

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**  
**TRƯỜNG ĐẠI HỌC KỸ THUẬT CÔNG NGHỆ TP HCM**

**GIÁO TRÌNH**  
**LẬP TRÌNH MẠNG**

**Lưu hành nội bộ**

**Ths. VĂN THIÊN HOÀNG**

**Thành phố Hồ Chí Minh 2009**

# MỤC LỤC

MỞ ĐẦU.....	9
<u>Chương 1 . TỔNG QUAN VỀ LẬP TRÌNH MẠNG.....</u>	<u>12</u>
1.1 Chức năng của một chương trình mạng.....	12
1.2 Mạng máy tính.....	12
1.3 Mô hình phân tầng.....	13
1.3.1 Mô hình OSI.....	13
1.3.2 Mô hình TCP/IP.....	15
1.4 Các giao thức mạng.....	16
1.4.1 TCP-Transmission Control Protocol.....	16
1.4.2 UDP-User Datagram Protocol.....	17
1.4.3 IP-Internet Protocol .....	19
1.5 Mô hình khách/chủ (client/server).....	22
1.6 Socket.....	22
1.7 Dịch vụ tên miền.....	22
1.7.1 Các server tên miền.....	23
1.7.2 Nslookup.....	24
1.8 Các vấn đề liên quan Internet.....	25
1.8.1 Intranet và Extranet.....	25
1.8.2 Firewall.....	25
1.8.3 Proxy Server.....	25
1.9 Bài tập.....	26
<u>Chương 2 . NGÔN NGỮ LẬP TRÌNH JAVA.....</u>	<u>27</u>
2.1 Giới thiệu.....	27
2.2 Môi trường lập trình Java.....	27
2.3 Một số ví dụ mở đầu.....	28
2.4 Các thành phần cơ bản của ngôn ngữ lập trình Java.....	30
2.4.1 Định danh.....	30

2.4.2 Các kiểu dữ liệu nguyên thủy (primitive datatype).....	30
2.4.3 Khai báo các biến.....	31
2.4.4 Các câu lệnh cơ bản.....	32
2.5 Lớp đối tượng.....	37
2.5.1 Cú pháp định nghĩa một lớp đối tượng.....	37
2.5.2 Cú pháp định nghĩa phương thức .....	38
2.5.3 Hàm khởi tạo - Constructor.....	38
2.5.4 Tham chiếu this.....	40
2.5.5 Thừa kế .....	40
2.5.6 Từ khóa super.....	41
2.5.7 Truyền tham số trong Java.....	42
2.5.8 Đa hình.....	42
2.5.9 Thành phần tĩnh.....	43
2.5.10 Các thành phần hằng (final).....	45
2.6 Lớp trừu tượng.....	46
2.7 Giao diện (Interface).....	48
2.8 Gói (Package).....	50
2.9 Quản lý ngoại lệ (Exception Handling).....	51
2.9.1 Lệnh kiểm soát ngoại lệ cơ bản.....	51
2.9.2 Tổ chức lớp biểu diễn ngoại lệ.....	53
2.9.3 Phát sinh ngoại lệ.....	54
2.9.4 Sự lan truyền ngoại lệ.....	59
2.9.5 Các phương thức chính của lớp Exception.....	60
2.9.6 Lớp ngoại lệ tự định nghĩa.....	61
2.9.7 Overriding với exception .....	63
2.10 Bài tập.....	64
Chương 3 . QUẢN LÝ CÁC LUỒNG VÀO RA.....	66
3.1 Giới thiệu.....	66
3.2 Các luồng Byte.....	67

3.2.1 Các luồng byte tổng quát.....	68
3.2.2 Các luồng đọc byte hiện thực.....	69
3.2.3 Các ví dụ.....	71
3.3 Các luồng ký tự.....	75
3.3.1 Các luồng ký tự tổng quát.....	76
3.3.2 Các luồng ký tự hiện thực.....	78
3.3.3 Các ví dụ.....	80
3.4 Các luồng lọc dữ liệu (Filter Stream).....	81
3.4.1 Các luồng lọc tổng quát.....	82
3.4.2 Các luồng lọc hiện thực.....	82
3.5 Các luồng đệm dữ liệu.....	84
3.6 Các lớp định dạng luồng dữ liệu nhập/xuất.....	88
3.7 Tuần tự hóa đối tượng (object serialization).....	91
3.8 Luồng viết đối tượng.....	91
3.9 Bài tập .....	94
<b>Chương 4 . LẬP TRÌNH ĐA TUYẾN.....</b>	<b>95</b>
4.1 Giới thiệu.....	95
4.1.1 Đơn tuyến trình.....	95
4.1.2 Đa tiến trình.....	95
4.2 Tiến trình.....	96
4.3 Lớp Thread.....	96
4.4 Quản lý Thread.....	97
4.4.1 Tạo Thread.....	97
4.4.2 Chính độ ưu tiên.....	98
4.4.3 Thực thi thread.....	98
4.4.4 Dừng thread.....	98
4.4.5 Một số phương thức quản lý khác của lớp Thread.....	99
4.5 Interface Runnable.....	99
4.6 Đồng bộ.....	101

4.6.1 Đồng bộ hóa sử dụng cho phương thức.....	101
4.6.2 Lệnh synchronized.....	103
4.7 Trao đổi dữ liệu giữa các thread.....	104
4.8 Nhóm các tuyến trình –ThreadGroup.....	105
4.8.1 Tạo một nhóm tuyến trình.....	106
4.8.2 Sử dụng một nhóm tuyến trình.....	106
4.8.3 Minh họa về lớp ThreadGroup.....	107
4.9 Bài tập.....	109
<b>Chương 5 . QUẢN LÝ ĐỊA CHỈ KẾT NỐI MẠNG VỚI CÁC LỚP</b> <b>INETADDRESS, URL VÀ URLCONNECTION.....</b>	<b>110</b>
5.1 Lớp InetAddress.....	110
5.1.1 Tạo các đối tượng InetAddress.....	111
5.1.2 Các thao tác đối dữ liệu của một đối tượng InetAddress.....	112
5.1.3 Các phương thức kiểm tra địa chỉ IP .....	113
5.2 Lớp URL.....	114
5.2.1 Tạo các URL.....	115
5.2.2 Nhận thông tin các thành phần của URL.....	116
5.2.3 Nhận dữ liệu từ máy đích trong URL.....	119
5.3 Lớp URLConnection.....	121
<b>Chương 6 . LẬP TRÌNH SOCKET CHO THỨC TCP.....</b>	<b>125</b>
6.1 Mô hình client/server.....	125
6.2 Mô hình truyền tin socket.....	126
6.3 TCP client - Socket .....	128
6.3.1 Các hàm khởi tạo của lớp Socket.....	128
6.3.2 Các phương thức phục vụ giao tiếp giữa các Socket.....	130
6.3.3 Các phương thức đóng Socket.....	131
6.3.4 Các phương thức thiết lập các tùy chọn cho Socket.....	132
6.3.5 Một số ví dụ chương trình client giao tiếp trên một số dịch vụ chuẩn...133	
6.4 Lớp ServerSocket.....	133

6.4.1 Các constructor.....	134
6.4.2 Chấp nhận và ngắt liên kết.....	135
6.5 Các bước cài đặt chương trình phía Client bằng Java.....	138
6.6 Các bước để cài đặt chương trình Server bằng Java.....	140
6.7 Ứng dụng đa tuyến đoạn trong lập trình Java.....	143
6.8 Kết luận.....	147
<b>Chương 7 . LẬP TRÌNH ỨNG DỤNG CHO GIAO THỨC UDP.....</b>	<b>148</b>
7.1 Tổng quan về giao thức UDP.....	148
7.1.1 Một số thuật ngữ UDP.....	149
7.1.2 Hoạt động của giao thức UDP.....	151
7.1.3 Các nhược điểm của giao thức UDP.....	151
7.1.4 Các ưu điểm của UDP.....	152
7.1.5 Khi nào thì nên sử dụng UDP.....	152
7.2 Lớp DatagramPacket.....	153
7.2.1 Các constructor để nhận datagram.....	154
7.2.2 Constructor để gửi các datagram.....	155
7.2.3 Các phương thức nhận các thông tin từ DatagramPacket.....	156
7.3 Lớp DatagramSocket.....	158
7.4 Nhận các gói tin .....	159
7.5 Gửi các gói tin .....	160
7.6 Ví dụ minh họa giao thức UDP.....	161
7.7 Kết luận.....	171
<b>Chương 8 . TUẦN TỰ HÓA ĐỐI TƯỢNG VÀ ỨNG DỤNG TRONG LẬP TRÌNH MẠNG.....</b>	<b>172</b>
8.1 Tuần tự hóa đối tượng.....	172
8.1.1 Khái niệm.....	172
8.1.2 1.2. Khả tuần tự (Serializable).....	173
8.1.3 Xây dựng lớp một lớp khả tuần tự.....	174
8.1.4 Cơ chế đọc và ghi đối tượng trên thiết bị lưu trữ ngoài.....	174

8.2 Truyền các đối tượng thông qua Socket.....	176
8.2.1 Lớp Socket.....	176
8.2.2 Lớp ServerSocket.....	176
8.2.3 Truyền và nhận dữ liệu trong mô hình lập trình Socket.....	177
8.2.4 Ví dụ minh họa.....	178
8.3 Truyền các đối tượng thông qua giao thức UDP.....	182
8.4 Kết luận.....	184
<b>Chương 9 . PHÂN TÁN ĐỐI TƯỢNG TRONG JAVA BẰNG RMI.</b>	<b>185</b>
9.1 Tổng quan.....	185
9.2 Mục đích của RMI.....	186
9.3 Một số thuật ngữ .....	186
9.4 Các lớp trung gian Stub và Skeleton.....	187
9.5 Cơ chế hoạt động của RMI.....	187
9.6 Kiến trúc RMI.....	191
9.7 Cài đặt chương trình.....	192
9.7.1 Cài đặt chương trình phía Server.....	193
9.8 Triển khai ứng dụng.....	195
9.9 Các lớp và các giao tiếp trong gói java.rmi.....	196
9.9.1 Giao tiếp Remote .....	196
9.9.2 Lớp Naming.....	196
9.10 Các lớp và các giao tiếp trong gói java.rmi.registry.....	197
9.10.1 Giao tiếp Registry.....	198
9.10.2 Lớp LocateRegistry.....	198
9.11 Các lớp và các giao tiếp trong gói java.rmi.server.....	199
9.11.1 Lớp RemoteObject.....	199
9.11.2 Lớp RemoteServer.....	199
9.11.3 Lớp UnicastRemoteObject.....	200
9.12 Kết luận.....	200
<b>TÀI LIỆU THAM KHẢO.....</b>	<b>201</b>





## MỞ ĐẦU

Hiện nay, mạng máy tính là công nghệ của của thời đại. Các ứng dụng mạng đóng vai trò không thể thiếu để khai thác tiềm năng của mạng máy tính, đặt biệt là mạng Internet. Do vậy, lập trình mạng là môn học không thể thiếu của sinh viên ngành công nghệ thông tin nói chung và sinh viên chuyên ngành mạng nói riêng. Mục đích của môn học lập trình mạng là cung cấp cho sinh viên biết kiến thức mạng liên quan cũng như cơ chế hoạt động và kiến trúc của các phần mềm mạng. Từ đó, sinh viên hiểu và biết cách viết các chương trình ứng dụng trong một hệ thống mạng quy mô nhỏ cũng như mạng Internet.

