, và luồng thứ nhất kết thúc việc cập nhật bằng cách thực hiện nốt bước 3. Quá trình này được mô tả như ở hình 1.3.



Hình 1.3. Truy cập không đồng bộ của hai luồng

Kết thúc luồng thứ nhất, accounts[to] có 6000, nhưng ngay sau đó luồng thứ hai kết thúc thì cũng chính tài khoản đó chưa chuyển tiền đi đâu cả, nhưng lại chỉ còn 5500. Trong trường hợp này, ở tài khoản nhận tiền chuyển đến phải có là 6500 đơn vị tiền mới là chính xác.

Như vậy, hoạt động giao dịch giữa các tài khoản trong ngân hàng sẽ không còn chính xác, nghĩa là xuất hiện sự sai lệch về dữ liệu, không đảm bảo tính toàn vẹn của dữ liệu. Nhưng, nếu tất cả các luồng cùng thực hiện với cùng một mức ưu tiên thì sẽ rất chậm, bởi vì mỗi luồng sau khi thực hiện “một chút” công việc lại phải đi “ngủ” để các luồng khác thực hiện, rồi lại tiếp tục, v.v.

Các vấn đề trên sẽ được giải quyết khi chúng ta sử dụng cơ chế điều khiển hoạt động của các luồng theo mức ưu tiên và đồng bộ hóa để đảm bảo rằng một luồng thực hiện xong việc cập nhật rồi mới trao quyền cho luồng tiếp theo.

Như trên đã nêu, Java sử dụng cơ chế đồng bộ khá hiệu quả là monitor. Một hàm sẽ không bị ngắt nếu bạn khai báo nó là synchronized, như

```

public synchronized void transfer(int from, int to, int amount) {
    if(accounts[from] < amount) return;
    accounts[from] -= amount;
    accounts[to] += amount;
    numTransacts++;
    if(numTransacts % NTEST == 0) test(); //NTEST-nội hàng số
  
```

```

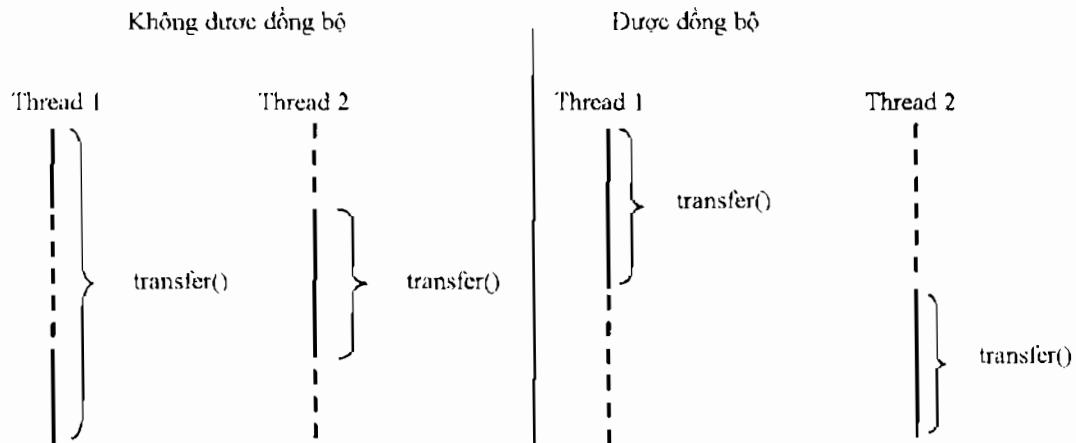
    }

    public synchronized void test(){
        int sum = 0;
        for(int i = 0; i < accounts.length; i++)
            sum += accounts[i];
        System.out.println("Giao dich: " + numTransacts
                           + " tong so: " + sum);
    }
}

```

Khi có một luồng gọi hàm được đồng bộ thì nó được đảm bảo rằng, hàm này phải thực hiện xong thì luồng khác mới được sử dụng đối với cùng những tài nguyên dùng chung đó.

Hoạt động của các luồng không đồng bộ và đồng bộ của hai luồng thực hiện gọi hàm transfer() được mô tả như hình 1.4.



*Hình 1.4. So sánh hoạt động của các luồng đồng bộ và không đồng bộ*

Viết lại chương trình trên với hai hàm transfer(), test() được khai báo đồng bộ synchronized như trên và chạy thử, chúng ta thấy sẽ không còn xuất hiện sự sai lệch dữ liệu nữa.

### Các khóa đối tượng

Khi một luồng gọi một hàm được đồng bộ thì đối tượng của nó bị “khóa”. Chúng ta hình dung như mỗi đối tượng có một “chìa khóa” để mở cửa vào “nhà”. Ban đầu chìa khóa để ở ngoài bậc cửa. Khi một luồng muốn sử dụng một hàm đồng bộ, nó dùng chìa khóa đối tượng “mở cửa” để vào bên trong, đặt chìa khóa phía trong “cánh cửa” và thực hiện một số công việc của mình. Như vậy, khi một luồng khác muốn gọi hàm đồng bộ của cùng đối tượng đó thì sẽ không mở được cửa để vào vùng chung đó vì không có chìa khóa. Sau khi thực hiện xong, luồng ở bên trong giải phóng hàm đồng bộ vừa sử dụng, ra khỏi đối tượng và đưa chìa khóa ra ngoài bậc cửa để những luồng khác có thể tiếp tục công việc của mình.

Một luồng có thể giữ nhiều khóa đối tượng ở cùng một thời điểm, như trong khi đang thực hiện một lời gọi hàm đồng bộ của một đối tượng, nó lại gọi tiếp hàm đồng bộ của đối tượng khác. Nhưng, tại mỗi thời điểm, mỗi khóa đối tượng chỉ được một luồng sở hữu.

Chúng ta hãy phân tích chi tiết hơn hoạt động của hệ thống ngân hàng. Một giao dịch chuyển tiền sẽ không thực hiện được nếu không còn đủ tiền. Nó phải chờ cho đến khi các tài khoản khác chuyển tiền đến và khi có đủ thì mới thực hiện được giao dịch đó.

```
public synchronized void transfer(int from, int to, int amount) {
    while(accounts[from] < amount)
        wait();
    // Chuyển tiền
}
```

Chúng ta sẽ làm gì khi trong tài khoản không có đủ tiền? Tất nhiên là phải chờ cho đến khi có đủ tiền trong tài khoản. Nhưng transfer() là hàm được đồng bộ. Do đó, khi một luồng đã chiếm dụng khóa đối tượng thì các luồng khác sẽ không có cơ hội sở hữu khóa đó, cho đến khi nó được giải phóng.

Khi wait() được gọi ở trong hàm được đồng bộ (như ở transfer()), luồng hiện thời sẽ bị chặn lại và trao lại khóa đối tượng cho luồng khác.

Có sự khác nhau thực sự giữa luồng đang chờ để sử dụng hàm đồng bộ với hàm bị chặn lại bởi hàm wait(). Khi một luồng gọi wait() thì nó được đưa vào danh sách hàng đợi. Cho đến khi các luồng chưa được đưa ra khỏi danh sách hàng đợi thì bộ lập lịch sẽ bỏ qua và do vậy chúng không thể tiếp tục được. Để đưa một luồng ra khỏi danh sách hàng đợi thì phải có một luồng khác gọi notify() hoặc notifyAll() trên *cùng một đối tượng*.

- *notify()* đưa một luồng bất kỳ ra khỏi danh sách hàng đợi.
- *notifyAll()* đưa tất cả các luồng ra khỏi danh sách hàng đợi.

Những luồng đưa ra khỏi danh sách hàng đợi sẽ được bộ lập lịch kích hoạt chúng. Ngay tức khắc, luồng nào chiếm được khóa đối tượng thì sẽ bắt đầu thực hiện. Như vậy, trong hàm transfer() chúng ta gọi notifyAll() khi kết thúc việc chuyển tiền để một trong các luồng có thể được tiếp tục thực hiện và tránh bế tắc. Cuối cùng chương trình sử dụng cơ chế đồng bộ được viết lại như sau.

```
// SynBankTransfer.java

public class SynBankTransfer{
    public static void main(String arg[]){
        Bank b = new Bank(NACCOUNTS,INI_BALANCE);
        for(int i = 0; i < NACCOUNTS; i++){
            TransferThread t = new TransferThread(b, i, INI_BALANCE);
            t.setPriority(Thread.NORM_PRIORITY + i % 2);
        }
    }
}
```

```
        t.start();
    }
}

public static final int NACCOUNTS = 10;
public static final intINI_BALANCE = 10000;
}

class Bank{
    public static final int NTEST = 1000;
    private int[] accounts;
    private long numTransacts = 0;
    public Bank(int n, int initBalance){
        accounts = new int[n];
        for(int i = 0; i < accounts.length; i++)
            accounts[i] = initBalance;
        numTransacts = 0;
    }
    public void transfer(int from, int to, int amount){
        while(accounts[from] < amount) wait();
        accounts[from] -= amount;
        accounts[to] += amount;
        numTransacts++;
        notifyAll();
        if(numTransacts % NTEST == 0) test();
    }
    public synchronized void test(){
        int sum = 0;
        for(int i = 0; i < accounts.length; i++)
            sum += accounts[i];
        System.out.println("Giao dich: " + numTransacts
                           + " tong so: " + sum);
    }
    public int size(){
        return accounts.length;
    }
}
```

```

}

class TransferThread extends Thread{
    private Bank bank;
    private int fromAcc;
    private int maxAmount;
    public TransferThread(Bank b, int from, int max){
        bank = b;
        fromAcc = from;
        maxAmount = max;
    }
    public void run(){
        try{
            while(!interrupted()){
                int toAcc = (int)(bank.size() * Math.random());
                int amount = (int)(maxAmount * Math.random());
                bank.transfer(fromAcc, toAcc, amount);
                sleep(1);
            }
        }catch(InterruptedException e){
        }
    }
}

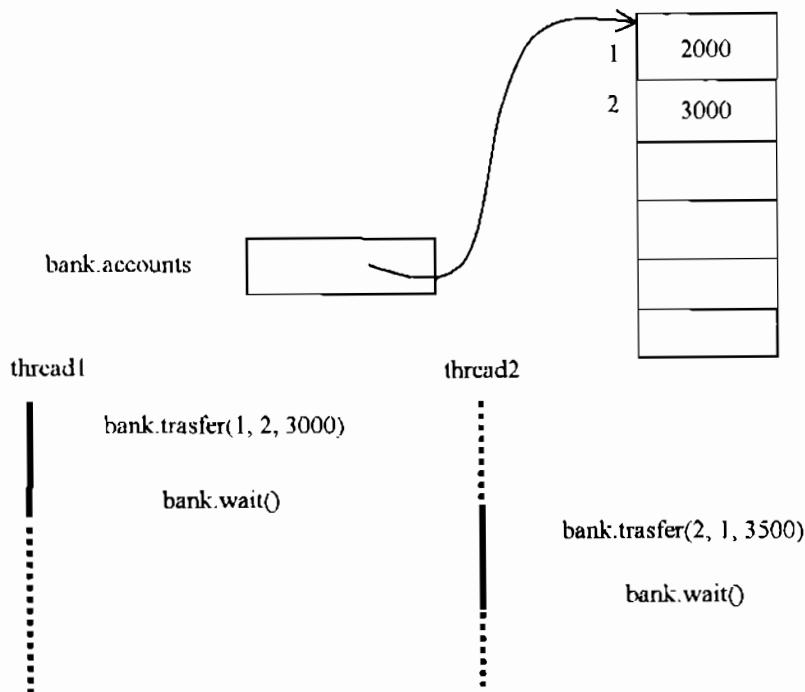
```

Nếu bạn chạy chương trình với các hàm transfer(), test() được đồng bộ thi mọi việc sẽ thực hiện chính xác đúng theo yêu cầu. Tuy nhiên, bạn cũng có thể nhận thấy chương trình sẽ chạy chậm hơn chút ít bởi vì phải trả giá cho cơ chế đồng bộ nhằm đảm bảo cho hệ thống hoạt động chính xác, đảm bảo nhất quán dữ liệu, hoặc tránh gây ra tắc nghẽn.

## 1.5. VẤN ĐỀ TẮC NGHẼN

Cơ chế đồng bộ trong Java là rất tiện lợi, khá mạnh, nhưng không giải quyết được mọi vấn đề này sinh trong quá trình xử lý đa luồng. Ví dụ, hãy xét tình huống: ở Account 1 có 2000\$, Account 2 có 3000\$ và luồng thread1 cần chuyển 3000\$ từ Account 1 sang Account 2, ngược lại thread2 phải chuyển 3500\$ từ Account 2 sang Account 1. Khi đó, thread1 và thread2 rơi vào tình trạng chết tắc, hoặc tắc nghẽn vì chúng chặn lẫn nhau. Một hệ thống mà tất cả các luồng (tiền trình) bị chặn lại để chờ lẫn nhau và không một

luồng (tiến trình) nào thực hiện tiếp thì được gọi là hệ thống bị chết tắc (tắc nghẽn). Trong tình huống ở trên, cả hai luồng đều phải gọi `wait()` vì cả hai tài khoản đều không đủ số tiền để chuyển đi. Hiện tượng chết tắc được mô tả như hình 1.5 dưới đây.



Hình 1.5. Hiện tượng tắc nghẽn

Trong chương trình `SynBankTransfer.java`, hiện tượng tắc nghẽn không xuất hiện bởi một lý do đơn giản. Mỗi giao dịch chuyển tiền nhiều nhất là 10000\$. Có 10 tài khoản với tổng số tiền là 100000\$. Do đó, ở mọi thời điểm đều có ít nhất một tài khoản có không ít hơn 10000\$, nghĩa là luồng phụ trách tài khoản đó được phép thực hiện.

Tuy nhiên, khi lập trình ta có thể gây ra tình huống khác có thể làm xuất hiện tắc nghẽn, chẳng hạn trong `SynBankTransfer.java` thay vì gọi `notifyAll()` ta gọi `notify()`. Như ở trên đã phân tích, `notifyAll()` thông báo cho tất cả các luồng đang chờ nếu ở tài khoản có đủ tiền chuyển đi thì có thể tiếp tục thực hiện, còn `notify()` chỉ báo cho một luồng được tiếp tục nếu có đủ tiền chuyển đi. Khi đó, nếu luồng được thông báo lại không thể thực hiện, vì không đủ tiền để chuyển chặng hạn, thì tất cả các luồng khác cũng sẽ bị chặn lại. Chúng ta hãy xét kịch bản sau:

- + Account 1: 19000\$
- + Tất cả các Account còn lại đều có 9000\$
- + Thread 1: chuyển 9500\$ từ Account 1 sang Account 2
- + Tất luồng khác đều chuyển sang tài khoản khác một lượng tiền là 9100\$.

Rõ ràng là chỉ có thread1 đủ tiền để chuyển còn các luồng khác bị chặn lại thread1 thực hiện chuyển tiền xong ta có:

- + Account 1: 9500\$
- + Account 2: 18500\$
- + Tất cả các Account còn lại đều có 9000\$

Giả sử thread1 gọi `notify()`. Hàm này chỉ thông báo cho một luồng ngẫu nhiên để nó có thể tiếp tục thực hiện. Giả sử đó là thread3. Nhưng luồng này cũng không chuyển được vì không đủ tiền ở tài khoản Account 3, nên phải chờ (gọi `wait()`). thread1 vẫn tiếp tục thực hiện. Một giao dịch mới ngẫu nhiên lại được tạo ra.

Ví dụ,

Thread 1: chuyển 9600\$ từ Account 1 sang Account 2.

Sau đó thread1 gọi hàm `wait()`, và như vậy tất cả các luồng đều rơi vào tình trạng tắc nghẽn.

Qua ví dụ trên cho thấy, một *ngôn ngữ lập trình có cơ chế hỗ trợ đồng bộ là chưa đủ để giải quyết vấn đề tắc nghẽn*. Quan trọng là khi thiết kế chương trình, ta phải đảm bảo rằng ở mọi thời điểm có ít nhất một luồng (tiền trình) được tiếp tục thực hiện và cố gắng không để xảy ra *hiện tượng chết đói*, nghĩa là trạng thái mà ở đó có một số luồng không bao giờ được tiếp tục thực hiện.

## 1.6. ĐA LUỒNG TRONG APPLET

Trong các mục trước, chúng ta đã tìm cách chia một chương trình thành nhiều tác vụ con thực hiện đồng thời. Mỗi tác vụ cần phải đặt vào hàm `run()` của lớp kế thừa từ lớp Thread. Nhưng, nếu chúng ta muốn bổ sung hàm `run()` vào một lớp được kế thừa từ một lớp khác, ví dụ lớp MyApplet, thì thực hiện như thế nào? Một lớp ứng dụng nhúng đã kế thừa từ JApplet (hoặc Applet), nên không thể kế thừa thêm một lớp cơ sở nữa. Trường hợp này chúng ta phải cài đặt Runnable như trên đã phân tích.

Sử dụng luồng trong Java, ta có thể tạo ra một MyApplet chạy trong luồng riêng của nó mà không quấy rầy đến các thành phần khác trong hệ thống. Ta có thể chạy nhiều MyApplet cùng một lúc trên một trang ứng dụng. Tuy nhiên cũng phải cẩn trọng, nếu mở quá nhiều luồng, sẽ làm chương trình chạy chậm lại mặc dù các luồng đó chạy độc lập với nhau [3].

**Ví dụ 1.6.** Xây dựng Animation với nhiều luồng để nạp một dãy ảnh nhằm tạo ra phim hoạt hình. Vì muốn sử dụng đa luồng trong một lớp đã kế thừa từ một lớp khác với Thread, nên ta phải cài đặt Runnable.

```
public class Animation extends JApplet
    implements Runnable{
    // ...
}
```

```

public void run(){
    // Hoạt động của luồng thực hiện ở đây
}
}

```

Ta phải đảm bảo tạo ra các luồng và tham chiếu tới các đối tượng của Runnable trong toán tử tạo lập. Luồng được tạo ra sẽ gọi hàm `run()` của đối tượng tương ứng. Khi gọi hàm này sẽ lại tự động gọi hàm `start()` được nạp chồng trong applet, ví dụ:

```

public class Animation extends JApplet
    implements Runnable{
    ...
    public void start(){
        runner = new Thread(this);
        runner.start();
        showStatus("Click to stop");
    }
    ...
    private Thread runner;
}

```

Đối số `this` trong toán tử tạo lập `Thread(this)` chỉ ra đối tượng mà hàm `run()` phải gọi thực hiện khi luồng thực hiện đối với đối tượng `Animation` hiện thời.

Mục đích của chúng ta là hiển thị một dãy các ảnh liên tiếp để tạo ra cảm giác chuyển động của các đối tượng ảnh. Mỗi ảnh được hiển thị trong một khung (frame). Ta có thể đặt mỗi frame trong một file (tệp) ảnh riêng hoặc có thể đặt tất cả trong cùng tệp. Trong ví dụ này, chúng ta sử dụng một tệp với 36 ảnh liên tiếp của một quả cầu quay. Ta sử dụng đối tượng của lớp `MediaTracker` để nạp ảnh. Việc nạp ảnh, nhất là ảnh từ Internet là tương đối chậm.

Khi ảnh đã được nạp, ta phải vẽ nó.

```
g.drawImage(image, 0, - i * imageHeight / imageCount, null);
```

Như vậy, chương trình `Animation` để hiển thị ảnh hoạt hình có thể viết đơn giản như sau.

```

// Animation.java

import java.awt.*;
import java.awt.image.*;
import java.awt.event.*;
import javax.swing.*;
import java.net.*;

public class Animation extends JApplet

```

```
    implements Runnable{

public void init(){
    addMouseListener(new MouseAdapter(){
        public void mousePress(MouseEvent evt){
            if(runner == null) start();
            else stop();
        }
    });
    try{
        imageName = getParameter("imagename");
        if(imageName == null) imageName = "";
        imageCount = 1;
        String param = getParameter("imagecount");
        if(param != null)
            imageCount = Integer.parseInt(param);
    }catch(Exception e){
        showStatus("Error: " + e);
    }

    image = null;
    loadImage();
}

public void loadImage(){
    try{
        URL url = new URL(getDocumentBase(), imageName);
        image = getImage(url);
        MediaTracker tracker = new MediaTracker(this);
        tracker.addImage(image, 0);
        tracker.waitForID(0);
        imageW = image.getWidth(null);
        imageH = image.getHeight(null);
        resize(imageW, imageH / imageCount);
    }catch(InterruptedException e){

```

```
        showStatus("Loading interrupted!");
    }catch(MalformedURLException e){
        showStatus("Bad URL!");
    }
}

public void paintComponent(Graphics g){
    if(image == null) return;
    g.drawImage(image, 0, -(imageH / imageCount)
               * current, null);
}

public void start(){
    runner = new Thread(this);
    runner.start();
    showStatus("Click to stop");
}

public void stop(){
    runner.interrupt();
    runner = null;
    showStatus("Click to restart");
}

public void run(){
    try{
        while(!Thread.interrupted()){
            repaint();
            current = (current + 1) % imageCount;
            Thread.sleep(200);
        }
    }catch(InterruptedException e){
        showStatus("Interrupted!");
    }
}

private int current;
```

```

private int imageCount;
private int imageW, imageH;
private Image image;
private String imageName;
private Thread runner;
}

```

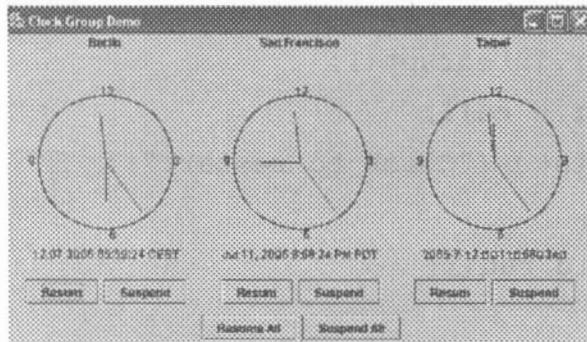
Chương trình Animation cần có tên của tệp ảnh và số lượng frame. Các thông số này có thể cung cấp bởi thẻ PARAM

```

<applet code = "Animation.class" width = 100 height = 100>
<param name = imagename value = "globe.gif">
<param name = imagecount value = "36">
</applet>

```

**Ví dụ 1.7.** Trong nhiều môi trường lập trình ta cần biết thời gian hoặc đặt lại thời gian của chương trình ứng dụng. Chương trình sau đây hiển thị hệ thống các đồng hồ theo nhiều múi giờ khác nhau. Mỗi đồng hồ có một luồng riêng, do vậy cần phải xây dựng lớp Timer kế thừa Thread. Chương trình ClockGroup thực hiện sẽ hiển thị các đồng hồ quốc tế như hình 1.6.



Hình 1.6. Hệ thống đồng hồ quốc tế

```

/* ClockGroup.java
 * Hiển thị hệ thống đồng hồ quốc tế
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;
import javax.swing.Timer;
import java.text.*;

```

```
public class ClockGroup extends JApplet implements ActionListener{
    private ClockPanel clockP1, clockP2, clockP3;
    private JButton btResum, btSus;
    public static void main(String[] arg){
        JFrame fr = new JFrame("Clock Group Demo");
        ClockGroup app = new ClockGroup();
        fr.getContentPane().add(app, BorderLayout.CENTER);
        app.init();
        app.start();
        fr.setSize(600, 350);
        fr.setVisible(true);
    }
    public void init(){
        // Tạo ra Panel p1 để chứa 3 đồng hồ
        JPanel p1 = new JPanel();
        p1.setLayout(new GridLayout(1,3));
        // Tạo lập đồng hồ Berlin
        p1.add(clockP1 = new ClockPanel());
        clockP1.setTitle("Berlin");
        clockP1.clock.setTimeZoneID("ECT");
        clockP1.clock.setLocale(Locale.GERMAN);
        // Tạo lập đồng hồ SanFrancisco
        p1.add(clockP2 = new ClockPanel());
        clockP2.setTitle("San Francisco");
        clockP2.clock.setTimeZoneID("PST");
        clockP2.clock.setLocale(Locale.US);
        // Tạo lập đồng hồ Taipei
        p1.add(clockP3 = new ClockPanel());
        clockP3.setTitle("Taipei");
        clockP3.clock.setLocale(Locale.CHINESE);
        clockP3.clock.setTimeZoneID("CTT");
        // Panel p2 để chứa 2 nhóm nút
        JPanel p2 = new JPanel();
        p2.setLayout(new FlowLayout());
```

```
p2.add(btResum = new JButton("Resume All"));
p2.add(btSus = new JButton("Suspend All"));
// Đưa p1, p2 vào Applet
getContentPane().setLayout(new BorderLayout());
getContentPane().add(p1,BorderLayout.CENTER);
getContentPane().add(p2,BorderLayout.SOUTH);
// Đăng ký listener
btResum.addActionListener(this);
btSus.addActionListener(this);
}
public void actionPerformed(ActionEvent e){
if(e.getSource() == btResum){
    // Khởi động tất cả các đồng hồ
    clockP1.resume();
    clockP2.resume();
    clockP3.resume();
}
else if (e.getSource() == btSus){
    // Tạm dừng các đồng hồ
    clockP1.suspend();
    clockP2.suspend();
    clockP3.suspend();
}
}
class ClockPanel extends JPanel implements ActionListener{
private JLabel labelT;
protected Clock clock = null;
private JButton btResum, btSus;
public ClockPanel(){
    JPanel bt = new JPanel();
    bt.add(btResum = new JButton("Resum"));
    bt.add(btSus = new JButton("Suspend"));
    // Đặt BorderLayout cho ClockPanel
    setLayout(new BorderLayout());
}
```

```
// Đưa tiêu đề ở phía trên
add(labelT = new JLabel(), BorderLayout.NORTH);
labelT.setHorizontalAlignment(JLabel.CENTER);
// Đưa đồng hồ vào giữa panel
add(clock = new Clock(),BorderLayout.CENTER);
add(bt,BorderLayout.SOUTH);
// Lắng nghe các sự kiện
btResum.addActionListener(this);
btSus.addActionListener(this);
}
public void setTitle(String title){
    labelT.setText(title);
}
public void actionPerformed(ActionEvent e){
    if(e.getSource() == btResum){
        clock.resume();
    }
    else if (e.getSource() == btSus){
        clock.suspend();
    }
}
public void resume(){
    if(clock != null) clock.resume();
}
public void suspend(){
    if(clock != null) clock.suspend();
}
}
class Clock extends StillClock implements ActionListener{
protected Timer timer;
public Clock(){
    this(Locale.getDefault(), TimeZone.getDefault());
}
public Clock(Locale l, TimeZone t){
```

```
        super(l, t);
        timer = new Timer(1000,this);
        timer.start();
    }

    public void resume(){
        timer.start();
    }

    public void suspend(){
        timer.stop();
    }

    public void actionPerformed(ActionEvent e){
        repaint();
    }

}

// Hiển thị đồng hồ ở JPanel
class StillClock extends JPanel{
    protected TimeZone timeZ;
    protected int xC, yC;
    protected int clockR;
    protected DateFormat form;
    public StillClock(){
        this(Locale.getDefault(), TimeZone.getDefault());
    }
    public StillClock(Locale l, TimeZone t){
        setLocale(l);
        this.timeZ = t;
    }
    public void setTimeZoneID(String newT){
        timeZ = TimeZone.getTimeZone(newT);
    }
}

// Viết đè paintComponent() để hiển thị đồng hồ
public void paintComponent(Graphics g){
    super.paintComponent(g);
```

```
clockR=(int)(Math.min(getSize().width,getSize().height)*0.7*0.5);
xC = (getSize().width)/2;
yC = (getSize().height)/2;
// Vẽ hình tròn
g.setColor(Color.black);
g.drawOval(xC-clockR, yC - clockR, 2*clockR,2*clockR);
g.drawString("12", xC - 5, yC - clockR);
g.drawString("9", xC - clockR-10, yC + 3);
g.drawString("3", xC + clockR, yC + 3);
g.drawString("6", xC + 3, yC + clockR + 10);
// Đọc thời gian từ máy tính sử dụng GregorianCalendar
GregorianCalendar cal = new GregorianCalendar(timeZ);
// Vẽ kim giây
int second = (int) cal.get(GregorianCalendar.SECOND);
int sLen = (int)(clockR *0.9);
int xS = (int)(xC+sLen*Math.sin(second*(2*Math.PI/60)));
int yS = (int)(yC-sLen*Math.cos(second*(2*Math.PI/60)));
g.setColor(Color.red);
g.drawLine(xC, yC, xS, yS);
// Vẽ kim phút
int minute = (int) cal.get(GregorianCalendar.MINUTE);
int mLen = (int)(clockR *0.75);
int xM = (int)(xC+mLen*Math.sin(minute*(2*Math.PI/60)));
int yM = (int)(yC-mLen*Math.cos(minute*(2*Math.PI/60)));
g.setColor(Color.blue);
g.drawLine(xC, yC, xM, yM);
// Vẽ kim giờ
int hour = (int) cal.get(GregorianCalendar.HOUR_OF_DAY);
int hLen = (int)(clockR *0.6);
int xH = (int)(xC+hLen*Math.sin((hour+minute/60.0)*(2*Math.PI/12)));
int yH = (int)(yC-hLen*Math.cos((hour+minute/60.0)*(2*Math.PI/12)));
g.setColor(Color.black);
g.drawLine(xC, yC, xH, yH);
```

```

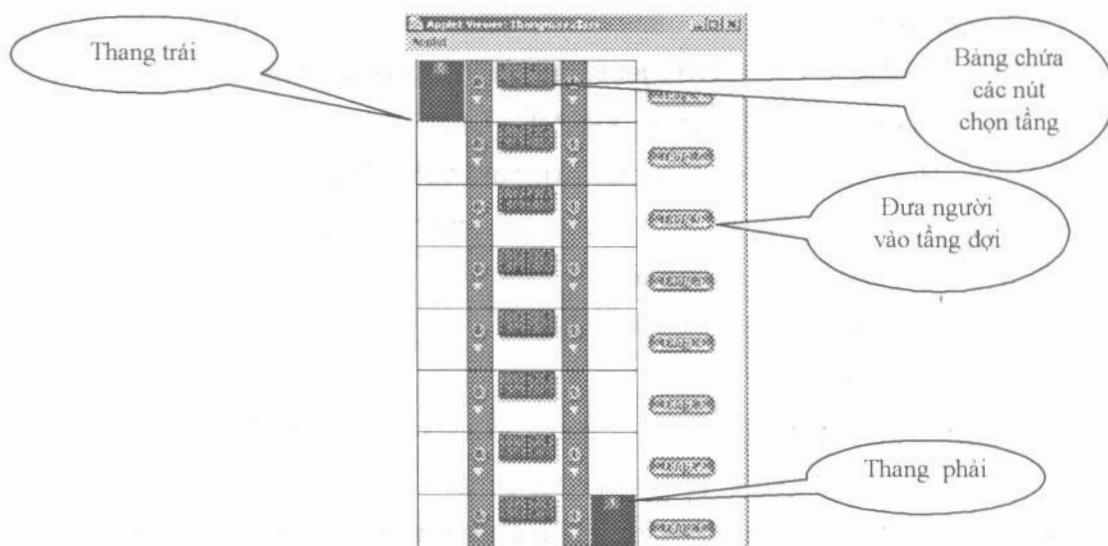
// Đặt chế độ hiển thị theo múi giờ địa phương và kiểu hiển thị
form = DateFormat.getDateInstance(
    DateFormat.MEDIUM, DateFormat.LONG, getLocale());
form.setTimeZone(timeZ);
// Hiển thị ngày giờ
String today = form.format(cal.getTime());
FontMetrics fm = g.getFontMetrics();
g.drawString(today, getSize().width-fm.stringWidth(today))/2,C+clockR+30);
}

```

## BÀI TẬP

- 1.1. Sử dụng kỹ thuật đa luồng để mô phỏng hoạt động hệ thống có n thang máy trong tòa nhà có m tầng. Một người ở trên một tầng bất kỳ yêu cầu đến một tầng nào đó sẽ có thang đến nhận người đó trong thời gian nhanh nhất. Khi có người vào thang máy rồi, nó vẫn tiếp nhận các yêu cầu khác. Nếu người yêu cầu đi lên mà thang di ngang qua đang ở trạng thái đi xuống thì nó không nhận yêu cầu đó, nó chỉ nhận những yêu cầu cùng chiều.

Hệ thống được minh họa như sau.



- 1.2. Xây dựng dịch vụ tán gẫu (Chat) giữa các máy bằng kỹ thuật đa luồng và Socket. Yêu cầu của chương trình:

- Chương trình dùng kỹ thuật đa luồng và Socket để quản lý nội dung và danh sách người tham gia chat.
- Mỗi người đăng ký thành công vào Chatroom đều được tạo một luồng riêng để hoạt động.
- Danh sách người dùng được nạp từ tập tin và lưu vào đối tượng bằng băm để quản lý.  
ss
- Danh sách những người đang tham gia chat và các nội dung chat được lưu trong đối tượng kiểu VectorV

**1.3.** Sử dụng kỹ thuật đa luồng để thực hiện các phép toán cộng, trừ, nhân song song hai ma trận [5].

## Chương 2

# LẬP TRÌNH RMI VÀ PHÂN TÁN ĐỐI TƯỢNG

---

Chủ đề chính của chương II đề cập đến:

- ✓ Triệu gọi phương thức từ xa RMI
- ✓ Lập trình phân tán đối tượng, thiết lập và trao đổi tin giữa các đối tượng phân tán
- ✓ Sử dụng RMI và Applet
- ✓ Kiến trúc môi giới yêu cầu đối tượng chung CORBA

### 2.1. TRIỆU GỌI PHƯƠNG THỨC CỦA ĐỐI TƯỢNG TỪ XA RMI

Hơn ai hết, nếu đã từng đảm nhận công việc lập trình ứng dụng bạn sẽ hiểu được công việc nǎng nhoc khi phải viết mã lệnh cho từng đơn vị chương trình. Vậy thì tại sao chúng ta không tận dụng những cái có sẵn, trong khi có hàng trăm triệu dòng mã lệnh đã được thiết kế và xây dựng sẵn, phân tán ở trên mạng, đang hoạt động rất hiệu quả.

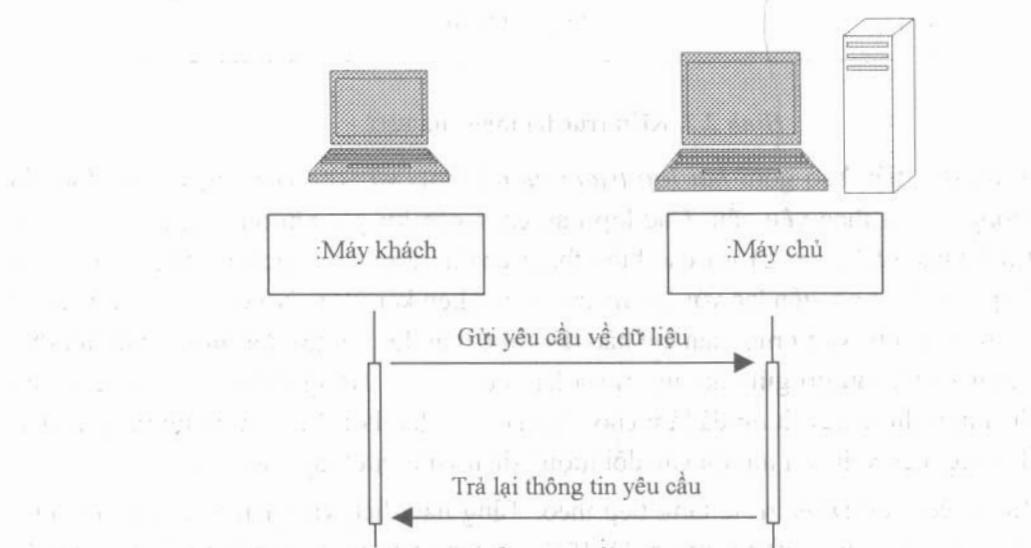
Ngày nay, cộng đồng những người lập trình bắt đầu nghĩ rằng “đối tượng có ở mọi nơi”, thường dưới dạng mã nguồn mở. Những đối tượng này, tất nhiên được hỗ trợ để trao đổi được với nhau theo những giao thức chuẩn trên mạng. Ví dụ, bạn có một đối tượng ở máy trạm (máy khách - Client), ở đó người sử dụng cần phải điền dữ liệu vào mẫu báo cáo. Đối tượng này (*đối tượng khách*) có thể gửi một thông điệp (thỉnh cầu) cho đối tượng trên máy chủ (*đối tượng phục vụ* - Server) trên mạng để yêu cầu cung cấp các thông tin cần thiết. Đối tượng trên máy chủ truy nhập vào các cơ sở dữ liệu (CSDL) để có được những thông tin mà khách yêu cầu và gửi lại cho đối tượng khách. Đó chính là ý tưởng của Java-RMI, *phương thức lập trình triệu gọi từ xa*. Lập trình phân tán đối tượng bằng cách *triệu gọi phương thức từ xa RMI* là cách hợp tác giữa các đối tượng Java có những mã lệnh cài đặt (bao gồm các phương thức và thuộc tính) nằm trên các máy khác nhau (chính xác là nằm trên các JVM – máy Java khác nhau) và có thể triệu gọi lẫn nhau để trao đổi tin [2].

Trong mô hình Client/Server truyền thống, các yêu cầu được dịch sang một format (dạng) trung gian (dạng từng cặp tên gọi / giá trị hoặc các dữ liệu trong XML). Máy chủ Server phân tích format yêu cầu, xử lý để có được kết quả và gửi trả lời cho máy

khách Client. Máy khách phân tích format trả lời và hiển thị thông tin cho người sử dụng.

### 2.1.1. Triệu gọi phương thức từ xa

Cơ chế triệu gọi từ xa có vẻ đơn giản hơn mô hình Client/Server. Nếu ta cần truy cập tới một đối tượng ở trên một máy khác thì ta chỉ cần triệu gọi phương thức của đối tượng ở trên máy đó. Tất nhiên, các tham số bằng cách nào đó phải được gửi đến cho máy kia và đối tượng nhận được phải chắc chắn thực hiện phương thức tương ứng để có được những thông tin cần thiết gửi trả lại cho đối tượng khách hàng.



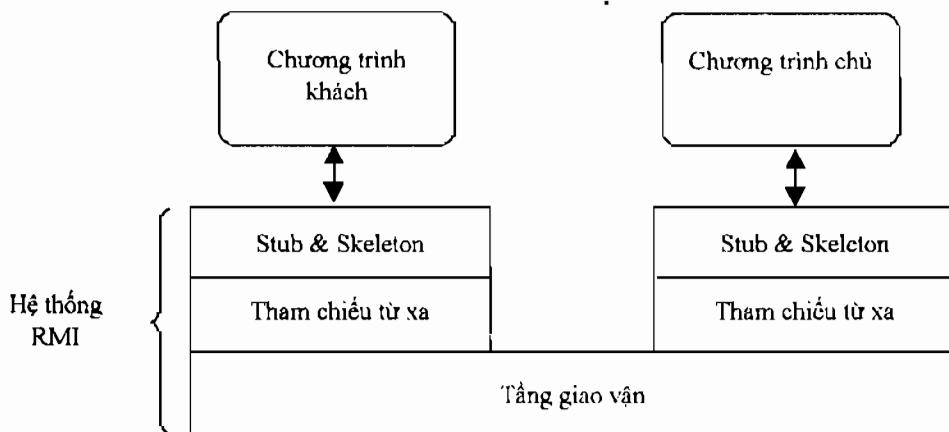
Hình 2.1. Sự trao đổi giữa đối tượng khách và phục vụ

Tuy nhiên, việc triệu gọi phương thức của các đối tượng từ xa chắc chắn sẽ phức tạp hơn nhiều so với lời gọi hàm ở trong cùng một chương trình ứng dụng. Như ở chương I đã phân tích, các đối tượng trên hai máy ảo khác nhau thì hoạt động trên hai tiến trình khác nhau, có hai không gian địa chỉ khác nhau, và vì thế việc tham chiếu tới các biến, địa chỉ đối tượng là hoàn toàn khác nhau. Lời gọi các hàm cục bộ thường luôn trả về kết quả sau khi thực hiện thành công, trong khi lời triệu gọi các thuộc tính biên từ xa phải thông qua kết nối mạng, có thể bị tắc nghẽn, bị ngắt do mạng gặp sự cố, v.v. Đối với lời gọi hàm trên máy cục bộ, các tham số truyền cho hàm thường đặt vào ngăn xếp, trong khi tham số truyền cho các phương thức của đối tượng ở xa phải được đóng gói và chuyển qua mạng theo giao thức chuẩn để đến được đích thông qua các đại diện trung gian.

### 2.1.2. Kiến trúc RMI Java

Mục đích của kiến trúc RMI là tạo ra một mô hình đối tượng phân tán và được tích hợp với ngôn ngữ lập trình Java. Kiến trúc này đã thành công trong việc tạo ra một hệ thống khá an toàn, kế thừa được sức mạnh của Java trong xử lý phân tán và đa luồng ([2], [3]).

Về cơ bản, RMI được xây dựng dựa trên *kiến trúc ba tầng* như hình 2.2.



Hình 2.2. Kiến trúc ba tầng của RMI

- **Tầng đầu tiên** bao gồm *hai lớp trung gian* Stub và Skeleton, chúng được hệ thống tạo ra theo yêu cầu. Các lớp này chặn các lời gọi phương thức của chương trình khách (Client) tới các biến tham chiếu và gửi tới dịch vụ triệu gọi từ xa. Lớp Skeleton liên lạc với Stub thông qua liên kết RMI. Nó đọc các tham số của lời triệu gọi từ xa từ một liên kết nào đó, thực hiện lời gọi tới đối tượng thực thi dịch vụ từ xa và sau đó gửi các giá trị trả lại cho Stub. Trong Java 2 SDK, các giao diện mới được xây dựng đã làm cho Skeleton lỗi thời. RMI sử dụng phép ánh xạ để thực hiện việc kết nối tới các đối tượng dịch vụ từ xa thay cho Skeleton.
- **Tầng tham chiếu từ xa** là tầng tiếp theo. Tầng này dịch và quản lý các tham chiếu tới các đối tượng dịch vụ từ xa. Ở JDK1.1, tầng này thực hiện kết nối theo cơ chế điểm – tới - điểm. Đến Java 2 SDK, tầng này được cải tiến để nâng cao việc hỗ trợ để kích hoạt các đối tượng dịch vụ từ xa đang chờ thực hiện thông qua ROA, đó là cách kết nối Client/Server.
- **Tầng giao vận** dựa trên kết nối TCP/IP giữa các máy tính trong mạng. Ngay cả khi hai chương trình chạy trên hai JVM trong cùng một máy, chúng cũng thực hiện kết nối thông qua TCP/IP của chính máy đó. Tầng giao vận RMI được thiết kế để thiết lập một kết nối giữa máy Client với máy Server.

Giả sử, ta có đối tượng C1 được cài đặt chạy trên máy phục vụ C. RMI của Java giúp ta tạo ra hai lớp trung gian C1\_Skel (không cần thiết đối với Java 2 SDK) và C1\_Stub. Lớp C1\_Stub sẽ được nạp về máy khách B. Khi đối tượng B1 trên máy B triệu gọi C1, máy ảo Java sẽ chuyển lời gọi đến lớp C1\_Stub. C1\_Stub sẽ chịu trách nhiệm đóng gói các tham số và chuyển chúng qua mạng đến cho máy C. Tại máy C, lớp C1\_Skel (C1\_Stub được nạp về và thay thế ở Java 2 SDK) sẽ nhận tham số để chuyển vào không gian địa chỉ tương thích với đối tượng C1 sau đó gọi phương thức tương ứng để thực hiện. Kết quả nếu có do phương thức của đối tượng C1 trả về sẽ được lớp C1\_Skel (C1\_Stub thay thế ở Java 2 SDK) đóng gói trả ngược cho C1\_Stub. C1\_Stub chuyển giao kết quả cuối cùng cho B1. Theo cơ chế này, có thể hình dung như B1 đang trao đổi trực tiếp với đối

tượng C1 ngay trên cùng một máy. Ngoài ra, với sự trợ giúp của lớp trung gian C1\_Stub, khi kết nối mạng gấp sự cố, lớp trung gian Stub sẽ luôn biết cách thông báo lỗi đến đối tượng B1.

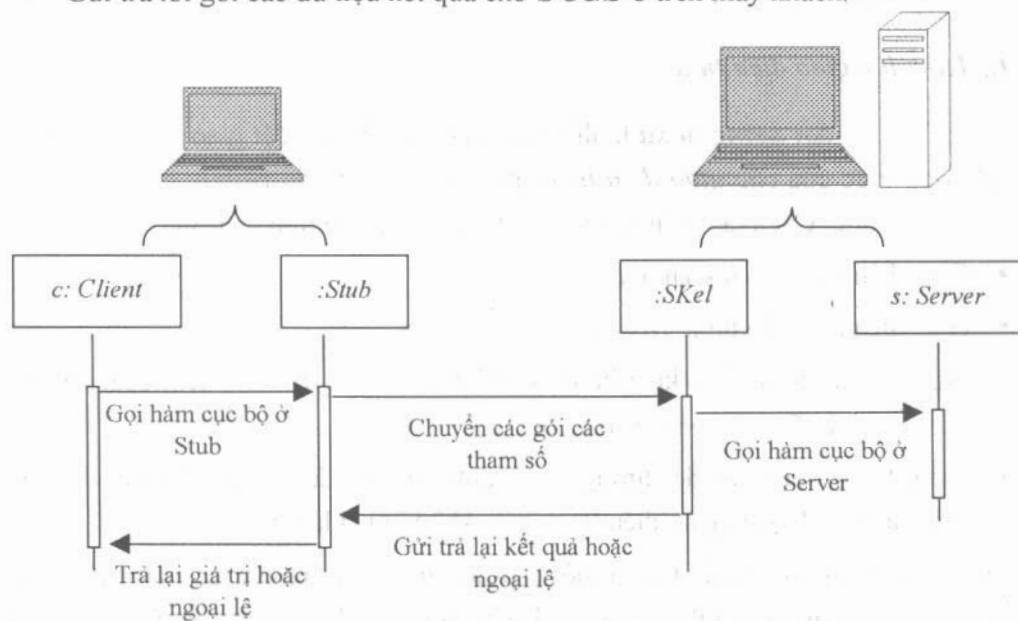
Thực tế có một câu hỏi là: Làm thế nào để B1 tham chiếu được đến C1 khi không có lớp C1 nào được cài đặt ở máy B? C1\_Stub trên máy B chỉ làm lớp trung gian chuyển đổi tham số và thực hiện các giao thức mạng, nó không phải là hình ảnh của đối tượng C1. Để làm được điều đó, lớp của đối tượng C1 cần cung cấp một giao diện tương ứng, được gọi là giao diện từ xa với các phương thức cho phép đối tượng B1 gọi được trên máy B.

Stub thường trực trên máy khách, không ở trên máy chủ. Nó có vai trò đóng gói các thông tin bao gồm:

- Định danh *đối tượng từ xa* cần sử dụng
- Mô tả về phương thức cần triệu gọi
- Mã hoá các tham số và truyền cho Skel.

Stub sẽ chuyển những thông tin trên cho máy chủ. Ở phía máy chủ, đối tượng Skel nhận thực hiện những công việc sau để gọi phương thức từ xa:

- Giải mã các tham số
- Xác định đối tượng để thực hiện lời gọi hàm tương ứng
- Thực hiện lời gọi phương thức theo yêu cầu
- Tập hợp kết quả để trả lời hoặc thông báo các lỗi ngoại lệ
- Gửi trả lời gói các dữ liệu kết quả cho Stub ở trên máy khách.



Hình 2.3. Sự trao đổi giữa đối tượng khách và phục vụ thông qua đối tượng trung gian

Hình trên mô tả quá trình tổ chức gói các tham số, các dữ liệu trả lời và sự trao đổi giữa các đối tượng trung gian trong kỹ thuật triệu gọi từ xa.

## 2.2. THIẾT LẬP MÔI TRƯỜNG TRIỆU GỌI TỪ XA

Để thực hiện được việc triệu gọi từ xa thì ta phải chạy chương trình ở trên cả hai máy, máy khách và máy chủ. Những thông tin cần thiết cũng phải được cài đặt tách biệt ở hai phía, máy khách và máy chủ. Tuy nhiên, không nhất thiết phải có hai máy tính riêng biệt. Nhờ có máy ảo Java, khi mở cửa sổ DOS-Prompt (môi trường DOS dưới Window) để chạy chương trình Java, chương trình này được xem như chạy trên một máy (ảo) JVM độc lập. Do đó, nếu hai chương trình Java chạy ở trong hai cửa sổ riêng thì có thể xem như là chúng thực hiện trên hai máy khác nhau.

**Ví dụ 2.1.** Chúng ta hãy bắt đầu từ một ví dụ đơn giản, chương trình trên máy chủ tạo ra hai sản phẩm: lò nướng bánh và lò vi sóng thuộc lớp Product. Sau đó chạy một chương trình trên máy khách để xác định thông tin về sản phẩm, giá bán của từng sản phẩm và in ra cho khách hàng.

**Lưu ý:** Những ví dụ này có thể chạy ở trên một máy đơn cũng như trên từng cặp máy kết nối mạng. Chúng ta có thể thực hiện theo cả hai kịch bản. Nhưng ngay cả khi bạn chạy trên một máy, thì cũng cần phải đảm bảo rằng các dịch vụ mạng là sẵn sàng. Đặc biệt phải chắc chắn rằng TCP/IP được hỗ trợ.

### 2.2.1. Trên máy phục vụ (Server)

#### (i) Thiết lập giao diện từ xa

Trong Java, *đối tượng từ xa* là thể hiện của một lớp cài đặt giao diện Remote. Tất cả các phương thức của *các giao diện từ xa* phải được khai báo public để các máy JVM khác có thể gọi được. Các giao diện từ xa phải đảm bảo những tính chất sau:

- Giao diện từ xa phải khai báo public.
- Giao diện từ xa kế thừa java.rmi.Remote.
- Mọi phương thức phải khai báo với mệnh đề throws để kiểm soát các ngoại lệ java.rmi.RemoteException.
- Kiểu dữ liệu của các đối tượng từ xa được truyền đi và giá trị nhận về phải được khai báo như là kiểu giao diện Remote, không phải là lớp.

Chúng ta trở lại ví dụ cần tạo ra một cặp đối tượng sản phẩm tại máy chủ (máy phục vụ) và máy khách. Chương trình trên máy khách cũng phải biết được nó cần cái gì ở những đối tượng trên máy phục vụ. Điều này thực hiện được nếu cả hai máy chia sẻ với một giao diện từ xa.

```

interface Product // Chia sẻ bởi Client và Server
    extends Remote{
    public String getDescription()
        throws RemoteException;
    public double getPrice()
        throws RemoteException;
}

```

Tất cả các phương thức của giao diện từ xa phải khai báo với throws RemoteException vì khi triệu gọi từ xa sẽ có rất nhiều nguyên nhân làm cho nó thất bại, ví dụ, máy chủ, kết nối mạng không sẵn sàng, và nhiều vấn đề không bình thường, ngoại lệ khác gặp phải trên mạng.

### **(ii) Xây dựng các lớp cài đặt các giao diện từ xa**

Ở phía máy chủ, chúng ta phải xây dựng lớp cài đặt các phương thức được khai báo trong giao diện từ xa.

Nói chung, các lớp các cài đặt đối tượng từ xa cần phải:

- Khai báo rằng nó cài đặt ít nhất một giao diện từ xa
- Định nghĩa toán tử tạo lập đối tượng từ xa
- Cài đặt các phương thức để có thể triệu gọi được từ xa.

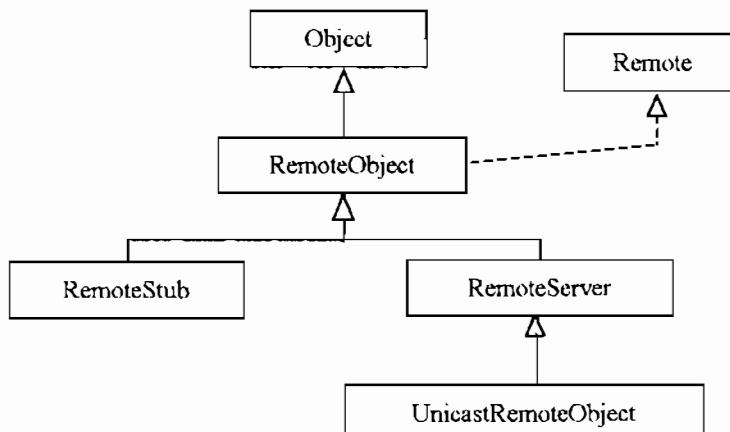
```

public class ProductImpl // Server
    extends UnicastRemoteObject implements Product{
    public ProductImpl(String s, double p) throws RemoteException {
        descr = s;    price = p;
    }
    public String getDescription() throws RemoteException {
        return descr ;
    }
    public double getPrice() throws RemoteException {
        return price ;
    }
    private String descr;
    private double price;
}

```

Lớp ProductImpl cài đặt hai hàm getDescription(), getPrice() cho phép gọi được từ xa trong chương trình khách.

Tất cả các lớp ở Server phải kế thừa từ lớp RemoteServer có ở trong gói java.rmi.server. Nhưng RemoteServer là lớp trừu tượng, nó định nghĩa các cơ chế cơ sở để trao đổi tin giữa các đối tượng dịch vụ và các đại diện của chúng từ xa. Lớp UnicastRemoteObject là lớp con của RemoteServer.



**Hình 2.4. Sơ đồ kế thừa các lớp đối tượng từ xa**

Lớp UnicastRemoteObject được sử dụng để tạo ra các đối tượng ở trên máy chủ. Đây là lớp cơ sở để xây dựng các lớp ứng dụng trao đổi thông tin từ xa trên máy chủ. Ngoài ra, ta có thể sử dụng lớp cơ sở MulticastRemoteObject để kế thừa, tạo ra những lớp ứng dụng chạy trên nhiều máy chủ đồng thời.

**Lưu ý:** Khi sử dụng RMI, vẫn đề đặt tên cho các lớp, giao diện là cần phải cần trọng. Để cho phù hợp, ta nên thực hiện theo những qui ước đặt tên như sau:

+ Không có hậu tố, ví dụ: Product	+ Giao diện từ xa
+ Có hậu tố Impl, ví dụ: ProductImpl	+ Lớp cài đặt giao diện
+ Có hậu tố Server, ví dụ: ProductServer	+ Chương trình tạo ra các đối tượng dịch vụ
+ Có hậu tố Client, ví dụ: ProductClient	+ Chương trình khách triệu gọi phương thức từ xa
+ Có hậu tố _Stub, ví dụ: ProductImpl_Stub	+ Lớp đại diện máy khách được tự động tạo ra bởi chương trình rmic
+ Có hậu tố _Skel, ví dụ: ProductImpl_Skel	+ Lớp đại diện máy chủ được tự động tạo ra bởi chương trình rmic

Bởi vì đối tượng xuất ra thường gặp phải ngoại lệ `java.rmi.RemoteException`, nên ta phải định nghĩa toán tử tạo lập với mệnh đề `throws RemoteException`, thậm chí cả khi toán tử tạo lập không làm gì. Nếu ta quên toán tử tạo lập thì chương trình dịch javac sẽ thông báo lỗi.

### (iii) Cài đặt các phương thức từ xa

Lớp cài đặt các đối tượng từ xa phải cài đặt tất cả các phương thức đã được khai báo trong giao diện từ xa. Ở ví dụ của chúng ta, phương thức từ xa phải cài đặt là:

```
public String getDescription()
    throws RemoteException{
    return descr;
}

public double getPrice()
    throws RemoteException{
    return price;
}
```

Các tham số, các giá trị trả lại của phương thức từ xa có thể là kiểu dữ liệu bất kỳ của Java, kể cả các đối tượng.

**Lưu ý:** Một lớp có thể định nghĩa những phương thức không được khai báo trong giao diện từ xa, được gọi là *phương thức cục bộ*. Những phương thức cục bộ chỉ được gọi trong cùng một ứng dụng (cùng một JVM), không triệu gọi từ xa được.

Sau khi hoàn tất lớp cài đặt, ta cần tạo ra các đại diện của lớp `ProductImpl` để mã hóa các tham số và nhận lại các kết quả của các lần triệu gọi phương thức từ xa. Người lập trình không sử dụng những lớp này trực tiếp và vì vậy cũng không phải tự viết chúng. Bộ công cụ rmic sẽ sinh ra chúng một cách tự động, ví dụ

```
C:\j2sdk1.4.0\bin>rmic -v1.2 ProductImpl
```

Kết quả là hai tệp lớp: `ProductImpl_Stub.class` và `ProductImpl_Skel.class` được sinh ra. Đối với Java 2 SDK thì tệp thứ hai không còn cần thiết nữa. Nếu lớp đặt trong một gói thì gọi rmic với tên của gói đó.

### (iv) Xác định các đối tượng dịch vụ

Để truy cập được đối tượng từ xa trên máy phục vụ, khách hàng cần có được đối tượng đại diện tại nơi đó. Khách yêu cầu đối tượng đó như thế nào? Phương thức chung là gọi phương thức từ xa của một đối tượng phục vụ và tạo ra một đối tượng đại diện để nhận kết quả trả lời.

Hệ thống RMI cung cấp một bộ đăng ký (RMI registry) đối tượng từ xa để ta kết hợp với tên được thiết lập theo URL dạng “`//host/objectname`” giúp ta xác định được đối tượng phục vụ. Tên gọi là dãy các ký tự (xâu) xác định duy nhất.

```
// Server
ProductImpl p1 = new ProductImpl("Lò nướng bánh", p);
Naming.bind("teaster");
```

Đối tượng p1 được ghi danh với tên gọi nhó “teaster” bằng hàm Naming.bind(), hoặc hàm Naming.rebind().

`java.rmi.server.Naming`

API

- `static Remote lookup(String url)`

Tìm đến đối tượng từ xa theo địa chỉ url. Nếu chưa có đối tượng được đăng ký thì phát sinh ngoại lệ NotBoundException.

- `static void bind(String name, Remote obj)`

Ghép (đóng gói) name với đối tượng từ xa obj. Nếu đối tượng đó đã có trong gói thì phát sinh ngoại lệ AlreadyBoundException.

- `static void unbind(String name)`

Mở để lấy name ra khỏi gói. Nếu name không có trong gói thì phát sinh ngoại lệ NotBoundException.

- `static void rebind(String name)`

Ghép (đóng gói) name với đối tượng từ xa obj. Cho phép thay thế những name đã được đóng gói.

- `static String[] list(String url)`

Lấy ra một danh sách các tên đối tượng (xâu) đã được đăng ký ở địa chỉ url.

Chương trình khách truy cập đến đối tượng phục vụ bằng cách chỉ ra tên của dịch vụ và tên đối tượng, sau đó ép về kiểu của giao diện từ xa như sau:

```
// Client
```

```
Product p = (Product)Naming.lookup("rmi://yourserver.com/teaster");
```

Đối tượng p ở trên máy khách muốn triệu gọi phương thức từ xa của đối tượng có tên được đăng ký là “teaster” ở máy phục vụ thì gọi hàm Naming.lookup() để truy tìm tham chiếu tới đối tượng đó từ xa.

RMI URL bắt đầu bằng “rmi:/” sau đó là Server, số hiệu cổng để lắng nghe (tùy chọn), dấu ‘/’ và sau đó là tên gọi của đối tượng triệu gọi từ xa. Ví dụ:

```
rmi://localhost:99/teaster
```

Số hiệu cổng mặc định là 1099.

Chương trình ở máy phục vụ Server được viết đầy đủ như sau.

```
// Product.java: giao diện từ xa Product
```

```
import java.rmi.*;
```

```
public interface Product extends Remote{
    public String getDescription()
        throws RemoteException;
    public double getPrice()
        throws RemoteException;
}

// ProductImpl.java: cài đặt lớp ProductImpl ở máy Server
import java.rmi.*;
import java.rmi.server.*;
public class ProductImpl extends
    UnicastRemoteObject implements Product{
    public ProductImpl(String s, double p)
        throws RemoteException{
        descr = s;
        price = p;
    }
    public String getDescription()
        throws RemoteException{
        return descr;
    }
    public double getPrice()
        throws RemoteException{
        return price;
    }
    private String descr;
    private double price;
}

// ProductServer.java: Chương trình phục vụ tạo ra 2 sản phẩm: Lò nướng bánh
// (có tên là teaster) và lò vi sóng (có tên là microwave).
import java.rmi.*;
import java.rmi.server.*;
import sun.applet.*;
public class ProductServer{
```

```

public static void main(String args[]) {
    try{
        System.out.println("Cài đặt dịch vụ ...");
        ProductImpl p1 = new ProductImpl("Lò nướng bánh", 200.5);
        ProductImpl p2 = new ProductImpl("Lò vi sóng", 350.2);
        System.out.println("Đăng ký dịch vụ ...");
        Naming.rebind("teaster", p1);
        Naming.rebind("microwave", p2);
        System.out.println("Cho máy khách triệu gọi ...");
    }catch(Exception e){
        System.out.println("Error: " + e);
    }
}
}

```

Tất cả các giao diện, lớp trên có thể gộp vào thành một gói. Sử dụng javac để dịch chúng và tạo ra các tệp lớp ( class) tương ứng trên thư mục hiện thời.

#### (v) Bộ đăng ký RMI registry

Chương trình dịch vụ của chúng ta hoàn toàn chưa sẵn sàng thực hiện. Vẫn để chính của chúng ta là cài đặt đối tượng của ProductImpl trên một máy và ở một máy khác (chương trình khác) muốn gọi phương thức getDescription() để biết được thông tin mô tả và giá bán của các sản phẩm có trên chương trình phục vụ. Như đã nói trước, ta không thể gọi trực tiếp hàm của ProductImpl mà phải thông qua các lớp trung gian Stub và Skel (không cần thiết đổi với Java 2 SDK). Ta thực hiện điều này nhờ trình biên dịch rmic như đã nêu ở trên.

Hơn nữa, máy khách muốn triệu gọi được phương thức ở xa thì trước tiên nó phải liên lạc được với bộ đăng ký RMI registry của máy ở xa đó. Các hàm giao diện của Java thực hiện nhiệm vụ này được gọi là các hàm giao diện API JNDI. Các hàm JNDI ở máy khách sẽ liên lạc với RMI registry của máy chủ để nhận tham chiếu của đối tượng trong khi các hàm JNDI ở máy chủ lại có nhiệm vụ đăng ký tên đối tượng với RMI registry. Để khởi động RMI registry dưới nền Window 95 hoặc NT thì phải thực hiện

```
start rmiregistry [port]
```

ở cửa sổ lệnh của DOS Prompt hoặc ở hộp thoại Run, trong đó port là số hiệu cổng giao diện để chờ phục vụ và trả lời kết quả. Và tất nhiên, khi thiết kế các đối tượng RMI, cụ thể là các đối tượng cài đặt chi tiết trên máy phục vụ, ta phải ghi nhớ lấy số hiệu cổng này cho khớp. Nếu không chỉ định số hiệu cổng thì RMI registry ngầm định lắng nghe ở

cổng 1099. Để khởi động registry ở một cổng khác, thì phải chỉ ra số hiệu của cổng đó trên dòng lệnh. Ví dụ, đăng ký ở cổng 2001:

```
start rmiregistry 2001
```

Một vấn đề khác đáng quan tâm ở đây nữa là RMI registry không cho phép đăng ký hai đối tượng cùng tên. Muốn chỉnh sửa lại hệ thống chương trình triệu gọi phương thức từ xa, ta phải, hoặc là khởi động lại chương trình RMI Registry, hoặc là ngay từ đầu khi thiết kế chương trình Server của đối tượng, ta sử dụng phương thức rebind() thay vì phương thức bind() để đăng ký với RMI registry của máy phục vụ.

#### *(vi) Chi tiết về RMI registry và các cách tự đăng ký đối tượng*

Bộ đăng ký RMI RMI registry đã được đề cập trên đây như một dịch vụ tìm kiếm đối tượng. Các đối tượng phục vụ muốn chương trình khách truy cập được từ xa thì phải đăng ký với RMI registry. Bộ đăng ký này là một chương trình dịch vụ chạy ở hậu trường, lắng nghe ở một cổng có số hiệu đã xác định. Hiện tại Java yêu cầu một máy ảo JVM chạy RMI registry và một máy JVM chạy chương trình Server (trên cùng một máy chủ).

Một dịch vụ RMI registry có thể tiếp nhận và quản lý cùng lúc nhiều đối tượng dịch vụ khác nhau. Java cho phép liên lạc với bộ đăng ký RMI registry để lấy về danh sách các đối tượng chủ mà nó đang quản lý thông qua phương thức list() của đối tượng đã đăng ký như ví dụ dưới đây.

```
//ListReg.java
import java.rmi.registry.*;
public class ListReg {
    public static void main(String[] args) throws Exception{
        //Địa chỉ của máy nơi mà RMI Registry đang chạy
        String hostName = "localhost";
        System.out.println("Connecting registry...");
        //Kết nối với bộ đăng ký
        Registry reg = LocateRegistry.getRegistry(hostName);
        //Lấy về danh sách các đối tượng do RMI Registry đang quản lý
        String objList[] = reg.list();
        System.out.println("Resgitry object: ");
        for (int i = 0; i < objList.length; i++)
            System.out.println(objList[i]);
    }
}
```

Java cho phép nhà lập trình tự tạo bộ đăng ký cho riêng mình mà không cần dùng đến rmiregistry.exe. Để tạo bộ đăng ký và tự đăng ký đối tượng, ta sử dụng phương thức `createRegistry()` của lớp `LocateRegistry`.

```
//CalculatorSetup.java
import java.rmi.server.*;
import java.rmi.*;
import java.rmi.registry.*;
public class ProductSetup {
    public static void main(String[] args) throws Exception {
        // Tạo bộ đăng ký registry
        LocateRegistry.createRegistry(2510);
        // Tạo đối tượng
        String name = "Máy in";
        double p = 150.0;
        ProductImpl product = new ProductImpl(name, p);
        // Yêu cầu JVM nhận dạng product
        UnicastRemoteObject.exportObject(product);
        // Đăng ký product với dịch vụ truy cập
        System.out.println("registering object ...");
        Naming.rebind("rmi://localhost:2510/printer", product);
        System.out.println("waiting for client request ...");
    }
}
```

Biên dịch và cho thực thi chương trình. Khi chương trình hoạt động, nó tương đương với hai chương trình trước kia: rmiregistry.exe và ProductServer.

### 2.2.2. Trên máy khách (Client)

Bây giờ ta hãy viết chương trình ở trên máy khách (chương trình khách) để yêu cầu chương trình phục vụ cung cấp những thông tin về các sản phẩm từ các đối tượng đã đăng ký.

Với Java, tất cả các thao tác kết nối và sao chép các tệp tin từ một máy tính về máy khách đều phải thông qua lớp bảo vệ gọi là RMISecurityManager.

Ví dụ, nếu bạn muốn cho chương trình khách ProductClient có thể nạp tự động `ProductImpl_Stub.class` từ Webserver, thì phải thiết lập lại cơ chế bảo vệ an ninh ở máy khách.

```
System.setSecurityManager(new RMISecurityManager());
```

Hệ thống an ninh RMISecurityManager sẽ sử dụng các quyền được thiết lập trong tệp chính sách `jre\lib\security\java.policy` để kiểm soát việc kết nối từ xa (`jre` là thư mục chứa các tài nguyên tạo nên môi trường thực thi của Java).

Để chương trình khách kết nối được với RMI registry và đối tượng phục vụ, ta cần sự hỗ trợ của tệp chính sách (tệp thường có đuôi `.policy`). Ở đây, tệp chính sách cho phép một chương trình ứng dụng tạo ra sự kết nối mạng qua các cổng có số hiệu ít nhất là 1024. Cổng RMI mặc định là 1099 và Server có thể sử dụng các cổng  $\geq 1024$ . Ta có thể soạn tệp `client.policy` cho phép kết nối các cổng  $\geq 1024$ :

```
grant {
    permission java.net.SocketPermission "*:1024-65535", "connect";
}
```

Khi thực hiện chương trình khách, ta phải sử dụng chính sách (`client.policy`) (chi tiết xem [2]).

```
java ProductClient -Djava.security.policy=client.policy
```

Chương trình ở máy khách được viết như sau.

```
// ProductClient.java: lớp ProductClient trên máy khách
import java.rmi.*;
import java.rmi.server.*;
public class ProductClient{
    public static void main(String args[]){
        System.setSecurityManager(new RMISecurityManager());
        String url = "rmi://localhost/";
        String name = "";
        double price = 0.0;
        // Thay bằng "rmi://yourserver.com"
        // Khi Server chạy trên máy từ xa yourServer.com
        try{
            Product c1 = (Product)Naming.lookup(url + "teaster");
            Product c2 = (Product)Naming.lookup(url + "microwave");
            name = c1.getDescription();
            price = c1.getPrice();
            System.out.println(name + ", giá bán: $" + price);
            price = c2.getPrice();
            name = c2.getDescription();
        }
    }
}
```

```

        price = c2.getPrice();
        System.out.println(name + ", giá bán: $" + price);
    } catch (Exception e) {
        System.out.println("Error " + e);
    }
    System.exit(0);
}
}

```

### **Biến tùy chọn CODEBASE**

Như ta đã biết, khi thực thi, một chương trình Java dựa vào biến môi trường CLASSPATH để tìm các tệp tin mã byte code: \*.class. Có nhiên, CLASSPATH giữa máy chủ và máy khách hoàn toàn có thể được thiết lập khác nhau. Ta phải biết rõ, các chương trình trong hệ thống muốn thực thi được thì cần phải có CLASSPATH phù hợp, chính xác hơn là cần cung cấp những tệp lớp .class, tệp thực thi ở thư mục nào, ở đâu để có thể thực thi chúng.

Chẳng hạn, trong ví dụ của chúng ta, máy phục vụ cần phải có các tệp tin mã byte code ở thư mục C:\server\product\Product.class, ProductImpl.class, ProductImpl\_Stub.class, ProductImpl\_Skel.class (tệp này không cần thiết ở Java 2 SDK), và ProductServer.class. Trong khi đó, tại máy khách ta chỉ cần có hai tệp tin ở thư mục D:\Client\product\ProductClient.class và ProductImpl\_Stub.class, còn bộ chương trình JDK được cài đặt ở thư mục C:\j2sdk1.4.0\bin.

Thực tế, chúng ta thấy, cách làm việc trên là chưa thể hiện rõ tính phân tán đối tượng một cách triệt để. Thật vậy, khi mà lớp đối tượng triệu gọi từ xa phải cập nhật thì các lớp giao diện trung gian Skel và Stub của nó cũng phải được cập nhật theo. Tuy nhiên, ở phía máy khách cũng phải cập nhật phiên bản Stub mới này, điều này dẫu sao cũng gây nên ít nhiều bất tiện. Công nghệ Java cung cấp giải pháp cho phép tự động nạp lớp Stub từ xa thông qua tùy chọn CODEBASE khi đăng ký đối tượng với RMI registry trên máy chủ. Chẳng hạn, ta đăng ký ProductServer như sau:

```
c:\j2sdk1.4.0\bin>java -Djava.rmi.server.codebase=
    "http://192.168.0.9/rmi/product/" ProductServer
```

Khi đăng ký như vậy, Server của chúng ta (ở đây là máy có địa chỉ IP là 192.168.0.9), ngoài RMI registry đang hoạt động, cần phải chạy thêm dịch vụ Web, và phải bảo đảm là tại địa chỉ http://192.168.0.9/rmi/product/ thì luôn có tệp tin mã byte code \_Stub.class. Bằng cách này, nhà cung cấp đối tượng triệu gọi từ xa chỉ cần cung cấp tệp tin mã byte code của đối tượng mà thôi. Và, khi máy khách có yêu cầu RMI registry trả về tham chiếu của đối tượng, nếu máy khách chưa có lớp Stub thì RMI

registry sẽ hướng dẫn máy khách tự động nạp lớp này từ địa chỉ xác định trong tùy chọn CODEBASE.

Trong ví dụ 2.1, ta giả sử tất cả các tệp Product\*.\* và client.policy được đặt ở D:\user\product, còn Java 2 SDK cài đặt ở C:\j2sdk1.4.0. Ta thực hiện các bước lần lượt như sau:

1. Dịch tất cả các tệp nguồn Java bắt đầu bằng Product

```
D:\user\product>C:\j2sdk1.4.0\bin\javac Product*.java
```

2. Tạo ra các lớp trung gian

```
D:\user\product>C:\j2sdk1.4.0\bin\rmic -v1.2 ProductImpl
```

3. Khởi động bộ đăng ký RMI registry

```
D:\user\product>start C:\j2sdk1.4.0\bin\rmiregistry
```

4. Bắt đầu thực hiện chương trình phục vụ

```
D:\user\product>start C:\j2sdk1.4.0\bin\java ProductServer
```

5. Thực hiện chương trình khách

```
D:\user\product>C:\j2sdk1.4.0\bin\java -
```

```
Djava.security.policy=client.policy ProducClient
```

Kết quả sẽ in ra:

Lò nướng bánh, giá bán: 200.5 \$

Lò vi sóng, giá bán: 350.2 \$

**Lưu ý:** Ta có thể thiết lập biến môi trường bằng cách đặt đường dẫn để khi thực hiện không cần phải chỉ định như trên, ví dụ

```
set classpath=D:\server\product;D:\client\product;c:\j2sdk1.4.0\bin
```

### 2.2.3. Triển khai ứng dụng Web

Triển khai một ứng dụng mà ta sử dụng RMI thì phải thận trọng vì nhiều vấn đề có thể xảy ra. Các bước thực hiện ở trên tạo ra nhiều tệp lớp (đuôi .class) tách biệt và chúng có thể được đặt ở ba thư mục con: server, download và client.

- Thư mục server chứa tất cả các tệp cần thiết để chạy chương trình ở Server. Trong ví dụ nêu trên có các tệp lớp: ProductServer.class, Product.class, ProductImpl.class, ProductImpl\_Stub.class.
- Thư mục download chứa tất cả các tệp sẽ được nạp về máy khách cũng như những lớp phụ thuộc. Trong ví dụ ở trên, ta cần bổ sung Product.class, ProductImpl\_Stub.class vào thư mục download.

- Cuối cùng ở thư mục client chứa tất cả các tệp cần thiết để bắt đầu chạy chương trình client. Trong ví dụ ở trên, đó là các tệp lớp: ProductClient.class, Product.class và client.policy.

Giả sử Web Server được khởi động trên máy tính của bạn. Bạn cũng có thể nhận được dịch vụ đó từ <ftp://java.sun.com/pub/jdk1.1/rmi/class-server.zip>. Dịch vụ này tương đối dễ cài đặt và có đủ các chức năng hỗ trợ các tệp lớp.

- + Trước tiên, chuyển thư mục download vào thư mục tư liệu của Web Server.
- + Sau đó, soạn thảo lại tệp client.policy. Nó phải cho phép chương trình khách những quyền sau:
- Kết nối qua các cổng 1024, đến được với RMI registry và những cài đặt ở Server.
- Kết nối với cổng HTTP (thường là 80) để nạp các tệp lớp trung gian Stub.
- Tệp client.policy cấp quyền kết nối mạng, sau khi được thay đổi và có dạng:

```
grant{
    permission java.net.SocketPermission
        "*:1024-65535", "connect";
    permission java.net.SocketPermission
        "*:80", "connect";
};
```

### 2.3. TRUYỀN THAM SỐ TRONG CÁC LỜI GỌI PHƯƠNG THỨC TỪ XA

Trong một chương trình Java, các biến kiểu đối tượng thì được truyền theo tham chiếu trong các lời gọi hàm. Nghĩa là khi đối tượng truyền cho phương thức bị thay đổi bên trong thân phương thức thì khi lời gọi phương thức kết thúc, giá trị các thành phần của đối tượng cũng bị thay đổi theo. Chẳng hạn ta xét sự hoạt động của chương trình sau đây:

```
class Number {
    public int value = 0;
    public Number(int v) {
        this.value = v;
    }
}
public class App {
    public static void main(String[] args) {
        Number num = new Number(12);
```

```

        System.out.println("Giá trị trước khi gọi hàm: " +
                           num.value);
        incNum(num);
        System.out.println("Giá trị sau khi gọi hàm: " + num.value);
    }

    public static void incNum(Number n) {
        n.value++;
    }
}

```

Trong ví dụ này, ta tạo ra lớp Number lưu giữ giá trị nguyên. Phương thức incNum() tiếp nhận tham số có kiểu đối tượng là Number với giá trị value khởi đầu là 12. Bên trong phương thức, thành phần của biến đối tượng (tăng value lên 1). Kết quả là sau khi lời gọi phương thức kết thúc, đối tượng num có giá trị value thay đổi (tăng lên 1).

Cũng cần nên biết rằng Java có hai kiểu dữ liệu chính: *kiểu dữ liệu nguyên thuỷ* và *kiểu dữ liệu phức hợp*. Các kiểu dữ liệu nguyên thuỷ, như int, float, double, char, byte, long, ... khi truyền cho các hàm thì chúng được xem như là *các tham trị*; ngược lại, các kiểu dữ liệu phức hợp, được xem là các đối tượng, thì luôn được truyền cho các hàm theo *tham chiếu*. Nghĩa là, khi sửa đổi chương trình nói trên về dạng các tham số truyền vào các lời gọi hàm là dữ liệu kiểu nguyên thuỷ thì giá trị của đối số truyền cho hàm sẽ không thay đổi sau khi thực hiện lời gọi hàm vì chúng được gọi theo tham trị.

```

public class App2 {

    public static void main(String[] args) {
        int num = 12;
        System.out.println("Giá trị trước khi gọi hàm: " + num);
        incNum(num);
        System.out.println("Giá trị sau khi gọi hàm: " + num);
    }

    public static void incNum(int n)
        n++;
    }
}

```

Giá trị sau khi gọi hàm vẫn là 12.

Tóm lại, trong Java, *đối tượng được truyền theo tham chiếu* còn các *kiểu dữ liệu nguyên thuỷ* thì *được truyền theo tham trị*. Vấn đề của chúng ta ở đây là truyền tham số qua các lời gọi phương thức từ xa thì được xem là truyền theo *kiểu tham chiếu* hay *tham trị*. Thật

sự, truyền tham số cho các lời gọi phương thức từ xa trong RMI có hơi khác so với nguyên tắc truyền tham số theo kiểu thông thường:

1. Tất cả các kiểu dữ liệu *nguyên thuỷ* đều được *truyền theo tham trị*.
2. Tất cả các kiểu dữ liệu kiểu lớp muốn truyền qua mạng đều buộc phải cài đặt một trong hai giao diện Remote hoặc Serializable. Các đối tượng cài đặt giao diện Remote *sẽ được truyền theo tham chiếu*, còn các đối tượng cài đặt theo giao diện Serializable *sẽ được truyền theo tham trị*.

**Lưu ý:** Nếu trong các lời gọi phương thức RMI, kiểu dữ liệu đối tượng nếu không cài đặt một trong hai giao diện Remote hoặc Serializable thì sẽ không thể dùng làm tham số chuyền qua mạng được.

### 2.3.1. Truyền đối tượng đến trình chủ theo tham trị

Các lớp đối tượng thông dụng của Java như String, Date, Time, ... đều được cài đặt giao diện Serializable cho nên chúng được chuyển cho các lời gọi hàm hay phương thức ở xa theo kiểu tham trị.

Với cơ chế truyền tham số đối tượng theo tham trị, khi gọi một phương thức của đối tượng từ xa, nếu trong lời gọi của phương thức này có yêu cầu tham số là kiểu đối tượng, đối tượng sẽ được đóng gói và chuyển toàn bộ đến máy chủ (nơi tiếp nhận tham số và thực thi phương thức). Tại máy chủ, đối tượng sẽ được khôi phục lại trạng thái ban đầu và đưa vào sử dụng. Quá trình đóng gói tham số được thực hiện bởi lớp trung gian \_Stub, ngược lại, quá trình mở gói dữ liệu để khôi phục lại tham số sẽ được thực hiện bởi lớp \_Skel.

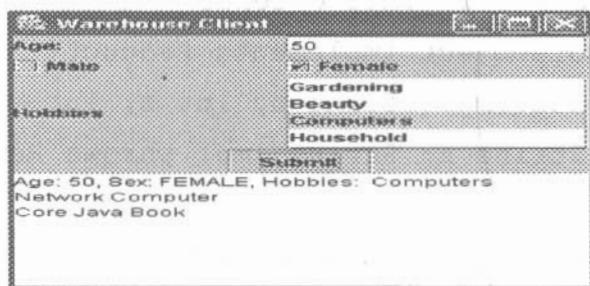
### 2.3.2. Chuyển đổi tương đến chương trình chủ theo tham chiếu

Đối với lập trình trên một máy cục bộ, việc truyền một đối tượng có kích thước lớn cho một hàm theo kiểu tham trị không phải là một giải pháp hay do vừa tốn bộ nhớ, vừa tốn thời gian tạo bản sao của đối số. Trong kỹ thuật lập trình triệu gọi từ xa cũng vậy, với một đối tượng có kích thước lớn, việc đóng gói toàn bộ đối tượng rồi chuyển đi chuyển lại trên mạng sẽ ảnh hưởng đến tốc độ thực thi của chương trình. Có cách nào mà trình chủ có thể tham chiếu và xử lý được trực tiếp đối tượng đang nằm trên máy khách hay không? Có nghĩa là, nếu trình chủ được trình khách truy xuất từ xa thì chính chương trình khách cũng có thể được gọi từ xa bởi trình chủ. Hay nói cách khác, trình khách không cần chuyển đổi đối tượng cho trình chủ theo trị mà có thể chuyển theo tham chiếu. Bằng cách này ta có thể cho đối tượng trên trình khách và trình chủ triệu gọi lẫn nhau.

Cách mà chương trình khách có thể tham chiếu được đến đối tượng trên máy chủ là máy chủ yêu cầu phải thực thi giao diện Remote (chẳng hạn ProductImpl extends Remote), tiếp đến là sinh ra các lớp trung gian \_Stub và \_Skel, sau cùng là dùng RMI registry để trình khách tham chiếu đến. Chúng ta sẽ áp dụng điều tương tự như vậy với

các đối tượng trên trình khách. Với Java, các đối tượng nếu cài đặt giao diện Remote thì sẽ được xem như là có khả năng chuyển qua mạng thông qua tham chiếu chứ không cần phải đóng gói chuyển đi (tạo ra bản copy) theo tham trị như trường hợp cài đặt giao diện Serializable. Đơn giản chỉ có vậy!

**Ví dụ 2.2.** Chương trình sau chỉ ra cách sao chép các tham biến và các giá trị trả lại khi thực hiện cài đặt với giao diện Serializable. Chương trình giới thiệu các sản phẩm từ xa (marketing trên mạng) theo sở thích, độ tuổi và để khách hàng lựa chọn tùy theo họ là nam hay nữ. Khách hàng có thể cho biết độ tuổi, chọn mục nam (Male) hay nữ (Female), có thể cả hai và sở thích (Hobbies) rồi nhấn nút “Submit” để có được những sản phẩm mà khách hàng giới thiệu. Trường hợp khách tầm tuổi 50, là nữ, sở thích là máy tính (chọn Computers) thì sẽ được giới thiệu hai cuốn sách như hình 2.5.



Hình 2.5. Giới thiệu sản phẩm từ máy chủ

Để thực hiện được những công việc trên, ta phải xây dựng hai giao diện từ xa: Product, Warehouse, các lớp từ xa ProductImpl, WarehouseImpl, WarehouseServer, WarehouseClient và một lớp cục bộ Customer.

Một đối tượng thuộc lớp Customer được gửi đến cho Server. Bởi vì Customer không phải là đối tượng từ xa, vì thế một bản sao của nó sẽ được tạo ra ở Server. Chương trình Server sẽ gửi trả lại một Vector (một dãy) các sản phẩm phù hợp với sở thích để khách hàng lựa chọn.