Java là ngôn ngữ lập trình hướng đối tượng thuần túy với nhiều đặc trưng ưu việt so với các ngôn ngữ lập trình hướng đối tượng khác như tính độc lập với nền, tính bảo mật,... Java cung cấp bộ thư viện hỗ trợ lập trình mạng đơn giản và rất hiệu quả.

Giáo trình này được thiết kế để cung cấp cho sinh viên những kiến cơ bản để biết phát triển những ứng dụng theo mô hình khách/chủ dựa vào TCP/IP Socket. Đồng thời, một số kiến thức hỗ trợ lập trình phân tán cũng được trang bị. Giáo trình bao gồm 10 chương:

- Chương 1: Giới thiệu vai trò của chương trình mạng, những khái niệm căn bản về mạng máy tính, cũng như kiến thức liên quan để người đọc có thể tiếp cận với các chương tiếp theo.
- Chương 2: Giới thiệu ngôn ngữ lập trình Java. Trình bày các thành phần cơ bản, các thư viện hỗ trợ có sẵn và các cơ chế xử lý ngoại lệ. Sinh viên mới tiếp cận cũng như đã học ngôn ngữ lập trình Java có thể học và ôn tập để có thể hiểu và viết các ứng dụng Java cơ bản.
- Chương 3: Các luồng vào ra. Chương này giới thiệu khái niệm vào ra bằng các luồng dữ liệu. Trước tiên ta sẽ tìm hiểu về các luồng và ý nghĩa của luồng trong chương trình Java. Tiếp đến chúng ta sẽ lần lượt tìm hiểu các luồng vào ra chuẩn trong gói làm việc với console. Các luồng trừu tượng `java.io.InputStream`, `java.io.OutputStream` là các luồng cơ bản để từ đó xây dựng nên các luồng cụ thể. Luồng được chia thành các nhóm như luồng byte và luồng ký tự. Từ phiên bản Java 1.4 một đặc trưng vào ra mới trong Java được đưa vào cũng được giới thiệu

trong chương này. Việc nắm vững kiến thức ở chương này cũng giúp cho việc lập trình ứng dụng mạng trở nên đơn giản hơn vì thực chất của việc truyền và nhận dữ liệu giữa các ứng dụng mạng là việc đọc và ghi các luồng.

- Chương 4: Lập trình đa tuyến đoạn. Trong các ngôn ngữ lập trình trước đây các ứng dụng hầu hết là các ứng dụng đơn tuyến đoạn. Để tăng tốc độ xử lý và giải quyết vấn đề tương tranh của các ứng dụng nói chung và ứng dụng mạng nói riêng ta cần sử dụng khái niệm đa tuyến đoạn. Phần đầu của chương này trình bày các khái niệm căn bản về tiến trình, tuyến đoạn. Tiếp đến chúng ta sẽ xem xét các cách cài đặt một ứng dụng tuyến đoạn trong Java bằng lớp Thread và thực thi giao tiếp Runnable. Sau đó ta sẽ đi vào tìm hiểu các phương thức của lớp Thread. Sự đồng bộ hóa và cách cài đặt một chương trình đồng bộ hóa cũng được giới thiệu trong chương này.
- Chương 5: Lập trình mạng với các lớp InetAddress, URL và URLConnection. Lớp InetAddress là lớp căn bản đầu tiên trong lập trình mạng mà ta cần tìm hiểu. Nó chỉ ra cách một chương trình Java tương tác với hệ thống tên miền. Tiếp đến ta sẽ đi vào tìm hiểu các khái niệm về URI, URL, URN và lớp biểu diễn URL trong Java. Cách sử dụng URL để tải về thông tin và tệp tin từ các server. Sau đó ta đi vào tìm hiểu lớp URLConnection, lớp này đóng vai trò như một động cơ cho lớp URL.
- Chương 6: Lập trình Socket cho giao thức TCP. Trong chương này chúng ta sẽ tìm hiểu cách lập trình cho mô hình client/server và các kiểu kiến trúc client/server. Các lớp Socket và ServerSocket được trình bày chi tiết trong chương này để lập các chương trình cho giao thức TCP.
- Chương 7: Lập trình ứng dụng cho giao thức UDP. Chương này giới thiệu giao thức UDP và các đặc trưng của giao thức này. Tiếp đến ta đi vào tìm hiểu các lớp DatagramPacket và DatagramSocket để viết các chương trình ứng dụng mạng cho giao thức UDP.
- Chương 8: Tuần tự hóa đối tượng và ứng dụng trong lập trình mạng. Trình bày các vấn đề về tuần tự hóa và ứng dụng của tuần tự hóa trong lập trình mạng.

- Chương 9: Phân tán đối tượng bằng Java RMI. Chương này tìm hiểu chủ đề về lập trình phân tán đối tượng bằng kỹ thuật gọi phương thức RMI (Remote Method Invocation).

# CHƯƠNG 1 . TỔNG QUAN VỀ LẬP TRÌNH MẠNG

Chương này trình bày các vấn đề, các khái niệm cơ bản liên quan đến lập trình mạng bằng ngôn ngữ Java bao gồm: chức năng của một chương trình mạng, các định nghĩa về mạng, mô hình phân tầng TCP/IP, mô hình OSI, các giao thức IP, TCP, UDP, mô hình khách /chủ (Client/Server), Socket và một số vấn đề liên quan Internet khác.

## 1.1 Chức năng của một chương trình mạng

Các chương trình mạng cho phép người sử dụng khai thác thông tin được lưu trữ trên các máy tính trong hệ thống mạng (đặt biệt các chương trình chạy trên mạng Internet với hàng triệu máy vi tính được định vị khắp nơi trên thế giới.), trao đổi thông tin giữa các máy tính trong hệ thống mạng, cho phép huy động nhiều máy tính cùng thực hiện để giải quyết một bài toán hoặc giám sát hoạt động mạng.

Ví dụ như các chương trình cho phép truy xuất dữ liệu. Các chương trình này cho phép máy khách (client) truy xuất dữ liệu lưu trên máy chủ (server) và định dạng dữ liệu và trình bày người sử dụng. Các chương trình này sử dụng các giao thức chuẩn như HTTP (gửi dữ liệu Web), FTP (gửi/nhận tập tin), SMTP (gửi mail) hoặc thiết kế những giao thức mới phục vụ cho việc truy xuất dữ liệu. Các chương trình này hoạt động theo mô hình Client/Server. Hoặc chương trình cho phép người dùng tương tác với nhau như Game online, Chat, ...

## 1.2 Mạng máy tính

Mạng máy tính là tập hợp các máy tính hoặc các thiết bị được nối với nhau bởi các đường truyền vật lý và theo một kiến trúc nào đó. Mạng máy tính có thể phân loại theo qui mô như sau:

- Mạng LAN (Local Area Network)-mạng cục bộ: kết nối các nút trên một phạm vi giới hạn. Phạm vi này có thể là một công ty, hay một tòa nhà.
- Mạng WAN (Wide Area Network): nhiều mạng LAN kết nối với nhau tạo thành mạng WAN.

- MAN (Metropolitan Area Network), tương tự như WAN, nó cũng kết nối nhiều mạng LAN. Tuy nhiên, một mạng MAN có phạm vi là một thành phố hay một đô thị nhỏ. MAN sử dụng các mạng tốc độ cao để kết nối các mạng LAN của trường học, chính phủ, công ty, ..., bằng cách sử dụng các liên kết nhanh tới từng điểm như cáp quang.

Trong một hệ thống mạng, mạng xương sống (Backbone) đóng vai trò quan trọng. Mạng Backbone là một mạng tốc độ cao kết nối các mạng có tốc độ thấp hơn. Một công ty sử dụng mạng Backbone để kết nối các mạng LAN có tốc độ thấp hơn. Mạng Backbone Internet được xây dựng bởi các mạng tốc độ cao kết nối các mạng tốc độ cao. Nhà cung cấp Internet hoặc kết nối trực tiếp với mạng backbone Internet, hoặc một nhà cung cấp lớn hơn.