Trước tiên ta tạo ra giao diện từ xa cho sản phẩm.

```
// Product.java khai báo giao diện từ xa Product
import java.rmi.*;
import java.awt.*;
public interface Product extends Remote{
    public String getDescription()
        throws RemoteException;
    static final int MALE = 1;
    static final int FEMALE = 2;
    static final int BOTH = MALE + FEMALE;
}
```

Thông tin về sản phẩm được lưu trữ bao gồm: mô tả sản phẩm, cho lớp người (nam/nữ) ở phạm vi độ tuổi nào đó và phù hợp cho những sở thích nhất định. Lớp ProductImpl bên cạnh việc cài đặt phương thức `getDescription()` như đã khai báo trong giao diện từ xa, được gọi là *phương thức từ xa*, còn cài đặt phương thức `match()`, `getImageFile()` không có trong giao diện, được gọi là *phương thức cục bộ*. Phương thức này chỉ cho phép gọi cục bộ trong một chương trình, không cho phép triệu gọi từ xa.

```
// ProductImpl.java cài đặt giao diện từ xa và bổ sung các phương thức cục bộ
import java.rmi.*;
import java.rmi.server.*;
import java.awt.*;
public class ProductImpl extends
    UnicastRemoteObject implements Product{
    public ProductImpl(String n, int s, int age1, int age2,
                       String h, String img) throws RemoteException{
        name = n; sex = s;
        ageLow = age1; ageHig = age2;
        hobby = h; imageFile = img;
    }
    public String getDescription() throws RemoteException{
        return name;
    }
    public String getImageFile() throws RemoteException{
        return imageFile;
    }
    public boolean match(Customer c){
        if(c.getAge() < ageLow || c.getAge() > ageHig)
            return false;
        if(!c.hasHobby(hobby)) return false;
        if((sex & c.getSex()) == 0) return false;
        return true;
    }
    private String name;
    private int ageLow;
    private int ageHig;
    private int sex;
```

```

    private String hobby;
    private String imageFile;
}

```

Tiếp theo, ta xây dựng lớp Customer cài đặt giao diện Serializable, không phải là lớp từ xa. Nghĩa là không một phương thức nào của nó có thể được triệu gọi từ xa. Tuy nhiên, các đối tượng của lớp này có thể chuyển từ một máy JVM này sang máy JVM khác.

```

// Customer.java tạo ra một lớp Customer cục bộ
import java.io.*;
public class Customer implements Serializable{
    public Customer(int theAge, int theSex, String[] theHobbies){
        age = theAge; sex = theSex;
        hobbies = theHobbies;
    }
    public int getAge(){
        return age;
    }
    public int getSex(){
        return sex;
    }
    public boolean hasHobby(String aHobby){
        if(aHobby == "") return true;
        for(int i = 0; i < hobbies.length; i++)
            if(hobbies[i].equals(aHobby)) return true;
        return false;
    }
    public void reset(){
        age = sex = 0;
        hobbies = null;
    }
    public String toString(){
        String res = "Age: " + age + ", Sex: ";
        if(sex == Product.MALE) res += "MALE";
        else if(sex == Product.FEMALE) res += "FEMALE";
    }
}

```

```

        else res += "FEMALE OR MALE";
        res += ", Hobbies: ";
        for(int i = 0; i < hobbies.length; i++)
            res += " " + hobbies[i];
        return res;
    }

    private int age;
    private int sex;
    private String[] hobbies;
}

```

Tương tự như đối với Product, ta tạo ra tiếp giao diện từ xa Warehouse.

```

// Warehouse.java khai báo giao diện từ xa cho kho hàng để tìm kiếm hàng theo yêu cầu.
import java.rmi.*;
import java.util.*;
public interface Warehouse extends Remote{
    public Vector find(Customer c) throws RemoteException;
}

```

Giống như lớp ProductImpl, WarehouseImpl cũng sẽ có phương thức cục bộ và phương thức từ xa. Phương thức cục bộ add() được sử dụng để bổ sung (mua thêm) các sản phẩm vào kho hàng. Phương thức find() là từ xa, được triệu gọi để tìm các sản phẩm tương ứng trong kho hàng.

```

// WarehouseImpl.java cài đặt một lớp từ xa có cả các hàm cục bộ.
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
public class WarehouseImpl extends
    UnicastRemoteObject implements Warehouse{
    public WarehouseImpl() throws RemoteException{
        products = new Vector();
    }
    public synchronized void add(ProductImpl p) { // Local method
        products.add(p);
    }
    public synchronized Vector find(Customer c) throws RemoteException{
        Vector res = new Vector();

```

```
    for(int i = 0; i < products.size(); i++){
        ProductImpl p = (ProductImpl)products.get(i);
        if(p.match(c)) res.add(p);
    }
    res.add(new ProductImpl("Core Java Book", Product.BOTH, 20, 100,
                           "", "corejava.jpg"));
    c.reset();
    return res;
}
private Vector products;
}
```

Chương trình WarehouseServer.java nhằm tạo ra các đối tượng từ xa và đăng ký chúng để cho phép triệu gọi từ xa.

```
// WarehouseServer.java tạo ra lớp dịch vụ từ xa.

import java.rmi.*;
import java.rmi.server.*;
public class WarehouseServer{
    public static void main(String args[]){
        try{
            System.out.println("Constructing server implementations ...");
            WarehouseImpl w = new WarehouseImpl();
            fillWarehouse(w);
            System.out.println("Binding server implementations to registry...");
            Naming.rebind("central_warehouse", w);
            System.out.println("Waiting for invocations from clients ...");
        }catch(Exception e){
            System.out.println("Error: " + e);
        }
    }
    public static void fillWarehouse(WarehouseImpl w) throws RemoteException{
        w.add(new ProductImpl("Blackwell Teaster", Product.BOTH,
                18, 100, "Household",""));
        w.add(new ProductImpl("Cosmetic Set", Product.FEMALE,
                15, 50, "Beauty",""));
    }
}
```

```

        w.add(new ProductImpl("Learn Java in 21 Days Book",
            Product.BOTH, 20, 100, "Computers",""));
        w.add(new ProductImpl("Handy Hand Grenade", Product.MALE,
            16, 60, "Gardening",""));
        w.add(new ProductImpl("Network Computer", Product.BOTH,
            18, 80, "Computers",""));
        // Tương tự có thể bổ sung nhiều mặt hàng khác
    }
}

```

Cuối cùng là chương trình khách. Sau khi đã cho biết tuổi (Age), và lựa chọn mục nam (Male), hay nữ (Female), có thể cả hai và sở thích (Hobbies), người sử dụng nhấn nút “Submit” thì một đối tượng mới được tạo ra và được truyền vào cho phương thức từ xa find(). Những thông tin về sản phẩm tìm thấy sẽ được hiển thị cho khách hàng lựa chọn.

```

//WarehouseClient.java triệu hồi từ xa để có được các sản phẩm phù hợp với khách hàng
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.rmi.*;
import java.rmi.server.*;
import javax.swing.*;
public class WarehouseClient{
    public static void main(String args[]){
        JFrame fr = new WarehouseClientFrame();
        fr.show();
    }
}
class WarehouseClientFrame extends JFrame
    implements ActionListener{
public WarehouseClientFrame(){
    initUI();
    System.setSecurityManager(new RMISecurityManager());
    try{
        String url = "rmi://localhost/central_warehouse";

```

```
        centralWarehouse = (Warehouse) Naming.lookup(url);
    }catch(Exception e){
        System.out.println("Error: Can't connect to warehouse! " + e);
    }
}

private void callWarehouse(Customer c){
    try{
        Vector rec = centralWarehouse.find(c);
        result.setText(c + "\n");
        for(int i = 0; i < rec.size(); i++){
            Product p = (Product)rec.get(i);
            String t = p.getDescription() + "\n";
            result.append(t);
        }
    }catch(Exception e){
        result.setText("Error: " + e);
    }
}

public void actionPerformed(ActionEvent evt){
    Object[] hobbyObjects = hobbies.getSelectedValues();
    String[] hobbyStrings = new String[hobbyObjects.length];
    System.arraycopy(hobbyObjects, 0, hobbyStrings, 0,
                      hobbyObjects.length );
    Customer c = new Customer(Integer.parseInt(age.getText()),
                               (male.isSelected())? Product.MALE : 0
                               + (female.isSelected())? Product.FEMALE : 0 ,
                               hobbyStrings);
    callWarehouse(c);
}

private void initUI(){// Khởi tạo màn hình giao diện cho khách hàng
    setTitle("Warehouse Client");
    setSize(300, 300);
    addWindowListener(new WindowAdapter(){
        public void WindowClosing(WindowEvent e){
```

```
        System.exit(0);
    }
});

getContentPane().setLayout(new GridBagLayout());

GridBagConstraints gbc = new GridBagConstraints();
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.weightx = 100;
gbc.weighty = 0;

add(new JLabel("Age:"), gbc, 0, 0, 1, 1);
age = new JTextField(4);
age.setText("20");
add(age, gbc, 1, 0, 1, 1);

male = new JCheckBox("Male", true);
add(male, gbc, 0, 1, 1, 1);
female = new JCheckBox("Female", true);
add(female, gbc, 1, 1, 1, 1);

gbc.weighty = 100;
add(new JLabel("Hobbies"), gbc, 0, 2, 1, 1);
String[] choices = {"Gardening", "Beauty", "Computers", "Household"};

gbc.fill = GridBagConstraints.BOTH;
hobbies = new JList(choices);
add(new JScrollPane(hobbies), gbc, 1, 2, 1, 1);

gbc.weighty = 0;
gbc.fill = GridBagConstraints.NONE;
JButton submitButton = new JButton("Submit");
add(submitButton, gbc, 0, 3, 2, 1);
submitButton.addActionListener(this);
gbc.weighty = 100;
```

```

        gbc.fill = GridBagConstraints.BOTH;
        result = new JTextArea(4,40);
        result.setEditable(false);
        add(result, gbc, 0, 4, 2,1);
    }

    private void add(Component c, GridBagConstraints gbc,
                     int x, int y, int w, int h){
        gbc.gridx = x;
        gbc.gridy = y;
        gbc.gridwidth = w;
        gbc.gridheight = h;
        getContentPane().add(c, gbc);
    }

    private Warehouse centralWarehouse;
    private JTextField age;
    private JCheckBox male, female;
    private JList hobbies;
    private JTextArea result;
}

```

Việc tạo ra tệp client.policy và năm bước: dịch, tạo ra lớp trung gian, đăng ký RMI registry rồi thực hiện chương trình phục vụ, chương trình khách được thực hiện tương tự như ở ví dụ 2.1 nêu trên.

### 2.3.3. Truyền gọi đối tượng từ xa

Truyền các đối tượng từ xa từ chương trình phục vụ (Server) tới chương trình khách (Client) là khá đơn giản. Chương trình khách nhận được đối tượng trung gian Stub và lưu vào biến đối tượng có cùng kiểu giao diện từ xa với đối tượng được truyền đi. Client có thể truy cập vào đối tượng hiện thời ở trên Server thông qua biến đó. *Nên nhớ là chỉ có các phương thức từ xa (khai báo trong giao diện từ xa, giao diện kế thừa từ Remote) mới được phép truy cập trong các đối tượng Stub.* Tất cả các phương thức cục bộ (không khai báo trong giao diện từ xa) không truy cập được bởi Stub.

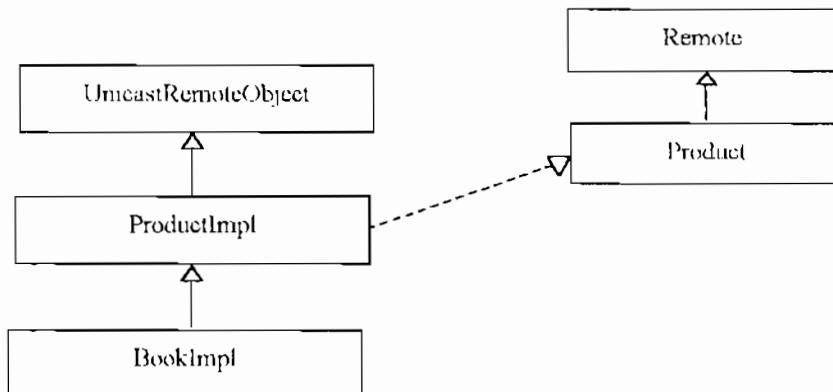
Trường hợp một lớp con không cài đặt giao diện từ xa, nhưng lớp cha của nó lại cài đặt giao diện từ xa, và đối tượng của lớp con được truyền gọi với phương thức từ xa thì chỉ những phương thức của lớp cha là được chấp nhận. Để hiểu rõ hơn vấn đề này, ta hãy xét ví dụ sau.

```

class BookImpl extends ProductImpl{
    public BookImpl(String title, String theISBN, int sex,
                   int age1,int age2, String hobby, String img){
        super(title + " Book", sex, age1, age2, hobby, img);
        ISBN = theISBN;
    }
    public String getStockCode(){
        return ISBN;
    }
    private String ISBN;
}

```

Giả sử ta truyền một đối tượng book của lớp BookImpl với lời gọi phương thức getStockCode() từ xa. Đối tượng nhận được sẽ là một đối tượng Stub. Nhưng đối tượng Stub này không đại diện cho book mà nó chỉ đại diện được cho đối tượng của lớp cha là ProductImpl. Mọi quan hệ giữa các giao diện và các lớp được mô tả như hình 2.5.



**Hình 2.6. Các phương pháp của ProductImpl truy cập được từ xa**

Trong trường hợp này, phương thức getStockCode() không truy cập được từ xa. Muốn nó truy cập được từ xa thì phải tạo ra giao diện từ xa, ví dụ Stock kế thừa từ Remote và BookImpl được viết lại như sau.

```

interface Stock extends Remote{
    public String getStockCode() throws RemoteException;
}

class BookImpl extends ProductImpl{
    public BookImpl(String title, String theISBN, int sex,
                   int age1,int age2, String hobby, String img){

```

```

        super(title + " Book", sex, age1, age2, hobby, img);
        ISBN = theISBN;
    }

    public String getStockCode() throws RemoteException{
        return ISBN;
    }

    private String ISBN;
}

```

Khi một đối tượng của lớp BookImpl được truyền vào lời gọi phương thức từ xa, nơi nhận sẽ được quyền truy cập vào tất cả các phương thức từ xa đã được khai báo cả trong Remote và Stock. Như vậy một đối tượng từ xa có thể là một cuốn sách. Các đối tượng trung gian Stub được sinh ra bởi chương trình rmic theo cơ chế của RMI và người lập trình không cần quan tâm đến chúng. Ta có thể sử dụng toán tử instanceof để kiểm tra xem đối tượng p được truyền đi thuộc loại nào.

```

if(p instanceof Stock){
    Stock s = (Stock)p;
    String c = s.getStock();
    //...
}

```

## 2.4. SỬ DỤNG RMI VỚI APPLET

Khi viết chương trình ứng dụng nhưng applet mà sử dụng RMI thì có một số điểm đặc biệt cần lưu ý. Các chương trình applet có bộ quản lý an ninh riêng bởi vì nó phải thực hiện bên trong trình duyệt. Vì vậy, không cần sử dụng RMISecurityManager ở chương trình khách.

Ngoài ra ta cũng cần quan tâm nơi cất giữ các tệp \*\_Stub.class, \*Server.class. Khi mở một trang Web với thẻ APPLET, trình duyệt sẽ nạp những tệp trên và tham chiếu đến thẻ này. Vì thế, các đối tượng phục vụ bắt buộc phải có mặt trên cùng máy của trang Web. Nghĩa là trên máy phục vụ phải có:

- Trang Web
- Mã của chương trình applet
- Các lớp Stub
- Các đối tượng phục vụ
- Bộ đăng ký RMI registry

**Ví dụ 2.3.** Ta xây dựng một chương trình WarehouseApplet để thực hiện như ở ví dụ 2.2.

Khi thực hiện các chương trình applet, ta cần phân tán các tệp mã lệnh như sau:

- `java.rmi.registry.RegistryImpl` ở đâu đó trên máy chủ và bộ đăng ký registry phải được chạy trước khi khởi động applet.
- `WarehouseServer` ở đâu đó trên máy chủ và bộ đăng ký registry phải được chạy trước khi khởi động applet.
- `WarehouseImpl` có thể ở đâu đó trên máy chủ sao cho `WarehouseServer` có thể tìm được nó.
- `WarehouseApplet` để ở thư mục mà thẻ APPLET tham chiếu được.
- Các tệp lớp `_Stub.class` phải ở cùng thư mục của `WarehouseApplet`.
- Chương trình applet tìm các đối tượng trong danh sách đăng ký RMI registry trên máy, nơi chứa chính applet đó. Để tìm được máy của nó, ta sử dụng `getCodeBase()` và phương thức `getHost()`.

```
String url = getCodeBase().getHost();
url = "rmi://" + url;
centralWarehouse = (Warehouse) Naming.lookup(url +
                                              "/centralWarehouse");
```

Chương trình `WarehouseApplet` được viết như sau.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.*;
import java.rmi.*;
import java.rmi.server.*;
import javax.swing.*;
public class WarehouseApplet extends JApplet implements
    ActionListener{
    public void init(){
        initUI();
        String url = getCodeBase().getHost();
        url = "rmi://" + url;
        try{
            centralWarehouse = (Warehouse) Naming.lookup(url +
                                              "/central_warehouse");
        }catch(Exception e){
            showStatus("Error: Can't connect to warehouse! " + e);
        }
    }
}
```

```
    }

}

private void callWarehouse(Customer c){
    try{
        products = centralWarehouse.find(c);
        //descriptionListModel.clear();
        result.setText(c + "\n");
        for(int i = 0; i < products.size(); i++){
            Product p = (Product)products.get(i);
            String t = p.getDescription() + "\n";
            result.append(t);
        }
    }catch(Exception e){
        showStatus("Error: " + e);
    }
}

public void actionPerformed(ActionEvent evt){
    Object[] hobbyObjects = hobbies.getSelectedValues();
    String[] hobbyStrings = new String[hobbyObjects.length];
    System.arraycopy(hobbyObjects, 0, hobbyStrings, 0,
                      hobbyObjects.length );
    Customer c = new Customer(Integer.parseInt(age.getText()),
                               (male.isSelected() ? Product.MALE : 0)
                               + (female.isSelected() ? Product.FEMALE : 0) ,
                               hobbyStrings);
    callWarehouse(c);
}

private void initUI(){
    getContentPane().setLayout(new GridBagLayout());

    GridBagConstraints gbc = new GridBagConstraints();
    gbc.fill = GridBagConstraints.HORIZONTAL;
    gbc.weightx = 100;
```

```
gbc.weighty = 0;

add(new JLabel("Age:"), gbc, 0, 0, 1, 1);
age = new JTextField(4);
age.setText("20");
add(age, gbc, 1, 0, 1, 1);

male = new JCheckBox("Male", true);
add(male, gbc, 0, 1, 1, 1);
female = new JCheckBox("Female", true);
add(female, gbc, 1, 1, 1, 1);
gbc.weighty = 100;
add(new JLabel("Hobbies"), gbc, 0, 2, 1, 1);
String[] choices = {"Gardening", "Beauty", "Computers", "Household"};
gbc.fill = GridBagConstraints.BOTH;
hobbies = new JList(choices);
add(new JScrollPane(hobbies), gbc, 1, 2, 1, 1);

gbc.weighty = 0;
gbc.fill = GridBagConstraints.NONE;
JButton submitButton = new JButton("Submit");
add(submitButton, gbc, 0, 3, 2, 1);
submitButton.addActionListener(this);
gbc.weighty = 100;
gbc.fill = GridBagConstraints.BOTH;
result = new JTextArea(4, 40);
result.setEditable(false);
add(result, gbc, 0, 4, 2, 1);
}

private void add(Component c, GridBagConstraints gbc,
                 int x, int y, int w, int h){
    gbc.gridx = x;
    gbc.gridy = y;
    gbc.gridwidth = w;
```

```

        gbc.gridheight = h;
        getContentPane().add(c, gbc);
    }
}

private Warehouse centralWarehouse;
private JTextField age;
private JCheckBox male, female;
private JList hobbies; //, descriptions;
private JTextArea result;
private JButton submitButton;
private Vector products;
// private DefaultListModel descriptionListModel;
}

```

Như đã nói ở trên, đối với các chương trình applet không cần sử dụng bộ RMISecurityManager nên cũng không cần soạn thảo tệp client.policy. Như chúng ta đã biết để chạy được chương trình applet thì phải nhúng chúng vào thẻ APPLET của HTML, ví dụ tệp WarehouseApplet.html đơn giản nhất có dạng:

```

<applet code = "WarehouseApplet.class" height="300" width="300">
</applet>

```

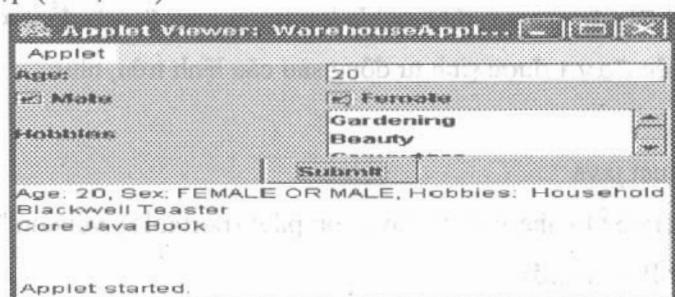
Quá trình dịch, thực hiện chương trình WarehouseApplet triệu gọi từ xa cũng gồm năm bước, trong đó bốn bước đầu giống như ở ví dụ 2.1. Bước thứ năm có thể sử dụng trình duyệt IE để mở tệp WarehouseApplet.html hoặc sử dụng appletviewer của JDK.

```

D:\users\Server\warehouse>C:\j2sdk1.4.0\bin\
                           appletviewer warehouseApplet.html

```

Kết quả chương trình WarehouseApplet thực hiện triệu gọi từ xa tương tự như ở chương trình độc lập (ví dụ 2.1).



Hình 2.7. Chương trình applet triệu gọi từ xa

## 2.5. NGÔN NGỮ ĐỊNH NGHĨA GIAO DIỆN JAVA (IDL) VÀ CORBA

Khác với RMI, CORBA cho phép ta thực hiện các lời gọi giữa các đối tượng Java với các đối tượng viết bằng các ngôn ngữ khác, ví dụ đối tượng C++. CORE sẽ phụ thuộc vào bộ môi giới yêu cầu ORB có sẵn sàng trên cả Server lẫn Client hay không.

### 2.5.1. Ngôn ngữ định nghĩa giao diện IDL

Sử dụng Java IDL như là ngôn ngữ định nghĩa giao diện CORBA trong Java 2 SDK. Bản thân IDL là ngôn ngữ định nghĩa giao diện ở dạng ngôn ngữ tự nhiên [2].

Với CORBA, một giao diện trong IDL có thể được viết đơn giản như sau:

```
interface Product{
    string getDescription();
}
```

Ở đây có sự khác biệt giữa IDL và Java. Trong IDL, kết thúc việc định nghĩa giao diện là ';' và string không bắt đầu bằng chữ in hoa 'S' như trong Java. Trong Java, các xâu (thuộc lớp String) bao gồm các ký tự Unicode 16-bit, còn trong CORBA, các xâu chỉ chứa các ký tự 8-bit. Do vậy, nếu ta gửi đi những xâu 16-bit qua ORB và những ký tự trong xâu có các bit cao khác 0 thì sẽ phát sinh ngoại lệ. Đây chính là một vấn đề không tương thích giữa các ngôn ngữ lập trình.

Chương trình dịch Java IDL Compiler (trong J2sdk.1.4.0 là idlj.exe) làm nhiệm vụ dịch các định nghĩa trong IDL sang Java. Ví dụ

```
idlj Product.idl
```

Chương trình sẽ tạo ra các tệp sau:

- Product.java, định nghĩa giao diện
- ProductHolder.java, định nghĩa lớp cất giữ cho các tham số out
- ProductHelper.java, định nghĩa lớp trợ giúp
- \_ProductImplBase.java, định nghĩa lớp cha của lớp cài đặt
- \_ProductStub.java, định nghĩa lớp trung gian Stub để trao đổi với ORB.

Một số giao diện Java được sinh tự động sau câu lệnh trên, như:

```
/*
 * Product.java .
 * Generated by the IDL-to-Java compiler (portable), version "3.1"
 * from Product.idl
 * Wednesday, July 20, 2005 6:28:29 PM ICT
 */
```

```
public interface Product extends ProductOperations,
org.omg.CORBA.Object, org.omg.CORBA.portable.IDLEntity
{
} // interface Product

/***
* ProductOperations.java .
* Generated by the IDL-to-Java compiler (portable), version "3.1"
* from Product.idl
* Wednesday, July 20, 2005 6:28:29 PM ICT
*/
public interface ProductOperations
{
    String getDescription ();
} // interface ProductOperations

/***
* ProductHelper.java .
* Generated by the IDL-to-Java compiler (portable), version "3.1"
* from Product.idl
* Wednesday, July 20, 2005 6:28:29 PM ICT
*/
abstract public class ProductHelper
{
    private static String _id = "IDL:Product:1.0";

    public static void insert (org.omg.CORBA.Any a, Product that)
    {
        org.omg.CORBA.portable.OutputStream out =
            a.create_output_stream ();
        a.type (type ());
        write (out, that);
        a.read_value (out.create_input_stream (), type ());
    }
}
```

```
public static Product extract (org.omg.CORBA.Any a)
{
    return read (a.create_input_stream ());
}

private static org.omg.CORBA.TypeCode __typeCode = null;
synchronized public static org.omg.CORBA.TypeCode type ()
{
    if (_typeCode == null)
    {
        _typeCode = org.omg.CORBA.ORB.init ().create_interface_
                    tc (ProductHelper.id (), "Product");
    }
    return _typeCode;
}

public static String id ()
{
    return _id;
}

public static Product read (org.omg.CORBA.portable.
                           InputStream istream)
{
    return narrow (istream.read_Object (_ProductStub.
                                         class));
}

public static void write (org.omg.CORBA.portable.
                           OutputStream ostream, Product value)
{
    ostream.write_Object ((org.omg.CORBA.Object) value);
}
```

```

public static Product narrow (org.omg.CORBA.Object obj)
{
    if (obj == null)
        return null;
    else if (obj instanceof Product)
        return (Product)obj;
    else if (!obj._is_a (id ()))
        throw new org.omg.CORBA.BAD_PARAM ();
    else
    {
        org.omg.CORBA.portable.Delegate delegate =
            ((org.omg.CORBA.portable.ObjectImpl)obj)._get_delegate ();
        _ProductStub stub = new _ProductStub ();
        stub._set_delegate(delegate);
        return stub;
    }
}
}

```

Java cũng cung cấp chương trình dịch ngược từ tệp .class (kết quả dịch của giao diện trong Java) sang IDL như sau:

```
rmic -idl YourInterface.class
```

Ta xét tiếp cách đặc tả các hàm trong IDL. Khi định nghĩa một phương thức, ta phải lựa chọn các tham số để truyền và nhận kết quả. Mỗi tham số có thể khai báo in, out hoặc inout.

- Những tham số khai báo in là những tham số được sử dụng để truyền dữ liệu.
- Những tham số khai báo out được sử dụng để lưu giữ những kết quả trước khi quay trở lại thực hiện chương trình chính.

Ví dụ, phương thức locate() có thể lưu giữ sản phẩm tìm thấy, được định nghĩa trong IDL như sau:

```

interface Warehouse{
    boolean locate(in string descr, out Product p);
    // ...
}
```

Đối với mỗi giao diện, chương trình dịch IDL sẽ sinh ra một lớp có hậu tố là Holder, được gọi là *lớp cất giữ*. Như trên ta thấy, sau khi dịch giao diện Product (viết trong IDL) sang Java, ngoài tệp Product.java, còn sinh tự động ra hai tệp ProductHolder.java và ProductOperation.java. Mọi lớp lưu giữ đều có một biến public là value.

Khi gọi một phương thức, ta phải truyền vào cho *đối tượng cất giữ*. Sau khi kết thúc lời gọi hàm, muốn tìm lại kết quả thì tìm qua value của đối tượng cất giữ đó. Ví dụ:

```
Warehouse w = ... ;
String descr = ...;
ProductHolder pHolder = new ProductHolder();
if(w.locate(descr, pHolder))
    p = pHolder.value;
```

Trong IDL, ta có thể sử dụng cấu trúc sequence để định nghĩa các mảng có kích cỡ biến đổi. Ví dụ, định nghĩa một kiểu chuỗi các sản phẩm:

```
typedef sequence<Product> ProductSeq;
```

Sau đó ta có thể sử dụng kiểu dữ liệu này để khai báo phương thức

```
interface Warehouse{
    ProductSeq find(in Customer c);
    .
    .
    .
```

Trong Java, cấu trúc sequence tương ứng với cấu trúc mảng.

Nếu một phương thức có thể phát sinh ngoại lệ, thì trước hết ta phải định nghĩa kiểu ngoại lệ và sau đó sử dụng khai báo nó với từ *raises*. Ví dụ, phương thức find() có thể gặp phải ngoại lệ BadCustomer.

```
interface Warehouse{
    exception BadCustomer{ string reason;}
    ProductSeq find(in Customer c) raises BadCustomer;
    .
    .
    .
```

Chương trình dịch IDL sẽ dịch sang lớp BadCustomer

```
final class BadCustomer extends org.omg.CORBA.UserException{
    public BadCustomer(){;}
    public BadCustomer(String _reason){reason = _reason;}
    public String reason;
}
```

Các giao diện cũng có thể chứa thuộc tính attribute. Một thuộc tính được xem như là một biến thể hiện, nhưng nó được gắn với một cặp hàm truy cập và cấm truy cập. Ví dụ: giao diện Book có thuộc tính isbn

```
interface Book{
    attribute string isbn;
    . . .
};
```

Thuộc tính này chuyển tương đương sang Java thành cặp hàm có cùng tên isbn:

```
String isbn() // Accessor
void isbn(String _isbn) // Mutator
```

Nếu thuộc tính khai báo là readonly thì các hàm này không cần phải sinh ra.

CORBA hỗ trợ kế thừa giữa các giao diện và sử dụng ký pháp gần giống với C++.

```
interface Book : Product{ /* ... */};
```

Dấu `:' ký hiệu cho sự kế thừa.

Trong IDL, ta có thể tạo ra một nhóm các interface và tạo thành các module.

```
module corejava {
    interface Product{ /* ... */};
    interface Warehouse{ /* ... */};
    . . .
};
```

Các module được chuyển tương ứng sang Java thành các package.

## 2.5.2. Phát triển ứng dụng với IDL và CORBA

Java ra đời với mong muốn tạo ra được những chương trình ứng dụng được viết, dịch ở một nơi nhưng chạy được ở mọi nơi. Tuy nhiên, Java không thể thay thế tất cả các ngôn ngữ khác vì nhiều lập trình viên đã quen thuộc với C/C++, Visual Basic,... và nhiều phần mềm đã được phát triển rất hiệu quả bằng những ngôn ngữ đó. Cộng đồng lập trình mong muốn tìm được tiếng nói chung cho các ngôn ngữ lập trình và thế là CORBA ra đời. CORBA tạm dịch là kiến trúc môi giới đối tượng chung được hình thành từ tổ chức quản trị đối tượng quốc tế OMG với sự hợp tác của hơn 800 công ty. Mục đích của OMG là đưa ra cách để các đối tượng viết bằng những ngôn ngữ lập trình hướng đối tượng khác nhau có thể triệu gọi lẫn nhau theo mô hình đối tượng phân tán.

CORBA không phải là ngôn ngữ lập trình, nó là ngôn ngữ đặc tả. CORBA quy định một tập các mô tả hàm, kiểu dữ liệu, cách khai báo để đặc tả đối tượng giống như giao diện trong Java. Chính vì thế CORBA còn được gọi là ngôn ngữ định nghĩa giao diện IDL.

Tất cả các chương trình ứng dụng độc lập và ứng dụng applet được phát triển trên Java 2 SDK đều có khả năng kết nối được với các đối tượng CORBA.

Để cài đặt các đối tượng CORBA, ta cần thực hiện:

1. Sử dụng IDL để viết giao diện để chỉ ra cách hoạt động của các đối tượng.
2. Sử dụng chương trình dịch IDL để dịch các chương trình viết trong các ngôn ngữ đích, sinh ra các lớp trung gian Stub và các lớp trợ giúp.
3. Bổ sung mã trình cài đặt cho các đối tượng Server viết bằng ngôn ngữ mà bạn lựa chọn. Dịch các mã trình cài đặt đó.
4. Viết chương trình phục vụ trong đó tạo lập và đăng ký các đối tượng dịch vụ. Phương thức phù hợp nhất để đăng ký là sử dụng các *dịch vụ đặt tên* của CORBA, một dịch vụ tương tự như rmiregistry.
5. Viết chương trình khách trong đó xác định các đối tượng phục vụ và gọi các dịch vụ của chúng.
6. Khởi động các dịch vụ đặt tên và chương trình phục vụ trên máy Server, sau đó là chương trình khách trên Client.

**Ví dụ 2.4.** Trước tiên chúng ta xét ví dụ, trong đó một chương trình khách (viết bằng Java) muốn triệu gọi các đối tượng từ xa ở máy chủ (viết bằng C++) có sử dụng sự hỗ trợ của CORBA và được phát triển trên nền Java 2 SDK.

*Ở phía máy chủ*, ta phải sử dụng omniORB, được cung cấp miễn phí ở <http://www.uk.research.att.com/omniORB/index.html>.

Đối tượng C++ trong ví dụ trên sử dụng biến môi trường ở Server, được khai báo trong giao diện:

```
interface Env{
    string getenv(in string name);
};
```

Lớp cài đặt giao diện này trong C++ có thể sử dụng hàm getenv() của thư viện chuẩn của C.

```
class EnvImpl : public virtual _Sk_Env{
public:
    virtual char* getenv(const char *name) {
        char* value = ::getenv(name);
        return CORBA::string_dup(value);
    }
};
```

Nhìn chung trên Server, ta phải viết chương trình để thực hiện:

1. Khởi động ORB.
2. Tạo ra đối tượng của lớp EnvImpl và đăng ký nó với ORB.

3. Sử dụng tên gọi nhớ ở Server để liên kết với đối tượng theo tên gọi.
4. Chờ lời triệu gọi từ xa của chương trình khách Client.

// EnvServer.cpp chương trình C++ ở trên Server

```
#include <iostream.h>
#include "omnithread.h"
#include "Env.hh"

class EnvImpl : public virtual _sk_Env{
public:
    virtual ~EnvImpl(){}
    virtual char* getenv(const char* name);
};

char* EnvImpl::getenv(const char* name){
    char* value = ::getenv(name);
    return CORBA::string_dup(value);
}

void bindObjectToName(CORBA::ORB_ptr obj){
    CosNaming::NamingContext_var rootContext;
    try{
        CORBA::Object_var initServ =
            orb->resolve_initial_references("NameService");
        rootContext = CosNaming::NamingContext::_narrow(initServ);
        if(CORBA::is_nil(rootContext)){
            cerr << "Failed to narrow naming context!" << endl;
            return;
        }
    }catch(CORBA::ORB::InvalidName& ex){
        cerr << "Service does not exist!" << endl;
    }
    try{
        CosNaming::Name contextName;
        contextName.length(1);
        contextName[0].id = "corejava";
        contextName[0].kind = "context";
```

```
CosNaming::NamingContext_var corejavaContext;
try{
    corejavaContext =
        rootContext -> bind_new_context(contextName);
} catch(CosNaming::NamingContext::AlreadyBound&){
    CORBA::Object_var contextObj =
        rootContext -> resolve(contextName);
    corejavaContext =
        CosNaming::NamingContext::_narrow(contextObj);
    if(CORBA::is_nil(corejavaContext)){
        cerr << "Service does not exist!" << endl;
        return;
    }
}
CosNaming::Name objectName;
objectName.length(1);
objectName[0].id = name;
objectName[0].kind = "Object";
corejavaContext -> rebind(objectName, obj);
} catch(CORBA::COMM_FAILURE&{
    cerr << "Comm Failure!" << endl;
}
int main(int argc, char *argv[]){
    cout << "Creating and initializing the ORN ..." << endl;
    CORBA::ORB_ptr orb = CORBA::ORB_init(argc, argv, "omniORB2");
    CORBA::BOA_ptr boa = orb -> BOA_init(argc, argv, "omniORB2_BOA");
    cout << "Registering server with the ORB ..." << endl;
    EnvImpl* impl = new EnvImpl();
    impl -> _obj_is_ready(boa);
    CORBA::String_var s = orb -> object_to_string(impl);
    cout << s << endl;
    cout << "Binding server to name service ... " << endl;
    Env_var env = impl->this();
    bindObjectToName(orb, "Env", env);
}
```

```

cout << "Waiting for invocation from client ... " << endl;
boa -> impl_is_ready();
return 0;
}

```

*Trên máy khách, ta cần thực hiện các bước sau:*

1. Khởi động ORB,
2. Xác định việc gắn tên gọi gợi nhớ thông qua tham chiếu vào “NameService” và NamingContext.
3. Xác định đối tượng có các phương thức cần triệu gọi,
4. Ép đối tượng nhận lại về cho phù hợp về kiểu và thực hiện triệu gọi từ xa.

// EnvClient.java chương trình khách

```

import org.omg.CosNaming.*;
import org.omg.CORBA.*;
public class EnvClient{
    public static void main(String args[]){
        try{
            ORB orb = ORB.init(args, null);
            org.omg.CORBA.Object namingContextObj =
                orb.resolve_initial_references("NameService");
            NamingContext namingContext =
                NamingContextHelper.narrow(namingContextObj);
            NameComponent[] path = { new NameComponent("corejava",
                "context"),new NameComponent("Env", "Object") };
            org.omg.CORBA.Object envObj =
                namingContext.resolve(path);
            Env env = EnvHelper.narrow(envObj);
            System.out.println(env.getenv("PATH"));
        }catch(Exception e){
            System.out.println("Error: " + e);
            e.printStackTrace(System.out);
        }
    }
}

```

Để kiểm tra chương trình trên, ta thực hiện như sau:

1. Dịch tệp IDL: Env.idl, sử dụng cả C++ và Java IDL Compiler
2. Dịch chương trình EnvServer.cpp
3. Dịch chương trình EnvClient.java
4. Khởi động dịch vụ đặt tên: naming service
5. Chạy chương trình phục vụ: EnvServer
6. Chạy chương trình khách: java EnvClient.

## 2.6. NHẬN XÉT VỀ PHƯƠNG THỨC LẬP TRÌNH PHÂN TÁN RMI

Xu hướng lập trình phân tán là xu hướng phát triển tự nhiên và là tất yếu của công nghệ phần mềm hiện nay. Đã có rất nhiều giải pháp cho vấn đề này: kiến trúc COM (Microsoft), CORBA (OMG), và hiện nay Web Services đang nổi lên như là một giải pháp lựa chọn tốt nhất.

Cơ chế kết nối theo giao thức TCP/IP bằng khái niệm Socket bắt buộc ta phải dùng đến khái niệm **cổng – port**. Một trình chủ phải hoạt động và trao đổi qua một cổng chỉ định để lắng nghe các yêu cầu đến từ các máy khách. Tuy vậy, việc mở cổng đối với một máy chủ là một vấn đề hết sức thận trọng, và không dễ để thực hiện vì nó liên quan đến sự an toàn của các hệ thống trên mạng. Nếu ta xây dựng các ứng dụng trên mạng nội hat (Intranet) thì điều này không thành vấn đề. Ta có thể chỉ định số hiệu của cổng bất kỳ cho một socket trên máy chủ, miễn là những cổng này không trùng với những cổng phổ dụng mặc định cho các ứng dụng như Web, E-Mail, FTP. Tuy nhiên, đối với mạng diện rộng (Internet), thường nó bị kiểm soát nghiêm ngặt bởi bức tường lửa. *Tường lửa* là phần mềm (có thể được cài đặt hoặc điều khiển bởi phần cứng tùy theo nhà cung cấp) chặn ở cửa ra / vào của đường kết nối trao đổi dữ liệu của mạng nội hat (mạng cục bộ) với bên ngoài (Internet). Mục đích là phòng tránh sự đột nhập, sự tấn công hay ăn cắp thông tin của những người không được phép truy cập, của các tay hacker. Nó cũng được sử dụng để kiểm soát sự truy xuất ra bên ngoài mạng Internet.

Vấn đề phát sinh chính là việc sử dụng các cổng kết nối. Hầu như tất cả bức tường lửa chỉ cho sử dụng một số hạn chế các cổng. Cổng kết nối giao thức http: 80 là cổng phổ dụng cho việc kết nối với Web Server. Một số cổng khác mà ta có thể được phép sử dụng như: cổng 21 cho ftp, 23 cho telnet và 110 cho POP3 Mail. Như vậy, ta sẽ gặp khó khăn khi sử dụng mô hình khách chủ vì gặp phải bức tường lửa.

Ứng dụng phân tán triệu gọi đối tượng từ xa có thể giải quyết vấn đề đối với tường lửa như sau:

1. Yêu cầu người quản trị bức tường lửa cung cấp một số cổng để kết nối.
2. Sử dụng cơ chế trung gian thông qua cổng 80, cổng phổ thông của dịch vụ Web Server. Cơ chế này còn được gọi là cơ chế “đường hầm”.

Như vậy, COM và RMI, về hiệu quả, cách thức hoạt động của chúng có nhiều nét tương đồng. Nhưng so với COM, RMI linh động hơn do sử dụng công nghệ Java, và do đó có thể được áp dụng cho nhiều hệ nền khác nhau (COM chỉ được áp dụng trên các hệ nền Windows). Cả hai đều có chung nhược điểm là đều phải thực hiện việc kết nối giữa các đối tượng qua các cổng đã định rõ trước. Web Services là công nghệ mới được phát triển gần đây, sử dụng SOAP cho phép trình chủ và trình khách giao tiếp với nhau qua giao thức HTTP, hay là qua cổng 80 – đây là cổng luôn được mở để phục vụ trên một HTTP Webserver, nên nhà phát triển dịch vụ này sẽ không lo lắng về vấn đề cổng giao tiếp nữa. Hơn thế nữa, Web Services được xây dựng dựa trên nền tảng là XML – độc lập với các hệ nền, nên nó có hầu như tất cả các ưu điểm của các phương thức lập trình phân tán trước đó. SOAP cho phép dữ liệu chuyển đi bằng HTTP và định dạng theo chuẩn XML. Các công nghệ mới hiện nay, .NET Framework, Java đều đưa khả năng làm việc với Web Services vào như là một thành phần quan trọng. Nó cho phép triệu gọi lẫn nhau, bắt chấp các đối tượng đó được viết bằng Java của Sun hay .NET của Microsoft.

Do vậy, việc nghiên cứu về RMI là một phần không thể thiếu đối với các nhà phát triển Java trước khi chuyển hướng tiếp cận một phương thức lập trình phân tán mới như CORBA và Web Services cũng như tiếp cận công nghệ EJB của Java.

## BÀI TẬP

- 2.1. Xây dựng một chương trình Server, trong đó có các đối tượng ở xa có chức năng xử lý cơ sở dữ liệu. Máy khách là một chương trình applet thể hiện các giá trị của cơ sở dữ liệu chạy trên trình duyệt Browser. Chương trình applet gửi yêu cầu đến đối tượng chủ yêu cầu bổ sung, sửa, xóa... một record bất kỳ trong cơ sở dữ liệu, sau khi thực hiện xong, các công việc theo yêu cầu của máy khách, đối tượng ở xa sẽ hiển thị thông tin từ cơ sở dữ liệu hiện tại lên cửa sổ máy khách.**

Như vậy, ứng dụng của chúng ta sẽ bao gồm một chương trình Java Applet thể hiện các giá trị của cơ sở dữ liệu, và chương trình ở phía máy chủ có thể thao tác, xử lý thông tin trong cơ sở dữ liệu (thêm, sửa, xóa và cập nhật dữ liệu) theo yêu cầu của applet.

- 2.2. Xây dựng chương trình Chatting (tán gẫu) trên mạng.**

Trên máy Client có các chức năng:

- Gửi tin nhắn đến một hay nhiều người khác,
- Xem có bao nhiêu thành viên tham gia.
- Kết thúc một phiên tán gẫu.

Trên máy chủ thực hiện:

- Khi có một Client kết nối đến thì phải ghi lại thông tin về người đó.
- Mỗi Client có thể gửi tin cho nhiều người khác đang kết nối.

Khi có một Client ra khỏi hệ thống thì Server phải thông báo cho tất cả các thành viên khác.

# Chương 3

## LẬP TRÌNH MẠNG

Chương ba giới thiệu:

- ✓ Kết nối với Server
- ✓ Mô hình tính toán Client/Server
- ✓ Kết nối mạng Internet tới một dịch vụ
- ✓ Trao đổi thông tin với Web Server
- ✓ Gửi, nhận E-mail

### 3.1. KẾT NỐI VỚI SERVER

Trước khi viết chương trình chạy trên mạng, ta phải tìm hiểu về chương trình đăng nhập từ xa telnet.exe. Tìm chương trình này ở thư mục Windows..., nếu không thấy, ta có thể chạy lại Setup.

Để thực hiện một kết nối từ một telnet Client, ta phải chọn cổng để kết nối mạng. Một hộp thoại nhắc ta cho một “Host Name”, địa chỉ IP của máy ở xa mà ta muốn kết nối và “Terminal Type” mô tả loại mô phỏng đầu cuối mà ta muốn thực hiện. Tất cả các quá trình xử lý và lưu trữ kết quả diễn ra trên máy tính từ xa.

Ta có thể sử dụng telnet để kết nối với máy từ xa và để kiểm tra hộp thư của mình, nhưng không thể dùng nó để trao đổi với các dịch vụ khác trên mạng. Ví dụ, ở dòng lệnh DOS-Prompt, ta có thể dùng telnet để xem “time of day” (thời gian trong ngày)

```
telnet time-A.timefreq.blrdoc.gov 13
```

Theo qui ước, dịch vụ “time of day” (xem giờ trong ngày) được gắn với cổng 13.

Một khi kết nối được thiết lập, chương trình từ xa sẽ gửi lại thông tin, ví dụ

```
50692 98-09-02 22:45:15 50 0 0 50.0 UTC(NIST) *
```

Trong Java ta có thể viết một chương trình khá đơn giản để thực hiện công việc giống như telnet.

**Ví dụ 3.1.** Chương trình Java thực hiện chức năng tương tự telnet.

```
// SocketTest.java chương trình thực hiện đăng nhập từ xa giống như telnet.
```

```

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

public class SocketTest{
    public static void main(String[] args){
        try{
            Socket s = new Socket("time-A.timefreq.bldrdoc.gov", 13);
            BufferedReader in = new BufferedReader
                (new InputStreamReader(s.getInputStream()));
            boolean more = true;
            while(more){
                String line = in.readLine();
                if(line == null) more = false;
                else System.out.println(line);
            }
        }catch(IOException e){
            System.out.println("Error: " + e);
        }
    }
}

```

Đoan mã lệnh đầu tiên ta cần tìm hiểu là

```

Socket s = new Socket("time-A.timefreq.bldrdoc.gov", 13);
BufferedReader in = new BufferedReader
    (new InputStreamReader(s.getInputStream()));

```

Trong đó, *Socket* tạm dịch là cơ chế “ô cắm”, được hiểu như là một bảng các chốt được thiết lập để cắm các thành phần tương ứng vào đó, giống như bảng các ô phích cắm thiết bị điện dân dụng. *Socket chính là lớp (trình tượng) thực hiện sự trao đổi giữa* các máy với nhau. Khi một *Socket* mở thành công (thiết lập được sự kết nối giữa hai máy), ta có thể sử dụng phương thức *getInputStream()* để đọc tin từ xa giống như đọc dữ liệu từ tệp đã được mở để đọc.

Sau đó, chương trình trên đọc tất cả các ký tự được gửi đi từ Server bằng lệnh *readLine()* và in ra màn hình từng dòng một.

Tiếp theo ta muốn tạo ra sự kết nối với Server để nhận tin về Client. Khi chương trình Server chạy, nó phải lắng nghe xem có chương trình Client nào kết nối vào cổng đang chờ, ví dụ cổng 8189 hay không.

```
ServerSocket s = new ServerSocket(8189);
```

Thiết lập một dịch vụ và lắng nghe sự kết nối ở cổng 8189.

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

Chương trình chờ cho đến khi có một Client kết nối qua cổng 8189. Một khi sự kết nối qua cổng này được thiết lập, các thông tin trao đổi giữa hai máy sẽ được thực hiện.

**Ví dụ 3.2.** Chương trình thể hiện tiếng vọng (echo) của một Client.

// EchoServer.java: Chương trình nhảc lại bản tin (tiếng vọng) nhận được.

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

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

```
public class EchoServer{
```

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

```
        try{
```

```
            ServerSocket s = new ServerSocket(8189);
```

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

```
            BufferedReader in = new BufferedReader
```

```
(new InputStreamReader(incoming.getInputStream()));
```

```
            PrintWriter out = new PrintWriter
```

```
(incoming.getOutputStream(), true);
```

```
            out.println("Hello! Enter Bye to exit.");
```

```
            boolean done = false;
```

```
            while(!done){
```

```
                String line = in.readLine();
```

```
                if(line == null) done = true;
```

```
                else {
```

```
                    out.println("Echo: " + line);
```

```
                    if(line.trim().equals("Bye"))
```

```
                        done = true;
```

```
}
```

```
            incoming.close();
```

```
}catch(IOException e){
```

```
    System.out.println("Error: " + e);
```

```
}
```

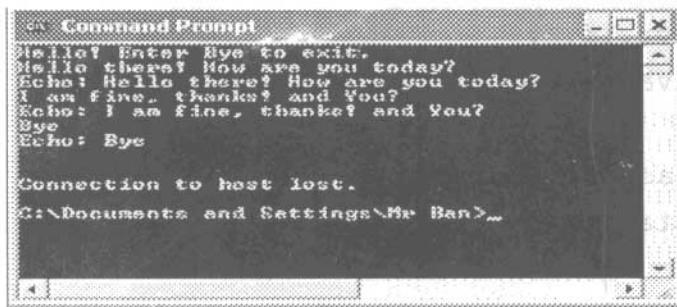
```
}
```

```
}
```

Dịch xong và để chạy chương trình tiếng vọng, ta cần sử dụng telnet để kết nối với Server: 127.0.0.1 và cổng 8189.

```
telnet 127.0.0.1 8189
```

Địa chỉ IP: 127.0.0.1 là địa chỉ máy cục bộ (local host) mà ta muốn chạy chương trình trên. Khi kết nối được với cổng 8189 thì ta có được màn hình trao đổi tin như hình 3.1. Gõ một mẫu tin bất kỳ, mẫu tin này sẽ được nhắc lại trên màn hình. Gõ “Bye” để kết thúc sự kết nối qua cổng đã được thiết lập đồng thời kết thúc chương trình.



Hình 3.1. Chương trình tiếng vọng

### 3.2. MÔ HÌNH TÍNH TOÁN CLIENT/SERVER

Khi nói tới *lập trình mạng* ta thường nghĩ đến cách trao đổi giữa một chương trình phục vụ (Server) với một hay nhiều chương trình khách (Client) [2]. Chương trình khách gửi một yêu cầu tới cho chương trình phục vụ, và chương trình này xử lý dữ liệu để trả lời cho chương trình khách. Như vậy, chương trình khách muốn gửi được yêu cầu thì trước hết phải tìm cách kết nối với Server. Server có thể chấp nhận hay từ chối sự kết nối này. Một khi sự kết nối đã được thiết lập thì Client và Server trao đổi với nhau thông qua Sockets. Các lớp trong gói `java.net` cung cấp các phương thức để kết nối mạng và trao đổi tin giữa các máy với nhau theo mô hình Client/Server. Mặt khác, trên Internet nhiều máy tính sử dụng các giao thức chuẩn để trao đổi với nhau.

*Socket chính là lớp (tríu tượng) thực hiện sự trao đổi giữa Server và Client.* Java xem sự trao đổi giữa Server và Client dựa trên Socket gần giống như các thao tác vào/ra (I/O), nghĩa là các chương trình có thể đọc, ghi vào Socket dễ dàng như chúng đọc, ghi lên tệp dữ liệu. Lớp Server tạo ra sự kết nối từ máy Client tới Server thông qua các phương thức tạo lập đối tượng.

Lớp `ServerSocket` dùng để tạo kết nối từ Server tới các máy Client. Đối tượng của lớp này được tạo ra trên Server và lắng nghe những kết nối từ máy Client theo cổng xác định. Server phải chạy thường trực trước khi Client bắt đầu thực hiện. Server chờ sự yêu cầu kết nối của Client. Các Socket được xác định tương ứng với địa chỉ của máy và cổng. Để thiết lập được một Server, ta phải tạo ra một đối tượng, ví dụ

serverSocket của lớp ServerSocket, gắn nó với một cổng nào đó và lắng nghe xem có Client nào cần kết nối hay không.

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

Trong đó, port là số hiệu của cổng nằm giữa 0 và 1023. Cổng cho phép trao đổi tin giữa hai chương trình. Địa chỉ của cổng là số 16 bit và thường phụ thuộc vào các giao thức ứng dụng. Thông thường, giao thức FTP sử dụng cổng 21, telnet dùng cổng 23, các dịch vụ E-mail (SMTP) sử dụng cổng 25, Web Server chạy trên cổng 80.

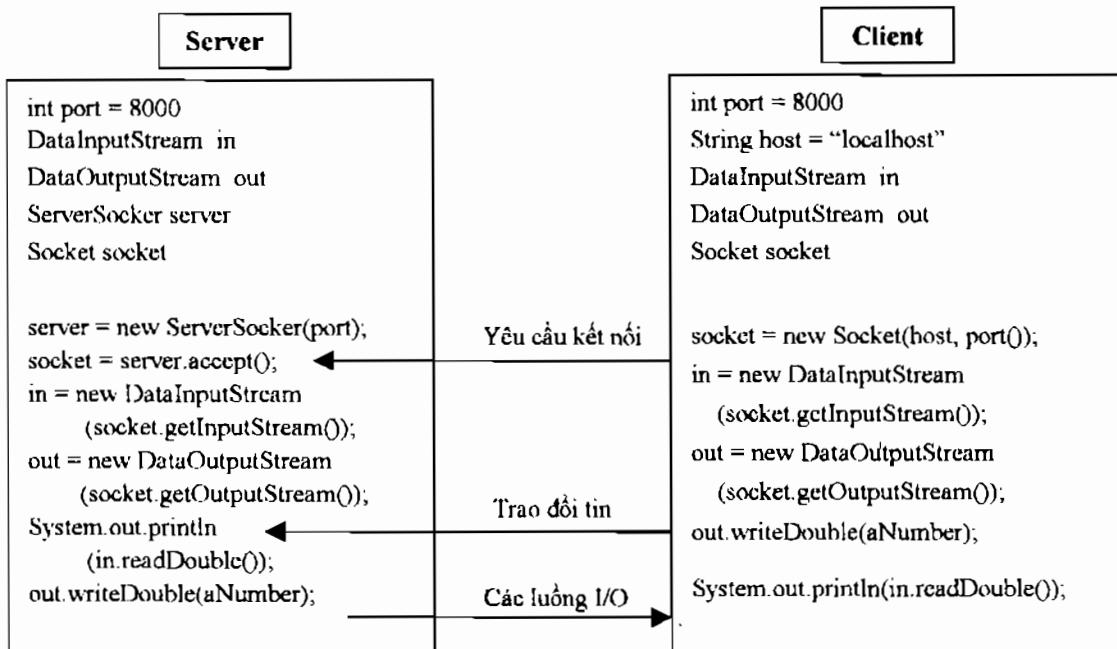
Sau khi đối tượng serverSocket được tạo ra, Server có thể sử dụng câu lệnh sau để lắng nghe yêu cầu kết nối của Client.

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

serverName là tên của máy chủ trên Internet hoặc địa chỉ IP, xác định duy nhất một máy tính trên toàn mạng Internet (cả thế giới). IP là giao thức được sử dụng để gửi tin từ một máy tới máy khác trên mạng Internet. IP phải xác định được máy gửi và máy nhận thông qua địa chỉ IP. Một địa chỉ IP gồm bốn số nguyên nằm giữa 0 và 255, phân cách với nhau bằng dấu ‘.’. Ví dụ

```
Socket connectToServer = new Socket("129.74.250.103", 8000);
```

Vì địa chỉ IP thường rất khó nhớ nên thay vì địa chỉ IP người ta thường sử dụng tên miền dịch vụ, ví dụ Gophers.nd.edu tương ứng với địa chỉ IP nêu trên.



Hình 3.2. Sự trao đổi giữa Server và Client

Ngày nay, trên Internet hai máy có thể trao đổi với nhau phần lớn đều dựa trên giao thức chuẩn TCP/IP.

Sau khi sự kết nối đã được Server chấp nhận, việc trao đổi giữa Client và Server giống như trên các luồng vào/ra I/O. Ví dụ

```
InputStream isFromServer = connectToServer.getInputStream();
OutputStream osToServer = connectToServer.getOutputStream();
```

Tạo ra luồng vào/ra: isFromServer để đọc từ Server và osToServer để ghi vào (gửi cho) Server. Hai lớp InputStream, OutputStream được sử dụng để đọc và ghi từng byte. Ta cũng có thể sử dụng DataOutputStream, DataInputStream, BufferedReader, PrintWriter để đọc, ghi dữ liệu kiểu int, double, String, v.v. Ví dụ, đoạn chương trình sau tạo ra isFromClient, osToClient để đọc, ghi các dữ liệu kiểu nguyên thủy ở Client.

```
DataInputStream isFromClient = new DataInputStream
(connectToClient.getInputStream());
DataOutputStream osFromClient = new DataOutputStream
(connectToClient.getOutputStream());
```

Server có thể sử dụng isFromClient.read() để nhận dữ liệu từ Client và isFromClient.write() để gửi dữ liệu cho Client.

**Ví dụ 3.3.** Chúng ta hãy xét một chương trình Client gửi dữ liệu là bán kính của đường tròn cho Server. Server nhận dữ liệu này, tính diện tích của hình tròn và sau đó gửi kết quả lại cho chương trình khách.

```
// Server.java nhận dữ liệu từ Client, tính diện tích hình tròn và gửi lại kết quả cho Client
import java.io.*;
import java.net.*;
public class Server{
    public static void main(String arg[]){
        try{
            // Tạo ra socket server
            ServerSocket serverSocket = new ServerSocket(8000);
            // Lắng nghe yêu cầu kết nối trên socket server
            Socket connectToClient = serverSocket.accept();
            // Tạo ra một luồng vào để nhận dữ liệu từ Client
            DataInputStream isFromClient = new DataInputStream(
                connectToClient.getInputStream());
            // Tạo ra một luồng ra để gửi dữ liệu cho Client
            DataOutputStream osToClient = new DataOutputStream(
                connectToClient.getOutputStream());
        }
    }
}
```

```

        // Liên tục nhận, xử lý và gửi kết quả lại cho Client
        while(true) {
            // Đọc một số double từ Client
            double radius = isFromClient.readDouble();
            System.out.println("Ban kinh nhan tu Client: " + radius);
            double area = radius * radius * Math.PI;
            // Gửi kết quả: diện tích kiểu double cho Client
            osToClient.writeDouble(area);
            osToClient.flush();
            System.out.println("Hình tròn có bán kính: " + radius + " & diện tích: " + area);
        }
    } catch(IOException ex) {
        System.err.println(ex);
    }
}

// Client.java nhập dữ liệu (bán kính) từ bàn phím, gửi cho Server, nhận và hiển thị
// diện tích hình tròn nhận được .
import java.io.*;
import java.net.*;
public class Client{
    public static void main(String arg[]){
        try{
            // Tạo ra một socket để kết nối với socket server
            Socket connectToServer = new Socket("localhost", 8000);
            // Tạo ra một luồng vào để nhận dữ liệu từ server
            DataInputStream isFromServer = new DataInputStream(
                connectToServer.getInputStream());
            // Tạo ra một luồng ra để gửi dữ liệu đến server
            DataOutputStream osToServer = new DataOutputStream(
                connectToServer.getOutputStream());
            // Tạo ra một luồng vào để nhận dữ liệu từ bàn phím
            DataInputStream str = new DataInputStream(System.in);

```

```

    // Liên tục gửi dữ liệu nhập vào từ bàn phím cho server và nhận lại diện tích
    while(true) {
        // Nhập bán kính từ bàn phím
        System.out.print("Hay cho biet ban kinh: ");
        double radius = Double.parseDouble(str.readLine());
        // Gửi dữ liệu cho server
        osToServer.writeDouble(radius);
        osToServer.flush();
        // Nhận lại kết quả từ server
        double area = isFromServer.readDouble();
        System.out.println(" Dien tich hinh tron nhan duoc tu Server: "+ area);
    }
} catch(IOException ex) {
    System.err.println(ex);
}
}
}

```

Hai chương trình trên thực hiện liên tục, chương trình Client chờ nhập bán kính của hình tròn và trao đổi với Server cho đến khi một trong hai chương trình kết thúc (nhấn CTRL+C) hay nhận biểu tượng kết thúc chương trình.

### 3.3. PHỤC VỤ NHIỀU CHƯƠNG TRÌNH CLIENT

Cùng một lúc, một chương trình Server có thể trao đổi với nhiều chương trình Client. Server chạy thường trực trên máy chủ và các chương trình Client trên mạng có thể kết nối với Server bất kỳ lúc nào [2]. Để Server có thể kết nối đồng thời với nhiều Client thì ta có thể sử dụng các luồng thực hiện như đã trình bày ở chương I. Server thiết lập việc kết nối với nhiều Client như sau.

```

while(true) {
    Socket connectToClient = serverSocket.accept();
    Thread thread = new ThreadClass(connectToClient);
    thread.start();
}

```

Socket server có thể có nhiều kết nối. Mỗi vòng lặp của chu trình while tạo ra một kết nối mới. Khi đã thiết lập được một kết nối, một luồng thực hiện (thread) mới được

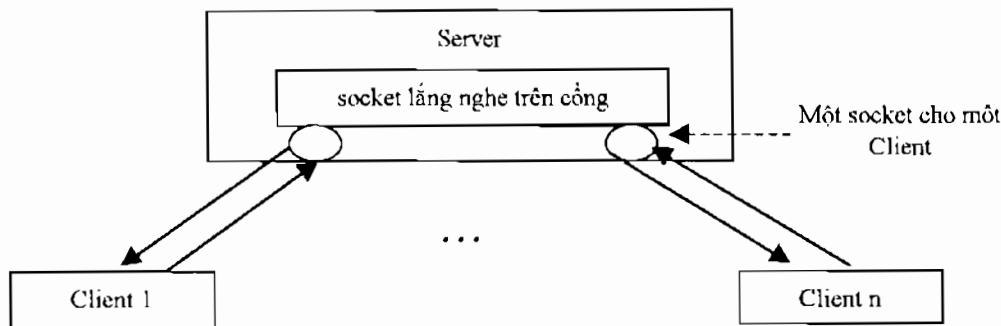
tạo ra để đảm nhận việc trao đổi giữa Server và Client. Sau đó Server chờ những yêu cầu kết nối khác. Nghĩa là có nhiều kết nối thực hiện theo cơ chế đa luồng thực hiện đồng thời.

Trong khi trao đổi với nhau, nhiều khi ta muốn biết những ai đang kết nối với Server. Để biết về chương trình khách đang kết nối với Server, ta sử dụng lớp InetAddress để xác định tên miền (host name) và địa chỉ IP.

```
InetAddress clientInetAddress = connectToClient.getInetAddress();
System.out.println("Host name của Client: " + clientNo + " là "
+ clientInetAddress.getHostName());
System.out.println("Địa chỉ IP của Client: " + clientNo + " là "
+ clientInetAddress.getHostAddress());
```

### 3.3.1. Xây dựng chương trình ứng dụng độc lập

**Ví dụ 3.2.** Xây dựng chương trình MultiThreadServer cho phép nhận được dữ liệu đồng thời từ nhiều Client. Mỗi kết nối thực hiện trên một luồng. Server liên tục nhận các số đo bán kính của hình tròn, tính diện tích và gửi kết quả tương ứng cho từng Client.



Hình 3.3. Cơ chế đa luồng cho phép Server xử lý độc lập các yêu cầu của Client

// MultiThreadServer.java: Server tạo ra n luồng để xử lý đồng thời yêu cầu của // n Client.

```
import java.io.*;
import java.net.*;
public class MultiThreadServer{
    public static void main(String arg[]){
        try{
            ServerSocket serverSocket = new ServerSocket(8000);
            int clientNo = 1;
            while(true){

```

```
        Socket connectToClient = serverSocket.accept();
        System.out.println("Khởi động luồng cho Client: " + clientNo);
        // Xác định tên miền và địa chỉ IP của Client có số hiệu clientNo
        InetAddress clientInetAddress = connectToClient.getInetAddress();
        System.out.println("Host name của Client: " + clientNo + " là " +
                           clientInetAddress.getHostName());
        System.out.println("Địa chỉ IP của Client: " + clientNo + " là " +
                           clientInetAddress.getHostAddress());
        HandleAClient thread = new HandleAClient(connectToClient);
        thread.start();
        clientNo++;
    }
} catch(IOException ex) {
    System.err.println(ex);
}
}

// Định nghĩa lớp kế thừa từ Thread để xử lý một kết nối mới
class HandleAClient extends Thread{
    private Socket connectToClient;
    public HandleAClient(Socket socket){
        connectToClient = socket;
    }
    // Cài đặt hàm run() để thực hiện theo luồng mới được tạo ra
    public void run(){
        try{
            DataInputStream isFromClient = new DataInputStream(
                connectToClient.getInputStream());
            DataOutputStream osToClient = new DataOutputStream(
                connectToClient.getOutputStream());
            while(true){
                double radius = isFromClient.readDouble();
                System.out.println("Bạn kinh nhận từ Client: " + radius);
                double area = radius * radius * Math.PI;
```

```

        osToClient.writeDouble(area);

        osToClient.flush();

        System.out.println("Hình tròn có bán kính: " +
                           radius + " có diện tích: " + area);

    }

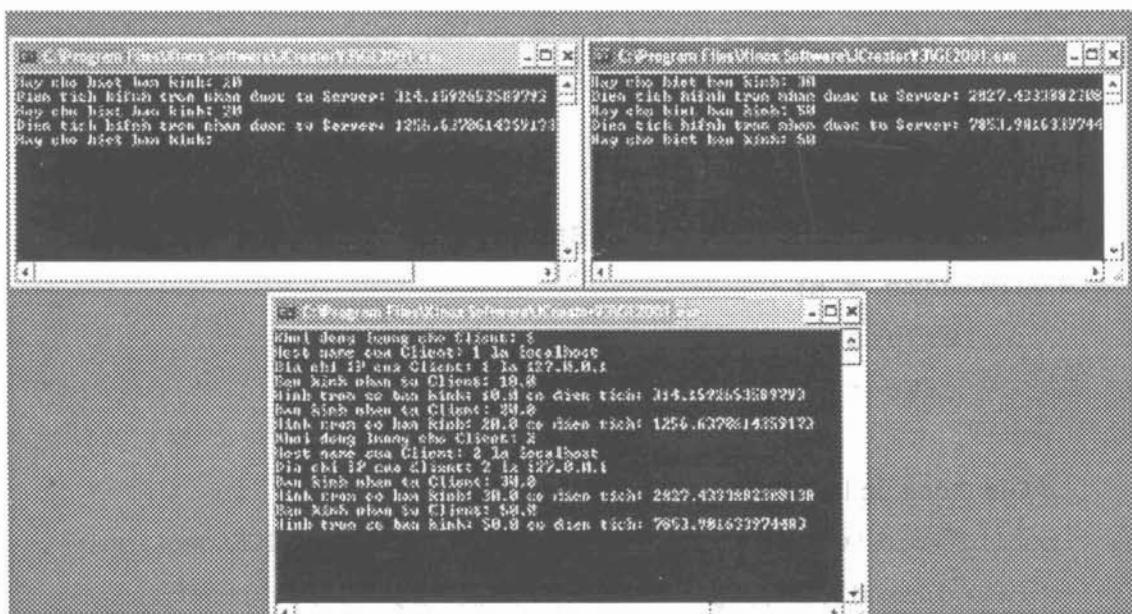
} catch(IOException ex){

    System.err.println(ex);

}

}

```



Hình 3.4. Hai Client thực hiện trên hai luồng trao đổi với Server

Trong chương trình trên, các luồng thực hiện được tạo ra để xử lý sự trao đổi giữa các Client với Server. Các luồng này thực hiện độc lập với nhau. Mỗi luồng tạo ra một vùng đệm (buffer) đọc và ghi để nhận và gửi dữ liệu cho Client tương ứng.

Chạy hai chương trình Client để thực hiện trên hai luồng trao đổi đồng thời với Server trong môi trường JCreator (<http://www.jcreator.com>) ta có kết quả như hình 3.4.

### 3.3.2. Xây dựng ứng dụng nhúng applet

Sau đây chúng ta hãy xét cách kết nối một applet với một Server.

**Ví dụ 3.5.** Chương trình cho phép sinh viên đăng ký (ghi danh) theo học một khóa đào tạo trên mạng. Sinh viên gửi những thông tin của mình về cho Server. Server nhận và ghi vào tệp truy cập ngẫu nhiên.

Trước tiên ta xây dựng RegistrationServer để nhận các thông tin mà sinh viên đăng ký từ chương trình applet và ghi lên tệp ngẫu nhiên.

```
// Student.java: Định nghĩa lớp Student bao gồm những thông tin cần đăng ký: họ tên,
// địa chỉ (phố, huyện, xã) và thành phố.

import java.io.*;
public class Student{
    private String name, street, city;
    final static int NAME_SIZE = 32;
    final static int STREET_SIZE = 32;
    final static int CITY_SIZE = 20;
    public Student(){
    }
    public Student(String n, String s, String c){
        name = n;
        street = s;
        city = c;
    }
    // Ghi thông tin lên tệp theo kích thước cố định
    public void writeStudent(DataOutput out) throws IOException{
        FixedLengthStringIO.writeFixedLengthString(name,
            NAME_SIZE, out);
        FixedLengthStringIO.writeFixedLengthString(street,
            STREET_SIZE, out);
        FixedLengthStringIO.writeFixedLengthString(city, CITY_SIZE, out);
    }
    // Đọc thông tin từ tệp theo kích thước cố định
    public void readStudent(DataInput in) throws IOException{
        name = FixedLengthStringIO.readFixedLengthString
            (NAME_SIZE, in);
        street = FixedLengthStringIO.readFixedLengthString
            (STREET_SIZE, in);
    }
}
```

```
    city = FixedLengthStringIO.readFixedLengthString
        CITY_SIZE, in);
}

public String getName(){return name;}
public String getStreet(){return street;}
public String getCity(){return city;}
public void setName(String s){name = s; }
public void setStreet(String s){street = s; }
public void setCity(String s){city = s; }

}

// FixedLengthStringIO.java: Định nghĩa lớp đọc/ghi các xâu theo kích cỡ cố định
class FixedLengthStringIO{
    // Đọc một xâu có độ dài cố định
    public static String readFixedLengthString(int size, DataInput in)
        throws IOException{
        char[] c = new char[size];
        for(int i = 0; i < size; i++)
            c[i] = in.readChar();
        return new String(c);
    }

    // Ghi một xâu có độ dài cố định
    public static void writeFixedLengthString(String s, int
        size, DataOutput out) throws IOException{
        char[] cBuffer = new char[size];
        s.getChars(0, s.length(), cBuffer, 0);
        for(int i = s.length(); i < cBuffer.length; i++)
            cBuffer[i] = ' ';
        String newS = new String(cBuffer);
        out.writeChars(newS);
    }

}
```

```
// RegistrationServer.java: chương trình nhận các thông tin đăng ký của sinh viên và
// ghi lên tệp ngẫu nhiên.

import java.io.*;
import java.net.*;
import java.awt.*;
import java.util.Date;
import javax.swing.*;

public class RegistrationServer extends JFrame{
    private static JTextArea jtaLog;
    // File dùng để ghi thông tin đăng ký của sinh viên
    private static RandomAccessFile raf = null;

    public static void main(String args[]){
        RegistrationServer server = new RegistrationServer();
    }

    public RegistrationServer(){
        // Thiết lập màn hình như hình 3.4.b
        JScrollPane scroll = new JScrollPane(jtaLog = new JTextArea());
        getContentPane().add(scroll, BorderLayout.CENTER);
        setSize(300, 300);
        setTitle("Dịch vụ đăng ký");
        setVisible(true);
        // Mở một tệp ở Server
        try{
            // Mở tệp có tên "student.dat" để đọc / ghi
            raf = new RandomAccessFile("student.dat", "rw");
        }catch(IOException ex){
            jtaLog.append(new Date() + ": Error: " + ex);
            System.exit(0);
        }
        try{

```

```

// Tạo lập một socket
ServerSocket serverSocket = new ServerSocket(8000);
jtaLog.append(new Date() + ": Khởi động Server mới\n");
// Đếm số luồng (số sinh viên đăng ký)
int count = 1;
while(true) {
    Socket socket = serverSocket.accept();
    jtaLog.append(new Date() + ": Client: " +
    socket.getInetAddress().getHostAddress() + "đã được kết nối\n");
    // Khởi động một luồng mới phục vụ cho việc đăng ký của Client
    new RegistrationThread(socket, count++).start();
}
} catch(IOException ex) {
    jtaLog.append(new Date() + ": Error: " + ex);
}
}

// Viết thông tin đã đăng ký vào tệp
private synchronized static void writeToFile(Student student) {
try{
    // Chuyển về cuối tệp
    raf.seek(raf.length());
    student.writeStudent(raf);
    // Hiển thị thông tin đã nhận được
    jtaLog.append("--- Thông tin sau đã được ghi vào tệp\n");
    String s = student.getName() + "\n" + student.getStreet() +
    "\n" + student.getCity() + "\n";
    jtaLog.append(s);
} catch(IOException ex) {
    jtaLog.append(new Date() + ": " + ex);
}
}

// Định nghĩa luồng để nhận yêu cầu đăng ký của Client
class RegistrationThread extends Thread{

```

```
private Socket socket;
private int clientNo;
// Tạo lập vùng đệm để nhận tin
private BufferedReader in;
// Tạo lập luồng để xử lý yêu cầu đăng ký
public RegistrationThread(Socket socket, int clientNo){
    this.socket = socket;
    this.clientNo = clientNo;
    jtaLog.append(new Date() + ".Thread: " + clientNo +
                  "được khởi động\n");
    // Tạo lập một dòng vào để nhận từ Client
    try{
        in = new BufferedReader(
            new InputStreamReader(socket.getInputStream()));
    }catch(IOException ex){
        jtaLog.append(new Date() + ": " + ex);
    }
}
public void run(){
    String name, street, city;
    try{
        // Nhận tin từ Client
        name = new String(in.readLine());
        street = new String(in.readLine());
        city = new String(in.readLine());
        // Tạo lập đối tượng sinh viên
        Student student = new Student(name, street, city);
        writeToFile(student);
    }catch(IOException ex){
        System.out.println(ex);
    }
}
```

```
// RegistrationClient.java: chương trình chạy cả chế độ độc lập lẫn chế độ Applet,  
// nhận các thông tin đăng ký của sinh viên và gửi về cho máy chủ.  
  
import java.io.*;  
import java.net.*;  
import java.awt.*;  
import javax.swing.border.*;  
import java.awt.event.*;  
import javax.swing.*;  
  
public class RegistrationClient extends JApplet implements  
    ActionListener{  
    private static JTextArea jtaLog;  
    // Thiết lập nút “Register” để đăng ký  
    private JButton jbRegister = new JButton("Register");  
    private boolean isStandalone = false;  
    JTextField jtfName = new JTextField(32);  
    JTextField jtfStreet = new JTextField(32);  
    JTextField jtfCity = new JTextField(20);  
    Container container;  
    JPanel p;  
    public void init(){  
        makePanel();  
        container = getContentPane();  
        container.add(p, BorderLayout.CENTER);  
        container.add(jbRegister, BorderLayout.SOUTH);  
        // Lắng nghe sự kiện khi người sử dụng nhấn nút Register  
        jtfName.addActionListener(this);  
        jbRegister.addActionListener(this);  
    }  
    // Thiết lập màn hình chức năng để nhập thông tin như hình 3.5.a  
    public void makePanel(){  
        p = new JPanel();  
        JPanel p1 = new JPanel();
```

```
    p1.setLayout(new GridLayout(3, 1));
    p1.add(new JLabel("Name"));
    p1.add(new JLabel("Street"));
    p1.add(new JLabel("City"));
    JPanel p2 = new JPanel();
    p2.setLayout(new GridLayout(3, 1));
    p2.add(jtfName);
    p2.add(jtfStreet);
    p2.add(jtfCity);
    p.setLayout(new BorderLayout());
    p.add(p1, BorderLayout.WEST);
    p.add(p2, BorderLayout.EAST);
}

public void actionPerformed(ActionEvent e){
    if(e.getSource() == jbRegister){
        try{
            Socket socket;
            if(isStandalone)
                socket = new Socket("localhost", 8000);
            else
                socket = new Socket(getCodeBase().getHost(), 8000);
            // Lập dòng output để gửi cho server
            PrintWriter toServer =
                new PrintWriter(socket.getOutputStream(), true);
            // Đọc các trường thông tin từ Client
            Student s = new Student(jtfName.getText().trim(),
                                    jtfStreet.getText().trim(),
                                    jtfCity.getText().trim());
            // Gửi các thông tin nhận được cho server
            toServer.println(s.getName());
            toServer.println(s.getStreet());
            toServer.println(s.getCity());
        }catch(IOException ex){
        }
    }
}
```

```

        System.out.println(ex);
    }
}

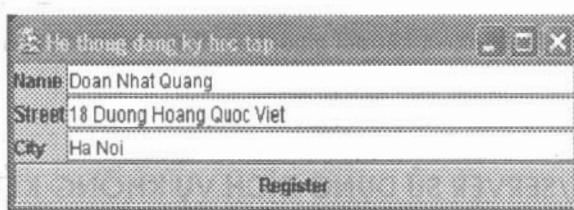
public static void main(String args[]){
    JFrame frame = new JFrame("Hệ thống đăng ký học tập");
    // Lập một frame
    RegistrationClient applet = new RegistrationClient();
    applet.isStandalone = true;
    // Đưa applet vào frame
    frame.getContentPane().add(applet, BorderLayout.CENTER);
    applet.init();
    applet.start();
    // Hiển thị frame
    frame.pack();
    frame.setVisible(true);
}
}

```

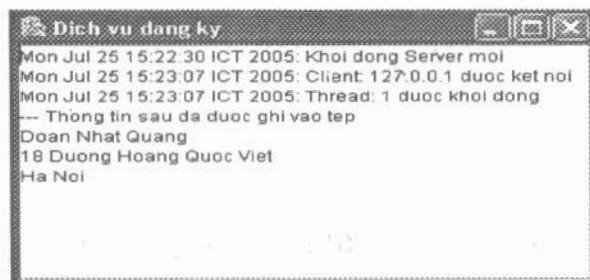
Chương trình Server nhận thông tin đăng ký từ nhiều máy Client trên mạng. Mỗi khi có một Client kết nối với Server để đăng ký thì nó tạo ra một luồng thực hiện (thread) để xử lý yêu cầu của khách.

Chương trình Client có thể chạy độc lập cũng như được nhúng vào Web browser (IE) dưới dạng applet. Muốn chương trình chạy được dưới dạng applet thì phải nhúng vào thẻ APPLET CODE như đã nêu và nó sử dụng getCodeBase().getHost() để xác định được địa chỉ IP của máy chủ.

Sau khi dịch và thực hiện, các chương trình trên cho kết quả ví dụ như hình 3.5.

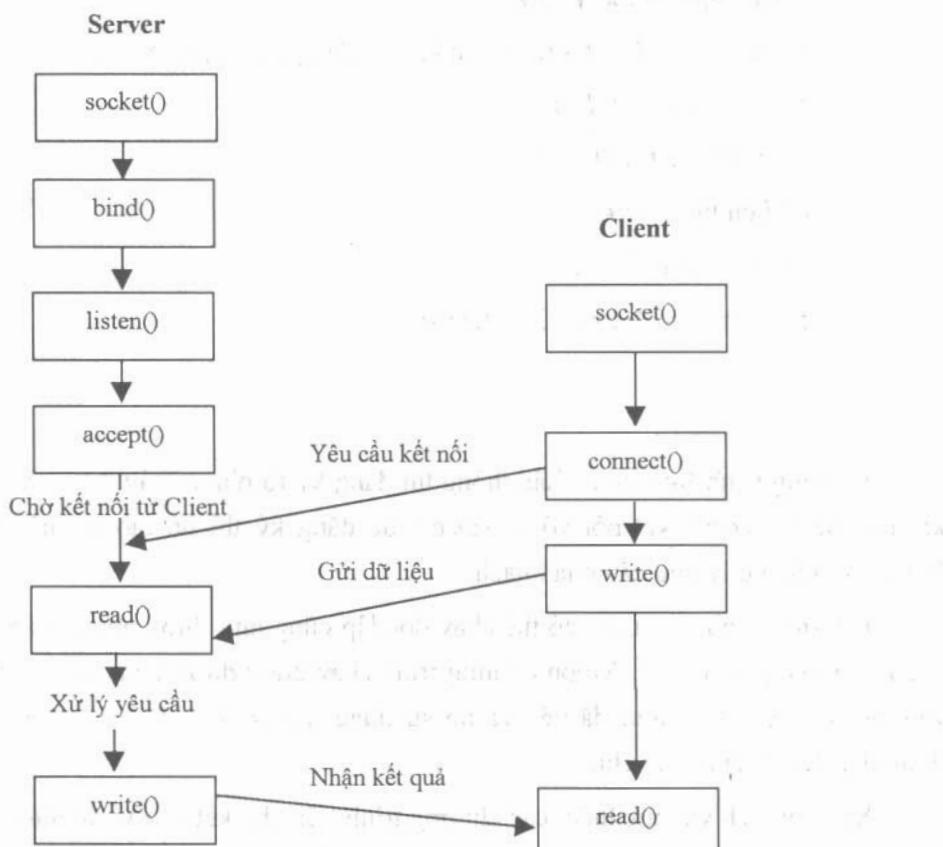


**Hình 3.5.a** Chương trình đăng ký học tập trên mạng (Client)



**Hình 3.5.b) Chương trình Server nhận các danh sách đăng ký trên mạng**

**Lưu ý:** Mô hình Client-Server nêu trên đều sử dụng hướng kết nối. Đầu tiên Server được khởi động và chờ đợi một sự kết nối từ Client. Sau khi kết nối được thiết lập, hai bên luân phiên nhau gửi, nhận dữ liệu thông qua đường truyền như hình 3.6.



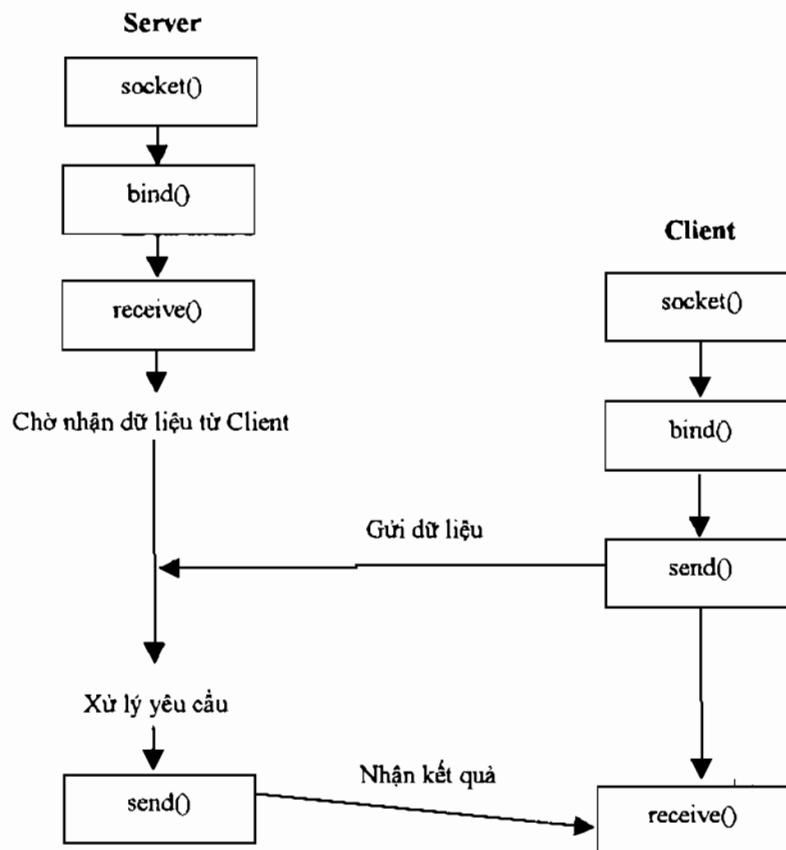
**Hình 3.6. Hệ Socket kết nối giao thức định hướng kết nối**

### 3.4. MÔ HÌNH CLIENT/SERVER SỬ DỤNG DỊCH VỤ KHÔNG KẾT NỐI

Trong mô hình Client/Server sử dụng dịch vụ kết nối, các lớp Socket, và ServerSocket sử dụng giao thức HTTP để trao đổi tin giữa hai máy tính được kết nối với

nhau. Một giao thức khác, giao thức không kết nối UDP cho phép chương trình ứng dụng truy cập trực tiếp đến gói tin của dịch vụ chuyển giao giống như giao thức IP. UDP sử dụng đồ hình dữ liệu để trao đổi giữa các chương trình Client với Server. Nó là giao thức không định hướng kết nối và nó cho phép các chương trình ứng dụng trao đổi tin với nhau qua mạng với ít thông tin điều khiển nhất nên khá nhanh. Tuy nhiên, độ tin cậy truyền tin thấp vì nó không có cơ chế kiểm tra tính chính xác của dữ liệu truyền.

Mô hình Client/Server sử dụng lớp DatagramSocket để cài đặt các socket ở máy Server và Client. Thay vì Client kết nối với Server, nó chỉ cần gửi dữ liệu đến cho Server bằng cách sử dụng sendto(). Ngược lại, Server sử dụng lời gọi recvfrom() để chờ đợi dữ liệu từ Client gửi đến. Phương thức recvfrom() cho lại địa chỉ của Client, nhờ đó Server có thể gửi kết quả trả lời cho Client như hình 3.7.



**Hình 3.7. Hệ Socket sử dụng giao thức không định hướng kết nối**

Để đóng gói dữ liệu gửi và nhận qua UDP ta sử dụng lớp DatagramPacket. Lớp này có những phương thức sau:

- `getAddress()`, `getPort()` để đọc địa chỉ IP và cổng của đồ hình dữ liệu.
- `getLength()`, `getData()` để nhận số byte và dữ liệu trong đồ hình dữ liệu.

- `setAddress()`, `setPort()`, `setLength()`, `setData()` để đặt lại địa chỉ, cổng, độ dài và dữ liệu trên đồ hình dữ liệu.

### Ví dụ 3.6.