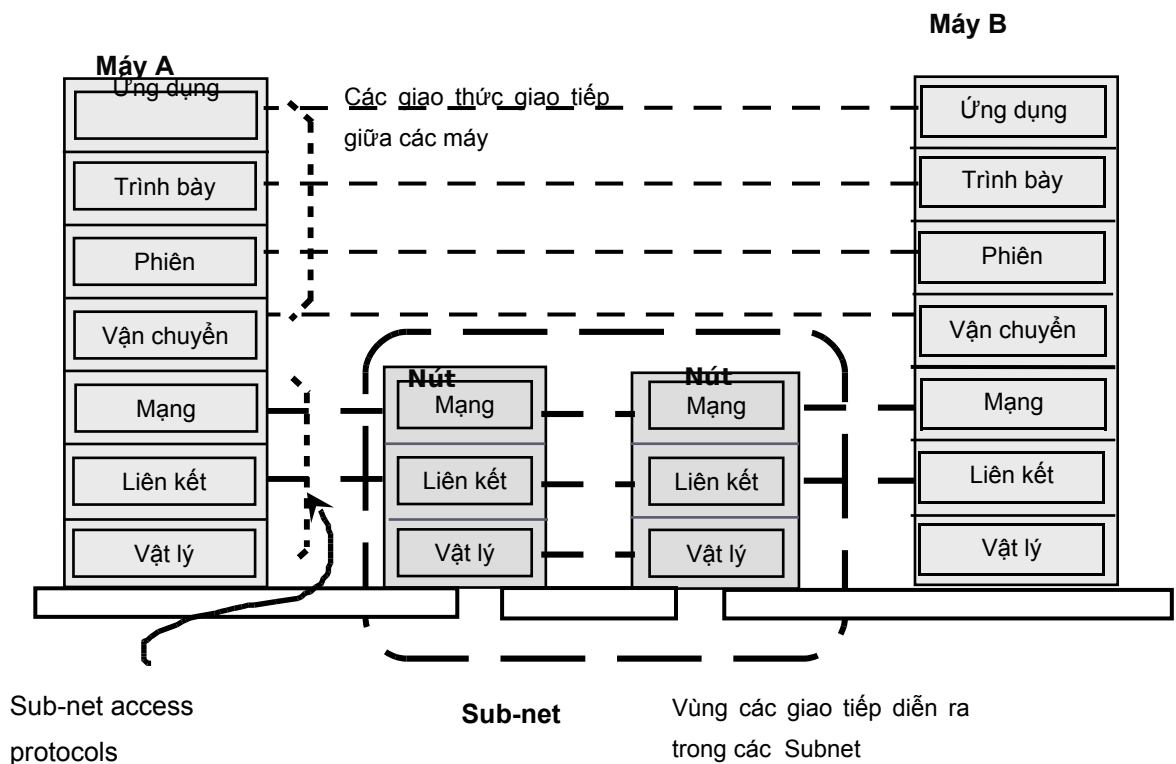
### **1.3 Mô hình phân tầng**

Hoạt động gửi dữ liệu đi trong một mạng từ máy trạm đến máy đích là hết sức phức tạp cả mức vật lý lẫn logic. Các vấn đề đặt ra như vấn đề biến đổi tín hiệu số sang tín hiệu tương tự, tránh xung đột giữa các gói tin (phân biệt các gói tin), phát hiện và kiểm tra lỗi gói tin, định tuyến gói tin đi tới đích, ... Nhằm quản lý được hoạt động này, các hoạt động giao tiếp mạng liên quan được phân tách vào các tầng, hình thành mô hình phân tầng. Có hai dạng mô hình phân tầng phổ biến: OSI, TCP/IP.

#### *1.3.1 Mô hình OSI*

Mô OSI trình bày bảy tầng đã rất thành công và nó hiện đang được sử dụng như là một mô hình tham chiếu để mô tả các giao thức mạng khác nhau và chức năng của chúng. Các tầng của mô hình OSI phân chia các nhiệm vụ cơ bản mà các giao thức mạng phải thực hiện, và mô tả các ứng dụng mạng có thể truyền tin như thế nào. Mỗi tầng có một mục đích cụ thể và được kết nối với các tầng ở ngay dưới và trên nó.

- Tầng ứng dụng (Application): định nghĩa một giao diện lập trình giao tiếp với mạng cho các ứng dụng người dùng, bao gồm các ứng dụng sử dụng các tiện ích mạng. Các ứng dụng này có thể thực hiện các tác vụ như truyền tệp tin, in ấn, e-mail, duyệt web,...



- Tầng trình diễn (Presentation): có trách nhiệm mã hóa/giải mã), nén/giải nén dữ liệu từ tầng ứng dụng để truyền đi trên mạng và ngược lại.
- Tầng phiên (Session): tạo ra một liên kết ảo giữa các ứng dụng. Ví dụ các giao thức FTP, HTTP, SMTP, ...
- Tầng giao vận (Transport): cho phép truyền dữ liệu với độ tin cậy cao chẳng hạn gói tin gửi đi phải được xác thực hay có thông điệp truyền lại nếu dữ liệu bị hỏng hay bị thất lạc, hay dữ liệu bị trùng lặp.
- Tầng mạng (Network): cho phép truy xuất tới các nút trong mạng bằng cách sử dụng địa chỉ logic được sử dụng để kết nối tới các nút khác. Các địa chỉ MAC của tầng 2 chỉ có thể được sử dụng trong một mạng LAN, nên địa chỉ IP được sử dụng để đánh địa chỉ của tầng 3 khi truy xuất tới các nút trong mạng WAN. Các router ở tầng 3 được sử dụng để định đường đi trong mạng.
- Tầng liên kết dữ liệu (Data Link): truy xuất tới một mạng vật lý bằng các địa chỉ vật lý. Địa chỉ MAC là địa chỉ của tầng 2. Các nút trên LAN gửi thông điệp cho nhau bằng cách sử dụng các địa chỉ IP, và các địa chỉ này phải được chuyển đổi

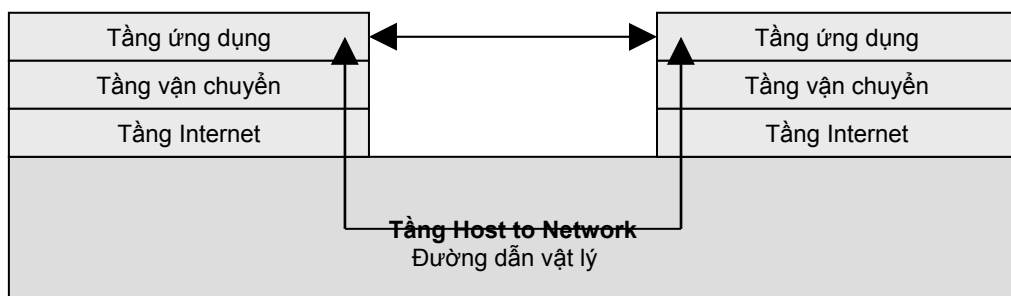
sang các địa MAC tương ứng. Giao thức phân giải địa chỉ (ARP: Address Resolution Protocol) chuyển đổi địa chỉ IP thành địa chỉ MAC. Một vùng nhớ cache lưu trữ các địa chỉ MAC để tăng tốc độ xử lý này, và có thể kiểm tra bằng tiện ích arp -a,

- Cuối cùng, tầng vật lý (Physical): bao gồm môi trường vật lý như các yêu cầu về cáp nối, các thiết bị kết nối, các đặc tả giao tiếp, hub và các repeater,...

### 1.3.2 Mô hình TCP/IP

Mô hình ISO phức tạp nên ít được sử dụng nhiều trong thực tế. Mô hình TCP/IP đơn giản và thích hợp được sử dụng cho mạng Internet. Mô hình này gồm có bốn tầng.

- Tầng ứng dụng: các phần mềm ứng dụng mạng như trình duyệt Web (IE, Firefox, ..), game online, chat, ... sử dụng các giao thức như HTTP (web), SMTP, POP, IMAP (email), FTP, FSP, TFTP (chuyển tải tập tin), NFS (truy cập tập tin), ...
- Tầng vận chuyển: đảm bảo vận chuyển gói tin tin cậy bằng cách sử dụng giao thức TCP hoặc UDP.
- Tầng Internet: cho phép định tuyến để gói tin tới đích trên mạng bằng cách sử dụng giao thức IP.
- Tầng Host to Network: vận chuyển dữ liệu thông qua thiết bị vật lý như dây cáp quang đến tầng Host to Network của hệ thống ở xa.



## 1.4 Các giao thức mạng

Các giao thức biểu diễn khuôn dạng thông tin dữ liệu tại mỗi mức giao tiếp trong mạng. Giao thức thường được phân loại theo mức độ áp dụng tại mỗi tầng trong các mô hình mạng. Trong mỗi loại giao thức được phân loại dựa vào sự phân tầng, giao thức lại được phân loại dựa vào chức năng. Ví dụ giao thức ở tầng phiên trong mô hình OSI, gồm HTTP (web), SMTP, POP, IMAP (email), FTP, FSP, TFTP (chuyển tải tập tin), NFS (truy cập tập tin), ...

### 1.4.1 TCP-Transmission Control Protocol

Giao thức TCP được sử dụng ở tầng vận chuyển (OSI) đảm bảo cho dữ liệu gửi đi tin cậy và xác thực giữa các nút mạng. Giao thức TCP phân chia dữ liệu thành các gói tin gọi là datagram. TCP gắn thêm trường header vào datagram. Trường head được mô tả trong hình vẽ bên dưới.

Source port			Destination port		
Sequence Number					
Acknowledge Number					
Offset	Reserved		Flags	Window	
Checksum			Urgent pointer		
Options				Padding	
Start of Data					

Bảng 1-1 Minh họa cấu của thông tin header của TCP

Trường	Mô tả
Cổng nguồn (source port)	Số hiệu cổng của nguồn
Cổng đích (destination port)	Số hiệu cổng đích
Số thứ tự (Sequence Number)	Số thứ tự được tạo ra bởi nguồn và được sử dụng bởi đích để sắp xếp lại các gói tin để tạo ra thông điệp ban đầu, và gửi xác thực tới nguồn.
Acknowledge Number	Cho biết dữ liệu được nhận thành công.
Data offset	Các chi tiết về nơi dữ liệu gói tin bắt đầu
Reserved	Dự phòng
Flags	chỉ ra rằng gói tin cuối cùng hoặc gói khẩn cấp
Window	chỉ ra kích thước của vùng đệm nhận. Phía nhận có thể thông báo cho phía gửi kích thước dữ liệu tối đa mà có thể được gửi đi bằng cách sử dụng các thông điệp xác thực



Checksum	xác định xem gói tin có bị hỏng không
Urgent Pointer	thông báo cho phía nhận biết có dữ liệu khẩn
Options	vùng dự phòng cho việc thiết lập trong tương lai
Padding	chỉ ra rằng dữ liệu kết thúc trong vòng 32 bit.