```
// DatagramApp.java mô tả cách sử dụng đồ hình dữ liệu DatagramSocket
import java.io.*;
import java.util.Date;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class DatagramApp{
    public static void main(String arg[]){
        try{
            DatagramSocket dsk = new DatagramSocket(1234);
            String localAddress =
                InetAddress.getLocalHost().getHostName();
            int lp = dsk.getLocalPort();
            System.out.println("Địa chỉ máy địa phương: " + localAddress);
            System.out.println("Máy chủ chờ ở cổng: " + lp);
            int i = 256;
            byte outBuffer[];
            byte inBuffer[] = new byte[i];
            DatagramPacket odg;
            DatagramPacket idg =
                new DatagramPacket(inBuffer, inBuffer.length);
            boolean finished = false;
            do {
                dsk.receive(idg);
                InetAddress daddr = idg.getAddress();
                String dbst = daddr.getHostName();
                int dpt = idg.getPort();
                System.out.println("Nhận dữ liệu từ: " + dbst
                    + "o cổng: " + dpt);
                String dt = new String(idg.getData());
            }
        }
    }
}
```

```
        System.out.println("Dữ liệu chưa là: " + dt);
        finished = true;
        String tm = new Date().toString();
        outBuffer = tm.getBytes();
        odg = new DatagramPacket(outBuffer,
                                outBuffer.length, daddr, dport);
        dsk.send(odg);
        System.out.println("Sent: " + tm + " to: " +
                           dbst + " at port: " + dport);

    }while(!finished);
}catch(IOException e){
    System.out.println("Error:" + e);
}
}
```

### **3.5. TRUY CẬP VÀO CÁC TRANG WEB**

### **3.5.1. Xem nội dung của trang Web**

Biết trước một địa chỉ URL, ví dụ <http://www.sun.com>, Web Browser có thể mở để xem các thông tin ở trang HTML đó. Tương tự, ta cũng có thể lập một applet trong Java để đọc tin từ một trang Web cho trước. Java cung cấp lớp `java.net.URL` để xử lý URL. Ví dụ,

```
try{
    URL url = new URL(http://www.sun.com);
}catch(MalformedURLException ex){}
```

Để xem nội dung của một trang HTML, ta có thể viết như sau:

```
AppletContext context = getAppletContext();  
context.showDocument(url);
```

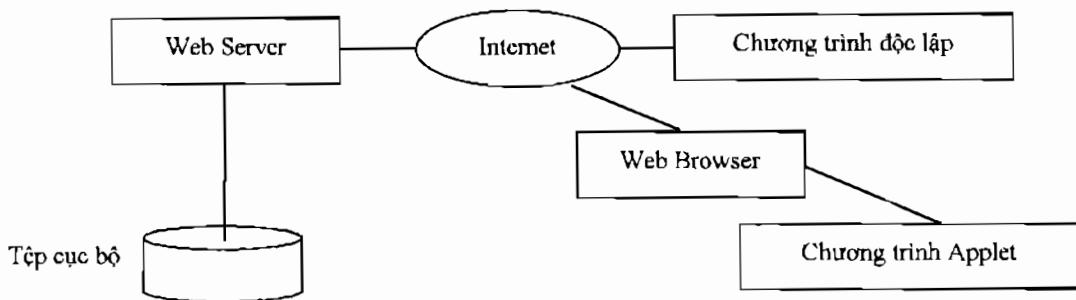
**Ví dụ 3.7.** Xây dựng một chương trình ViewWebPage đơn giản để đọc tin trong một trang Web: Nhập vào địa chỉ URL, nhấn nút “Go” thì trang Web đó sẽ được hiển thị lên màn hình.

```
// ViewWebPage.java: Truy cập vào trang HTML thông qua applet
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.applet.*;

public class ViewWebPage extends JApplet implements ActionListener{
    // Lập nút "Go" để bắt đầu truy cập theo địa chỉ đã khai báo
    private JButton button = new JButton("Go");
    private JTextField textURL = new JTextField(20);
    public void init(){
        getContentPane().setLayout(new FlowLayout());
        getContentPane().add(new JLabel("URL"));
        getContentPane().add(textURL);
        getContentPane().add(button);
        button.addActionListener(this);
    }
    // Xử lý các sự kiện
    public void actionPerformed(ActionEvent evt){
        if(evt.getSource() == button)
            try{
                AppletContext context = getAppletContext();
                // Nhận địa chỉ đã khai báo
                URL url = new URL(textURL.getText());
                context.showDocument(url);
            }catch(Exception e){
                showStatus("Error: " + e);
            }
    }
}
```

### 3.5.2. Tìm các tập tin ở Web Server

Nhiều khi ta muốn truy cập vào nội dung của các tập tin ở Web Server. Ta có thể sử dụng chương trình độc lập hoặc applet (qua Web Browser) để đọc tin trong các tập đó như hình sau.



**Hình 3.8. Các chương trình đọc tập tin trên Web Server**

Để truy cập vào một tập, ta phải sử dụng `openStream()` được định nghĩa trong lớp URL để mở một tập từ xa url.

```
InputStream is = url.openStream();
```

**Ví dụ 3.8.** Hiển thị nội dung của một tập từ Web Server. Chương trình gồm một giao diện là trường văn bản để nhập tên tập từ URL và một vùng văn bản để hiển thị nội dung của tập đó.

```
// ViewRemoteFile.java: đọc và hiển thị nội dung của một tập tin từ xa.

import java.net.*;
import java.awt.*;
import java.io.*;
import java.awt.event.*;
import javax.swing.*;

public class ViewRemoteFile extends JApplet implements
    ActionListener{
    // Tạo lập một nút "View" để nhấn khi cần xem một tập từ xa.
    private JButton view = new JButton("View");
    private JTextField jbfURL = new JTextField(20);
    private JTextArea jtaFile = new JTextArea();
    private JLabel status = new JLabel();
```

```
public void init(){
    Panel p1 = new Panel();
    p1.setLayout(new BorderLayout());
    p1.add(new JLabel("Filename "), BorderLayout.WEST);
    p1.add(jbfURL, BorderLayout.CENTER);
    p1.add(view, BorderLayout.EAST);

    getContentPane().setLayout(new FlowLayout());
    getContentPane().add(new JScrollPane(jtaFile),
                        BorderLayout.CENTER);
    getContentPane().add(p1, BorderLayout.NORTH);
    getContentPane().add(status, BorderLayout.SOUTH);
    view.addActionListener(this);
}

// Xử lý sự kiện khi nhấn “View”
public void actionPerformed(ActionEvent evt){
    if(evt.getSource() == view)
        showFile();
}

private void showFile(){
    BufferedReader infile = null;
    String inLine;
    URL url = null;
    try{
        String urlStr;
        urlStr = jbfURL.getText().trim();
        url = new URL(urlStr);
        InputStream is = url.openStream();
        // Tạo lập vùng đệm để đọc tin
        infile = new BufferedReader(new InputStreamReader(is));
        // Đọc từng dòng và ghép vào vùng văn bản được hiển thị
        while((inLine = infile.readLine()) != null){
            jtaFile.append(inLine + "\n");
        }
    }
}
```

```

        status.setText("File loaded successfully");

    }catch(FileNotFoundException e){
        status.setText("URL not found: " + url);
    }catch(IOException e){
        status.setText(e.getMessage());
    }finally{
        try{
            if(infile != null) infile.close();
        }catch(IOException e){}
    }
}

public static void main(String arg[]){
    JFrame fr = new JFrame("Read File from Web Server");
    ViewRemoteFile applet = new ViewRemoteFile();
    fr.getContentPane().add(applet, BorderLayout.CENTER);
    applet.init();
    applet.start();
    fr.setSize(400, 300);
    fr.setVisible(true);
}
}

```

### 3.6. DỊCH VỤ THƯ ĐIỆN TỬ E-MAIL

Thư điện tử cho phép ta gửi các thông điệp, các tài liệu giữa các máy tính được kết nối mạng với nhau. Đây là một trong những ứng dụng phổ biến nhất trên mạng hiện nay. Rất nhiều dịch vụ E-mail được cung cấp miễn phí trên mạng như Microsoft Mail, Yahoo Mail, Netscape Mail, v.v. Các dịch vụ này cho phép ta chuyển tài liệu (thư) từ nơi này (máy này) đến nơi khác (máy khác) tương tự như việc gửi các bức thư qua bưu điện. Trong cách gửi truyền thống, ta phải cung cấp (viết trên phong bì) địa chỉ người gửi, người nhận sau đó bỏ thư vào hòm thư bưu điện. Đối với thư điện tử E-mail cũng tương tự. Máy chủ cung cấp địa chỉ người gửi, ta cần ghi địa chỉ E-mail của người nhận. Dịch vụ E-mail sẽ chuyển thư gửi đi đến máy chủ hay Mail Server theo địa chỉ người nhận. Mail Server nơi nhận sẽ đặt bức thư nhận được vào hộp thư (Mail Box) trên máy chủ nơi nhận (dưới dạng

hệ thống tệp hay một CSDL). Người nhận muốn đọc thư phải đăng nhập vào hòm thư của mình để đọc những bức thư từ những nơi khác gửi tới.

Dịch vụ E-mail rất đa dạng, nhưng chủ yếu tập trung vào việc thiết lập:

- **Mail Server:** Chương trình phía máy chủ để nhận, phân phối, gửi thư đến các máy khác. Mail Server thực ra là một chương trình mở socket để lắng nghe các yêu cầu (gửi) từ các máy khác. Chính vì vậy, trước khi gửi hay nhận thư, ta phải biết rõ địa chỉ IP của Mail Server. Địa chỉ này thường được gọi là host mail. Các chương trình như Email, Daemon, Send Mail, Mail Exchange, v.v. là các Mail Server.
- **Mail Client:** Chương trình hoạt động phía máy khách, cho phép người dùng đăng nhập vào hộp thư của mình để đọc, viết và gửi thư cho người khác (ở máy khác). Có những chương trình Mail Client rất phổ biến như Outlook Express, Netscape Communicator. Nếu chương trình này được viết dưới dạng giao diện Web sẽ được gọi là Web Mail. Thật ra, Web Mail tương tác khó khăn hơn các ứng dụng Mail Client vì nó phải dựa vào Web Server để đọc và gửi đến Mail Server. Tuy nhiên, ưu điểm lớn nhất của Web Mail là ta có thể truy cập vào hòm thư của mình ở mọi nơi, mọi lúc, bất cứ khi nào kết nối được vào Internet. Ví dụ Hot Mail và Yahoo Mail là hai Web Mail được sử dụng rộng rãi hiện nay.
- **Giao thức E-mail SMTP:** Hiện nay người ta thường dùng giao thức chuyển thư điện tử SMTP để gửi E-mail. Chương trình chủ tiếp nhận thư điện tử theo giao thức này được gọi là SMTP Server, còn chương trình khách sử dụng nó nhận thư được gọi là SMTP Client.
- **Giao thức lưu trữ và nhận thư:** Khi Mail Server nhận thư, nó phải tổ chức lưu trữ thư theo một cách nào đó để các chương trình khách dễ dàng truy cập và tải được thư về. Hiện nay POP3 và IMAP là hai giao thức lưu trữ và lấy thư từ hộp thư khá phổ biến. Các chương trình Mail Client thường sử dụng POP3 để lấy thư từ Mail Server và hiển thị chúng ở máy khách.

### 3.6.1. Định dạng thư điện tử

Trước khi bắt tay vào việc xây dựng Java để gửi, nhận thư điện tử, ta cũng nên tìm hiểu về định dạng chuẩn của thư điện tử.

Mặc dù không bắt buộc, nhưng để mọi chương trình khách có thể dễ dàng hiển thị nội dung những bức thư nhận được thì cấu trúc của chúng cũng nên theo một khuôn dạng nhất định. Theo định dạng chuẩn RFC MIME, nội dung của mỗi bức thư điện tử gồm các phần: phần đầu (Header Mail), phần nội dung thư (Body Mail) và phần đính kèm (Mail Attachment) chứa nhiều loại tệp dữ liệu: hình ảnh, âm thanh, hay các loại tài liệu khác, v.v.

Phần đầu thư cần ghi rõ:

From:  
Date:  
To:  
Subject:

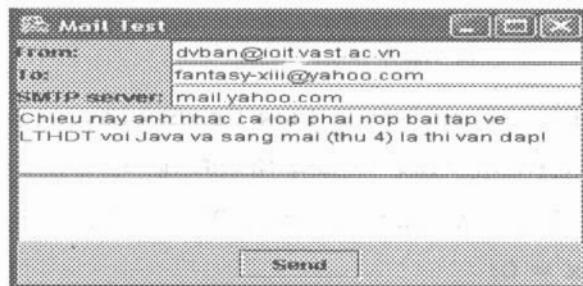
Dựa vào những qui định chuẩn, người nhận biết được thư ai gửi, ngày viết, gửi cho ai và chủ đề chính của bức thư là gì.

### 3.6.2. Gửi thư điện tử

Trong nhiều trường hợp, chúng ta muốn tự gửi đi một mẫu tin mà không cần sử dụng những dịch vụ E-mail như trên thì phải làm như thế nào?

Để gửi E-mail, ta có thể sử dụng kết nối Socket thông qua cổng 25 với giao thức SMTP. SMTP là giao thức vận chuyển mail đơn giản tạo ra format cho thư điện tử.

**Ví dụ 3.9.** Một chương trình gửi thư đơn giản.



Hình 3.9. Chương trình gửi thư MailTest

Như yêu cầu ở hình 3.9, sau khi điền các thông tin về địa chỉ người gửi, người nhận, SMTP Server, ta viết thư (ở vùng văn bản ở trên), sau đó nhấn nút “Send” để gửi thư. Trong vùng văn bản ở dưới sẽ hiển thị các câu lệnh gửi đến cho SMTP Server và sự trả lời khi nó nhận được nếu kết nối thành công, ngược lại sẽ thông báo lỗi.

// MailTest.java: Chương trình gửi thư đơn giản.

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;
import java.net.*;
import javax.swing.*;
```

```
public class MailTest{  
    public static void main(String[] args){  
        JFrame fr = new MailTestFrame();  
        fr.show();  
    }  
}  
  
class MailTestFrame extends JFrame implements ActionListener{  
    public MailTestFrame(){  
        setTitle("Mail Test");  
        setSize(300, 300);  
        addWindowListener(new WindowAdapter(){  
            public void windowClosing(WindowEvent e){  
                System.exit(0);  
            }  
        });  
        getContentPane().setLayout(new GridBagLayout());  
        GridBagConstraints gbc = new GridBagConstraints();  
        gbc.fill = GridBagConstraints.HORIZONTAL;  
        gbc.weightx = 0;  
        gbc.weighty = 0;  
  
        gbc.weightx = 0;  
        add(new JLabel("From: "), gbc, 0, 0, 1, 1);  
        gbc.weightx = 100;  
        from = new JTextField(20);  
        add(from, gbc, 1, 0, 1, 1);  
  
        gbc.weightx = 0;  
        add(new JLabel("To: "), gbc, 0, 1, 1, 1);  
        gbc.weightx = 100;  
        to = new JTextField(20);  
        add(to, gbc, 1, 1, 1, 1);  
    }  
}
```

```
gbc.weightx = 0;
add(new JLabel("SMTP server: "), gbc, 0, 2, 1, 1);
gbc.weightx = 100;
smtpServer = new JTextField(20);
add(smtpServer, gbc, 1, 2, 1, 1);

gbc.fill = GridBagConstraints.BOTH;
gbc.weighty = 100;
message = new JTextArea();
add(new JScrollPane(message), gbc, 0, 3, 2, 1);

response = new JTextArea();

add(new JScrollPane(response), gbc, 0, 4, 2, 1);
gbc.weighty = 0;
JButton sendButton = new JButton("Send");
sendButton.addActionListener(this);
JPanel buttonPanel = new JPanel();
buttonPanel.add(sendButton);
add(buttonPanel, gbc, 0, 5, 2, 1);

}

private void add(Component c, GridBagConstraints gbc, int x,
    int y, int w, int h){
    gbc.gridx = x;
    gbc.gridy = y;
    gbc.gridwidth = w;
    gbc.gridheight = h;
    getContentPane().add(c, gbc);
}

public void actionPerformed(ActionEvent e){
    SwingUtilities.invokeLater(new Runnable(){
        public void run(){
            sendMail();
        }
    });
}
```

```
public void sendMail(){
    try{
        Socket s = new Socket(smtpServer.getText(), 25);
        out = new PrintWriter(s.getOutputStream());
        in = new BufferedReader(new
            InputStreamReader(s.getInputStream()));
        String hostName =
            InetAddress.getLocalHost().getHostName();
        send(null);
        send("Hello " + hostName);
        send("RCPT to: " + to.getText());
        send("Data");
        out.println(message.getText());
        send(".");
        s.close();
    }catch(IOException e){
        response.append("Error: " + e);
    }
}

public void send(String s) throws IOException{
    if(s != null){
        response.append(s + "\n");
        out.println(s);
        out.flush();
    }
    String line;
    if((line = in.readLine()) != null)
        response.append(line + "\n");
}

private BufferedReader in;
private PrintWriter out;
private JTextField from, to, smtpServer;
private JTextArea message, response;
}
```

### 3.6.3. Kết nối theo bộ định vị tài nguyên URL để tìm tin

Nếu ta muốn phát triển chương trình ứng dụng, trong đó có dịch vụ E-mail riêng thì chắc chắn ta phải sử dụng thư viện ở mức cao hơn, bao gồm cả các giao thức chi tiết trong đó. Sun Microsoft đã phát triển JavaMail API như là mở rộng chuẩn nền của Java. Trong JavaMail API, ta chỉ cần gọi

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

để chuyển một thông điệp message đi. Thư viện này quan tâm đầy đủ đến giao thức truyền tin, vấn đề nhiều người nhận, việc xử lý phần đính kèm, v.v.

Vấn đề đặt ra ở đây là truy tìm các thông tin từ xa như thế nào?

Trong Java có hai lớp hỗ trợ để truy tìm thông tin từ xa, đó là URL và URLConnection. Việc tạo ra một đối tượng để định vị tài nguyên có thể thực hiện như sau:

```
URL url = new URL(protocol:resource);
```

Trong đó, protocol là giao thức, có thể là HTTP hoặc FTP (Java 2SDK hỗ trợ cả hai), còn resource là địa chỉ nơi cần mở để trao đổi tin. Sau đó ta có thể tạo ra vùng đệm để đọc từng dòng tin và xử lý chúng.

Nếu ta chỉ cần biết về nội dung tin chứa ở một tài nguyên xác định thì chỉ cần sử dụng lớp URL. Ví dụ,

```
URL url = new
URL("http://java.sun.com/j2se/1.4.2/docs/guide/rmi/getstart.doc.html");
InputStream uin = url.openStream();
BufferedReader in = new
BufferedReader(new InputStreamReader(uin));
String line;
while((line = in.readLine()) != null){
    process line; // Xử lý từng dòng
}
```

Ở mục 3.5 ta đã nêu cách sử dụng URL để truy tìm tin từ các nguồn từ xa.

Song, nếu ta muốn biết thêm các thông tin khác, hoặc muốn điều khiển việc truy cập vào các tài nguyên của Web thì phải sử dụng lớp URLConnection. Khi làm việc với đối tượng của URLConnection, ta cần phải thực hiện theo các bước sau.

1. Gọi `openConnection()` của URL để có được đối tượng URLConnection.  

```
URLConnection connect = url.openConnection();
```
2. Sử dụng các phương thức sau để đặt lại chúng.

```

    setDoInput()
    setDoOutput()
    setIfModifiedSince()
    setUserCaches()
    setAllowUserInteraction()
    setRequestProperty()

```

3. Kết nối với tài nguyên từ xa bằng cách gọi phương thức `connect()`.
 

```
connection.connect();
```
4. Sau khi đã kết nối được với Server, ta có thể truy vấn theo những tiêu đề, từ khoá, v.v. Có hai phương thức giúp ta liệt kê được tất cả các trường tiêu đề: `getHeaderFieldKey()` và `getHeaderField()`. Ngoài ra ta có thể sử dụng những phương thức sau:
 

<code>getContentType()</code>	- Xác định kiểu của nội dung tin
<code>getContentLength()</code>	- Xác định độ dài của nội dung tin
<code>getContentEncoding()</code>	- Xác định mã của nội dung tin
<code>getDate()</code>	- Xác định ngày tháng
<code>getLastModified()</code>	- Xác định lần cập nhật nội dung tin cuối cùng
5. Cuối cùng là truy cập vào đối tượng ở nơi cần tìm. Sử dụng phương thức `getInputStream()` để nhận được những luồng tin đọc được từ nguồn tài nguyên.

Chúng ta hãy tìm hiểu chi tiết hơn một số phương thức nêu trên. Trước tiên là `setDoInput()` và `setDoOutput()`. Theo mặc định, mọi kết nối chỉ cho phép đọc tin từ Server và không cho phép ghi. Nếu ta muốn tạo ra một luồng dữ liệu gửi đến cho Web Server thì cần gọi

```
connection.setDoOutput(true);
```

Phương thức `setIfModifiedSince()` báo cho đối tượng cần thực hiện kết nối biết rằng chúng ta chỉ quan tâm đến dữ liệu đã được cập nhật lần cuối. `setUserCaches()`, `setAllowUserInteraction()` chỉ sử dụng ở trong các Applet. Phương thức `setRequestProperty()` cho phép đặt lại các đặc tính như tên gọi người sử dụng, mật khẩu, mẫu tài liệu, v.v.

#### Ví dụ 3.10. Chương trình kết nối với tài nguyên từ xa.

```

// URLConnection.java

import java.io.*;
import java.net.*;
import java.util.*;

public class URLConnectionTest{

```

```
public static void main(String[] args) {
    try{
        String urlName;
        if(args.length > 0)
            urlName = args[0];
        else
            urlName = "http://java.sun.com";
        URL url = new URL(urlName);
        URLConnection connection = url.openConnection();
        // Lấy tên người sử dụng, mật khẩu từ các tham số trên dòng lệnh
        if(args.length > 2){
            String username = args[1];
            String password = args[2];
            String input = username + ":" + password;
            String encoding = base64Encode(input);
            connection.setRequestProperty("Authorization",
                "Basic " + encoding);
        }
        connection.connect();
        // In ra từ khoá và tiêu đề
        int n = 1;
        String key;
        while((key = connection.getHeaderField(n)) != null){
            String value = connection.getHeaderField(n);
            System.out.println(key + ": " + value);
            n++;
        }
        // In ra các tính chất của nội dung tin
        System.out.println(" -----");
        System.out.println("getContentType: " +
            connection.getContentType());
        System.out.println("getContentLength: " +
            connection.getContentLength());
```

```
System.out.println("getContentEncoding: " +
    connection.getContentEncoding());
System.out.println("getDate: " +
    connection.getDate());
System.out.println("getLastModified: " +
    connection.getLastModified());
BufferedReader in = new BufferedReader(new
    InputStreamReader(connection.getInputStream()));
System.out.println(" -----");
// In ra 10 dòng đầu tiên
String line;
n = 1;
while((line = in.readLine()) != null && n <= 10){
    System.out.println(line);
    n++;
}
if(line != null)System.out.println("... ");
}catch(IOException e){
    System.out.println("Error: " + e);
}
}

public static String base64Encode(String s){
    ByteArrayOutputStream bOut = new ByteArrayOutputStream();
    Base64OutputStream out = new Base64OutputStream(bOut);
    try{
        out.write(s.getBytes());
        out.flush();
    }catch(IOException e){
    }
    return bOut.toString();
}
}

// BASE64 mã 3 byte cho 4 ký tự. |11111122|22223333|33444444|. Mỗi tập gồm 6 bit
// được mã hóa theo ánh xạ toBase64. Mỗi dòng in ra dài nhất là 76 ký tự.
```

```
class Base64OutputStream extends FilterOutputStream{
    private static char[] toBase64 = {
        'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
        'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',
        'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
        'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f',
        'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
        'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
        'w', 'x', 'y', 'z', '0', '1', '2', '3',
        '4', '5', '6', '7', '8', '9', '+', '/'
    };
    private int col = 0;
    private int i = 0;
    private int inbuf[] = new int[3];

    public Base64OutputStream(OutputStream out){
        super(out);
    }

    public void write(int c) throws IOException{
        inbuf[i] = c;
        i++;
        if(i == 3){
            super.write(toBase64[((inbuf[0] & 0xFC) >> 2)]);
            super.write(toBase64[((inbuf[0] & 0x03) << 4) |
                ((inbuf[1] & 0xF0) >> 4)]);
            super.write(toBase64[((inbuf[1] & 0x0F) << 2) |
                ((inbuf[2] & 0xC0) >> 6)]);
            super.write(toBase64[inbuf[2] & 0x3F]);
            col += 4;
            i = 0;
            if(col >= 76){
                super.write('\n');
                col = 0;
            }
        }
    }
}
```

```
public void flush() throws IOException{
    if(i == 1){
        super.write(toBase64[(inbuf[0] & 0xFC) >> 2]);
        super.write(toBase64[((inbuf[0] & 0x03) << 4));
        super.write('=');
        super.write('=');
    }else{
        super.write(toBase64[(inbuf[0] & 0xFC) >> 2]);
        super.write(toBase64[((inbuf[0] & 0x03) << 4) |
            ((inbuf[1] & 0xF0) >> 4)]);
        super.write(toBase64[(inbuf[1] & 0x0F) << 2]);
        super.write('=');
    }
}
```

Sau khi dịch xong, ta có thể chạy chương trình trên, ví dụ

java URLConnectionTest http://home.netnam.vn user pw  
sẽ được kết quả như hình 3.10.

```
Command Prompt

D:\Users\Ban\JavaProgramming\JavaAdvanced>c:\j2sdk1.4.0\bin\java URLConnectionTest
http://home.netman.vn user pu
Server: Microsoft-IIS/4.0
Date: Thu, 28 Jul 2005 09:09:26 GMT
Content-Type: text/html
Content-Length: 1030
Content-Encoding: null
Last-Modified: Sat, 27 Jul 2005 10:00:00 GMT
Transfer-Encoding: chunked
...
getContentType: text/html
getContentLength: -1
getContentEncoding: null
getCreateDate: 1122541766000
getLastModified: 0
...
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0 Transitional//EN">
<html>
<head><meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="Author" content="Nguyen The Binh">
<meta name="description" content="NetName Corp., ISP since 1993, ICP since 2001,
Network Solution Provider, B2B, B2C, EG Portal Company in Vietnam.">
<meta name="keywords" content="vietnam, vn, internet, netman, iot, nest, isp, i
cp, intranet, extranet, firewall, database, dns, dynamic web,
vietnam gateway, news, clubs, culture, fashion, sport, b2b, b2c, egm, econ
merce,
technology, software, portal, computer science, it, information, application, as
y">
...

```

Hình 3.10. Chương trình truy vấn tin từ xa dựa trên URL

## BÀI TẬP

- 3.1. Viết chương trình để thực hiện trò chơi đánh cờ ca-rô (TicTacToe) trên mạng: cho phép nhiều người chơi ở những máy khác nhau ở mọi nơi trên Internet. Máy Server tạo ra một socket phục vụ kết nối từng cặp người tham gia chơi cờ ca-rô theo từng ván cờ. Mỗi phiên là một luồng đảm bảo sự trao đổi giữa hai người chơi và xác định trạng thái (thắng / thua) của ván cờ. Trong mỗi ván cờ, người đầu kết nối với Server được gọi là người thứ nhất được gán với ký hiệu ‘X’, còn người thứ hai được gán với ký hiệu ‘O’. Họ thay phiên nhau đánh vào những ô dự kiến (nhấn vào ô mà mình định đánh) để nhanh chóng đạt được số ô (ví dụ 3, hay 4) thắng hàng theo hàng ngang, hàng dọc và đường chéo. Người nào đánh được số ô đó trước, người đó thắng cờ. Số các ván cờ được tạo ra trên Server là không hạn chế.
- 3.2. Viết một chương trình Applet để đếm số người đã và đang mở một trang Web. Mỗi khi trang Web được một người mở xem thì Applet gửi yêu cầu cho Server tăng số đếm và gửi lại để Applet hiển thị số đó lên màn hình theo dõi.
- 3.3. Phát triển chương trình đăng ký học tập trên mạng (ví dụ 3.3) bằng cách bổ sung thêm nút “View”. Ngoài chức năng nhập họ tên, địa chỉ và thành phố rồi nhấn nút “Register” để đăng ký như ở ví dụ 3.3, khách hàng cũng có thể nhập họ tên rồi nhấn “View” để xem tên mình đã có trong danh sách đăng ký hay chưa. Nếu có thì toàn bộ thông tin về người đó được hiển thị tương ứng trong các trường, ngược lại thông báo “Không có tên bạn trong danh sách”.
- 3.4. Xây dựng một diễn đàn (Forum) để mọi người đưa ra ý kiến bàn bạc, thảo luận liên quan đến một chủ đề nào. Toàn bộ thông tin về các thành viên và nội dung trao đổi giữa họ được lưu giữ ở Server. Trong Forum ta có thể đăng ký là thành viên của nhóm. Tại mỗi chủ đề, một thành viên có thể đưa ra ý kiến mới hay trả lời, thảo luận những ý kiến đã được nêu ra trên diễn đàn. Việc truy cập, truyền dữ liệu được thực hiện thông qua mạng nhằm phục vụ cho việc kiểm soát, cập nhật, trao đổi thông tin nhanh chóng và tiện lợi.
- 3.5. Xây dựng chương trình có menu gồm hai chức năng chính. Chức năng thứ nhất là “Đọc thư” cho phép người dùng đăng nhập vào hộp thư khi biết địa chỉ IP của Mail Server, tên người dùng và mật khẩu. Khi kết nối được với Mail Server thì hiển thị hộp thư gồm: tên người gửi, ngày gửi, chủ đề từng bức thư. Chức năng thứ hai “Cấu hình” cho phép người dùng thêm mới thông tin Account, sửa, hay xoá các thông tin, v.v. của mỗi người.

## Chương 4

# LẬP TRÌNH VỚI SWING NÂNG CAO

---

Chương IV trình bày cách sử dụng các thành phần của Swing phát triển:

- ✓ Các cấu trúc cây Tree
- ✓ Tổ chức bảng Table
- ✓ Cách tổ chức các thành phần của Swing trong các chương trình ứng dụng như thanh trượt và thước đo.

### 4.1. CẤU TRÚC CÂY TREE

Những ai đã sử dụng máy tính thì đều biết rằng, hệ thống tệp, thư mục trong mỗi ổ đĩa đều được tổ chức theo cấu trúc cây. Những người lập trình hướng đối tượng với Java cũng đã rất quen với các cây thừa kế của các lớp. Mọi lớp của Java đều là lớp con của lớp Object ([2], [6]). Trong thực tế cũng có rất nhiều cơ cấu được tổ chức thành cấu trúc cây, như cơ cấu tổ chức của các cơ quan, xí nghiệp, cơ quan hành chính của các quốc gia, các tinh thành, v.v.

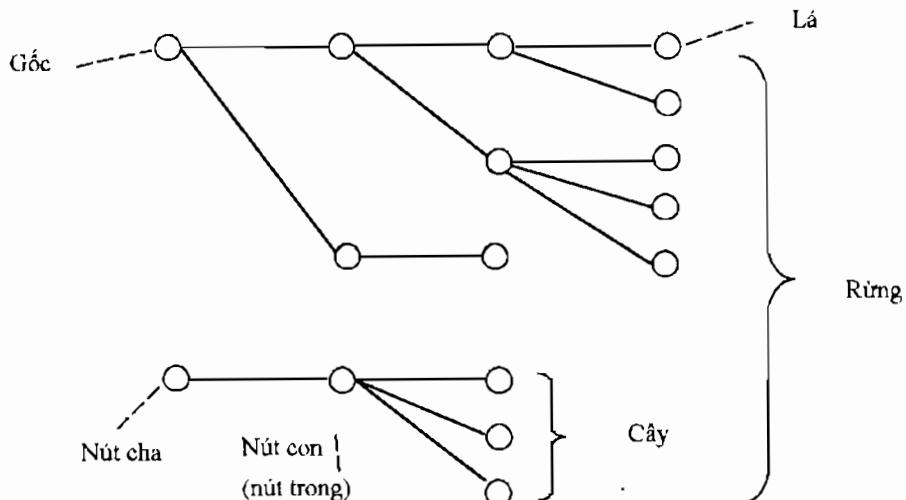
Người lập trình thường xuyên phải hiển thị những thông tin nêu trên dưới dạng cấu trúc cây. Rất may, thư viện javax.swing có lớp JTree giúp ta dễ dàng thực hiện được mục đích đó. javax.swing hỗ trợ để tạo ra những chương trình hoạt động giống như một thành phần hay một nhóm các thành phần. Trong tài liệu “Lập trình hướng đối tượng với Java” [1] chúng ta đã làm quen với những lớp cơ bản của javax.swing, tiếp theo ở đây chúng ta tìm hiểu sâu hơn về những đặc tính nâng cao của chúng.

Trước khi đi sâu tìm hiểu về cách sử dụng JTree, ta nhắc lại một số khái niệm cơ sở liên quan đến cấu trúc cây. Cây bao gồm *các nút* (đỉnh). Mỗi nút có thể là *nút lá* (gọi tắt là lá, nút không có nút con) hoặc là *nút trong* (gốc của cây con, nút có ít nhất một nút con). Trong cây có một nút đặc biệt không có nút cha được gọi là *gốc của cây*. Một tuyển tập các cây được gọi là *một rừng*. Hình 4.1 mô tả các thành phần của cây và rừng.

### 4.1.1. Cây đơn giản

Giống như hầu hết các thành phần khác của javax.swing, JTree được tổ chức theo mẫu điều khiển. Ta chỉ cần cung cấp mô hình dữ liệu phân cấp, thành phần này sẽ hiển thị chúng theo cấu trúc cây cho chúng ta. Để tạo ra JTree, ta sử dụng mô hình cây TreeModel.

```
TreeModel model = ...;
JTree tree = new JTree(model);
```



**Hình 4.1. Cấu trúc của cây**

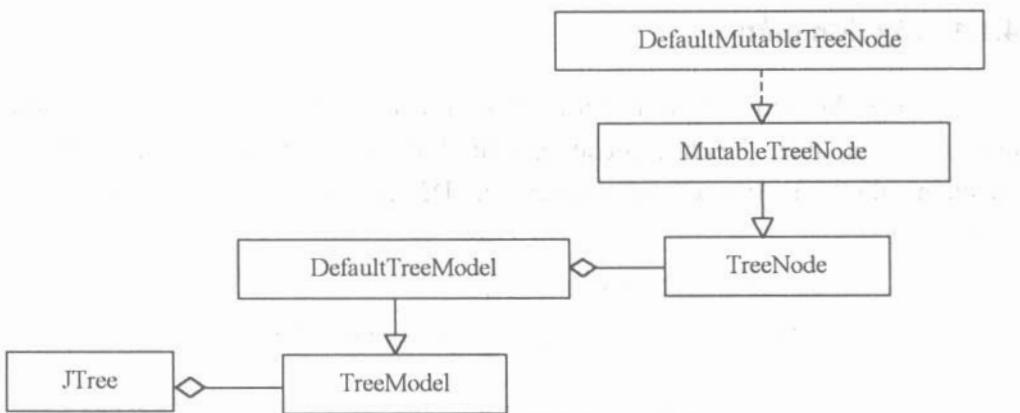
Vấn đề đặt ra là làm thế nào tạo ra được mô hình cây? Phần sau chúng ta sẽ đề cập đến cách xây dựng lớp mô hình cây riêng trên cơ sở cài đặt giao diện TreeModel. Ở đây, để đơn giản ta có thể áp dụng DefaultTreeModel trong javax.swing để tạo ra mô hình cây với nút gốc là root theo giao diện TreeNode.

```
TreeNode root = ...
DefaultTreeModel model = new DefaultTreeModel(root);
```

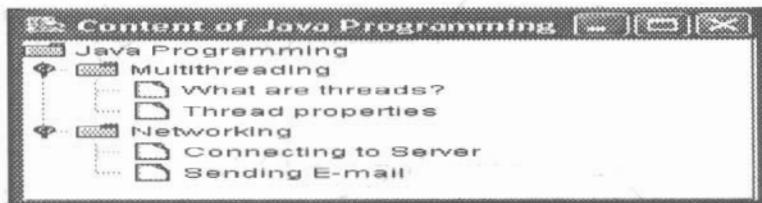
Để lưu một đối tượng cụ thể ở một nút của cây ta sử dụng DefaultMutableTreeNode. Lớp này cài đặt giao diện MutableTreeNode như hình 4.2.

Ta có thể đưa các đối tượng vào trong toán tử tạo lập của DefaultMutableTreeNode hoặc sử dụng phương thức setUserObject(). Ví dụ đặt obj vào node:

```
DefaultMutableTreeNode node = new DefaultMutableTreeNode(obj1);
node.setUserObject(obj2); //Đặt tiếp obj2 vào node
```

**Hình 4.2.** Các lớp của mô hình cây

**Ví dụ 4.1.** Xây dựng cây hiển thị mục lục của cuốn sách lập trình Java.

**Hình 4.3.** Hiển thị cây mục lục của cuốn sách “Java Programming”

// SimpleTree.java Tạo lập cây mục lục của một cuốn sách

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.tree.*;

public class SimpleTree{
    public static void main(String args[]){
        JFrame fr = new SimpleTreeFrame();
        fr.show();
    }
}

class SimpleTreeFrame extends JFrame{
    public SimpleTreeFrame(){
        setTitle("Content of Java Programming");
        setSize(300, 200);
        addWindowListener(new WindowAdapter(){
    
```

```

        public void windowClosing(WindowEvent e){
            System.exit(0);
        }
    });

// Thiết lập mô hình dữ liệu cây phân cấp
DefaultMutableTreeNode root
    = new DefaultMutableTreeNode("Java Programming");
DefaultMutableTreeNode chapter1
    = new DefaultMutableTreeNode("Multithreading");
root.add(chapter1);
DefaultMutableTreeNode title1
    = new DefaultMutableTreeNode("What are threads?");
chapter1.add(title1);
DefaultMutableTreeNode title2
    = new DefaultMutableTreeNode("Thread properties");
chapter1.add(title2);
DefaultMutableTreeNode chapter2
    = new DefaultMutableTreeNode("Networking");
root.add(chapter2);
DefaultMutableTreeNode title3
    = new DefaultMutableTreeNode("Connecting to Server");
chapter2.add(title3);
DefaultMutableTreeNode title4
    = new DefaultMutableTreeNode("Sending E-mail");
chapter2.add(title4);

// Tạo lập một cây và đặt nó vào thanh trượt
JTree tree = new JTree(root);
Container contentPane = getContentPane();
contentPane.add(new JScrollPane(tree));
}
}

```

Khi chạy chương trình trên, cây đầu tiên ta nhìn thấy chỉ có nút gốc và các nút con trực tiếp của nó là tên các chương. Nếu muốn xem nội dung của chương nào thì phải nhấn vào nút

tương ứng (*nút đóng có đầu nhọn chỉ sang phải*) để mở cây con thể hiện nội dung của chương đó. Sau khi xem xong, muốn trở về nội dung chính trước đó (đóng lại nút đó) ta nhấn lại nút vừa rồi (*nút được mở có đầu nhọn chỉ xuống dưới*) một lần nữa. Tương tự, ta thực hiện cho mọi cây con giống như đối với nút gốc.

### javax.swing.JTree

### API

- `JTree(TreeModel model)`: Tạo lập một cây từ mô hình cây.
- `JTree(TreeNode root)`: Tạo lập một cây có gốc là `root`.
- `JTree(TreeNode root, boolean asksAllowChildren)`: Tạo lập một cây và không cho phép các nút con đến được biểu tượng lá nếu biến `boolean` là `true`.
- `void setShowsRootHandles(boolean b)`: Nếu `b` là `true`, thì nút gốc có bộ điều khiển cho phép đóng lại hoặc mở ra xem các cây con của nó.
- `void setRootVisible(boolean b)`: Nếu `b` là `true`, thì nút gốc được hiển thị, ngược lại nó bị che khuất.

### javax.swing.tree.TreeNode

### API

- `boolean isLeaf()`: Trả lại `true` nếu nút tương ứng là lá.
- `boolean getAllowsChildren()`: Trả lại `true` nếu nút đó có các nút con.

### javax.swing.tree.MutableTreeNode

### API

- `void setUserObject(Object obj)`: Đặt đối tượng `obj` vào nút hiện thời.

### javax.swing.tree.DefaultMutableTreeNode

### API

- `DefaultMutableTreeNode(Object obj)`: Tạo ra một nút bằng đối tượng `obj`.
- `void add(MutableTreeNode child)`: Bổ sung nút con `child` vào nút hiện thời.
- `void setAllowsChildren(boolean b)`: Nếu `b` là `true` thì được phép bổ sung các nút con cho nút đó.

## 4.1.2. Soạn thảo cây và đường đi

Trong cấu trúc cây, nhiều khi đòi hỏi phải bổ sung thêm những nút mới hoặc bỏ đi một số nút, nghĩa là phải soạn thảo lại cây cho phù hợp với bài toán ứng dụng. Ngoài ra, để theo dõi được cấu trúc của cây thì phải tìm ra được những nút nào đã được chọn, nghĩa là phải chỉ ra được đường đi từ gốc đến nút đó. *Đường đi trong cây* là dãy các nút con bắt đầu từ gốc. Đường đi được sử dụng để xác định các nút trên cây. Trong Swing có lớp `TreePath` quản

lý dãy các tham chiếu đối tượng (không phải là các nút). Khi ta có một đường đi của cây, nếu muốn biết nút cuối cùng thì gọi phương thức getLastPathComponent(), còn nếu muốn biết nút hiện thời có ở trên cây hay không thì sử dụng getSelectionPath().

Khi một nút được chọn, ví dụ selectedNode, ta có thể thêm, hoặc xoá nút đó. Truy nhiên không thể sử dụng hàm add() để chèn thêm một nút con vào

```
selectedNode.add(newNode); // Không thực hiện được
```

Ngược lại, nếu muốn thay đổi cấu trúc của cây thì phải thay đổi cấu trúc của các nút. Ta có thể sử dụng

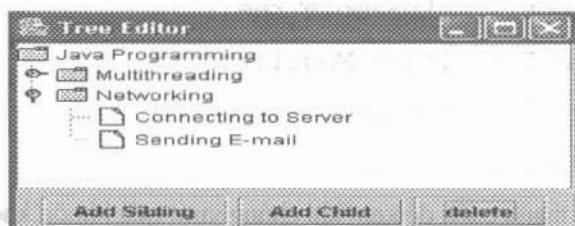
```
model.insertNodeInto(newNode, selectedNode, selectedNode.getChildCount());
```

để chèn thêm một nút mới như là nút lá vào cây con có gốc là selectedNode theo mô hình cây mặc định model.

Tương tự, ta có thể xoá đi một nút (một cây con) đã chọn

```
model.removeNodeFromParent(selectedNode);
```

**Ví dụ 4.2.** Xây dựng một cấu trúc cây, trên đó thực hiện được các chức năng thêm hoặc loại bỏ đi một nút (cây con) bất kỳ. Khi đã chọn một nút trong và muốn thêm một nút mới cùng mức với nút đó thì nhấn “Add Sibling”, thêm một nút lá của cây con thì nhấn “Add Child”, còn muốn xoá nút đó thì nhấn “Delete”. Mỗi nút mới được bổ sung vào có tên là “New Node” sau đó có thể nhấn đúp chuột để sửa lại tên của các nút.



Hình 4.4. Thêm bớt các nút trên cây

/\* TreeEdit.java Tạo ra cấu trúc cây và khả năng chèn thêm, xoá bỏ đi các nút như hình 4.4.

```
/*
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.tree.*;
public class TreeEdit{
    public static void main(String args[]){
        JFrame fr = new TreeEditFrame();
        fr.show();
    }
}
```

```
        }

    }

class TreeEditFrame extends JFrame implements ActionListener{
    private DefaultTreeModel model;
    private JTree tree;
    private JButton addSiblingButton, addChildButton;
    private JButton deleteButton, editButton;
    public TreeEditFrame(){
        setTitle("Tree Editor");
        setSize(300, 200);
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
        //Xây dựng cây
        TreeNode root = makeSampleTree();
        model = new DefaultTreeModel(root);
        tree = new JTree(model);
        tree.setEditable(true);
        //Đưa cây vào thanh cuộn để xem khi chúng vượt quá khuôn khổ khung nhìn
        Container content = getContentPane();
        JScrollPane scrollPane = new JScrollPane(tree);
        content.add(scrollPane, "Center");
        //Tạo lập bảng các nút chèn thêm, xoá bỏ các nút
        JPanel panel = new JPanel();
        addSiblingButton = new JButton("Add Sibling");
        addSiblingButton.addActionListener(this);
        panel.add(addSiblingButton);
        addChildButton = new JButton("Add Child");
        addChildButton.addActionListener(this);
        panel.add(addChildButton);
        deleteButton = new JButton("delete");
        panel.add(deleteButton);
    }
}
```

```
deleteButton.addActionListener(this);
panel.add(deleteButton);
content.add(panel, "South");
}

public TreeNode makeSampleTree() {
    DefaultMutableTreeNode root
        = new DefaultMutableTreeNode("Java Programming");
    DefaultMutableTreeNode chapter1
        = new DefaultMutableTreeNode("Multithreading");
    root.add(chapter1);
    DefaultMutableTreeNode title1
        = new DefaultMutableTreeNode("What are threads?");
    chapter1.add(title1);
    DefaultMutableTreeNode title2
        = new DefaultMutableTreeNode("Thread properties");
    chapter1.add(title2);
    DefaultMutableTreeNode chapter2
        = new DefaultMutableTreeNode("Networking");
    root.add(chapter2);
    DefaultMutableTreeNode title3
        = new DefaultMutableTreeNode("Connecting to Server");
    chapter2.add(title3);
    DefaultMutableTreeNode title4
        = new DefaultMutableTreeNode("Sending E-mail");
    chapter2.add(title4);
    return root;
}

public void actionPerformed(ActionEvent event) {
    DefaultMutableTreeNode selectedNode = (DefaultMutableTreeNode)
        tree.getLastSelectedPathComponent();
    if(selectedNode == null) return;
    if(event.getSource().equals(deleteButton)) {
        if(selectedNode.getParent() != null)
```

```

        model.removeNodeFromParent(selectedNode);
        return;
    }

    // Chèn thêm nút mới
    DefaultMutableTreeNode newNode
        = new DefaultMutableTreeNode("New Node");
    if(event.getSource().equals(addSiblingButton)) {
        DefaultMutableTreeNode parent
            = (DefaultMutableTreeNode)selectedNode.getParent();
        if(parent != null){
            int selectedIndex = parent.getIndex(selectedNode);
            model.insertNodeInto(newNode, parent, selectedIndex + 1);
        }
    } else if(event.getSource().equalsaddChildButton))
        model.insertNodeInto(newNode, selectedNode,
            selectedNode.getChildCount());
}

// Hiển thị nút mới
TreeNode[] nodes = model.getPathToRoot(newNode);
TreePath path = new TreePath(nodes);
tree.scrollPathToVisible(path);
}
}

```

**javax.swing.JTree****API**

- `TreePath getSelectionPath()`: Xác định đường đi tới nút hiện thời
- `Object getLastSelectedComponent()`: Xác định đối tượng biểu diễn cho thành phần của nút hiện thời (nút được chọn).
- `void makeVisible(TreePath path)`: Mở ra để nhìn được tất cả các nút theo đường dẫn.
- `void scrollPathToVisible(TreePath path)`: Mở các nút theo đường dẫn và nếu cây để trong thanh cuộn thì nó đảm bảo luôn nhìn thấy được những nút cuối.

**javax.swingTreeNode****API**

- `TreeNode getParent()`: Tìm nút cha của nút hiện thời.

- `TreeNode getChildAt(int index)`: Tìm nút con ở vị trí index.
- `int getChildCount()`: Đếm số nút con của nút hiện thời.

javax.swing.DefaultTreeModel

API

- `void insertNodeInto (MutableTreeNode newChild, MutableTreeNode parent, int index)`: Chèn thêm nút newChild vào như là nút con của parent và ở vị trí index.
- `void removeNodeFromParent(MutableTreeNode node)`: Loại bỏ đi node.
- `void nodeChanged(TreeNode node)`: Thông báo cho mô hình cây rằng nút này phải thay đổi.
- `void reload()`: Nạp tất cả các nút vào mô hình model.

#### 4.1.3. Đánh số các nút trên cây

Để tìm một nút trên cây, người ta thường bắt đầu từ gốc sau đó tìm qua tất cả các nút. Đây là vấn đề khó, mất nhiều thời gian khi số nút trên cây là khá lớn. Trong lý thuyết đồ thị, các thuật toán duyệt trên cây có thể thực hiện theo chiều rộng hoặc theo chiều sâu. Lớp DefaultMutableTreeNode cũng có một số phương thức khá tiện lợi cho việc đánh số các nút và thực hiện tìm lặp lại theo các nút. Phương thức `breadthFirstEnumeration()` đánh số theo chiều rộng và `depthFirstEnumeration()` đánh số theo chiều sâu được thực hiện như hình 4.5. Đánh số theo chiều rộng thực hiện theo *thuật toán duyệt gốc trước*, sau đó duyệt tất cả các nút con của nó. Thuật toán này lặp lại đệ qui cho các nút con của cây. Đánh số theo chiều sâu thực hiện theo *thuật toán duyệt gốc sau cùng*, duyệt qua tất cả các nút lá, sau đó mới đến nút gốc của chúng. Tất nhiên, ở đây giả thiết một đối tượng được lặp lại hai lần trên cây.

Ta có thể sử dụng mẫu sau để duyệt cây theo chiều sâu:

```
Enumeration breadthFirst = node.breadthFirstEnumeration();
while(breadthFirst.hasMoreElement()) {
    // Thực hiện đối với breadthFirst.nextElement()
}
```

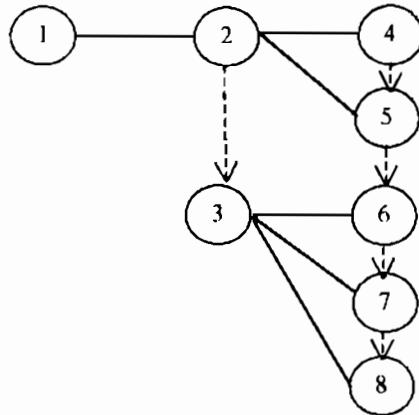
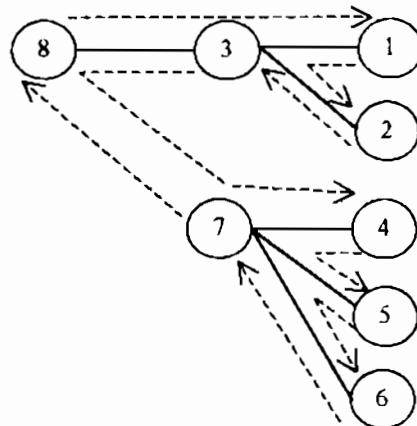
Để tìm một đối tượng trên cây, ta có thể sử dụng phương thức dạng:

```
public DefaultMutableTreeNode findUserObject(Object obj) {
    Enumeration e = root.breadthFirstEnumeration();
    while(e.hasMoreElement()) {
        DefaultMutableTreeNode node
            = (DefaultMutableTreeNode)e.nextElement();
        if(node.getUserObject().equals(obj))
            return node;
    }
}
```

```

    }
    return null;
}

```

**Hình 4.5. Duyệt theo chiều rộng****Duyệt theo chiều sâu**

#### 4.1.4. Thay đổi biểu diễn các nút của cây

Trong các chương trình ứng dụng ta thường có nhu cầu thay đổi cách vẽ các nút của cây. Một trong những thay đổi hay gặp phải là dùng một biểu tượng khác, phù hợp với bài toán ứng dụng, với trực quan của người sử dụng để biểu diễn các nút trên cây. Theo mặc định, lớp `JTree` sử dụng `DefaultTreeCellRenderer` để vẽ cho các nút. Ta cũng có thể thay đổi lại chúng theo ba cách.

1. Thay đổi biểu tượng, font chữ và màu nền được sử dụng bởi `DefaultTreeCellRenderer`.
2. Cài đặt lớp mới kế thừa `DefaultTreeCellRenderer` để thay đổi biểu tượng, font chữ và màu nền cho mỗi nút.
3. Xây dựng một lớp cài đặt giao diện `TreeCellRenderer` để vẽ ảnh cho từng nút của cây.

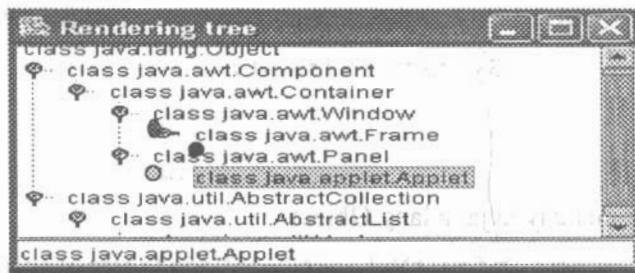
Ví dụ, có ba hình tròn: muốn hình xanh biểu diễn cho lá, hình đỏ cho nút đóng và hình vàng cho nút mở, ta có thể làm như sau:

```

DefaultTreeCellRenderer render
    = new DefaultTreeCellRenderer();
render.setLeafIcon(new ImageIcon("blueBall.gif"));
render.setClosedIcon(new ImageIcon("redBall.gif"));
render.setOpenIcon(new ImageIcon("yellowBall.gif"));
tree.setCellRenderer(render);

```

**Ví dụ 4.3.** Xây dựng cây các lớp kế thừa của `java.lang.Object`, trong đó hình tròn màu xanh biểu diễn cho lá, màu đỏ là nút đóng, còn nút vàng là nút mở. Ta có thể sử dụng những hệ soạn thảo ảnh như Photoshop để tạo ra các biểu tượng: hình tròn (tương đối nhỏ) với các màu nêu trên. Từ trường văn bản ở phía dưới, ta có thể nhập vào tên các lớp trong thư viện của Java, ví dụ `java.applet.Applet`. Nếu lớp đó chưa có trên cây thì nó được bổ sung vào cây theo đúng thứ tự thừa kế giữa các lớp.



Hình 4.6. Cây thừa kế giữa các lớp của Java

// ClassTree.java: Xây dựng cây thừa kế giữa các lớp của Java.

```

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.lang.reflect.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.tree.*;

public class ClassTree{
    public static void main(String args[]){
        JFrame fr = new ClassTreeFrame();
        fr.show();
    }
}

class ClassTreeFrame extends JFrame implements
    ActionListener{
    private DefaultMutableTreeNode root;
    private DefaultTreeModel model;
    private JTree tree;
    
```

```
private JTextField textField;

public ClassTreeFrame(){
    setTitle("Rendering tree");
    setSize(300, 200);
    addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e){
            System.exit(0);
        }
    });
    // Gốc của cây là java.lang.Object
    root = new DefaultMutableTreeNode(java.lang.Object.class);
    model = new DefaultTreeModel(root);
    tree = new JTree(model);
    tree.setEditable(true);
    // Bổ sung lớp hiện thời vào cây cùng với dữ liệu
    addClass(getClass());
    /* Đặt thêm các biểu cho các nút: hình xanh cho lá, đỏ cho nút đóng và vàng
    cho nút mở
    */
    ClassNameTreeCellRender render
        = new ClassNameTreeCellRender();
    render.setLeafIcon(new ImageIcon("blueBall.gif"));
    render.setClosedIcon(new ImageIcon("redBall.gif"));
    render.setOpenIcon(new ImageIcon("yellowBall.gif"));
    tree.setCellRenderer(render);

    Container content = getContentPane();
    JScrollPane scrollPane = new JScrollPane(tree);
    content.add(scrollPane, "Center");
    // Tên các lớp mới được nhập vào từ trường văn bản.
    textField = new JTextField();
    textField.addActionListener(this);
    content.add(textField, "South");
}
```

```
public void actionPerformed(ActionEvent event) {
    // Bổ sung thêm lớp mới được nhập vào cây
    try{
        String text = textField.getText();
        addClass(Class.forName(text));
        // Xoá trường văn bản
        textField.setText("");
    }catch(ClassNotFoundException e){
        Toolkit.getDefaultToolkit().beep();
    }
}

// Tìm nút chứa đối tượng của người sử dụng
public DefaultMutableTreeNode findUserObject(Object obj){
    Enumeration e = root.breadthFirstEnumeration();
    while(e.hasMoreElements()){
        DefaultMutableTreeNode node
            = (DefaultMutableTreeNode)e.nextElement();
        if(node.getUserObject().equals(obj))
            return node;
    }
    return null;
}

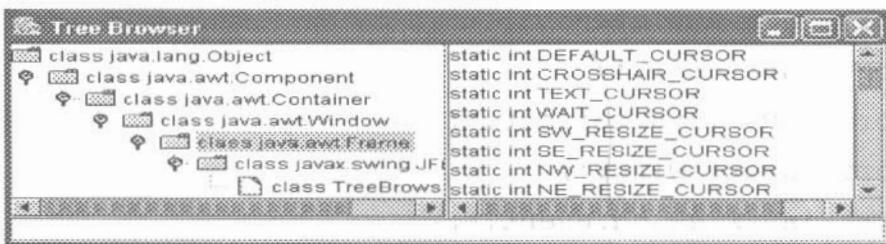
public DefaultMutableTreeNode addClass(Class c){
    // Bổ sung lớp mới vào cây
    if(c.isInterface() || c.isPrimitive()) return null;
    // Bỏ qua những kiểu không phải lớp
    DefaultMutableTreeNode node = findUserObject(c);
    if(node != null) return node;
    // Nếu lớp không có thì bổ sung lớp cha vào cây và lặp lại
    Class s = c.getSuperclass();
    DefaultMutableTreeNode parent;
    if(s == null)
        parent = root;
    else
```

```
        parent = addClass(s);
        // Bổ sung lớp mới như là lớp con
        DefaultMutableTreeNode newNode = new DefaultMutableTreeNode(c);
        model.insertNodeInto(newNode, parent, parent.getChildCount());
        // Cho phép hiển thị
        TreePath path = new TreePath(model.getPathToRoot(newNode));
        tree.makeVisible(path);
        return newNode;
    }
}

class ClassNameTreeCellRender extends DefaultTreeCellRenderer{
    public Component getTreeCellRendererComponent(JTree tree, Object value, boolean selected, boolean expanded, boolean leaf, int row,boolean hasFocus){
        super.getTreeCellRendererComponent(tree, value, selected, expanded, leaf, row, hasFocus);
        // Xác định đối tượng nhập vào
        DefaultMutableTreeNode node = (DefaultMutableTreeNode)value;
        Class c = (Class)node.getUserObject();
        // Lần đầu, đặt font chữ nghiêng thay font chữ chân phương
        if(plainFont == null){
            plainFont = getFont();
            if(plainFont != null)
                italicFont = plainFont.deriveFont(Font.ITALIC);
        }
        // Đặt font nghiêng nếu lớp là lớp trừu tượng
        if((c.getModifiers() & Modifier.ABSTRACT) == 0)
            setFont(plainFont);
        else
            setFont(italicFont);
        return this;
    }
    private Font plainFont;
    private Font italicFont;
}
```

#### 4.1.5. Hiển thị thông tin của các nút

Thông thường, mỗi thành phần (nút) của cây được gắn với một thành phần khác. Khi người sử dụng lựa chọn một số nút trên cây thì những thông tin chứa trên những nút đó có thể được hiển thị ở cửa sổ bên cạnh. Ví dụ, trên cây thừa kế các lớp của Java, nếu ta chọn `java.awt.Frame` thì các thuộc tính (biến tham chiếu, biến tĩnh) của nó được hiển thị ở cửa sổ bên phải như hình 4.7.



Hình 4.7. Hiển thị các thuộc tính tĩnh và tham chiếu của lớp

**Ví dụ 4.4.** Chương trình này khác với chương trình trước ở chỗ phải xử lý các sự kiện khi người sử dụng chọn một lớp trên cây để xem nó có những thành phần nào. Mỗi khi một nút lớp được chọn thì nội dung tương ứng với nó được hiển thị ở cửa sổ bên cạnh.

Để xử lý các sự kiện như thế ta viết thêm

```
public void valueChanged(TreeSelectionEvent event) {
    // Bạn chọn các nút khác để xem các thành phần của lớp
    TreePath path = tree.getSelectionPath();
    if(path == null) return;
    DefaultMutableTreeNode selectedNode
        = (DefaultMutableTreeNode)path.getLastPathComponent();
    Class c = (Class)selectedNode.getUserObject();
    String descr = getFieldDescription(c);
    textArea.setText(descr);
}
```

Ngoài ra, việc chọn các nút trên cây có thể chọn từng nút hoặc chọn một dãy các nút liền nhau, hoặc các nút không liên tục. Cách chọn này được xác định theo mode *chọn*. Lớp `JTree` sử dụng `TreeSelectionMode` để điều khiển mode chọn bằng cách chọn từng nút là `SINGLE_SELECTION`, hoặc `CONTIGUOUS_TREE_SELECTION` để chọn liên tục, `DISCONTINUED_SELECTION` (chọn không liên tục là mặc định). Trong ví dụ này ta chọn từng nút (ứng với các lớp) để hiển thị.

```
int mode = TreeSelectionModel.SINGLE_TREE_SELECTION;
```

Để tìm được nút hiện thời trên cây, ta truy vấn vào cây qua phương thức

TreePath path = tree.getSelectionPath();  
nhằm tìm ra một đối tượng trên đường dẫn ứng với nút đã được chọn. Ta sử dụng phương thức getFieldDescription(c) để hiển thị các thành phần của lớp c trong vùng văn bản bên cạnh.

```
// TreeBrowser.java
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.lang.reflect.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.tree.*;

public class TreeBrowser{
    public static void main(String args[]){
        JFrame fr = new TreeBrowserFrame();
        fr.show();
    }
}

class TreeBrowserFrame extends JFrame
    implements ActionListener, TreeSelectionListener{
    // Các thuộc tính
    private DefaultMutableTreeNode root;
    private DefaultTreeModel model;
    private JTree tree;
    private JTextField textField;
    private JTextArea textArea;

    public TreeBrowserFrame(){
        setTitle("Tree Browser");
        setSize(300, 200);
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
    }

    public void actionPerformed(ActionEvent e){
        if(e.getSource() == tree){
            TreePath path = tree.getSelectionPath();
            Object[] components = path.getPath();
            for(int i = 0; i < components.length; i++){
                Object component = components[i];
                if(component instanceof DefaultMutableTreeNode){
                    DefaultMutableTreeNode node = (DefaultMutableTreeNode) component;
                    String description = node.getFieldDescription("name");
                    textField.setText(description);
                }
            }
        }
    }

    public void valueChanged(TreeSelectionEvent e){
        TreePath path = e.getNewSelectionPath();
        Object[] components = path.getPath();
        for(int i = 0; i < components.length; i++){
            Object component = components[i];
            if(component instanceof DefaultMutableTreeNode){
                DefaultMutableTreeNode node = (DefaultMutableTreeNode) component;
                String description = node.getFieldDescription("name");
                textField.setText(description);
            }
        }
    }
}
```

```
});  
// Gốc của cây là Object  
root = new DefaultMutableTreeNode(java.lang.Object.class);  
model = new DefaultTreeModel(root);  
tree = new JTree(model);  
tree.setEditable(true);  
// Bổ sung lớp vào cây cùng với dữ liệu  
addClass(getClass());  
// Đặt mode để chọn  
tree.addTreeSelectionListener(this);  
int mode = TreeSelectionModel.SINGLE_TREE_SELECTION;  
tree.getSelectionModel().setSelectionMode(mode);  
// Tạo lập vùng để hiển thị văn bản  
textArea = new JTextArea();  
// Bổ sung vùng văn bản vào panel  
JPanel panel = new JPanel();  
panel.setLayout(new GridLayout(1, 2));  
panel.add(new JScrollPane(tree));  
panel.add(new JScrollPane(textArea));  
  
Container content = getContentPane();  
content.add(panel, "Center");  
// Nhận tên của các lớp mới từ trường văn bản  
textField = new JTextField();  
textField.addActionListener(this);  
content.add(textField, "South");  
}  
  
public void actionPerformed(ActionEvent event) {  
try{  
    String text = textField.getText();  
    addClass(Class.forName(text));  
    //  
    textField.setText("");  
}
```

```
        }catch(ClassNotFoundException e){
            Toolkit.getDefaultToolkit().beep();
        }
    }

    public void valueChanged(TreeSelectionEvent event){
        // Bạn chọn các nút khác để xem các thành phần của lớp
        TreePath path = tree.getSelectionPath();
        if(path == null) return;
        DefaultMutableTreeNode selectedNode
            = (DefaultMutableTreeNode)path.getLastPathComponent();
        Class c = (Class)selectedNode.getUserObject();
        String descr = getFieldDescription(c);
        textArea.setText(descr);
    }

    public DefaultMutableTreeNode findUserObject(Object obj){
        // Tìm nút chứa đối tượng của người sử dụng
        Enumeration e = root.breadthFirstEnumeration();
        while(e.hasMoreElements()){
            DefaultMutableTreeNode node
                = (DefaultMutableTreeNode)e.nextElement();
            if(node.getUserObject().equals(obj))
                return node;
        }
        return null;
    }

    public DefaultMutableTreeNode addClass(Class c){
        // Bỏ qua những kiểu không phải là lớp
        if(c.isInterface() || c.isPrimitive()) return null;
        // Tìm đối tượng
        DefaultMutableTreeNode node = findUserObject(c);
        if(node != null) return node;
        // Nếu lớp đó không có mặt thì bổ sung lớp cha của nó và lặp lại
    }
}
```

```

        Class s = c.getSuperclass();
        DefaultMutableTreeNode parent;
        if(s == null)
            parent = root;
        else
            parent = addClass(s);
        // Bổ sung lớp mới như là lớp con
        DefaultMutableTreeNode newNode = new DefaultMutableTreeNode(c);
        model.insertNodeInto(newNode, parent, parent getChildCount());
        // Cho phép hiển thị nút
        TreePath path = new TreePath(model.getPathToRoot(newNode));
        tree.makeVisible(path);
        return newNode;
    }

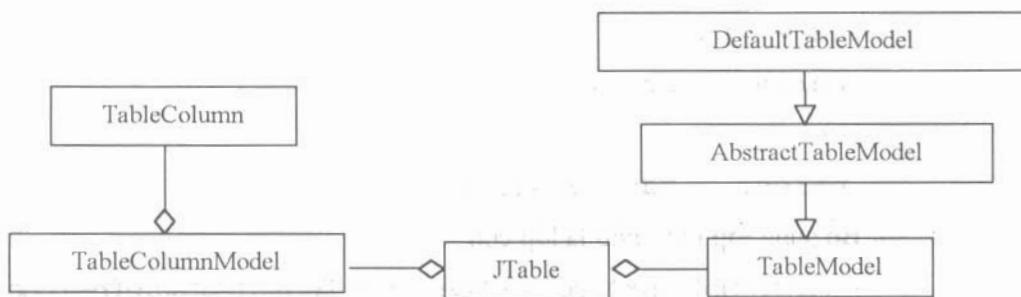
    public static String getFieldDescription(Class c) {
        // Tìm kiểu và tên của các trường
        String r = "";
        Field[] fields = c.getDeclaredFields();
        for(int i = 0; i < fields.length; i++) {
            Field f = fields[i];
            if((f.getModifiers() & Modifier.STATIC) != 0)
                r += "static ";
            r += f.getType().getName() + " ";
            r += f.getName() + "\n";
        }
        return r;
    }
}

```

## 4.2. CÁC BẢNG DỮ LIỆU

Bên cạnh cấu trúc cây như đã nêu trên, cấu trúc bảng cho phép hiển thị các mục dữ liệu thành bảng hai chiều cũng rất phổ biến. Nhóm xây dựng Swing đã có nhiều cố gắng để tạo ra các điều khiển bảng giúp người lập trình sử dụng chúng dễ dàng hơn.

Mỗi quan hệ giữa các lớp được thiết kế để tổ chức bảng dữ liệu được mô tả như hình dưới đây.



Hình 4.8. Mối quan hệ giữa các lớp để tạo lập bảng Table

#### 4.2.1. Bảng đơn giản

Cũng giống như đối với cấu trúc cây, JTable không chỉ lưu trữ dữ liệu của riêng nó mà còn phải xác định *mô hình bảng* cho những dữ liệu đó. Mô hình bảng ta sẽ đề cập ở phần sau.

**Ví dụ 4.5.** Trước tiên, ta xây dựng một bảng kiểu mẫu bao gồm các đặc tính của các hành tinh trong Thái Dương hệ. Một hành tinh là thể khí nếu nó chứa phần lớn chất hydrogen (hyđro) và helium (helium). Bảng mô tả các hành tinh có các thuộc tính là các cột bao gồm: tên hành tinh (Planet), bán kính (Radius) theo đơn vị km, số vệ tinh (Moons), có là thể khí (Gaseous) hay không và màu (Color). Bảng ở hình 4.9 bao gồm các hành tinh: Mercury (Sao Thuỷ), Venus (Sao Kim), Earth (Trái Đất), Mars (Sao Hỏa), Jupiter (Sao Mộc), Sarturn (Sao Thổ), Uranus (Sao Thiên Vương), Neptune (Sao Hải Vương) và Pluto (Sao Diêm Vương).

Planet	Radius	Moons	Gaseous	Color
Mercury	2440.0	0	false	java.awt.Color[r=255,g=255,b=0]
Venus	6052.0	0	false	java.awt.Color[r=255,g=255,b=0]
Earth	6378.0	1	false	java.awt.Color[r=0,g=0,b=255]
Mars	3397.0	2	false	java.awt.Color[r=255,g=0,b=0]
Jupiter	71492.0	16	true	java.awt.Color[r=255,g=200,b=0]
Saturn	60268.0	18	true	java.awt.Color[r=255,g=200,b=0]
Uranus	25558.0	17	true	java.awt.Color[r=0,g=0,b=255]
Neptune	24766.0	8	true	java.awt.Color[r=0,g=0,b=255]

Hình 4.9. Một bảng mô tả các hành tinh trong Thái Dương hệ

Thông thường dữ liệu của bảng được lưu dưới dạng một mảng hai chiều, ví dụ

```

Object[][] cells = {
    {"Mercury", new Double(2440), new Integer(0), Boolean.FALSE,
     Color.yellow},
    ...
};
  
```

Bảng này gọi phương thức `toString()` tới từng đối tượng để hiển thị chúng ở các cột. Vì thế, ta dễ hiểu cột màu sẽ hiển thị dạng `java.awt.Color[r = ..., g = ..., b = ...]`.

Tên của các cột có thể tổ chức thành mảng một chiều, ví dụ

```
String[] columnNames = {
    "Planet", "Radius", "Moons", "Gaseous", "Color"
};
```

Cuối cùng là tạo lập một bảng có dữ liệu tương ứng với mảng hai chiều `cells` và các cột có tên `columnNames`:

```
JTable table = new JTable(cells, columnNames);
```

Lệnh trên tạo ra bảng có số hàng, số cột bằng số hàng, số cột tương ứng của mảng `cells` với độ rộng của từng hàng mặc định là bằng nhau. Tuy nhiên, sau đó ta có thể thay đổi vị trí, độ rộng của mỗi cột sao cho phù hợp nhất.

```
//PlanetTable.java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;

public class PlanetTable{
    public static void main(String args[]){
        JFrame fr = new PlanetTableFrame();
        fr.show();
    }
}

class PlanetTableFrame extends JFrame
{
    public PlanetTableFrame(){
        setTitle("Planet Table");
        setSize(300, 200);
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
    }
}
```

```

// Tạo lập một bảng
JTable table = new JTable(cells, columnNames);

Container content = getContentPane();
content.add(new JScrollPane(table), "Center");
}

// Định nghĩa mảng dữ liệu để đưa vào bảng
private Object[][] cells = {
    {"Mercury", new Double(2440), new Integer(0),
     Boolean.FALSE, Color.yellow},
    {"Venus", new Double(6052), new Integer(0),
     Boolean.FALSE, Color.yellow},
    {"Earth", new Double(6378), new Integer(1), Boolean.FALSE,
     Color.blue},
    {"Mars", new Double(3397), new Integer(2), Boolean.FALSE,
     Color.red},
    {"Jupiter", new Double(71492), new Integer(16),
     Boolean.TRUE, Color.orange},
    {"Saturn", new Double(60268), new Integer(18),
     Boolean.TRUE, Color.orange},
    {"Uranus", new Double(25558), new Integer(17),
     Boolean.TRUE, Color.blue},
    {"Neptune", new Double(24766), new Integer(8),
     Boolean.TRUE, Color.blue},
    {"Pluto", new Double(1137), new Integer(1), Boolean.FALSE,
     Color.black}
};

// Định nghĩa dãy tên các cột của bảng
private String[] columnNames = {
    "Planet", "Radius", "Moons", "Gaseous", "Color"
};

```

**Lưu ý:** Trên bảng ta có thể thực hiện một số chức năng:

- Thay đổi kích cỡ của các cột, ta chỉ cần đặt con trỏ (cursor) giữa hai cột và di chuột để thay đổi độ rộng của từng cột cho phù hợp như hình 4.8.
- Thay đổi thứ tự của các cột bằng cách kích chuột vào tên của cột và di đến vị trí cần sắp xếp.

#### 4.2.2. Mô hình bảng TableModel

Ở trên ta đã xét trường hợp tạo lập một bảng đơn giản, trong đó dữ liệu được tổ chức thành mảng hai chiều. Tuy nhiên, trong nhiều chương trình ứng dụng, ta không thể sử dụng cách này để tổ chức dữ liệu được mà phải tạo ra một *mô hình bảng* riêng.

Trong Java có lớp trừu tượng AbstractTableModel, trong đó có các phương thức:

```
public int getRowCount();           // Xác định số hàng
public int getColumnCount();       // Xác định số cột
public Object getValueAt(int row, int column); // Xác định đối tượng
```

**Ví dụ 4.6.** Xây dựng bảng thể hiện được lợi nhuận đầu tư thu được hàng năm theo tỷ lệ %. Giá vốn ban đầu là \$100.000 và các tỷ lệ tăng trưởng hàng năm là 5%, 6%, 7%, 8%, 9% và 10%. Khi đó ta có bảng mô tả sự tăng trưởng của vốn đầu tư như hình 4.10 sau đây.

5%	6%	7%	8%	9%	10%
\$100,000	\$100,000	\$100,000	\$100,000	\$100,000	\$100,000
\$105,000	\$106,000	\$107,000	\$108,000	\$109,000	\$110,000
\$110,250	\$112,360	\$114,490	\$116,640	\$118,810	\$121,000
\$115,762	\$119,101	\$122,504	\$125,971	\$129,502	\$133,100
\$121,550	\$126,247	\$131,079	\$136,048	\$141,158	\$146,410
\$127,628	\$133,922	\$140,255	\$146,932	\$153,862	\$161,051
\$134,009	\$141,891	\$150,073	\$158,587	\$167,110	\$177,156
\$140,710	\$150,363	\$160,578	\$171,382	\$182,803	\$194,871
\$147,745	\$159,384	\$171,818	\$185,093	\$199,256	\$214,358

Hình 4.10. Bảng thể hiện tỷ lệ đầu tư theo mô hình bảng

Phương thức getValueAt() tính giá trị ở hàng r và cột c được viết như sau:

```
public Object getValueAt(int r, int c){
    double rate = (c + minRate) / 100.0;
    int nperiods = r;
    double futureBalance = INITIAL_BALANCE * Math.pow(1+rate, nperiods);
    return
        NumberFormat.getNumberInstance().format(futureBalance);
}
```

Tên các cột mặc định trong AbstractTableModel là A, B, C, v.v. Nếu muốn thay đổi tên gọi của chúng thì ta viết đè phương thức getColumnNames(), ví dụ

```
public String getColumnNames(int c){
    double rate = (c + minRate) / 100.0;
    return NumberFormat.getNumberInstance().format(rate);
}
```

Chương trình xây dựng bảng thể hiện sự tăng trưởng vốn đầu tư được viết như sau.

```
// InvestmentTable.java

import java.awt.*;
import java.awt.event.*;
import java.text.*;
import javax.swing.*;
import javax.swing.table.*;

public class InvestmentTable{
    public static void main(String args[]){
        JFrame fr = new InvestmentTableFrame();
        fr.show();
    }
}

class InvestmentTableFrame extends JFrame
{
    public InvestmentTableFrame(){
        setTitle("Investment Table");
        setSize(300, 200);
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
        // Mô hình dữ liệu tính các ô của bảng mỗi khi yêu cầu
        TableModel model = new InvestmentTableModel(30, 5, 10);
        JTable table = new JTable(model);
        Container content = getContentPane();
        content.add(new JScrollPane(table), "Center");
    }
}
```

```
class InvestmentTableModel extends AbstractTableModel{  
    private int years;  
    private int minRate;  
    private int maxRate;  
    private static double INITIAL_BALANCE = 100000.0;  
  
    public InvestmentTableModel(int y, int r1, int r2){  
        years = y;  
        minRate = r1;  
        maxRate = r2;  
    }  
    public int getRowCount(){  
        return years;  
    }  
  
    public int getColumnCount(){  
        return maxRate - minRate + 1;  
    }  
  
    public Object getValueAt(int r, int c){  
        double rate = (c + minRate) / 100.0;  
        int nperiods = r;  
        double futureBalance = INITIAL_BALANCE *  
            Math.pow(1+rate, nperiods);  
        return  
            NumberFormat.getCurrencyInstance().format(futureBalance);  
    }  
  
    public String getColumnName(int c){  
        double rate = (c + minRate) / 100.0;  
        return NumberFormat.getPercentInstance().format(rate);  
    }  
}
```

#### 4.2.3. Hiển thị các record dữ liệu từ cơ sở dữ liệu

Hầu hết các thông tin cần tổ chức dưới dạng bảng đều được lấy từ các cơ sở dữ liệu quan hệ. Nếu ta phát triển những chương trình ứng dụng chuyên nghiệp thì có thể sử dụng Java Bean (được đề cập ở chương sau) để truy cập vào CSDL để lấy thông tin dưới dạng các record.

**Ví dụ 4.7.** Đơn giản hơn, ta có thể sử dụng cơ chế kết nối JDBC [1] để truy vấn vào một CSDL và hiển thị kết quả dưới dạng các bảng.

Trước tiên ta định nghĩa ResultSetTableModel để tổ chức dữ liệu kết quả thu được từ truy vấn vào một CSDL.

Ta có thể biết được số cột, tên các cột từ đối tượng siêu dữ liệu kết quả:

```
public int getColumnCount() {
    try{
        return rsmd.getColumnCount();
    }catch(SQLException e){
        System.out.println("Error: " + e);
        return 0;
    }
}

public String getColumnName(int c) {
    try{
        return rsmd.getColumnName(c + 1);
    }catch(SQLException e){
        System.out.println("Error: " + e);
        return "";
    }
}
```

Chương trình truy vấn vào một CSDL và hiển thị kết quả dưới dạng bảng được viết như sau.