#### 1.4.2 UDP-User Datagram Protocol

Giao thức UDP này được sử dụng ở tầng giao vận (OSI) thực hiện cơ chế không liên kết. Nó không cung cấp dịch vụ tin cậy như TCP. UDP sử dụng IP để phát tán các gói tin này.

Nhược điểm của UDP là: Các thông điệp có thể được nhận theo bất kỳ thứ tự nào. Thông điệp được gửi đầu tiên có thể được nhận sau cùng. Không có gì đảm bảo là các gói tin sẽ đến đích, và các thông điệp có thể bị thất lạc, hoặc thậm chí có thể nhận được hai bản sao của cùng một thông điệp.

Ưu điểm của UDP là: UDP một giao thức có tốc độ truyền tin nhanh vì nó chỉ xác định cơ chế tối thiểu để truyền dữ liệu. Cụ thể trong cách thức truyền tin unicast, broadcast và multicast.

- Một thông điệp unicast được gửi từ nút này tới nút khác. Kiểu truyền tin là truyền tin điểm-điểm. Giao thức TCP chỉ hỗ trợ truyền tin unicast.
- Truyền tin broadcast nghĩa là một thông điệp có thể được gửi tới tất cả các nút trong một mạng.
- Multicast cho phép các thông điệp được truyền tới một nhóm các nút được lựa chọn.

UDP có thể được sử dụng cho truyền tin unicast nếu cần tới tốc độ truyền tin nhanh, như truyền tin đa phương tiện, nhưng ưu điểm chính của UDP là truyền tin broadcast và truyền tin multicast. Vì nếu dùng giao thức TCP thì tất cả các nút gửi về các xác thực cho server sẽ làm cho server quá tải.

Bảng 1-2 Cấu trúc trường Header của UDP

Trường thông tin	Mô tả
Source port (Cổng nguồn)	Xác định cổng nguồn là một tùy chọn với UDP. Nếu trường này được sử dụng, phía nhận thông điệp có thể gửi một phúc đáp tới cổng này
Destination Port	Số hiệu cổng đích

Length	Chiều dài của thông điệp bao gồm header và dữ liệu
Checksum	Để kiểm tra tính đúng đắn

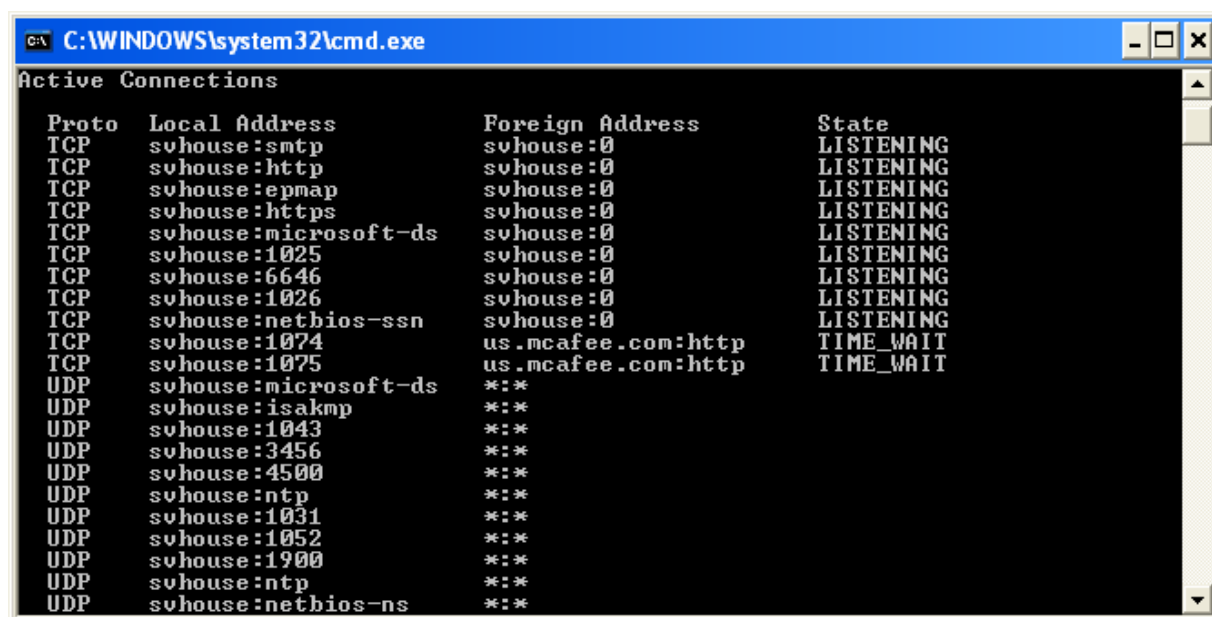
- Số hiệu cổng

Các số hiệu cổng của TCP và UDP được phân thành ba loại

- Các số hiệu cổng hệ thống
- Các số hiệu cổng người dùng
- Các số hiệu cổng riêng và động

Các số hiệu cổng hệ thống nằm trong khoảng từ 0 đến 1023. Các cổng hệ thống chỉ được sử dụng bởi các tiến trình được quyền ưu tiên của hệ thống. Các giao thức nổi tiếng có các số hiệu cổng nằm trong khoảng này.

Các số hiệu cổng người dùng nằm trong khoảng từ 1024 đến 49151. Các ứng dụng server của bạn sẽ nhận một trong các số này làm cổng, hoặc bạn có thể đăng ký số hiệu cổng với IANA .



Hình 1-1 Minh họa lệnh netstat để xem thông tin các cổng đang sử dụng

Các cổng động nằm trong khoảng từ 49152 đến 65535. Khi không cần thiết phải biết số hiệu cổng trước khi khởi động một ứng dụng, một số hiệu cổng trong khoảng này sẽ là thích hợp. Các ứng dụng client kết nối tới server có thể sử dụng một cổng như vậy.

Nếu chúng ta sử dụng tiện ích netstat với tùy chọn -a, chúng ta sẽ thấy một danh sách tất cả các cổng hiện đang được sử dụng, nó cũng chỉ ra trạng thái của liên kết-nó đang nằm trong trạng thái lắng nghe hay liên kết đã được thiết lập.

### 1.4.3 IP-Internet Protocol

Giao thức IP được thiết kế để định tuyến truyền gói tin trong mạng từ nút nguồn tới nút đích. Mỗi nút được định danh bởi một địa chỉ IP (32 bit). Khi nhận gói dữ liệu ở tầng trên (như theo khuôn dạng TCP hoặc UDP), giao thức IP sẽ thêm vào trường header chứa thông tin của nút đích.

Version	IHL	TOS	Total length	
Identification			Flags	Fragmentation Offset
Time to Live	Protocol		Header Checksum	
TCP Header				
Start of Data				

Bảng 1-3 Thông tin chi tiết của cấu trúc header của giao thức IP

Trường	Mô tả
Version (Phiên bản IP)	Phiên bản IP. (Phiên bản giao thức hiện nay là IPv4)
IP Header Length (Chiều dài Header)	Chiều dài của header.
Type of Service (Kiểu dịch vụ)	Kiểu dịch vụ cho phép một thông điệp được đặt ở chế độ thông lượng cao hay bình thường, thời gian trễ là bình thường hay lâu, độ tin cậy bình thường hay cao. Điều này có lợi cho các gói được gửi đi trên mạng. Một số kiểu mạng sử dụng thông tin này để xác định độ ưu tiên
Total Length (Tổng chiều dài)	Hai byte xác định tổng chiều dài của thông điệp-header và dữ liệu. Kích thước tối đa của một gói tin IP là 65,535, nhưng điều này là không thực tế đối với các mạng hiện nay. Kích thước lớn nhất được chấp nhận bởi các host là 576 bytes. Các thông điệp lớn có thể phân thành các đoạn-quá trình này được gọi là quá trình phân đoạn
Identification (Định danh)	Nếu thông điệp được phân đoạn, trường định danh trợ giúp cho việc lắp ráp các đoạn thành một thông điệp. Nếu một thông điệp được phân thành nhiều đoạn, tất cả các đoạn của một thông điệp có cùng một số định danh.
Flags	Các cờ này chỉ ra rằng thông điệp có được phân đoạn hay không, và liệu gói tin hiện thời có phải là đoạn cuối cùng của thông điệp hay không.
Fragment Offset	13 bit này xác định offset của một thông điệp. Các đoạn có thể đến theo một thứ tự khác với khi gửi, vì vậy trường offset là cần thiết để

	xây dựng lại dữ liệu ban đầu. Đoạn đầu tiên của một thông điệp có offset là 0
Time to Live	Xác định số giây mà một thông điệp tồn tại trước khi nó bị loại bỏ.
Protocol	Byte này chỉ ra giao thức được sử dụng ở mức tiếp theo cho thông điệp này. Các số giao thức
Header Checksum	Đây là chỉ là checksum của header. Bởi vì header thay đổi với từng thông điệp mà nó chuyển tới, checksum cũng thay đổi.
Source Address	Cho biết địa chỉ IP 32 bit của phía gửi
Destination Address	Địa chỉ IP 32 bit của phía nhận
Options	
Padding	

- Địa chỉ IP

IPv4-32 bit được dùng để định danh mỗi nút trên mạng TCP/IP. Thông thường một địa chỉ IP được biểu diễn bởi bốn phần x.x.x.x, chẳng hạn 192.168.0.1. Mỗi phần là một số có giá trị từ 0 đến 255. Một địa chỉ IP gồm hai phần: phần mạng và phần host.