```
// ResultSetTable.java
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.util.*;
import javax.swing.*;
```

```
import javax.swing.table.*;
public class ResultSetTable{
    public static void main(String args[]){
        JFrame fr = new ResultSetTableModel();
        fr.show();
    }
}

class ResultSetTableModel extends JFrame implements
    ActionListener {
    private JScrollPane scrollPane;
    private ResultSetTableModel model;
    private JComboBox tableNames;
    private JButton nextButton;
    private JButton previousButton;
    private ResultSet rs;
    private Connection con;
    private Statement stmt;
    private static boolean SCROLLABLE = false;

    public ResultSetTableModel(){
        setTitle("Result Set Table");
        setSize(300, 200);
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
        // Tìm tất cả các bảng trong CSDL và đưa vào hộp Combo Box
        Container content = getContentPane();
        tableNames = new JComboBox();
        tableNames.addActionListener(this);
        JPanel p = new JPanel();
        p.add(tableNames);
```

```
content.add(p, "North");

try{
    Class.forName("com.pointbase.jdbc.JdbcDriver");
    // Nạp Driver
    String url = "jdbc:pointbase:corejava";
    String user = "PUBLIC";
    String pw = "PUBLIC";
    con = DriverManager.getConnection(url, user, pw);
    if(SCROLLABLE)
        stmt = con.createStatement(
            ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_READ_ONLY);
    else
        stmt = con.createStatement();
    DatabaseMetaData md = con.getMetaData();
    ResultSet mrs = md.getTables(null, null, null, new
        String[]{"TABLE"});
} ;
while(mrs.next())
    tableNames.addItem(mrs.getString(3));
mrs.close();
} catch(ClassNotFoundException e){
    System.out.println("Error: " + e);
}
catch(SQLException e){
    System.out.println("Error: " + e);
}

}

public void actionPerformed(ActionEvent evt){
    if(evt.getSource() == tableNames){
        // Hiển thị bảng đã được chọn trong bảng Combo Box
        if(scrollPane != null)
            getContentPane().remove(scrollPane);
```

```
try{
    String tableName
        = (String)tableNames.getSelectedItem();
    if(rs != null) rs.close();
    String query = "SELECT * FROM " + tableName;
    if(SCROLLABLE)
        model = new ScrollResultSetTableModel(rs);
    else
        model = new CachingResultSetTableModel(rs);
    JTable table = new JTable(model);
    scrollPane = new JScrollPane(table);
    getContentPane().add(scrollPane, "Center");
    pack();
    doLayout();
}catch(SQLException e){
    System.out.println("Error: " + e);
}
}

}

// Lớp cơ sở để điều chỉnh việc cuộn dữ liệu và cất giữ mô hình bảng của tập kết quả
abstract class ResultSetTableModel extends
    AbstractTableModel{
private ResultSet rs;
private ResultSetMetaData rsmd;

public ResultSetTableModel(ResultSet aRes){
    rs = aRes;
    try{
        rsmd = rs.getMetaData();
    }catch(SQLException e){
        System.out.println("Error: " + e);
    }
}
}
```

```
public int getColumnCount(){
    try{
        return rsmd.getColumnCount();
    }catch(SQLException e){
        System.out.println("Error: " + e);
        return 0;
    }
}

public ResultSet getResultSet(){
    return rs;
}

public String getColumnName(int c){
    try{
        return rsmd.getColumnName(c + 1);
    }catch(SQLException e){
        System.out.println("Error: " + e);
        return "";
    }
}

// Lớp sử dụng con trỏ cuộn dữ liệu có ở JDBC 2
class ScrollResultSetTableModel extends
    ResultSetTableModel{
    public ScrollResultSetTableModel(ResultSet aRes){
        super(aRes);
    }
    public Object getValueAt(int r, int c){
        try{
            ResultSet rs = getResultSet();
            rs.absolute(r + 1);
            return rs.getObject(c + 1);
        }catch(SQLException e){
            System.out.println("Error: " + e);
        }
    }
}
```

```
        return null;
    }
}

public int getRowCount(){
    try{
        ResultSet rs = getResultSet();
        rs.last();
        return rs.getRow();
    }catch(SQLException e){
        System.out.println("Error: " + e);
        return 0;
    }
}

// Lớp cắt giữ dữ liệu kết quả
class CachingResultSetTableModel extends
    ResultSetTableModel{
    public CachingResultSetTableModel(ResultSet aRes){
        super(aRes);
        try{
            cache = new ArrayList();
            int cols = getColumnCount();
            ResultSet rs = getResultSet();
            // Tìm các bản ghi tiếp theo
            while(rs.next()){
                Object[] row = new Object[cols];
                for(int j = 0; j < row.length; j++)
                    row[j] = rs.getObject(j + 1);
                cache.add(row);
            }
        }catch(SQLException e){
            System.out.println("Error: " + e);
        }
    }
}
```

```

    }

    public Object getValueAt(int r, int c){
        if(r < cache.size())
            return ((Object[])cache.get(r))[c];
        else
            return null;
    }

    public int getRowCount(){
        return cache.size();
    }

    private ArrayList cache;
}

```

#### 4.2.4. Sắp xếp các hàng trong bảng

Bảng dữ liệu thường được yêu cầu tổ chức dưới nhiều dạng khác nhau. Để tiện theo dõi và tìm kiếm, bảng kết quả cần được sắp xếp theo thứ tự tăng hoặc giảm theo các giá trị của một cột nào đó.

**Ví dụ 4.8.** Ta hãy tổ chức lại bảng các hành tinh ở ví dụ 4.5 sao cho khi nhấn đúp vào tên của một cột nào đó thì bảng sẽ được sắp xếp lại theo thứ tự các giá trị của cột đó. Ví dụ, nhấn đúp vào cột Moons thì bảng được sắp xếp theo thứ tự tăng dần của số lượng vệ tinh của các hành tinh như hình 4.11.

Planet	Radius	Moons	Gaseous	Color
Mercury	2440.0	0	false	java.awt.Color[r=255,g=
Venus	6052.0	0	false	java.awt.Color[r=255,g=
Pluto	1137.0	1	false	java.awt.Color[r=0,g=0,
Earth	6378.0	1	false	java.awt.Color[r=0,g=0,
Mars	3397.0	2	false	java.awt.Color[r=255,g=
Neptune	24768.0	8	true	java.awt.Color[r=0,g=0,
Jupiter	71492.0	16	true	java.awt.Color[r=255,g=
Uranus	25558.0	17	true	java.awt.Color[r=0,g=0,
Saturn	60268.0	18	true	java.awt.Color[r=255,g=

Hình 4.11. Bảng các hành tinh được sắp xếp khi nhấn đúp vào cột Moons

Để sắp xếp được các hàng ta sử dụng *mô hình bộ lọc*, nó lưu trữ các tham chiếu tới mô hình bảng hiện thời. Khi JTable cần tìm một giá trị, mô hình bộ lọc sẽ tính chỉ số của cột hiện thời (cột được nhấn đúp) và nhận lại giá trị từ mô hình bảng.

```

public Object getValueAt(int r, int c){
    return model.getValueAt(rows[r].index, c);
}

```

Mô hình bộ lọc làm nhiệm vụ sắp xếp giữa JTable và TableModel như sau.



**Hình 4.12. Bộ lọc sắp xếp cho bảng**

Ta thực hiện sắp xếp các đối tượng của Row. Mỗi đối tượng của Row có chứa chỉ số index của cột trong mô hình bảng. So sánh hai đối tượng đó như sau: tìm các phần tử trong mô hình và so sánh chúng.

```
model.getValueAt(r1, c).compareTo(model.getValueAt(r2, c))
```

Trong đó, r1, r2 là hai chỉ số của cột trong Row và c là cột mà các phần tử của nó cần sắp xếp.

Lớp Row được xây dựng riêng ở trong lớp SortFilterModel vì phương thức compareTo() chỉ cần truy cập đến mô hình và cột hiện thời trong mô hình sắp xếp.

```

class SortFilterModel extends AbstractTableModel{
    //...
    public SortFilterModel(TableModel m) {
        model = m;
        rows = new Row[model.getRowCount()];
        for(int i = 0; i < rows.length; i++){
            rows[i] = new Row();
            rows[i].index = i;
        }
    }
    public void sort(int c){
        sortColumn = c;
        Arrays.sort(rows);
        fireTableDataChanged();
    }
    //...
    private class Row implements Comparable{
        public int index;
        public int compareTo(Object other) {
            Row otherRow = (Row)other;

```

```

        Object a = model.getValueAt(index, sortColumn);
        Object b = model.getValueAt(otherRow.index, sortColumn);

        if(a instanceof Comparable)
            return ((Comparable)a).compareTo(b);
        else
            return index - otherRow.index;
    }
}
}

```

Cuối cùng, chương trình hiển thị bảng các hành tinh, trong đó cho phép sắp xếp theo các cột khi được nhấn đúp, được viết như sau.

```

// TableSort.java

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.table.*;

public class TableSort{
    public static void main(String args[]){
        JFrame fr = new TableSortFrame();
        fr.show();
    }
}

class SortFilterModel extends AbstractTableModel{

    private TableModel model;
    private int sortColumn;
    private Row[] rows;

    public SortFilterModel(TableModel m){
        model = m;
        rows = new Row[model.getRowCount()];
    }
}

```

```
        for(int i = 0; i < rows.length; i++) {
            rows[i] = new Row();
            rows[i].index = i;
        }
    }

    public void sort(int c) {
        sortColumn = c;
        Arrays.sort(rows);
        fireTableDataChanged();
    }

    public void addMouseListener(final JTable table) {
        table.getTableHeader().addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent evt) {
                // Kiểm tra xem có kích chuột đúp hay không
                if(evt.getClickCount() < 2) return;
                // Nếu có cột được kích đúp
                int tableColumn = table.columnAtPoint(evt.getPoint());
                // Chuyển vào chỉ số của mô hình và sắp xếp
                int modelColumn = table.convertColumnIndexToModel(tableColumn);
                sort(modelColumn);
            }
        });
    }
}

public Object getValueAt(int r, int c) {
    return model.getValueAt(rows[r].index, c);
}

public boolean isCellEditable(int r, int c) {
    return model.isCellEditable(rows[r].index, c);
}

public void setValueAt(Object aValue, int r, int c) {
```

```
model.setValueAt aValue, rows[r].index, c);  
}  
  
// Định nghĩa các phương thức còn lại của mô hình  
public int getRowCount(){  
    return model.getRowCount();  
}  
  
public int getColumnCount(){  
    return model.getColumnCount();  
}  
  
public String getColumnName(int c){  
    return model.getColumnName(c);  
}  
  
public Class getColumnClass(int c){  
    return model.getColumnClass(c);  
}  
  
// Lớp bên trong lưu giữ các chỉ số của các hàng.  
private class Row implements Comparable{  
    public int index;  
    public int compareTo(Object other){  
        Row otherRow = (Row)other;  
        Object a = model.getValueAt(index, sortColumn);  
        Object b = model.getValueAt(otherRow.index, sortColumn);  
  
        if(a instanceof Comparable)  
            return ((Comparable)a).compareTo(b);  
        else  
            return index - otherRow.index;  
    }  
}  
  
class TableSortFrame extends JFrame  
{
```

```
public TableSortFrame(){
    setTitle("Table Sorting");
    setSize(300, 200);
    addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e){
            System.exit(0);
        }
    });
    // Xây dựng mô hình bảng và bộ sắp xếp.
    DefaultTableModel model
        = new DefaultTableModel(cells, columnNames);
    SortFilterModel sorter = new SortFilterModel(model);
    // Hiển thị bảng
    JTable table = new JTable(sorter);
    Container content = getContentPane();
    content.add(new JScrollPane(table), "Center");
    // Đặt bảng vào bộ xử lý kích đúp chuột
    sorter.addMouseListener(table);
}

// Mảng dữ liệu về các hành tinh
private Object[][] cells = {
    {"Mercury", new Double(2440), new Integer(0),
     Boolean.FALSE, Color.yellow},
    {"Venus", new Double(6052), new Integer(0),
     Boolean.FALSE, Color.yellow},
    {"Earth", new Double(6378), new Integer(1),
     Boolean.FALSE, Color.blue},
    {"Mars", new Double(3397), new Integer(2),
     Boolean.FALSE, Color.red},
    {"Jupiter", new Double(71492), new Integer(16),
     Boolean.TRUE, Color.orange},
    {"Saturn", new Double(60268), new Integer(18),
     Boolean.TRUE, Color.orange},
    {"Uranus", new Double(25558), new Integer(17),
     Boolean.TRUE, Color.orange}
}
```

```

        Boolean.TRUE, Color.blue},
        {"Neptune", new Double(24766), new Integer(8),
        Boolean.TRUE, Color.blue},
        {"Pluto", new Double(1137), new Integer(1),
        Boolean.FALSE, Color.black}
    };
    // Tên của các cột
    private String[] columnNames = {
        "Planet", "Radius", "Moons", "Gaseous", "Color"
    };
}

```

javax.swing.table.TableModel

API

- int getRowCount(): Cho biết số cột trong mô hình bảng
- int getColumnCount(int row, int column): Cho biết số hàng
- Object getValueAt(int row, int column): Cho biết đối tượng ở ô tương ứng
- void setValueAt(Object newValue, int row, int column): Đặt lại giá trị newValue vào ô tương ứng.
- boolean isCellEditable(int row, int column): Kết quả là true nếu ô tương ứng là thay đổi được.
- String getColumnName(int column): Đọc tiêu đề (tên) của cột.
- Class getColumnClass(int columnIndex): Xác định lớp chứa giá trị ở cột columnIndex.

javax.swing.table.JTable

API

- JTableHeader getTableHeader(): Đọc tiêu đề của bảng
- int columnAtPoint(Point p): Cho số hiệu của cột ở toạ độ p.
- int convertColumnIndexToModel(int tableColumn): Cho lại chỉ số mô hình của cột sau khi được chuyển đổi. Giá trị kết quả khác với tableColumn nếu một số cột bị loại khỏi bảng hoặc bị che giấu.
- void setRowHeight(int height): Đặt chiều cao cho các hàng là height pixels.
- Rectangle getCellRect(int row, int column, boolean includeSpacing): Xác định khung chữ nhật bao quanh ô tương ứng. Nếu includeSpacing thì kể cả các dấu cách.

- `Color getSelectionBackground()`: Xác định màu nền
- `Color getSelectionForeground()`: Xác định màu vẽ
- `int convertColumnIndexToModel (int TableColumn)`:

#### 4.2.5. Soạn thảo bảng

Khi tổ chức dữ liệu dưới dạng bảng, nhiều khi ta muốn hiển thị dữ liệu ở các cột với nhiều kiểu khác nhau để có nhiều thông tin hơn. Khi ta định nghĩa phương thức

```
Class getColumnClass(int columnIndex)
```

trong mô hình bảng để biết được *kiểu của cột*, thì `JTable` sẽ cung cấp các kiểu

<code>ImageIcon</code>	Kiểu hình ảnh
<code>Boolean</code>	Hộp kiểm tra (Check box)
<code>Object</code>	Xâu ký tự (String)

Các ô trong bảng có thể thay đổi màu, trạng thái và thực hiện cập nhật giống như các nút trên cây mà ta đã tìm hiểu ở phần trước. Giao diện `TableCellRenderer` có phương thức

```
public Component getTableCellRendererComponent(JTable table,
                                              Object value, boolean isSelected, boolean hasFocus, int row, int column)
```

được cài đặt để cho phép thay đổi màu nền, màu vẽ, các tham số của từng ô trong bảng.

Để soạn thảo (thay đổi tham số của ô), mô hình bảng phải chỉ ra những cột, ô nào được phép thực hiện thông qua phương thức `isCellEditable()`. Ví dụ

```
public boolean isCellEditable(int r, int c) {
    return c == NAME_COLUMN || c == MOON_COLUMN ||
           c == COLOR_COLUMN || c == GASEOUS_COLUMN;
}
```

cho phép thay đổi 4 cột tương ứng với các hằng số trên.

**Ví dụ 4.9.** Hãy tổ chức một bảng mô tả các hành tinh với các dạng biểu diễn thông tin ở các cột: Planet, Moons, Gaseous, và Color là có thể thay đổi, cập nhật được. Khi chạy chương trình, ta có thể kích vào hộp Combo Box để thay đổi trạng thái được đánh dấu (tích vào hộp Combo Box) hoặc không tương ứng với hành tinh là thế kia hay không. Tương tự, ta có thể kích đúp chuột vào ô ở cột (Moons), cột tên gọi (Planet) và thay đổi được số vệ tinh, tên gọi của hành tinh tương ứng, nhưng không thay đổi được bán kính của nó. Nghĩa là ở những ô đó ta có thể cập nhật được các giá trị, thay đổi được các trạng thái, v.v. như hình 4.13.

Planet	Radius	Moons	Gaseous	Color
Earth	6,378	1	0	Black
Mars	3,397	2	0	Dark Gray
Jupiter	71,492	16	1	Light Gray

Hình 4.13. Bảng với một số cột có thể cập nhật, thay đổi được

```
// TableCellRender.java
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.table.*;

public class TableCellRender{
    public static void main(String args[]){
        JFrame fr = new TableCellRenderFrame();
        fr.show();
    }
}
/*

```

Mô hình bảng cho phép xác định các đặc tính của từng ô, từng cột trong bảng. Trong đó cho phép thay đổi cột tên gọi (NAME\_COLUMN), số vệ tinh (MOON\_COLUMN), thê khí (GASEOUS\_COLUMN) và cột màu (COLOR\_COLUMN).

```
*
class PlanetTableModel extends AbstractTableModel{

    public static final int NAME_COLUMN = 0;
    public static final int MOON_COLUMN = 2;
    public static final int GASEOUS_COLUMN = 3;
    public static final int COLOR_COLUMN = 4;

    public Class getColumnClass(int c){
        return cells[0][c].getClass();
    }
}
```

```
}

public String getColumnNome(int c){
    return columnNames[c];
}

public int getColumnCount(){
    return cells[0].length;
}

public Object getValueAt(int r, int c){
    return cells[r][c];
}

public void setValueAt(Object aValue, int r, int c){
    cells[r][c] = aValue;
}

public int getRowCount(){
    return cells.length;
}

// Được phép thay đổi giá trị trong các cột tương ứng
public boolean isCellEditable(int r, int c){
    return c == NAME_COLUMN || c == MOON_COLUMN ||
           c == COLOR_COLUMN || c == GASEOUS_COLUMN;
}

private Object[][] cells = {
    {"Mercury", new Double(2440), new Integer(0),
     Boolean.FALSE, Color.yellow},
    {"Venus", new Double(6052), new Integer(0),
     Boolean.FALSE, Color.yellow},
    {"Earth", new Double(6378), new Integer(1),
     Boolean.FALSE, Color.blue},
    {"Mars", new Double(3397), new Integer(2),
     Boolean.FALSE, Color.red},
    {"Jupiter", new Double(71492), new Integer(16),
     Boolean.TRUE, Color.orange},
```

```
        {"Saturn", new Double(60268), new Integer(18),
         Boolean.TRUE, Color.orange},
        {"Uranus", new Double(25558), new Integer(17),
         Boolean.TRUE, Color.blue},
        {"Neptune", new Double(24766), new Integer(8),
         Boolean.TRUE, Color.blue},
        {"Pluto", new Double(1137), new Integer(1),
         Boolean.FALSE, Color.black}
    };
    private String[] columnNames = {
        "Planet", "Radius", "Moons", "Gaseous", "Color"
    };
}

class TableCellRenderFrame extends JFrame
{
    public TableCellRenderFrame(){
        setTitle("Table Cell Rendering");
        setSize(300, 200);
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
        TableModel model = new PlanetTableModel();
        JTable table = new JTable(model);
        // Thay đổi màu và cho phép soạn thảo, thay đổi các giá trị trong các ô được phép
        table.setDefaultRenderer(Color.class, new ColorTableCellRenderer());
        table.setDefaultEditor(Color.class, new ColorTableCellEditor());

        JComboBox moonCombo = new JComboBox();
        for(int i = 0; i <= 20; i++)
            moonCombo.addItem(new Integer(i));
        TableColumnModel columnModel = table.getColumnModel();
```

```
    TableColumn moonColumn
        = columnModel.getColumn(PlanetTableModel.MOON_COLUMN);
    // Hiển thị bảng sau khi thay đổi
    table.setRowHeight(100);
    Container content = getContentPane();
    content.add(new JScrollPane(table), "Center");
}

}

// Lớp hỗ trợ thay đổi màu
class ColorTableCellRenderer implements TableCellRenderer{
    private JPanel panel = new JPanel();
    public Component getTableCellRendererComponent(JTable table,
        Object value, boolean isSelected, boolean hasFocus, int row, int column)
    {
        panel.setBackground((Color)value);
        return panel;
    }
}

class ColorTableCellEditor extends ColorTableCellRenderer
    implements TableCellEditor{
    private Color color;
    private JColorChooser colorChooser;
    private JDialog colorDialog;
    private EventListenerList listenerList = new EventListenerList();
    private ChangeEvent event = new ChangeEvent(this);

    ColorTableCellEditor(){
        // Chuẩn bị hội thoại về màu
        colorChooser = new JColorChooser();
        colorDialog = JColorChooser.createDialog(null,
            "Planet Color", false, colorChooser, new ActionListener(){
                public void actionPerformed(ActionEvent event){
                    fireEditingStopped();
                }
            });
    }
}
```

```
        },
        new ActionListener(){
            public void actionPerformed(ActionEvent event){
                fireEditingCanceled();
            }
        });
    }

    public Component getTableCellEditorComponent(JTable table,
                                                Object value, boolean isSelected, int row, int column){
        // Ở đây ta nhận được màu để thay đổi
        colorChooser.setColor((Color)value);
        return getTableCellRendererComponent(table,value,
                                             isSelected, true, row, column);
    }

    public boolean isCellEditable(EventObject anEvent){
        return true;
    }

    // Bắt đầu soạn thảo
    public boolean shouldSelectCell(EventObject anEvent){
        colorDialog.setVisible(true);
        return true;
    }

    public void cancelCellEditing(){
        // Huỷ bỏ sự thay đổi
        colorDialog.setVisible(false);
    }

    public boolean stopCellEditing(){
        // Kết thúc việc soạn thảo
        colorDialog.setVisible(false);
        return true;
    }
}
```

```

        public Object getCellEditorValue(){
            return colorChooser.getColor();
        }

        public void addCellEditorListener(CellEditorListener l){
            listenerList.add(CellEditorListener.class, l);
        }

        public void removeCellEditorListener(CellEditorListener l){
            listenerList.remove(CellEditorListener.class, l);
        }

        protected void fireEditingStopped(){
            Object[] listeners = listenerList.getListenerList();
            for(int i = listeners.length; i >= 0; i -= 2)
                ((CellEditorListener)listeners[i + 1]).editingStopped(event);
        }

        protected void fireEditingCanceled(){
            Object[] listeners = listenerList.getListenerList();
            for(int i = listeners.length; i >= 0; i -= 2)
                ((CellEditorListener)listeners[i + 1]).editingCanceled(event);
        }
    }
}

```

**Lưu ý:** Ngoài những khả năng đã đề cập ở trên, ta còn có thể thao tác trên các hàng, các cột của bảng. Ta có thể đặt lại kích cỡ của các cột, bổ sung hay loại bỏ một số hàng, lựa chọn một số hàng, cột, một số ô và che giấu hay hiển thị chúng, v.v. (chi tiết hơn có thể xem [2]).

### 4.3. CÁC THANH TRƯỢT VÀ CÁC THƯỚC ĐO TIẾN ĐỘ

Trong phần này ta tìm hiểu về hai công cụ điều khiển tương đối phổ dụng của Swing: thanh trượt cho phép ta xác định đầu vào theo một đại lượng trong một phạm vi tuyến tính và thước đo tiến độ cho phép chương trình đưa tỉ lệ % hoàn thành một công việc.

#### 4.3.1. Thanh trượt Slider

Tranh trượt Slider hay còn gọi là thước đo, gần giống như thanh cuộn ScrollBar. Tuy nhiên, ScrollBar được sử dụng để di chuyển công nhìn các vùng dữ liệu, còn

Slider được sử dụng để xác định giá trị của dữ liệu trong một lỳ lê cho trước, ví dụ như thước kẽ của học sinh. ScrollBar cài đặt giao diện Adjustable, còn Slider không cài đặt giao diện đó và có thể xuất hiện như một giao diện sử dụng chung cho cả hai.

Cách chung nhất để tạo ra một Slider được thực hiện như sau:

```
JSlider slider = new JSlider(min, max, initialValue);
```

Tạo ra một thước đo nằm ngang có giá trị cực tiểu là min, cực đại là max và đầu đọc ở initialValue. Các giá trị của đầu đọc được dịch chuyển trong khoảng từ min tới max. Khi giá trị ở đầu đọc thay đổi, một sự kiện ChangeEvent được gửi tới tất cả các đối tượng đang lắng nghe để xử lý cho phù hợp với sự thay đổi đó.

Trường hợp sử dụng toán tử tạo lập Slider() mặc định

```
JSlider slider = new JSlider()
```

sẽ tạo ra một thước đo có giá trị cực tiểu là 0 và cực đại là 100, đầu đọc ở giữa (50).

Nếu muốn tạo ra thước đo nằm dọc, ta sử dụng

```
JSlider slider = new JSlider(SwingConstants.VERTICAL, min,
                               max, initialValue);
```

Để tạo được những thanh trượt có các vạch đo, ví dụ các vạch nhỏ cách nhau 5 đơn vị, các vạch lớn cách nhau 20, ta phải đưa thêm các ràng buộc:

```
slider.setMajorTickSpacing(20);
```

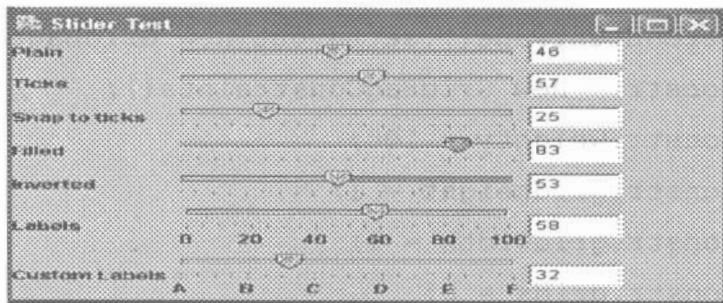
```
slider.setMinorTickSpacing(5);
```

Lưu ý, các vạch trên thước đo là các đơn vị đo, không phải là pixel.

**Ví dụ 4.10.** Hãy tạo ra các loại thước đo:

- + Thước đo đơn giản Plain: không có thang độ đo,
- + Thước đo Ticks: có thang độ đo và cho phép đo ở mọi cấp độ,
- + Thước đo Snap to ticks: đầu đọc luôn bám theo độ đo đã được vạch sẵn trên thước (làm tròn theo mép vạch gần nhất),
- + Thước đo Filled: độ đo được xác định bởi đầu đọc được tô màu,
- + Thước đo đảo ngược Inverted: độ đo còn lại được tô màu,
- + Thước đo có đánh số Labels: ở những độ đo chỉ định như 0, 20, 40, 60, 80, 100 sẽ được đánh số tương ứng,
- + Thước đo có đánh số Custom labels: ở những độ đo chỉ định như 0, 20, 40, 60, 80, 100 sẽ được đánh nhãn tùy chỉnh, ví dụ là 'A', 'B', 'C', 'D', 'E', 'F'.

Khi các đầu đọc di chuyển thì giá trị tương ứng của mỗi đầu đọc được hiển thị ở ô bên phải.



Hình 4.14. Các loại thước đo và thanh trượt

```
// SliderTest.java
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;

public class SliderTest{
    public static void main(String args[]){
        JFrame fr = new SliderTestFrame();
        fr.show();
    }
}

// Tạo ra khung chương trình gồm các loại thước đo và thanh trượt khác nhau
class SliderTestFrame extends JFrame {
    private GridBagConstraints constr;
    public SliderTestFrame(){
        setTitle("Slider Test");
        setSize(400, 300);
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
    }
}

// Đặt layout cho khung chương trình là GridBagLayout và các ràng buộc
```

```
getContentPane().setLayout(new GridBagLayout());
constr = new GridBagConstraints();
constr.weighty = 100;
constr.gridheight = 1;
constr.gridx = 1;
constr.gridy = 0;
// Tạo ra thanh trượt đơn giản Plain, không có vạch đo
JSlider slider = new JSlider();
addSlider(slider, "Plain");
// Tạo ra thanh trượt Tick có vạch đo, và định vị được mọi độ đo từ 0 đến 100
slider = new JSlider();
slider.setPaintTicks(true);
slider.setMajorTickSpacing(20);
slider.setMinorTickSpacing(5);
addSlider(slider, "Ticks");
// Tạo ra thanh trượt Snap to ticks có vạch đo, nhưng đầu đọc luôn
// bám // theo vạch thước.
slider = new JSlider();
slider.setPaintTicks(true);
slider.setSnapToTicks(true);
slider.setMajorTickSpacing(20);
slider.setMinorTickSpacing(5);
addSlider(slider, "Snap to ticks");
// Tạo ra thước đo Filled, khoảng cách đo được tô màu
slider = new JSlider();
slider.setPaintTicks(true);
slider.setMajorTickSpacing(20);
slider.setMinorTickSpacing(5);
slider.putClientProperty("JSlider.isFilled", Boolean.TRUE);
addSlider(slider, "Filled");
// Tạo ra thước đo Inverted, ngược lại với thước đo trên, khoảng cách còn lại
// được tô màu
slider = new JSlider();
```

```
slider.setPaintTicks(true);
slider.setMajorTickSpacing(20);
slider.setMinorTickSpacing(5);
slider.putClientProperty("JSlider.isFilled", Boolean.TRUE);
slider.setInverted(true);
addSlider(slider, "Inverted");

// Tạo ra thuộc do Labels, thuộc được đánh số cách nhau 20 đơn vị
slider = new JSlider();
slider.setPaintTicks(true);
slider.setPaintLabels(true);
slider.setMajorTickSpacing(20);
slider.setMinorTickSpacing(5);
addSlider(slider, "Labels");

// Tạo ra thuộc do Labels, đánh nhau 'A', 'B', ..., 'F' cách nhau 20 đơn vị
slider = new JSlider();
slider.setPaintLabels(true);
slider.setPaintTicks(true);
slider.setMajorTickSpacing(20);
slider.setMinorTickSpacing(5);

Hashtable labelTable = new Hashtable();
labelTable.put(new Integer(0), new JLabel("A"));
labelTable.put(new Integer(20), new JLabel("B"));
labelTable.put(new Integer(40), new JLabel("C"));
labelTable.put(new Integer(60), new JLabel("D"));
labelTable.put(new Integer(80), new JLabel("E"));
labelTable.put(new Integer(100), new JLabel("F"));

slider.setLabelTable(labelTable);
addSlider(slider, "Custom Labels");

slider = new JSlider();
slider.setPaintTicks(true);
slider.setPaintLabels(true);
slider.setSnapToTicks(true);
```

```
    slider.setMajorTickSpacing(20);
    slider.setMinorTickSpacing(20);
}

public void addSlider(JSlider s, String descr) {
    // Tạo lập trường text để hiển thị số đo được xác định bởi đầu đọc thanh trượt
    final TextField textField = new TextField(4);
    // Cập nhật lại số đo khi đầu đọc thanh trượt di chuyển và xử lý các sự kiện
    s.addChangeListener(new ChangeListener() {
        public void stateChanged(ChangeEvent event) {
            JSlider source = (JSlider)event.getSource();
            textField.setText(" " + source.getValue());
        }
    });
    // Bổ sung 3 thành phần
    constr.gridx = 0;
    constr.anchor = GridBagConstraints.WEST;
    constr.fill = GridBagConstraints.NONE;
    constr.weightx = 0;
    // Bổ sung tên của thước đo
    getContentPane().add(new JLabel(descr), constr);
    // Bổ sung thước đo ở giữa
    constr.gridx++;
    constr.anchor = GridBagConstraints.CENTER;
    constr.fill = GridBagConstraints.HORIZONTAL;
    constr.weightx = 0;
    getContentPane().add(s, constr);
    // Bổ sung trường text hiển thị số đo
    constr.gridx++;
    constr.anchor = GridBagConstraints.WEST;
    constr.fill = GridBagConstraints.NONE;
    constr.weightx = 200;
    getContentPane().add(textField, constr);
    // Tăng tọa độ y cho thước đo tiếp theo
```

```

        constr.gridx++;
    }
}

```

javax.swing.JSlider

API

- JSlider()
  - JSlider(int direction)
  - JSlider(int min, int max)
  - JSlider(int min, int max, int init)
  - JSlider(int direction, int min, int max, int init)
- Tạo ra thước đo theo hướng direction và các giá trị min, max, init tương ứng. Trong đó
- + direction có thể là SwingConstraints.HORIZONTAL thước nǎm ngang hoặc SwingConstraints.VERTICAL để tạo ra thước nǎm dọc. Mặc định là nǎm ngang.
  - + min, max là các độ đo cực tiểu, cực đại. Mặc định là 0 và 100.
  - + init là giá trị ban đầu của đầu đọc. Giá trị mặc định là 50.
  - void setPaintTicks(boolean b): nếu b là true thì thước hiển thị
  - void setMajorTickSpacing(int units): Đặt khoảng cách cực đại giữa các vạch
  - void setMinorTickSpacing(int units): Đặt khoảng cách cực tiểu giữa các vạch
  - void setPaintLabels(boolean b): nếu b là true thì thước hiển thị nhãn
  - void setSnapToTicks(boolean b): nếu b là true thì đầu đọc luôn bám theo vạch đo.

#### 4.3.2. Thước tiến độ JProgressBar

Thước tiến độ JProgressBar là một thành phần đơn giản, nó chính là một hình chữ nhật được tô màu một phần để chỉ ra tiến độ thực hiện của một thao tác (công việc) nào đó. Mặc định, thanh tiến độ thể hiện số % hoàn thành công việc, ví dụ công việc cài đặt một phần mềm đã hoàn thành cho đến thời điểm đó.

Ta có thể thiết lập thước tiến độ giống như đối với thước đo. Ví dụ tạo ra thước tiến độ nǎm ngang:

```
JProgressBar progressBar = new JProgressBar(0, 100);
```

hay tạo ra thước tiến độ nǎm dọc:

```
progressBar = new JProgressBar(SwingConstants.VERTICAL, 0, 100);
```

Ta cũng có thể đặt các độ đo cực tiêu, cực đại thông qua phương thức `setMinimum()` và `setMaximum()`.

Tuy nhiên, thước tiến độ khác với thước đo ở chỗ người sử dụng không tự chỉnh sửa được độ đo tiến độ. Chương trình của ta phải gọi `setValue()` để thay đổi mức độ hoàn thành công việc.

Nếu ta gọi

```
progressBar.setStringPaint(true);
```

thì thước tiến độ sẽ tính n% hoàn thành công việc và hiển thị dưới dạng xâu “n%”. Nếu muốn hiển thị những thông tin khác thì sử dụng phương thức `setString()`, ví dụ:

```
if(progressBar.getValue() > 900)
    progressBar.setString("Almost Done");
```

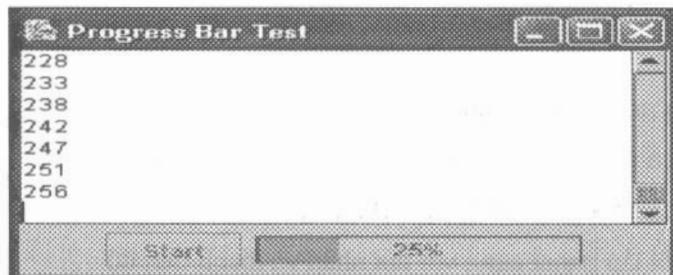
**Ví dụ 4.11.** Xây dựng chương trình có một thanh ghi tiến độ thể hiện sự tiêu tốn thời gian của máy tính như hình 4.15. Thước tiến độ có giá trị cực tiêu là 0 kể từ lúc nhấn nút bắt đầu và giá trị cực đại là 1000 đơn vị thời gian của máy tính.

Việc thay đổi mức độ tiêu tốn thời gian ở thước tiến độ được mô phỏng bởi lớp `SimulatedActivity`. Lớp này cài đặt để thực hiện theo luồng sao cho giá trị của `current` tăng lên 10 lần trong một giây. Khi `current` đạt đến `target` hoặc luồng thực hiện bị ngắt thì nó kết thúc.

```
class SimulatedActivity extends Thread{
    private int current;
    private int target;
    //...
    public void run(){
        while(current < target && !interrupted()){
            try{
                sleep(100);
            }catch(InterruptedException e){
                return;
            }
            current++;
        }
    }
}
```

Khi ta nhấn nút Start, một luồng mới bắt đầu thực hiện. Ta biết rằng javax.swing.Timer gọi phương thức actionPerformed() để lắng nghe các sự kiện và tính thời gian hiện thời current như sau:

```
public void actionPerformed(ActionEvent evt) {
    int current = activity.getCurrent();
    // Hiển thị ở thanh tiến độ
    textArea.append(current + "\n");
    progressBar.setValue(current);
    // Kiểm tra xem công việc đã kết thúc chưa
    if(current == activity.getTarget()) {
        activityMonitor.stop();
        startButton.setEnabled(false);
    }
}
```



Hình 4.15. Thanh tiến độ hoàn thành công việc

```
// ProgressTest.java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class ProgressTest{
    public static void main(String args[]){
        JFrame fr = new ProgressTestFrame();
        fr.show();
    }
}
```

// Khung chương trình thực hiện mô phỏng thanh tiến độ tiêu tốn thời gian

```
class ProgressTestFrame extends JFrame
{
    private Timer activityMonitor;
    private JButton startButton;
    private JProgressBar progressBar;
    private JTextArea textArea;
    private SimulatedActivity activity;
    public ProgressTestFrame(){
        setTitle("Progress Bar Test");
        setSize(300, 200);
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
        // Tạo ra vùng văn bản để lưu giữ kết quả
        Container content = getContentPane();
        textArea = new JTextArea();
        // Thiết lập panel cùng với nút nhấn và thanh tiến độ
        JPanel panel = new JPanel();
        startButton = new JButton("Start");
        progressBar = new JProgressBar();
        progressBar.setStringPainted(true);
        panel.add(startButton);
        panel.add(progressBar);
        content.add(new JScrollPane(textArea), "Center");
        content.add(panel, "South");
        // Khởi động hoạt động khi nhấn nút Start
        startButton.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent evt){
                    progressBar.setMaximum(1000);
                    activity = new SimulatedActivity(1000);
                }
            }
        );
    }
}
```

```
        activity.start();
        activityMonitor.start();
        startButton.setEnabled(false);
    }
});

// Khởi động hoạt động của đồng hồ Timer
activityMonitor = new Timer(500,
    new ActionListener(){
        public void actionPerformed(ActionEvent evt){
            int current = activity.getCurrent();
            // Hiển thị tiến độ
            textArea.append(current + "\n");
            progressBar.setValue(current);
            // Kiểm tra xem công việc đã kết thúc chưa
            if(current == activity.getTarget()){
                activityMonitor.stop();
                startButton.setEnabled(false);
            }
        }
    });
}

class SimulatedActivity extends Thread{
    private int current;
    private int target;
    public SimulatedActivity(int t){
        current = 0;
        target = t;
    }
    public int getTarget(){
        return target;
    }
    public int getCurrent(){

```

```

        return current;
    }

    public void run() {
        while(current < target && !interrupted()) {
            try{
                sleep(100);
            }catch(InterruptedException e) {
                return;
            }
            current++;
        }
    }
}

```

**javax.swing.JProgressBar****API**

- `JProgressBar()`
  - `JProgressBar(int direction)`
  - `JProgressBar(int min, int max)`
  - `JProgressBar(int min, int max)`
  - `JProgressBar(int direction, int min, int max)`
- Tạo ra thước đo theo hướng direction và các giá trị min, max tương ứng. Trong đó
- + direction có thể là `SwingConstraints.HORIZONTAL` hoặc `SwingConstraints.VERTICAL`. Mặc định là nằm ngang.
  - + min, max là các độ đo cực tiểu, cực đại. Mặc định là 0 và 100.
- `int getMinimum()`
  - `int getMaximum():` Xác định các giá trị cực tiểu, cực đại
  - `void setMinimum(int val)`
  - `void setMaximum(int val):` Đặt các giá trị cực tiểu, cực đại
  - `void setValue(int val):` Đặt lại giá trị hiện thời
  - `int getValue(int val):` Đọc giá trị hiện thời
  - `void setString(String s):` Đặt lại thông báo hiển thị
  - `String getString():` Đọc thông báo hiển thị

Thước tiền độ là một thành phần đơn giản có thể đặt bên trong một cửa sổ, một khung chương trình. Ngược lại, ProgressMonitor là một hộp thoại trong đó chứa một thước tiền độ. Tất nhiên, hộp thoại thường có hai nút: OK và Cancel. Nếu ta nhấn nút Cancel thì công việc đang được theo dõi sẽ bị huỷ bỏ mặc dù chưa hoàn thành. Chi tiết hơn ta có thể tham khảo ở tài liệu [2].

## BÀI TẬP

- 4.1. Viết chương trình hiển thị cây các chương, mục của giáo trình “Lập trình hướng đối tượng với Java”, trong đó ta có thể thêm vào hay bỏ đi một mục bất kỳ.
- 4.2. Biết rằng một chương trình Java có nhiều lớp, mỗi lớp lại có nhiều phương thức. Viết chương trình biểu diễn một cây, trong đó có các nút là các lớp. Các phương thức của mỗi lớp được biểu diễn là các nút con của nút lớp. Khi người sử dụng nhấn đúp vào nút lớp thì toàn bộ các thành phần của lớp bao gồm các thuộc tính, phương thức được hiển thị ở cửa sổ bên cạnh. Tương tự, khi nhấn đúp vào nút phương thức thì phần định nghĩa phương thức đó được hiển thị ở cửa sổ bên cạnh.
- 4.3. Viết chương trình để hiển thị bảng điểm kết quả học tập của sinh viên bao gồm: họ tên sinh viên, ngày sinh, điểm kỳ 1, điểm kỳ 2 và điểm tổng kết cuối năm. Điểm tổng kết cả năm được tính theo công thức: trung bình cộng của điểm kỳ 1 với hệ số 1 và điểm kỳ 2 với hệ số 2.
- 4.4. Viết chương trình truy cập được vào một CSDL, ví dụ CSDL tiền lương của một Công ty và hiển thị bảng lương trả cho nhân viên trong công ty.

# Chương 5

## JAVA BEAN

---

Chương này giới thiệu mô hình thành phần JavaBean:

- ✓ Bean và các thành phần của JavaBean
- ✓ Bộ công cụ phát triển Bean BDK
- ✓ Tạo ra các Bean riêng
- ✓ Xây dựng ứng dụng với Bean

### 5.1. GIỚI THIỆU VỀ JAVABEAN

JavaBean tăng cường cho Java các tính năng cho phép các đối tượng tương tác với nhau theo công nghệ thành phần. *Mô hình phần mềm thành phần* gồm: các thành phần được xem như là hạt nhân; các Container (bộ chứa), nơi các đối tượng được lắp ghép lại và tạo thành chương trình, người lập trình viết thêm các dòng lệnh thực hiện sự tương tác giữa các thành phần theo một kịch bản để thực hiện nhiệm vụ của bài toán. Một Container là nơi các thành phần có thể tự đăng ký và tạo ra các giao diện cho các thành phần khác biết cách tương tác với chúng. Chú ý rằng nhiều tính năng hỗ trợ OpenDoc cũng hỗ trợ JavaBean.

JavaBean là một mô hình thành phần hoàn chỉnh, hỗ trợ các tính năng chung cho kiến trúc thành phần, các thuộc tính, sự kiện, và lưu trữ. JavaBean được mô tả như sau:

- Interface Publishing and Discovery: Khi một thành phần được đặt vào Container của JavaBean, tính năng đăng ký đối tượng giúp nó hiện hữu giữa các đối tượng khác, phát hành các giao diện để các đối tượng khác có thể dùng được.
- Event Handling: Cho phép các đối tượng trao đổi với nhau qua thông báo.
- Persistence: Cung cấp khả năng lưu trữ thông tin về một sự kiện hay đối tượng để dùng sau này.
- Layout Control: Cung cấp các điều khiển để xây dựng giao diện trực quan, và bố trí các thành phần bên trong một Container.
- Application Builder Controls: Cho phép các thành phần bọc lô các thuộc tính và hành vi ra các công cụ phát triển, giúp các nhà phát triển xây dựng ứng dụng một cách nhanh chóng.

- InfoBus: Giao diện lập trình Java dạng nén và cho phép kết hợp các applet hoặc JavaBean trên trang Web hay trong ứng dụng Java để truyền thông dữ liệu với nhau. InfoBus là một cơ chế truyền thông được thiết kế để chạy trong một Client hay một Server đơn lẻ. Nó không phải là cơ chế truyền thông mạng. CORBA và RMI được thiết kế cho truyền thông mạng.

## 5.2. TẠI SAO PHẢI PHÁT TRIỂN JAVABEAN?

JavaBean là mô hình thành phần phần mềm. Nó tương tác với các đối tượng thành phần mềm khác và quyết định cách tạo lập, cách sử dụng lại các thành phần phần mềm đã được tạo dựng sẵn. Những thành phần này được xem như là các *hạt nhân (Bean)*. Hạt nhân hỗ trợ để phát triển chương trình mới trên cơ sở sử dụng lại những hạt nhân đã có sẵn và thiết lập mối quan hệ giữa chúng.

Những người đã quen với lập trình trên nền Window, đặc biệt là lập trình Visual Basic, Visual C++ hoặc Delphi, v.v., thì đều biết tại sao việc sử dụng lại các thành phần lại quan trọng. Một nguyên nhân làm cho Visual Basic trở nên phổ biến là nó hỗ trợ để tạo lập, sử dụng lại các thành phần phần mềm rất dễ dàng và tiện lợi. Người lập trình chỉ cần lựa chọn những thành phần thích hợp trong bộ công cụ điều khiển (Control) là những mẫu phần mềm được xây dựng sẵn, đặt chúng vào cửa sổ chương trình ứng dụng, khai báo, bổ sung và đặt lại các thuộc tính cho các thành phần, viết thêm một số đoạn mã lệnh để xử lý các sự kiện, kết nối các thành phần với nhau, v.v. Tuy nhiên, dễ nhận thấy những ngôn ngữ lập trình trực quan như kiểu Visual Basic không phải là công cụ tốt cho mọi bài toán. Mặt khác, công nghệ Java lại ngày càng tỏ rõ được phạm vi ứng dụng rộng rãi hơn. Và một điều may mắn là công nghệ JavaBean được hình thành trên cơ sở phát triển những ưu điểm của Visual Basic để hỗ trợ phát triển phần mềm và rất dễ sử dụng.

- Các hạt nhân Bean cùng với các chức năng điều khiển như Visual Basic Control luôn sẵn sàng.
- Bộ công cụ xây dựng công nghệ Java có các chức năng sử dụng tương tự như ở môi trường Visual Basic.
- Các thành phần được phát triển theo yêu cầu của môi trường lập trình phân tán.
- Java là ngôn ngữ lập trình độc lập với nền, vì vậy cần phải có công nghệ mới cho phù hợp. JavaBean là dễ chuyển đổi, có tính tương thích cao, nó có thể sử dụng với các thư viện với nền xác định và hỗ trợ Java Applet.

## 5.3. BỘ CÔNG CỤ PHÁT TRIỂN BEAN BDK

Trước khi bắt đầu viết chương trình sử dụng các thành phần hạt nhân Bean, ta cần phải tìm hiểu cách sử dụng và kiểm tra chúng như thế nào. Hiện nay có nhiều môi trường

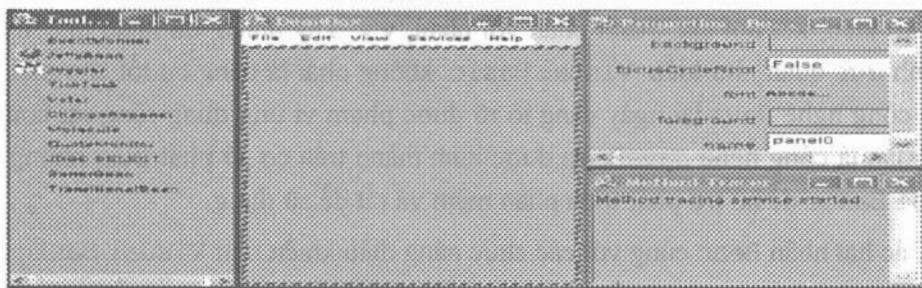
công cụ đã được xây dựng để phát triển thành phần Bean như: JavaPlan của MicroSystem, Java Studio của SunMicroSystem, Visual Café cho Java 2.5 của Symantec, Visual J++ của Microsoft, JBuilder của Borland, v.v. Ở đây chúng ta sử dụng BDK được JavaSoft phát triển, nó hỗ trợ cả Window 95, 96, NT và Solaris 2.4, 2.5, v.v. Ta có thể tải về từ [http://java.sun.com/beans/software/bdk\\_download.html](http://java.sun.com/beans/software/bdk_download.html), hay từ Java Web site (<http://java.sun.com/beans>). BDK chiếm khoảng gần 4MB, bao gồm:

- Bộ công cụ xây dựng BeanBox
- Chương trình hướng dẫn học sử dụng Tutorial và các tài liệu về Bean
- Bộ chứa JavaBean Container
- Một số chương trình ví dụ về Bean.

### 5.3.1. BeanBox

Bộ công cụ BeanBox là chương trình ứng dụng Java, nó chạy trên JDK 1.1 và độc lập với các nền. Để chạy được BeanBox thì phải chuyển về thư mục BeanBox bên trong thư mục BDK. Thư mục này được tự động tạo ra khi cài đặt BDK. Viết ‘run’ ở DOS Prompt, hoặc nhấn đúp vào ‘run.bat’ khi ở Window Explore để chạy BeanBox.

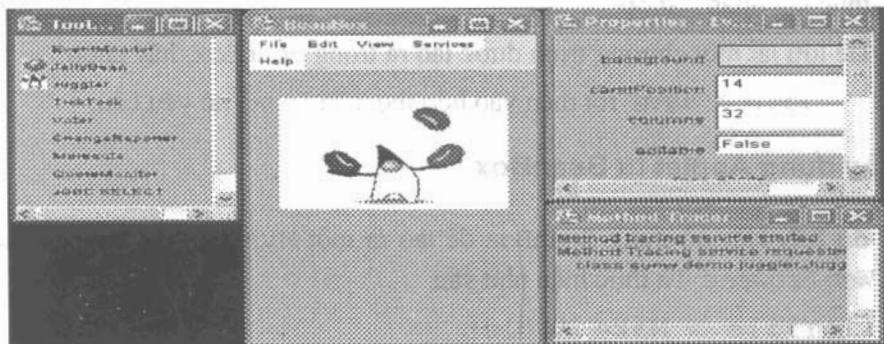
BeanBox khi khởi động sẽ có bốn Window có thể dịch chuyển và thay đổi kích cỡ: ToolBox, BeanBox, Properties và Method Tracer Windows như hình 5.1.



Hình 5.1. Các cửa sổ chính của BeanBox

- **ToolBox:** cửa sổ này liệt kê tất cả các thành phần Bean có sẵn trong BDK. Chúng được xây dựng như là các hạt nhân (gọi là Bean) để tạo ra những thành phần phức tạp hơn, những chương trình ứng dụng hoặc applet.
- **BeanBox:** là một bộ chứa Container hỗ trợ để ta thiết lập, thao tác trên các Bean. Nó là công cụ phát triển các thành phần trực quan. Để quan sát được một thành phần, ta phải chọn biểu tượng của thành phần đó và đặt vào vị trí thích hợp. Ví dụ, kích chuột vào biểu tượng nghệ sĩ tung hứng Juggler trong ToolBox, và sau đó kích vào vị trí thích hợp trong BeanBox Window. Juggler sẽ được đặt vào cửa sổ BeanBox.

- **Properties:** hiển thị các thuộc tính của Bean đang được chọn trong BeanBox. Khi Juggler được chọn như trên, thì cửa sổ về các thuộc tính của nó được hiển thị và người lập trình có thể sửa đổi chúng theo ý muốn.



Hình 5.2. Properties Window

### 5.3.2. Xây dựng chương trình ứng dụng bằng BeanBox

Trước tiên ta tìm hiểu cách sử dụng các Bean và thực hiện sửa đổi các đặc tính của chúng. Một *thành phần được chọn* (đường biên có sọc vằn) trong BeanBox Window, có thể thực hiện việc soạn thảo, sửa đổi, sao chép và cắt, dán ở chỗ nào khác trên Window. Đối với những Bean được chọn, có những đặc tính sau có thể thay đổi được: Background (màu nền), Foreground (màu vẽ), AnimationRate (tỷ lệ hoạt hình), Name (tên) và Font chữ. Nếu AnimationRate bị thay đổi, tốc độ hoạt hình sẽ thay đổi theo.

**Ví dụ 5.1.** Hãy mở BeanBox, chọn Juggler trong ToolBox, di chuột vào BeanBox Window và kích chuột để đặt nó ở vị trí nào đó trên màn hình.

Ta có thể đặt thêm hai nút, “Start” để bắt đầu hoạt hình và “Stop” để kết thúc hoạt hình vào bên dưới hình Juggler. Những nút này hoàn toàn giống như Button trong Java, nó có màu nền, màu chữ, trường văn bản và nhãn. Những thuộc tính này có thể thay đổi cho phù hợp với yêu cầu bài toán.

Tiếp theo ta thiết lập mối quan hệ tương tác giữa các nút được tạo ra với Juggler.

1. Chọn nút “Start”
2. Chọn Edit > Event > button push rồi chọn tiếp sự kiện actionPerformed.

Nếu di chuột vào vùng BeanBox, ta sẽ thấy có một đường màu đỏ nối giữa nút “Start” với Juggler.

3. Khi mối tương tác giữa các thành phần được thiết lập, cửa sổ thông báo các sự kiện EventTargetDialog sẽ xuất hiện, trong đó có một số phương pháp khác nhau cần cài đặt cho phù hợp với yêu cầu.

Ta muốn rằng, khi nhấn nút “Start” thì hình người tung hứng bắt đầu hoạt động, do đó ta chọn startJuggling() rồi nhấn OK, BeanBox sẽ sinh ra đoạn mã chương trình cần thiết.

Tương tự như trên ta thực hiện đói với nút “Stop”.

Một khi sự tương tác như trên được thiết lập, một chương trình Java đã được tạo ra để hiển thị, bắt đầu thực hiện hoạt hình tung hùng khi nhấn nút “Start” và nó kết thúc khi nhấn “Stop”.

4. Để lưu lại một chương trình được tạo ra trong BeanBox, hãy chọn  
File > save rồi điền vào hộp thoại File mà nó yêu cầu.

### 5.3.3. Xây dựng Applet từ BeanBox

**Ví dụ 5.2.** Hãy sử dụng BeanBox để tạo ra một MyApplet cho phép thực hiện được dưới các Web Browser. Ta thực hiện như sau.

1. Chọn File > Make Applet
2. Trong hộp thoại Dialog Box (hình 5.8), ta cho tên mới cho JAR File, tên tệp lớp MyApplet chính hoặc chấp nhận tên mặc định. Nhấn OK để BeanBox tạo ra một mã HTML, ví dụ MyApplet.html.

## 5.4. TẠO LẬP MỚI THÀNH PHẦN BEAN

Để xây dựng một thành phần hạt nhân (Bean) mới ta làm như sau:

1. Tạo ra một Bean mới
2. Dịch và lưu vào Java Archive File (JAR File).
3. Nạp thành phần được tạo ra vào ToolBox
4. Đặt một đại diện của thành phần trên vào cửa sổ BeanBox
5. Kiểm tra lại các thuộc tính, các phương pháp và các sự kiện cho chính xác
6. Phát sinh báo cáo thanh tra về thành phần đó.

**Ví dụ 5.3.** Xây dựng một thành phần hạt nhân CustomBean để vẽ một hình chữ nhật đơn giản. Thành phần này chính là lớp CustomBean được lưu trong tệp CustomBean.java.

+ Viết chương trình cho thành phần mới.

```
// CustomBean.java
import java.awt.*;
import java.io.Serializable;

public class CustomBean extends Canvas implements Serializable{
    // Phương pháp tạo lập đặt lại các thuộc tính được kế thừa
    public CustomBean(){
        setSize(60, 40);
        setBackground(Color.red);
    }
}
```

**Lưu ý:**

1. Mọi thành phần hạt nhân (Bean) đều phải cài đặt giao diện Serializable. Thành phần trên kế thừa Canvas, cài đặt giao diện Serializable, làm nhiệm vụ ban đầu đơn giản là đặt lại cỡ của hình và màu nền là màu đỏ.
2. Đảm bảo rằng biến môi trường CLASSPATH được đặt để truy cập được tất cả các tệp lớp (.class) và các tệp lưu trữ (.jar).
  - + Kết quả dịch thành công chương trình trên là tệp CustomBean.class.
  - + Tiếp theo, tạo ra một tệp thẻ hiện, đặt tên là manifest.tmp có nội dung như sau:

Name: CustomBean.class

Java-Bean: True

**Lưu ý: Đừng quên dấu cách sau Name: và sau thẻ Java-Bean:**

- + Tạo lập tệp lưu trữ (.jar) chứa manifest.tmp và CustomBean.class. Trong DOS-Prompt, thực hiện

`jar cfm CustomBean.jar manitest.tmp CustomBean.class`

- + Nạp tệp lưu trữ CustomBean.jar vào ToolBox. Trong môi trường BDK:

Chọn File > LoadJar

Hệ thống mở ra File Browser, hãy chọn thư mục nơi có chứa CustomBean.jar xác định thành phần cần đưa vào ToolBox. Kết quả là thành phần được chọn sẽ được bổ sung vào ToolBox.

- + Sử dụng thành phần hạt nhân mới trong chương trình ứng dụng theo các bước như đã nêu ở ví dụ 5.2.

#### **5.4.1. Kiểm tra các thuộc tính và các sự kiện của Bean**

Như trên ta đã thấy, khi một hạt nhân được chọn thì trang màn hình thuộc tính của nó sẽ được hiển thị, bao gồm: foreground, background, font, name. Những thuộc tính này không được khai báo trong CustomBean, mà được kế thừa từ lớp Canvas. BeanBox cung cấp các kiểu nguyên thuỷ mặc định cho những thuộc tính trên. Các Bean sẽ gửi và nhận các thông báo về các sự kiện xảy ra để trao đổi với các Bean khác.

#### **5.4.2. Phát sinh báo cáo về các thuộc tính**

Các đặc tính của các thành phần hạt nhân đóng vai trò quan trọng trong việc quyết định sử dụng chúng trong thiết kế chương trình ứng dụng, cho nên cần phải có các báo cáo mô tả được các tính năng chính của các thành phần đó.

Để tạo ra một báo cáo về một thành phần, chọn Edit > Report trong BDK. Báo cáo sẽ liệt kê các sự kiện, thuộc tính, các phương pháp và các đặc tính của thành phần đã được xây dựng.

## 5.5. THIẾT LẬP CÁC THUỘC TÍNH CHO BEAN

Trong phần này ta tìm hiểu về cách thiết lập các thuộc tính cho các Bean. Các thuộc tính của Bean được phân thành:

- **Simple Properties:** Giải thích cách xác định các thuộc tính cho Bean.
- **Bound Properties:** Mô tả cách cài đặt các thuộc tính và các sự kiện trao đổi với các đối tượng khác.
- **Constrained Properties:** Mô tả cách các thuộc tính khi bị thay đổi có được chấp nhận hay không.
- **Indexed Properties:** Mô tả các thuộc tính được chỉ số hoá.

### 5.5.1. Các thuộc tính đơn giản Simple Properties

Các thuộc tính đơn giản mô tả trạng thái xuất hiện và các hành vi của Bean mà ta có thể thay đổi được trong thời gian thiết kế. Đây là những thuộc tính riêng và muốn truy cập các phương pháp để đọc hoặc ghi chúng. Bộ công cụ dựa trên JavaBean sử dụng những phương pháp được thiết lập theo mẫu thiết kế để thực hiện:

- Phát hiện các thuộc tính của Bean
- Xác định các thuộc tính đọc/ghi
- Xác định các kiểu của thuộc tính
- Hiển thị các thuộc tính
- Thay đổi các thuộc tính trong thời gian thiết kế.

Ví dụ, ta cần bổ sung thuộc tính Color vào CustomBean, do vậy lớp CustomBean có thêm các lệnh như sau:

```
public class CustomBean extends Canvas implements Serializable{
    private Color color = Color.green;
    // Phương pháp tạo lập đặt lại các thuộc tính được kế thừa
    public CustomBean() {
        setSize(60, 40);
        setBackground(Color.red);
    }
    public Color getColor() {
        return color;
    }
    public void setColor(Color newColor) {
        color = newColor;
    }
}
```

```

        repaint();
    }
    // Viết đè phương pháp paint()
    public void paint(){
        g.setColor(color);
        g.fillRect(20, 5, 20, 30);
    }
}

```

CustomBean sẽ hiển thị hình chữ nhật màu xanh lá cây ở giữa khung cửa sổ. Khi muốn thay đổi màu, ta kích đúp vào vùng màu ứng với color ở cửa sổ Properties, một hộp thoại soạn thảo tương ứng được hiển thị để ta có thể thay đổi theo ý muốn.

### 5.5.2. Các thuộc tính biên Bound Properties

Nhiều khi một thành phần thay đổi thuộc tính thì những đối tượng khác phải biết được những sự thay đổi đó để ứng xử cho thích hợp. Khi những *thuộc tính biên* thay đổi, một thông báo về sự thay đổi đó được gửi đến cho những đối tượng liên quan.

Để thực hiện được những yêu cầu trên, lớp PropertyChangeSupport phải cài đặt phương pháp bổ sung, loại bỏ đối tượng của PropertyChangeListener khỏi danh sách. Ta thực hiện thêm các lệnh sau trong CustomBean:

```

// Nhập gói java.beans để truy cập vào PropertyChangeSupport
import java.beans;

// Tạo lập một đối tượng changes để lắng nghe sự thay đổi
private PropertyChangeSupport changes
= new PropertyChangeSupport(this);

// Bổ sung đối tượng thể hiện sự thay đổi vào danh sách
public void addPropertyChangeListener(PropertyChangeListener l){
    changes.addPropertyChangeListener(l);
}

// Loại một đối tượng ra khỏi danh sách
public void removePropertyChangeListener(PropertyChangeListener l){
    changes.removePropertyChangeListener(l);
}

```

```
// Đặt tên mới cho nhãn

public void setLabel(String newLabel) {
    String oldLabel = label;
    label = newLabel;
    sizeToFit();
    changes.firePropertyChange("label", oldLabel, newLabel);
}
```

**Lớp java.beans.PropertyChangeListener****API**

- `void propertyChange(PropertyChangeEvent event)`: Được gọi khi xuất hiện một kiện thay đổi thuộc tính.

**Lớp java.beans.PropertyChangeSupport****API**

- `void propertyChangeSupport(Object sourceBean)`: Tạo ra một đối tượng, trong đó sourceBean thường this.
- `void addPropertyChangeListener(PropertyChangeListener listener)`: Đưa thêm listener vào danh sách các phần tử lắng nghe.
- `void RemovePropertyChangeListener(PropertyChangeListener listener)`: Đưa listener ra khỏi danh sách các phần tử lắng nghe.
- `void firePropertyChange(String propertyName, Object oldValue, Object newValue)`: Gửi PropertyChangeEvent tới các phần tử lắng nghe.

**Lớp java.beans.PropertyChangeEvent****API**

- `PropertyChangeEvent(Object source, String propertyName, Object oldValue, newValue)`: Tạo ra một đối tượng sự kiện mới.
- `Object getNewValue()`: Đọc giá trị mới của thuộc tính.
- `Object getOldValue()`: Đọc giá trị cũ của thuộc tính.
- `String getPropertyName()`: Đọc tên của thuộc tính.
- `PropertyChangeEvent(Object source, String propertyName, Object oldValue, newValue)`: Tạo ra một đối tượng sự kiện mới.
- `Object getNewValue()`: Đọc giá trị mới của thuộc tính.
- `Object getOldValue()`: Đọc giá trị cũ của thuộc tính.
- `String getPropertyName()`: Đọc tên của thuộc tính.

### 5.5.3. Các thuộc tính bị khống chế Constrained Properties

Thuộc tính của Bean bị khống chế khi có một sự thay đổi thuộc tính mà không được chấp nhận.

JavaBean API cung cấp một cơ chế xử lý các sự kiện tương tự như đối với các thuộc tính biến, cho phép không chấp nhận sự thay đổi của các thuộc tính. Do vậy, phải cài đặt ít nhất một thuộc tính bị khống chế trong Bean.

```

private VetoableChangeSupport vetos
    = new VetoableChangeSupport(this);
// Bổ sung đối tượng phủ định thay sự đổi vào danh sách
public void addVetoableChangeListener(VetoableChangeListener l) {
    vetos.addVetoableChangeListener(l);
}
// Loại một đối tượng ra khỏi danh sách
public void removeVetoableChangeListener(VetoableChangeListener l) {
    vetos.removePropertyChangeListener(l);
}
// Đặt tên mới cho nhãn
public void setPriceInCents(int newPrice)
    throws PropertyVetoException{
    int oldPrice = ourPrinceInCents;
    vetos.fireVetoableChange("priceInCents",
        new Integer(oldPrice), new Integer(newPrice));
    ourPrinceInCents = newPrice;
    sizeToFit();
    changes.firePropertyChange("priceInCents",
        new Integer(oldPrice), new Integer(newPrice));
}

```

Lớp `java.beans.PropertyChangeListener`.

API

- `void VetoableChange(PropertyChangeEvent event)`: được gọi khi xuất hiện một kiện thay đổi thuộc tính.

Lớp `java.beans.VetoableChangeSupport`

API

- `void VetoableChangeSupport(Object sourceBean)`: Tạo ra một đối tượng, trong đó `sourceBean` thường là `this`.

- void addVetoableChangeListener(VetoableChangeListener listener): **Đưa thêm** listener vào danh sách các phần tử lắng nghe.
- void RemoveVetoableChangeListener(VetoableChangeListener listener): **Đưa** listener ra khỏi danh sách các phần tử lắng nghe.
- void firePropertyChange(String propertyName, Object oldValue, Object newValue): **Gửi** PropertyChangeEvent tới các phần tử lắng nghe.

#### 5.5.4. Các thuộc tính chỉ số hóa Indexed Properties

Thuộc tính chỉ số hoá biểu diễn những tuyển tập các giá trị cho phép truy cập giống như đối với mảng. Nó có dạng:

```
// Truy cập vào mảng các giá trị
public <.PropertyType>[] get <PropertyName>[];
public void <PropertyName>(<.PropertyType>[] value);

// Truy cập vào từng giá trị
public <.PropertyType> get <PropertyName>(int index);
public void set <.PropertyType> (int index, <.PropertyType> val);
```

### 5.5. CÁC PHẦN TỬ LẮNG NGHE CÁC SỰ KIỆN TRÊN CÁC THÀNH PHẦN

Để các thành phần tương tác được với các thành phần khác khi có một sự kiện được phát sinh bởi một thành phần thì sẽ phải có nhiều hơn một thành phần Bean nhận được và xử lý.

*Mô hình sự kiện đại biểu* hỗ trợ để thực hiện công việc trên. Nó cung cấp cơ chế chuẩn để một nguồn có thể phát sinh ra các sự kiện và gửi chúng đến những phần tử lắng nghe. Một *đối tượng sự kiện* mô tả sự thay đổi trạng thái của nguồn. Ví dụ, khi nhấn vào một nút Button, kết thúc một dòng văn bản (nhấn phím Enter), hay đóng Window, v.v. là sẽ phát sinh ra các sự kiện. *Nguồn sự kiện* gây ra các sự kiện. Trách nhiệm của nguồn các sự kiện là:

- Cung cấp các phương thức để những phần tử lắng nghe ghi nhận được các kiểu của các sự kiện
- Phát sinh ra các sự kiện
- Gửi tất cả các sự kiện đến những phần tử đang lắng nghe.

*Phần tử lắng nghe* nhận thông báo về các sự kiện và có trách nhiệm:

- Ghi nhận các sự kiện bằng cách gọi những phương thức tương ứng với nguồn của sự kiện
- Cài đặt một giao diện để nhận sự kiện loại đó

- Huỷ bỏ sự kiện đã được ghi nhận khi nó không còn cần thiết.

## 5.6. THIẾT LẬP MỘT BEAN RIÊNG CỦA BẠN

Giả sử rằng ta muốn ghi nhận số lần kích chuột vào các *nút liên kết* trên trang Web của mình. Nghĩa là, ta muốn xây dựng một thành phần Bean để đếm những lần nhấp vào các nút đó. Sau đây ta xây dựng hai thành phần Bean. Thành phần thứ nhất là MyBean có bộ đếm counter, giá trị này được hiển thị ở trường văn bản và sẽ được tăng lên 1 khi nhấp vào MyBean. Thành phần này được viết như sau.

### Ví dụ 5.4

```
import java.awt.*;
import java.awt.event.*;
public class MyBean extends Canvas implements MouseListener{
    int num;
    ActionListener a1;
    public MyBean(){
        addMouseListener(this);
    }
    public Dimension getPreferredSize(){
        return new Dimension(100, 100);
    }
    public void setValue(int x){
        num = x;
    }
    public void paint(Graphics g){
        g.drawString(" " + num, 50, 50);
    }
    public int getValue(){
        return num;
    }
    public void increment(){
        num++;
        repaint();
    }
}
```

```

public void addActionListener(ActionListener a) {
    a1 = a;
}
public void removeActionListener(ActionListener a) {
    a1 = a;
}

public void mouseClicked(MouseEvent me) {
    if(me.getSource() == new TextLink())
        a1.actionPerformed(new ActionEvent(this, 1, "" + num));
}
public void mouseEntered(MouseEvent me) {}
public void mouseExited(MouseEvent me) {}
public void mouseReleased(MouseEvent me) {}
public void mousePressed(MouseEvent me) {}

}

```

Thành phần Bean thứ hai là TexLink làm nhiệm vụ tạo ra trường văn bản liên kết.

```

import java.awt.*;
import java.awt.event.*;
public class TextLink extends TextField{
    public TextLink(){
        super("www.hotmail.com");
    }
    public Dimension getPreferredSize(){
        return new Dimension(100, 100);
    }
}

```

Các sự kiện hoạt động được phát sinh khi một nút được nhấn, một mục trong danh sách được kích đúp, một mục trong thực đơn được lựa chọn. Ở đây khi sự kiện hoạt động xảy ra khi Button được nhấn và nó sẽ gọi increment() để tăng số lần đếm.

Sau khi dịch hai thành phần trên, ta tạo ra tệp biểu thị 1 tên là MyBean.mf chứa:

```

Manifest_version: 1.0
Name: MyBean.class
Java-Bean: True;

```

Tương tự, tệp biểu thị 2: TexLink.mf gồm có:

```

Manifest_version: 1.0

```

Name: TexLink.class

Java-Bean: True;

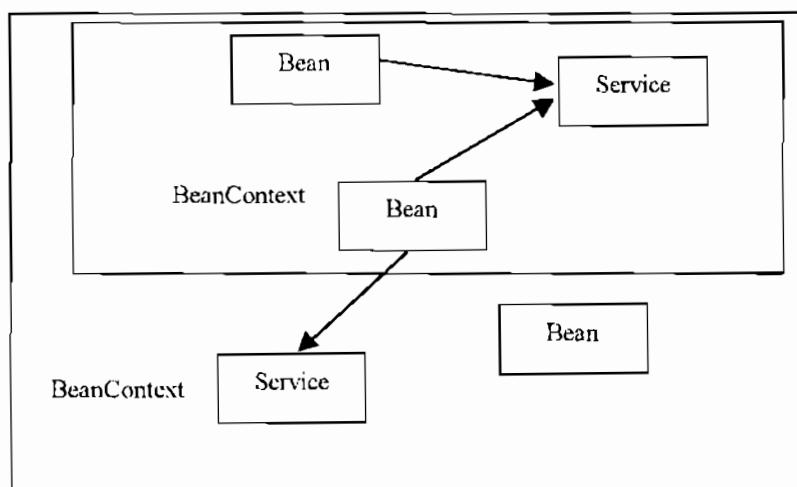
Bước cuối là tạo ra tệp lưu trữ JAR có tên: Counter.jar ở trong môi trường DOS-Prompt:

```
jar -cfm Counter.jar MyBean.mf MyBean.class
```

Khi tệp Counter.jar được tạo ra ở thư mục hiện thời, ta cần phải sao sang thư mục JAR của BDK để có thể nhìn thấy nó trong ToolBox sau khi tải nó vào BDK như đã nói ở trên.

### 5.7. NGỮ CẢNH CỦA BEAN

Chúng ta tiếp tục tìm hiểu cách tạo ra các Bean để tận dụng được các thế mạnh của chúng. Vẫn đề cơ bản là cài đặt các Bean để chúng tương tác được với các Bean khác hoặc sử dụng được các dịch vụ mà BeanBox cung cấp. Muốn thế ta phải biết được *ngữ cảnh* của Bean. Ngữ cảnh của Bean chứa các Bean, các dịch vụ và ngữ cảnh Bean khác, giống như bộ chứa Container của AWT. Ví dụ BeanBox của BDK1.1 chính là một ngữ cảnh của Bean. Nó cung cấp dịch vụ theo dõi các vết của việc hiển thị các thông điệp.



Hình 5.3. Ngữ cảnh của Beam

Để Bean trao đổi được với ngũ cảnh xung quanh, nó phải cài đặt giao diện `BeanContextChild`. Giao diện này đã khai báo:

```
void setBeanContext(BeanContext bc)
```

```
BeanContext getBeanContext(BeanContext bc)
```

```
void addPropertyChangeListener(String name,
```

```
PropertyChangeListener listener)
```

```
void removePropertyChangeListener(String name,
```

```
PropertyChangeListener listener)
```

```

void addVetoableChangeListener(String name,
    VetoableChangeListener listener)

void removeVetoableChangeListener(String name,
    VetoableChangeListener listener)

```

Ngữ cảnh của Bean sẽ gọi setBeanContext() khi nó bắt đầu quản lý các Bean. Công việc cài đặt các phần tử lắng nghe các sự kiện thay đổi của các thuộc tính là khá vất vả, nên tốt nhất là sử dụng lớp BeanContextChildSupport để cài đặt chúng. Nhưng Bean của ta không thể đồng thời kế thừa một lớp thành phần và BeanContextChildSupport, nên ta phải thực hiện theo một cơ chế đặc biệt như sau:

1. Cài đặt giao diện BeanContextProxy
2. Tạo ra đối tượng của BeanContextChildSupport

```

private BeanContextChildSupport childSupport
    = new BeanContextChildSupport();

```

3. Viết hàm getBeanContextProxy() để nhận lại đối tượng được tạo ra

```

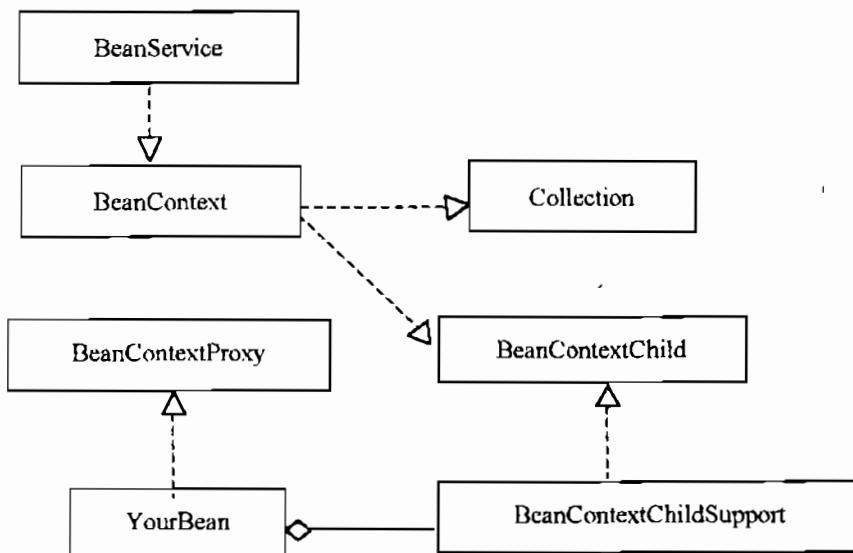
public BeanContextChildSupport getBeanContextProxy() {
    return childSupport;
}

```

4. Gọi getBeanContext() để xác định được ngữ cảnh của nó

```
BeanContext context = childSupport.getBeanContext();
```

Các lớp nêu trên có quan hệ với nhau như hình 5.4



**Hình 5.4. Mối quan hệ giữa các lớp trong ngữ cảnh của Bean**

**Ví dụ 5.5.** Hãy xét một spin Bean, một phần tử điều khiển nhỏ có hai nút Button để tăng hay giảm một giá trị.



Hình 5.5. Spin Bean đơn giản

Tự bản thân spin Bean thì chẳng hữu dụng. Nó cần phải cặp bô với Bean khác. Ví dụ, khi cặp bô với IntTextBean chứa các giá trị nguyên, thì giá trị số nguyên sẽ tăng lên 1 khi nhấn nút '+', hoặc giảm khi nhấn nút '-'.