Bảng 1-4 Minh họa phân loại địa chỉ lớp mạng

Lớp	Byte 1	Byte 2	Byte 3	Byte 4
A (0)	Networks (1-126)	Host (0-255)	Host (0-255)	Host (0-255)
B (10)	Networks (128-191)	Networks (0-255)	Host (0-255)	Host (0-255)
C (110)	Networks (192-223)	Networks (0-255)	Networks (0-255)	Host (0-255)

Trong đó địa chỉ lớp D(1110) được sử dụng cho địa chỉ multicast. Địa chỉ dự phòng (01111111). Địa chỉ 127.0.0.1 là địa chỉ của localhost, và địa chỉ 127.0.0.0 là địa chỉ loopback.

Để tránh cạn kiệt các địa chỉ IP, các host không được kết nối trực tiếp với Internet có thể sử dụng một địa chỉ trong các khoảng địa chỉ riêng. Các địa chỉ IP riêng không duy nhất về tổng thể, mà chỉ duy nhất về mặt cục bộ trong phạm vi mạng đó. Tất cả các lớp mạng dự trữ các khoảng nhất định để sử dụng như là các địa chỉ riêng cho các host không cần truy cập trực tiếp tới Internet. Các host như vậy vẫn có thể truy cập Internet thông qua một gateway mà không cần chuyển tiếp các địa chỉ IP riêng.

Lớp	Khoảng địa chỉ riêng
A	10

B	172.16-172.31
C	192.168.0-192.168.255

IPv6 được sử dụng 128 bit để biểu diễn địa chỉ nhằm biểu diễn nhiều hơn số địa chỉ của nút trên mạng.

Lớp	Cấu trúc địa chỉ IP	Format	Số bit mạng/số bit host	Tổng số mạng/lớp	Tổng số host/mạng	Vùng địa chỉ IP
A	0 netid hostid	N.H.H.H	7/24	$2^7-2=126$	$2^{24}-2=17.777.214$	1.0.0.1-126.0.0.0
B	1 0 netid hostid	N.N.H.H	14/16	$2^{14}-2=16382$	$2^{16}-2=65.536$	128.0.0.0-191.255.255.255
C	1 1 0 netid hostid	N.N.N.H	22/8	$2^{22}-2=4194302$	$2^8-2=254$	192.0.0.0-223.255.255.255
D	1 1 1 0 địa chỉ multicast	-	-	-	-	224.0.0.0-239.255.255.255
E	1 1 1 1	-	-	-	-	240.0.0.0-255.255.255.255
Loopback	-	-	-	-	-	127.x.x.x

- Các subnet

Việc kết nối hai nút của hai mạng khác nhau cần có một router. Định danh host của mạng lớp A cần có 24 bit; trong khi mạng lớp C, chỉ có 8 bit. Để cấp phát địa chỉ IP cho các mạng khác nhau một cách hiệu quả và dễ quản lý, một kỹ thuật được sử dụng gọi là subnet. Subnet sẽ vay mượn một số bit của hostid để làm subnet mask (mặt nạ mạng).

- Subnet mask có tất cả các bit network và subnet đều bằng 1, các bit host đều bằng 0
- Tất cả các máy trên cùng một mạng phải có cùng một subnet mask
- Để phân biệt được các subnet (mạng con) khác nhau, bộ định tuyến dùng phép logic AND

Ví dụ: địa chỉ lớp mạng lớp B 128.10.0.0 có thể dùng 8 bit đầu tiên của hostid để subnet:  
Subnet mask = 255.255.255.0

Network	Network	Subnet	Host
11111111	11111111	11111111	00000000

255	255	255	0
-----	-----	-----	---

Như vậy, số bit dành cho subnet sẽ là 8 -> có tất cả  $2^8=256$  subnet (mạng con). Địa chỉ của các subnet lần lượt là 128.10.0.1, 128.10.0.2, 128.10.0.3, ..., 128.10.0.255. 8 bit dành cho host nên mỗi subnet sẽ có  $2^8=256$  host, địa chỉ của các host lần lượt là 128.10.xxx.1, 128.10.xxx.2, 128.10.xxx.3, ..., 128.10.xxx.255

### 1.5 Mô hình khách/chủ (client/server)

Hầu hết các chương trình mạng hiện nay sử dụng theo mô hình client/server. Một chương trình chạy ở máy chủ-server mạnh, quản lý một lượng dữ liệu lớn. Một chương trình chạy ở máy khách-client (máy tính cá nhân) thực hiện khai thác dữ liệu ở máy chủ. Trong hầu hết trường hợp, máy chủ gửi dữ liệu, máy khách nhận dữ liệu. Thông thường, client sẽ thiết lập cuộc giao tiếp, và server sẽ đợi yêu cầu thiết lập từ client và giao tiếp với nó.

### 1.6 Socket

Socket là biểu diễn trừu tượng hóa một cơ chế kết nối giữa hai ứng dụng trên hai máy bằng cách sử dụng kết hợp địa chỉ IP và số hiệu cổng. Nó cho phép các ứng dụng gửi và nhận dữ liệu cho nhau. Cả client và server đều sử dụng socket để giao tiếp với nhau. Trong ngôn ngữ Java, một socket được biểu diễn bằng một đối tượng của một trong các lớp java.net như Socket, ServerSocket, DatagramSocket hoặc MulticastSocket.

### 1.7 Dịch vụ tên miền

Địa chỉ IP được viết dưới dạng 4 nhóm con số nên người sử dụng rất khó nhớ. Vì vậy hệ thống tên miền được sử dụng để hỗ trợ cho người sử dụng. Các máy tính chuyên dụng để lưu trữ và phân giải tên miền bằng cách lưu danh sách địa chỉ IP và tên miền được gọi là Máy chủ DNS. Ví dụ [www.microsoft.com](http://www.microsoft.com), [www.bbc.co.uk](http://www.bbc.co.uk). Các tên này không bắt buộc phải có ba phần, nhưng việc đọc bắt đầu từ phải sang trái, tên bắt đầu với miền mức cao. Các miền mức cao là các tên nước cụ thể hoặc tên các tổ chức và được định nghĩa bởi tổ chức IANA. Các tên miền cấp cao được liệt kê trong bảng sau. Trong những năm gần đây, một số tên miền cấp cao mới được đưa vào.

Bảng 1-5 Một số tên miền cao cấp

Tên miền	Mô tả
.aero	Công nghiệp hàng không
.biz	Doanh nghiệp
.com	Các tổ chức thương mại
.coop	Các quan hệ hợp tác
.info	Không ràng buộc về sử dụng
.museum	Các viện bảo tàng
.name	Các tên cá nhân

Bảng 1-6 Một số tên miền cao cấp khác

Tên miền	Mô tả
.net	Các mạng
.org	Các tổ chức phi chính phủ
.pro	Các chuyên gia
.gov	Chính phủ Hoa Kỳ
.edu	Các tổ chức giáo dục
.mil	Quân đội Mỹ
.int	Các tổ chức được thành lập bởi các hiệp ước quốc tế giữa các chính phủ.

Bảng 1-7 Tên miền quốc gia

Tên miền	Mô tả
.at	Australia
.de	Germany
.fr	France
.uk	United Kingdom
.vn	Vietnam

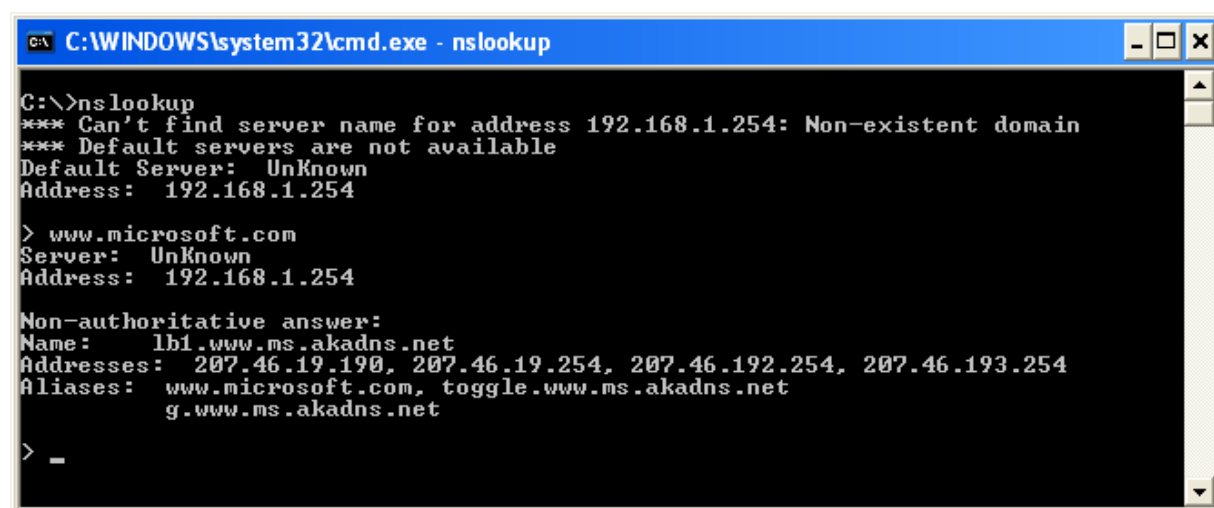
### 1.7.1 Các server tên miền

Các hostname được phân giải bằng cách sử dụng các server DNS (Domain Name Service). Các server này có một cơ sở dữ liệu các hostname và các bí danh ánh xạ các tên thành địa chỉ IP. Ngoài ra, các DNS cũng đăng ký thông tin cho các Mail Server, các số ISDN, các tên hộp thư, và các dịch vụ.

Trong Windows, chính các thiết lập TCP/IP xác định server DNS được sử dụng để truy vấn. Lệnh ipconfig/all chỉ ra các server DNS đã được thiết lập và các thiết lập cấu hình khác. Khi kết nối với một hệ thống ở xa sử dụng hostname, trước tiên server DNS được truy vấn để tìm địa chỉ IP. Trước tiên, DNS kiểm tra trong bộ cơ sở dữ liệu của riêng nó và bộ nhớ cache. Nếu thất bại trong việc phân giải tên, server DNS truy vấn server DNS gốc.