```
// SpinBean.java
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import java.beans.beancontext.*;
import java.lang.reflect.*;
import java.io.*;
import java.util.*;
import javax.swing.*;

public class SpinBean extends JPanel implements Serializable,
    BeanContextProxy{
    public SpinBean(){
        setLayout(new GridLayout(2, 1));
        JButton plusButton = new JButton("+");
        JButton minusButton = new JButton("-");
        add(plusButton);
        add(minusButton);
        plusButton.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent evt){
                spin(1);
            }
        });
        minusButton.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent evt){
                spin(-1);
            }
        });
    }
    public void spin(int value){ // This method is not shown in the code snippet
}
}
```

```
childSupport = new BeanContextChildSupport(){
    public void setBeanContext(BeanContext context)
        throws PropertyVetoException{
        super.setBeanContext(context);
        setTracer(context);
    }
};

public BeanContextChild getBeanContextProxy(){
    return childSupport;
}

public void setBuddy(Component b, PropertyDescriptor p){
    buddy = b;
    prop = p;
}

public void spin(int incr){
    if(buddy == null) return;
    if(prop == null) return;
    Method readMethod = prop.getReadMethod();
    Method writeMethod = prop.getWriteMethod();
    try{
        int value = ((Integer)readMethod.invoke(buddy, null)).intValue();
        if(tracer != null){
            String text = "spin: value ='" + value
                + "increment = " + incr;
            Method logText = tracerClass.getMethod("logText",
                new Class[] {String.class});
            logText.invoke(tracer, new Object[] {text});
        }
        value += incr;
        writeMethod.invoke(buddy,
            new Object[] {new Integer(value)} );
    }
}
```

```
        }catch(Exception e){  
        }  
    }  
  
    public void setTracer(BeanContext context){  
        try{  
            BeanContextServices services  
                = (BeanContextServices) context;  
            tracerClass = Class.forName  
                ("sunw.demo.methodtracer.MethodTracer");  
            if(services.hasService(tracerClass)){  
                BeanContextServiceRevokedListener revokedListener  
                    = new BeanContextServiceRevokedListener(){  
                        public void serviceRevoked(  
                            BeanContextServiceRevokedEvent event){  
                                tracer = null;  
                            }  
                        };  
                tracer = services.getService(childSupport, this,  
                    tracerClass, null, revokedListener);  
            }  
        }catch(Exception e){  
            tracer = null;  
        }  
    }  
  
    public Dimension getPreferredSize(){  
        return new Dimension(MINSIZE, MINSIZE);  
    }  
  
    private static final int MINSIZE = 20;  
    private Component buddy;  
    private PropertyDescriptor prop;  
    private Object tracer;  
    private Class tracerClass;  
    private BeanContextChildSupport childSupport;  
}
```

```
// SpinBeanCustom.java

import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import java.beans.beancontext.*;
import java.text.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;

public class SpinBeanCustom extends JPanel implements Customizer{

    private SpinBean bean;
    private PropertyChangeSupport support
        = new PropertyChangeSupport(this);
    private JList buddyList;
    private JList propList;
    private DefaultListModel buddyModel;
    private DefaultListModel propModel;
    private PropertyDescriptor[] props;
    private Component[] buddies;

    public SpinBeanCustom(){
        setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.weightx = 0;
        gbc.weighty = 100;
        gbc.fill = GridBagConstraints.NONE;
        gbc.anchor = GridBagConstraints.EAST;
        add(new JLabel("Buddy"), gbc, 0, 0, 1, 1);
        add(new JLabel("Property"), gbc, 0, 1, 1, 1);
        gbc.weightx = 100;
        gbc.fill = GridBagConstraints.WEST;
        gbc.anchor = GridBagConstraints.BOTH;
```

```
buddyModel = new DefaultListModel();
propModel = new DefaultListModel();
buddyList = new JList(buddyModel);
propList = new JList(propModel);
add(new JScrollPane(buddyList), gbc, 1, 0, 1, 1);
add(new JScrollPane(propList), gbc, 1, 1, 1, 1);
JButton setButton = new JButton("Set Buddy");
JPanel p = new JPanel();
p.add(setButton);
add(p, gbc, 0, 2, 2, 1);

buddyList.addListSelectionListener(
    new ListSelectionListener(){
        public void valueChanged(ListSelectionListener evt){
            findBuddyMethods();
        }
    });
setButton.addActionListener(
    new ActionListener(){
        public void actionPerformed(ActionEvent evt){
            int buddyIndex = buddyList.getSelectedIndex();
            if (buddyIndex < 0) return;
            int propIndex = propList.getSelectedIndex();
            if (propIndex < 0) return;
            bean.setBuddy(buddies[buddyIndex],
                props[propIndex]);
        }
    });
}

public void add(Component c, GridBagConstraints gbc, int x,
    int y, int w, int h){
    gbc.gridx = x;
    gbc.gridy = y;
```

```
        gbc.gridx = w;
        gbc.gridy = h;
        add(c, gbc);
    }

    public void findBuddyMethods(){
        int buddyIndex = buddyList.getSelectedIndex();
        if(buddyIndex < 0) return;
        Component buddy = buddies[buddyIndex];
        propModel.removeAllElements();
        try{
            BeanInfo info
            = Introspector.getBeanInfo(buddy.getClass());
            props = info.getPropertyDescriptors();
            int j = 0;
            for(int i = 0; i < props.length; i++){
                Class propertyType = props[i].getPropertyType();
                if(int.class.equals(propertyType)){
                    String name = props[i].getName();
                    propModel.addElement(name);
                    props[j++] = props[i];
                }
            }
        }catch(IntrospectionException e){}
    }

    public Dimension getPreferredSize(){
        return new Dimension(300, 200);
    }

    public void setObject(Object obj){
        bean = (SpinBean)obj;
        BeanContext context
            = bean.getBeanContextProxy().getBeanContext();
        buddies = new Component[context.size()];
        buddyModel.removeAllElements();
    }
}
```

```

        Iterator iter = context.iterator();
        int i = 0;
        while(iter.hasNext()){
            Object buddy = iter.next();
            if(buddy instanceof Component){
                buddies[i] = (Component)buddy;
                String className = buddies[i].getClass().getName();
                buddyModel.addElement(className);
                i++;
            }
        }
    }

    public void addPropertyChangeListener(PropertyChangeListener l){
        support.addPropertyChangeListener(l);
    }

    public void removePropertyChangeListener(PropertyChangeListener l)
    {
        support.removePropertyChangeListener(l);
    }
}

//SpinBeanBeanInfo.java
import java.awt.*;
import java.beans.*;

public class SpinBeanBeanInfo extends SimpleBeanInfo{
    public BeanDescriptor getBeanDescriptor(){
        return new BeanDescriptor(SpinBean.class,
                                  SpinBeanCustom.class);
    }
}

```

Lớp `java.beans.beancontext.BeanContextChild`

API

- `void setBeanContext(BeanContext bc)`: được gọi khi đưa thêm một bean vào như là nút con.
- `BeanContext getBeanContext()`: xác định ngữ cảnh của bean.

- `void addPropertyChangeListener(String name, PropertyChangeListener listener)`: đưa thêm listener vào thuộc tính của bean
- `void removePropertyChangeListener(String name, PropertyChangeListener listener)`: loại listener ra khỏi thuộc tính của bean
- `void addVetoableChangeListener(String name, VetoableChangeListener listener)`: đưa thêm listener có khả năng phủ nhận vào thuộc tính của bean
- `void removeVetoableChangeListener(String name, VetoableChangeListener listener)`: loại listener có khả năng phủ nhận ra khỏi thuộc tính của bean

## BÀI TẬP

**5.1.** Chọn những từ thích hợp để điền vào chỗ trống trong các mệnh đề sau.

1. ----- là mô hình thành phần mềm được tạo ra bằng Java.
2. Bạn có thể sử dụng bộ công cụ như ----- để cá nhân hóa các thành phần.
3. ----- là các thành phần chứa những thành phần khác.
4. Trong JavaBean, bạn cần sử dụng những phương thức giao diện đặc biệt được gọi là phương thức ----- để truy cập vào các đặc tính.
5. ----- cho phép những công cụ lập trình ào hoắc những bộ công cụ để phân tích các cấu trúc bên trong của một thành phần.
6. Các Bean có thể được nén như các tệp -----.

**5.2.** Xây dựng một Bean thể hiện Juggler trên màn hình, trong đó có hai nút “Start” và nút “Stop” để khởi động và kết thúc hoạt động của người tung hứng.

## CHƯƠNG 6

# PHÁT TRIỂN CÁC DỊCH VỤ SERVLET VÀ JSP

Chương VI giới thiệu mô hình dịch vụ Web từ phía máy chủ:

- ✓ Giới thiệu về Servlet
- ✓ Chu trình sống của Servlet
- ✓ Sử dụng Servlet để truy tìm thông tin
- ✓ Sử dụng Servlet để gửi tin
- ✓ Sử dụng Servlet để truy cập vào CSDL
- ✓ JSP (Java Server Page).

### 6.1. GIỚI THIỆU VỀ SERVLET

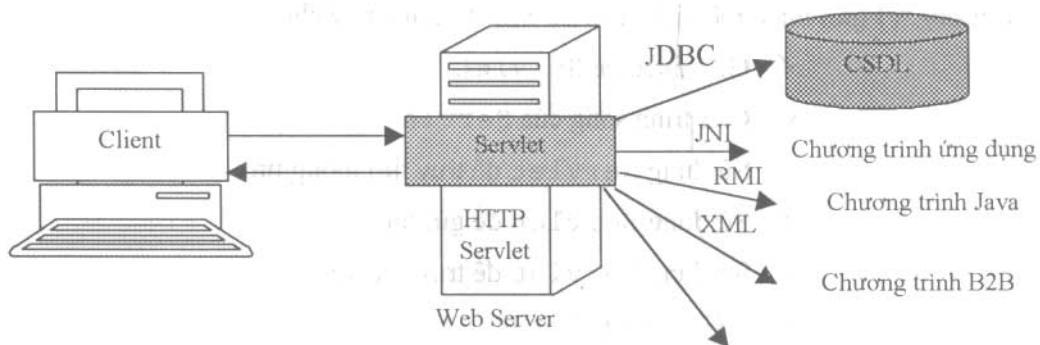
Hiện nay, trong lập trình có một xu hướng rất quan trọng đang được tập trung phát triển ứng dụng, đó là xây dựng các chương trình dịch vụ Java ở phía máy chủ (Server) ([2], [4]).

Servlet là thành phần chính được sử dụng để phát triển các chương trình dịch vụ Java ở phía máy chủ. Các Servlet là các chương trình Java thực hiện ở các ứng dụng Server (tên gọi “Servlet” cũng gần giống như “Applet” ở phía máy Client) để trả lời cho các yêu cầu của Client. Các Servlet không bị ràng buộc chặt với một giao thức Client-Server cụ thể nào cả, nhưng giao thức thường được sử dụng là HTTP, do vậy, khi nói tới Servlet nghĩa là nói tới HTTP Servlet. Servlet là sự phát triển mở rộng của CGI để đảm bảo Server thực hiện được các chức năng của mình. Ta có thể sử dụng Servlet của Java để tùy chỉnh lại một dịch vụ bất kỳ, như Web Server, Mail Server, v.v.

Web Server hiển thị các tư liệu được viết trong HTML và hồi đáp cho yêu cầu của người sử dụng qua HTTP. Các tư liệu HTML chứa các văn bản được đánh dấu (định dạng) để các trình duyệt như IE, Netscape đọc được.

Một trình duyệt chấp nhận đầu vào ở dạng HTML, khi người sử dụng nhấn một nút để yêu cầu một số thông tin nào đó, một Servlet đơn giản được gọi để xử lý các yêu cầu đó. Các công việc chính của Servlet được mô tả khái quát trong hình 6.1, bao gồm:

- Đọc các dữ liệu tường minh được Client gửi đến từ các yêu cầu (dữ liệu theo các khuôn dạng – form data).
- Đọc các dữ liệu không tường minh được Client gửi đến từ các yêu cầu (dữ liệu trong phần đầu của yêu cầu – request headers).
- Xử lý và lưu trữ các dữ liệu được cung cấp dưới dạng HTML.
- Gửi trả lời dữ liệu tường minh cho Client (dạng HTML), cung cấp các nội dung động, ví dụ trả lời yêu cầu Client về các câu truy vấn vào các CSDL.
- Quản lý các thông tin trạng thái và trả lời dữ liệu không tường minh cho Client (các mã trạng thái và các phần đầu của trả lời).



Hình 6.1. Vai trò của Servlet

Viết một Servlet là tương đối dễ. Ta chỉ cần có Tomcat, nó là tổ hợp của Java Server Pages™ 1.1 và Servlets 2.2. Tomcat có thể nạp miễn phí từ <http://java.sun.com/products/jsp/tomcat/>, phần cài đặt sẽ được mô tả ở phần sau.

Các Servlet cũng được sử dụng thay cho kịch bản giao diện cổng chung CGI Script. Khi tạo ra một trang Web, ta cũng sẽ tạo ra một ứng dụng Web.

Trước khi sử dụng Servlet để tạo ra các ứng dụng Web, chúng ta đi tìm hiểu xem có những khả năng lựa chọn nào khác để phát triển những ứng dụng Web.

- **CGI:** Theo cách thông thường, khi cần bổ sung các chức năng mới cho một Web Server người ta hay sử dụng Common Gateway Interface (CGI), một giao diện độc lập với ngôn ngữ cho phép một Server khởi động một tiến trình ngoại để nhận thông tin được yêu cầu thông qua các biến môi trường. Mỗi yêu cầu được trả lời bởi một tiến trình riêng thông qua một đại diện riêng của một chương trình CGI hoặc bởi một kịch bản CGI (thường được viết bằng ngôn ngữ thông dịch như Perl).
- **Fast CGI:** Open Marked đã phát triển một chuẩn khác thay cho CGI được gọi là Fast CGI. Fast CGI hành động giống như CGI. Nó khác ở chỗ, Fast CGI tạo ra một tiến trình bền vững cho từng chương trình.

- Một số chương trình ứng dụng khác như ASP và Java Script cũng hỗ trợ để tạo ra các ứng dụng Web. ASP được Microsoft phát triển để tạo ra các nội dung cho các trang Web động. Trong ASP, trang HTML có thể nhúng những phần nhỏ được viết bằng VBScript hoặc JScript. Netscape đưa ra kỹ thuật được gọi là JavaScript, cho phép đưa các phần mã lệnh nhỏ nhúng vào trang HTML, nhằm tạo ra những nội dung Web động một cách linh hoạt hơn. Ngoài ra, Netscape còn cung cấp NSAPI, Microsoft đưa ra ISAPI cho các Web Server của họ.

Servlet có một số ưu điểm so với CGI:

- Một Servlet không làm việc trong một tiến trình riêng. Điều này loại bỏ được việc phải tạo ra quá nhiều tiến trình mới cho mỗi yêu cầu.
- Một Servlet sẽ thường trực trong bộ nhớ giữa các yêu cầu, trong khi các chương trình CGI cần phải tải xuống và được khởi động cho từng yêu cầu CGI.
- Chỉ cần một Servlet trả lời đồng thời cho tất cả các yêu cầu. Điều này cho phép tiết kiệm được bộ nhớ và đảm bảo nó dễ dàng quản lý được dữ liệu một cách thống nhất.
- Một Servlet có thể thực hiện bởi một Servlet Engine trong phạm vi kiểm soát Sandbox để đảm bảo an toàn trong việc sử dụng các Servlet.

Các lớp Servlet của Java có thể được nạp tự động để mở rộng các chức năng của Server. Các Servlet của Java thực hiện bên trong JVM. Chúng được đảm bảo an toàn và chuyên đổi tương thích giữa các hệ điều hành và giữa các Server với nhau. Điều này khác với các Applet, Servlet chỉ thao tác được trong miền của một Server.

Servlet API được phát triển dựa trên những điểm mạnh của Java platform nhằm giải quyết vấn đề tồn tại của CGI và Server API. Nó là một API đơn giản, hỗ trợ tất cả các Web server và thậm chí cho phép các ứng dụng máy chủ dùng để kiểm tra và quản lý các công việc trên Server. Nó giải quyết vấn đề thực thi bằng việc thực hiện tất cả các yêu cầu như các luồng Thread trong quá trình xử lý, hoặc việc cân bằng tải trên một Server trong các cụm máy tính Cluster. Các Servlet dễ dàng chia sẻ tài nguyên với nhau.

Trong định nghĩa Servlet, vấn đề bảo mật được cải tiến theo nhiều cách. Trước hết, bạn hiếm khi thực thi được các câu lệnh trên Shell với dữ liệu cung cấp bởi người dùng mà Java API đã cung cấp với những khả năng truy cập đến tất cả các hàm thông dụng. Bạn có thể sử dụng Java Mail để đọc và gửi mail, kết nối vào các CSDL (qua JDBC), tệp lớp (class) và những lớp liên quan để truy cập hệ thống tệp, CSDL, RMI, CORBA, Enterprise JavaBean (EJB), ... Vấn đề bảo mật thông tin sẽ được đề cập chi tiết ở chương 7.

## 6.2. ƯU ĐIỂM CỦA SERVLET

Servlet được sử dụng để thay thế cho những công nghệ Web động. Việc sử dụng Servlet mang lại những lợi thế:

- *Dễ di chuyển.* Servlet được viết bằng Java nên nó có tính di động cao, thực hiện được trên nhiều hệ điều hành, trên các Web Server khác nhau. Khái niệm “Viết một lần, chạy ở mọi nơi” cũng rất đúng với Servlet.
- *Mạnh mẽ.* Servlet hỗ trợ rất hiệu quả cho việc sử dụng các giao diện lõi API như lập trình mạng, xử lý đa luồng, xử lý ảnh, nén dữ liệu, kết nối các CSDL, bảo mật, xử lý phân tán và triệu gọi từ xa RMI, CORBA, v.v. Nó cũng thích hợp để trao đổi tin, truyền thông giữa Client và Server một cách bình thường.
- *Hiệu quả.* Servlet có tính hiệu quả cao. Một khi được tải về, nó sẽ được lưu lại trong bộ nhớ của máy chủ. Servlet duy trì các trạng thái của nó, do vậy những tài nguyên ngoại như việc kết nối với CSDL cũng sẽ được lưu giữ lại.
- *An toàn.* Bởi vì Servlet được viết bằng Java nên nó kế thừa được tính an toàn của Java. Cơ chế tự động dọn rác và việc không sử dụng con trỏ số học của Java giúp cho Servlet thoát khỏi nhiều công việc quản lý bộ nhớ phức tạp. Đồng thời nó xử lý các lỗi rất an toàn theo cơ chế xử lý ngoại lệ của Java.
- *Tích hợp.* Các Servlet được tích hợp với các Server. Chúng cộng tác với các Server tốt hơn các chương trình CGI.
- *Tinh linh hoạt.* Các Servlet hoàn toàn mềm dẻo. Một HTTP Servlet được sử dụng để tạo ra một trang Web, sau đó ta có thể sử dụng thẻ <Servlet> để đưa nó vào trang Web tĩnh, hoặc sử dụng với các Servlet khác để lọc ra các nội dung cần thiết.

### 6.3. MÔI TRƯỜNG THỰC HIỆN SERVLET

Các Servlet thường là sự mở rộng (kế thừa) các lớp chuẩn của Java trong gói javax.servlet (chứa các khuôn mẫu cơ bản của Servlet) và javax.servlet.http (mở rộng các khuôn mẫu cơ bản của Servlet và các yêu cầu theo HTTP).

Servlet là một lớp Java và vì thế cần được thực thi trên một máy ảo Java (JVM) bằng một dịch vụ được gọi là mô hình Servlet (Servlet Engine). Servlet Engine tải lớp Servlet lần đầu tiên nó được yêu cầu, hoặc ngay khi Servlet Engine được bắt đầu. Servlet ngừng tài để xử lý nhiều yêu cầu khi Servlet Engine bị dừng lại.

Như vậy, để dịch và thực hiện các Servlet, việc có các Servlet là chưa đủ, mà cần phải có một mô hình Servlet để kiểm tra và triển khai chúng. Hiện nay có một số mô hình tương thích với nhiều loại Web Server khác nhau, nhưng nguyên lý hành động tương đối giống nhau. Người ta chia chúng thành ba loại.

- Mô hình Servlet đơn
- Mô hình Servlet gộp
- Mô hình Servlet nhúng.

### 6.3.1. Mô tơ Servlet đơn

Đây là loại Server được xây dựng để hỗ trợ cho các Servlet. Ưu điểm của nó là mọi thứ làm việc với các hộp kết quả đầu ra rất phong phú. Tuy nhiên, nó có nhược điểm là ta phải chờ những phiên bản mới của Web Server để nhận được những hỗ trợ mới nhất cho Servlet. Hiện nay có các loại mô tơ đơn sau:

- *Java Server Web Development (JSWDK) của Sun Microsystem*: được viết hoàn toàn bằng Java: [http://java.sun.com.products/servlet/](http://java.sun.com/products/servlet/). Nó được sử dụng như là một Server độc lập để kiểm tra các Servlet và các trang JSP trước khi phát triển thành một Web Server thực sự.
- *Jigsaw Server của WWW Consortium*, cũng được viết bằng Java. Chi tiết hơn, xem <http://www.w3.org/Jigsaw>.
- *Netscape Enterprise Server*. Đây là Web Server rất nổi tiếng, nó hỗ trợ để xây dựng Servlet.
- *Lotus Domino Go WebServer*. Một loại Web Server khác cũng hỗ trợ để xây dựng Servlet.

### 6.3.2. Mô tơ Servlet gộp

Servlet gộp là loại mô tơ được viết cho nhiều loại Server khác nhau, kể cả Apache, Fast Track Server, Enterprise Server của Netscape, Personal Web Server, v.v. Hiện nay có các loại sau:

- *Apache Tomcat*: Mô tơ này hỗ trợ thêm cho Apache. Nó được sử dụng như là một Server độc lập để kiểm tra các Servlet và các trang JSP, hoặc được tích hợp vào Apache Web Server, [http://java.sun.com.products/servlet/](http://java.sun.com/products/servlet/).
- *Jrun của Live Software*: Jrun là mô tơ cho Servlet và JSP, hỗ trợ đầy đủ Servlet API trong các Web Server phổ biến trên mọi hệ điều hành, <http://www.allaire.com.products/jrun/>.
- *WebSphere Application Server của IBM*: còn được gọi là *ServletExpress*.
- *ServletExec của New Atlanta*: ServletExec là mô tơ cho Servlet và JSP, hỗ trợ đầy đủ Servlet API trong các Web Server phổ biến trên mọi hệ điều hành, <http://newatlanta.com/>.

### 6.3.3. Mô tơ Servlet nhúng

Loại mô tơ này có thể nhúng vào phần mềm ứng dụng khác. Hiện nay có các loại sau:

- *Java Server Engine của Sun*. Đây là loại mô tơ được viết hoàn toàn bằng Java và là Web Server đầu tiên hỗ trợ đầy đủ cho các đặc tả của Servlet 2.1 và JSP 1.0, <http://www.sun.com/software/jwebserver/try>.

- Nexus Web Server của Anders Kristensen. Nó có thể dễ dàng nhúng vào các chương trình ứng dụng Java.

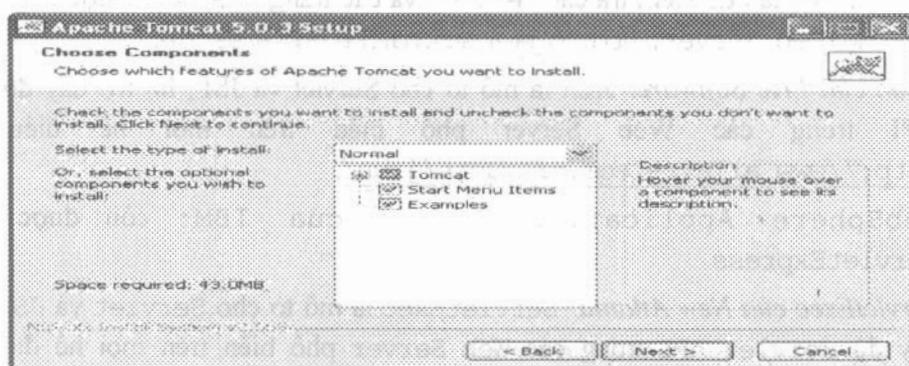
### 6.3.4. Cài đặt trình chủ Apache Tomcat

Để thực hiện được Servlet ta cần cài đặt một trong các mô tơ Servlet đã nêu. Một trong số đó được sử dụng khá phổ biến là Tomcat. Tomcat, tên một dự án con trong dự án Jakarta được gọi là một Servlet Container, tức là một mô tơ thực thi và triển khai công nghệ Servlet của Sun. Tất nhiên, Tomcat còn chứa một môi trường triển khai cho cả công nghệ JSP. Nói cách khác, Tomcat chính là chương trình dịch và thực thi các tệp .class và .jsp vốn được viết bằng ngôn ngữ Java để tạo ra các trang Web động.

Tomcat có thể xem như là một Web Server đơn giản với các tính năng cơ bản giống như Apache. Nhưng, trong khi Apache không hiểu gì về các trang Web động của công nghệ Java thì Tomcat lại là một phần mềm được xây dựng để chuyên xử lý các chương trình Web làm bằng công nghệ này.

Đĩa cài Tomcat có thể có sẵn ở các hiệu đĩa CD, nhưng để có phiên bản mới nhất hỗ trợ đặc tả Servlet và JSP hiện đại nhất và nhiều tính năng nhất thì tốt nhất là tải trực tiếp trên trang chủ về phiên bản Tomcat 5 (kích thước khoảng 4.5MB). Thông tin trực tuyến về Tomcat có thể tìm tại <http://jakarta.apache.org/tomcat/>.

Hiện tại, Tomcat đã có tới phiên bản 5.5.9 (jakarta-tomcat-5.5.9.exe). Trong tài liệu này chúng ta sẽ dùng phiên bản Tomcat 5.5.9 để cài đặt chương trình ứng dụng. Khi Download bản cài đặt về và chạy chương trình sẽ xuất hiện giao diện sau:



Hình 6.2. Màn hình giao diện cài đặt Tomcat

Ở đây, chúng ta phải lựa chọn các thành phần cài đặt cho Tomcat. Nếu mới làm quen với Tomcat thì hãy sử dụng tùy chọn mặc định của Tomcat bằng cách kích vào Next. Thường thì không nên cài đặt Tomcat với một đường dẫn có khoảng trống. Chúng ta hãy lựa chọn một thư mục cài đặt Tomcat khác với mặc định, ví dụ: C:\Tomcat5.

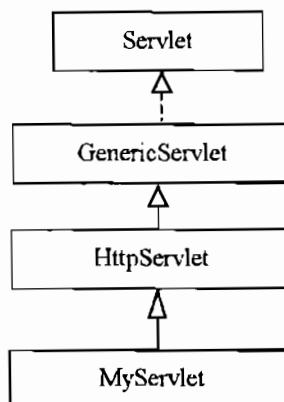
Cũng cần phải chú ý thêm: cổng mặc định của Tomcat là 8080, và ta có thể thay đổi được. Số cổng này trùng với số cổng của Apache 2.0.x nếu như ta đã cài Apache 2 và cho

nó chạy dưới hình thức Manually, tức là khởi động thủ công thay vì cho nó tự động hành động như là một dịch vụ (Service). Khi đó, sẽ có sự xung đột về cổng giữa hai ứng dụng này. Vì thế, ta phải thay đổi số cổng của một trong hai ứng dụng. Ví dụ, với Tomcat ta vẫn để là 8080 nhưng Apache 2 thì chuyển thành 8081, hoặc ngược lại.

Để kiểm tra quá trình cài đặt có thành công hay không, sau khi kích đúp vào tomcat\_install\_dir\tomcat5.5\bin\tomcat5.exe để khởi động Server, hay có thể truy cập vào địa chỉ <http://localhost:8080/>, nếu có một trang Web hiện ra thì quá trình cài đặt Tomcat hoàn tất, ngược lại thì phải xem lại các bước đã thực hiện.

## 6.4. KIẾN TRÚC CỦA SERVLET

Gói javax.servlet cung cấp các giao diện và các lớp để xây dựng các Servlet. Kiến trúc của chúng được mô tả như sau.



Hình 6.3. Kiến trúc của Servlet

### 6.4.1. Giao diện Servlet

Giao diện Servlet là một khái niệm trung tâm trong Servlet API. Tất cả các Servlet đều cài đặt trực tiếp hoặc gián tiếp giao diện này hoặc mở rộng (kế thừa) những lớp đã cài đặt nó.

Giao diện này khai báo ba phương thức định nghĩa vòng đời của Servlet.

- public void init(ServletConfig config) throws ServletException

Fương thức này được gọi một lần khi Servlet được tải vào trong Servlet Engine, trước khi Servlet được yêu cầu để xử lý một yêu cầu nào đó. Phương thức `init()` có một thuộc tính là đối tượng của `ServletConfig`, và Servlet có thể đọc các đối số khởi tạo của nó thông qua đối tượng `ServletConfig`. Chúng thường được định nghĩa trong một tệp cấu hình. Một ví dụ thông thường của một đối số khởi tạo là định danh database cho một CSDL.

```

    ...
    private String databaseURL;
    public void init(ServletConfig config) throws
        ServletException {
        super.init(config);
        databaseURL = config.getInitParameter("database");
    }

```

- `public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException`

Phương thức này được gọi để xử lý các yêu cầu. Nó có thể không được gọi, gọi một lần hay nhiều lần cho đến khi Servlet được ngưng tải. Nhiều Thread (mỗi Thread cho một yêu cầu) có thể thực thi phương thức này song song, vì thế nó trở nên an toàn và hiệu quả hơn.

- `public void destroy()`

Phương thức này chỉ được gọi một lần trước khi Servlet được ngưng tải và sau khi đã kết thúc các dịch vụ.

Servlet API có cấu trúc để Servlet có thể cho phép bổ sung một giao thức khác với HTTP. Gói `javax.servlet` chứa các lớp và các giao diện được kế thừa giao diện Servlet một cách độc lập. Gói `javax.servlet.http` chứa các lớp và giao diện HTTP cụ thể.

#### 6.4.2. Lớp cơ sở HttpServlet

Như ta đã biết, theo giao thức HTTP, dữ liệu được trao đổi giữa máy chủ Server và các máy Client theo một trong hai phương thức GET hay POST. Java định nghĩa một lớp có tên là `HttpServlet` ở trong gói `javax.servlet` để truyền và nhận dữ liệu theo cả hai phương thức trên.

Lớp trừu tượng `HttpServlet` cung cấp một khung làm việc để xử lý các yêu cầu GET, POST của giao thức HTTP. `HTTPServlet` kế thừa giao diện `Servlet` cộng với một số các phương thức hữu dụng khác.

Một tập các phương thức trong `HTTPServlet` là những phương thức xác định dịch vụ trong giao diện `Servlet`. Việc bổ sung dịch vụ trong `HTTPServlet` giống như một kiểu của các yêu cầu được xử lý (GET, POST, HEAD, ...) và gọi một phương thức cụ thể cho mỗi kiểu. Bằng việc làm này, các nhà phát triển `Servlet` sẽ an tâm khi xử lý chi tiết những yêu cầu như HEAD, TRACE, OPTIONS, ... và có thể tập trung vào những yêu cầu thông dụng hơn như GET và POST.

HTTP sinh ra các trang HTML và ta có thể nhúng các `Servlet` vào một trang HTML.

Khi có một yêu cầu được gửi tới, đầu tiên nó phải chỉ ra lệnh cho HTTP bằng cách gọi một phương thức tương ứng. Phương thức này chỉ cho Server biết kiểu hành động mà nó muốn thực hiện.

Khi có một Client gửi tới một yêu cầu, Server sẽ xử lý yêu cầu nhận được và gửi trả lại kết quả cho Client. Hai phương thức `doGet()` và `doPost()` được sử dụng chung để nhận và gửi tin trong các Servlet.

Một Servlet bất kỳ, ví dụ `MyServlet` phải kế thừa `HttpServlet` và viết đè ít nhất một trong các phương thức `doGet()` để thực thi thao tác GET của HTTP, hay `doPost()` để thực thi thao tác POST của HTTP.

Trong ví dụ đầu tiên, chúng ta sẽ viết đè phương thức `doGet()` dạng

```
protected void doGet(HttpServletRequest request,
                     HttpServletResponse response) throws ServletException,
                     IOException {
    ...
}
```

Phương thức `doGet()` có hai tham số đối tượng thuộc hai lớp `HttpServletRequest` và `HttpServletResponse` (cả hai lớp này được định nghĩa trong `javax.servlet.http`). Hai đối tượng này cho phép chúng ta truy cập đầy đủ tất cả các thông tin yêu cầu và cho phép gửi dữ liệu kết quả cho Client để trả lời cho yêu cầu đó.

Với CGI, các biến môi trường và `stdin` được sử dụng để nhận thông tin về yêu cầu, tuy nhiên việc đặt tên các biến môi trường có thể khác nhau giữa các chương trình CGI, và một vài biến có thể không được cung cấp bởi tất cả các Web Server.

Đối tượng của `HttpServletRequest` cung cấp thông tin giống như biến môi trường của CGI nhưng theo một hướng chuẩn hóa. Nó cũng cung cấp những phương thức để mở ra các tham số HTTP từ dãy các câu truy vấn hoặc từ nội dung của yêu cầu phụ thuộc vào kiểu yêu cầu (GET hay POST).

**Ví dụ 6.1.** Chương trình `FirstServlet` để đếm và hiển thị số lần nó được truy cập.

```
package myservlet;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet{
    int i = 0;
    public void doGet(HttpServletRequest re, HttpServletResponse resp)
        throws ServletException, IOException{
        resp.setContentType("text/html");
    }
}
```

```

        PrintWriter out = resp.getWriter();
        i++;
        out.println("So lan duoc truy cap: " + i);
        out.close();
    }
}

```

doGet(HttpServletRequest req, HttpServletResponse resp) sử dụng đối số thứ nhất req để đọc các phần đầu, tiêu đề (header) của HTTP gửi tới (ví dụ như dữ liệu dạng HTML mà người dùng nhập vào), và sử dụng resp để xác định dòng trả lời cho HTTP (xác định kiểu nội dung trao đổi, đặt Cookie). Điều quan trọng nhất là phải nhận được một đối tượng của PrintWriter (ở trong gói java.io) thông qua resp.getWriter() để gửi kết quả trả lại cho Client. Ngoài doGet() và doPost(), HttpServlet còn có các phương thức:

- service(): thực hiện khi đối tượng của lớp được tạo lập và triệu gọi doGet() hoặc doPost().
- doPut(): thực hiện thao tác PUT của HTTP.
- doDelete(): thực hiện thao tác DELETE của HTTP.
- init() và destroy(): khởi tạo và huỷ bỏ các Servlet.
- getServletInfo(): nhận các thông tin về Servlet.

**Lưu ý:** Cả doGet() và doPost() đều có thể phát sinh ra một trong hai ngoại lệ ServletException, hay IOException, do vậy ta phải khai báo chúng như trên. Ngoài ra, một điều nữa chúng ta cũng chú ý là hai phương thức doGet() và doPost() sẽ được phương thức service() gọi để thực hiện và đôi lúc ta có thể viết đè service() thay cho hai phương thức trên.

## 6.5. DỊCH VỤ CÀI ĐẶT SERVLET

Trước tiên phải lưu ý rằng việc cài đặt chi tiết các Servlet là thay đổi tùy theo từng Web Server khác nhau. Ví dụ, khi ta sử dụng Java Web Server (JWS) 2.0 thì các Servlet đòi hỏi phải được đặt ở thư mục được gọi là servlets trong cây thư mục cài đặt của JWS. Tuy nhiên, ta có thể tạo ra một thư mục riêng, ví dụ myservlet và đặt các Servlet vào đó để tránh xung đột với các Servlet khác.

### 6.5.1. Dịch chương trình Servlet

Dịch các Servlet có thể sử dụng JDK <http://www.javasoft.com> hoặc các chương trình biên dịch Java khác và thực hiện hoàn toàn giống như đối với các Applet hay

chương trình ứng dụng độc lập. Nếu các gói javax.servlet và javax.servlet.http không có trong chương trình biên dịch và cũng không có trong Servlet Engine thì phải cài đặt chúng riêng bằng cách nạp từ JSDK (Java Servlet Development Kit) và đặt đường dẫn chứa các lớp của JSDK trong biến môi trường CLASSPATH cho phù hợp.

Như vậy, để chạy được các chương trình Servlet như trên, ta có thể thực hiện các bước như sau:

- Cài đặt JSDK (Java Servlet Development Kits)
- Cài đặt một trong các mô hình Servlet như đã nêu ở mục 6.3
- Dịch chương trình Servlet.

Có hai cách chính để dịch các chương trình Servlet trong các gói.

- + *Cách thứ nhất là đặt CLASSPATH để chỉ tới thư mục mà ở đó có chứa các Servlet của chúng ta.* Ví dụ, JSDK cài đặt ở c:\jsdk2.0, và gói của ta có tên là myservlet (thư mục d:\myservlet) chứa các Servlet cần dịch, ở Window ta thực hiện như sau:

```
DOS> set CLASSPATH=.;C:\jsdk2.0\lib\jsdk.jar;%CLASSPATH%
DOS> cd d:\myservlet
DOS> javac FirstServlet.java
```

- + *Cách thứ hai là dịch các lớp được đặt trong các gói được chỉ rõ trong câu lệnh dịch.* Ví dụ, nếu thư mục cơ sở cài đặt JSDK là c:\jsdk2.0 và gói của ta có tên là myservlet (thư mục d:\myservlet) chứa các Servlet cần dịch, ở Window ta thực hiện như sau:

```
DOS> set CLASSPATH =.;C:\jsdk2.0\lib\jsdk.jar;%CLASSPATH%
DOS> javac myservlet\FirstServlet.java
```

Hiển nhiên là trước đó ta phải đặt biến môi trường PATH chỉ tới nơi cài đặt chương trình dịch javac.exe. Giả sử J2DK1.4.2 được cài đặt ở c:\j2sdk1.4.2, thì biến môi trường PATH được đặt như sau

```
DOS> set PATH=.;C:\j2sdk1.4.2\bin
```

Để tiện lợi cho việc dịch các Servlet, ta có thể đặt các biến môi trường như trên vào trong autoexec.bat.

### 6.5.2. Thực hiện Servlet

1. Với Java Web Server, các Servlet được đặt ở thư mục servlets bên trong cây thư mục cài đặt JWS và để chạy, chúng ta triệu gọi <http://host/servlet/ServletName> sau khi đã khởi động Server.

**Chú ý:** Thư mục servlets là số nhiều còn địa chỉ URL gọi tới là servlet, số ít. Nếu Servlet của chúng ta là FirstServlet đặt ở gói myservlet thì phải triệu gọi:

<http://host/servlet/myservlet/FirstServlet>

Những Web Server khác có thể quy định cách cài đặt Servlet khác nhau. Phần lớn chúng cho phép sử dụng cách đặt biệt danh (alias) cho các Servlet sao cho có thể triệu gọi một tệp HTML bất kỳ (any-file) ở một thư mục bất kỳ (any-path):

<http://host/any-path/any-file.html>

2. Với Tomcat 5.5, chúng ta cũng thực hiện tương tự như trên.

- Đặt các chương trình Java Servlet vào một thư mục, ví dụ  
c:\Servlets + JSP
- Khởi động DOS và viết “javac FirstServlet”
- Đặt tệp lớp FirstServlet.class vào thư mục của Servlet, ví dụ  
C:\Tomcat5.5\webapps\ROOT\WEB-INF\classes
- Khởi động Server: kích đúp vào startup.bat hoặc biểu tượng tomcat5.exe.
- Triệu gọi Servlet

<http://localhost/servlet/FirstServlet>

Hoặc

<http://localhost:8080/FirstServlet>

nếu Tomcat được cài đặt ở cổng 8080.

**Ví dụ 6.2.** Xây dựng một Servlet đơn giản để sinh ra HTML và hiển thị lời chào “Xin chào các bạn” mỗi khi thực hiện.

Phần lớn các Servlet sinh ra các kết quả dạng HTML. Để làm được điều này, chúng ta cần thực hiện theo hai bước:

- Chỉ cho trình duyệt biết là ta sẽ gửi thông tin lại bằng HTML. Điều này được thực hiện bằng cách đặt kiểu nội dung là

```
res.setContentType("text/html");
```

- Thay đổi lệnh println() để tạo ra một trang Web hợp lệ với các đối số là các thẻ HTML.

```
package myservlet;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletExample extends HttpServlet{
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException{
        res.setContentType("text/html");
    }
}
```

```

        PrintWriter out = res.getWriter();
        out.println("<!DOCTYPE HTML PUBLIC "-//W3C//DTD
                     HTML "Transitional//EN\">\n" +
                    "<HTML>\n" +
                    "<HEAD><TITLE>Mời chào</TITLE></HEAD>\n" +
                    "<BODY>\n" +
                    "<H1> Xin chào các bạn</H1>\n" +
                    "</BODY></HTML>");

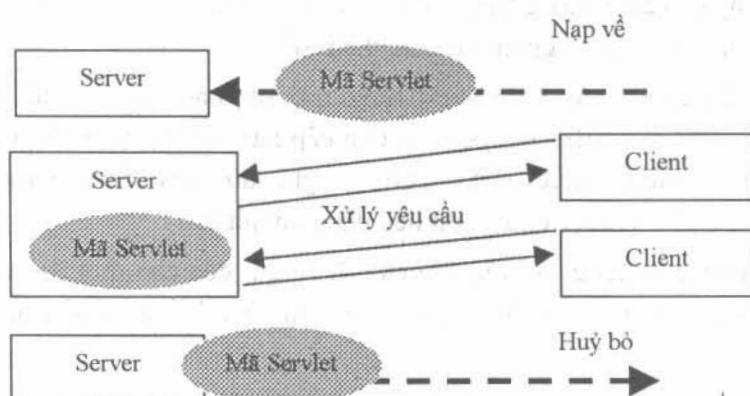
    }
}

```

## 6.6. CHU TRÌNH SỐNG CỦA CÁC SERVLET

Mỗi Servlet đều có chu trình sống như sau:

- Server nạp và khởi tạo Servlet
- Servlet xử lý các yêu cầu của các Client
- Server huỷ bỏ Servlet khi không còn cần thiết.



Hình 6.4. Chu trình sống của Servlet

Như trên đã phân tích, mọi Servlet đều có một chu trình sống nhất định. Nó được khởi tạo (nạp về), xử lý các yêu cầu của Client và sau khi hoàn tất các dịch vụ thì bị Server huỷ bỏ (Hình 6.4).

### 6.6.1. Khởi động Servlet

Việc khởi động một Servlet được thực hiện mặc định bởi HttpServlet. Để khởi động một Servlet riêng, ta phải viết đè phương thức init().

```

public class MyServlet ...{
    public void init() throws ServletException{
        // Khởi tạo các giá trị ban đầu
    }
    ...
}

```

Khi một Server nạp xong một Servlet thì nó gọi `init()` để bắt đầu Servlet đó.

**Lưu ý:** Server chỉ gọi `init()` một lần khi nạp Servlet và sau đó sẽ không gọi lại nữa, trừ khi phải nạp lại nó. Server không thể nạp lại nếu Servlet đó chưa bị huỷ bỏ bởi lời gọi `destroy()`. Phương thức `init()` có thể được nạp chòng theo mẫu

public void init(ServletConfig config) throws ServletException  
như đã nêu ở trên.

### 6.6.2. Tương tác với các Client

Lớp `HttpServlet` xử lý các yêu cầu của Client thông qua các dịch vụ của nó. Các phương thức trong `HttpServlet` xử lý yêu cầu của Client đều có hai đối số:

- Một là đối tượng của `HttpServletRequest`, bao gồm cả dữ liệu từ Client. Nó cho phép nhận được các tham số mà Client gửi đến như một phần của các yêu cầu, thông qua các phương thức `getParameterName()`, `getParameterValue()` để xác định tên gọi và giá trị của các tham số.
- Hai là đối tượng của `HttpServletResponse`, bao gồm cả dữ liệu hồi đáp cho Client. `HttpServletResponse` cung cấp hai phương thức để trả lại kết quả cho Client. Phương thức `getWriter()` ghi dữ liệu dưới dạng văn bản còn `getOutputStream()` cho lại dữ liệu dạng nhị phân.

Ngoài ra, để xử lý được các yêu cầu của HTTP gửi đến cho một Servlet thì phải mở rộng lớp `HttpServlet` và viết đè các phương thức xử lý các yêu cầu như `doGet()`, `doPost()`.

### 6.6.3. Hủy bỏ Servlet

Sau khi nạp các Servlet và xử lý chúng xong thì cần phải dọn dẹp hệ thống, loại bỏ những Servlet không còn được sử dụng tiếp. Phương thức `destroy()` của lớp `HttpServlet` được sử dụng để loại bỏ một đối tượng Servlet khi nó không còn cần thiết. Để loại bỏ một tài nguyên nào đó cụ thể trong một Servlet riêng thì phải viết đè phương thức `destroy()`.

Server sẽ gọi `destroy()` để huỷ một Servlet, sau khi nó kết thúc tất cả các dịch vụ theo yêu cầu. Ví dụ sau đây mô tả một Servlet mở một kết nối với CSDL thông qua phương thức `init()` và sau đó nó sẽ bị phá huỷ bởi phương thức `destroy()`.

```

public class DBServlet extends HttpServlet{
    Connection connection = null;
    public void init() throws ServletException{
        // Mở một kết nối với CSDL
        try{
            databaseUrl = getInitParameter("databaseUrl");
            // Đọc tên người sử dụng và mật khẩu
            connection = DriverManager.getConnection(
                userName, password);
        }catch(Exception e){
            throw new UnavailableException(this,
                "Không mở được CSDL");
        }
    }
    // ...
    public void destroy(){
        // Đóng các kết nối để dọn dẹp bộ nhớ
        connection.close();
        connection = null;
    }
}

```

Server sẽ gọi `destroy()` sau khi tất cả các dịch vụ đã được kết thúc.

## 6.7. XỬ LÝ CÁC YÊU CẦU

Nhiệm vụ của các Servlet là nhận các yêu cầu, xử lý chúng và gửi kết quả trả lời cho các Client.

### 6.7.1. Truy tìm thông tin

Để xây dựng thành công một WebSite, ta cần có những thông tin từ Server, nơi thực hiện các Servlet. Servlet có những phương thức sau giúp cho việc nhận được những thông tin yêu cầu.

- `int port = req.getServerPort();`: Xác định cổng trao đổi tin của máy chủ.
- `public String ServletConfig.getInitParameter(String name)`: Cho lại giá trị ban đầu của tham số có tên name.

- `public Enumeration ServletConfig.getInitParameterNames():` Xác định dãy liệt kê tên gọi của tất cả các tham số khởi đầu của Servlet như là đối tượng của Enumeration. Lớp GenericServlet sử dụng phương thức này để truy cập đến các Servlet.

**Ví dụ 6.3.** Servlet sau in ra tên và giá trị ban đầu của tất cả các tham số.

```
import java.io.*;
import javax.servlet.*;
import java.util.*;
public class ReadServlet extends GenericServlet{

    public void service(ServletRequest re, ServletResponse resp)
        throws IOException{
        resp.setContentType("text/plain");
        PrintWriter out = resp.getWriter();
        out.println("Tham so khai dau la:");
        Enumeration eno = getInitParameterNames();
        while(eno.hasMoreElements()){
            String nm = (String)eno.nextElement();
            out.println(nm + " : " + getInitParameter(nm));
        }
    }
}
```

### 6.7.2. Gửi thông tin

Các Servlet cần phải trả lời cho Client về các thông tin dưới dạng HTML bao gồm:

- *Các mã hiện trạng.* Mã hiện trạng là một số nguyên, nó mô tả tình trạng của việc hồi đáp thành công hay thất bại. Phần lớn các mã này đã được định nghĩa trong lớp `HttpServletResponse` như ở bảng 6.1.
- *Các phần đầu (tiêu đề) của HTTP.* Phương thức `setHeader()` của lớp `HttpServletResponse` cho phép đặt lại các giá trị cho các tiêu đề.
- *Nội dung hồi đáp.* Nội dung hồi đáp dưới dạng một trang HTML.

Bảng 6.1

Tên gọi nhứ	Mã số	Thông báo
SC_OK	200	Ok (tốt)
SC_NO_CONTENT	204	Không có nội dung
SC_MOVED_PERMANENTLY	301	Đã chuyển di vĩnh viễn
SC_MOVED_TEMPORARILY	302	Đã chuyển di tạm thời
SC_NOT_FOUND	404	Không tìm thấy
SC_UNAUTHORIZED	401	Không được phép

**Ví dụ 6.4.** Servlet gửi cho Client một trang ngẫu nhiên trong danh sách các trang Web của nó.

```
// RandomSend.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
public class RandomSend extends HttpServlet{
    Vector st = new Vector();
    Random rd = new Random();
    public void init(ServletConfig config) throws ServletException{
        super.init(config);
        st.addElement("http://www.yahoo.com");
        st.addElement("http://www.hotmail.com");
        st.addElement("http://www.zednet.com");
        st.addElement("http://www.java.sun.com");
    }
    public void doGet(HttpServletRequest re, HttpServletResponse res)
        throws IOException{
        res.setContentType("text/html");
        PrintWriter out = resp.getWriter();
        int siteIndex = Math.abs(rd.nextInt()) % st.size();
        String st = (String) st.elementAt(siteIndex);
        res.setStatus(res.SC_MOVED_TEMPORARILY);
        res.setHeader("Location", st);
    }
}
```

Khi một Client dùng HTML để gửi đi một yêu cầu thì nó có thể sẽ gửi đi một số các tiêu đề (Header).

Việc đọc các tiêu đề là tương đối đơn giản, có thể sử dụng hàm `getHeader()` của `HttpServletRequest` để nhận được các tiêu đề (header) yêu cầu. Sau đây là các tiêu đề phổ dụng trong các yêu cầu.

- **Accept:** Các kiểu tệp được trình duyệt chấp nhận, như các tệp `image/gif`, `image/jpeg`, hay `application/msword`, v.v.
- **Accept-Charset:** Tập các ký tự mà trình duyệt mong đợi.
- **Accept-Encoding:** Các kiểu mã hóa dữ liệu (ví dụ `gzip`) mà trình duyệt biết cách giải mã.
- **Accept-Language:** Ngôn ngữ mà trình duyệt chấp nhận.
- **Authorization:** Thông tin về giấy phép, ủy quyền.
- **Connection:** Khẳng định có kết nối thường xuyên hay không? Nếu một Servlet đặt là `Keep-Alive` thì nó có thể tận dụng được những ưu thế của cơ chế kết nối thường xuyên.
- **Content-Length:** Số dữ liệu được đưa vào yêu cầu.
- **Cookie:** Một trong các tiêu đề quan trọng nhất. Một Cookie là một xâu (dãy ký tự) được gửi tới cho một Client để bắt đầu một phiên làm việc.
- **Host:** Máy chủ và cổng được chỉ ra trong URL.
- **User-Agent:** Loại trình duyệt.

Các tiêu đề trên là tùy chọn, trừ `Content-Length` là được yêu cầu chỉ đối với yêu cầu POST.

Để tìm một tiêu đề cụ thể, trước tiên ta cần sử dụng `getHeaderNames()` để có được một dãy (đối tượng của `Enumeration`) tất cả các tiêu đề nhận được từ một yêu cầu cụ thể, sau đó tìm lần lượt tiêu đề đó.

Một vấn đề nữa cần biết khi phải tìm các tiêu đề của một yêu cầu là phải biết thông tin về phương thức chính được sử dụng. Phương thức `getMethod()` sẽ cho ta biết về phương thức chính của mỗi yêu cầu (thường là GET, POST, hoặc cũng có thể là HEAD, PUT và DELETE). Phương thức `getRequestURI()` sẽ cho lại URI (một phần của URL, phần đứng sau tên của địa chỉ máy chủ với cổng kết nối và trước phần dữ liệu mầu).

**Ví dụ 6.5.** Xây dựng một Servlet để đọc, hiển thị một bảng các tiêu đề và nội dung của một yêu cầu.

```
// ShowRequestHeaders.java
package myservlet;
import java.io.*;
import javax.servlet.*;
```

```
import javax.servlet.http.*;
import java.util.*;

/* Chỉ ra tất cả các tiêu đề của một yêu cầu cụ thể dưới dạng một bảng bao gồm tiêu đề
và nội dung
*/
public class ShowRequestHeaders extends HttpServlet {
    public void doGet(HttpServletRequest request,
                       HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Servlet Example: Showing Request Headers";
        out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML
4.0 Transitional//EN\">" + "<HTML>\n"
+ "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" + "<BODY BGCOLOR=\"#FDF5E6\">\n" +
"<H1 ALIGN=CENTER>" + title + "</H1>\n" +
"<B>Request Method: </B>" +
request.getMethod() + "<BR>\n" +
"<B>Request URI: </B>" +
request.getRequestURI() + "<BR>\n" +
"<B>Request Protocol: </B>" +
request.getProtocol() + "<BR><BR>\n" +
"<TABLE BORDER=1 ALIGN= CENTER>\n" +
"<TR BGCOLOR=\"#FFAD00\">\n" +
"<TH>Header Name<TH>Header Value");
        Enumeration headerNames = request.getHeaderNames();
        while(headerNames.hasMoreElements()) {
            String headerName = (String)headerNames.nextElement();
            out.println("<TR><TD>" + headerName);
            out.println("      <TD>" + request.getHeader(headerName));
        }
        out.println("</TABLE>\n</BODY></HTML>");
    }
}
```

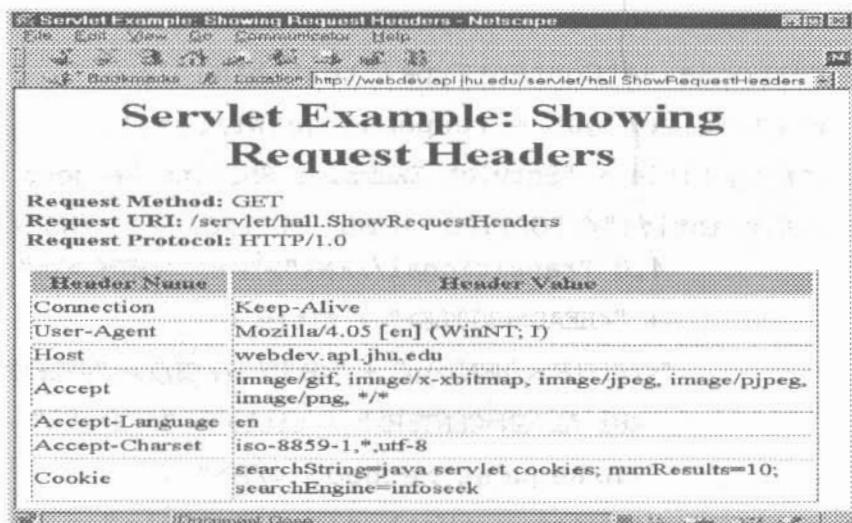
```

    }

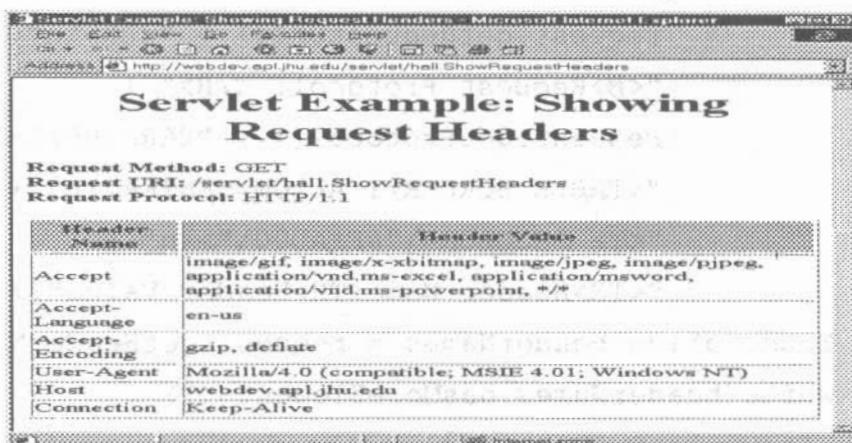
    public void doPost(HttpServletRequest request,
                        HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request, response);
    }
}
}

```

Sau đây là kết quả hiển thị đối với hai loại trình duyệt Netscape và Internet Explore.



Hình 6.5. Hiển thị các tiêu đề yêu cầu với Netscape



Hình 6.6. Hiển thị các tiêu đề yêu cầu với IE

### 6.7.3. Xử lý các dữ liệu mẫu

Khi sử dụng máy tìm kiếm Web Search Engine, bạn có thể truy cập dạng

http://host/path?user=Marty+Hall&origin=bwi&dest=lax

Phản tin sau dấu ‘?’ (`user=Marty+Hall&origin=bwi&dest=lax`) được xem như là *dữ liệu mẫu*, đây là cách chung thường được sử dụng để chương trình Server nhận dữ liệu vào từ trang Web.

Việc tách những thông tin cần thiết từ dữ liệu dạng trên là một phần việc mà các chương trình CGI thường làm.

- Trước tiên, đọc dữ liệu vào cho các tham số yêu cầu của GET hoặc POST
- Sau đó, chặt ra từng cặp ở dấu ‘&’ rồi lại tách tiếp để xác định tên của các tham biến (phần bên trái dấu ‘=’) và giá trị của những tham biến đó (phần bên phải của dấu ‘=’).
- Thực hiện giải mã URL theo những giá trị đó.

*Lưu ý:* Tất cả các chữ cái, chữ số được gửi đi là không thay đổi, nhưng dấu cách ‘ ’ được chuyển thành dấu ‘+’. Những ký tự khác được chuyển thành %XX, trong đó XX là giá trị (hex) của ký tự ASCII (khác với chữ cái, chữ số). Ví dụ, nếu bạn muốn nhập vào dữ liệu mẫu “~hall, ~gates” vào trường văn bản có tên “user” ở dạng HTML thì dữ liệu gửi đi phải là “user=%7Ehall%2C+%7Egates”.

Những công việc nhảm chán trên khi lập trình với CGI đã được Java hỗ trợ để xử lý các dữ liệu mẫu một cách tự động. Đơn giản là bạn gọi `getParameter()` của `HttpServletRequest` để nhận được tên gọi của tham số như là một đối số. Dữ liệu mẫu được gửi đi trong GET và POST là giống hệt nhau. Khi một tham số có nhiều hơn một giá trị thì gọi `getParameterValues()` thay vì gọi `getParameter()`, kết quả trả lại là mảng các xâu. Tương tự, sử dụng `getParameterNames()` để xác định tập các tên gọi của các tham số, kết quả của nó là danh sách các tên gọi thuộc lớp `Enumeration`.

**Ví dụ 6.6.** Chương trình đọc và hiển thị các tham số được gửi đến cho Servlet qua GET hay POST.

```
// ShowParameters.java

package hall;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

/** Hiển thị tất cả các tham số được gửi đến cho Servlet qua GET hoặc POST. Các
 * tham số đặc biệt có thẻ * không có giá trị hoặc có nhiều giá trị.
 */
public class ShowParameters extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        
```

```

String title = "Reading All Request Parameters";
out.println("<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN"> +
"<BODY BGCOLOR=#FDF5E6>\n" +
"<H1 ALIGN=CENTER>" + title + "</H1>\n" +
"<TABLE BORDER=1 ALIGN=CENTER>\n" +
"<TR BGCOLOR=#FFAD00>\n" +
"<TH>Parameter Name<TH>Parameter Value(s)");
Enumeration paramNames = request.getParameterNames();
while(paramNames.hasMoreElements()) {
    String paramName = (String)paramNames.nextElement();
    out.println("<TR><TD>" + paramName + "\n<TD>");
    String[] paramValues = request.getParameterValues(paramName);
    if (paramValues.length == 1) {
        String paramValue = paramValues[0];
        if (paramValue.length() == 0)
            out.print("<I>No Value</I>");
        else
            out.print(paramValue);
    } else {
        out.println("<UL>");
        for(int i=0; i<paramValues.length; i++) {
            out.println("<LI>" + paramValues[i]);
        }
        out.println("</UL>");
    }
}
out.println("</TABLE>\n</BODY></HTML>");
}
public void doPost(HttpServletRequest request,
                    HttpServletResponse response)
throws ServletException, IOException {
    doGet(request, response);
}
}

```

Để chương trình Servlet trên nhận được các tham số thì cần phải có một trang HTML gửi chúng. Trang HTML (PostForm.html) sau đây sử dụng POST để gửi dữ liệu (theo các mẫu forms có các mục đầu vào), thể hiện các giá trị mà Servlet nhận được theo cả hai phương thức doGet () và doPost () .

Tập PostForm.html

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML>
<HEAD> -
    <TITLE>A Sample FORM using POST</TITLE>
</HEAD>
<BODY BGCOLOR = "#FDF5E6">
<H1 ALIGN = "CENTER">A Sample FORM using POST</H1>
<FORM ACTION = "/servlet/hall.ShowParameters"

```

```

METHOD = "POST">
Item Number:
<INPUT TYPE="TEXT" NAME="itemNum"><BR>
Quantity:
<INPUT TYPE="TEXT" NAME="quantity"><BR>
Price Each:
<INPUT TYPE="TEXT" NAME="price" VALUE="$"><BR>
<HR>
First Name:
<INPUT TYPE="TEXT" NAME="firstName"><BR>
Last Name:
<INPUT TYPE="TEXT" NAME="lastName"><BR>
Middle Initial:
<INPUT TYPE="TEXT" NAME="initial"><BR>
Shipping Address:
<TEXTAREA NAME="address" ROWS=3 COLS=40></TEXTAREA><BR>
Credit Card:<BR>
    <INPUT TYPE="RADIO" NAME="cardType"
          VALUE="Visa">Visa<BR>
    <INPUT TYPE="RADIO" NAME="cardType"
          VALUE="Master Card">Master Card<BR>
    <INPUT TYPE="RADIO" NAME="cardType"
          VALUE="Amex">American Express<BR>
    <INPUT TYPE="RADIO" NAME="cardType"
          VALUE="Discover">Discover<BR>
    <INPUT TYPE="RADIO" NAME="cardType"
          VALUE="Java SmartCard">Java
SmartCard<BR>
Credit Card Number:
<INPUT TYPE="PASSWORD" NAME="cardNum"><BR>
Repeat Credit Card Number:
<INPUT TYPE="PASSWORD" NAME="cardNum"><BR><BR>
<CENTER>
    <INPUT TYPE="SUBMIT" VALUE="Submit Order">
</CENTER>
</FORM>
</BODY>
</HTML>

```

A Sample FORM using POST

Item Number: 1234

Quantity: 5

Price Each: \$59.95

First Name: Marcy

Last Name: P. S.

Middle Initial: M

Shipping Address:  
University Applied Physics Lab  
11100 John Hopkins Rd  
Laurel, MD 20708

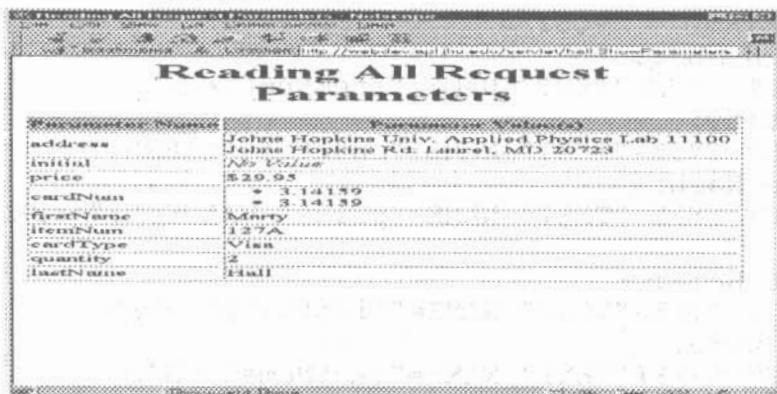
Credit Card:

- Visa
- Master Card
- American Express
- Discover
- Java SmartCard

Credit Card Number: \_\_\_\_\_

Repeat Credit Card Number: \_\_\_\_\_

Hình 6.7. Trang PostForm.html



Hình 6.8. Chương trình đọc các tham số ShowParameters.java

## 6.8. CÁC PHIÊN LÀM VIỆC SESSION

Session được sử dụng để duy trì sự kết nối giữa Client và Server. Khi trình duyệt yêu cầu Web Server cung cấp một trang tài liệu, nó thiết lập một kết nối, lấy về nội dung của trang yêu cầu và sau đó huỷ bỏ kết nối đó ngay lập tức. Để lưu lại dấu vết các trạng thái mà trình duyệt yêu cầu thực hiện trước đó, Web Server cài đặt sẵn đối tượng của HttpSession. HttpSession dựa vào khái niệm Cookie qui định giữa Server và Client. *Cookie là một mẫu tin được gửi cho trình duyệt ở Client khi có yêu cầu về một trang tin.* Mỗi khi trình duyệt gửi một yêu cầu cho Server, nó lại chuyển mẫu tin Cookie trả lại cho Server. Dựa vào các Cookie mà chương trình Server và Client có được những thông tin trạng thái thông báo cho nhau.

Giao diện HttpSession có 3 phương thức thường xuyên sử dụng:

- public void setAttribute(String name, Object value) throws IllegalStateException: Kết hợp một tên khoá với giá trị của biến.
- public Object setAttribute(String name) throws IllegalStateException: Trả về đối tượng tương ứng với thuộc tính có tên là name.
- public void removeAttribute(String name) throws IllegalStateException: Loại bỏ thuộc tính có tên là name.

Để sử dụng Session, ta thực hiện như sau:

- Nhận về một đối tượng (của HttpSession) cho người sử dụng bằng cách gọi phương thức getSession() đối với đối tượng của lớp HttpServletRequest.

```
HttpSession userSession = request.getSession();
```

- Lưu trữ và nhận dữ liệu từ Session. Giao diện cung cấp những phương thức để đặt lại giá trị cho các thuộc tính hoặc xác định chúng như đã nêu ở trên.

**Ví dụ 6.6.** Servlet thực hiện lưu trữ và đọc dữ liệu từ một Session

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SessionServlet extends HttpServlet{
    public void doGet(HttpServletRequest re, HttpServletResponse res)
        throws ServletException, IOException{
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        HttpSession session = re.getSession(true);
        // In các thông tin về Session
        Date dateCreated = new Date(session.getCreationTime());
        Date dateAccessed = new Date(session.getLastAccessedTime());
        out.println("ID: " + session.getId());
        out.println("Created: " + dateCreated);
        out.println("Last Accessed: " + dateAccessed);
        // Đặt lại các thông tin cần thiết
        String dataName = re.getParameter("dataName");
        if(dataName != null && dataName.length() > 0){
            String dataValue = re.getParameter("dataValue");
            session.setAttribute(dataName, dataValue);
        }
        // In nội dung nhận được từ Session
        Enumeration e = session.getAttributeNames();
        while(e.hasMoreElements()){
            String name = (String)e.nextElement();
            String value = session.getAttribute(name).toString();
            out.println(name + " = " + value);
        }
    }
}
```

## 6.9. SỰ TRUYỀN THÔNG GIỮA CÁC SERVLET

Các Servlet có nhiều cách trao đổi tin với nhau. Chúng cần trao đổi với nhau vì:

- Một Servlet có thể truy cập trực tiếp đến các Servlet được nạp về và cần chúng thực hiện một số công việc nào đó. Một Servlet sử dụng đối tượng của ServletContext để nhận tin từ các Servlet khác.

```
public Servlet getServlet(String name)
    throws ServletException
```

- Một Servlet có thể sử dụng lại những khả năng của Servlet khác để thực hiện một nhiệm vụ nào đó.
- Hai hoặc nhiều Servlet cần chia sẻ thông tin với nhau.

Ta có thể sử dụng phương thức getServlets():

```
public Enumeration getServlets()
```

để lấy về các đối tượng Servlet được nạp về từ ngũ cành Servlet hiện thời ServletContext.

**Ví dụ 6.7.** Chương trình sau mô tả sự trao đổi tin giữa các Servlet với nhau.

```
// InterServlet.java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class InterServlet extends HttpServlet{
    public void doGet(HttpServletRequest re, HttpServletResponse res)
        throws IOException{
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        ServletContext ct = getServletContext();
        Enumeration num = ct.getServletNames();
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Demo Servlet </TITLE></HEAD>");
        out.println("<h3>!!Warning </h3>");
        out.println("getServletNames() ----");
        try{
            while(num.hasMoreElement()){
                String nm = (String)num.nextElement();
                out.println(nm);
            }
        }
    }
}
```

```

        Servlet s = ct.getServlet(nm);
        out.println("Servlet Name: " + nm);
        out.println("Servlet classe: " + s.getClass().getName());
        out.println("Servlet Information: " + s.getServletInfo());
        out.println();
    }
} catch (Exception e) {
    System.out.println("Error: " + e);
}
}
}

```

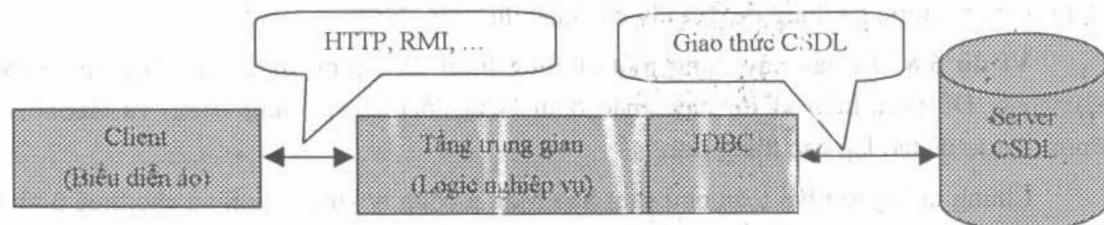
## 6.10. SERVLET KẾT NỐI CÁC CƠ SỞ DỮ LIỆU

Các WebSite ngày nay không chỉ hiển thị những thông tin của các trang HTML tĩnh. Ví dụ, trong thương mại điện tử, một lĩnh vực đang thịnh hành trên thế giới, khách hàng có thể vào một trang Web bán hàng, chọn những mặt hàng cần mua, điền vào phiếu mua hàng sau đó thanh toán (bằng các phương thức trả tín phiếu, check, chuyển khoản, v.v.) thì sẽ nhận được các mặt hàng theo yêu cầu. Như vậy, các thông tin chi tiết về khách hàng phải được lưu trữ trong CSDL ở máy dịch vụ để thực hiện được các dịch vụ thương mại điện tử. Trong những kịch bản như thế, các Website phải được kết nối với các CSDL khác.

Servlet và JDBC [1] là hai công cụ rất hiệu quả cho người lập trình ứng dụng Web kết nối Web với CSDL.

### 6.10.1. Vai trò của Servlet trong mô hình kết nối CSDL

Các chương trình ứng dụng trên mạng hiện nay đang chuyển từ mô hình Client/Server sang *mô hình ba tầng*, hoặc n-tầng. Trong mô hình ba tầng, Client không thực hiện truy vấn trực tiếp các CSDL mà thông qua tầng trung gian ở Server để truy vấn vào các CSDL.



Hình 6.5. Mô hình chương trình ứng dụng ba tầng

Mô hình ba tầng có ưu điểm là nó tách riêng phần biểu diễn ảo (ở phía Client) ra khỏi phần logic nghiệp vụ (tầng trung gian) và dữ liệu thật ở Server. Do vậy, nó cho phép nhiều Client có thể truy cập vào cùng một dữ liệu, cùng một nghiệp vụ và nhiều loại chương trình Java thực hiện như chương trình ứng dụng độc lập, chương trình applet hay chương trình Web. Servlet có một vị trí quan trọng trong việc truy cập vào CSDL ở tầng trung gian. Ví dụ, hãy tưởng tượng có một chương trình đặt mua hàng trên mạng. Nếu không có tầng trung gian thì chương trình này phải kết nối trực tiếp với CSDL trên máy chủ và máy chủ phải ghi lại đơn hàng, cập nhật lại các trường dữ liệu trong CSDL (trừ đi những mặt hàng đã bán theo đơn đặt mua). Chương trình của khách có thể bị ngắt nếu Server của CSDL bị thay đổi theo một cách nào đó. Hoặc có thể một ai đó can thiệp vào chương trình của khách, Server nhận được đơn đặt hàng nhưng không nhận được sự thanh toán của khách, mặc dù khách hàng đã thực hiện thanh toán theo đơn đặt hàng, v.v.

Tầng trung gian sử dụng logic nghiệp vụ để trừu tượng hóa quá trình xử lý đặt hàng. Nó nhận thông tin từ đơn đặt hàng, kiểm tra tính xác thực của thẻ tín dụng, tài khoản, v.v, và chuyển những thông tin đó cho hệ CSDL. Khi hệ CSDL thay đổi thì tầng trung gian sẽ được cập nhật mà không làm ảnh hưởng đến chương trình của khách. Ngoài ra, tầng trung gian còn giúp ta tăng hiệu quả xử lý công việc vì nó hỗ trợ để kết nối chéo với nhiều hệ CSDL khác nhau.

### 6.10.2. Xử lý giao dịch với JDBC

Sự truyền thông giữa Client và tầng trung gian thường sử dụng HTTP (khi người sử dụng dùng Web Browser), RMI (khi người sử dụng dùng phần mềm ứng dụng hay Applet triệu gọi từ xa như đã đề cập ở chương II). Bộ điều khiển kết nối JDBC được sử dụng để quản lý sự trao đổi thông tin giữa tầng trung gian và CSDL.

JDBC là giao diện với SQL, một giao diện với hầu như tất cả các CSDL mô hình dữ liệu quan hệ hiện đại.

Ta hãy xét tiếp chương trình đặt mua hàng trực tuyến. Khách hàng điền vào đơn đặt mua và gửi đến cho Cửa hàng (hệ thống bán hàng trực tuyến). CSDL của hệ thống phải được cập nhật và chèn thêm bản ghi thông tin về khách hàng đó. Tương tự, sau khi giao dịch mua/bán kết thúc thì một số bảng trong hệ CSDL cũng sẽ phải được cập nhật.

Những vấn đề trên được các câu lệnh của SQL thực hiện. Đối tượng của lớp Connection được sử dụng để quản lý các giao dịch với JDBC.

**Ví dụ 6.8.** Ta hãy xây dựng một chương trình ứng dụng thời gian thực có sử dụng Servlet. Để thực hiện ví dụ này, mặc định là ta đã biết sử dụng HTML và Microsoft FrontPage để tạo lập các trang Web.

Chúng ta hãy xét bài toán như sau: Sao Mai là công ty kinh doanh và cho thuê ô tô. Họ bán, cho thuê các ô tô gia đình và các ô tô chở khách cho các tua du lịch. Công ty muốn thành lập một câu lạc bộ cùng với một Website: CarSaoMai.com để thực hiện các dịch vụ trực tuyến nêu trên.

Để tham gia được vào CarSaoMai.com thì bạn phải điền vào một thẻ đăng ký gia nhập câu lạc bộ và chỉ những người đã đăng ký mới được cung cấp các dịch vụ trực tuyến để mua và sử dụng ô tô. Những người chưa đăng ký thì chỉ được quyền truy cập để biết được những thông tin về Công ty và những thông tin tóm tắt về các dịch vụ mà Công ty cung cấp.

Trước tiên, ta hãy thiết kế quá trình đăng ký như sau.

- Tao ra trang HTML chứa mẫu đăng ký.
- Cập nhật lại CSDL về các Servlet làm việc ở tầng trung gian khi có một người mới đăng ký.
- Tạo ra một CSDL chính để lưu trữ thông tin về các thành viên của câu lạc bộ.
- Tạo ra Servlet khác để cho phép tất cả các thành viên câu lạc bộ kết nối được vào Website và được phục vụ.

## 1. Thiết lập CSDL

Đầu tiên, ta có thể sử dụng Microsoft Access để tạo ra một CSDL nhỏ, ví dụ carsSaoMai.mdb. Để sử dụng CSDL này, ta sử dụng 32-bit ODBC (chọn trong Control Panel\Administrative Tools\Data Sources (ODBC)). Sau khi mở được hộp thoại ODBC Data Sources Administrator, hãy chọn System DSN và nhấn vào nút Add để đưa thêm CSDL mới vào nguồn dữ liệu. Nhập tên CSDL carsSaoMai vào trường Data Source Name và nhấn vào Selection để thiết lập đường dẫn tới thư mục chứa tệp carsSaoMai.mdb.

carsSaoMai.mdb có hai bảng được thiết kế như sau.

### (i) Bảng Login

Tên trường	Kiểu dữ liệu	Các ràng buộc
LogName	Text(20)	Khoá nguyên thuỷ
Passwd	Text(20)	Không rỗng

### (ii) Bảng Members: lưu trữ các thành viên câu lạc bộ

Tên trường	Kiểu dữ liệu	Các ràng buộc
LogName	Text(20)	Khoá tham chiếu
FName	Text(30)	Không rỗng
LName	Text(30)	Không rỗng
Age	Number(2)	Không rỗng
Gender	Text(10)	Không rỗng
Address	Text(30)	Không rỗng
City	Text(30)	Không rỗng
PIN	Number(7)	Không rỗng
EMail	Text(30)	Không rỗng
Phone	Number(10)	Không rỗng
Salary	Number(20)	Không rỗng

## 2. Tạo lập trang chủ và mẫu đăng ký câu lạc bộ

Ta có thể tạo ra các trang HTML cho trang chủ và các mẫu đăng ký đã được thiết kế

Lưu lại trang Web trên vào tệp carsSaoMai\_HomePage.shtml. Tệp này có thêm dòng lệnh:

```
<servlet code = http://128.28.10.1:8080/servlet/pageValidator.class>
</servlet>
```

Servlet pageValidator.class cùng với một phần mã HTML sẽ tạo ra Servlet ở phía máy chủ. Lưu ý là cần phải thay đổi địa chỉ URL tới URL của Web Server của bạn.

Tệp carsSaoMai\_HomePage.shtml có các điểm liên kết tới các trang HTML bao gồm:

- SaoMai.html: trang chứa các thông tin về câu lạc bộ.
- member.html: ghi nhận các thành viên đăng ký vào câu lạc bộ
- getRegisteredNow.html và registrationContD.html: cho phép các thành viên mới đăng ký vào câu lạc bộ.

## 3. Viết các Servlet cho tầng trung gian

carsSaoMai.com sử dụng các Servlet để nhận được các thông tin từ Client Browser, thẩm định thông tin này, tương tác với CSDL để chèn các bản ghi mới, tìm trong CSDL những thành viên đã đăng ký và thực hiện các dịch vụ mà các thành viên câu lạc bộ yêu cầu. Đó là các Servlet:

- NewRegistry và MemberDetailEntry: ghi nhận những thành viên mới.
- RegisteredMember: tìm trong CSDL những thành viên đã đăng ký và truy cập để xác định những thông tin cần thiết.

## 4. Dịch và thực hiện chương trình ứng dụng

Tóm lại, việc tạo lập một trang Web (diễn đàn) đáp ứng các yêu cầu trên, được thực hiện các bước như sau:

- Tạo ra một CSDL carsSaoMai.mdb, và để truy cập được vào nó thì phải cho biết tên và mật khẩu của người đã đăng ký tham gia Câu lạc bộ. CSDL này có hai bảng Login và Members.
- Chuyển tất cả các tệp HTML mà Website yêu cầu vào thư mục, ví dụ public\_html của JavaWebServer. Đồng thời chuyển các tệp ảnh vào thư mục này. Nhớ là phải thay đổi URL cho thích hợp như đã nêu ở trên.
- Tiếp theo, chuyển các tệp lớp đã được dịch (.class) của các chương trình Servlet để nạp vào thư mục Servlet của JavaWebServer. Ở đây cũng cần nhớ là phải thay đổi URL đã chỉ ra trong các tệp .java trước khi dịch và chuyển chúng về thư mục \JavaWebServer2.0\servlets.

- Khởi động Java Web Server bằng cách thực hiện httpd.exe ở thư mục Web server\bin.
- Khi Web Server đã hoạt động, hãy mở trình duyệt của bạn và gõ vào

[http://localhost:8080/SaoMai\\_homepage.shtml](http://localhost:8080/SaoMai_homepage.shtml)

- Nếu trang Web này chưa được nạp về thì thay localhost trong URL ở trên bằng địa chỉ IP của máy bạn, ví dụ:

[http://128.28.10.1:8080/SaoMai\\_homepage.html](http://128.28.10.1:8080/SaoMai_homepage.html)

Tương tự như trên nếu ta muốn thực hiện trên Web Server.

## 6.11. JSP

Java Server Pages (JSP) cho phép tách riêng phần động của những trang Web ra khỏi HTML tĩnh. JSP không chỉ hỗ trợ để tạo ra những trang Web độc lập với các nền, độc lập với các Server, mà còn là công nghệ Java rất hiệu quả để thể hiện nguyên lý WYSIWYG (Những gì bạn nhìn thấy là bạn có được chúng) trên các trang HTML tĩnh ([2], [4]).

Các trang JSP gồm có:

- Các thành phần tĩnh HTML/XML
- Các thẻ đặc biệt của JSP
- Có thẻ có một số các đoạn mã chương trình viết bằng Java, được gọi là những kịch bản nhỏ Scriptlet.

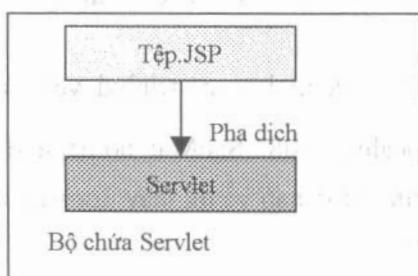
Do vậy, ta có thể tạo lập và duy trì các trang JSP bằng các công cụ truyền thống như đối với HTML/XML.

Một điều quan trọng cần lưu ý: đặc tả JSP là một chuẩn mở rộng, được định nghĩa trên định của Servlet API. Cả Servlet và JSP cùng có một số đặc tính chung, có thể sử dụng chúng phục vụ cho các nội dung Web động. Tuy nhiên, cũng có một số khác biệt giữa JSP và công nghệ Servlet như đã trình bày ở trên. JSP được sử dụng rộng rãi hơn. Nó không chỉ được sử dụng đối với các nhà phát triển hệ thống, mà cả những người thiết kế các trang Web, những người đóng vai trò rất quan trọng trong quá trình phát triển các trang Web ứng dụng hiện nay.

### 6.11.1. Kiến trúc của JSP

Mục đích của JSP là cung cấp các phương thức để khai báo, biểu diễn cho sự phát triển của các Servlet.

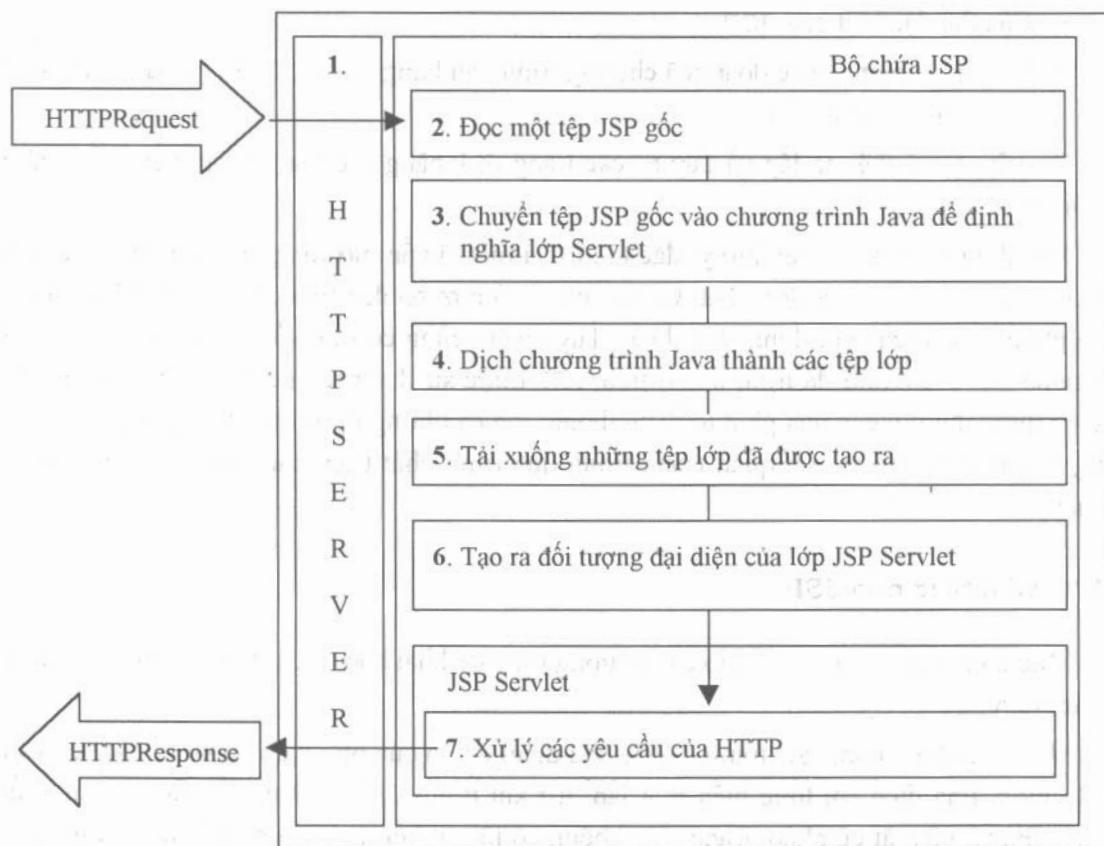
Một cách điển hình, các trang JSP là chủ điểm chính của pha dịch và xử lý các yêu cầu trên Server. Pha dịch chỉ thực hiện một lần, trừ khi trang JSP bị thay đổi thì nó mới phải dịch lại. Giả sử về mặt cú pháp, trang JSP không có lỗi thì kết quả của pha dịch sẽ là tệp lớp cài đặt trang JSP và giao diện Servlet như hình dưới đây.



Hình 6.6. Cơ chế dịch của JSP

Pha dịch được thực hiện bởi JSP Engine khi nó nhận được các yêu cầu lần đầu của một trang JSP. Lưu ý rằng, JSP 1.1 cho phép dịch trước (tiền dịch) các trang JSP thành các tệp lớp. Việc dịch trước này đặc biệt hữu dụng trong việc loại bỏ những khởi đầu chậm trễ khi một trang JSP được phân phát từ các nguồn để nhận được các yêu cầu của Client. Nguồn gốc của tệp lớp được tạo ra bởi Tomcat.

Như trên đã nêu, một chương trình JSP có thể chứa các đoạn chương trình Java và chúng kết hợp với nhau để tạo ra các JSP Servlet nhằm xử lý các yêu cầu của Client gửi đến thông qua HttpServletRequest. JSP Servlet xử lý các yêu cầu của Client (qua HTTP) và gửi kết quả cho HttpServletResponse.

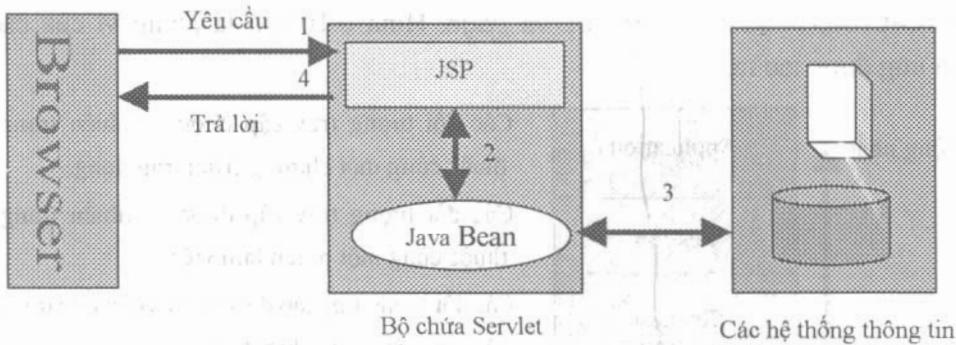


Hình 6.7. Cơ chế hành động của bộ chứa JSP

### 6.11.2. Các mô hình truy cập của JSP

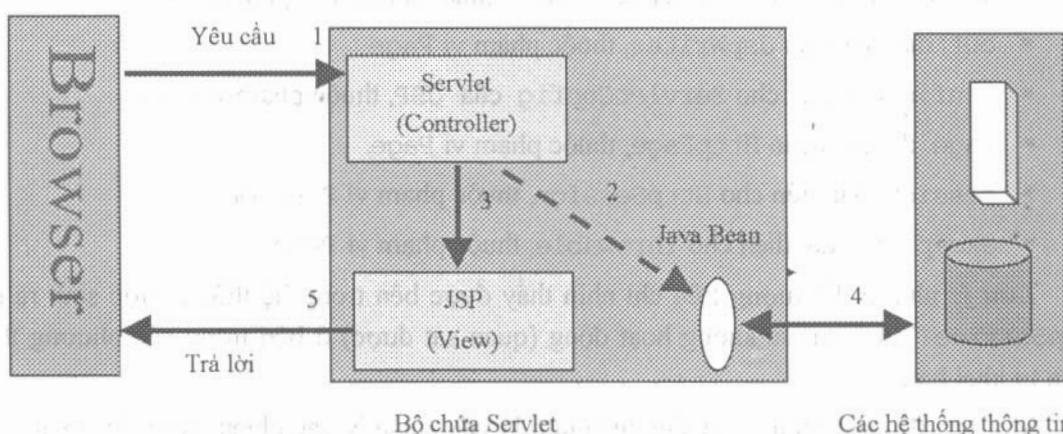
Có hai mô hình truy cập phổ biến để thực hiện truy cập thông tin theo công nghệ JSP.

- Mô hình 1:* các yêu cầu đầu vào từ Web Browser được gửi trực tiếp tới trang JSP, ở đó chúng được xử lý và trả lời cho Client. Trong mô hình này, mọi vấn đề truy cập đều được đảm nhận bởi các Java Bean.



Hình 6.8. Mô hình 1

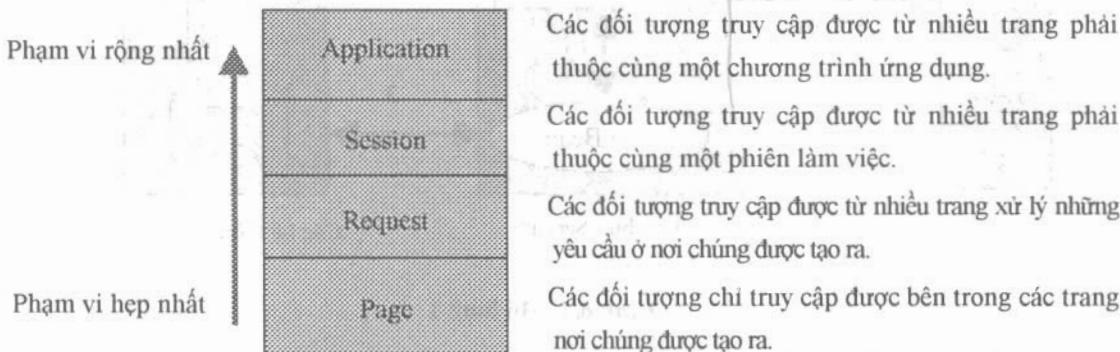
- Mô hình 2:* còn được biết đến với tên gọi MVC (Model/View/Controller). Việc xử lý thông tin được tách thành hai phần, thành phần biểu diễn và thành phần điều khiển. Thành phần biểu diễn là những trang JSP làm nhiệm vụ tạo ra câu trả lời dạng HTML/XML và xác định giao diện của người sử dụng mà Web Browser cung cấp. Thành phần điều khiển (còn được gọi là thành phần phía trước), không đảm nhận công việc biểu diễn mà là xử lý tất cả các yêu cầu của HTTP. Các thành phần điều khiển cũng đảm nhận việc tạo ra các Java Bean hoặc các đối tượng phục vụ cho các thành phần biểu diễn sử dụng, đồng thời quyết định các thành phần phụ thuộc vào hành động của người sử dụng.



Hình 6.9. Mô hình 2

### 6.11.3. Các phạm vi đối tượng

Trước khi tìm hiểu về cú pháp và ngữ nghĩa của JSP, điều quan trọng là phải hiểu rõ phạm vi hay khả năng quan sát được của các đối tượng Java xử lý các yêu cầu trong các trang JSP. Các đối tượng có thể được tạo ra một cách tương minh bằng cách sử dụng các thẻ chỉ dẫn JSP, hoặc một cách gián tiếp thông qua các thẻ hành động. Những đối tượng được sinh ra sẽ được gắn tương ứng với những phạm vi hoạt động nhất định. Có bốn phạm vi hoạt động: Application, Session, Request và Page. Hình 6.10 mô tả phạm vi của các đối tượng tương ứng được tạo ra.



**Hình 6.10. Phạm vi hoạt động của các đối tượng**

Để cho thuận tiện, bộ chứa JSP tạo ra một số đối tượng tiềm năng để sử dụng trong các kịch bản và các biểu thức mà không cần phải khởi tạo ra chúng. Có chín đối tượng không tương minh được tạo lập sẵn:

- request: đại diện cho HttpServletRequest, thuộc phạm vi Request.
- response: đại diện cho HttpServletResponse, thuộc phạm vi Page.
- pageContext: đại diện cho PageContext, thuộc phạm vi Page.
- application: đại diện cho ServletContext, thuộc phạm vi Application.
- out: đại diện cho JspWriter, thuộc phạm vi Page.
- config: đại diện cho ServletConfig của JSP, thuộc phạm vi Page.
- page: đại diện cho HttpPage, thuộc phạm vi Page.
- session: đại diện cho HttpSession, thuộc phạm vi Session.
- exception: đại diện cho Throwable, thuộc phạm vi Page.

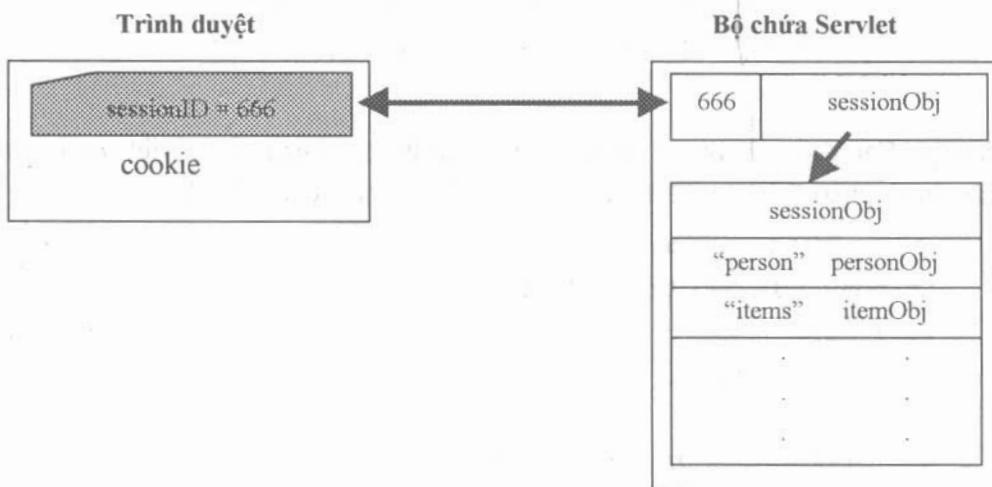
**Lưu ý:** những đối tượng trên chỉ nhìn thấy được bên trong hệ thống được sinh ra bởi `_jspService()`. Chúng sẽ không hoạt động (quan sát được) ở bên trong các phương thức mà ta tự khai báo.

Một vấn đề quan trọng nữa cần quan tâm là việc quản lý các phiên làm việc trong JSP. Theo mặc định, tất cả các trang JSP đều tham gia vào một phiên session của HTTP. Các phiên session là những nơi thích hợp để tổ chức lưu trữ các Bean và những đối tượng cần thiết sử dụng chung cho các trang JSP với các Servlet mà người sử dụng cần truy cập. Mỗi

session được xác định thông qua định danh ID và được lưu ở trình duyệt cùng với mẩu tin cookie.

Số lượng các đối tượng trong một phiên session là không giới hạn. Song khi số lượng chúng nhiều quá thì hiệu suất thực hiện sẽ kém đi. Theo mặc định, phần lớn các Server cho phép mỗi đối tượng tham gia vào một phiên session với thời gian tham gia là 30 phút. Nếu muốn thay đổi, bạn có thể đặt lại thời gian hoạt động của đối tượng trong một phiên session thông qua hàm `setMaxInvalidationInterval(int secs)`. Cơ chế quản lý các phiên session được mô tả như hình 6.11.

Mô tí JSP giữ tham chiếu tới đối tượng được đặt vào trong session cho đến khi hết thời gian có hiệu lực, sau đó những đối tượng đó bị huỷ bỏ để dọn dẹp bộ nhớ.



Hình 6.11. Cơ chế quản lý các session

#### 6.11.4. Cơ sở cú pháp của JSP

Ngoài các qui ước của HTML, JSP có ba cấu trúc chính:

- Các phần tử kịch bản (Scripting Elements) cho phép xác định những đoạn mã chương trình Java sẽ trở thành bộ phận của Servlet,
- Các chỉ dẫn (Directives) cho phép điều khiển tất cả các cấu trúc của Servlet,
- Các hành động (Actions) cho phép chỉ ra những thành phần được sử dụng để xử lý và những thành phần còn lại để điều khiển trong mô tí JSP.

**Lưu ý:** Bạn có thể tìm hiểu chi tiết về JSP Tutorial ở địa chỉ:

<http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/>.

##### 1. Các phần tử kịch bản

Các phần tử kịch bản được sử dụng để chèn các đoạn mã chương trình Java vào trong một Servlet được sinh ra từ trang JSP hiện thời. Có ba thẻ chính:

- (i) Các thẻ *biểu thức* dạng `<%= exp %>`, được sử dụng để tính giá trị của biểu thức `exp` và chèn kết quả (sau khi đã được chuyển về dạng xâu, theo các chỉ dẫn) vào trang kết quả. `exp` là một biểu thức trong Java được tính giá trị lúc thực hiện chương trình (khi trang tin được yêu cầu). Ví dụ, biểu thức JSP sau sẽ xác định ngày/giờ hiện thời của hệ thống khi trang tin yêu cầu:

Thời gian: `<%= new java.util.Date() %>`

- (ii) Các thẻ *kịch bản* dạng `<% Java code %>`, được sử dụng để chèn các phương thức dịch vụ Java code vào trang JSP. Ví dụ:

```
<%
String queryData = request.getQueryString();
out.println("GET Data: " + queryData);
%>
```

**Lưu ý:** Các đoạn chương trình bên trong các thẻ kịch bản, được viết trước hoặc sau thành phần tĩnh của HTML tạo ra các kịch bản hành động. Ví dụ

```
<% if(Math.random() < 0.5) { %>
Là ngày <B> đẹp trời </B>!
<% } else { %>
Là ngày <B> nhiều mây </B>!
<% } %>
<% for(int i = 0; i < 5; i++) { %>
<H<%= i %>>Xin chào </H<%=i %>>
<% } %>
```

- (iii) Các thẻ *khai báo* dạng `<%! Java code %>`, được sử dụng để chèn vào nội dung của các lớp Servlet, chèn vào bên ngoài của phương thức code. Các thẻ khai báo không sinh ra những kết quả, chúng được sử dụng kết hợp các thẻ biểu thức hoặc các kịch bản của JSP. Ví dụ, đoạn chương trình sau in ra số lần trang Web được truy cập sau khi nó được khởi tạo.