### 1.7.2 Nslookup

Dịch vụ tên miền (Domain Name Service) Là tập hợp nhiều máy tính được liên kết với nhau và phân bố rộng trên mạng Internet. Các máy tính này được gọi là name server. Chúng cung cấp cho người dùng tên, địa chỉ IP của bất kỳ máy tính nào nối vào mạng Internet hoặc tìm ra những name server có khả năng cung cấp thông tin này.



```
C:\>nslookup
*** Can't find server name for address 192.168.1.254: Non-existent domain
*** Default servers are not available
Default Server: UnKnown
Address: 192.168.1.254

> www.microsoft.com
Server: UnKnown
Address: 192.168.1.254

Non-authoritative answer:
Name: lb1.www.ms.akadns.net
Addresses: 207.46.19.190, 207.46.19.254, 207.46.192.254, 207.46.193.254
Aliases: www.microsoft.com, toggle.www.ms.akadns.net
g.www.ms.akadns.net

> _
```

Hình 1-2 Minh họa dùng lệnh nslookup để tìm địa chỉ IP

Cơ chế truy tìm địa chỉ IP thông qua dịch vụ DNS

Giả sử trình duyệt cần tìm tập tin hay trang Web của một máy chủ nào đó, khi đó cơ chế truy tìm địa chỉ sẽ diễn ra như sau:

1. Trình duyệt yêu cầu hệ điều hành trên client chuyển hostname thành địa chỉ IP.
2. Client truy tìm xem hostname có được ánh xạ trong tập tin localhost, hosts hay không?
  - a. Nếu có client chuyển đổi hostname thành địa chỉ IP và gửi về cho trình duyệt.
  - b. Nếu không client sẽ tìm cách liên lạc với máy chủ DNS.



3. Nếu tìm thấy địa chỉ IP của hostname máy chủ DNS sẽ gửi địa chỉ IP cho client.
4. Client gửi địa chỉ IP cho trình duyệt.
5. Trình duyệt sử dụng địa chỉ IP để liên lạc với Server.
6. Quá trình kết nối thành công. Máy chủ gửi thông tin cho client.

## **1.8 Các vấn đề liên quan Internet**

### *1.8.1 Intranet và Extranet*

Một intranet có thể sử dụng các công nghệ TCP/IP tương tự như với Internet. Sự khác biệt là intranet là một mạng riêng, trong đó tất cả mọi người đều biết nhau. Intranet không phục vụ cho việc truy xuất chung, và một số dữ liệu cần phải được bảo vệ khỏi những truy xuất từ bên ngoài.

Một extranet là một mạng riêng giống như intranet nhưng các extranet kết nối nhiều Intranet thuộc cùng một công ty hoặc các công ty đối tác thông qua Internet bằng cách sử dụng một tunnel. Việc tạo ra một mạng riêng ảo trên Internet tiết kiệm chi phí nhiều cho công ty so với việc thuê riêng một đường truyền để thiết lập mạng.

### *1.8.2 Firewall*

Có những kẻ phá hoại trên mạng Internet!. Để ngăn chặn chúng, người ta thường thiết lập các điểm truy cập tới một mạng cục bộ và kiểm tra tất cả các luồng truyền tin vào và ra khỏi điểm truy nhập đó. Phần cứng và phần mềm giữa mạng Internet và mạng cục bộ, kiểm tra tất cả dữ liệu vào và ra, được gọi là firewall.

Firewall đơn giản nhất là một bộ lọc gói tin kiểm tra từng gói tin vào và ra khỏi mạng, và sử dụng một tập hợp các quy tắc để kiểm tra xem luồng truyền tin có được phép vào ra khỏi mạng hay không. Kỹ thuật lọc gói tin thường dựa trên các địa chỉ mạng và các số hiệu cổng.

### *1.8.3 Proxy Server*

Khái niệm proxy có liên quan đến firewall. Một firewall ngăn chặn các host trên mạng liên kết trực tiếp với thế giới bên ngoài. Một máy bị ngăn kết nối với thế giới bên ngoài bởi một firewall sẽ yêu cầu truy xuất tới một trang web từ một proxy server cục bộ, thay vì yêu cầu một trang web trực tiếp từ web server ở xa. Proxy server sau đó sẽ yêu cầu trang web từ một web server, và sau đó chuyển kết quả trở lại cho bên yêu cầu ban đầu.

Các proxies cũng được sử dụng cho FTP và các dịch vụ khác. Một trong những ưu điểm bảo mật của việc sử dụng proxy server là các host bên ngoài chỉ nhìn thấy proxy server. Chúng không biết được các tên và các địa chỉ IP của các máy bên trong, vì vậy khó có thể đột nhập vào các hệ thống bên trong.

Trong khi các firewall hoạt động ở tầng giao vận và tầng internet, các proxy server hoạt động ở tầng ứng dụng. Một proxy server có những hiểu biết chi tiết về một số giao thức mức ứng dụng, như HTTP và FTP. Các gói tin đi qua proxy server có thể được kiểm tra để đảm bảo rằng chúng chứa các dữ liệu thích hợp cho kiểu gói tin. Ví dụ, các gói tin FTP chứa các dữ liệu của dịch vụ telnet sẽ bị loại bỏ.

Vì tất cả các truy nhập tới Internet được chuyển hướng thông qua proxy server, vì thế việc truy xuất có thể được kiểm soát chặt chẽ. Ví dụ, một công ty có thể chọn giải pháp phong tỏa việc truy xuất tới [www.playboy.com](http://www.playboy.com) nhưng cho phép truy xuất tới [www.microsoft.com](http://www.microsoft.com).

## **1.9 Bài tập**

1. Tìm hiểu các giao thức chuẩn hóa phổ biến khác ở các tầng khác nhau của mô hình OSI.
2. Tìm hiểu các giao thức được sử dụng phổ biến trong các ứng dụng trên mạng Internet. Cho biết số hiệu cổng của các ứng dụng này. Liệt kê các ứng dụng thực tế sử dụng giao thức này. Liệt kê các phần mềm nguồn mở đang phát triển sử dụng các giao thức này.

## CHƯƠNG 2 . NGÔN NGỮ LẬP TRÌNH JAVA

### 2.1 Giới thiệu

Ngôn ngữ lập trình Java được phát triển bởi Sun Microsystems vào năm 1991 như một phần của dự án xanh. Một thành viên của dự án là Jame Gosling đã không đánh giá cao ngôn ngữ lập trình C++ và đã đưa ra một ngôn ngữ mới đặt tên là Oak. Sau đó, Sun đổi tên lại là Java.

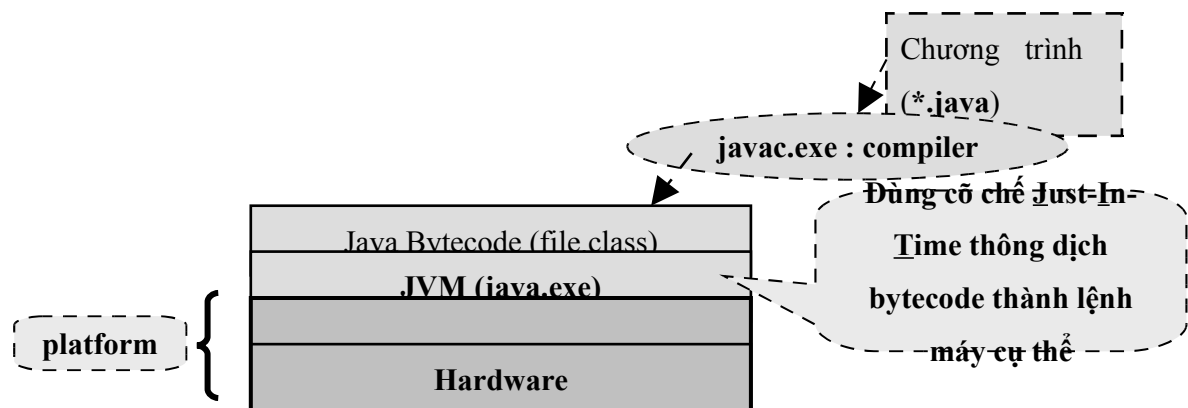
Java được chính thức công bố năm 1995 và ngay lập tức đã tạo lên một trào lưu mới trên toàn thế giới và từ đó đến nay vẫn tạo được sức cuốn hút mạnh mẽ.

Java là ngôn ngữ lập trình thuần hướng đối tượng và nó cung cấp những đặc trưng mạnh mẽ cho lập trình mạng (lập trình Internet). Ưu điểm của ngôn ngữ lập trình java thể hiện qua kiến trúc và bộ thư viện rất lớn.

### 2.2 Môi trường lập trình Java

Để viết một ứng dụng java và chạy được, môi trường lập trình yêu cầu cần có bộ công cụ JDK. Bộ công cụ này có thể download tại website của Sum (java.sun.com). Bộ công cụ này chứa trình biên dịch, trình thông dịch, trình gỡ rối và tất cả thư viện có sẵn hỗ trợ cho lập trình. Trong đó, có 2 tập tin chính của việc lập trình java là

- javac.exe là trình biên dịch tập tin \*.java thành mã trung gian \*.class.
- java.exe là trình thông dịch (thành phần máy ảo) để thông dịch chạy tập tin \*.class



Để soạn thảo câu lệnh, một trình soạn thảo văn bản bất kỳ đều có thể sử dụng như Notepad, Notepad++, Editplus, Jcreator hoặc sử dụng môi trường tích hợp IDE như NetBean.