```
<%! private int accessCount = 0; %>
```

Số lần truy cập:

```
<%= ++accessCount %>
```

**Lưu ý:** Cũng giống như các kịch bản, nếu ta sử dụng những ký tự như “%>” thì phải viết thành “%\>”.

## 2. Các chỉ dẫn JSP

Một chỉ dẫn trong JSP được sử dụng để gắn cấu trúc của lớp Servlet. Các thẻ chỉ dẫn JSP là những thông điệp dành cho mô tơ JSP. Thẻ chỉ dẫn được đặt ở đầu của tất cả các trang JSP. Có hai thẻ chỉ dẫn chính: thẻ `page`, `include`.

- (i) *Chi dẫn JSP page.* Thẻ chi dẫn page cho phép bạn định nghĩa một trong các thuộc tính sau:
- `<%@ page import = "package.Class" %>`  
Ví dụ: `<%@ page import = "java.util.*" %>`
  - `<%@ page contentType = "MIME-Type" %>` hoặc  
`<%@ page contentType = "MIME-Type; charset=Character-Set" %>`  
Trong đó, MIME là kiểu của kết quả. Ví dụ:  
`<%@ page contentType = "text/plain" %>`
  - `<%@ page isThreadSafe = "true | false" %>`  
Một giá trị true (mặc định) chỉ ra rằng việc xử lý của Servlet là bình thường, trong đó có thể có nhiều yêu cầu được xử lý đồng thời bởi cùng một Servlet. Giá trị false chỉ ra rằng, Servlet cài đặt SingleThreadModel để thực hiện các yêu cầu một cách tuần tự hay được thực hiện bởi nhiều Servlet cùng một lúc.
  - `<%@ page session = "true | false" %>`  
Một giá trị true (mặc định) chỉ ra rằng biến được định nghĩa trước session (thuộc kiểu HttpSession) sẽ được tìm thấy trong session nếu nó tồn tại. Giá trị false chỉ ra rằng, những session như thế không được sử dụng, mọi cố gắng truy cập vào session đều thất bại.
  - `<%@ page buffer = "sizekb | none" %>`  
Thuộc tính này xác định cỡ của vùng đệm buffer được sử dụng bởi đối tượng out của JspWriter, cỡ nhỏ nhất phải là 8 KB.
  - `<%@ page autoflush = "true | false" %>`  
Một giá trị true (mặc định) chỉ ra rằng, buffer sẽ được làm sạch khi nó bị đầy. Giá trị false, ít khi sử dụng, chỉ ra những ngoại lệ, khi buffer bị đầy.
  - `<%@ page extends = "package.class" %>`  
Thuộc tính này chỉ ra lớp cha của Servlet được tạo ra.
  - `<%@ page language = "java" %>`  
Thuộc tính này chỉ ra ngôn ngữ được sử dụng.
- JSP còn cung cấp một cơ chế để xử lý ngoại lệ (lỗi) khi thực thi chương trình bằng cách sử dụng thẻ page với thuộc tính `errorPage`:
- ```
<%@ page isErrorPage = "false" errorPage = "errorHandler.jsp" %>
```
- thông báo cho mô tả JSP biết rằng những ngoại lệ được mô tả trong `errorHandler.jsp`.

- (ii) *Chi dẫn JSP include.* Thẻ chi dẫn include cho phép bạn đưa các tệp chương trình, hay tệp tin vào trong trang JSP lúc nó được dịch thành Servlet.

```
<%@ include file="relative url" %>
```

Trong đó, "url" là địa chỉ chứa những tệp cần phải được đưa vào trang JSP.

### 3. Thẻ chú thích

Thẻ chú thích của HTML có thể sử dụng trong các trang JSP. Tuy nhiên, người sử dụng vẫn có thể nhìn thấy những chú thích đó trong những trang gốc. Nếu bạn muốn những chú thích đó không nhìn thấy được bởi người sử dụng thì phải dùng thẻ chú thích của JSP.

```
<%-- Chú thích chỉ dành cho Server --%>
```

Những chú thích này chỉ có ý nghĩa giải thích dành cho người viết chương trình, sau khi dịch thành tệp lớp thì những người sử dụng không đọc chúng được nữa.

**Ví dụ 6.9.** Viết chương trình JSP để đọc dữ liệu được nhập vào từ dòng lệnh và hiển thị tên của máy tính chạy chương trình, số lần truy cập và ngày/giờ hiện thời của hệ thống.

```
<%-- jspTest.jsp --%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional //EN">
<HTML>
<HEAD>
<TITLE> Sử dụng Java Server Pages </TITLE>
<LINK REL=STYLESHEET
      HREF="My-Style-Sheet.css"
      TYPE="text/css">
</HEAD>
<BODY BGCOLOR= "#FDF5E6" TEXT="#000000" LINK="0000EE"
      VLINK="#551A8B" ALINK="FF0000">
<CENTER>
<TABLE BORDER=5 BGCOLOR="#EF8429">
<TR><TH CLASS="TITLE">
      Sử dụng Java Server Pages </TABLE>
</CENTER>
<p>
Những nội dung được tạo ra trong trang JSP
<UL>
<LI><B>Thẻ biểu thức. </B><BR>
```

```

        Hostname: <%=request.getRemoteHost() %>.

<LI><B>Thẻ kịch bản. </B><BR>
        <% out.println("Dữ liệu đọc được: " +
                    request.getQueryString()); %>.

        <LI><B>Thẻ khai báo. </B><BR>
        <% private int accessCout = 0; %>
        Số lần truy cập vào Web site: <%= ++accessCout %>
<LI><B>Thẻ chỉ dẫn. </B><BR>
        <%@ page import = "java.util.*" %>
        Ngày/giờ hiện thời: <%= new Date() %>
</UL>

</BODY>
</HTML>
```

### 6.11.5. Các hành động của JSP

Các hành động JSP sử dụng các cấu trúc XML để điều khiển hành vi của Servlet Engine. JSP có những hành động sau:

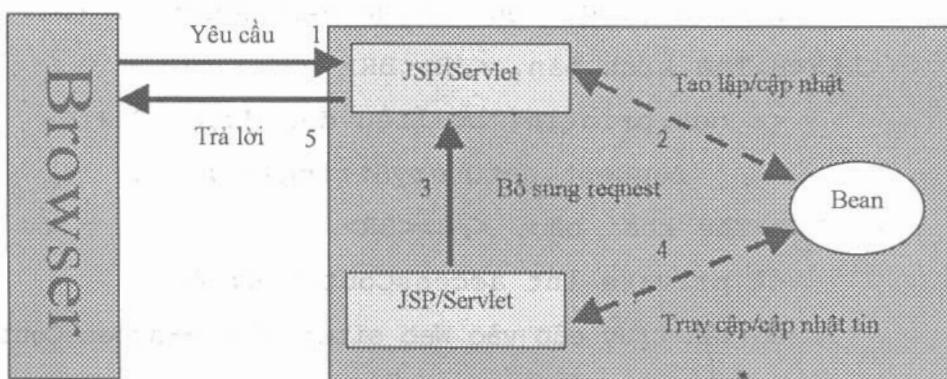
- **jsp:include** – Đưa vào một tệp khi trang tin được yêu cầu.
- **jsp:useBean** – Sử dụng đối tượng JavaBean.
- **jsp:setProperty** – Đặt lại thuộc tính cho JavaBean.
- **jsp:getProperty** – Đọc thuộc tính của JavaBean.
- **jsp:forward** – Hướng tới một trang mới.
- **jsp:plugin** – Cho phép chèn vào trình duyệt được xác định bởi OBJECT hoặc EMBED cần thiết để chạy Applet.

#### 1. Thẻ **jsp:include**

Thẻ này cho phép bạn chèn các tệp cần thiết vào trang tin được tạo ra. Nó có dạng

```
<jsp:include page = "relative URL" flush = "true" />
```

Nó được dùng để định hướng yêu cầu tới những nguồn tài nguyên tĩnh hoặc động trong cùng ngữ cảnh theo địa chỉ relative URL như khi gọi trang JSP. Cơ chế thực hiện bằng cách tạo ra các tham số là các JavaBean tài nguyên (được giới thiệu ở chương V) và đưa chúng vào yêu cầu, được mô tả như hình 6.12.



Hình 6.12. Cơ chế xử lý yêu cầu include

**Ví dụ 6.10.** Viết chương trình JSP để đọc, hiển thị các tin được đưa vào từ các tệp chứa chúng.

```

<%-- WhatsNew.jsp --%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional //EN">
<HTML>
<HEAD>
<TITLE> Trang tin tức </TITLE>
<LINK REL = "STYLESHEET"
      HREF = "My-Style-Sheet.css"
      TYPE = "text/css">
</HEAD>
<BODY BGCOLOR = "#FDF5E6" TEXT = "#000000" LINK = "0000EE"
      VLINK = "#551A8B" ALINK = "FF0000">
<CENTER>
<TABLE BORDER = 5 BGCOLOR = "#EF8429">
    <TR><TH CLASS = "TITLE">
        Nhũng tin mới nhận được từ JspNews.com </TABLE>
    </CENTER>
    <p>
        Nhũng tin mới nhận được:
    <OL>
        <LI><jsp:include page="news/Item1.html" flush="true"/>
        <LI><jsp:include page="news/Item1.htm2" flush="true"/>
        <LI><jsp:include page="news/Item1.htm3" flush="true"/>
    </OL>
    </BODY>
</HTML>
  
```

## 2. Thẻ `jsp:useBean`

Thẻ hành động này cho phép tải các JavaBean (được trình bày ở chương V) vào trang JSP để sử dụng chúng. Điều này rất tiện lợi và hiệu quả vì nó cho phép ta khai thác được những khả năng sử dụng lại của các lớp được xây dựng sẵn trong Java mà không đòi hỏi phải thỏa mãn những yêu cầu cụ thể, những yêu cầu này sẽ được JSP đưa vào trong Servlet để thực hiện. Nó có cú pháp đơn giản như sau:

```
<jsp:useBean id = "name" class = "package.class" />
```

Trước khi truy cập vào một Bean (JavaBean) trong trang JSP thì phải chỉ rõ nó có tên là gì (định danh), tên lớp tương ứng ở đâu. Khởi tạo một đối tượng của lớp được chỉ ra bởi class và nó liên kết với biến thông qua tên gọi id. Sẽ là hiệu quả hơn khi ta tham chiếu tới những JavaBean đã có sẵn, và `jsp:useBean` cho biết rằng, một đối tượng mới được khởi tạo chỉ khi chưa có đối tượng nào có cùng id đã được tạo ra trước đó. Khi đã tạo ra một Bean, ta có thể thay đổi các tính chất của nó bằng cách sử dụng thẻ `jsp:setProperty`, hoặc sử dụng các thẻ kịch bản và gọi những phương thức tường minh của đối tượng đó thông qua định danh id. Ví dụ, hãy xét thẻ sau:

```
<jsp:useBean id = "user" class = "company.Person"
    scope = "session" />
```

Một đối tượng của lớp Person được tạo ra và được đặt vào phiên làm việc session. Nếu sau đó, ở một trang JSP khác gặp phải thẻ `jsp:useBean` trên thi nó sẽ tham chiếu tới đối tượng ban đầu đã được sinh ra ở trong phiên session.

Khi đã khai báo một thành phần JavaBean, ta có thể truy cập vào các tính chất của nó thông qua thẻ `jsp:getProperty` và có thể thay đổi những tính chất đó thông qua thẻ `jsp:setProperty`.

**Ví dụ 6.11.** Ví dụ đơn giản hướng dẫn cách sử dụng Bean và đặt / xác định các thuộc tính của JavaBean.

```
<%-- BeanTest.jsp --%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional //EN">
<HTML>
<HEAD>
<TITLE> Sử dụng lại JavaBean trong JSP </TITLE>
<LINK REL=STYLESHEET
    HREF="My-Style-Sheet.css"
    TYPE="text/css">
</HEAD>
<CENTER>
<TABLE BORDER=5 BGCOLOR="#EF8429">
```

```

<TR><TH CLASS="TITLE">
    Sử dụng lại JavaBean trong JSP </TABLE>
</CENTER>

<p>
<jsp:useBean id = "test" class = "hall.SimpleBean" />
<jsp:setProperty name = "test"
    property = "message"
    value = "Xin chào các bạn" />
<H1> Thông báo: <I>
<jsp:getProperty name = "test" property = "message" />
</I>
</BODY>
</HTML>

```

Chương trình của SimpleBean được sử dụng trong BeanTest có thể tải về từ địa chỉ trên mạng. SimpleBean có nội dung đơn giản như sau:

```

package hall;
public class SimpleBean{
    private String message = "Chưa xác định thông báo";
    public String getMessage(){
        return (message);
    }
    public void setMessage(String message){
        this.message = message;
    }
}

```

### 3. Thủ thuật

Bạn có thể sử dụng thẻ `jsp:setProperty` để đặt lại các tính chất của các Bean mà trang JSP tham chiếu tới. Có thể thực hiện theo hai cách.

- + *Cách thứ nhất:* sử dụng `jsp:setProperty`, nhưng bên ngoài của `jsp:useBean` như sau:

```

<jsp:useBean id="myName" ... />
...
<jsp:setProperty name="myName"
    property="someProperty" ... />

```

Trong trường hợp này, `jsp:setProperty` được thực hiện mà không cần để ý đến việc một đối tượng mới được tạo ra hay đã có sẵn đối tượng đó.

- + *Cách thứ hai:* sử dụng `jsp:setProperty` bên trong của thẻ `jsp:useBean` như sau:

```
<jsp:useBean id="myName" ... >
...
<jsp:setProperty name="myName"
                  property="someProperty" ... />
</jsp:useBean>
```

Ở đây, `jsp:setProperty` chỉ thực thi được khi chưa có sẵn đối tượng Bean và một đối tượng mới được tạo ra.

`jsp:setProperty` có các thuộc tính sau:

Thuộc tính	Mô tả cách sử dụng
name	Thuộc tính yêu cầu chỉ định Bean có các tính chất sẽ được đặt lại. Thẻ <code>jsp:useBean</code> phải xuất hiện trước <code>jsp:setProperty</code> .
property	Thuộc tính yêu cầu chỉ ra tính chất mà bạn muốn đặt lại. Song, ở đây có trường hợp đặc biệt: giá trị "*" nghĩa là tất cả các tham số yêu cầu của Bean có tên sánh được với id sẽ truyền tới những phương thức tương ứng.
value	Thuộc tính được lựa chọn xác định giá trị của tính chất đó. Các giá trị xâu (String) được tự động chuyển sang các số kiểu boolean, Boolean, byte, Byte, char, và Character thông qua phương thức <code>valueOf()</code> của các lớp bao gói. Ví dụ, giá trị <code>s = "42"</code> được chuyển sang kiểu int hoặc Integer, thông qua <code>Integer.valueOf(s)</code> .
param	Thuộc tính được lựa chọn chỉ định tham số yêu cầu từ tính chất được dẫn xuất ra. Nếu yêu cầu hiện thời không có tham số như trên thì hệ thống không làm gì cả. <pre>&lt;jsp:setProperty name="orderBean"                   property="numberOfItems"                   param="numItems" /&gt;</pre> <p>Nếu ta bỏ qua cả value và param, thì nó có tên gọi giống như tên được cung cấp bởi param thông qua tên của property.</p>

**Ví dụ 6.12.** Sử dụng Bean để tạo ra một bảng các chữ số nguyên tố. Nếu tham số có tên là numDigits trong dữ liệu yêu cầu, nó sẽ được truyền vào cho tính chất numDigits của Bean.

```
<%-- JspPrimes.jsp --%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML>
<HEAD>
<TITLE>Reusing JavaBeans in JSP</TITLE>
```

```

<LINK REL=STYLESHEET
      HREF="My-Style-Sheet.css"
      TYPE="text/css">
</HEAD>
<BODY>
<CENTER>
<TABLE BORDER=5>
  <TR><TH CLASS="TITLE">
    Reusing JavaBeans in JSP</TABLE>
</CENTER>
<P>
<jsp:useBean id="primeTable" class="hall.NumberedPrimes" />
<jsp:setProperty name="primeTable" property="numDigits" />
<jsp:setProperty name="primeTable" property="numPrimes" />
Some <jsp:getProperty name="primeTable" property="numDigits" />
digit primes:
<jsp:getProperty name="primeTable" property="numberedList" />

</BODY>
</HTML>

```

Chương trình NumberedPrimes.java để tìm các số nguyên tố được viết như sau:

```

package hall;
import java.util.*;
public class NumberedPrimes {
    private Vector primes;
    private int numPrimes = 15;
    private int numDigits = 50;

    public String getNumberedList() {
        if (primes == null) {
            PrimeList newPrimes =
                new PrimeList(numPrimes, numDigits, false);
            primes = newPrimes.getPrimes();
        }
        StringBuffer buff = new StringBuffer("<OL>\n");
        for(int i=0; i<numPrimes; i++) {
            buff.append("  <LI>");
            buff.append(primes.elementAt(i));
            buff.append("\n");
        }
        buff.append("</OL>");
        return(buff.toString());
    }
}

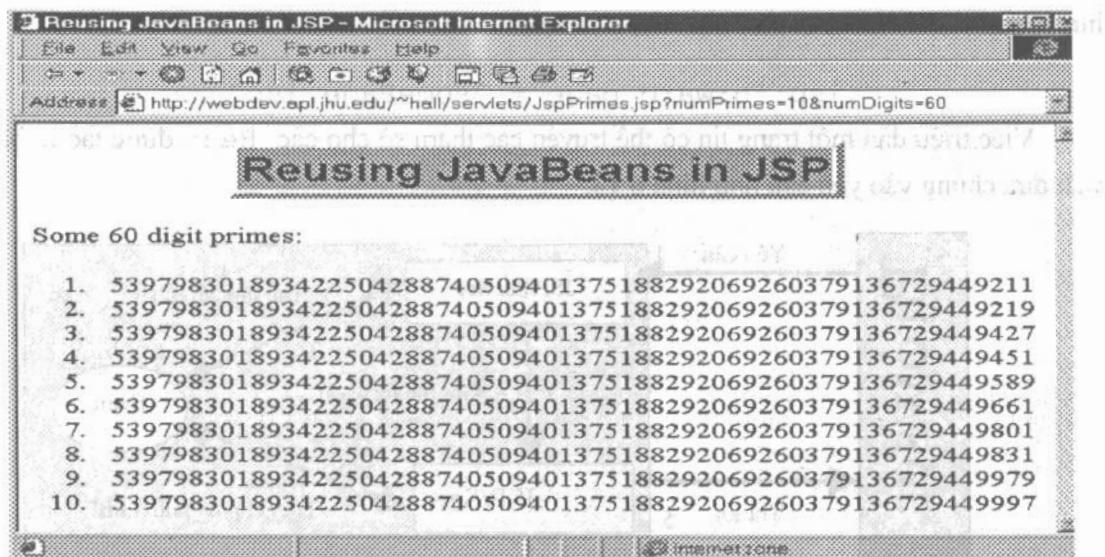
```

```

public int getNumPrimes() {
    return(numPrimes);
}
public void setNumPrimes(int numPrimes) {
    if (numPrimes != this.numPrimes)
        primes = null;
    this.numPrimes = numPrimes;
}
public int getNumDigits() {
    return(numDigits);
}
public void setNumDigits(int numDigits) {
    if (numDigits != this.numDigits)
        primes = null;
    this.numDigits = numDigits;
}
public Vector getPrimes() {
    return(primes);
}
public void setPrimes(Vector primes) {
    this.primes = primes;
}
}

```

Khi thực hiện, ta có kết quả như sau:



Hình 6.12. Chương trình sử dụng JavaBean trong JSP

#### 4. Thẻ `jsp:getProperty`

Thẻ này tìm các giá trị thẻ hiện các tính chất của Bean, được chuyển sang xâu và chèn vào kết quả. Có hai thuộc tính yêu cầu là:

- Tên của Bean được tham chiếu trước đó bởi `jsp:useBean`,
- Tính chất có giá trị tương ứng được chèn vào.

```
<jsp:useBean id="itemBean" ... />
```

```
...
```

```
<UL>
```

`<LI>Số các mặt hàng:`

```
<jsp:getProperty name="itemBean"
    property="numItems"/>
```

`<LI>Giá bán mỗi mặt hàng:`

```
<jsp:getProperty name="itemBean" property="unitCost" />
```

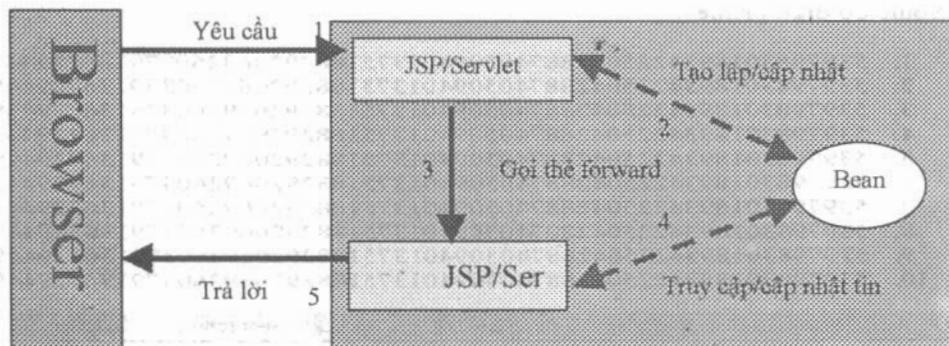
```
</UL>
```

#### 5. Thẻ `jsp:forward`

Thẻ này cho phép bạn khai báo trước về yêu cầu tới một trang tin khác. Nó cho phép hướng dẫn để vào một trang JSP, Servlet, hay một trang HTML tĩnh bên trong ngữ cảnh như là câu khẩn tới những trang đó. Kết quả là mọi công việc xử lý của trang hiện thời sẽ bị dừng lại ở nơi mà thẻ hướng dẫn này xuất hiện, mặc dù những công việc đó vẫn phải tiếp tục thực hiện.

```
<jsp:forward page = "somePage.jsp" />
```

Việc triệu dẫn một trang tin có thể truyền các tham số cho các Bean được tạo ra bằng cách đưa chúng vào yêu cầu như hình 6.13.



Hình 6.13. Cơ chế xử lý yêu cầu forward

## BÀI TẬP

- 6.1.** Xây dựng các lớp và các tệp HTML tương ứng để thực hiện các nhiệm vụ của Công ty Sao Mai như đã nêu trong ví dụ 6.8.
- 6.2.** Xây dựng một diễn đàn trao đổi (forum) với JSP và Servlet. Yêu cầu chính của bài toán như sau.
  1. Một diễn đàn trao đổi thông tin trên mạng cần cho phép các thành viên đăng ký tham gia, tạo các chủ đề, trả lời các bài viết; còn đối với người quản trị thì được phép tạo ra các hộp (Box) chủ đề, phân quyền cho các thành viên, quản lý thông tin diễn đàn ...
  2. Thông tin của diễn đàn được thể hiện thông qua các bài viết của thành viên. Các bài viết được phân loại theo từng chủ đề, mỗi chủ đề sẽ đề cập xung quanh một vấn đề, sự kiện, ý kiến bình luận nào đó. Tập hợp các chủ đề liên quan đến một thể loại thông tin nhất định nào đó lại được nhóm vào trong một hộp (Box), ví dụ ta có thể có hộp Thể thao, hộp Âm nhạc, hộp Văn học .v.v. Việc phân cấp tạo ra các hộp chủ đề sẽ giúp cho người đọc dễ dàng theo dõi các bài viết cũng như giúp việc quản lý diễn đàn hiệu quả và linh động hơn.
  3. Người quản lý cũng là thành viên của diễn đàn có chức năng quản lý diễn đàn. Người quản lý có thể thêm, xóa, sửa ... các thông tin bên trong diễn đàn mà họ trực tiếp quản lý. Việc chỉ định ai là người quản lý của từng hộp chủ đề sẽ do người quản trị (Administrator) quyết định.
  4. Bạn có thể chỉnh sửa hay xoá bài viết của bạn bất kỳ lúc nào bằng cách vào diễn đàn có chứa bài viết, hoặc nhấn vào biểu tượng sửa hoặc xoá (ແຟັງ). Không ai khác có quyền sửa bài viết của bạn, ngoại trừ người quản lý hay quản trị diễn đàn. Khi bài viết được chỉnh sửa, một dòng ghi chú sẽ xuất hiện ở cuối bài viết để thông báo về thời gian sửa và người sửa bài viết đó. Bạn cũng có thể xóa bài viết do mình đã tạo ra trước đó. Nếu quản trị mạng hay quản lý các diễn đàn chọn chức năng này, bài viết cũng sẽ bị xoá.
  5. Bạn có thể chỉnh sửa thông tin cá nhân của mình dễ dàng bằng cách nhấn vào "Thông tin cá nhân" trên mỗi trang, điền các thông tin về bạn (Ngoại trừ tên đăng nhập không được sửa).

# Chương 7

## VẤN ĐỀ BẢO MẬT VÀ AN NINH THÔNG TIN

Chương VII trình bày những kỹ thuật bảo mật thông tin và đảm bảo an toàn hệ thống thông tin bằng Java.

- ✓ Những vấn đề quan trọng của bảo mật hệ thống thông tin
- ✓ Vai trò của các bộ nạp lớp ClassLoader và kiểm định byte code
- ✓ Bộ quản lý bảo mật và các lớp được cấp đặc quyền
- ✓ Những vấn đề bảo mật và chữ ký số
- ✓ Vấn đề chứng thực: chứng chỉ số, chữ ký số và xác thực danh tính người chứng thực
- ✓ Mật mã khóa công khai RSA và ứng dụng trong lập trình Java.

### 7.1. GIỚI THIỆU VỀ VẤN ĐỀ BẢO MẬT, AN TOÀN HỆ THỐNG THÔNG TIN

Bảo mật thông tin là vấn đề rất quan trọng, đang thu hút nhiều người tập trung nghiên cứu và tìm mọi giải pháp để đảm bảo an toàn, an ninh cho hệ thống phần mềm, đặc biệt là các hệ thống thông tin trên mạng. Việc bảo mật cho hệ thống có thể thực hiện theo nhiều phương diện, ở nhiều tầng khác nhau, bao gồm từ phương diện kiểm soát truy nhập vật lý vào hệ thống; thực hiện sửa chữa, cập nhật, nâng cấp hệ điều hành cũng như vá mọi lỗ hổng về an ninh tìm thấy; quản lý các hoạt động trao đổi thông tin trên mạng (giám sát qua bức tường lửa, các bộ định vị Router, phát hiện và phòng ngừa sự xâm nhập, v.v.); xây dựng các giải pháp bảo mật ở mỗi phần mềm đến quản lý người dùng thông qua việc cấp quyền sử dụng, mật khẩu, mật mã, v.v.

Song, cũng cần phải hiểu rằng, không có một hệ thống thông tin nào được bảo mật 100%. Bất kỳ một hệ điều hành nào mà bạn đang sử dụng cũng có thể có những lỗ hổng về an ninh và chưa phát hiện ra. Bộ phận quản trị mạng cũng có thể không phát hiện nhanh được những xâm nhập trái phép có thể làm hư hỏng, thậm chí phá huỷ thông tin của bạn. Ngay cả những thuật toán mật mã nổi tiếng như MD5, SHA-1 cũng không thể khẳng định 100% mức độ an toàn, bởi vì có thể đến thời điểm này chưa phát hiện ra người có thể bẻ gãy được những thuật toán đó, nhưng cũng không có gì đảm bảo sẽ không có ai làm được điều đó trong tương lai.

Chính vì thế, ta cần phải làm hai việc. Việc thứ nhất là cần phải xác định *chính sách bảo mật* phù hợp cho hệ thống của mình. Chính sách bảo mật cho phép người dùng truy cập vào hệ thống theo những quyền đã được cấp và theo những thủ tục nhất định. Tuy nhiên, khi đưa ra một chính sách bảo mật thì ta lại phải trả giá cho những phí tổn về thời gian và tài nguyên vì những qui định bắt buộc (thường là bất tiện) đối với người sử dụng và những tính phiền phức của các thủ tục bảo mật. Việc thứ hai là phải xây dựng các chiến lược *phát hiện* và *phòng chống* các cuộc tấn công vào hệ thống.

Việc đầu tiên cần phải thực hiện khi lập kế hoạch bảo mật hệ thống là bạn phải xác định rõ các yêu cầu cần bảo mật, những gì bạn cần phải bảo vệ trước các cuộc tấn công, chính sách bảo mật phải dựa trên những cơ sở nào? Thông thường có hai loại yêu cầu bảo mật chung cho các hệ thống phần mềm trao đổi giữa các doanh nghiệp (B2B) [2].

- *Vấn đề bảo mật trong truyền thông*. Phần lớn các hệ thống của chúng ta hiện nay được kết nối mang. Khi thông tin được gửi đi trên mạng (Internet) thì nó phải được bảo vệ để chống lại những kẻ nghe (xem) trộm. Bất kỳ một người nào đó cũng có thể truy cập vào các cáp mạng hoặc qua bộ định vị Router để làm thay đổi nội dung trao đổi trên đường truyền. Ta không thể biết được liệu đã có một sự thay đổi nào đã được thực hiện đối với các nội dung trao đổi kể từ lúc chúng được gửi đi hay chưa. Ngay cả khi bạn tin rằng những nội dung thông tin trao đổi được bảo vệ chống những kẻ nghe trộm và những người lạ thì những giao thức truyền thông vẫn có thể còn có những lỗ hổng mà ta chưa phát hiện ra. Trong giao thức TCP/IP, người gửi gói tin được xác định thông qua địa chỉ IP và cung kết nối ở phần đầu của gói (Package Header) và chính các trường thông tin này có thể dễ dàng thay đổi được. Điều này dẫn tới vấn đề quan trọng là cần chứng thực, xác thực người gửi. Nói chung, có bốn yêu cầu cơ bản về bảo mật truyền thông.

- (i) *Đảm bảo tin cậy*. Các nội dung thông tin không bị theo dõi hoặc sao chép bởi những thực thể không được ủy thác.
- (ii) *Đảm bảo toàn vẹn*. Các nội dung thông tin không bị thay đổi bởi những thực thể không được ủy thác.
- (iii) *Chiứng minh xác thực*. Không ai có thể tự trả hình như là một bên hợp pháp trong quá trình trao đổi tin.
- (iv) *Không thể thoái thác trách nhiệm*. Người gửi tin không thể thoái thác về những sự việc và những nội dung thông tin mà thực tế họ đã gửi đi.

Sự phát triển của công nghệ mật mã cho phép ta thực hiện được ba yêu cầu đầu tiên. Ví dụ, bạn có thể sử dụng SSL/TLS [4], được định nghĩa bởi Netscape Communication Corporation cho việc bảo mật kết nối theo HTTP, để bảo mật chương trình ứng dụng dựa vào Java. Yêu cầu thứ tư có thể thực hiện được bằng chữ ký số.

- *Kiểm soát truy cập.* Khi nhận được một thông tin từ một kênh truyền được bảo mật, ta cần phải dựa vào chính sách bảo mật để biết được những thao tác nào là được phép thực hiện đối với những thông tin đó. Việc kiểm soát truy cập đôi khi cũng nhầm lẫn với việc xác thực. Trong nhiều trường hợp, cơ chế kiểm soát truy cập và cơ chế xác thực được tích hợp vào trong một cơ chế chung. Ví dụ, bạn có thể truy cập vào các trang Web của Apache Web Server thông qua các mật khẩu chỉ khi bạn đã được xác thực. Nhưng, nói chung ta cần phân biệt cơ chế kiểm soát truy cập với cơ chế xác thực. Xác thực được sử dụng để khẳng định sự đồng nhất của người tham gia trao đổi tin. Trong những hệ thống lớn, thường có một cơ sở dữ liệu xác thực chứa các định danh và mật khẩu của nhiều người dùng, nhưng chỉ tập con trong số họ được quyền truy cập vào những thành phần đặc biệt của hệ thống.

Như chúng ta đã biết, mục đích của công nghệ Java là “*Viết chương trình một lần và chạy ở mọi nơi*”. Tất nhiên, việc phát tán các applet là thực sự có ý nghĩa trong thực tế chỉ khi những người sử dụng được đảm bảo rằng các mã lệnh đó không bị thay đổi và không phá huỷ trên máy của họ. Vì vậy, vấn đề bảo mật thông tin được xem là vấn đề chính được đề cập bởi cả những người thiết kế lẫn người sử dụng công nghệ Java để phát triển các phần mềm ứng dụng phân tán.

Trong công nghệ Java có ba cơ chế đảm bảo an toàn dữ liệu [2].

- *Dựa vào những đặc trưng của ngôn ngữ như:* kiểm soát giới hạn của các cấu trúc dữ liệu như cấu trúc mảng, kiểm tra chặt chẽ sự chuyển đổi giữa các kiểu đối tượng, không sử dụng con trỏ số học, v.v.
- *Cơ chế điều khiển việc thực hiện của các mã lệnh* truy cập vào tệp, truy cập mạng, v.v.
- *Cơ chế hỗ trợ ký, xác nhận mã lệnh.* Người viết chương trình có thể sử dụng những thuật toán mã hoá chuẩn để xác thực các mã lệnh trong chương trình. Người dùng những chương trình này có thể xác thực được danh tính người tạo ra chúng và kiểm tra xem liệu nó đã bị thay đổi (lần cuối cùng) sau khi đã được ký xác nhận hay không.

Máy ảo Java JVM sẽ giúp chúng ta thực hiện được cơ chế thứ nhất, nghĩa là kiểm tra chỉ số các phần tử của mảng, sự tương thích về kiểu giữa các lớp đối tượng, .v.v.

Một khi các tệp lớp (.class) được nạp vào JVM, chúng sẽ được kiểm tra xem có nguyên vẹn hay không. Điều quan trọng là, ngoài bộ nạp lớp mặc định, ta có thể tạo ra các bộ nạp lớp Loader riêng để kiểm soát các hoạt động trong JVM giúp ta thực hiện được cơ chế thứ hai.

Để đảm bảo an toàn cao hơn, ta nên sử dụng cả những bộ nạp ClassLoader chung của hệ thống (mặc định) và những bộ nạp lớp riêng, kết hợp với lớp quản trị an ninh SecurityManager để kiểm soát sự hoạt động của các đoạn mã lệnh. Nói chung ta nên xây dựng những lớp quản trị an ninh riêng cho từng ứng dụng.

Cơ chế thứ ba có thể dễ dàng thực hiện bằng cách sử dụng các thuật toán mã được cung cấp bởi các lớp trong gói `java.security` hoặc xây dựng những thuật toán mã mới để ký nhận và xác thực chương trình. Cơ chế này sẽ được đề cập cụ thể ở phần cuối chương.

## 7.2. BỘ NẠP LỚP VÀ KIỂM TRA BYTE CODE

Chương trình dịch của Java chuyển đổi mã nguồn sang ngôn ngữ máy giả thuyết, máy ảo JVM. Mỗi lớp trong chương trình Java được tạo ra trong JVM là một tệp lớp có đuôi `.class`. Các tệp lớp này muốn thực hiện lại phải được thông dịch để chuyển các lệnh trong JVM sang mã máy của những máy đích [1].

Lưu ý rằng, chương trình thông dịch của JVM chỉ nạp những lớp cần thiết tại mỗi thời điểm để thực hiện chương trình. Ví dụ, ta xét quá trình bắt đầu thực hiện chương trình ứng dụng độc lập `MyProgram.class`. JVM sẽ thực hiện các bước như sau:

1. JVM có cơ chế để nạp các tệp lớp liên quan, đọc từ đĩa hay tải xuống từ những Website chứa các lớp đó. Nó sử dụng cơ chế này để nạp `MyProgram.class`.
2. Nếu lớp `MyProgram` có các trường dữ liệu hoặc kế thừa từ một lớp cha khác thì những lớp đó cũng được nạp về.
3. JVM sẽ thực hiện phương thức `main()` trong `MyProgram`, vì phương thức này là tĩnh nên không cần phải tạo ra đối tượng để gọi nó.
4. Nếu trong `main()` lại yêu cầu những đối tượng lớp khác thì chúng sẽ được nạp bổ sung theo yêu cầu.

Nói chung, ta không cần can thiệp vào quá trình trên, hệ thống tự động nạp và kiểm soát các lớp được tải xuống. Tuy nhiên, để hệ thống an toàn hơn, ta nên xây dựng bộ kiểm soát nạp lớp riêng. Nó cho phép ta kiểm tra tính toàn vẹn của các mã byte code trước khi đưa vào JVM.

### 7.2.1. Viết bộ nạp lớp ClassLoader riêng

Mỗi bộ nạp lớp đều cài đặt `ClassLoader`. Phương thức `loadClass()` ở lớp này sẽ xác định cách nạp lớp ở mức định. Một khi có một lớp được nạp thì tất cả các lớp khác mà nó tham chiếu tới đều được nạp bởi cùng một bộ nạp.

Để tạo ra được một bộ nạp, ví dụ `MyClassLoader`, ta phải viết đè phương thức

```
loadClass(String className, boolean resolve)
```

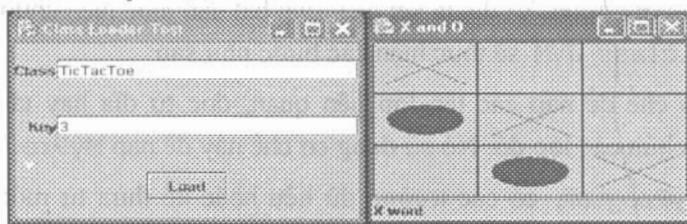
theo các bước như sau.

1. Kiểm tra xem bộ nạp lớp này đã được nạp hay chưa. Mục đích là bộ nạp lớp của ta chỉ phải lưu lại một bản ghi về những lớp mà nó đã nạp.

2. Nếu là lớp mới thì kiểm tra xem nó có phải là lớp hệ thống hay không. Trường hợp không phải là lớp hệ thống, các mã bytecode của lớp đó được nạp từ các hệ thống tệp hoặc từ những nguồn khác.
3. Gọi phương thức `defineClass()` của `ClassLoader` để giới thiệu các bytecode cho JVM.

Thông thường, bộ nạp lớp sử dụng một phép ánh xạ (thường là bảng băm) để lưu giữ các tham chiếu tới các lớp đã được nạp.

**Ví dụ 7.1.** Xây dựng bộ nạp lớp để nạp những tệp lớp đã được mã hoá. Người sử dụng phải nhập vào tên của lớp chính (lớp chứa hàm `main()`), một ứng dụng cần thực hiện và khóa mật mã. Chương trình sử dụng bộ nạp lớp riêng để nạp các lớp chỉ định và gọi hàm `main()`. Bộ nạp lớp sẽ giải mã các lớp được tải về.



Hình 7.1. Chương trình nạp và kiểm soát lớp

Người sử dụng cho biết tên tệp lớp chính, ví dụ chương trình `TicTacToe.class`, xác định lại khóa (mặc định là 3) rồi nhấn nút Load. Chương trình được nạp theo bộ nạp được xây dựng và hai người chơi có thể đánh cờ ca rô như hình 7.1.

```
// MyClassLoaderTest.java
import java.util.*;
import java.io.*;
import java.lang.reflect.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class MyClassLoaderTest{
    public static void main(String[] args){
        JFrame f = new MyClassLoaderFrame();
        f.show();
    }
}
class MyClassLoaderFrame extends JFrame{
    public MyClassLoaderFrame(){

```

```
setTitle("Class Loader Test");
setSize(300, 200);
addWindowListener(new WindowAdapter(){
    public void windowClosing(WindowEvent e){
        System.exit(0);
    }
});
getContentPane().setLayout(new GridBagLayout());
GridBagConstraints gbc = new GridBagConstraints();
gbc.weightx = 0;
gbc.weighty = 100;
gbc.fill = GridBagConstraints.NONE;
gbc.anchor = GridBagConstraints.EAST;
add(new JLabel("Class"), gbc, 0, 0, 1, 1);
add(new JLabel("Key"), gbc, 0, 1, 1, 1);
gbc.weightx = 100;
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.anchor = GridBagConstraints.WEST;
add(nameField, gbc, 1, 0, 1, 1);
add(keyField, gbc, 1, 1, 1, 1);
gbc.fill = GridBagConstraints.NONE;
gbc.anchor = GridBagConstraints.CENTER;
JButton loadButton = new JButton("Load");
add(loadButton, gbc, 0, 2, 2, 1);
loadButton.addActionListener(
    new ActionListener(){
        public void actionPerformed(ActionEvent event){
            runClass(nameField.getText(), keyField.getText());
        }
    });
}
public void add(Component c, GridBagConstraints gbc, int x, int y,
               int w, int h) {
    gbc.gridx = x;
```

```
        gbc.gridx = y;
        gbc.gridwidth = w;
        gbc.gridheight = h;
        getContentPane().add(c, gbc);
    }

    public void runClass(String name, String key){
        try{
            ClassLoader loader =
                new CryptoClassLoader(Integer.parseInt(key));
            Class c = loader.loadClass(name);
            String[] args = new String[]{};
            Method m = c.getMethod("main", new Class[]{args.getClass()});
            m.invoke(null, new Object[]{args});
        }catch(Throwable e){
            JOptionPane.showMessageDialog(this, e);
        }
    }

    private JTextField nameField = new JTextField(30);
    private JTextField keyField = new JTextField("3", 4);
}

// Xây dựng lớp nạp riêng
class CryptoClassLoader extends ClassLoader{
    private Map classes = new HashMap();
    private int key;
    public CryptoClassLoader(int k){
        key = k;
    }
    protected synchronized Class loadClass(String name, boolean resolve)
        throws ClassNotFoundException{
        // Kiểm tra xem lớp đã được nạp chưa?
        Class c1 = (Class)classes.get(name);
        if(c1 == null){
```

```

    // Lớp mới
    try{
        // Kiểm tra xem có phải lớp hệ thống không?
        return findSystemClass(name);
    }catch(ClassNotFoundException e){}
    catch(NoClassDefFoundError e){}
    // Nạp lớp theo từng byte, tùy vào từng bộ nạp lớp
    byte[] classBytes = loadClassBytes(name);
    if(classBytes == null) throw new ClassNotFoundException(name);
    c1 = defineClass(name, classBytes, 0, classBytes.length);
    if(c1 == null) throw new ClassNotFoundException(name);
    classes.put(name, c1);      // Ghi nhớ lớp được nạp
}
if(resolve) resolveClass(c1);
return c1;
}

private byte[] loadClassBytes(String name){
    // Nạp từng byte và mã hoá chúng theo thuật toán "caesar"
    String cname = name.replace('.', '/') + ".caesar";
    FileInputStream in = null;
    try{
        in = new FileInputStream(cname);
        ByteArrayOutputStream buff =
            new ByteArrayOutputStream();
        int ch;
        while((ch = in.read()) != -1){
            byte b = (byte)(ch - key);
            buff.write(b);
        }
        in.close();
        return buff.toByteArray();
    }catch(IOException e){
        if(in != null){

```

```

        try{
            in.close();
        }catch(IOException e1){
        }
        return null;
    }
}

```

java.lang.ClassLoader

API

- **Class defineClass(String name, byte data[], int offset, int length):** Đưa thêm một lớp mới vào JVM, trong đó có các tham số:
  - name:** là tên của một lớp có thể chứa cả tên của gói chứa nó, nhưng không cần chỉ rõ cả .class
  - data:** một mảng để chứa các byte code của lớp đó
  - offset:** byte code bắt đầu trên mảng
  - length:** độ dài của mảng.
- **void loadClass(String name, boolean resolve):** Được cài đặt để nhận được các byte code của lớp cần nạp vào JVM nếu resolve là true. Trong đó, các tham số:
  - name:** là tên của một lớp có thể chứa cả tên của gói chứa nó, nhưng không cần chỉ rõ cả .class
  - resolve:** có giá trị true nếu resolveClass() cần gọi để kiểm tra sau khi lớp được nạp.
- **Class findSystemClass(String name):** Tìm lớp hệ thống được nạp vào.
- **void resolveClass(Class c):** Được gọi thực hiện khi cờ resolve là true.

### 7.2.2. Kiểm tra byte code

Khi một bộ nạp lớp giới thiệu các byte code cho JVM thì trước tiên những byte code này phải được kiểm định bởi một *bộ kiểm tra*. Bộ kiểm tra đảm bảo rằng những lệnh được nạp vào khi thực hiện sẽ không gây ra thiệt hại nào cả. Tất cả các ngoại lệ của các lớp hệ thống sẽ được kiểm tra và được thông báo khi chúng xuất hiện.

Bộ kiểm tra có thể thẩm định:

- Các biến phải được khởi tạo giá trị trước khi chúng được sử dụng.
- Trong các lời gọi hàm, các kiểu tham chiếu phải phù hợp, tương thích với các tham số hình thức.
- Đảm bảo các luật truy cập vào các thành phần riêng không bị vi phạm.
- Đảm bảo chương trình khi thực hiện không bị tràn bộ nhớ, v.v.

Một khi phát hiện ra một sai sót bất kỳ thì lớp đó được xem là không hợp lệ và sẽ không được nạp vào JVM

Quan trọng hơn, trong thế giới mở với Internet, ta phải tìm cách bảo vệ để chống lại những người khác có tinh phá hoại. Ví dụ, họ có thể thay đổi kết quả tính toán của chương trình, thay đổi các trường dữ liệu riêng của các đối tượng trong hệ thống, hay một chương trình có thể bị ngắt bởi hệ thống *bao mật* của một trình duyệt

Song, bạn có thể phân vân tại sao không có một bộ kiểm tra đặc biệt để kiểm định tất cả những điều nêu trên?. Trước hết phải biết rằng, trong Java chương trình biên dịch luôn kiểm duyệt và không cho phép tạo ra những tệp lớp, trong đó có những biến được sử dụng nhưng chưa khởi tạo giá trị hay những biến riêng (*private*) mà lại bị các đối tượng của lớp khác truy nhập, v.v. Nhân đây cũng cần lưu ý là những vấn đề kiểm soát này không được đảm bảo ở những ngôn ngữ khác như C/C++, hay Pascal. Tuy nhiên, dạng byte code sử dụng trong tệp lớp là dạng tài liệu hoá và việc sửa nó không có gì khó khăn đối với những người lập trình Assembly có kinh nghiệm, họ có thể sử dụng những hệ soạn thảo hexa và sửa bằng tay để tạo ra các tệp lớp hợp lệ nhưng có những lệnh không an toàn trong JVM

Ví dụ, sau đây chỉ ra khả năng thay đổi tệp lớp và có thể dẫn đến những kết quả không mong muốn.

**Ví dụ 7.2.** Chương trình đơn giản chạy được cả độc lập lẫn Applet.

```
// VerifierTest.java
import java.awt.*;
import java.applet.*;
public class VerifierTest extends Applet{
    public static void main(String args[]){
        System.out.println("1 + 2 = " + fun());
    }
    static int fun(){
        int n, m;
        m = 1; n = 2;
        // Sử dụng hệ soạn thảo như Hex Workshop thay đổi "n = 2" bằng "m = 2"
        int r = m + n;
    }
}
```

```

        return r;
    }

    public void paint(Graphics g){
        g.drawString("1 + 2 = " + fun(), 20, 20);
    }
}

```

Chương trình thực hiện sẽ hiển thị kết quả lên màn hình

1 + 2 = 3

Nếu ta thay đổi hàm fun() ở ví dụ trên thành:

```

static int fun(){
    int n, m;
    m = 1;
    m = 2;
    int r = m + n;
    return r;
}

```

Trong trường hợp này, biến n chưa được khởi tạo giá trị, và do vậy nó có thể nhận giá trị ngẫu nhiên trong bộ nhớ mà những ngôn ngữ lập trình khác như C/C++, Pascal không kiểm soát được. Riêng đối với Java, chương trình dịch phát hiện ra vấn đề này và nó thông báo là có lỗi.

Để tạo ra một tệp lớp nhập nhằng, ta phải can thiệp vào các câu lệnh ở mức sâu hơn, mức mã lệnh. Trước tiên, sử dụng javap để dịch chương trình VerifierTest.java. Câu lệnh

```
javap -c VerifierTest
```

cho các byte code trong tệp lớp dưới dạng các câu lệnh ký hiệu gợi nhớ.

Phương thức int fun() đã được dịch

```

0  iconst_1
1  istore_0
2  iconst_2
3  istore_1
4  iload_0
5  iload_1
6  iadd
7  istore_2
8  iload_2
9  ireturn

```

Tiếp theo, sử dụng hệ soạn thảo hexa (như Hex Workshop) để thay đổi lệnh thứ ba là `istore_1` bằng `istore_0`. Bởi vì biến cục bộ `istore_0` ứng với `m`, do vậy nó được khởi tạo hai lần và biến `n` không được khởi tạo giá trị. Ghi lại tệp vừa sửa vào tên cũ `VerifierTest.class`.

Thử chạy chương trình `VerifierTest` với chế độ kiểm tra mặc định ta nhận được thông báo lỗi:

```
Exception in thread "main" java.lang.VerifyError: (class: VerifierTest, method: fun signature: ()I) Accessing value from uninitialized register 1
```

Nhưng, nếu ta chạy chương trình trên với tùy chọn `-noverify`,

```
java -noverify VerifierTest
```

nghĩa là không cần kiểm tra thì chương trình thực hiện và cho một kết quả ngẫu nhiên, ví dụ

```
1 + 2 = 15102330
```

vì `n` chưa được khởi tạo nên nó có thể nhận kết quả ngẫu nhiên trong bộ nhớ.

**Lưu ý:** Bộ kiểm tra luôn cảnh trừng để chống lại sự thay đổi ác ý đối với những tệp lớp, nhưng không kiểm tra những tệp được sinh bởi chương trình dịch.

### 7.3. LỚP SECURITYMANAGER VÀ PERMISSION

Như ở trên đã nêu, một lớp được nạp vào JVM bằng một bộ nạp lớp và được kiểm định bởi bộ kiểm tra. Cơ chế đảm bảo an ninh thứ ba trong môi trường Java được thực hiện bởi bộ quản lý *bảo mật* thông qua lớp `SecurityManager`. Nó kiểm tra xem những thao tác chỉ định có được phép hay không. Những thao tác cần kiểm tra tính an ninh bao gồm:

- Luồng hiện thời có thể tạo ra một bộ nạp lớp mới;
- Luồng hiện thời có thể tạo ra một tiến trình con;
- Luồng hiện thời có thể dừng JVM;
- Luồng hiện thời có thể truy cập vào các thành phần của lớp khác;
- Luồng hiện thời có thể truy cập hoặc làm thay đổi các thuộc tính của hệ thống;
- Luồng hiện thời có thể đọc hoặc ghi vào một tệp;
- Luồng hiện thời có thể xoá một tệp;
- Luồng hiện thời có thể kết nối với một Socket từ một máy xác định thông qua số hiệu của cổng;
- Luồng hiện thời có thể phải chờ, hay mở kết nối với một Socket từ một máy xác định thông qua số hiệu của cổng;
- Luồng hiện thời có thể gọi các phương thức: `stop()`, `suspend()`, `resume()`, `setPriority()`, `setName()`, `setDaemon()` của một luồng hoặc của một nhóm luồng;

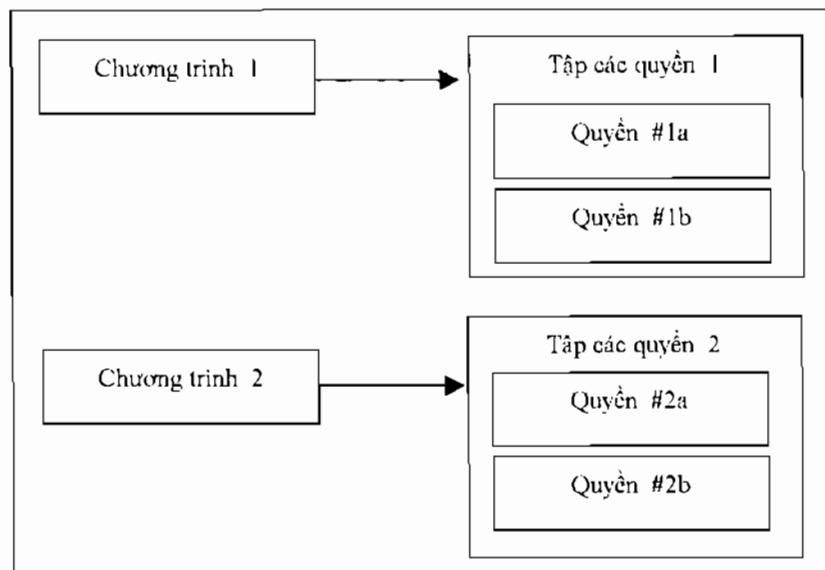
- Luồng hiện thời có thể bắt đầu in một tệp;
- Luồng hiện thời có thể truy cập vào vùng nhớ giành riêng của hệ thống;
- v.v.

Khi chạy chương trình ứng dụng Java, mặc định nó được thực hiện không có quản lý an ninh. Muốn cài đặt nó trong chương trình ứng dụng phải gọi phương thức `setSecurityManager()` của lớp `System`.

### 7.3.1. Vấn đề bảo mật trong Java 2 Platform

Các JDK 1.0, 1.1 có mô hình bảo mật khá đơn giản: các đối tượng trong chương trình ứng dụng độc lập có toàn quyền truy cập tới các tài nguyên cục bộ, còn bộ quản lý an ninh của các applet từ chối mọi truy cập tới các tài nguyên cục bộ.

Java 2 Platform có cơ chế quản lý an ninh linh hoạt hơn. Nó sử dụng một *chính sách bảo mật* để cấp quyền (giấp phép) được phép thực hiện cho từng chương trình khác nhau.



**Hình 7.2. Chính sách cấp quyền được thực hiện cho các chương trình**

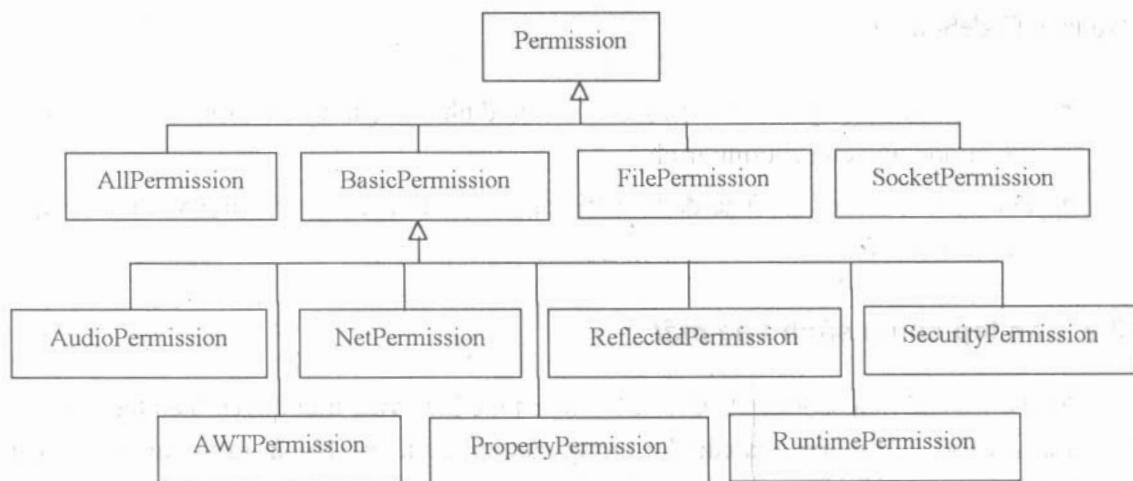
Trong đó, *quyền* là một đặc tính được phép thực hiện và được kiểm soát bởi bộ quản lý an ninh, còn được gọi là *đặc quyền*. JDK 1.2 hỗ trợ tạo ra những lớp để tạo ra các quyền cho chương trình. Ví dụ

`FilePermission p = new FilePermission("/tmp/*", "read, write");`  
cho phép đọc, ghi một tệp bất kỳ ở thư mục /tmp.

Các lớp quản lý quyền thao tác trong JDK 1.2 tạo ra một cấu trúc phân cấp như hình 7.3.

JDK cung cấp mô hình chuẩn để kiểm tra quyền thao tác dựa trên hai lớp:

`java.security.SecurityClassLoader`  
`java.lang.SecurityManager`



Hình 7.3. Cấu trúc phân cấp của các lớp Permission trong JDK 1.2

Ngoài ra, mô hình chuẩn còn dựa vào đối tượng của lớp Policy để cấp quyền thao tác cho các chương trình nguồn. Tại mỗi thời điểm chỉ có một đối tượng của Policy được quyền vận dụng. Ta có thể sử dụng

```
Policy currentPolicy = Policy.getPolicy()
```

để xác định chính sách hiện thời được phép thực hiện của chương trình!

Chi tiết hơn về vấn đề *bảo mật* với Java, nhất là mô hình *bảo mật* cơ sở của Java ứng dụng trong việc phát triển những hệ thống nhúng, thẻ thông minh, v.v., bạn có thể đọc ở tài liệu trực tuyến <http://www.securingjava.com>.

#### java.lang.SecurityManager

#### API

- void checkPermission(Permission p)
  - void checkPermission(Permission p, Object context)
- Kiểm tra xem chính sách *bảo mật* hiện thời có cho phép hay không.

#### java.lang.SecurityManager

#### API

- static Policy getPolicy(): Xác định chính sách hiện thời.
  - PermissionCollection getPermissions(CodeSource source)
- Lấy lại quyền được thao tác đối với source cho trước.

#### java.lang.PermissionCollection

#### API

- void add(Permission p): Bổ sung thêm quyền p vào tuyển tập các quyền cấp trước.
- Enumeration elements(): Lấy ra dãy liệt kê các quyền trong tuyển tập.

- Certificate[] getCertificates(): Xác định các chứng chỉ đối với tệp lớp ứng với nguồn gốc của chương trình.
- URL getLocation(): Xác định vị trí (địa chỉ) của các tệp lớp ứng với nguồn gốc của chương trình.

### 7.3.2. Các tệp chính sách bảo mật

Như ở trên đã nêu, SecurityClassLoader thực hiện việc trao quyền thao tác cho các lớp được nạp vào JVM thông qua các đối tượng của lớp Policy. Ngoài ra ta còn có thể cài đặt một lớp Policy riêng để cấp quyền cho chương trình. Ở chương V ta cũng đã sử dụng tệp chính sách (.policy) để có được quyền truy cập từ xa vào các máy tính trên mạng. Ví dụ, một tệp MyApp.policy mẫu có dạng

```
grant codeBase www.horstmann.com/classes{
    permission java.io.FilePermission "/tmp/*", "read,write";
}
```

cấp quyền đọc, ghi các tệp vào thư mục /tmp cho tất cả các mã được nạp từ www.horstmann.com/classes. Ta có thể sử dụng bất kỳ hệ soạn thảo nào, như Notepad, Office Word, Jcreator, v.v, để soạn thảo các tệp chính sách (dưới dạng tệp văn bản).

#### Lưu ý:

1. Lớp chính sách mặc định được đặt trong tệp java.security ở thư mục con /jre/lib của JDK. Theo mặc định, tệp này chứa dòng
 

```
policy.provider=sun.security.provider.policyFile
```
2. Ta có thể thay đổi các vị trí của các tệp chính sách trong java.security. Tệp chính sách của java: java.policy được đặc tả mặc định là:
 

```
policy.url.1=file:${java.home}/lib/security/java.policy
policy.url.2=file:${user.home}/java.policy
```

 Người quản trị hệ thống có thể thay đổi tệp java.policy và chỉ rõ những địa chỉ URL thường trực trên những Server khác.

Thực hiện MyApp với quyền được cấp trong tệp MyApp.policy như sau:

```
java -Djava.security.policy = MyApp.policy MyApp
```

Tương tự đối với Applet, ta thực hiện

```
appletviewer -J-Djava.security.policy=MyApplet.policy
MyApplet.html
```

Tệp chính sách chứa một dãy các mục được đảm bảo. Mỗi mục có dạng như sau:

```

grant codesource
{
    permission-1;
    permission-2;
    ...
};


```

Trong đó,

- codesource là cơ sở của mã lệnh codeBase (có thể không cần khai báo nếu mục đầu vào áp dụng cho tất cả các nguồn) và tên của những người ký giấy chứng nhận (có thể bỏ qua nếu không có yêu cầu về người ký xác nhận).
- + codeBase xác định nơi chứa các lớp sẽ được tải về:

codeBase "url"

- Nếu URL kết thúc bằng '/', nghĩa là nó chỉ tới một thư mục, ngược lại là tên của một tệp JAR. Ví dụ,

grant codeBase "www.horstman.com/classes/" {...}

grant codeBase "www.horstman.com/classes/MyApp.jar" {...}

- codeBase là một địa chỉ URL và chứa phần tử ngăn cách '/' giữa các tệp, ngay cả đổi với URL trong Window, ví dụ

grant codeBase "file:d:users/myapps/classes" {...}

- Các đặc quyền permission có dạng cấu trúc như sau:

permission className targetName, actionList

- + className là tên đầy đủ của lớp đặc quyền thực hiện, ví dụ như java.net.NetPermission, java.net.SocketPermission, java.io.FilePermission, java.security.SecurityPermission, java.util.PropertyPermission, java.awt.AWTPermission, java.security.AllPermission

- + targetName là tên tệp hay thư mục được đặc quyền thực hiện, có dạng:

fileName	Một tệp
directory/	Một thư mục
directory/*	Tất cả các tệp của thư mục directory
*	Tất cả các tệp của thư mục hiện thời
directory/-	Tất cả các tệp của thư mục directory hoặc ở một thư mục con nào đó của nó
-	Tất cả các tệp của thư mục hiện thời hoặc ở một thư mục con nào đó của nó
<<ALL FILES>>	Tất cả các tệp trong hệ thống.

+ actionList là danh sách các hành động như: read, write, delete, execute, accept, connect, listen, resolve.

Ví dụ:

```
grant codeBase "file:d:/myapps/classes/"{
    permission java.io.FilePermission "/myapp/-", "read, write";
}
```

**Lưu ý:** Một số lớp đặc quyền không cần chỉ ra targetName và actionList, ví dụ java.security.AllPermission.

Lớp đặc quyền java.net.NetPermission yêu cầu tên và cổng của máy kết nối dạng:

hostname hoặc IPaddress	Máy chủ
localhost hoặc xâu rỗng	Máy khách
*.domainSuffix	Hậu tố miền của máy thực hiện
*	Cho tất cả các máy.

Các cổng kết nối có thể liệt kê theo đây:

:n	Một cổng n
:n-	Tất cả được đánh số từ n trở lên
:-n	Tất cả được đánh số từ n trở xuống
:n1-n2	Tất cả được đánh số từ n1 đến n2

Ví dụ,

```
permission java.net.SocketPermission "*.*.horstman.com:8000-8999", "connect";
```

Lớp đặc quyền java.util.PropertyPermission khai báo về các thuộc tính đích cần thực hiện có hai dạng chính:

property	Chỉ rõ một đặc tính
propertyPrefix.*	Tất cả các đặc tính có cùng tiếp đầu ngữ.

Ví dụ “java.home” hay “java.vm.\*”.

```
java.util.PropertyPermission "java.vm.*", "read";
```

cho phép chương trình đọc tất cả các thuộc tính bắt đầu bằng java.vm

Ngoài ra, ta có thể sử dụng các thuộc tính của hệ thống trong các tệp chính sách. Dấu hiệu \${property} có thể được thay thế bằng giá trị xác định. Ví dụ, \${user.home} được thay thế bằng thư mục của người sử dụng.

```
permission java.io.FilePermission "${user.home}", "read, write";
```

Để tạo ra các tệp chính sách độc lập với các nền thì phải sử dụng thuộc tính file.separator thay thế cho ‘/’ hoặc ‘\\’. Người ta sử dụng ký hiệu rút gọn \${/} thay thế cho \${file.separator}. Ví dụ

```
permission java.io.FilePermission "${user.home}${/}-", "read, write";
```

cho phép đọc, ghi tất cả các tệp của thư mục của người sử dụng và bất kỳ một thư mục con của thư mục đó.

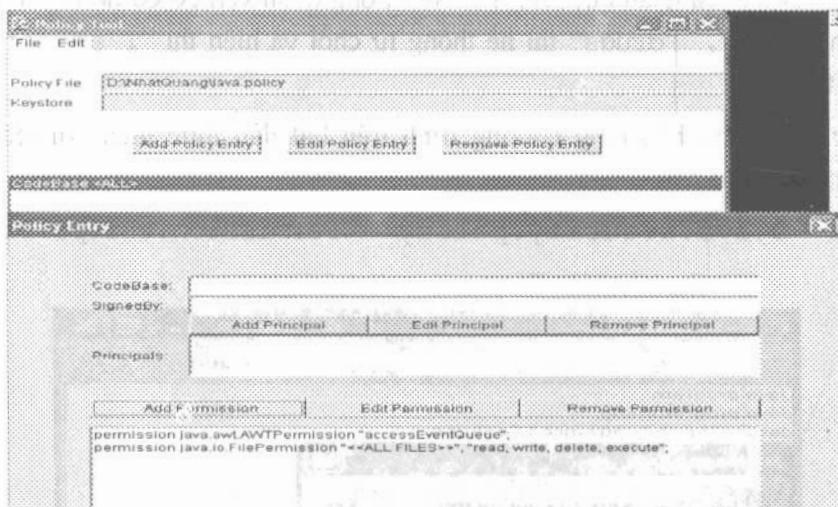
- Ta phải sử dụng “\” thay cho “\” trong Window.

```
permission java.io.FilePermission "d:\\myapps\\-", "read, write, delete";
```

JDK cung cấp công cụ policytool để ta sử dụng mà soạn thảo các tệp chính sách. Khi bắt đầu sử dụng policytool ta có thể tạo ra một tệp chính sách mới, ví dụ tệp java.policy ở thư mục NhatQuang.

```
d:\\NhatQuang\\>policytool java.policy
```

policytool sẽ hiển thi tất cả các chức năng cho phép ta lựa chọn các đặc quyền, rồi gán cho chúng các đặc tính, hành động như trên đã nêu. Ta có thể bổ sung thêm những đặc quyền mới, sửa chữa chúng hoặc loại bỏ đi những đặc tính không có nhu cầu, v.v., như hình 7.4.



Hình 7.4. Soạn thảo tệp chính sách với policytool

**Ví dụ 7.4.** Xây dựng một lớp để kiểm soát việc ghi chèn thêm các từ vào vùng văn bản. Chương trình phải đảm bảo rằng những từ “xấu”, như các từ “sex”, “drugs”, v.v., không được phép ghi chèn vào văn bản. Để thực hiện được yêu cầu này, ta có thể xây dựng lớp cấp quyền tùy biến, lớp WordCheckTextArea như sau.

```
class WordCheckTextArea extends JTextArea{
    public void append(String text){
        WordCheckPermission p =
            new WordCheckPermission(text, "insert");
        SecurityManager manager = System.getSecurityManager();
        if (manager != null) manager.checkPermission(p);
        super.append(text);
    }
}
```

Nếu bộ quản lý đảm bảo được an ninh cho WordCheckTextArea thì những từ được chấp nhận sẽ được phép chèn vào văn bản.

Việc kiểm tra các từ với hai hành động cụ thể: insert (được phép chèn) và avoid (cho phép bổ sung một văn bản bất kỳ nhưng loại bỏ những từ “xấu”, bị cấm). Do vậy, chương trình trên phải thực hiện với tệp .policy sau:

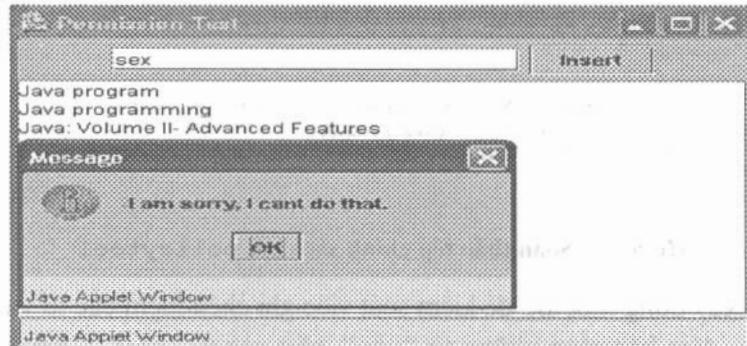
```
grant {
    permission WordCheckPermission "sex, drugs", "avoid";
};
```

cho phép chèn thêm các từ khác với những từ được liệt kê: "sex", "drugs".

Như vậy, giao diện của chương trình trên có dạng như hình 7.5. Người sử dụng nhập vào các dòng văn bản và nhấn nút Insert. Nếu trong đoạn văn đó có những từ không được phép nhập như "sex", "drugs" thì hệ thống từ chối và hiển thị "I am sorry, but I cannot do that.".

**Lưu ý:** Phải đảm bảo rằng chương trình trên bắt đầu thực hiện với tệp chính sách PermissionTest.policy.

```
java -Djava.security.policy = PermissionTest.policy
      PermissionTest
```



Hình 7.5. Chương trình kiểm duyệt các từ được chèn vào văn bản

```
// PermissionTest.java
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.net.*;
import java.security.*;
import javax.swing.*;
```

```
public class PermissionTest{
    public static void main (String args[]){
        System.setSecurityManager(new SecurityManager());
        JFrame frame = new PermissionTestFrame();
        frame.show();
    }
}

class PermissionTestFrame extends JFrame{
    public PermissionTestFrame(){
        setTitle("Permission Test");
        setSize(400, 300);
        addWindowListener(new WindowAdapter(){
            public void windowClosing (WindowEvent e){
                System.exit(0);
            }
        });
        textField = new JTextField(20);
        JPanel panel = new JPanel();
        panel.add(textField);
        JButton openButton = new JButton("Insert");
        panel.add(openButton);
        openButton.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                insertWords(textField.getText());
            }
        });
        Container contentPane = getContentPane();
        contentPane.add(panel, "North");
        textArea = new WordCheckTextArea();
        contentPane.add(new JScrollPane(textArea), "Center");
    }

    private JTextField textField;
    private WordCheckTextArea textArea;
```

```
public void insertWords (String words) {
    try{
        textArea.append(words + "\n");
    }
    catch (SecurityException e){
        JOptionPane.showMessageDialog(this, "I am sorry, I cant do that.");
    }
}

class WordCheckTextArea extends JTextArea{
    public void append(String text){
        WordCheckPermission p = new WordCheckPermission(text, "insert");
        SecurityManager manager = System.getSecurityManager();
        if (manager != null) manager.checkPermission(p);
        super.append(text);

    }
}

// WordCheckPermission.java
import java.security.*;
import java.util.*;

public class WordCheckPermission extends Permission{
    private String action;
    public WordCheckPermission(String target, String anAction){
        super(target);
        action = anAction;
    }

    public String getActions(){
        return action;
    }
}
```

```
public boolean equals(Object other) {
    if (other == null) return false;
    if (!getClass().equals(other.getClass())) return false;
    WordCheckPermission b = (WordCheckPermission)other;
    if (!action.equals(b.action)) return false;
    if (action.equals("insert")) return getName().equals(b.getName());
    else if (action.equals("avoid"))
        return badWordSet().equals(b.badWordSet());
    else return false;
}

public int hashCode(){
    return getName().hashCode() + action.hashCode();
}

public boolean implies(Permission other){
    if (!(other instanceof WordCheckPermission)) return false;
    WordCheckPermission b = (WordCheckPermission)other;
    if (action.equals("insert")){
        return b.action.equals("insert")
            && getName().indexOf(b.getName()) >=0;
    }
    else if (action.equals("avoid")){
        if (b.action.equals("avoid"))
            return b.badWordSet().containsAll(badWordSet());
        else if (b.action.equals("insert")){
            Iterator iter = badWordSet().iterator();
            while (iter.hasNext()){
                String badWord = (String)iter.next();
                if (b.getName().indexOf(badWord) >= 0)
                    return false;
            }
            return true;
        }
        else return false;
    }
}
```

```

        }
        else return false;
    }

    public Set badWordSet(){
        StringTokenizer token =
            new StringTokenizer(getName(), ",");
        Set set = new HashSet();
        while (token.hasMoreTokens())
            set.add(token.nextToken());
        return set;
    }
}

```

### 7.3.3. Bộ quản lý bảo mật tùy biến

Trong phần này ta xây dựng bộ quản lý *bảo mật* tùy biến đơn giản, đó là lớp CheckSecurityManager. Nó kiểm soát việc truy cập vào tệp, đảm bảo rằng những tệp văn bản (.text) mà nội dung của nó có các từ bị cấm thì không được phép truy cập.

Ta kiểm soát truy cập vào tệp bằng cách viết đè phương thức checkPermission() của lớp SecurityManager.

**Ví dụ 7.5.** Chương trình kiểm duyệt đọc các nội dung của các tệp. Mở một tệp .text và đọc nội dung của nó. Nếu nội dung của tệp văn bản không chứa các từ bị cấm thì cho phép truy cập.

```

// CheckSecurityManager.java
import java.security.*;
import java.io.*;

public class CheckSecurityManager extends SecurityManager{
    private String[] badWords = {"sex", "drugs", "C++"};

    public void checkPermission(Permission p){
        if(p instanceof FilePermission
            && p.getActions().equals("read")){
            if(inSameManager()) return;
            String fileName = p.getName();
            if(containsBadWords(fileName)) throw

```

```
        new SecurityException("Bad words in " + fileName);
    }
    else super.checkPermission(p);
}
boolean inSameManager(){
    Class[] cc = getClassContext();
    // Bỏ qua tập các lời gọi tới bộ quản lý
    int i = 0;
    while(i < cc.length && cc[0] == cc[i])
        i++;
    // Kiểm tra xem có lời gọi khác tới bộ quản lý không
    while(i < cc.length){
        if(cc[0] == cc[i]) return true;
        i++;
    }
    return false;
}

boolean containsBadWords(String fileName){
    if (!fileName.toLowerCase().endsWith("text")) return false;
    // Chỉ kiểm tra tệp .text
    BufferedReader in = null;
    try{
        in = new BufferedReader(new FileReader(fileName));
        String s;
        while((s = in.readLine()) != null){
            for(int i = 0; i < badWords.length; i++)
                if(s.toLowerCase().indexOf(badWords[i]) != -1)
                    return true;
        }
        in.close();
        return false;
    }catch(IOException e){
        return true;
    }
}
```

```
        }finally{
            if(in != null)
                try{in.close();}
            }catch(IOException e){
            }
        }
    }
}

// SecurityManagerTest.java
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.net.*;
import javax.swing.*;

public class SecurityManagerTest{
    public static void main (String args[]){
        System.setSecurityManager(new CheckSecurityManager());
        JFrame frame = new SecurityManagerTestFrame();
        frame.show();
    }
}

class SecurityManagerTestFrame extends JFrame{
    private JTextField fileNameField;
    private JTextArea fileText;
    public SecurityManagerTestFrame(){
        setTitle("Security Manager Test");
        setSize(400, 300);
        addWindowListener(new WindowAdapter(){
            public void windowClosing (WindowEvent e){
                System.exit(0);
            }
        });
    }
}
```

```

        fileNameField = new JTextField(20);
        JPanel panel = new JPanel();
        panel.add(new JLabel("Text File: "));
        panel.add(fileNameField);
        JButton openButton = new JButton("Open");
        panel.add(openButton);
        openButton.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                loadFile(fileNameField.getText());
            }
        });
        Container contentPane = getContentPane();
        contentPane.add(panel, "North");
        fileText = new JTextArea();
        contentPane.add(new JScrollPane(fileText), "Center");
    }

    public void loadFile(String fileName){
        try{
            fileText.setText("");
            BufferedReader in =
                new BufferedReader(new FileReader(fileName));
            String s;
            while((s = in.readLine()) != null)
                fileText.append(s + "\n");
            in.close();
        }catch(IOException e){
            fileText.append("I am sorry, I cant do that.");
        }
    }
}

```

Để tách tệp CheckSecurityManager.class ra khỏi các tệp lớp khác, ta có thể tạo ra một tệp .jar để lưu giữ tệp đó.

```
jar cvf FileCheck.jar CheckSecurityManager.class
```

Lệnh này đồng thời xóa CheckSecurityManager.class khỏi thư mục hiện thời.

Tương tự như ví dụ trước, để chạy được chương trình trên thì phải tạo ra tệp chính sách, tệp CheckSecurity.policy dạng:

```
grant codeBase "file:FileCheck.jar"{
    permission java.security.AllPermission;
}
```

Chính sách này đảm bảo các lớp trong FileCheck.jar được thực hiện hầu như tất cả mọi đặc quyền.

Sau cùng, ta thực hiện chương trình nêu trên như sau:

```
java -Djava.security.policy=CheckSecurity.policy
      -classpath FileCheck.jar;. CheckSecurityManager.java
```

`java.lang.SecurityManager`

API

- `Class[] getClassContext()`  
Trả lại một mảng các lớp cho các phương thức đang thực hiện.
- `void checkCreateClassLoader()`  
Kiểm tra xem luồng hiện thời có tạo ra một bộ nạp lớp hay không.
- `void checkAccess(Thread g)`  
Kiểm tra xem luồng hiện thời có thể gọi các phương thức stop(), suspend(), resume(), setName(), setDaemon() của luồng g.
- `void checkExit(int status)`  
Kiểm tra xem luồng hiện thời có thể thoát khỏi JVM với trạng thái status.
- `void checkExec(String cmd)`  
Kiểm tra xem luồng hiện thời có thể thoát khỏi JVM với trạng thái status.
- `void checkRead(FileDescriptor fd)`
- `void checkRead(String file)`
- `void checkWrite(FileDescriptor fd)`
- `void checkWrite(String file)`
- `void checkDelete(String file)`  
Kiểm tra xem luồng hiện thời có thể đọc, ghi, xoá tệp.

## 7.4. VĂN ĐỀ BẢO MẬT TRONG GÓI JAVA.SECURITY

Trong hơn 50 năm qua, các nhà toán học, tin học đã phát triển nhiều thuật toán rất tinh tế đảm bảo tính toàn vẹn của dữ liệu và chữ ký điện tử. Gói `java.security` đã cài đặt khá nhiều trong số các thuật toán đó. Rất may là ta không cần phải hiểu tường tận cơ sở toán học mà vẫn có thể sử dụng được chúng. Sau đây chúng ta tìm hiểu cách để phát hiện ra những sự thay đổi trong các tệp dữ liệu và cách xác thực chữ ký số.

### 7.4.1. Dấu vết thông điệp

*Tóm tắt đặc trưng của thông điệp là một dạng dấu vết tóm tắt của một khối dữ liệu, một thông điệp, gọi tắt là bản dấu vết thông điệp (message digest). Mỗi bản dấu vết thông điệp sẽ có hai đặc tính:*

1. Khi bản lưu trữ dữ liệu có một hay một số bit bị thay đổi thì bản tóm tắt dấu vết cũng sẽ bị thay đổi,
2. Rất khó (hầu như không thể) kiến thiết được một thông điệp mới (một văn bản) có cùng một bản tóm tắt dấu vết như bản gốc.

Đặc tính thứ hai có thể thực hiện được một cách ngẫu nhiên, nhưng với một xác suất rất thấp.

Hiện nay có một số thuật toán nén dữ liệu để tạo ra bản dấu vết thông điệp. Hai thuật toán nổi tiếng nhất là SHA-1 (do National Institute of Standard and Technology đề xuất) nén một khối dữ liệu bất kỳ thành dãy 160 bit (20 byte) và MD5 (của Ronald Rivest ở MIT) [2].

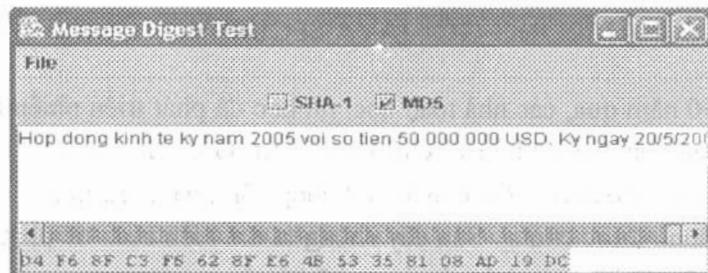
Java cài đặt cả hai thuật toán trên. Lớp `MessageDigest` được xem như một cái máy sản xuất ra các đối tượng thực hiện các thuật toán xử lý các tóm tắt dấu vết. Ví dụ, để nhận được đối tượng thực hiện thuật toán SHA-1, ta viết

```
MessageDigest alg = MessageDigest.getInstance("SHA-1");
```

Sau đó truyền các byte của một thông điệp cho đối tượng `alg` để nó tạo ra bản tóm tắt dấu vết. Ví dụ, muốn tạo ra bản tóm tắt dấu vết cho tệp có tên là `fileName`, ta thực hiện như sau.

```
FileInputStream in = new FileInputStream(fileName);
int ch;
while((ch = in.read()) != -1)
    alg.update((byte)ch);
```

**Ví dụ 7.6.** Chương trình sử dụng thuật toán SHA-1 hoặc MD5 để tạo ra bản tóm tắt dấu vết của một tệp bất kỳ hoặc một thông điệp được gõ trực tiếp từ bàn phím.



Hình 7.6. Chương trình sinh bản tóm tắt dấu vết thông điệp

Người sử dụng có thể nhập vào một đoạn văn bản như: “Hop dong kinh te ky nam 2005 voi so tien 500 000 000 USD. Ky ngay 20/5/2005”, chọn thuật toán cần thực hiện, sau đó chọn File rồi Text area digest thì bản tóm tắt dấu vết tương ứng sẽ được sinh ra ở hàng dưới của khung cửa sổ là: D4 F6 8F C3 F6 62 8F E6 4B 53 35 81 08 AD 19 DC (hình 7.6).

Nếu ta sửa đi một chữ nào, ví dụ chữ số 50 000 000 thành 20 000 000 thì bản dấu vết sẽ cho là: 39 17 7D 82 8F DD FD 94 54 72 64 1B F3 2F 51 89. Hai bản dấu vết là khác nhau, ta phát hiện ra đoạn văn bản trên đã bị sửa.

Tương tự, ta có thể chọn một tệp bất kỳ trong mục chọn File\File Digest để sinh ra bản dấu vết nhằm phát hiện ra những dữ liệu đã bị sửa.

```
// MessageDigestTest.java
import java.awt.*;
import java.awt.event.*;
import java.security.*;
import java.io.*;
import javax.swing.*;

public class MessageDigestTest{
    public static void main (String args[]){
        System.setSecurityManager(new CheckSecurityManager());
        JFrame frame = new MessageDigestFrame();
        frame.show();
    }
}
class MessageDigestFrame extends JFrame{
    private JTextField digest = new JTextField();
    private JTextArea message = new JTextArea();
    private MessageDigest currentAlgorithm;
```

```
public MessageDigestFrame(){
    setTitle("Message Digest Test");
    setSize(400, 200);
    addWindowListener(new WindowAdapter(){
        public void windowClosing (WindowEvent e){
            System.exit(0);
        }
    });
}

JPanel panel = new JPanel();
ButtonGroup group = new ButtonGroup();
ActionListener listener =
    new ActionListener(){
        public void actionPerformed(ActionEvent e){
            JCheckBox b = (JCheckBox)e.getSource();
            setAlgorithm(b.getText());
        }
    };
addCheckBox(panel, "SHA-1", group, true, listener);
addCheckBox(panel, "MD5", group, false, listener);

Container contentPane = getContentPane();

contentPane.add(panel, "North");
contentPane.add(new JScrollPane(message), "Center");
contentPane.add(digest, "South");
digest.setFont(new Font("Monospaced", Font.PLAIN, 12));

setAlgorithm("SHA-1");      // Đặt mặc định, chọn SHA-1

JMenuBar menuBar = new JMenuBar();
JMenu menu = new JMenu("File");
JMenuItem fileItem = new JMenuItem("File digest");
fileItem.addActionListener(
    new ActionListener(){
```

```
        public void actionPerformed(ActionEvent e) {
            loadFile();
        }
    });
    menu.add(fileItem);
    JMenuItem text = new JMenuItem("Text area digest");
    text.addActionListener(
        new ActionListener(){
            public void actionPerformed(ActionEvent e) {
                String m = message.getText();
                computeDigest(m.getBytes());// Tính bัน đầu vết
            }
        });
    menu.add(text);
    menuBar.add(menu);
    setJMenuBar(menuBar);
}

// Đọc nội dung của tệp và tính bัน đầu vết
public void loadFile(){
    JFileChooser chooser = new JFileChooser();
    chooser.setCurrentDirectory(new File("."));

    int r = chooser.showOpenDialog(this);
    if(r == JFileChooser.APPROVE_OPTION){
        String name =
            chooser.getSelectedFile().getAbsolutePath();
        computeDigest(loadBytes(name));
    }
}

public void addCheckBox(Container c, String name,
    ButtonGroup g, boolean selected, ActionListener listener){
    JCheckBox b = new JCheckBox(name, selected);
    c.add(b);
}
```

```
        g.add(b);
        b.addActionListener(listener);
    }

// Xác định thuật toán theo lựa chọn của người sử dụng
public void setAlgorithm(String alg){
    try{
        currentAlgorithm = MessageDigest.getInstance(alg);
        digest.setText("");
    }catch(NoSuchAlgorithmException e){
        digest.setText("") + e;
    }
}

public void computeDigest(byte[] b){
    currentAlgorithm.reset();
    currentAlgorithm.update(b);
    byte[] hash = currentAlgorithm.digest();
    String d = "";
    for(int i = 0; i < hash.length; i++){
        int v = hash[i] & 0xFF;
        if(v < 16) d += "0";
        d += Integer.toString(v, 16).toUpperCase() + " ";
    }
    digest.setText(d);
}

public byte[] loadBytes(String name){
    FileInputStream in = null;
    try{
        in = new FileInputStream(name);
        ByteArrayOutputStream buff = new ByteArrayOutputStream();
        int ch;
        while((ch = in.read()) != -1)
            buff.write(ch);
        return buff.toByteArray();
    }
```

```

        } catch (IOException e) {
            if (in != null) {
                try { in.close(); }
                ... } catch (IOException e1) { ... }
            }
        }
    }

}

// MessageDigest.policy
grant{
    permission java.security.AllPermission;
};

}

```

Sau khi dịch và chạy chương trình:

```
java -Djava.security.policy=MessageDigest.policy MessageDigestTest
```

**Lưu ý:** Giống như dấu vân tay, hy vọng rằng hầu như sẽ không có hai thông điệp có cùng một bản tóm tắt dấu vết. Tuy nhiên, hy vọng đó không hiện thực về mặt lý thuyết, vì như trong thuật toán SHA-1 “chỉ có” khoảng  $2^{160}$  bộ tóm tắt dấu vết có thể tạo ra, nên vẫn có thể có một số thông điệp khác nhau cho cùng một bản tóm tắt dấu vết. Tuy nhiên, xác suất trùng lặp ở đây là rất thấp, nên ta có thể yên tâm sử dụng những thuật toán trên để phát hiện những thông tin bị thay đổi trong các thông điệp.

java.security.MessageDigest

API

- static MessageDigest getInstance(String alg)  
Cho lại đối tượng của MessageDigest cài đặt thuật toán alg.
- void update(byte input)
- void update(byte[] input)
- void update(byte[] input, int offset, int len)  
Cập nhật lại bản dấu vết đối với các byte đầu vào.
- void reset(): Tính lại bản dấu vết.

#### 7.4.2. Chữ ký số

Trong phần trước ta đã tìm hiểu cách sinh ra bản tóm tắt dấu vết cho một thông điệp. Khi một thông điệp bị thay đổi thì bản tóm tắt dấu vết được sinh ra sẽ không tương thích với

bản gốc, do vậy dễ dàng phát hiện ra là nó đã bị thay đổi. Nếu ta gửi đi một bản tin và bản tóm tắt dấu vết của nó một cách riêng biệt, người nhận có thể sử dụng bản dấu vết đó để kiểm tra xem bản tin mình nhận có phải là bản gốc hay không. Nhưng nếu những kẻ gian, bằng một cách nào đó có được cả bản tin và bản dấu vết, họ có thể dễ dàng sửa đổi bản tin, tính lại bản dấu vết tương ứng. Điều này hoàn toàn thực hiện được bởi vì các thuật toán tính bản dấu vết là công khai, hoàn toàn không có gì bí mật cả. Trong trường hợp này, người nhận bản tin đã bị sửa và khi tính bản dấu vết (cũng bị sửa theo) có thể sẽ không phát hiện được là bản tin đó đã bị sửa.

Để chống lại những sự tấn công nêu trên, ta có thể sử dụng thêm một số phương pháp bảo mật, xác thực khác, ví dụ sử dụng chữ ký số.

Trước tiên ta cần tìm hiểu về việc xác thực được thực hiện như thế nào. *Khi một thông điệp được xác thực*, nghĩa là

- Thông điệp đó không bị thay đổi
- Thông điệp này là đúng của người gửi.

Nếu như cả bên gửi và bên nhận không có sự không thống nhất nào về xuất xứ cũng như nội dung của thông điệp, thì việc trao đổi như vậy được xác nhận là hoàn tất. Cả hai bên đều tin rằng, không có một người thứ ba can thiệp vào quá trình trao đổi tin này.

Tuy nhiên, có những đoạn tin gian lận xuất phát từ bên gửi hoặc do bên nhận tự tạo ra trong các giao dịch thương mại, thanh toán, trao đổi trên mạng, v.v. Các tranh chấp có thể xảy ra và cũng có nhiều trường hợp người bị lừa khó mà nhận biết được, nếu không có biện pháp phòng ngừa và phát hiện hữu hiệu. Ngay cả khi dùng mật mã khóa bí mật mà cả hai bên gửi và nhận đều biết, người gửi có thể tự tạo thêm một mã để kiểm tra xem thông điệp nhận được có xác thực hay không.

Trong thực tế, các hoạt động thương mại, quản lý hành chính, hoạt động nghiệp vụ, các tài liệu trên giấy có giá trị cam kết giao hẹn với nhau (như ngân phiếu, hợp đồng) thì bên gửi là bên có khả năng làm giả nhiều nhất. Ngược lại, cũng có khi một số trường hợp phía bên nhận lại chối bỏ trách nhiệm của mình vì thấy những điều đó bất lợi cho mình. Trong các trường hợp đó, việc xác thực thường được dựa vào chữ ký của hai bên để xác nhận các điều khoản đã cam kết, giao kèo với nhau trên “giấy trắng mực đen”, và đó cũng là cơ sở pháp lý để giải quyết khi có tranh chấp.

Nhưng nếu các hoạt động trên thực hiện trao đổi với nhau trên mạng truyền số liệu thì vấn đề phức tạp hơn nhiều. Ví dụ, nếu bên B mang đến Tòa án một tài liệu nhận được qua mạng truyền số liệu (Internet) và bên A lại chối bỏ trách nhiệm gửi của mình thì Tòa án cũng rất khó phân xử rạch rối. Bởi vì cũng có khả năng bên B làm giả đoạn tin và cũng có khi bên A có gửi thật nhưng lại chối bỏ trách nhiệm.

Vấn đề đặt ra là làm thế nào để phân xử được trong những trường hợp như trên. Muốn giải quyết được vấn đề xác thực thì cần phải có một cơ chế nào đó giống như chữ ký tay để cả

hai bên gửi và nhận cùng kiểm tra và không thể tạo giả mạo chữ ký đó. Một trong các biện pháp để thực hiện xác thực là sử dụng *chữ ký số*.

Để hiểu rõ về chữ ký số, ta cần nói rõ hơn về một số khái niệm liên quan đến *mật mã khóa công khai*. Mật mã khóa công khai dựa trên hai khái niệm: *khóa công khai* và *khóa riêng*. Khóa công khai thì có thể công bố cho mọi người biết. Khóa riêng thì chỉ người giữ nó được biết và điều quan trọng là không để người khác biết. Hai khóa này có mối quan hệ với nhau theo các quan hệ toán học, nhưng có thể tin rằng, thực tế không thể từ một khóa này tìm ra được khóa kia. Mặc dù đã biết khóa công khai, nhưng cả đời người cũng không tìm ra được khóa riêng, cho dù ta có dùng bao nhiêu máy tính, với loại máy tính nào đi nữa cũng không thực hiện được [2]. Điều này có thể khó tin, song cho đến hiện nay chưa ai tìm được thuật toán để tính được khóa riêng từ khóa công khai trong thời gian chấp nhận được.

Đã có khá nhiều thuật toán được sử dụng để mã hóa và giải mã thông tin. Trong số đó thuật toán được nói tới nhiều là thuật toán RSA của Rivest, Shamir và Adleman. Thuật toán này được xây dựng dựa trên độ khó của việc phân tích các số lớn thành các thừa số, nhất là các thừa số nguyên tố. Phần lớn các nhà mật mã tin rằng những khóa với “Modulus” của số gồm 2000 bit hoặc lớn hơn thì hoàn toàn yên tâm đối với mọi sự tấn công từ bên ngoài.

Có hai cặp khóa công khai và khóa riêng được sử dụng để mã hóa và chứng thực. Nếu một người gửi cho bạn một bản tin được mã hóa bằng khóa công khai thì bạn có thể giải mã bằng khóa riêng để đọc bản tin đó mà những người khác không làm được. Ngược lại, bạn có thể “ký” (đánh dấu) vào một bản tin bằng khóa chứng thực riêng, sau đó người nhận có thể kiểm tra tính xác thực của chữ ký bằng khóa công khai mà bạn cung cấp. Việc kiểm chứng này chỉ cần thực hiện được đối với những bản tin đã được đánh dấu (được ký) và nó sẽ thất bại nếu ai đó sử dụng khóa của họ để đánh dấu vào bản tin đó, nghĩa là làm thay đổi bản tin đã được ký.

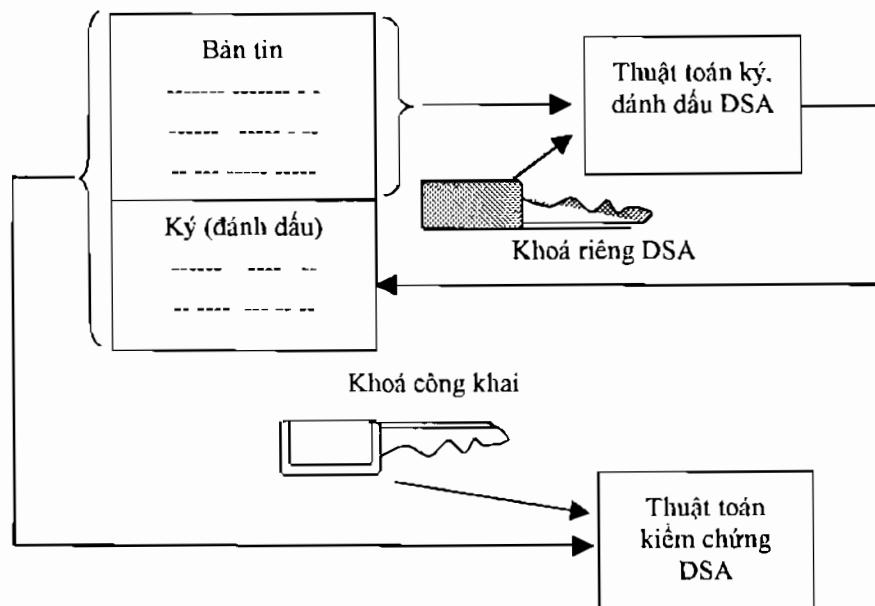
Ta hãy tìm hiểu cách hoạt động của thuật toán DSA để sinh ra cặp khóa công khai/riêng và cách sử dụng chúng như thế nào.

Giả sử rằng Nam muốn gửi cho Hoa một bản tin và Hoa muốn biết xem bản tin này có phải đúng do Nam gửi hay không. Nam viết bản tin rồi đánh dấu (tạo ra bản dấu vết) bằng khóa riêng của mình. Hoa nhận được bản copy của khóa công khai và sử dụng nó để kiểm tra tính xác thực của bản tin. Nếu việc kiểm chứng này thành công thì Hoa tin rằng:

1. Bản tin gốc không bị thay đổi.
2. Bản tin được ký bởi Nam.

Một chữ ký số cần phải phụ thuộc vào tất cả các bit của đoạn tin với mục đích là giữ sự bền vững của đoạn tin đó. Điều quan trọng ở đây là, phải đảm bảo không một ai có thể làm thay đổi nội dung của bản tin rõ trong lúc phải giữ nguyên chữ ký số.

Quá trình trao đổi và xác thực theo thuật toán DSA được mô tả như sau:



**Hình 7.7. Sự trao đổi giữa các khóa của thuật toán DSA**

Tuy nhiên, về mặt lý thuyết sẽ không có một giải pháp nào là tuyệt đối an toàn. Do vậy, chữ ký số cũng không phải là giải pháp toàn năng để chống lại sự giả mạo và có thể phân xử được mọi tranh chấp, cho nên cần có cơ chế trọng tài. Chữ ký số có thể kết hợp với mật mã để tăng tính hiệu quả của vấn đề xác thực.

Gói *bảo mật* của Java đã cài đặt hai thuật toán DSA, SHA-1 nêu trên. Nếu bạn muốn sử dụng RSA thì phải mua các lớp của RSA ([www.rsa.com](http://www.rsa.com)).

Phản tiếp theo chúng ta tập trung khai thác DSA. Có ba thuật toán được sử dụng.

1. Thuật toán sinh ra cặp khóa
2. Thuật toán ký (danh dấu) thông điệp
3. Thuật toán xác thực chữ ký.

### 1. Tạo ra cặp khóa

Để sinh ra những cặp khóa một cách hoàn toàn ngẫu nhiên, người ta thường sử dụng bộ sinh số ngẫu nhiên của lớp SecureRandom. Lớp này phát sinh các số ngẫu nhiên hơn (đảm bảo an ninh hơn) các số sinh bởi Random. Ví dụ,

```
SecureRandom secrand = new SecureRandom();
byte[] b = new byte[20];
secrand.setSeed(b); // Điều các bit một cách ngẫu nhiên
```

Trong Java, để tính một cặp khóa theo thuật toán DSA mới, ta sử dụng lớp KeyPairGenerator như sau:

```
KeyPairGenerator keygen = KeyPairGenerator.getInstance("DSA");
```

Nếu muốn sinh ra một khóa theo modulus của 512 bit thì hãy viết

```
keygen.initialize(512, secrand);
```

Sử dụng tiếp các lớp KeyPair, PublicKey, PrivateKey để sinh ra cặp khóa công khai và khóa riêng.

```
KeyPair keys = keygen.generateKeyPair();
```

```
PublicKey pubKey = keys.getPublic();
```

```
PrivateKey privKey = keys.getPrivate();
```

## 2. Đánh dấu (ký xác nhận) thông điệp

Muốn đánh dấu (ký xác nhận) một thông điệp (đoạn tin), ta phải tạo ra đối tượng của lớp Signature

```
Signature sigalg = Signature.getInstance("DSA");
```

Đối tượng này được sử dụng cho cả việc ký xác nhận lẫn việc xác thực thông điệp. Trước khi ký chứng thực, ta sử dụng phương thức initSign() và truyền khóa riêng cho nó.

```
sigalg.initSign(privKey);
```

Tiếp theo, sử dụng phương thức update() để đưa các byte của thông điệp (đọc theo từng byte) vào đối tượng đánh dấu.

```
while((ch = in.read()) != -1)
    sigalg.update((byte)ch);
```

Cuối cùng là chữ ký số.

```
byte[] signature = sigalg.sign();
```

## 3. Xác thực chữ ký

Người nhận được thông điệp cũng sẽ nhận được khóa công khai và cần phải kiểm tra xem tài liệu nhận được có đúng bản gốc của người gửi hay không.

```
Signature verifyAlg = Signature.getInstance("DSA");
```

```
verifyAlg.initVerify(pubKey);
```

```
while((ch = in.read()) != -1) // Truyền thông điệp vào đối tượng xác thực
```

```
    verifyAlg.update((byte)ch);
```

```
boolean check = verifyAlg.verify(signature); // Xác thực chữ ký
```

Nếu check là true thì chữ ký là xác thực, thông điệp nhận là đoạn tin đã được ký nhận bởi một khóa riêng tương ứng. Nghĩa là, cả người gửi và nội dung thông điệp được xác thực.

### Ví dụ 7.6

/\* SinatureTest.java: Chương trình sinh ra hai cặp khóa, sử dụng khóa riêng thứ nhất để ký nhận thông điệp gửi đi. Sau đó sử dụng hai khóa công khai để kiểm tra xem thông điệp nhận được có đúng được xác thực hay không?

```
/*
import java.security.*;
public class SignatureTest{
    public static void main (String args[]){
        try{
            // Tạo ra bộ sinh khóa ngẫu nhiên theo thuật toán DSA
            KeyPairGenerator keygen
                = KeyPairGenerator.getInstance("DSA");
            SecureRandom secrand = new SecureRandom();
            keygen.initialize(512, secrand);
            // Sinh ra cặp khóa thứ nhất
            KeyPair keys1 = keygen.generateKeyPair();
            PublicKey pubkey1 = keys1.getPublic();
            PrivateKey privkey1 = keys1.getPrivate();
            // Sinh ra cặp khóa thứ hai
            KeyPair keys2 = keygen.generateKeyPair();
            PublicKey pubkey2 = keys2.getPublic();
            PrivateKey privkey2 = keys2.getPrivate();
            // Tạo ra đối tượng để ký xác nhận thông điệp gửi đi theo thuật toán DSA
            Signature signalg = Signature.getInstance("DSA");
            signalg.initSign(privkey1);
            String message = "Trà tác giả tiền thường là 100 000.";
            System.out.println("Thông điệp gửi đi: " + message);
            signalg.update(message.getBytes());
            byte[] signature = signalg.sign();
            // Kiểm tra tính xác thực theo khóa công khai thứ nhất
            Signature verifyalg = Signature.getInstance("DSA");
            verifyalg.initVerify(pubkey1);
            verifyalg.update(message.getBytes());
            System.out.println("Thông điệp nhận được: " + message);
        }
    }
}
```

```

        if(!verifyalg.verify(signature))
            System.out.println("Not ");
        System.out.println("Signed with private key 1");
        // Kiểm tra tính xác thực theo khóa công khai thứ hai
        verifyalg.initVerify(pubkey2);
        verifyalg.update(message.getBytes());
        if(!verifyalg.verify(signature))
            System.out.print("Not ");
        System.out.println("Signed with private key 2");
    }
    catch(Exception e){
        System.out.println("Error: " + e);
    }
}
}
}

```

Chương trình trên chạy và cho kết quả:

Thông điệp gửi đi: Trà tác giả tiền thường là 100 000.

Thông điệp nhận được: Trà tác giả tiền thường là 100 000.

Signed with private key 1

No Signed with private key 2

### java.lang.KeyPairGenerator

### API

- `static KeyPairGenerator getInstance(String alg)`

Trả lại một đối tượng của KeyPairGenerator để sinh ra các cặp khóa theo thuật toán alg.

- `void initialize(int strleng, SecureRandom random)`

Khởi tạo đối tượng hiện thời (đối tượng sinh cặp khóa) với số bit là strleng và đối tượng ngẫu nhiên random.

- `KeyPair generateKeyPair()`

Sinh ra cặp khóa mới.

### java.lang.KeyPair

### API

- `PrivateKey getPrivate()`

Nhận lại khóa riêng của cặp khóa được sinh ra.

- `PublicKey getPublic()`

Nhận lại khóa công khai của cặp khóa được sinh ra.

`java.lang.Signature`

API

- `static Signature getInstance(String alg)`

Nhận đối tượng chữ ký của `Signature` được cài đặt theo thuật toán `alg`.

- `void initSign(PrivateKey privKey)`

Khởi động việc ký thông điệp theo khóa riêng.

- `void update(byte input)`

- `void update(byte[] input)`

- `void update(byte[] input, int offset, int len)`

Cập nhật bộ đệm thông điệp theo các byte.

- `byte[] sign()`

Hoàn tất việc ký xác nhận và trả lại đối tượng chữ ký.

- `void initVerify(PublicKey publicKey)`

Khởi động đối tượng kiểm tra theo khóa công khai.

- `boolean verify(byte[] signature)`

Kiểm tra tính xác thực của chữ ký.

## 7.5. VẤN ĐỀ CHỨNG THỰC

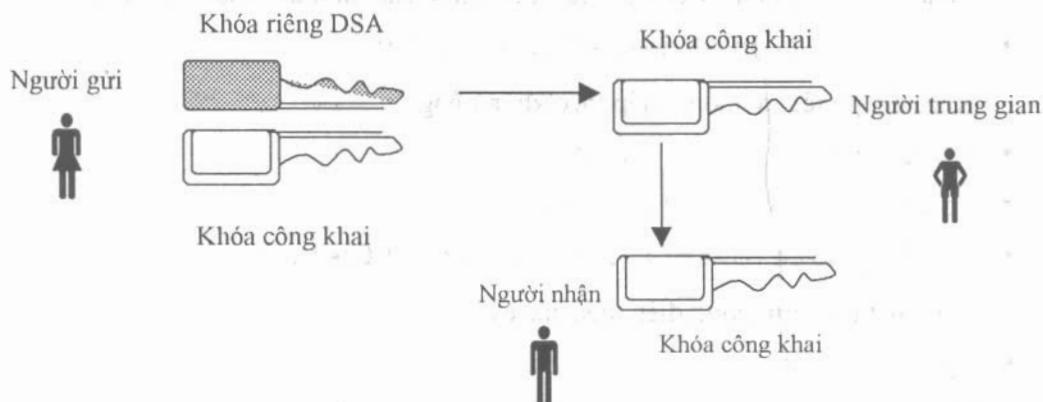
Như trên đã đề cập, nếu ta nhận một thông điệp được ký xác nhận bởi một người quen biết (người bạn) thì ta dễ dàng sử dụng khóa công khai được cung cấp để kiểm tra tính xác thực của thông điệp đó. Nay giờ ta xét tiếp trường hợp nhận được một tài liệu của một người không quen biết, người lạ. Tương tự như trên, nếu ta nhận được khóa công khai (vấn đề này không khó) thì vẫn kiểm chứng được xem tài liệu đó có sánh được với tài liệu được ký bằng khóa riêng tương ứng hay không.

Tuy nhiên, ở đây có vấn đề ta cần phải thận trọng vì ta *không có một khái niệm gì về người gửi*. Bất kỳ một người nào đó có thể tạo ra một cặp khóa, ký nhận vào tài liệu và gửi nó cho chúng ta. Vấn đề xuất hiện ở đây là, việc *xác định danh tính của người gửi hay còn gọi vấn đề chứng thực* được thực hiện như thế nào?

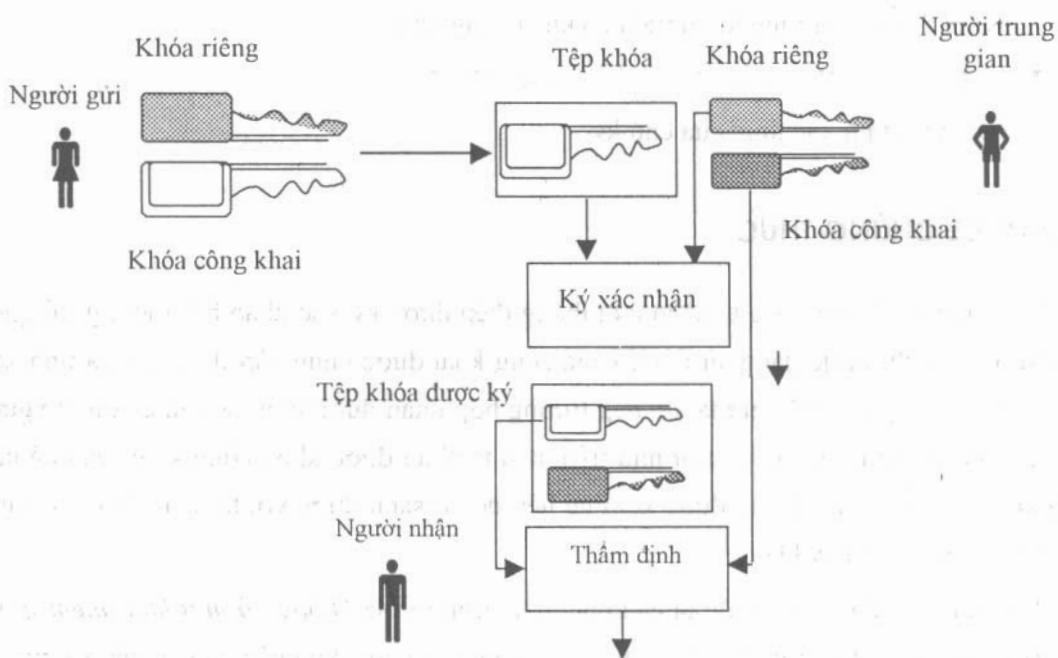
Cách giải quyết vấn đề trên trong thực tế thường là thông qua một đối tượng trung gian. Giả sử cả người gửi (là người không quen biết trước) và bạn có một người “quen” trung gian

(người, hay một tổ chức môi giới, một cơ quan có thẩm quyền). Người gửi gặp người trung gian trao tài liệu và khóa công khai, sau đó anh ta trao lại cho bạn (xem hình 7.8).

Trong thực tế, người trung gian cũng không cần gặp cả người gửi lẫn người nhận tài liệu. Anh ta có thể sử dụng một khóa riêng của mình để ký tiếp vào tài liệu của người gửi và gửi nó với khóa công khai cho người nhận (xem hình 7.9). Người nhận sẽ sử dụng cả hai khóa công khai nhận được xác thực tính trung thực của tài liệu nhận được.



Hình 7.8. Vấn đề chứng thực thông qua trung gian



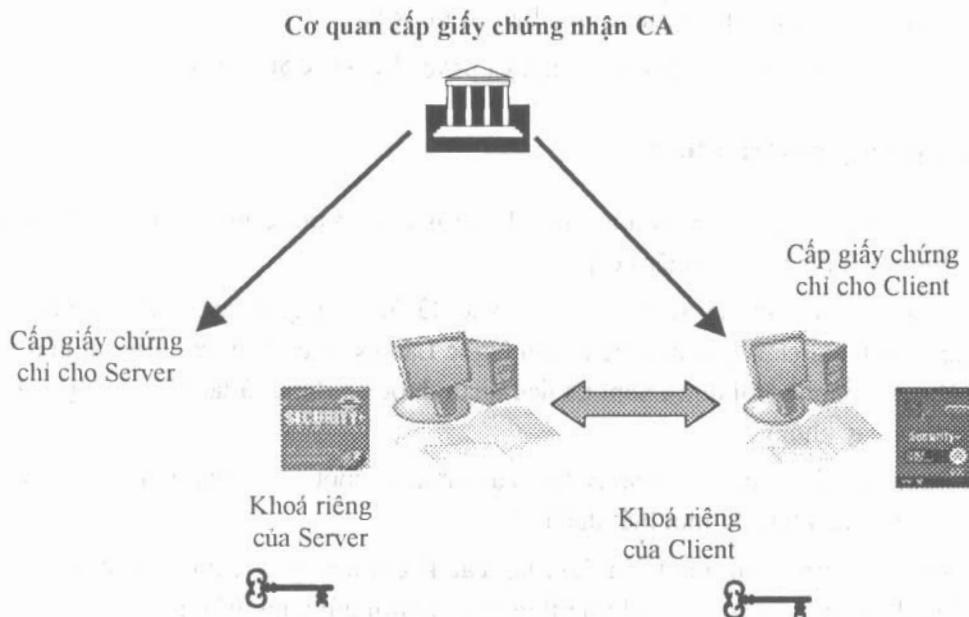
Hình 7.9. Vấn đề xác thực thông qua chữ ký của người trung gian

Song, vấn đề giải quyết như thế nào khi không có người môi giới trung gian. Khi đó ta có thể sử dụng giấy chứng nhận để chứng thực danh tính của người gửi.

### 7.5.1. Giấy chứng nhận Certificate X.509

Tương tự như trong thực tế, mỗi xí nghiệp được cấp một giấy phép sản xuất hay kinh doanh, chúng ta cũng có thể tạo ra các giấy chứng nhận để xác thực trong quá trình trao đổi tin. Giấy chứng nhận được một cơ quan, một tổ chức có thẩm quyền (được luật pháp thừa nhận) CA cấp, những người được cấp sẽ được xác thực và dựa vào giấy chứng nhận đó để bảo mật tin (mã hoá tin), rồi gửi đi. Những người nhận cũng được cấp một giấy chứng nhận cho phép họ xác thực và giải mã những thông tin nhận được. Một trong các mẫu ký xác nhận được sử dụng phổ biến là mẫu X.509 được định nghĩa bởi ITU-T.

Khi một máy trạm (Client) kết nối với một máy chủ (Server) sử dụng SSL/TLS [4], thì cả máy Client và Server phải thoả thuận với nhau về thuật toán mật mã chung sẽ được sử dụng để mã hoá tin cần trao đổi. Quá trình xác thực trong SSL/TLS sử dụng X.509 được thực hiện như sau.



Hình 7.10. Qui trình xác thực sử dụng X.509

1. Server gửi chứng chỉ X.509 chứa khoá công khai cho các máy trạm cần trao đổi tin.
2. Máy trạm (Client) tạo ra các số bí mật ngẫu nhiên chiếm khoảng 48 byte và sử dụng khoá công khai của Server để mật mã số bí mật đó, rồi gửi nó cho Server.
3. Server sử dụng khoá riêng để giải mã số bí mật đã được mã hoá nhận được.
4. Cả Server và Client chia sẻ với nhau số bí mật đó và người khác không biết được. Từ số bí mật đó phát sinh ra các khoá đối xứng và bắt đầu trao đổi tin với nhau.

Ngoài ra giấy chứng nhận X.509 được sử dụng rộng rãi bởi Microsoft, Netscape và nhiều hãng khác để ký xác nhận các thư điện tử E-mail, chứng thực mã chương trình và nhiều loại dữ liệu khác. Ở dạng đơn giản nhất, X.509 chứa các thông tin sau:

- Phiên bản (Version) của bằng chứng thực,
- Số hiệu của bằng chứng thực,
- Định danh của thuật toán ký xác nhận (tên gọi và các tham số),
- Tên người ký,
- Thời gian có hiệu lực (ngày bắt đầu và kết thúc),
- Định danh của người được chứng thực,
- Khóa công khai của giấy chứng nhận,
- Chữ ký (mã băm cho các trường dữ liệu, mã hoá chữ ký bằng khóa riêng).

Chi tiết hơn về cấu trúc của X.509 bạn có thể xem

<http://www.ietf.org/1etf/cnri/reston.va.us/ids/by.wg/x.509.html>

### 7.5.2. Lớp giấy chứng nhận

Thông thường các giấy chứng chỉ (chứng nhận) sẽ được một tổ chức CA có đủ thẩm quyền (được luật pháp công nhận) cấp.

Trong JDK có chương trình keytool.exe để lập ra giấy chứng nhận và quản lý chúng. Các phần tiếp theo, chúng ta tìm hiểu cách Ngọc Lan ký xác nhận vào một tài liệu và gửi nó cho Lê Bá; người nhận kiểm tra xem tài liệu nhận được có đúng là tài liệu do Ngọc Lan ký và có bị sửa đổi hay không?

Chương trình keytool quản lý kho các khóa và một CSDL các giấy chứng nhận. Mỗi mục trong kho các khóa có một biệt danh (alias).

Sau đây ta tìm hiểu quá trình thiết lập các kho khóa, giấy chứng nhận và trao đổi các thông điệp đã được ký xác nhận, chứng thực giữa người gửi và người nhận.

**Ví dụ 7.7.** Ngọc Lan tạo ra một kho các khóa riêng ngoclan.store và sinh ra một cặp khóa với biệt danh là ngoclan như sau:

```
keytool -genkey -keystore ngoclan.store -alias ngoclan
```

Khi sinh ra một khóa, hệ thống nhắc ta nhập các thông tin:

Enter keystore password: password

What is your first and last name?

[Unknown]: Ngọc Lan

What is the name of your organization unit?

[Unknown]: Engineering Department

What is the name of your organization?

[Unknown]: Information Technology Institute

What is the name of your city or Locality?

[Unknown]: Ha Noi

What is the name of your state or province?

[Unknown]: Ha Noi

What is the two-letter country code for this unit?

[Unknown]: VN

Is <CN = Ngoc Lan, OU = Engineering Department, O = Information Technology Institute, L = Ha Noi, ST = Ha Noi, C = VN> correct?

[no]: Y

keytool sử dụng các tên gọi phân biệt của X.509: CN (Common Name), OU (Organization Unit), O (Organization), L (Location), ST (State), và C (Country) để xác định khóa và cấp giấy chứng nhận. Lưu ý mật khẩu phải chứa ít nhất là 6 ký tự và ta phải nhớ để khai báo khi cần thiết về sau.

Giả sử Ngọc Lan muốn đưa khóa công khai cho Lê Ba thì phải đưa ra dưới dạng tệp:

keytool -export -keystore ngoclan.store -alias ngoclan -file ngoclan.cert

Hệ thống sẽ hỏi mật khẩu:

Enter keystore password: password

Nếu nhập đúng mật khẩu thì hệ thống sẽ tạo ra tệp ngonlan.cert chứa bằng chứng nhận của Ngọc Lan.

Sau đó Ngọc Lan có thể gửi tệp ngoclan.cert cho Lê Ba. Khi Lê Ba nhận được có thể in ra để biết được các thông tin cần thiết:

keytool -printcert -file ngoclan.cert

Kết quả nhận được trên màn hình:

Owner: CN = Ngoc Lan, OU = Engineering Department, O = Information Technology Institute, L = Ha Noi, ST = Ha Noi, C = VN

Issuer: CN = Ngoc Lan, OU = Engineering Department, O = Information Technology Institute, L = Ha Noi, ST = Ha Noi, C = VN

Serial number: 38107867

Valid from: Fri Aug 22 07:44:55 ITC 2005 until: Thu Nov 20 06:44:55 ITC 2005

Certificate fingerprints:

MD5: 5D:00:0F:95:01:30:B4:FE:24:CE:98:34:F0:C8:90:BB

SHA1:F4:10:2F:34:01:AC:B8:ED:15:E6:A8:01:89:75:88:CB:09:B2:A1:65

Giấy chứng nhận này nó tự xác nhận, vì vậy Lê Ba không thể dùng giấy chứng nhận khác để kiểm tra tính hợp lệ của bằng chứng nhận này được. Tuy nhiên, Lê Ba có thể hỏi

Ngọc Lan (qua điện thoại chặng hạn) để biết được dấu vết số (fingerprint) của giấy chứng nhận để đối sánh.

Sau khi nhận được giấy chứng nhận, Lê Ba có thể chuyển vào kho riêng của mình, ví dụ leba.store như sau:

```
keytool -import -keystore leba.store -alias ngoclan -file ngoclan.cer
```

Hệ thống hỏi Lê Ba về mật khẩu nhập vào và sau đó hiển thị lại những thông tin về giấy chứng nhận đã nêu trên, nếu thấy đúng thì nhấn Y hoặc y để khẳng định.

Sau khi hoàn tất, Ngọc Lan gửi các tài liệu đã được ký xác nhận cho Lê Ba. Bộ công cụ jarsigner của Java thực hiện ký và kiểm định các tệp JAR. Ngọc Lan chỉ cần đưa tài liệu cần ký xác nhận document.txt vào tệp JAR document.jar.

```
jar cvf document.jar document.txt
```

Sau đó Ngọc Lan sử dụng jarsigner để đưa chữ ký vào tệp document.jar.

```
jarsigner -keystore ngoclan.store document.jar ngoclan
```

Ngọc Lan phải nhập đúng mật khẩu thì tài liệu document.txt sẽ được ký xác nhận.

Khi nhận được tệp tài liệu document.jar, Lê Ba có thể sử dụng tùy chọn –verify của chương trình jarsigner.

```
jarsigner -verify -keystore leba.store document.jar
```

Nếu tệp document.jar không bị xâm phạm và chữ ký được đối sánh thì jarsigner sẽ in ra kết quả khẳng định tệp JAR được xác thực.

```
jar verified
```

Ngược lại sẽ thông báo lỗi.

Cơ chế đóng gói và lưu trữ các tệp JAR sẽ được đề cập chi tiết ở phần sau.

Để tách tất cả các tệp được nén trong document.jar, Lê Ba có thể sử dụng

```
jar xf document.jar
```

trong đó có tệp document.txt nhận được từ Ngọc Lan.

### 7.5.3. Ký giấy chứng nhận

Phần trên ta đã xem xét cách sử dụng giấy chứng nhận “tự ký” để gửi khóa công khai cho các bộ phận khác. Song, người nhận phải được đảm bảo rằng giấy chứng nhận này là hợp lệ.

JDK không cung cấp công cụ để ký giấy chứng nhận nêu trên. Phần tiếp theo chúng ta tìm hiểu cách viết chương trình Java để ký nhận giấy chứng nhận bằng khóa riêng từ kho các khóa.

**Ví dụ 7.8.** Viết chương trình ký giấy chứng nhận bằng khóa riêng lấy ra từ kho các khóa đã được thiết lập từ trước (như ở ví dụ 7.7).

Trước tiên ta phải thiết lập đối tượng của KeyStore. Kho các khóa được tạo ra bởi keytool có kiểu là "JKS" và nguồn cung cấp là "SUN".

```
KeyStore store = KeyStore.getInstance("JKS", "SUN");
```

Tiếp theo là phải tải dữ liệu của kho store vào chương trình, tất nhiên là phải cho biết tên của luồng vào và mật khẩu (password). Lưu ý, mật khẩu thường (nên) tổ chức dưới dạng mảng (Array) các ký tự chứ không phải là xâu (String). JVM có thể lưu giữ các xâu một thời gian dài trước khi chúng bị dọn dẹp và do vậy các hacker có thể xâm nhập vào các tệp trao đổi để phát hiện ra mật khẩu còn các đối tượng của Array sẽ bị xóa bỏ ngay sau khi chúng không cần để sử dụng tiếp ở giai đoạn sau.

```
InputStream in = new FileInputStream(ksName);
char[] password = readPassword(console, "Keystore password");
store.load(in, password);
Arrays.fill(password, ' '); // Xoá mật khẩu
in.close();
```

Ta sử dụng getKey() tìm khóa riêng để ký. Hàm getKey() yêu cầu truyền alias và mật khẩu.

```
char[] keyPassword
= readPassword(console, "Keystore password for " + alias);
PrivateKey privKey
= (PrivateKey) store.getKey(alias, keyPassword);
Arrays.fill(keyPassword, '');
```

Chúng ta đã sẵn sàng để đọc giấy chứng nhận và ký xác nhận. Lớp CertificateFactory đọc giấy chứng nhận từ tệp đầu vào.

```
in = new FileInputStream(inName);
CertificateFactory factory
= CertificateFactory.getInstance("X.509");
```

Sau đó gọi hàm generateCertificate() để sinh ra giấy chứng nhận tương ứng.

```
X509Certificate inCert
= (X509Certificate) factory.generateCertificate(in);
in.close();
```

Mục đích của chương trình là ký các byte trong giấy chứng nhận. Ta tìm các byte bằng hàm getTBSCertificate().

```
byte[] inCertBytes = inCert.getTBSCertificate();
```

Giấy chứng nhận đã được ký cần lưu vào tệp

```
X509CertInfo info = new X509CertInfo(inCertBytes);
info.set(X509CertInfo.ISSUER,
         new CertificateIssuerName((X500Name)issuer));
X509CertImpl outCert = new X509CertImpl(info);
outCert.sign(privKey, issuerAlg);
outCert.derEncode(out);
```

Lưu ý, ta sử dụng gói thư viện sun.security.x509 để sinh ra các giấy chứng nhận, cài đặt các thuật toán đảm bảo an ninh dữ liệu như RSA, v.v.

```
// CertificateSigner.java
import java.io.*;
import java.security.*;
import java.security.cert.*;
import java.util.*;
import sun.security.x509.X509CertImpl;
import sun.security.x509.X509CertInfo;
import sun.security.x509.X500Name;
import sun.security.x509.CertificateIssuerName;

public class CertificateSigner{
    public static void main (String args[]){
        String ksName = null;          // Tên của kho
        String alias = null;           // Bí danh của khóa riêng
        String inName = null;           // Tên của tệp đầu vào
        String outName = null;          // Tên của tệp đầu ra
        for(int i = 0; i < args.length; i += 2){
            if(args[i].equals("-keystore"))
                ksName = args[i+1];
            else if(args[i].equals("-alias"))
                alias = args[i+1];
            else if(args[i].equals("-infile"))
                inName = args[i+1];
            else if(args[i].equals("-outfile"))
                outName = args[i+1];
        }
        try {
            KeyStore ks = KeyStore.getInstance("JKS");
            ks.load(new FileInputStream(ksName), "changeit".toCharArray());
            Key key = ks.getKey(alias, "changeit".toCharArray());
            if(key == null)
                System.out.println("Không tìm thấy khóa");
            else if(key.getAlgorithm().equals("RSA"))
                sign(ks, alias, inName, outName);
            else
                System.out.println("Khóa không phải RSA");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    private void sign(KeyStore ks, String alias, String inName, String outName) throws IOException, GeneralSecurityException {
        CertificateIssuerName issuer = new CertificateIssuerName((X500Name)ks.getEntry(alias, null).getCertificate());
        X509CertInfo info = new X509CertInfo(inName);
        info.set(X509CertInfo.ISSUER, issuer);
        X509CertImpl cert = new X509CertImpl(info);
        cert.sign(key, "SHA1withRSA");
        cert.getDerEncoded();
        FileOutputStream fos = new FileOutputStream(outName);
        fos.write(cert.getDerEncoded());
        fos.close();
    }
}
```

```
        else usage();

    }

    if(ksName == null || alias == null || inName == null
        || outName == null) usage();

try{
    PushbackReader console =
        new PushbackReader(new InputStreamReader(System.in));

    KeyStore store = KeyStore.getInstance("JKS", "SUN");
    InputStream in = new FileInputStream(ksName);
    char[] password = readPassword(console, "Keystore password");

    store.load(in, password);
    Arrays.fill(password, ' '); // Xoá mật khẩu
    in.close();

    char[] keyPassword
        = readPassword(console, "Keystore password for " + alias);
    PrivateKey privKey
        = (PrivateKey)store.getKey(alias, keyPassword);
    Arrays.fill(keyPassword, ' ');

    if(privKey == null)
        error("No such private key");

    in = new FileInputStream(inName);
    CertificateFactory factory
        = CertificateFactory.getInstance("X.509");

    X509Certificate inCert
        = (X509Certificate)factory.generateCertificate(in);
    in.close();
    byte[] inCertBytes = inCert.getTBSCertificate();

    X509Certificate issuerCert
        = (X509Certificate)store.getCertificate(alias);
```

```
Principal issuer = issuerCert.getSubjectDN();
String issuerAlg = issuerCert.getSigAlgName();

FileOutputStream out = new FileOutputStream(outName);

X509CertInfo info = new X509CertInfo(inCertBytes);
info.set(X509CertInfo.ISSUER,
        new CertificateIssuerName((X500Name)issuer));

X509CertImpl outCert = new X509CertImpl(info);
outCert.sign(privKey, issuerAlg);
outCert.derEncode(out);

out.close();
}

catch(Exception e){
    System.out.println("Error: " + e);
}

}

public static char[] readPassword(PushbackReader in,
                                  String promp) throws IOException{
System.out.print(promp + ": ");
System.out.flush();
final int MAX_PASSWORD_LENGTH = 100;
int leng = 0;
char[] buff = new char[MAX_PASSWORD_LENGTH];

while(true){
    int ch = in.read();
    if(ch == '\r' || ch == '\n' || ch == -1 ||
       ch == MAX_PASSWORD_LENGTH){
        if(ch == '\r')      // DOS xem như kết thúc dòng "\r\n"
        {
            ch = in.read();
        }
    }
}
```

```

        if(ch == '\n' && ch != -1) in.unread(ch);
    }
    char[] password = new char[leng];
    System.arraycopy(buff, 0, password, 0, leng);
    Arrays.fill(buff, ' ');
    return password;
} else{
    buff[leng] = (char)ch;
    leng++;
}
}

public static void usage(){
    System.out.println("Usage: java CertificateSigner"
        + " -keystore keyStore - alias issuerKeyAlias"
        + " -infile inputFile -outfile outputFile");
    System.exit(1);
}

public static void error(String message){
    System.out.println(message);
    System.exit(1);
}
}

```

Để sử dụng được chương trình trên, ta phải sử dụng một kho các khóa đã được thiết lập. Giả sử chúng ta tạo ra một kho tên là engsoft.store với biệt danh là engroot. Giấy phép được tạo ra và đưa vào tệp engroot.cert.

```

keytool -genkey -keystore engsoft.store -alias engroot
keytool -export -alias engroot -keystore engsoft.store
    -file engroot.cert

```

Tiếp theo, đưa vào kho các khóa của người nhận, ví dụ kho có tên lenguyen.store

```

keytool -import -alias engroot -keystore lenguyen.store
    -file engroot.cert

```

Chương trình đọc giấy chứng nhận từ tệp ngoclan.cert lấy khóa riêng từ kho engsoft.store và ký xác nhận rồi ghi lên tệp ngoclan\_engisoft.cert.

```
java CertificateSigner -keystore engsoft.store -alias engroot
    -infile ngoclan.cert -outfile ngoclan_engisoft.cert
```

Ta phải nhập mật khẩu tương ứng để chương trình ký xác nhận vào giấy chứng nhận.

Ngọc Lan muốn gửi tài liệu cho những nhân viên trong phòng Engineering Department, ví dụ gửi tệp ngoclan\_engisoft.cert cho Lê Nguyên.

### java.security.KeyStore

### API

- static getInstance(String type)
- static getInstance(String type, String provider)

Xác định đối tượng của kho các khóa. Nếu sử dụng keytool của Java thì type là “JKS” và provider là “SUN”.

- void load(InputStream in, char[] password)

Nhập một kho các khóa từ luồng vào in khi mật khẩu được chấp nhận.

- Key getKey(String alias, char[] password)

Nhận lại một khóa riêng với biệt danh alias được lưu trong kho hiện thời.

- Certificate getCertificate(String alias)

Nhận lại một giấy chứng nhận với biệt danh alias được lưu trong kho hiện thời.

### java.security.cert.CertificateFactory

### API

- CertificateFactory getInstance(String type)

Tạo ra một đối tượng để làm ra các giấy chứng nhận kiểu type. Kiểu sử dụng ở đây là “X509”.

- Certificate generateCertificate(InputStream in)

Sinh ra giấy chứng nhận từ luồng vào in.

### java.security.cert.Certificate

### API

- PublicKey getPublic()

Nhận lại khóa công khai được đính kèm với giấy phép.

- byte[] getEncoded()

Đọc mã của giấy chứng nhận.

- String getType()

Đọc kiểu của giấy chứng nhận, ví dụ “X509”.

## java.security.cert.X509 Certificate

## API

- Principal getSubjectDN()
- Principal getIssuerDN()

Xác định tên của người chứng nhận và người được cấp giấy chứng nhận.

- Date getNotBefore()
- Date getNotAfter()

Xác định khoảng thời hạn hợp lệ của giấy phép.

- String getSigAlgName()
- byte[] getSignature()

Xác định thuật toán và chữ ký của giấy chứng nhận.

- byte[] getTBCertificate()

Đọc thông tin được mã hóa cần thiết để ký giấy chứng nhận.

#### 7.5.4. Ký nhận các tệp JAR

Trước tiên chúng ta đi tìm hiểu cách đóng gói và tổ chức các tệp lưu trữ JAR trong Java. Cơ chế gói gọn (package) là rất tiện lợi để tổ chức những lớp có liên quan với nhau thành các nhóm và nhất là trong quá trình phát triển phần mềm, ta cần phải đóng gói các tệp chương trình ứng dụng thành các gói để tiện lợi cho việc di chuyển và cài đặt ứng dụng. Chính các gói giúp chúng ta dễ dàng định vị, sử dụng các tệp lớp và hỗ trợ để điều khiển hoạt động truy cập vào dữ liệu của lớp lúc thực thi chương trình.

Khi chương trình đã được hoàn tất và chuyển sang giai đoạn thẩm định, kiểm tra để chuẩn bị triển khai, thì nên tổ chức thành các *tệp JAR*. Tệp JAR là dạng nén theo tệp ZIP và đóng gói những tệp thực thi (tệp lớp), cùng với những tệp ứng dụng liên quan sao cho chúng có thể được triển khai như là một đơn thể.

*Các ưu điểm của tệp dạng JAR là:*

- *Bảo mật:* Bạn có thể ký số vào nội dung của một tệp JAR. Người sử dụng nếu nhận dạng được chữ ký của bạn sẽ có cơ hội được đảm bảo an toàn hơn đối với những người khác.
- *Giảm thiểu thời gian chuyển tải:* Nếu các tệp của một chương trình được tổ chức thành một tệp JAR thì chúng sẽ được tải xuống trình duyệt trong cùng một giao dịch HTTP mà không cần tạo ra các kết nối cho từng tệp trong đó.
- *Được nén:* Dạng JAR cho phép nén các tệp để cho việc lưu trữ hiệu quả và sau đó việc tải các tệp nén là khá tiện lợi.

- *Đóng dấu xi vào gói (Version 1.2)*: Tất cả các lớp được định nghĩa trong một gói được tìm thấy trong cùng một tệp JAR.
- *Tương thích*: Cơ chế xử lý các tệp JAR là một bộ phận được chuẩn hóa của API với Java platform.

*Để nén và gói những tệp liên quan với nhau vào một tệp JAR, sử dụng chương trình jar của JDK và viết một dòng lệnh DOS command dạng:*

```
jar cf jarFileName.jar fileList
```

Trong đó, jarFileName là tên của tệp JAR, fileList là danh sách các tệp lớp hoặc những tệp khác có liên quan, ngăn cách với nhau bằng dấu cách.

Ví dụ, nếu muốn tạo ra một gói applet.jar để gói các lớp RMIApp.class, Send.class, index.html, java.policy, MessagesBundle\_en\_US.properties thì thực hiện như sau:

```
jar cf applet.jar
```

```
RMIApp.class Send.class index.html java.policy  
MessagesBundle_en_US.properties
```

*Khi muốn rời nén và mở gói applet.jar, ta sử dụng chương trình jar để tách ra với tuỳ chọn xf:*

```
jar xf applet.jar
```

Tiếp theo, chúng ta đi tìm hiểu cách ký chứng nhận một applet để sử dụng trong Java Plug-in. Có hai kịch bản chính:

1. Các applet được ký xác nhận trên mạng nội hạt (Intranet)
2. Các applet được ký xác nhận trên mạng quốc tế (Internet).

Trong kịch bản thứ nhất, bộ phận quản trị hệ thống có thể cài đặt các giấy chứng nhận và các tệp chính sách trên các máy địa phương. Khi Java Plug-in nạp về một applet được ký xác nhận, nó sẽ thông qua kho chứa các khóa và tệp chính sách để kiểm duyệt các quyền được thực hiện của applet đó.

Ở kịch bản thứ hai, các hàng bán phần mềm nhận được giấy chứng nhận được ký bởi các tổ chức có thẩm quyền như Thrawte hay Verign. Khi người sử dụng đầu cuối truy cập vào một trang Web có chứa một applet đã được ký xác nhận, hệ thống xác định được danh tính của người cung cấp applet này và cho phép nó thực hiện theo một số giới hạn nào đó hoặc toàn quyền sử dụng.

Phần này chúng ta chủ yếu tìm hiểu cách xây dựng các tệp chính sách để cấp quyền cho các applet từ các nguồn đã biết. Việc xây dựng, triển khai các tệp chính sách không phải là những công việc dành cho những người sử dụng đầu cuối, mà chính là những người quản trị hệ thống phải đảm nhiệm trong mạng nội hạt.

**Ví dụ 7.9.** Chúng ta xét tiếp ví dụ về yêu cầu trao đổi thông tin của Engineering Department. Phòng này muốn nhân viên của mình chạy một số chương trình applet và chúng được phép truy cập được các tệp cục bộ tại các máy đó. Bởi vì những applet này không thể chạy được ở bên trong hộp cát Sandbox [2] nên họ phải cài đặt các tệp trên các máy đó. Điều này có nghĩa là họ sẽ phải cập nhật lại các tệp chính sách mỗi khi mã chương trình applet phải chuyển sang Web Server khác. Để khắc phục nhược điểm đó, họ quyết định ký xác nhận các tệp JAR chứa mã lệnh của các applet. Quá trình thực hiện như sau:

1. Tạo ra tệp MyApplet.jar để chứa các tệp mã chương trình applet

```
jar cvf MyApplet.jar *.class
```

2. Chạy jarsigner để ký xác nhận tệp MyApplet.jar theo khóa riêng từ kho các khóa engsoft.store với biệt danh là engroot.

```
jarsigner -keystore engsoft.store MyApplet.jar engroot
```

Tất nhiên, kho chứa khóa gốc cần phải để ở vị trí an toàn. Muốn thế, ta hãy lập kho thứ hai là certs.store để cho các giấy chứng nhận.

```
keytool -genkey -keystore engsoft.store -alias engroot
```

```
keytool -export -keystore engsoft.store -alias engroot -file engroot.cer
```

```
keytool -import -keystore certs.store -alias engroot -file engroot.cer
```

Lệnh đầu tiên hệ thống sẽ hỏi chúng ta về mật khẩu và các thông tin cần thiết để cấp giấy chứng nhận và thiết lập kho chứa các cặp khóa cùng các giấy chứng nhận. Lưu ý: cần nhớ mật khẩu đó để bảo cho hệ thống thẩm định ở các câu lệnh sau đó.

Tiếp theo ta phải tạo ra tệp chính sách để cấp phép cho tất cả các applet được ký xác nhận trong kho vừa được thiết lập.

Ta phải đưa thêm vị trí lưu kho và kiểu kho vào đầu tệp chính sách.

```
keystore "keystoreURL", "keystoreType";
```

Ví dụ, “JKS” là kiểu kho được sinh bởi keytool, dòng

```
keystore "file:certs.store", "JKS";
```

sẽ được đưa vào đầu tệp chính sách.

Ngoài ra cũng cần bổ sung signedBy “alias” vào sau một hoặc nhiều mệnh đề grant trong tệp chính sách.

Như vậy, tệp chính sách trong ví dụ của chúng ta: applets.policy sẽ phải là

```
keystore "file:certs.store", "JKS";
```

```
grant signedBy "engroot"
```

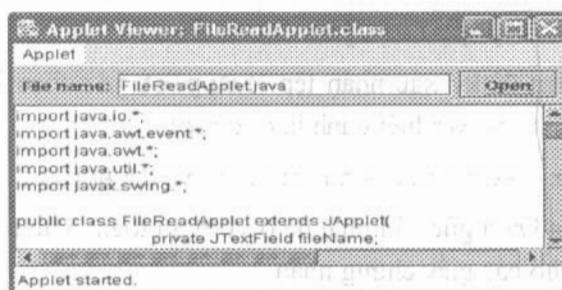
```
{ permission java.io.FilePermission "<<ALL FILES>>", "read";  
};
```

Hãy xây dựng một chương trình FileReadApplet để đọc các tệp ở thư mục cơ sở và tất cả các thư mục con của nó.

Sau khi đã hoàn tất các bước nêu trên, engsoft.store, certs.store với bí danh là engroot, applets.policy và FileReadApplet.class đã được tạo ra, ta có thể sử dụng appletviewer có sử dụng tệp chính sách như sau:

```
appletviewer -J-Djava.security.policy=applets.policy
```

FileReadApplet.html



Hình 7.11. Chương trình FileReadApplet được phép đọc các tệp cục bộ

```
// FileReadApplet.java
import java.io.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;
import javax.swing.*;
```

```
public class FileReadApplet extends JApplet{
```

```
    private JTextField fileName;
```

```
    private JTextArea fileText;
```

```
    public FileReadApplet()
```

```
{
```

```
    fileName = new JTextField(20);
```

```
    JPanel panel = new JPanel();
```

```
    panel.add(new JLabel("File name:"));
```

```
    panel.add(fileName);
```

```
    JButton openButton = new JButton("Open");
```

```
    panel.add(openButton);
```

```
    openButton.addActionListener(
```

```

        new ActionListener(){
            public void actionPerformed(ActionEvent evt){
                loadFile(fileName.getText());
            }
        });

Container contentPane = getContentPane();
contentPane.add(panel, "North");

fileText = new JTextArea();
contentPane.add(new JScrollPane(fileText), "Center");
}

public void loadFile(String fileName){
    try{
        fileText.setText("");
        BufferedReader in = new BufferedReader
            (new FileReader(fileName));
        String s;
        while((s = in.readLine()) != null)
            fileText.append(s + "\n");
        in.close();
    }catch(IOException e){
        fileText.append(e + "\n");
    }catch(SecurityException e){
        fileText.append("I am sorry, but I cannot do that!\n");
    }
}
}
}

```

Để chạy được trong chế độ Java Plug-in, ta phải thay đổi tệp java.security ở thư mục {java.home}/jre/lib/security.

```

# The default is to have a single system wide policy file
policy.url.1=file:${java.home}/lib/security/java.policy
policy.url.2=file:${user.home}/java.policy

```

Và bổ sung thêm file URL cho tệp chính sách như sau:

```
policy.url.3=file:///home}/applets.policy
```

Nếu ta sử dụng Plug-in trong Netscape 4 hoặc Internet Explore thì cần phải thay đổi tệp HTML để nạp Plug-in cùng với các thẻ EMBED hoặc OBJECT. Netscape 5 sẽ tự động nạp Plug-in khi ta sử dụng thẻ OBJECT.

## BÀI TẬP

- 7.1. Cho trước một danh sách các từ, các thư mục cấm truy cập. Xây dựng một applet để kiểm soát việc xem (đọc) nội dung của các tệp tin, ở những thư mục được phép theo tệp chính sách.
- 7.2. Một đơn vị muốn trao đổi thông tin và cung cấp các phần mềm cho nhân viên của mình trên mạng nội bộ. Hãy xây dựng giấy chứng nhận cho người đứng đầu đơn vị. Mỗi thông điệp hay phần mềm cần đưa lên mạng sẽ được đưa vào tệp nén JAR cùng với giấy chứng nhận. Người nhận sẽ kiểm tra lại tính xác thực của thông tin, của phần mềm căn cứ vào tính xác thực của giấy chứng nhận.
- 7.3. Xây dựng hệ thống bán hàng qua mạng của Công ty Sao Mai.
  - + Khách hàng có thể đặt mua hàng bằng cách gửi đơn đặt hàng, trong đó lựa chọn những mặt hàng, số lượng cần mua và thông báo số tài khoản. Đơn đặt mua hàng được ký xác nhận bởi khách hàng bằng khóa riêng. Khách mua hàng gửi đơn mua hàng và khóa công khai cho Công ty Sao Mai.
  - + Công ty Sao Mai (người bán hàng) sẽ kiểm tra tính xác thực đơn đặt mua hàng của khách bằng khóa công khai. Nếu đúng thì lập hóa đơn bán hàng, ký xác nhận rồi gửi hàng cùng hóa đơn cho khách hàng.

# Chương 8

## LẬP TRÌNH THEO CHUẨN QUỐC TẾ VÀ BẢN ĐỊA HÓA PHẦN MỀM

Chương VIII giới thiệu những kỹ thuật xây dựng phần mềm đáp ứng các qui định chuẩn của mỗi quốc gia (bản địa hoá) gồm những nội dung chính:

- ✓ Vai trò của các lớp Locale và các dạng biểu diễn dữ liệu
- ✓ Lớp xử lý ngày tháng Date, thời gian Time và tiền tệ
- ✓ Những vấn đề quốc tế hoá phần mềm
- ✓ Bản địa hoá giao diện đồ họa.

### 8.1. VĂN ĐỀ BIỂU DIỄN DỮ LIỆU PHỤ THUỘC VÀO VĂN HÓA CỦA TÙNG DÂN TỘC

Trong thời đại hội nhập kinh tế thế giới hiện nay, sự trao đổi giữa các công ty dù lớn hay nhỏ với nhau phần lớn đều sử dụng nhiều ngôn ngữ khác nhau. Văn đề trao đổi thông tin giữa các dân tộc, giữa nhiều cộng đồng trên thế giới với nhau luôn gặp phải những khó khăn, những trở ngại do việc qui định về cách viết và hiểu các thông điệp, số liệu là rất khác nhau. Ví dụ đơn giản như dấu ‘.’ được sử dụng để phân biệt các đơn vị hàng ngàn, triệu, tỉ, v.v. của các số nguyên ở Việt Nam, trong khi người Mỹ lại sử dụng dấu ‘,’ để phân biệt các đơn vị đó. Như vậy, *dữ liệu phụ thuộc vào văn hoá* là dữ liệu có thể thay đổi (cách biểu diễn) theo văn hoá, dân tộc, theo từng chuẩn hoá của từng quốc gia, từng khu vực [2].

Thực tế hiện nay có nhiều người trên thế giới có thể đọc và hiểu được tiếng Anh. Song, người sử dụng sẽ cảm thấy thoải mái, tự tin hơn khi sử dụng những applet hay chương trình ứng dụng hiển thị các thông tin được viết bằng tiếng dân tộc của họ và biểu diễn dữ liệu theo những khuôn dạng mà họ quen thuộc. Hãy tưởng tượng, một AppletCalculator (được giới thiệu ở cuối chương) sẽ được nhiều người sử dụng hiệu quả hơn nhiều nếu nó hiển thị được các kết quả tính toán theo khu vực của họ.

Java 2 Platform cung cấp các đặc trưng để thực hiện việc quốc tế hoá, cho phép tách các dữ liệu phụ thuộc vào các nền văn hoá từ những phần mềm và làm cho thích ứng với nhiều nền văn hoá theo nhu cầu (địa phương hoá hay còn gọi là bản địa hoá). Chúng ta hy

vọng phần lớn những vấn đề về thông tin, dữ liệu được cung cấp cho dân cư trú ở những vùng khác nhau được biểu diễn phù hợp với nền văn hoá của họ trên các phần mềm ứng dụng, hay trên các applet.

Nhiều người lập trình tin rằng, họ cần quốc tế hoá những phần mềm của mình bằng cách sử dụng Unicode và dịch các thông báo sang giao diện của người sử dụng. Tuy nhiên, như ở chương này chúng ta sẽ thấy, còn nhiều vấn đề liên quan đến việc quốc tế hoá trong lập trình hơn là sự hỗ trợ của Unicode. Nhiều hệ điều hành, thậm chí nhiều trình duyệt có thể không cần thiết phải hỗ trợ Unicode. Vấn đề quan trọng là phải dịch, chuyên đổi giữa tập các ký tự và font chữ của máy người sử dụng với JVM có sự hỗ trợ Unicode.

Dễ dàng nhận thấy, *ngày tháng, thời gian, tiền tệ lưu hành và các số liệu được biểu diễn khác nhau theo những vùng khác nhau trên thế giới*. Do vậy, chúng ta cần sử dụng một cách nào đó để định dạng lại các thực đơn, tên của các nút nhấn, các xâu thông điệp, các phím nóng, v.v. trong các phần mềm ứng dụng theo những ngôn ngữ khác nhau để đáp ứng những yêu cầu nêu trên.

## 8.2. LỚP LOCALE

Khi cần tìm một phần mềm phù hợp cho thị trường quốc tế thì vấn đề khác biệt hiển nhiên cần chú ý đầu tiên là ngôn ngữ. Vấn đề ngôn ngữ luôn là trở ngại lớn nhất trong giao lưu quốc tế, trong đó có cả vấn đề quốc tế hoá phần mềm. Ngay cả khi một số nước sử dụng chung một ngôn ngữ, nhưng từng quốc gia, dân tộc có thể có những qui định khác nhau, do vậy vẫn phải tạo ra những phần mềm sao cho phù hợp với họ nhất có thể.

Trong tất cả các trường hợp, các thực đơn, tên các nút nhấn, các thông báo của chương trình sẽ phải được dịch sang ngôn ngữ bản địa. Ở đây, các thuật ngữ, các qui định biểu diễn dữ liệu luôn có khá nhiều sự khác biệt rất tinh vi. Ví dụ, các số được biểu diễn theo những khuôn dạng khác nhau giữa Anh (Mỹ) và Đức. Người Đức sử dụng dấu chấm ‘.’ để phân biệt các đơn vị hàng ngàn, triệu và dấu phẩy ‘,’ thập phân để phân biệt phần số nguyên với phần thập phân, như số 124.456,45, nhưng người Anh thì sử dụng ngược lại. Vấn đề tương tự đối với việc hiển thị ngày tháng, thời gian, tiền tệ, v.v. Ngày tháng ở Mỹ được hiển thị theo mẫu tháng/ngày/năm, người Đức sử dụng theo thứ tự ngày/tháng/năm, trong khi người Trung Quốc lại sử dụng năm/tháng/ngày.

Để giải quyết những vấn đề nêu trên, Java cung cấp lớp Locale. Lớp này tập trung mô tả:

- language: xác định ngôn ngữ, ví dụ English, German, Chinese, v.v.
- location: tên quốc gia, bản địa sử dụng ngôn ngữ nêu trên, ví dụ United State (Hoa Kỳ), Germany (Đức), China (Trung Quốc), v.v.
- variant: (Tuỳ chọn) ít sử dụng, thường sử dụng trong những trường hợp ngoại lệ hoặc những tình huống phụ thuộc vào hệ thống qui định. Ví dụ, người Na-Uy sử

dùng hai tập qui tắc viết vần: qui tắc truyền thống là Bokmål và qui tắc mới được gọi là Nynorsk. Khi muốn chọn Bokmål thì khai báo

language = Norwegian, location = Norway, variant = Bokmål

Các ngôn ngữ bản địa được mã hoá trong ISO-639 (Bảng 8.1) bằng hai chữ cái thường cho ngôn ngữ và hai chữ cái hoa trong ISO-3166 (Bảng 8.2) cho tên của các quốc gia. Ta có thể xem danh sách đầy đủ các ngôn ngữ trong bảng mã ISO-639 ở <http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt> và danh sách đầy đủ các quốc gia trong bảng mã ISO-3166 ở <http://www.niso.org/3166.html>.

*Bảng 8.1. Bảng ISO-639 cho tên các ngôn ngữ*

Ngôn ngữ	Mã
Chinese (Trung Quốc)	zh
Danish (Đan Mạch)	da
Dutch (Hà Lan)	nl
English (Anh)	en
French (Pháp)	fr
Finnish (Phản Lan)	fi
German (Đức)	de
Greek (Hy Lạp)	el
Italian (Ý)	it
Japanese (Nhật Bản)	ja
Korean (Triều Tiên)	ko
Norwegian (Na Uy)	no
Portuguese (Bồ Đào Nha)	pt
Spanish (Tây Ba Nha)	sp
Swedish (Thụy Điển)	sv
Turkish (Thổ Nhĩ Kỳ)	tr

Để xác định một vùng bản địa, ta phải khai báo mã ngôn ngữ, mã vùng (quốc gia) và biến thể nếu cần. Ví dụ:

```
Locale germanGermany = new Locale("de", "DE");
```

```
Locale uSA = new Locale("en", "US");
```

```
Locale norwegianNorwayBokmål = new Locale("no", "NO", "B");
```

Để tiện lợi, JDK định nghĩa trong Locale một số ngôn ngữ phổ biến như:

Locale.CHINESE

Locale.ENGLISH

Locale.FRENCH

Locale.GERMAN

Locale.ITALIAN

Locale.JAPANESE, v.v.

Tương tự, một số tên nước được định nghĩa trong Locale.

Locale.CANADA	Locale.CHINA
Locale.FRANCE	Locale.GERMANY
Locale.ITALY	Locale.JAPAN
Locale.UK	Locale.US, v.v.

Bảng 8.2. Bảng ISO-31666 cho tên các quốc gia

Quốc gia	Mã
Austria (Áo)	AT
Belgium (Bỉ)	BE
Canada (Ca – na – da)	CA
China (Trung Quốc)	CN
Denmark (Đan Mạch)	DK
Finland (Phần Lan)	FI
Germany (Đức)	DE
Great Britain (Anh)	GB
Greece (Hy Lạp)	GR
Ireland (Ai – Len)	IE
Italy (Ý)	IT
France (Pháp)	FR
Japan (Nhật bản)	JP
Korea (Triều Tiên)	KR
The Netherlands (Hà Lan)	NL
Norway (Na Uy)	NO
Portugal (Bồ Đào Nha)	PT
Spain (Tây Ba Nha)	ES
Sweden (Thụy Điển)	SE
Switzerland (Thụy Sĩ)	CH
Taiwan (Đài Loan)	TW
Turkey (Thổ Nhĩ Kỳ)	TR
United States (Hoa Kỳ)	US

- static Locale getDefault()

Nhận lại một đối tượng bản địa mặc định.

- `static void setDefault(Locale l)`  
Đặt lại đối tượng bản địa mặc định là l.
- `String getDisplayName()`  
Nhận lại tên của đối tượng hiển thị hiện thời.
- `String getDisplayName(Locale l)`  
Nhận lại tên của đối tượng hiển thị được xác định bởi l cho trước.
- `String getLanguage()`  
Nhận lại mã của ngôn ngữ theo bảng mã ISO-639.
- `String getDisplayLanguage()`  
Nhận lại tên của ngôn ngữ được xác định bởi đối tượng hiện thời.
- `String getDisplayLanguage(Locale l)`  
Nhận lại tên của ngôn ngữ được xác định bởi l cho trước.
- `String getCountry()`  
Nhận lại mã của nước được xác định trong bảng mã ISO-3166.
- `String getDisplayCountry()`  
Nhận lại tên của nước được xác định bởi đối tượng hiện thời.
- `String getDisplayCountry(Locale l)`  
Nhận lại tên của nước được xác định bởi l cho trước.
- `String getVariant()`  
Nhận lại tên của biến thể.
- `String getDisplayVariant()`  
Nhận lại tên của biến thể được xác định bởi đối tượng hiện thời.
- `String getDisplayVariant(Locale l)`  
Nhận lại tên của biến thể được xác định bởi l cho trước.
- `String toString()`  
Nhận lại mô tả về đối tượng hiện thời bao gồm mã tiếng, nước, biến thể được nối với nhau bằng dấu ‘\_’, ví dụ “en\_US”.

### 8.3. CÁC DỮ LIỆU SỐ VÀ ĐƠN VỊ TIỀN TỆ

Như trên đã lưu ý, các đại lượng số và giá trị tiền tệ được biểu diễn theo những khuôn dạng rất khác nhau, tùy thuộc vào từng vùng trên thế giới. Java cung cấp một tuyển tập các

lớp giúp cho việc định dạng và phân loại các giá trị số sang lớp tương ứng trong `java.text`. Để định dạng khuôn mẫu các con số ta có thể thực hiện các bước như sau:

1. Xác định đối tượng bản địa hiện thời
2. Sử dụng phương thức như `getNumberInstance()`, `getCurrencyInstance()`, v.v. để xác định đối tượng định dạng hiện thời.
3. Sử dụng đối tượng định dạng hiện thời để định dạng và phân loại theo yêu cầu.

Ví dụ, ta muốn hiển thị số tiền 123456.78 theo đơn vị tiền và qui định của người Đức.

```
Locale loc = new Locale("de", "DE");
```

```
NumberFormat currFmt = NumberFormat.getCurrencyInstance(loc)
double val = 123456.78;
System.out.println(currFmt.format(val));
```

Kết quả in ra màn hình sẽ là

```
123.456,78 DM
```

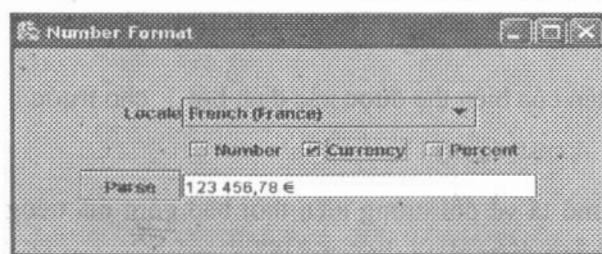
Lưu ý, DM là đơn vị tiền tệ của Đức, được viết ở sau số tiền.

Ngược lại, nếu ta muốn đọc các đại lượng số đã được nhập vào, lưu trữ theo qui ước của một vùng nào đó thì phải sử dụng phương thức `parse()` để phân loại các đại lượng đó. Ví dụ, đọc một giá trị số nhập vào từ `inputField`, chuyển nó về khuôn dạng của vùng bản địa hiện thời trên máy tính.