## 2.3 Một số ví dụ mở đầu

Ví dụ 1: Tạo chương trình Java cho phép nhập một dòng ký tự từ đối dòng lệnh và hiển thị xâu ký tự đó lên trên màn hình:

```
class Hello
{
    public static void main(String[] args)
    {
        System.out.println(args[0]);
    }
}
```

Biên dịch chương trình

```
C:\>javac Hello.java
```

Thực hiện chương trình

```
C:\>java Hello "Chao ngon ngu lap trinh Java"
```

Kết quả in ra là:

```
Chao ngon ngu lap trinh Java
```

Ví dụ 2: Tạo một applet hiển thị một xâu ký tự lên màn hình

```
import java.applet.*;
import java.awt.*;
public class HelloApplet extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("Hello applet", 30, 30);
    }
}
```

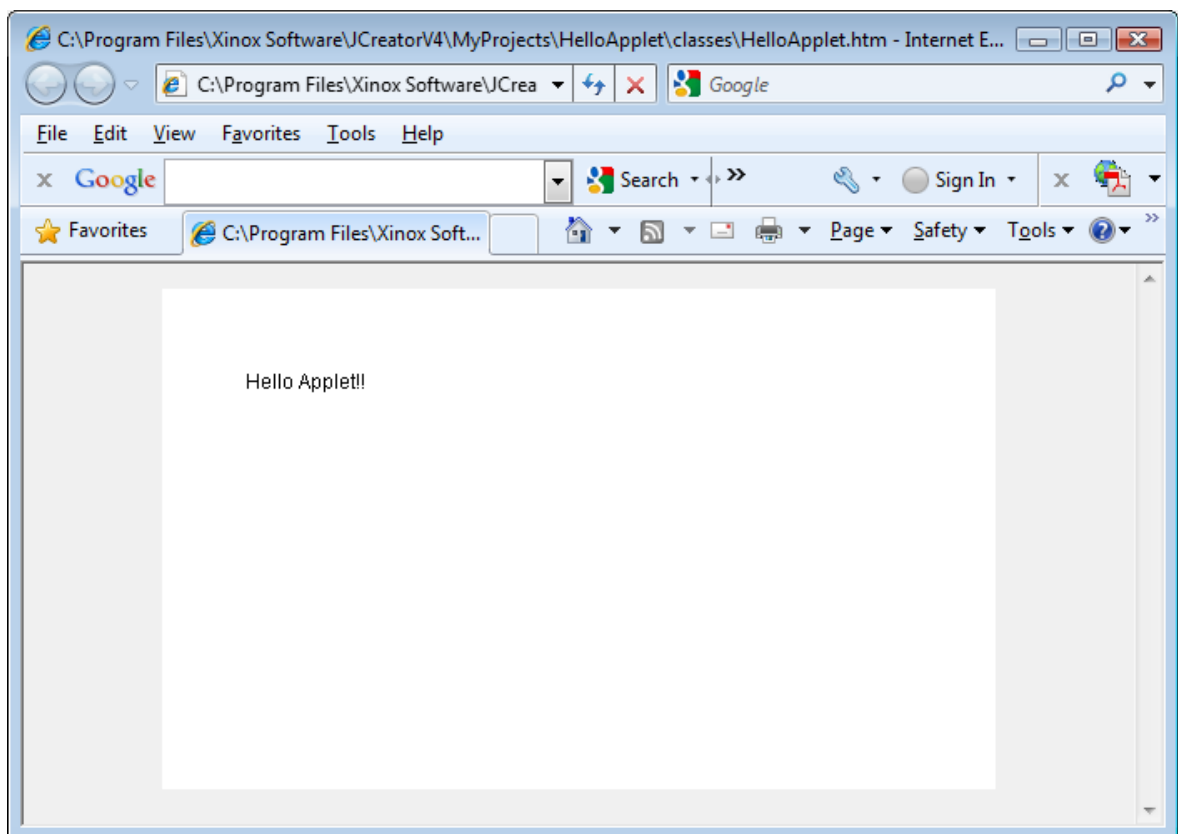
Biên dịch applet

```
C:\>javac HelloApplet.java
```

Tạo một trang VDApplet.html để nhúng applet

```
<html>
  <head>      </head>
  <body bgcolor="#f0f0f0">
    <center>
      <applet    code = "HelloApplet.class"
                width = "500"
                height      = "300" >
      </applet>
    </center>
  </body>
</html>
```

Thực thi applet bằng cách mở tập tin VDAppllet.html trong trình duyệt. Kết quả được thể hiện ở hình dưới đây:



Hình 2-3 Minh họa chạy chương trình applet

## 2.4 Các thành phần cơ bản của ngôn ngữ lập trình Java

### 2.4.1 Định danh

Định danh (Identifier) Tên gọi của các thành phần trong chương trình được gọi là định danh. Định danh thường được sử dụng để xác định biến, kiểu, phương thức, lớp.

Qui tắc cho định danh:

- Định danh là một dãy các ký tự gồm các chữ cái, chữ số và các ký tự khác: '\_', '\$',...
- Định danh không bắt đầu bằng chữ số.
- Độ dài của định danh không giới hạn.
- Java phân biệt chữ hoa và chữ thường.

Qui ước đặt tên

- Định danh cho các lớp: chữ cái đầu của mỗi từ trong định danh đều viết hoa

Ví dụ: MyClass, HocSinh, SocketServer, URLConnection,...

- Định danh cho các biến, phương thức, đối tượng: chữ cái đầu của mỗi từ trong định danh đều viết hoa trừ từ đầu tiên.

Ví dụ: hoTen (họ tên), namSinh (năm sinh), tinhTong (tính tổng).

### 2.4.2 Các kiểu dữ liệu nguyên thủy (primitive datatype)

Kiểu dữ liệu cơ bản định nghĩa sẵn được gọi là kiểu nguyên thủy. Biến thuộc kiểu dữ liệu nguyên thủy lưu giá trị. Kiểu nguyên thủy bao gồm các kiểu:

- Kiểu nguyên: char (2byte), byte(1byte), short(2byte), int(4byte), long(8byte).
- Kiểu số thực: float(4byte), double(8byte).
- Kiểu logic: bool.

Mỗi kiểu dữ liệu nguyên thủy có một lớp bao bọc(wrapper class) cung cấp các chức năng thao tác trên kiểu dữ liệu này.

Kiểu dữ liệu	Lớp gói
char	Char
byte	Byte

short	Short
int	Integer
long	Long
float	Float
double	Double

### 2.4.3 Khai báo các biến

Cú pháp: <tên kiểu> <tên biến>;

ví dụ:

```
int i;
i=5;
URL u ;
HocSinh hs = new HocSinh("Tuan Anh");
```

Mỗi biến được khai báo với một kiểu xác định. Có hai kiểu biến. Biến kiểu thuộc dữ liệu nguyên thủy sẽ lưu giá trị. Biến thuộc kiểm đối tượng sẽ lưu địa chỉ tham chiếu tới đối tượng (địa chỉ đối tượng, địa chỉ định nghĩa lớp đối tượng). Cần chú ý rằng một biến khi được khai báo, cần phải thiết lập giá trị của nó trước khi sử dụng.

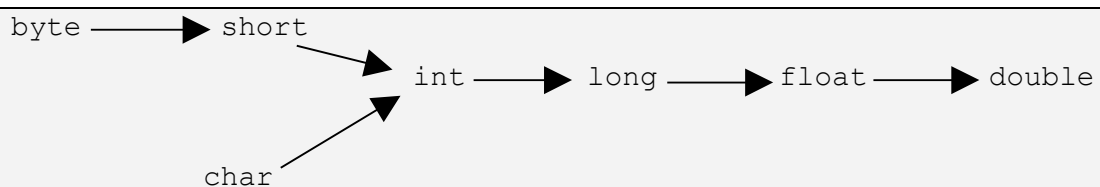
Quy tắc chuyển đổi kiểu trong Java

(<kieu>) <bieu\_thuc>

Ví dụ

```
float f = (float)100.15D;
```

Mở rộng và thu hẹp kiểu



Ví dụ mở rộng kiểu

```
char c = 'A';
int k = c;
```

Ví dụ thu hẹp kiểu

```
int k =10;
char c=(char)k;
```

#### 2.4.4 Các câu lệnh cơ bản

- Khối lệnh

Khối lệnh trong Java tương tự như khối lệnh trong C/C++, là những lệnh nằm trong cặp ngoặc mở { và đóng }. Một khối lệnh là một dãy không hoặc nhiều lệnh hoặc các khai báo biến hoặc khai báo lớp theo bất kỳ thứ tự nào được đặt trong cặp ngoặc {}

```
{
    S1;
    ...
    Sn;
}
```

- Lệnh gán

```
int a, b, c, d;
d=b*b-4*a*c;
```

- Biểu thức điều kiện

Biểu thức điều kiện A?B:C trả về giá trị B nếu A có giá trị true, trả về giá trị C nếu A có giá trị false.

Ví dụ:

```
byte b;
int i=b>=0?b:b+255;
```

- Trong ví dụ trên thực hiện việc chuyển đổi các số nguyên kiểu byte có dấu về số nguyên kiểu int không có dấu. Nếu b lớn hơn hoặc bằng 0 thì I nhận giá trị là b, ngược lại I sẽ nhận giá trị là 255+b.
- Các lệnh điều khiển rẽ nhánh chương trình

- Lệnh if đơn giản

Cú pháp

```
if<biểu_thức_đk> <câu_lệnh>
```

- Lệnh if – else



### Cú pháp

```
if <biểu_thức_đk>
<câu_lệnh_1>;
else
<câu_lệnh_2>;
```

Ví dụ: Viết chương trình nhập vào một dãy số nguyên từ dòng lệnh, sắp xếp dãy số đó và hiển thị dãy số sau khi sắp xếp lên màn hình.

```
class SapXep
{
    public static void main(String[] args)
    {
        int a[]=null;
        int i,j,tg;
        if(args.length>0)
        {
            a=new int[args.length];
        }
        for(i=0;i<args.length;i++)
        {
            a[i]=Integer.parseInt(args[i]);
            System.out.print(a[i]+" ");
        }
        System.out.println();
        for(i=0;i<a.length-1;i++)
            for(j=i+1;j<a.length;j++)
                if(a[i]>a[j])
                {
                    tg=a[i];
                    a[i]=a[j];
                    a[j]=tg;
                }
        System.out.println("Day so sau khi sap xep la:");
        for(i=0;i<a.length;i++)    System.out.print(a[i]+" ");
    }
}
```

```
}
```

Biên dịch chương trình

```
C:\>javac SapXep.java
```

Thực hiện chương trình

```
C:\>java SapXep -2 3 1 4 -5 6 -10
-2 3 1 4 -5 6 -10
Day so sau khi sap xep la:
-10 -5 -2 1 3 4 6
```

- **Lệnh switch**

Lệnh này cho phép rẽ nhánh theo nhiều nhánh tuyến chọn dựa trên các giá trị nguyên của biểu thức.