```
TextField inputField;
```

```
NumberFormat fmt = NumberFormat.getNumberInstance();
Number input = fmt.parse(inputFiled.getText().trim());
double val = input.doubleValue();
```

**Ví dụ 8.1.** Chương trình hiển thị giá trị 123456.78 như là đại lượng số, tiền tệ hay tỷ lệ % theo các nước được lựa chọn. Ví dụ, chọn French(France) và chọn đơn vị tiền tệ, chương trình sẽ hiển thị như sau:



**Hình 8.1.** Chương trình hiển thị số, tiền tệ và tỷ lệ % theo từng vùng

```
// NumberFormatTest.java
```

```
import javax.swing.*;
```

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

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

```
import java.text.*;
import java.util.*;
public class NumberFormatTest{
    public static void main(String args[]){
        JFrame fr = new NumberFormatFrame();
        fr.show();
    }
}
class NumberFormatFrame extends JFrame{

    private Locale[] locales;
    private double currentNumber;

    private JComboBox combo = new JComboBox();
    private JButton parseButton = new JButton("Parse");
    private JTextField numText = new JTextField(30);
    private JCheckBox numberCheckBox = new JCheckBox("Number");
    private JCheckBox currencyCheckBox = new JCheckBox("Currency");
    private JCheckBox percentCheckBox = new JCheckBox("Percent");
    private ButtonGroup group = new ButtonGroup();
    private NumberFormat currentNumberFormat;

    public NumberFormatFrame(){
        setSize(400, 200);
        setTitle("Number Format");
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
        getContentPane().setLayout(new GridBagLayout());
        ActionListener listener = new ActionListener(){
            public void actionPerformed(ActionEvent evt){
                updateDisplay();
            }
        };
    }
}
```

```
JPanel p = new JPanel();
addCheckBox(p, numberCheckBox, group, listener, false);
addCheckBox(p, currencyCheckBox, group, listener, true);
addCheckBox(p, percentCheckBox, group, listener, false);

GridBagConstraints gbc = new GridBagConstraints();
gbc.fill = GridBagConstraints.NONE;
gbc.anchor = GridBagConstraints.EAST;
add(new JLabel("Locale"), gbc, 0, 0, 1, 1);
add(p, gbc, 1, 1, 1, 1);
add(parseButton, gbc, 0, 2, 1, 1);
gbc.anchor = GridBagConstraints.WEST;
add(combo, gbc, 1, 0, 1, 1);
gbc.fill = GridBagConstraints.HORIZONTAL;
add(numText, gbc, 1, 2, 1, 1);

locales = NumberFormat.getAvailableLocales();
for(int i = 0; i < locales.length; i++)
    combo.addItem(locales[i].getDisplayName());
combo.setSelectedItem(Locale.getDefault().getDisplayName());
currentNumber = 123456.78;
updateDisplay();

combo.addActionListener(listener);

parseButton.addActionListener(
    new ActionListener(){
        public void actionPerformed(ActionEvent evt){
            String s = numText.getText();
            try{
                Number n = currentNumberFormat.parse(s);
                if(n != null){
                    currentNumber = n.doubleValue();
                    updateDisplay();
                }else{

```

```
        numText.setText("Parse error: " + s);
    }
} catch(ParseException e){
    numText.setText("Parse error: " + s);
}
}
});
}

public void add(Component c, GridBagConstraints gbc,
    int x, int y, int w, int h){
    gbc.gridx = x;
    gbc.gridy = y;
    gbc.gridheight = h;
    gbc.gridwidth = w;
    getContentPane().add(c, gbc);
}

public void addCheckBox(Container p, JCheckBox checkBox,
    ButtonGroup g, ActionListener listener, boolean v){
    checkBox.setSelected(v);
    checkBox.addActionListener(listener);
    g.add(checkBox);
    p.add(checkBox);
}

public void updateDisplay(){
    Locale currentLocale = locales[combo.getSelectedIndex()];
    currentNumberFormat = null;
    if(numberCheckBox.isSelected())
        currentNumberFormat =
            NumberFormat.getNumberInstance(currentLocale);
    else if(currencyCheckBox.isSelected())
        currentNumberFormat =
            NumberFormat.getCurrencyInstance(currentLocale);
}
```

```

        else if(percentCheckBox.isSelected())
            currentNumberFormat =
                NumberFormat.getPercentInstance(currentLocale);
            String n = currentNumberFormat.format(currentNumber);
            numText.setText(n);
    }
}

```

java.text.NumberFormat

API

- static Locale[] getAvailableLocales()

Nhận lại danh sách các đối tượng khu vực mà NumberFormat hỗ trợ.

- static NumberFormat getInstance()

- static NumberFormat getInstance(Locale l)

- static NumberFormat getCurrencyInstance()

- static NumberFormat getCurrencyInstance(Locale l)

- static NumberFormat getPercentInstance()

- static NumberFormat getPercentInstance(Locale l)

Nhận lại định dạng formatter của đại lượng số, tiền tệ và tỷ lệ % của đối tượng bản địa hiện thời, hoặc đối tượng l cho trước.

- String format (double x)

- String format (long x)

Nhận lại các số double, long theo định dạng hiện thời dưới dạng xâu.

- Number parse (String s)

Chuyển đổi xâu s sang giá trị số tương ứng, là Double nếu đầu vào là số thực, ngược là Long.

- void setMinimumIntegerDigits(int n)

- void getMinimumIntegerDigits()

- void setMaximumIntegerDigits(int n)

- void getMaxnimumIntegerDigits()

- void setMinimumFractionDigits(int n)

- void getMinimumFractionDigits()

- void setMaximumFractionDigits(int n)

- void getMaximumFractionDigits()

Các phương thức trên đặt lại, đọc ra số các chữ số phần nguyên và phần thập phân của đại lượng số.

## 8.4. NGÀY THÁNG VÀ THỜI GIAN

Khi cần định dạng ngày tháng và thời gian thì chúng ta phải quan tâm đến bốn vấn đề chính phụ thuộc vào từng khu vực trên thế giới.

- Tên của các tháng trong năm, tên của ngày trong tuần theo từng ngôn ngữ.
- Thứ tự viết ngày, tháng, năm.
- Lịch Gregorian có phù hợp với khu vực của người sử dụng hay không.
- Mùi giờ của mỗi vùng.

Rất may là lớp DateFormat nhận xử lý tất cả những vấn đề nêu trên. Việc sử dụng nó cũng rất giống như NumberFormat. Trước tiên, ta cần xác định đối tượng khu vực của máy tính. Ta có thể sử dụng đối tượng mặc định hoặc thông qua getAvailableLocales() để nhận được danh sách các đối tượng khu vực mà NumberFormat hỗ trợ. Sau đó gọi một trong ba phương thức sau:

```
fmt = DateFormat.getDateInstance(dateStyle, loc);
fmt = DateFormat.getTimeInstance(timeStyle, loc);
fmt = DateFormat.getDateTimeInstance(dateStyle, timeStyle, loc);
```

Trong đó dateStyle, timeStyle có thể là một trong các hằng sau DateFormat.DEFAULT, DateFormat.FULL, DateFormat.LONG, DateFormat.MEDIUM, DateFormat.SHORT, còn loc là đối tượng khu vực.

Ta có thể sử dụng phương thức parse() của NumberFormat để phân tích Date và Time. Ví dụ, việc nhập vào ngày tháng và chuyển đổi sang dạng qui định của máy mặc định được thực hiện như sau.

```
TextField inputField;
DateFormat fmt =
    NumberFormat.getDateInstance(DateFormat.MEDIUM);
Date input = fmt.parse(inputFiled.getText().trim());
```

Nếu dữ liệu nhập vào không tương thích thì sẽ phát sinh ngoại lệ thuộc loại ParseException. Ví dụ, nếu format nhập là MEDIUM theo qui ước của Mỹ thì dữ liệu ngày tháng nhập vào phải có dạng

Sep 18, 2005

Nếu người sử dụng nhập vào

Sep 18 2005

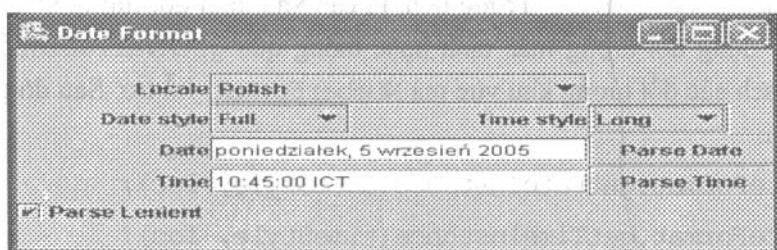
không có dấu ‘,’ giữa ngày và năm, hoặc dạng

18/9/2005

thì chương trình sẽ phát sinh lỗi.

Cờ lenient báo cho chương trình thông dịch chỉnh lại ngày tháng cho phù hợp với lịch Dương, nghĩa là tha thứ một số lỗi trong nhập liệu. Ví dụ, nếu cờ này được bật (nhận giá trị true) thì hệ thống sẽ tự động chuyển ngày tháng nhập vào không thật chính xác theo lịch, ví dụ February 30, 1999 thành March 2, 1999. Điều này có vẻ nguy hiểm, song hệ thống Java vẫn chuyển theo mặc định, nếu không đặt lại cờ lenient.

**Ví dụ 8.2.** Viết chương trình hiển thị ngày giờ của máy tính hiện thời theo các khu vực và theo dạng: DEFAULT, LONG, MEDIUM, SHORT được người sử dụng lựa chọn ở các ComboBox: Locale, Date style, Time style. Ví dụ, khi người sử dụng chọn vùng Ba Lan (Polish) và dạng hiển thị ngày là FULL, dạng thời gian là LONG thì hệ thống sẽ hiển thi:



Hình 8.2. Chương trình hiển thị ngày giờ theo qui ước của Ba Lan

trong đó Poniedziałek là thứ Hai, Wrzesień là tháng Chín.

```
// DateFormatTest.java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.text.*;
import java.util.*;

public class DateFormatTest{
    public static void main(String args[]){
        JFrame fr = new DateFormatFrame();
        fr.show();
    }
}

class DateFormatFrame extends JFrame{
    private Locale[] locales;
    private Date currentDate;
    private Date currentTime;
```

```
private DateFormat currentDateFormat;
private DateFormat currentTimeFormat;

private JComboBox combo = new JComboBox();
private EnumCombo dateStyleCombo
    = new EnumCombo(DateFormat.class,
        new String[]{"Default", "Full", "Long", "Medium", "Short"});
private EnumCombo timeStyleCombo
    = new EnumCombo(DateFormat.class,
        new String[]{"Default", "Full", "Long", "Medium", "Short"});

private JButton dateButton = new JButton("Parse Date");
private JButton timeButton = new JButton("Parse Time");
private JTextField dateText = new JTextField(30);
private JTextField timeText = new JTextField(30);
private JTextField parseText = new JTextField(30);
private JCheckBox lenientCheckBox = new JCheckBox("Parse Lenient", true);

public DateFormatFrame(){
    setSize(400, 200);
    setTitle("Date Format");
    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
    getContentPane().setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.fill = GridBagConstraints.NONE;
    gbc.anchor = GridBagConstraints.EAST;
    add(new JLabel("Locale"), gbc, 0, 0, 1, 1);
    add(new JLabel("Date style"), gbc, 0, 1, 1, 1);
    add(new JLabel("Time style"), gbc, 2, 1, 1, 1);
}
```

```
add(new JLabel("Date"), gbc, 0, 2, 1, 1);
add(new JLabel("Time"), gbc, 0, 3, 1, 1);

gbc.anchor = GridBagConstraints.WEST;
add(combo, gbc, 1, 0, 2, 1);
add(dateStyleCombo, gbc, 1, 1, 1, 1);
add(timeStyleCombo, gbc, 3, 1, 1, 1);
add(dateButton, gbc, 3, 2, 1, 1);
add(timeButton, gbc, 3, 3, 1, 1);
add(lenientCheckBox, gbc, 0, 4, 1, 1);
gbc.fill = GridBagConstraints.HORIZONTAL;
add(dateText, gbc, 1, 2, 2, 1);
add(timeText, gbc, 1, 3, 2, 1);

locales = DateFormat.getAvailableLocales();
for(int i = 0; i < locales.length; i++)
    combo.addItem(locales[i].getDisplayName());
combo.setSelectedItem(Locale.getDefault().getDisplayName());
currentDate = new Date();
currentTime = new Date();
updateDisplay();

ActionListener listener =
    new ActionListener(){
        public void actionPerformed(ActionEvent evt){
            updateDisplay();
        }
    };
combo.addActionListener(listener);
dateStyleCombo.addActionListener(listener);
timeStyleCombo.addActionListener(listener);

dateButton.addActionListener
    new ActionListener(){
        public void actionPerformed(ActionEvent evt){
```

```
String d = dateText.getText();
try{
    currentDateFormat.setLenient(
        lenientCheckBox.isSelected());
    Date date = currentDateFormat.parse(d);
    currentDate = date;
    updateDisplay();
} catch(ParseException e){
    dateText.setText("Parse error: " + d);
}
catch(IllegalArgumentException e){
    dateText.setText("Argument error: " + d);
}
}

});

timeButton.addActionListener(
    new ActionListener(){
        public void actionPerformed(ActionEvent evt){
            String t = timeText.getText();
            try{
                currentDateFormat.setLenient(
                    lenientCheckBox.isSelected());
                Date date = currentDateFormat.parse(t);
                currentTime = date;
                updateDisplay();
            } catch(ParseException e){
                timeText.setText("Parse error: " + t);
            }
            catch(IllegalArgumentException e){
                timeText.setText("Argument error: " + t);
            }
        }
    });
}
```

```
public void add(Component c, GridBagConstraints gbc,
    int x, int y, int w, int h){
    gbc.gridx = x;
    gbc.gridy = y;
    gbc.gridheight = h;
    gbc.gridwidth = w;
    getContentPane().add(c, gbc);
}

public void updateDisplay(){
    Locale currentLocale =
        locales[combo.getSelectedIndex()];
    int dateStyle = dateStyleCombo.getValue();
    currentDateFormat =
        DateFormat.getDateInstance(dateStyle, currentLocale);
    String s = currentDateFormat.format(currentDate);
    dateText.setText(s);

    int timeStyle = timeStyleCombo.getValue();
    currentTimeFormat =
        DateFormat.getTimeInstance(timeStyle, currentLocale);
    String t = currentTimeFormat.format(currentTime);
    timeText.setText(t);
}

class EnumCombo extends JComboBox {
    public EnumCombo(Class cl, String[] labels){
        for(int i = 0; i < labels.length; i++){
            String label = labels[i];
            String name = label.toUpperCase().replace(' ', '_');
            int value = 0;
            try{
                java.lang.reflect.Field f = cl.getField(name);
                value = f.getInt(cl);
            }
            catch(NoSuchFieldException e){} // ignore
        }
    }
}
```

```

        } catch (Exception e) {
            label = "(" + label + ")";
            table.put(label, new Integer(value));
            addItem(label);
        }
    setSelectedItem(labels[0]);
}

public int getValue(){
    return ((Integer)table.get(getSelectedItem())).intValue();
}
private Map table = new HashMap();
}

```

**java.text.DateFormat****API**

- static Locale[] getAvailableLocales()

Nhận lại một danh sách các đối tượng khu vực mà DateFormat hỗ trợ.

- static DateFormat getDateInstance(int dateStyle)
- static DateFormat getDateInstance(int dateStyle, Locale l)
- static DateFormat getTimeInstance(int timeStyle)
- static DateFormat getDateTimeInstance(int timeStyle, Locale l)
- static DateFormat getDateTimeInstance(int dateStyle, int timeStyle)
- static DateFormat getDateTimeInstance(int dateStyle, int timeStyle, Locale l)

Nhận lại khuôn dạng cho ngày, giờ, ngày và giờ theo khu vực của máy mặc định hoặc khu vực cho trước bởi l.

- String format(String s)

Nhận lại xâu kết quả của ngày / giờ theo khuôn dạng qui định.

- Date parse(String s)

Chuyển xâu s sang ngày / giờ theo khuôn dạng qui định.

- void setLenient(boolean b)

- boolean isLenient()

Đặt cờ và kiểm tra trạng thái của cờ lenient phục vụ cho việc chỉnh sửa ngày theo lịch khi dữ liệu đầu vào không tương thích.

- void setCalendar(Calendar cal)
- Calendar getCalendar()

Đặt và đọc đối tượng lịch biểu để tách ra các thông tin về ngày, tháng, năm, giờ, phút, giây. Lịch biểu mặc định là Gregorian còn được gọi là Lịch Dương.

- void setTimezone(TimeZone f)
- TimeZone getTimeZone()

Đặt và đọc đối tượng múi giờ phục vụ cho việc định dạng cho đúng với khu vực hiện thời và đúng lịch biểu.

- void setNumberFormat(NumberFormat f)
- NumberFormat getNumberFormat()

Đặt và đọc đối tượng khuôn số liệu sử dụng để định dạng ngày, tháng, năm, giờ, phút, giây.

## 8.5. VĂN BẢN TEXT

Việc hiển thị các văn bản cũng rất nhiều vấn đề cần phải chú ý. Trong phần này chúng ta tìm hiểu cách biểu diễn và thao tác trên các xâu văn bản.

### 8.5.1. Sắp xếp văn bản

Sắp xếp các xâu văn bản theo một thứ tự nào đó là vấn đề tương đối đơn giản, nếu chúng được xây dựng từ các ký tự trong bảng mã ASCII (tiếng Anh). Thường ta có thể sử dụng

`s1.compareTo(s2)`

để so sánh xâu s1 với s2 để sắp xếp chúng theo thứ tự từ điển.

Nhưng nếu giữa các xâu có một số chữ thường, chữ in hoa lẫn lộn thì dãy các xâu có thể không được sắp theo thứ tự mà ta mong muốn. Biết rằng, phương thức `compareTo()` trong Java sử dụng bộ mã Unicode để sắp xếp. Trong bảng mã Unicode, mã của chữ cái thường lớn hơn mã chữ cái in hoa, các chữ có dấu thâm chí còn có mã cao hơn. Ví dụ, dãy gồm năm xâu: “An”, “Áng”, “anna”, “Phe”, “phan” nếu sắp theo thứ tự với phương thức `compareTo()` sẽ được dãy:

“An”, “Phe”, “anna”, “phan”, “Áng”

Nhưng theo cách sắp xếp của nhiều nước, trong đó có Việt Nam thì chữ Á phải đứng sau A, nghĩa là dãy trên cần phải xếp theo thứ tự như sau:

“An”, “Áng”, “anna”, “phan”, “Phe”

Ngược lại, người sử dụng một số nước khác lại không chấp nhận thứ tự nêu trên, ví dụ Đan Mạch. Theo qui định của họ, chữ cái có dấu (có trọng âm) khác với chữ không có dấu,

nghĩa là chữ ‘Á’ phải xếp sau ‘Z’. Do vậy, theo yêu cầu của người Đan Mạch, dây trên phải được xếp theo thứ tự như sau:

“An”, “anna”, “phan”, “Phe”, “Áng”

Những vấn đề nêu trên có thể giải quyết được bằng cách sử dụng lớp Collator để đổi chiều. Như thường lệ, ta phải xác định đối tượng khu vực của Locale. Sau đó gọi getInstance() để nhận được đối tượng của Collator. Cuối cùng để so sánh ta sử dụng compare() của Collator.

```
Locale loc = ...
Collator coll = Collator.getInstance(loc);
if(coll.compare(s1, s2) < 0) // s1 đứng trước s2
```

Điều quan trọng hơn, lớp Collator cài đặt giao diện Comparator. Vì thế ta có thể sử dụng Collections.sort(strings, coll) để sắp xếp strings theo thứ tự của đối tượng coll.

Hơn nữa, trong lý thuyết sắp xếp từ vựng, người ta còn phân chia sự khác nhau giữa các ký tự thành ba mức: *cơ sở*, *thứ cấp* và *tam cấp*. Ví dụ, trong tiếng Anh, sự khác nhau giữa ‘A’ và ‘Z’ là cơ sở (PRIMARY), sự khác nhau giữa ‘A’ và ‘Á’ là thứ cấp (SECONDARY) và giữa ‘A’ và ‘a’ là tam cấp (TERTIARY). Hiển nhiên ta có thể đặt lại mức độ cho công việc sắp xếp.

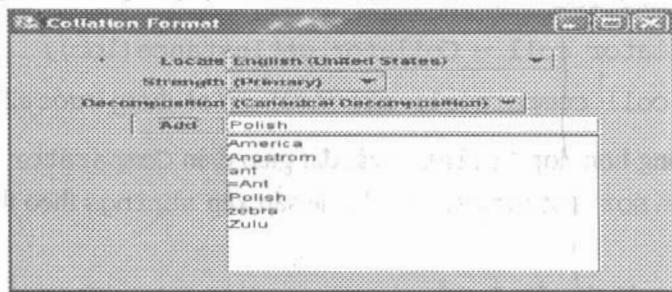
Việc chọn mức độ Collator.PRIMARY là muốn báo cho hệ thống biết rằng chỉ cần chú ý vào sự khác nhau cơ sở. Chúng ta hãy xét ví dụ sau.

```
// Giả sử dụng khu vực của những người nói tiếng Anh
String s1 = "Angstrom";
String s1 = "Ángstrom";
coll.setStrength(Collator.PRIMARY)
if(coll.compare(s1, s2) == 0)
    System.out.println("Giống nhau"); // in ra "Giống nhau"
else System.out.println("Khác nhau");
coll.setStrength(Collator.SECONDARY)
if(coll.compare(s1, s2) == 0)
    System.out.println("Giống nhau");
else System.out.println("Khác nhau"); // in ra "Khác nhau"
```

Vấn đề cuối cùng liên quan đến việc sắp xếp là các cách thức “phân rã” (decomposition mode). Cách thức hay sử dụng nhất là phân rã chuẩn tắc (canonical decomposition), nó thường được đặt mặc định. Nếu ta chọn cách không phân rã (no decomposition) thì các chữ có dấu sẽ không được tách ra khỏi dạng cơ sở và dấu của

chúng. Sự lựa chọn này sẽ giúp cho việc sắp xếp nhanh hơn, song chỉ cho kết quả đúng nếu dãy cần sắp xếp không có các chữ có dấu. Khả năng lựa chọn thứ ba là phân rã hoàn toàn (full decomposition), được sử dụng để phân rã các biến thể của Unicode.

**Ví dụ 8.3.** Chương trình thực hiện sắp xếp một dãy các xâu được nhập vào theo từng khu vực (Locate), mức độ (Strength) và cách thức (Decomposition) được lựa chọn trong các hộp ComboBox. Mỗi lần nhập một xâu, hay thay đổi một trong các lựa chọn trên thì dãy các xâu đó sẽ được sắp xếp lại. Dấu '=' chỉ ra rằng những mục đó được xem là đồng nhất với mục trước đó trong thứ tự sắp xếp (xem hình 8.3).



Hình 8.3. Chương trình sắp xếp dãy các xâu theo các tùy chọn

```
// CollationTest.java
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.text.*;
import java.util.*;
import java.util.List;

public class CollationTest{
    public static void main(String args[]){
        JFrame fr = new CollationFrame();
        fr.show();
    }
}

class CollationFrame extends JFrame{
    String[] d = {"America", "ant", "Zulu", "zebra", "Ant", "Angstrom"};
    private Locale[] locales;
}
```

```
private List strings = new ArrayList();
private Collator currentCollator;

private JComboBox combo = new JComboBox();
private EnumCombo strengthCombo
    = new EnumCombo(DateFormat.class,
                    new String[]{"Primary", "Secondary", "Tertiary"});
private EnumCombo decompositionCombo
    = new EnumCombo(DateFormat.class,
                    new String[]{"Canonical Decomposition",
                                "No Decomposition", "Full Decomposition"});

private JButton addButton = new JButton("Add");
private JTextField newWord = new JTextField(20);
private JTextArea sortedWords = new JTextArea(10, 20);

public CollationFrame() {
    setSize(400, 400);
    setTitle("Collation Format");
    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
}

getContentPane().setLayout(new GridBagLayout());
GridBagConstraints gbc = new GridBagConstraints();
gbc.fill = GridBagConstraints.NONE;
gbc.anchor = GridBagConstraints.EAST;
add(new JLabel("Locale"), gbc, 0, 0, 1, 1);
add(new JLabel("Strength"), gbc, 0, 1, 1, 1);
add(new JLabel("Decomposition"), gbc, 0, 2, 1, 1);
add(addButton, gbc, 0, 3, 1, 1);

gbc.anchor = GridBagConstraints.WEST;
add(combo, gbc, 1, 0, 1, 1);
```

```
        add(strengthCombo, gbc, 1, 1, 1, 1);
        add(decompositionCombo, gbc, 1, 2, 1, 1);

        gbc.fill = GridBagConstraints.HORIZONTAL;
        add(newWord, gbc, 1, 3, 1, 1);
        gbc.fill = GridBagConstraints.BOTH;
        add(new JScrollPane(sortedWords), gbc, 1, 4, 1, 1);

        locales = Collator.getAvailableLocales();
        for(int i = 0; i < locales.length; i++)
            combo.addItem(locales[i].getDisplayName());
        combo.setSelectedItem(Locale.getDefault().getDisplayName());

        for(int i = 0; i < d.length; i++)
            strings.add(d[i]);

        updateDisplay();

        addButton.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent evt){
                    strings.add(newWord.getText());
                    updateDisplay();
                }
            });
    });

    ActionListener listener =
        new ActionListener(){
            public void actionPerformed(ActionEvent evt){
                updateDisplay();
            }
        };
    combo.addActionListener(listener);
    strengthCombo.addActionListener(listener);
    decompositionCombo.addActionListener(listener);
}
```

```
public void add(Component c, GridBagConstraints gbc,
    int x, int y, int w, int h){
    gbc.gridx = x;
    gbc.gridy = y;
    gbc.gridheight = h;
    gbc.gridwidth = w;
    getContentPane().add(c, gbc);
}

public void updateDisplay(){
    Locale currentLocale = locales[combo.getSelectedIndex()];
    currentCollator =
        Collator.getInstance(currentLocale);
    currentCollator.setStrength(strengthCombo.getValue());
    currentCollator.setDecomposition(decompositionCombo.getValue());
    Collections.sort(strings, currentCollator);

    sortedWords.setText("");
    for(int i = 0; i < strings.size(); i++){
        String s = (String)strings.get(i);
        if(i > 0 && currentCollator.compare(s, strings.get(i-1)) == 0 )
            sortedWords.append("=");
        sortedWords.append(s + "\n");
    }
}

class EnumCombo extends JComboBox {
    public EnumCombo(Class cl, String[] labels){
        for(int i = 0; i < labels.length; i++){
            String label = labels[i];
            String name = label.toUpperCase().replace(' ', '_');
            int value = 0;
            try{
                java.lang.reflect.Field f = cl.getField(name);
```

```

        value = f.getInt(cl);
    } catch(Exception e){
        label = "(" + label + ")";
    }
    table.put(label, new Integer(value));
    addItem(label);
}
setSelectedItem(labels[0]);
}

public int getValue(){
    return ((Integer)table.get(getSelectedItem())).intValue();
}
private Map table = new HashMap();
}

```

## java.text.Collator

## API

- static Locale[] getAvailableLocales()

Nhận lại một mảng các đối tượng của Locale mà Collator hỗ trợ.

- static Collator getInstance()

- static Collator getInstance(Locale l)

Đưa lại đối tượng đối sánh Collator sử dụng cho khu vực mặc định hay khu vực được cho bởi l.

- int compare(String a, String b)

So sánh hai xâu, cho lại giá trị âm nếu a đứng trước b, bằng 0 nếu a và b đồng nhất với nhau, giá trị dương trong trường hợp ngược lại.

- boolean equals(String a, String b)

Cho lại giá trị true nếu hai xâu a, b đồng nhất với nhau, ngược lại là false.

- void setStrength(int strength)

- int getStrength()

Đặt và nhận lại mức độ của bộ đối sánh hiện thời. Tham số strength có thể là một trong các hằng Collator.PRIMARY, Collator.SECONDARY, Collator.TERTIARY.

- void setDecomposition(int decomp)

- int getdDecomposition()

Đặt và lấy ra đối tượng phân tích của bộ đối sánh. decompose có thể là Collator.NO\_DECOMPOSITION, Collator.CANONICAL\_DECOMPOSITION, Collator.FULL\_DECOMPOSITION.

### 8.5.2. Ranh giới của các văn bản Text

Một vấn đề rất quan trọng trong xử lý văn bản là phải xác định được các ranh giới (phản tử ngắt) giữa các ký tự, từ, câu và các dòng.

*Ngắt từ* là ký tự bắt đầu và kết thúc của các từ. Trong tiếng Anh, người ta định nghĩa *một từ* là một dãy các ký tự, trong đó không có dấu cách ‘ ’. Ví dụ, câu

The quick, brown fox jump-ed over the lazy dog.

được ngắt bởi các dấu cách ‘ ’, dấu ‘,’ và dấu ‘.’ thành các từ như sau:

The| |quick|, |brown| |fox| |jump-ed| |over| |the| |lazy| |dog| .|

*Ngắt dòng* là các vị trí mà ở đó một dòng bị ngắt ở trên màn hình hoặc ở trên văn bản được in ra. Trong văn bản tiếng Anh, các dòng được ngắt bởi một từ trước đó hoặc bởi một dấu gạch nối. Lưu ý, các điểm ngắt dòng là những vị trí mà ở đó một dòng có thể bị ngắt chứ không nhất thiết phải là những điểm mà nó bị ngắt thực sự. Ví dụ, dòng sau có các điểm ngắt là:

The |quick, |brown |fox |jump-ed |over |the |lazy |dog.|

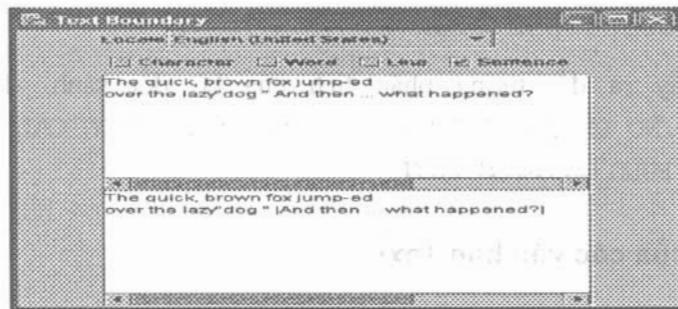
Việc xác định ranh giới giữa các ký tự, các từ và các dòng là tương đối đơn giản đối với ngôn ngữ tiếng Anh, tiếng Việt và nhiều ngôn ngữ của châu Âu, Á. Tuy nhiên, nó có thể rất phức tạp đối với một số ngôn ngữ khác, ví dụ như tiếng Hindi.

Tiếp theo chúng ta tìm hiểu về việc *ngắt các câu văn bản*. Trong tiếng Anh (tiếng Việt), một câu bị ngắt bởi các dấu chấm câu ‘.’, các dấu cảm thán ‘!’ và dấu ‘?’.

Ta có thể sử dụng lớp BreakIterator để tìm các chỗ ngắt của ký tự, từ, dòng và các câu trong một văn bản. Ta cũng có thể sử dụng chúng để soạn thảo, hiển thị và in các tài liệu văn bản.

```
Locale loc = ...; // Xác định khu vực của hệ thống
// Xác định đối tượng tách từ
BreakIterator wordIter = BreakIterator.getWordInstance(loc);
String msg = "The quick, brown fox";
wordIter.setText(msg); // Đặt msg vào bộ lặp để tách các từ
```

**Ví dụ 8.4.** Chương trình cho phép nhập vào một văn bản ở phía trên và lựa chọn Character, Word, Line, Sentence để xác định các ranh giới giữa các ký tự, từ, dòng hay câu tương ứng theo khu vực đã chọn ở ComboBox (Xem hình 8.4).



Hình 8.4. Chương trình xác định các ranh giới trong văn bản

```
// TextBoundaryTest.java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.text.*;
import java.util.*;

public class TextBoundaryTest{
    public static void main(String args[]){
        JFrame fr = new TextBoundaryFrame();
        fr.show();
    }
}

class TextBoundaryFrame extends JFrame{
    private Locale[] locales;
    private BreakIterator currentBreak;
    private JComboBox combo = new JComboBox();
    private JTextArea inputText = new JTextArea(6, 40);
    private JTextArea outputText = new JTextArea(6, 40);
    private ButtonGroup cbButton = new ButtonGroup();
    private JCheckBox charBox = new JCheckBox("Character");
    private JCheckBox wordBox = new JCheckBox("Word");
    private JCheckBox lineBox = new JCheckBox("Line");
    private JCheckBox sentenceBox = new JCheckBox("Sentence");
}
```

```
public TextBoundaryFrame() {
    setSize(400, 400);
    setTitle("Text Boundary");
    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
}

ActionListener listener =
    new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            updateDisplay();
        }
};

JPanel p = new JPanel();
addCheckBox(p, charBox, cbButton, listener, false);
addCheckBox(p, wordBox, cbButton, listener, false);
addCheckBox(p, lineBox, cbButton, listener, false);
addCheckBox(p, sentenceBox, cbButton, listener, true);

getContentPane().setLayout(new GridBagLayout());
GridBagConstraints gbc = new GridBagConstraints();
gbc.fill = GridBagConstraints.NONE;
gbc.anchor = GridBagConstraints.EAST;
add(new JLabel("Locale"), gbc, 0, 0, 1, 1);
gbc.anchor = GridBagConstraints.WEST;
add(combo, gbc, 1, 0, 1, 1);
add(p, gbc, 0, 1, 2, 1);
gbc.fill = GridBagConstraints.BOTH;
gbc.weighty = 100;
add(new JScrollPane(inputText), gbc, 0, 2, 2, 1);
add(new JScrollPane(outputText), gbc, 0, 3, 2, 1);
```

```
locales = BreakIterator.getAvailableLocales();
for(int i = 0; i < locales.length; i++)
    combo.addItem(locales[i].getDisplayName());
combo.setSelectedItem(Locale.getDefault().getDisplayName());

combo.addActionListener(listener);

inputText.setText("The quick, brown fox jump-ed\n" +
    "over the lazy\"dog.\" And then ... what happened?");
updateDisplay();
}

public void addCheckBox(Container p, JCheckBox checkBox,
    ButtonGroup g, ActionListener listener, boolean v)
{
    checkBox.setSelected(v);
    checkBox.addActionListener(listener);
    g.add(checkBox);
    p.add(checkBox);
}

public void add(Component c, GridBagConstraints gbc,
    int x, int y, int w, int h){
    gbc.gridx = x;
    gbc.gridy = y;
    gbc.gridheight = h;
    gbc.gridwidth = w;
    getContentPane().add(c, gbc);
}

public void updateDisplay(){
    Locale currentLocale = locales[combo.getSelectedIndex()];
    BreakIterator currentBreak = null;
    if(charBox.isSelected())
        currentBreak = BreakIterator.getCharacterInstance(currentLocale);
```

```

        else if(wordBox.isSelected())
            currentBreak = BreakIterator.getWordInstance(currentLocale);
        else if(lineBox.isSelected())
            currentBreak = BreakIterator.getLineInstance(currentLocale);
        else if(sentenceBox.isSelected())
            currentBreak = BreakIterator.getSentenceInstance(currentLocale);
        String text = inputText.getText();
        currentBreak.setText(text);
        outputText.setText("");
        int from = currentBreak.first();
        int to;
        while((to = currentBreak.next()) != BreakIterator.DONE){
            outputText.append(text.substring(from, to) + "|");
            from = to;
        }
    }
}

```

java.text.BreakIterator

API

- static Locale[] getAvailableLocales()

Nhận lại một mảng các đối tượng của Locale mà BreakIterator hỗ trợ.

- static BreakIterator getCharacterInstance()

- static BreakIterator getCharacterInstance(Locale l)

- static BreakIterator getWordInstance()

- static BreakIterator getWordInstance(Locale, l)

- static BreakIterator getLineInstance()

- static BreakIterator getLineInstance(Locale, l)

- static BreakIterator getSentenceInstance()

- static BreakIterator getSentenceInstance(Locale, l)

Nhận lại bộ lặp để tách các ký tự, từ, dòng và câu theo khu vực mặc định hay khu vực được cho bởi l.

- void setText(String text)

- void setText(CharacterIterator text)

- `CharacterIterator getText()`

Đặt và nhận lại văn bản text để phân tích.

- `int first()`

Chuyển về vị trí của ranh giới đầu tiên của xâu đang đọc và đưa lại chỉ số của nó.

- `int next()`

Chuyển về ranh giới tiếp theo và đưa lại chỉ số của vị trí đó. Nếu đạt đến cuối xâu thì trả lại `BreakIterator.DONE`.

- `int previous()`

Chuyển về ranh giới phía trước và đưa lại chỉ số của vị trí đó. Nếu đạt đến đầu xâu thì trả lại `BreakIterator.DONE`.

- `int last()`

Chuyển về ranh giới cuối cùng và đưa lại chỉ số của vị trí đó.

- `int current()`

Cho lại vị trí của ranh giới hiện thời.

- `int next(int n)`

Chuyển đến ranh giới thứ n kể từ vị trí hiện thời và trả lại chỉ số của vị trí đó. Nếu n là số âm thì chuyển về phía trước, ngược lại chuyển về phía sau. Nếu đạt đến đầu xâu thì trả lại `BreakIterator.DONE`.

## 8.6. BỘC TÀI NGUYÊN

Khi thực hiện bản địa hóa một phần mềm, ta thường gặp phải vấn đề gây nhiều phiền phức là phải dịch rất nhiều thông báo, tên gọi của các nút, nhãn, các mục, v.v. sang tiếng bản địa. Để cho tiện lợi hơn, ta có thể định nghĩa các xâu ở bên ngoài và đặt vào các *bộc tài nguyên*, độc lập với chương trình, được gọi là *tệp tài nguyên*. Các tệp tài nguyên được biên soạn, dịch sang tiếng bản địa mà không ảnh hưởng gì đến mã nguồn của chương trình.

### 8.6.1. Bản địa hóa tài nguyên

Công nghệ Java cung cấp lớp  `ResourceBundle` hỗ trợ để ta thực hiện được các yêu cầu trên. Ta có thể tạo ra một lớp khác cho mỗi khu vực, sau đó gọi phương thức `getBundle()` của  `ResourceBundle` để xác định tự động lớp theo yêu cầu. Các lớp đó phải kế thừa từ lớp  `ResourceBundle`.

Để thực hiện được các yêu cầu trên, ta phải tuân theo qui ước đặt tên cho các lớp này tương ứng với từng khu vực. Ví dụ, tài nguyên dành riêng người Đức thì lớp đó là

ProgramResources\_de\_de, còn tài nguyên cho khu vực những người nói tiếng Đức là ProgramResourcesde\_de\_DE. Ta sử dụng qui ước

ProgramResources\_language\_country

để đặt tên các tệp tài nguyên cho tất cả các nước theo country cụ thể, và

ProgramResources\_language

để đặt tên các tệp tài nguyên cho tất cả các khu vực nói tiếng language.

Khi đã có một lớp bọc tài nguyên, ta có thể tải nó xuống bằng lệnh

```
 ResourceBundle currentResource =
```

```
 ResourceBundle.getBundle("ProgramResources", currentLocale);
```

Hàm getBundle () sẽ thử tải xuống một trong các lớp tài nguyên sau:

ProgramResources\_language\_country\_variant

ProgramResources\_language\_country

ProgramResources\_language

ProgramResources

Nếu không tải xuống được thì nó thử làm một lần nữa, nhưng lần này không phải áp dụng với đối tượng khu vực hiện thời mà là với khu vực mặc định. Trường hợp không thể tải xuống được thì nó phát sinh ra ngoại lệ MissingResourceException.

### 8.6.2. Đặt tài nguyên vào các bọc

Trong Java, ta có thể đặt tài nguyên vào trong các lớp được mở rộng từ lớp ResourceBundle. Mỗi bọc tài nguyên cài đặt như là một bảng tra cứu tìm các từ tương ứng theo các từ khoá.

Khi thiết kế chương trình, ta phải cung cấp xâu chính (từ khoá) cho mỗi lần thiết lập vùng bản địa và sau này sử dụng nó để truy tìm đối tượng khu vực.

```
 String computeButtonLabel
     = resources.getString("computeButton");
```

Ta cũng có thể sử dụng hàm getObject () để xác định đối tượng bất kỳ từ bọc tài nguyên.

```
 Color backgroundColor
     = (Color)resources.getObject("backgroundColor");
 double[] paperSize
     = (double[])resources.getObject("defaultPaperSize");
```

Để cài đặt lớp bọc tài nguyên riêng, ta phải cài đặt hai hàm

```
 Enumeration getKeys()
```

```
 Object handleGetObject(String key)
```

Ví dụ, lớp sau đây cung cấp tài nguyên để dịch cho khu vực nói Đức và người Mỹ nói tiếng Anh.

```
public class ProgramResources extends ResourceBundle{
    // Đặt hàm getKeys() vào lớp cơ sở
    public Enumeration getKeys(){
        return Collections.enumeration(Arrays.asList(keys));
    }
    private String[] keys = {"computeButton",
                            "backgroundColor", "defaultPaperSize"};
}

public class ProgramResources_de extends ProgramResources{
    // Lớp tài nguyên cho khu vực nói tiếng Đức
    public Object handleGetObject(String key){
        if(key.equals("computeButton"))
            return "Rechmen";
        else if(key.equals("backgroundColor"))
            return Color.black;
        else if(key.equals("defaultPaperSize"))
            return new double[]{210, 297};
    }
}

public class ProgramResources_en_US extends ProgramResources{
    // Lớp tài nguyên cho khu vực người Mỹ nói tiếng Anh
    public Object handleGetObject(String key){
        if(key.equals("computeButton"))
            return "Compute";
        else if(key.equals("backgroundColor"))
            return Color.blue;
        else if(key.equals("defaultPaperSize"))
            return new double[]{216, 279};
    }
}
```

Hiển nhiên, công việc viết các mã lệnh như trên là khá buồn chán. Thư viện chuẩn của Java cung cấp hai loại lớp tài nguyên khác là `ListResourceBundle` và `PropertyResourceBundle` giúp cho việc tổ chức chúng dễ dàng hơn.

a) `ListResourceBundle` cho phép đặt tài nguyên vào một mảng các đối tượng.

```
public class ProgramResources_language_country
    extends ListResourceBundle{
    public Object[] getContents(){
        return contents;
    }
    static final Object[][] contents = {
        // Các thông tin được bản địa hóa
    };
}
```

Ví dụ trên có thể viết lại như sau:

```
public class ProgramResources_de extends ListResourceBundle{
    public Object[] getContents(){
        return contents;
    }
    static final Object[][] contents = {
        {"computeButton", "Rechmen"},
        {"backgroundColor", Color.black}
        {"defaultPaperSize", new double[]{210, 297}}
    };
}

public class ProgramResources_en_US extends ListResourceBundle{
    public Object[] getContents(){
        return contents;
    }
    static final Object[][] contents = {
        {"computeButton", "Compute"},
        {"backgroundColor", Color.blue}
        {"defaultPaperSize", new double[]{216, 279}}
    };
}
```

b) `PropertyResourceBundle` cho phép đặt các tài nguyên một cách linh hoạt hơn. Ta có thể tạo ra *tệp tài sản* của tài nguyên đơn giản là *tệp văn bản bao gồm từng cặp giá trị trên mỗi dòng*, viết dưới dạng

```
computeButton=Compute
backgroundColor=blue
defaultPaperSize=216x279
```

Sau đó ta mở tệp tài sản này và truyền nó vào cho toán tử tạo lập của `PropertyResourceBundle`.

```
InputStream in = ...; // Mở tệp tài sản
PropertyResourceBundle currentResources
= new PropertyResourceBundle(in);
```

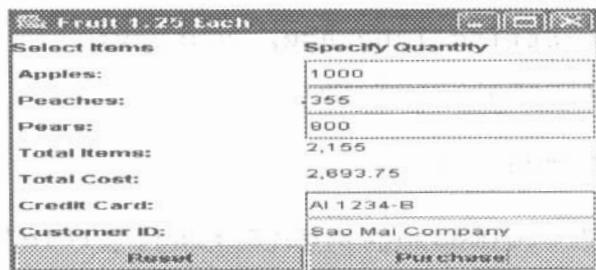
Các tệp tài sản thường có tên trùng với tên của các tài nguyên và có phần mở rộng là `.property`, ví dụ `ProgramResources_en_US.property` là tệp tài sản dùng cho những người Mỹ nói tiếng Anh. Ta có thể mở tệp tài sản thông qua hàm `getResourceAsStream()` của lớp `Class` như sau:

```
in = Program.class.getResourceAsStream(
    "ProgramResources_en_US.property");
PropertyResourceBundle currentResources
= new PropertyResourceBundle(in);
```

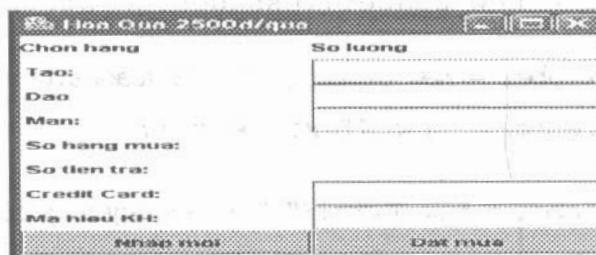
**Lưu ý:** Như vậy đối với những chương trình cần được bản địa hoá thì cần phải tạo ra hai loại tệp: các tệp lớp (`.class`) và các tệp tài sản (`.property`). Tệp tài sản là các tệp văn bản gồm các cặp giá trị từ khoá / giá trị (`key/value`), trong đó giá trị `value` chính là tên được bản địa hoá của từ khoá `key`. Cả hai loại tệp này phải được đặt vào cùng một thư mục hoặc gộp vào trong một tệp `JAR`.

Khi cần tìm hiểu sâu về vấn đề quốc tế hoá và bản địa hoá phần mềm, bạn có thể sử dụng công cụ `jikit` (<http://java.sun.com/products/jikit>) để quản lý việc tạo lập và duy trì các bọc tài nguyên.

**Ví dụ 8.5.** Chương trình hiển thị các mục trong đơn đặt mua hoa quả (táo, đào, mận) theo ngôn ngữ và các đại lượng số theo khuôn mẫu qui định của từng khu vực. Người sử dụng nhập vào số lượng các loại hoa quả cần mua, và số thẻ tín dụng, mã khách hàng (tùy chọn), rồi nhấn nút “Đặt mua” (“Purchase” nếu là khu vực nói tiếng Anh) thì tổng số hoa quả đặt mua và số tiền phải trả sẽ được hiển thị theo đúng qui định của khu vực đã lựa chọn, như hình 8.5 và 8.6. Mặc định là khu vực nói tiếng Anh như hình 8.5. Khi cần nhập lại, hay đặt đơn hàng mới thì nhấn nút “Nhập lại” (đối với người Việt Nam) hay (“Reset” nếu là khu vực nói tiếng Anh).



Hình 8.5. Đơn đặt hàng bằng tiếng Anh



Hình 8.6. Đơn đặt hàng bằng tiếng Việt

```
// ClientOrder.java

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
import java.util.*;
import java.text.*;

public class ClientOrder extends JFrame implements ActionListener {
    JLabel col1, col2;
    JLabel totalItems, totalCost;
    JLabel cardNum, custID;
    JLabel appleLabel, pearLabel, peachLabel;
    JButton purchase, reset;
    JPanel panel;
    JTextField appleNum, pearNum, peachNum;
    JTextField creditCard, customer;
    JTextArea items, cost;

    static Locale currentLocale;
    static ResourceBundle messages;
```

```
static String language, country;
NumberFormat numFormat;

ClientOrder() {
    setTitle(messages.getString("title"));
    col1 = new JLabel(messages.getString("1col"));
    col2 = new JLabel(messages.getString("2col"));

    appleLabel = new JLabel(" " + messages.getString("apples"));
    appleNum = new JTextField();

    pearLabel = new JLabel(" " + messages.getString("pears"));
    pearNum = new JTextField();

    peachLabel = new JLabel(" " + messages.getString("peaches"));
    peachNum = new JTextField();

    cardNum = new JLabel(" " + messages.getString("card"));
    creditCard = new JTextField();

    customer = new JTextField();
    custID = new JLabel(" " + messages.getString("customer"));

    totalItems = new JLabel(" " + messages.getString("items"));
    totalCost = new JLabel(" " + messages.getString("cost"));

    items = new JTextArea();
    cost = new JTextArea();

    purchase = new JButton(messages.getString("purchase"));
    purchase.addActionListener(this);

    reset = new JButton(messages.getString("reset"));
    reset.addActionListener(this);

    panel = new JPanel();
    panel.setLayout(new GridLayout(0, 2));
```

```
        panel.setBackground(Color.white);
        getContentPane().add(panel);
        panel.add(coll);
        panel.add(col2);

        panel.add(appleLabel);
        panel.add(appleNum);

        panel.add(peachLabel);
        panel.add(peachNum);

        panel.add(pearLabel);
        panel.add(pearNum);

        panel.add(totalItems);
        panel.add(items);

        panel.add(totalCost);
        panel.add(cost);

        panel.add(cardNum);
        panel.add(creditCard);

        panel.add(custID);
        panel.add(customer);

        panel.add(reset);
        panel.add(purchase);
    }

    public void actionPerformed(ActionEvent event){
        Object source = event.getSource();
        if (source == reset){
            creditCard.setText("");
            appleNum.setText("");
            peachNum.setText("");
            pearNum.setText("");
        }
    }
}
```

```
        customer.setText("");
        items.setText("");
        cost.setText("");
    }else if(source == purchase){
        int totalNum;
        double totalCost;
        totalNum = Integer.valueOf(appleNum.getText()).intValue()
            + Integer.valueOf(pearNum.getText()).intValue()
            + Integer.valueOf(peachNum.getText()).intValue();
        totalCost = totalNum * 1.25;
        numFormat = NumberFormat.getInstance(currentLocale);
        String text = numFormat.format(totalNum);
        items.setText(text);
        text = numFormat.format(totalCost);
        cost.setText(text);
    }
}

public static void main (String args[]){
    if (args.length != 2){ //Mặc định
        language = new String("en");
        country = new String("US");
        System.out.println("English");
    }
    else{//Đặt lại theo khu vực được đưa vào dưới dạng tham số của chương trình
        language = new String(args[0]);
        country = new String(args[1]);
        System.out.println(language + " " + country);
    }

    currentLocale = new Locale (language, country);
    //Đọc tệp bọc tài nguyên theo khu vực (tên quốc gia và ngôn ngữ)
    messages = ResourceBundle.getBundle("MessagesBundle", currentLocale);
```

```

WindowListener l = new WindowAdapter(){
    public void windowClosing(WindowEvent e){
        System.exit(0);
    }
};

ClientOrder frame = new ClientOrder();
frame.addWindowListener(l);
frame.setSize(300, 300);
frame.setVisible(true);
}
}

```

Như trên đã phân tích, để chương trình trên hiển thị đơn hàng theo khu vực thì ta phải tạo ra các tệp tài sản tương ứng. Ví dụ, muốn hiển thị đơn đặt mua hàng bằng tiếng Việt thì phải sử dụng một hệ soạn thảo nào đó để tạo ra tệp dạng văn bản (text) là `MessagesBundle_vn_VN.properties` (xem hình 8.6) có nội dung như sau.

```

apples = Táo:
peaches = Đào
pears = Mận:
items = Số hàng mua:
cost = Số tiền trả:
card = Credit Card:
customer = Mã hiệu KH:
title = Hoa Quà 2500đ/quà
1col = Chọn hàng
2col = Số lượng
reset = Nhập mới
view = Xem
purchase = Đặt mua

```

Hoặc, để hiển thị đơn đặt mua hàng bằng tiếng Pháp thì phải sử dụng một hệ soạn thảo nào đó để tạo ra tệp dạng văn bản (text) là:

MessagesBundle\_fr\_FR.properties có nội dung như sau.

```

apples=Pommes:
peaches=Peches:
pears=Poires:
items=Partial total:
cost=Cost total:
card=Carte de Credit:
customer=Numero de Client:
title=Fruit 1,25 pièce
1col=Choisissez les éléments
2col=Indiquez la quantité
reset=Réinitialisez
view=Visualisez
purchase=Achetez

```

Tương tự, ta có thể tạo ra các tệp tài sản cho tất cả các khu vực mà ta muốn.

Sau khi dịch thành công tệp ClientOrder.java, ta thực hiện chương trình này với tham số mặc định sẽ được như hình 8.5.

>java ClientOrder

Để chương trình hiển thị đơn đặt hàng bằng tiếng Việt Nam như hình 8.6, ta thực hiện chương trình với tệp tài nguyên như sau:

>java ClientOrder vn VN

**java.util.ResourceBundle**

**API**

- static ResourceBundle getBundle(String baseName, Locale l)
- static ResourceBundle getBundle(String baseName)

Nạp lớp bọc tài nguyên có tên baseName và dịch sang khu vực l hoặc khu vực mặc định.

- Object getObject(String name)

Tìm đối tượng từ bọc tài nguyên hoặc từ các lớp cha của nó.

- String getString(String name)

Tìm đối tượng từ bọc tài nguyên hoặc từ các lớp cha của nó và ép kiểu về String

- `String[] getStringArray(String name)`

Tìm đối tượng từ bọc tài nguyên hoặc từ các lớp cha của nó và ép kiểu về mảng của các xâu.

- `Enumeration getKeys()`

Lấy ra dãy các từ khoá theo thứ tự được liệt kê.

## 8.7. BẢN ĐỊA HÓA CÁC GIAO DIỆN ĐỒ HỌA

Ở trên chúng ta đã đề cập đến những vấn đề chính của công việc bản địa hóa phần mềm. Phần này, chúng ta giải thích cách địa phương hóa các yêu cầu để thay đổi các mã lệnh mà bạn đã viết. Chúng ta xét phương thức lập trình sau:

```
public class MyApplet extends JApplet implements ActionListener{
    public void init(){
        JButton cancelButton = new JButton("Cancel");
        cancelButton.addActionListener(this);
        ...
    }

    public void actionPerformed(ActionEvent evt){
        String arg = evt.getActioncommand();
        if(arg.equals("Cancel")){
            doCancel();
        }
        else ...
    }
}
```

Để nhận thấy, tên gọi của nút nhấn “Cancel” sử dụng tốt cho những người biết tiếng Anh, nhưng khi chương trình sử dụng cho người Việt thì phải dịch thành “Huỷ bỏ”. Do vậy, các tên gọi, thông báo phải được cập nhật tự động cả ở phương thức `init()` và `actionPerformed()`. Nhưng nếu vì một lý do nào đó mà ta quên không cập nhật một trong các hàm đó thì những nút này sẽ không thực hiện được như mong muốn. Để khắc phục vấn đề nêu trên, ta có thể thực hiện theo các chiến lược:

1. Sử dụng các lớp bên trong thay vì sử dụng các phương thức riêng `actionPerformed()`.

2. Xác định các thành phần thông qua các tham chiếu của chúng chứ không phải bằng các tên của các nhãn.
3. Sử dụng thuộc tính name của lớp Component để xác định các thành phần.

Chúng ta lần lượt xét các cách nêu trên.

## 1. Sử dụng lớp bên trong

Thay vì tạo ra một bộ xử lý để xử lý cho nhiều hành động, ta có thể định nghĩa một bộ xử lý riêng cho mỗi thành phần. Ví dụ, đoạn chương trình trên được viết thành:

```
public class MyApplet extends JApplet implements ActionListener{
    public void init(){
        JButton cancelButton = new JButton("Cancel");
        cancelButton.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent evt){
                    doCancel();
                }
            });
        ...
    }
}
```

Chương trình tạo ra một lớp bên trong để lắng nghe sự kiện khi nút nhấn Cancel. Do vậy, chỉ có một chỗ xuất hiện tên của thành phần cần được dịch sang tiếng bản địa.

## 2. Xác định các thành phần thông qua các tham chiếu

Nhiều khi ta không muốn sử dụng những lớp bên trong một lớp khác, bởi vì như thế có thể dẫn đến sự nhập nhằng trong một số thao tác, như đọc dữ liệu chặng hạn. Chúng ta có cách thực hiện khác là đặt các thành phần vào các biến thể hiện và so sánh chúng với câu lệnh tương ứng để xử lý các sự kiện.

```
public class MyApplet extends JApplet implements ActionListener{
    JButton cancelButton;
    public void init(){
        cancelButton = new JButton("Cancel");
        cancelButton.addActionListener(this);
        ...
    }
}
```

```

public void actionPerformed(ActionEvent evt) {
    Object source = evt.getSource();
    if(source == cancelButton)
        doCancel();
    else ...
}

```

### 3. Sử dụng thuộc tính name của lớp Component để xác định các thành phần

Các thành phần mà chúng ta xây dựng đều là lớp con của Component, vì thế có thể sử dụng thuộc tính name thông qua hàm getName() để xác định thành phần tương ứng. Đặc trưng tên gọi của thành phần là bất biến đối với mọi sự thay đổi tên (nhãn) của chúng khi cần bản địa hóa chương trình. Ví dụ, nếu ta gắn cho nút “Huỷ bỏ” với tên gọi là “Cancel1” thông qua name bằng cách sử dụng setName (“Cancel1”) trong chương trình thì nó không phải là nhãn được hiển thị của nút đó, mà là xâu tương ứng.

```

public class MyApplet extends JApplet implements ActionListener{
    public void init(){
        JButton cancelButton = new JButton("Cancel");
        cancelButton.setName("Cancel1");
        cancelButton.addActionListener(this);
        ...
    }
    public void actionPerformed(ActionEvent evt){
        Component source = evt.getSource();
        if(source.getName().equals("Cancel1"))
            doCancel();
        else ...
    }
}

```

Như vậy, nếu nhãn của nút huỷ bỏ được hiển thị là “Cancel” (trong tiếng Anh), và khi cần được dịch sang một ngôn ngữ khác, ví dụ được dịch sang tiếng Việt Nam là “Huỷ bỏ” thì tên đặt trong chương trình của nút đó vẫn không bị thay đổi, nó vẫn là “Cancel1”.

**Ví dụ 8.6.** Chương trình applet tính toán và hiển thị biểu đồ số tiền tiết kiệm và khả năng thu nhập khi nghỉ hưu (nghỉ làm việc) ở nhiều nước khác nhau như hình 8.7, 8.8.

Người sử dụng có thể chọn khu vực (English, China, Germany) và nhập vào:

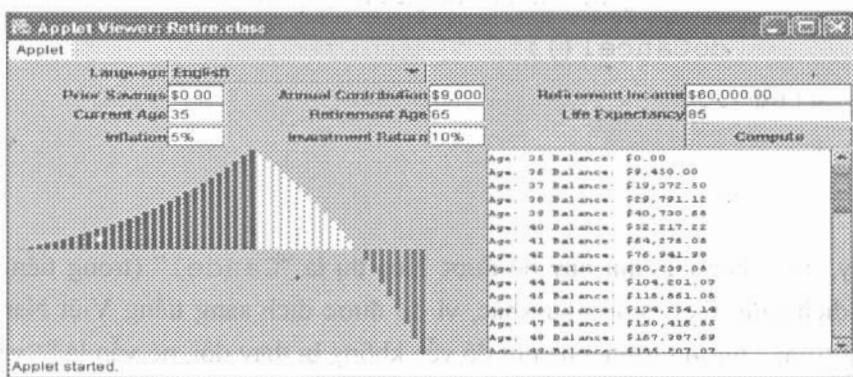
- Số tiền tiết kiệm được từ trước (Prior Savings),
- Số tiền tiết kiệm hàng năm (Annual Contribution),
- Thu nhập khi nghỉ việc (Retirement Income),
- Tuổi hiện nay (Current Age),
- Tuổi nghỉ hưu (Retirement Age),
- Dự kiến tuổi thọ (Life Expectancy),
- Tỷ lệ lạm phát (Inflation),
- Tỷ lệ tăng trưởng (Investment Return).

Sau khi nhập các số liệu nêu trên, nhấn nút “Compute” để tính và vẽ lại biểu đồ về thu nhập theo các độ tuổi về hưu.

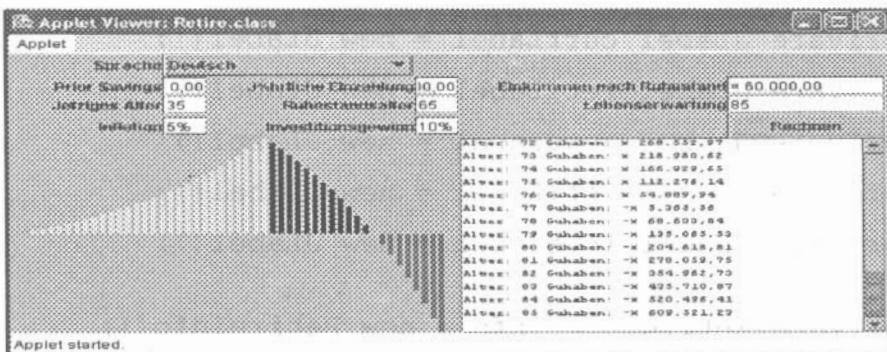
Vùng văn bản hiển thị số tiền dư hàng năm từ năm hiện thời cho đến tuổi thọ dự kiến và tương ứng là biểu đồ mô tả ở vùng bên trái.

Chương trình tính toán tiền tiết kiệm và khả năng thu nhập khi nghỉ làm việc chạy thử cho ba nước: Anh, Đức và Việt Nam. Sau đây là một số điểm cần chú ý trong chương trình.

- Các nhãn, tên nút và các thông báo được dịch sang tiếng Đức, tiếng Việt tương ứng có thể tìm thấy trong các lớp: RetireResources, RetireResources\_de, RetireResources\_zh.
- Các trường văn bản xử lý các dữ liệu số, tiền tệ và tỷ lệ % theo khuôn dạng của từng vùng bản địa.
- Trường tính toán sử dụng MessageFormat.
- Ta có thể chọn màu khác nhau để hiển thị biểu đồ cho từng nước theo yêu cầu của người sử dụng.



Hình 8.7. Tính toán tiền được lĩnh khi nghỉ việc (về hưu) ở Anh (Mỹ)



Hình 8.8. Tính toán tiền được lĩnh khi nghỉ việc (về hưu) ở Đức

```
// Retire.java

import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.text.*;
import java.util.*;
import java.applet.*;

public class Retire extends JApplet{
    private JTextField savingsField = new JTextField(10);
    private JTextField contribField = new JTextField(10);
    private JTextField incomeField = new JTextField(10);
    private JTextField currentAgeField = new JTextField(4);
    private JTextField retireAgeField = new JTextField(4);
    private JTextField deathAgeField = new JTextField(4);
    private JTextField inflationPercentField = new JTextField(6);
    private JTextField investPercentField = new JTextField(6);
    private JTextArea retireText = new JTextArea(10, 25);
    private RetireCanvas retireCanvas = new RetireCanvas();
    private JButton computeButton = new JButton();
    private JLabel langLabel = new JLabel();
    private JLabel saviLabel = new JLabel();
    private JLabel contLabel = new JLabel();
    private JLabel incoLabel = new JLabel();
```

```
private JLabel currLabel = new JLabel();
private JLabel retiLabel = new JLabel();
private JLabel deadLabel = new JLabel();
private JLabel inflLabel = new JLabel();
private JLabel inveLabel = new JLabel();

private RetireInfo info = new RetireInfo();

private Locale[] locales;
private Locale currentLocale;
private JComboBox comboBox = new JComboBox();
private ResourceBundle res;
private NumberFormat currencyFmt;
private NumberFormat numberFmt;
private NumberFormat percentFmt;

public void init(){
    GridBagLayout gbl = new GridBagLayout();
    getContentPane().setLayout(gbl);

    GridBagConstraints gbc = new GridBagConstraints();
    gbc.weightx = 100;
    gbc.weighty = 0;

    gbc.fill= GridBagConstraints.NONE;
    gbc.anchor = GridBagConstraints.EAST;
    add(langLabel, gbc, 0, 0, 1, 1);
    add(saviLabel, gbc, 0, 1, 1, 1);
    add(contLabel, gbc, 2, 1, 1, 1);
    add(incoLabel, gbc, 4, 1, 1, 1);
    add(currLabel, gbc, 0, 2, 1, 1);
    add(retiLabel, gbc, 2, 2, 1, 1);
    add(deadLabel, gbc, 4, 2, 1, 1);
    add(inflLabel, gbc, 0, 3, 1, 1);
    add(inveLabel, gbc, 2, 3, 1, 1);
```

```
gbc.fill= GridBagConstraints.HORIZONTAL;
gbc.anchor = GridBagConstraints.WEST;
add(comboBox, gbc, 1, 0, 2, 1);
add(savingsField, gbc, 1, 1, 1, 1);
add(contribField, gbc, 3, 1, 1, 1);
add(incomeField, gbc, 5, 1, 1, 1);
add(currentAgeField, gbc, 1, 2, 1, 1);
add(retireAgeField, gbc, 3, 2, 1, 1);
add(deathAgeField, gbc, 5, 2, 1, 1);
add(inflationPercentField, gbc, 1, 3, 1, 1);
add(investPercentField, gbc, 3, 3, 1, 1);

computeButton.setName("computeButton");
computeButton.addActionListener(
    new ActionListener(){
        public void actionPerformed(ActionEvent evt){
            getInfo();
            updateData();
            updateGraph();
        }
    });
add(computeButton, gbc, 5, 3, 1, 1);

gbc.weighty = 100;
gbc.fill= GridBagConstraints.BOTH;
add(retireCanvas, gbc, 0, 4, 4, 1);
add(new JScrollPane(retireText), gbc, 4, 4, 2, 1);
retireText.setEditable(false);
retireText.setFont(new Font("Monospaced", Font.PLAIN, 10));

info.setSavings(0);
info.setContrib(9000);
info.setIncome(60000);
info.setCurrentAge(35);
```

```
info.setRetireAge(65);
info.setDeathAge(85);
info.setInvestPercent(0.1);
info.setInflationPercent(0.05);

comboBox.addActionListener(
    new ActionListener(){
        public void actionPerformed(ActionEvent evt){
            setCurrentLocale(comboBox.getSelectedIndex());
        }
    });
}

locales = new Locale[]
{Locale.US, Locale.CHINA, Locale.GERMAN};

int localeIndex = 0; // Mặc định là US
for(int i = 0; i < locales.length; i++)
    if(getLocale().equals(locales[i])) localeIndex = i;

setCurrentLocale(localeIndex);
}

public void add(Component c, GridBagConstraints gbc,
    int x, int y, int w, int h){
    gbc.gridx = x;
    gbc.gridy = y;
    gbc.gridheight = h;
    gbc.gridwidth = w;
    getContentPane().add(c, gbc);
}

public void setCurrentLocale(int localeIndex){
    currentLocale = locales[localeIndex];

    comboBox.removeAllItems();
    String language;
```

```
for(int i = 0; i < locales.length; i++){
    language = locales[i].getDisplayName(currentLocale);
    comboBox.addItem(language);
}

comboBox.setSelectedIndex(localeIndex);
res = ResourceBundle.getBundle("RetireResources", currentLocale);

currencyFmt = NumberFormat.getCurrencyInstance(currentLocale);
numberFmt = NumberFormat.getNumberInstance(currentLocale);
percentFmt = NumberFormat.getPercentInstance(currentLocale);

updateDisplay();
updateInfo();
updateData();
updateGraph();
}

public void updateDisplay(){
    langLabel.setText(res.getString("language"));
    saviLabel.setText(res.getString("savings"));
    contLabel.setText(res.getString("contrib"));
    incoLabel.setText(res.getString("income"));
    currLabel.setText(res.getString("currentAge"));
    retiLabel.setText(res.getString("retireAge"));
    deadLabel.setText(res.getString("deathAge"));
    inflLabel.setText(res.getString("inflationPercent"));
    inveLabel.setText(res.getString("investPercent"));
    computeButton.setText(res.getString("computeButton"));

    validate();
}

public void updateInfo(){
    savingsField.setText(currencyFmt.format(info.getSavings()));
    contribField.setText(currencyFmt.format(info.getContrib()));
}
```

```
incomeField.setText(currencyFmt.format(info.getIncome()));
currentAgeField.setText(numberFmt.format(info.getCurrentAge()));
retireAgeField.setText(numberFmt.format(info.getRetireAge()));
deathAgeField.setText(numberFmt.format(info.getDeathAge()));
inflationPercentField.setText(percentFmt.format(info.getInflationPercent()));
investPercentField.setText(percentFmt.format(info.getInvestPercent()));

}

public void updateData(){
    retireText.setText("");
    MessageFormat retireMsg = new MessageFormat("");
    retireMsg.setLocale(currentLocale);
    retireMsg.applyPattern(res.getString("retire"));

    for(int i = info.getCurrentAge(); i <= info.getDeathAge(); i++){
        Object[] args = {new Integer(i), new Double(info.getBalance(i))};
        retireText.append(retireMsg.format(args) + "\n");
    }
}

public void updateGraph(){
    retireCanvas.setColorGain((Color)res.getObject("colorGain"));
    retireCanvas.setColorPre((Color)res.getObject("colorPre"));
    retireCanvas.setColorLoss((Color)res.getObject("colorLoss"));

    retireCanvas.setInfo(info);
    repaint();
}

public void getInfo(){
    try{
        info.setSavings(currencyFmt.parse(
            savingsField.getText()).doubleValue());
        info.setContrib(currencyFmt.parse(
            contribField.getText()).doubleValue());
    }
}
```

```
        info.setIncome(currencyFmt.parse(
            incomeField.getText()).doubleValue());
        info.setCurrentAge(numberFmt.parse(
            currentAgeField.getText()).intValue());
        info.setRetireAge(numberFmt.parse(
            retireAgeField.getText()).intValue());
        info.setDeathAge(numberFmt.parse(
            deathAgeField.getText()).intValue());
        info.setInvestPercent(percentFmt.parse(
            investPercentField.getText()).doubleValue());
        info.setInflationPercent(percentFmt.parse(
            inflationPercentField.getText()).doubleValue());
    }catch(ParseException e){
    }
}

}

class RetireInfo{
    public double getBalance(int year){
        if(year < currentAge) return 0;
        else if(year == currentAge){
            age = year;
            balance = savings;
            return balance;
        }else if(year == age)
            return balance;
        else if(year != age + 1)
            getBalance(year - 1);
        age = year;
        if(age < retireAge)
            balance += contrib;
        else
            balance -= income;
        balance = balance * (1 + (investPercent - inflationPercent));
    }
}
```

```
        return balance;
    }

    public double getSavings(){
        return savings;
    }

    public double getContrib(){
        return contrib;
    }

    public int getCurrentAge(){
        return currentAge;
    }

    public int getRetireAge(){
        return retireAge;
    }

    public double getIncome(){
        return income;
    }

    public int getDeathAge(){
        return deathAge;
    }

    public double getInflationPercent(){
        return inflationPercent;
    }

    public double getInvestPercent(){
        return investPercent;
    }

    public void setSavings(double x){
        savings = x;
    }

    public void setContrib(double x){
        contrib = x;
    }

    public void setIncome(double x){
```

```
        income = x;
    }

    public void setCurrentAge(int x){
        currentAge = x;
    }

    public void setRetireAge(int x){
        retireAge = x;
    }

    public void setDeathAge(int x){
        deathAge = x;
    }

    public void setInflationPercent(double x){
        inflationPercent = x;
    }

    public void setInvestPercent(double x){
        investPercent = x;
    }

    private double savings;
    private double contrib;
    private double income;
    private double inflationPercent;
    private double investPercent;
    private int currentAge;
    private int retireAge;
    private int deathAge;

    private int age;
    private double balance;
}

class RetireCanvas extends JPanel{
    public RetireCanvas(){
        setSize(300, 300);
    }
}
```

```
public void setInfo(RetireInfo newInfo){  
    info = newInfo;  
}  
  
public void paint(Graphics g){  
    if(info == null) return;  
    double minValue = 0;  
    double maxValue = 0;  
    int i;  
    for(i = info.getCurrentAge(); i <= info.getDeathAge(); i++){  
        double v = info.getBalance(i);  
        if(minValue > v) minValue = v;  
        if(maxValue < v) maxValue = v;  
    }  
    if(maxValue == minValue) return;  
  
    int barW = getWidth() / (info.getDeathAge() - info.getCurrentAge() + 1);  
    double scale = getHeight() / (maxValue - minValue);  
    for(i = info.getCurrentAge(); i <= info.getDeathAge(); i++){  
        int x1 = (i - info.getCurrentAge()) * barW + 1;  
        int y1;  
        double v = info.getBalance(i);  
        int h;  
        int y = (int) (maxValue * scale);  
  
        if(v >= 0){  
            y1 = (int) ((maxValue - v) * scale);  
            h = y - y1;  
        } else{  
            y1 = y;  
            h = (int) (-v * scale);  
        }  
        if(i < info.getRetireAge())  
            g.setColor(colorPre);  
        else  
            g.setColor(colorRetire);  
        g.fillRect(x1, y1, barW, h);  
    }  
}
```

```
        else if( v >= 0)
            g.setColor(colorGain);
        else
            g.setColor(colorLoss);
        g.fillRect(x1, y1, barW - 2, h);
        g.fillRect(x1, y1, barW - 2, h);
    }
}

public void setColorPre(Color c){
    colorPre = c;
}
public void setColorGain(Color c){
    colorGain = c;
}
public void setColorLoss(Color c){
    colorLoss = c;
}

private RetireInfo info = null;
private Color colorPre;
private Color colorGain;
private Color colorLoss;
}

// RetireResources.java: Lớp hiển thị tiếng Anh
import java.util.*;
import java.awt.*;
public class RetireResources extends ListResourceBundle{
    public Object[][] getContents(){
        return contents;
    }
    static final Object[][] contents ={{
        {"language", "Language"}, {
        {"computeButton", "Compute"},
```

```
        {"savings", "Prior Savings"},  
        {"contrib", "Annual Contribution"},  
        {"income", "Retirement Income"},  
        {"currentAge", "Current Age"},  
        {"retireAge", "Retirement Age"},  
        {"deathAge", "Life Expectancy"},  
        {"inflationPercent", "Inflation"},  
        {"investPercent", "Investment Return"},  
        {"retire", "Age: {0, number} Balance: {1, number, currency}" },  
        {"colorPre", Color.blue},  
        {"colorGain", Color.white},  
        {"colorLoss", Color.red}  
    };  
}  
  
// RetireResources_de.java: Lớp hiển thị tiếng Đức  
  
import java.util.*;  
import java.awt.*;  
  
public class RetireResources_de extends ListResourceBundle{  
    public Object[][] getContents(){  
        return contents;  
    }  
  
    static final Object[][] contents ={  
        {"language", "Sprache"},  
        {"computeButton", "Rechnen"},  
        {"savings", "Prior Savings"},  
        {"contrib", "Jährliche Einzahlung"},  
        {"income", "Einkommen nach Ruhestand"},  
        {"currentAge", "Jetziges Alter"},  
        {"retireAge", "Ruhestandsalter"},  
        {"deathAge", "Lebenserwartung"},  
        {"inflationPercent", "Inflation"},  
    };
```

```
        {"investPercent", "Investitionsgewinn"},  
        {"retire", "Alter: {0, number} Guhaben: {1, number, currency}"},  
        {"colorPre", Color.yellow},  
        {"colorGain", Color.black},  
        {"colorLoss", Color.red}  
    };  
}  
  
// RetireResources_zh.java: Lớp hiển thị tiếng Việt  
import java.util.*;  
import java.awt.*;  
public class RetireResources_zh extends ListResourceBundle{  
    public Object[][] getContents(){  
        return contents;  
    }  
    static final Object[][] contents ={  
        {"language", "Ngon ngu"},  
        {"computeButton", "Tinh"},  
        {"savings", "Tich luy tu truoc"},  
        {"contrib", "Thu nhap hang nam"},  
        {"income", "Thu nhap khi nghi huu"},  
        {"currentAge", "Tuoi hien nay"},  
        {"retireAge", "Tuoi nghi huu"},  
        {"deathAge", "Du kien tuoi tho"},  
        {"inflationPercent", "Ty le lam phat"},  
        {"investPercent", "Ty le tang truong"},  
        {"retire", "Tuoi: {0, number} So du: {1, number, currency}"},  
        {"colorPre", Color.yellow},  
        {"colorGain", Color.black},  
        {"colorLoss", Color.red}  
    };  
}
```

## BÀI TẬP

- 8.1.** Viết chương trình đặt mua và bán hàng qua mạng sử dụng RMI. Đơn đặt hàng được hiển thị theo khu vực của khách hàng (ví dụ bằng tiếng Việt) tương tự như ở ví dụ 8.5. Khi người mua nhấn nút “Đặt mua” thì đơn hàng này được gửi cho người bán và được lại được hiển thị bằng ngôn ngữ của người bán, ví dụ bằng tiếng Anh.
- 8.2.** Viết chương trình đặt mua và bán hàng qua mạng sử dụng Servlet. Đơn đặt hàng được hiển thị theo khu vực của khách hàng (ví dụ bằng tiếng Việt) tương tự như ở ví dụ 8.5 . Khi người mua nhấn nút “Đặt mua” thì đơn hàng này được gửi cho người bán và được lại được hiển thị bằng ngôn ngữ của người bán, ví dụ bằng tiếng Anh.

## DANH SÁCH CÁC THUẬT NGỮ ANH - VIỆT VÀ TỪ VIẾT TẮT

Account	Tài khoản
Application Programming Interface	Giao diện lập trình ứng dụng
Authenticated	Được xác thực, được chứng thực
Bean	Hạt nhân, thành phần hạt nhân
Bean Context	Ngữ cảnh của Bean
Blocked	Bị chặn
Bound property	Thuộc tính biên
Browser	Trình duyệt
Certificate	Chứng chỉ, giấy chứng nhận
Certificate Authority	Bộ phận, cơ quan chứng thực
Collection	Tuyển tập
Common Object Broker Request Architecture	Kiến trúc môi giới yêu cầu đối tượng chung: CORBA
Constrained property	Thuộc tính bị không chế
Constructor	Toán tử tạo lập, tạo dựng, cấu tử
Datagram	Đồ hình dữ liệu
Deadlock	Chết tắc, tắc nghẽn
Destructor	Huỷ tử, toán tử huỷ bỏ
Digital Signature	Chữ ký số
Digital Certificate	Chứng chỉ số, chứng nhận số
Directory	Thư mục
Distributed	Phân tán
Encapsulation	Bao gói, gói gọn
Engine	Mô tả, động cơ

Extend	Kế thừa, mở rộng
File	Tệp
Filter Model	Mô hình bộ lọc
Firewall	Bức tường lửa
Hierarchical file system	Hệ thống tệp phân cấp
Holder class	Lớp cất giữ
Holder object	Đối tượng cất giữ
Implement	Cài đặt
Inherit	Kế thừa
Intranet	Mạng nội hạt, nội bộ
Interface Definition Language	Ngôn ngữ định nghĩa giao diện: IDL
Internationalization	Quốc tế hóa
Introspection Report	Báo cáo thanh tra
Interrupted	Bị ngắt
Java Runtime Environment	Môi trường thực thi Java
Java Virtual Machine	Máy Java ảo
Mail Standard Format	Định dạng chuẩn thư điện tử
Message	Thông điệp, tin nhắn, thông báo
Message Digest	Bản tóm tắt thông điệp, dấu vết thông điệp
Method	Phương thức, hàm
Multitasking	Đa nhiệm
MultiThreading	Xử lý đa luồng
Localization	Bản địa hóa, địa phương hóa
Lool & feel	Thiết kế và cảm nhận được
Object lock	Khoá đối tượng
Object Request Broker	Bộ môi giới yêu cầu
Object Oriented	Hướng đối tượng
Option	Tuỳ chọn, tuỳ chỉnh
Overflow	Tràn bộ nhớ

Reference	Tham chiếu
Register	Thanh ghi
Remove interface	Giao diện từ xa
Remove Method Invocation	Triệu gọi phương thức từ xa: RMI
Remove object	Đối tượng từ xa
Remove Reference	Tham chiếu từ xa
Resource Bundle	Bộ tài nguyên
Package	Gói
Private key	Khoá riêng
Process	Tiến trình
Progress Meter	Thước đo tiến độ
Protocol	Giao thức
Public key cryptography	Mật mã khoá công khai
Safety	An toàn (sự an toàn)
Security	Bảo mật, an ninh
Security policy	Chính sách bảo mật, an ninh
Server's Internet Host	Máy chủ (Server) trên Internet
Slider	Thanh trượt, thước đo
Socket	Cơ chế ô cắm, bảng cắm
Standard Protocol	Giao thức chuẩn
Stub	Đại diện trên máy khách
Synchronization	Đồng bộ hoá
Thread	Luồng
Time-Sharing	Chia sẻ thời gian
Three-tier model	Mô hình ba tầng
Transport Layer	Tầng giao vận
Tree	Cây, cấu trúc cây
Verifier	Bộ kiểm tra

API	Application Programming Interface
ASP	Active Server Page
BDK	Bean Development Kit
B2B	Business-to-Business
CA	Certificate Authority
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
DSA	Digital Signature Algorithm
EJB	Enterprise Java Bean
FTP	File Transfer Protocol
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDL	Interface Definition Language
IE	Internet Explore
IMAP	Internet Message Access Protocol
IP	Internet Protocol
ITU-T	International Telecommunication Union- Telecommunication Standardization Sector
JDBC	Java DataBase Connectivity
JDK	Java Development Kit
JNDI	Java Naming Directory Interface
JSP	Java Server Page
JVM	Java Virtual Machine
MIME	Multipurpose Internet Mail Extension
ODBC	Open Data Base Connectivity
COM	Common Object Model
OMG	Object Management Group
ORB	Object Request Broker
POP3	Post Office Protocol
RFC	Requests For Comments
RMI	Remote Method Invocation
ROA	Remote Object Activation

RSA	Rivest – Shamir - Adleman
TCP	Transmission Control Protocol
SDK	Servlet Development Kit
SHA1	Security Hash Algorithm # 1
SMTP	Simple Mail Server Transport Protocol
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UDP	User Datagram Protocol
URL	Uniform Resource Locator
XML	Extensible Markup Language
WYSIWYG	What you see is what you get

## TÀI LIỆU THAM KHẢO

---

- [1] Đoàn Văn Ban, *Lập trình hướng đối tượng với Java*, Nhà xuất bản Khoa học và Kỹ thuật, Hà Nội, 2005 (tái bản).
- [2] *Core Java, Volume I, II- Advanced Features*, Cay S. Horstmann, Gary Cornell, Sun Microsystems Press, 2000.
- [3] Daniel Y. Liang, *Introduction to Java Programming*, Prentice Hall, 2000.
- [4] Hiroshi Maruyama, et all., *XML and Java<sup>TM</sup> Developing Web Application*, Second Eddition, Addison-Wesley, 2002.
- [5] Joseph JáJá, *An Introduction to Parallel Algorithms*, Addison-wesley, 1992.
- [6] Khalid A.Mughal & Rolf W.Rasmussen, *A Programmer's Guide to Java <sup>TM</sup> Certification*, Addison-Wesley, 2000.

<http://java.sun.com/j2se/1.4/docs/guide/rmi>

<http://www.securingjava.com>

<http://archives.java.sun.com>

<http://www.apache.org>

<http://www.oreilly.com/catalog/javasec2/>

# CHỈ MỤC

- API, 52, 110, 174, 188  
ASP, 188  
Bản địa hoá, 312, 321  
bảo mật thông tin, 5, 188, 233, 235  
Bean, 138, 165, 166, 168, 178, 180, 224, 226  
BeanBox, 167, 168, 170, 178  
bị chặn, 15, 16, 24  
Bộ đăng ký RMI, 52  
bộ kiểm tra, 240, 241  
bộ nạp lớp, 236, 237, 240  
bọc tài nguyên, 313, 321  
CA, 272  
Các mức ưu tiên của, 20  
cấu trúc bảng, 6, 133  
cấu trúc cây, 117, 121, 133  
CGI, 187, 188, 194, 205  
chia sẻ thời gian, 9, 10  
*chính sách bảo mật*, 233, 234, 245, 246  
chữ ký số, 263, 264  
*chứng thực*, 269, 270, 271  
CLASSPATH, 55, 196  
cookie, 209, 220  
CORBA, 72, 77, 81  
CSDL, 214, 215  
*đa luồng*, 9, 33, 91  
đa nhiệm, 8, 9  
dấu vết thông điệp, 258  
đối tượng pshục vụ, 49, 68  
*đối tượng từ xa*, 46, 48, 50, 59, 60, 63, 67, 78  
đồng bộ hoá, 8, 28  
*đồng thời logic*, 9  
DSA, 264, 265, 267  
E-mail, 87, 106, 107, 110  
giao diện từ xa, 47, 48, 60, 67, 68  
giấy chứng nhận, 270, 272, 274  
*hàm đồng bộ*, 22, 23  
HTML, 186, 195, 201, 215  
HTTP, 57, 82, 100, 111, 189, 193  
IDL, 72, 75, 77  
IP, 44, 56, 84, 234  
JavaBean, 165, 166, 171  
javax.swing, 117, 122, 127, 150  
JDBC, 138, 188, 212, 213  
.ini, 190

- JSP, 216, 217, 220, 223  
JVM, 10, 44, 46, 49, 236, 243  
kế thừa, 10, 13, 17, 25, 33, 43, 47, 126, 170, 193  
khóa công khai, 263, 264, 269  
khóa đối tượng, 29, 30  
khóa riêng, 264, 266  
*lập trình mạng*, 87, 189  
luồng, 8, 10, 27, 33, 36, 93  
MD5, 233, 258  
*mô hình ba tầng*, 212  
Mô hình ba tầng, 212  
monitor, 21, 22, 23, 24  
mức ưu tiên của luồng, 17  
ODBC, 214  
phân tán đối tượng, 42, 55  
*phương thức*, 42, 43, 47, 50, 57, 75  
Policy, 245, 246  
quốc tế hóa, 5, 286, 287, 316  
Remote, 46, 47, 59  
RMI, 43, 48, 49, 52, 54, 69, 72, 213  
RSA, 264, 265  
SecurityManager, 243, 254  
Server, 42, 44, 50, 57, 60, 87  
Servlet, 186, 188, 190, 192, 195, 199, 211, 215, 221  
Session, 209, 210, 219, 226  
SHA-1, 233, 258, 262  
Signature, 266, 268  
Skel, 44, 45, 59  
SOAP, 82  
Socket, 81, 85, 87, 100, 107  
SSL/TLS, 234, 271  
tắc nghẽn, 22, 31, 32  
TCP/IP, 44, 46, 81, 88  
thẻ chỉ dẫn, 222, 223  
thẻ chú thích, 223  
thẻ khai báo, 221  
thẻ kịch bản, 221  
thông điệp, 7, 42, 106, 178, 222, 263, 266, 269  
Thread, 10, 13, 15, 33  
thuộc tính biên, 172  
tiến trình, 8, 10, 187  
toán tử tạo lập, 34, 118  
Tomcat, 191, 192, 197  
Tranh trượt Slider, 154  
Triệu gọi phương thức từ xa, 42  
UDP, 100, 101  
Unicode, 72, 286, 302  
URL, 102, 104, 110, 206, 215, 224  
WebSite, 201, 212  
X509, 271, 273  
xác thực, 267, 270, 271  
XML, 216, 218, 224  
xử lý đa luồng, 5, 31, 189

**206192**



8 935048 961926

Giá 68.000