Cú pháp:

```
switch(<biểu_thức_nguyên>)
{
    case nhan_1:<câu_lệnh_1>;
    case nhan_2:<câu_lệnh_2>;
    case nhan_3:<câu_lệnh_3>;
    ...
    case nhan_n:<câu_lệnh_n>;
    default:<câu_lệnh>;
}
```

- **Lệnh lặp**

Lệnh lặp cho phép một khối các câu lệnh thực hiện một số lần lặp lại.

- **Lệnh while**

```
while (<Điều_kiện>) <Thân_chu_trình>
```

Chú ý:

Có thể <Thân\_chu\_trình> không được thực hiện lần nào nếu ngay từ đầu

<Điều\_kiện> có giá trị false.

<Điều\_kiện> là biểu thức boolean.

Ví dụ: Lập chương trình in ra dãy số Fibonacci có các giá trị nhỏ hơn 50.

```
class Fibonacci
{
```

```

public static void main(String[] args)
{
    int lo=1;
    int hi =1;
    System.out.println(lo);
    while(hi<50)
    {
        System.out.println(hi);
        hi=lo+hi;
        lo=hi-lo;
    }
}

```

#### - **Lệnh do – while**

##### Cú pháp

```

do
{
    //Các lệnh
}
while(<Điều_kiện>);

```

Chú ý:

Thân chu trình được thực hiện ít nhất một lần

#### - **Lệnh for**

##### Cú pháp

```

for(<Biểu_thức_bắt_đầu>;<Điều_kiện_lặp>;Biểu_thức_gia_tăng>)
    <Thân_chu_trình>

```

- <Biểu\_thức\_bắt\_đầu>: Khai báo và gán các giá trị khởi đầu cho các biến điều khiển quá trình lặp.
- <Điều\_kiện\_lặp>: Biểu thức logic.
- Sự tương đương giữa vòng for và while

```

<Biểu_thức_bắt_đầu>;
while(<Điều_kiện_lặp>)
{

```

```
<lệnh>;  
<Biểu_thức_gia_tăng>;  
}
```

- Các câu lệnh nhảy

Java hỗ trợ ba lệnh nhảy: break, continue, và return. Các lệnh này truyền điều khiển sang phần khác của chương trình.

- **Lệnh break:** Được dùng để kết thúc trình tự thực hiện lệnh trong lệnh switch hoặc được sử dụng để thoát ra khỏi một vòng lặp.

```
class BreakDemo  
{  
    public static void main(String[] args)  
    {  
        for(int i=0;i<20;i++){  
            if(i==3)break;  
            System.out.println("i="+i);  
        }  
    }  
}
```

Chương trình này in ra kết quả là:

```
i=0  
i=1  
i=2
```

Như chúng ta có thể thấy, vòng lặp for được thiết kế để chạy với các giá trị từ 0 đến 20, lệnh break làm cho vòng lặp kết thúc sớm khi i bằng 3.

- **Lệnh continue:** Lệnh continue sẽ bỏ qua không xử lý một lệnh nào đó trong vòng lặp nhưng vẫn tiếp tục thực hiện phần còn lại của vòng lặp.

```
class ContinueDemo  
{  
    public static void main(String[] args)  
    {  
        for(int i=0;i<5;i++){  
            if(i==2)continue;  
        }  
    }  
}
```

```

        System.out.println("i="+i);
    }

}
}

```

Kết quả thực hiện chương trình trên như sau:

```

i=0
i=1
i=3
i=4

```

Chương trình không in ra giá trị i=3.

- **Lệnh return:** Được sử dụng để kết thúc việc thực hiện của hàm hiện thời và chuyển điều khiển chương trình về cho chương trình đã gọi phương thức

Ví dụ

```

protected double giaTriKhongAm(double v)
{
    if(v<0)
        return 0;
    else
        return v;
}

```

## 2.5 Lớp đối tượng

Đơn vị cơ bản trong lập trình Java là lớp. Lớp đối tượng miêu tả đặc trưng (thuộc tính) và khả năng (phương thức) của đối tượng. Thuộc tính lưu dữ liệu về đối tượng. Phương thức là các hàm thao tác trên dữ liệu của đối tượng. Java hỗ sẵn một thư viện rất lớn các lớp đối tượng trong gói java.\*, javax.\* và nhiều gói khác.

### 2.5.1 Cú pháp định nghĩa một lớp đối tượng

```

[<phạm vi>] class <Tên lớp> [extends <Tên lớp cha>]
                                   [implements <Tên giao diện>]
{
    < Các thành phần lớp:

```

```

        kiểu_dữ_liệu1 biến1
        ...
        kiểu_dữ_liệun biếnn
        phương_thức1()
        ...
        phương_thứcm()    >
    }

```

### 2.5.2 Cú pháp định nghĩa phương thức

```

[<Phạm vi>] <Kiểu trả về> <Tên phương thức>
([<Danh sách tham biến hình thức>]) [<Mệnh đề throws>]
{
    <Nội dung phương thức>
}

```

Trong đó

- <Kiểu trả về> có thể là kiểu nguyên thủy, kiểu lớp hoặc không có giá trị trả lại (kiểu void)
- <Danh sách tham biến hình thức> bao gồm dãy các tham biến (kiểu và tên) phân cách với nhau bởi dấu phẩy.
- Các kiểu phạm vi
  - o public: Các thành phần được khai báo là public có thể được truy cập ở bất kỳ nơi nào có thể truy cập được và chúng được thừa kế bởi các lớp con của nó.
  - o private: Các thành phần được khai báo là private chỉ có thể được truy cập trong chính lớp đó
  - o protected: Các thành phần được khai báo là protected có thể được truy cập bởi mã lệnh trong lớp, lớp thừa kế và các đối tượng cùng gói.

### 2.5.3 Hàm khởi tạo - Constructor

Hàm khởi tạo là một phương thức đặc biệt không có giá trị trả về và có tên trùng với tên lớp. Trường hợp không có constructor nào được đưa ra trình biên dịch cung cấp constructor mặc định cho lớp đó.

Khi tạo ra một đối tượng, hàm khởi tạo ứng với bộ tham số truyền vào sẽ tự động gọi lên thực hiện. Hàm này cho phép thiết lập những giá trị ban đầu cho các thuộc tính của đối tượng vừa tạo ra.

```
public class Point
{
    protected double x,y;
    public Point(double x,double y)
    {
        this.x=x;
        this.y=y;
    }
    public void move(double dx, double dy)
    {
        x=x+dx;
        y=y+dy;
    }
    public void print()
    {
        System.out.println("x="+x+", y="+y);
    }
    public static void main(String[] args)
    {
        Point p=new Point(3.0,6.0);
        System.out.println("Thong tin ve toa do diem ban
                                dau:");

        p.print();
        p.move(-1.5,2.0);
        System.out.println("Thong tin ve toa do diem sau khi
                                tinh tien theo vec to:");

        p.print();
    }
}
```

```
C:\>java Point
```

Thông tin về tọa độ điểm ban đầu:

x=3.0, y=6.0

Thông tin về tọa độ điểm sau khi tính tiến theo vectơ:

x=1.5, y=8.0

#### 2.5.4 Tham chiếu this

Thông thường, đối tượng nhận phương thức cần phải biết tham chiếu của nó.

Trong lớp Point có constructor

```
public Point(double x, double y)
{
    this.x=x;
    this.y=y;
}
```

Giả sử nếu sửa đổi constructor bằng cách thay đổi như sau:

```
public Point(double x, double y)
{
    x=x;
    y=y;
}
```

Khi biên dịch chương trình sẽ báo lỗi vì có sự nhập nhằng trong các lệnh gán. Tham chiếu `this` đã khắc phục được điều này, `this.x` và `this.y` muốn đề cập tới trường thông tin của lớp Point còn `x`, `y` là tham biến truyền vào của constructor.

#### 2.5.5 Thừa kế

```
class Point3C extends Point2C
{
    protected double z;
    public Point3C(double x, double y, double z)
    {
        super(x, y);
        this.z=z;
    }
    public void move(double dx, double dy, double dz)
    {
```



```

        super.move(dx,dy);
        z+=dz;
    }
    public void print()
    {
        super.print();
        System.out.print("    z="+z);
    }
    public static void main(String[] args)
    {
        Point3C p=new Point3C(3.0,4.5,5.0);
        System.out.println("Toa do ban dau:");
        p.print();
        System.out.println();
        p.move(-1.0,0.5,-1.0);
        System.out.println("Toa do sau khi tinh tien:");
        p.print();
        System.out.println();

    }
}

```

#### 2.5.6 Từ khóa super

Một lớp con kế thừa đầy đủ các thuộc tính và phương thức của lớp cha. Trong đó, có một số thuộc tính và phương thức của lớp cha không truy xuất được do từ khóa phạm vi của nó hoặc do phương thức lớp cha viết đè. Để truy xuất các thành phần của lớp cha này, từ khóa super được sử dụng.

Đoạn mã thứ nhất khai báo lớp Point2C biểu diễn một đối tượng điểm hai chiều, đoạn mã thứ hai khai báo lớp Point3C biểu diễn một đối tượng điểm ba chiều. Lớp Point3C được kế thừa lớp từ lớp Point2C. Lời gọi super(x,y) trong lớp Point3C gọi tới constructor Point2C hay super.move(dx,dy) gọi tới phương thức move(dx,dy) của lớp Point2C.

Biên dịch chương trình

```
C:\>javac Point3C.java
```

Thực thi chương trình

```
C:\>java Point3C
```

### Kết quả chương trình

```
Toa do ban dau:  
x=3.0, y=4.5 z=5.0  
Toa do sau khi tinh tien:  
x=2.0, y=5.0 z=4.0
```

### 2.5.7 Truyền tham số trong Java

Thông thường, trong một ngôn ngữ lập trình thường có hai cách truyền tham biến cho một thủ tục: truyền theo tham trị và truyền theo tham chiếu. Các giá trị của các biến tham số trong một phương thức là các bản sao của các giá trị do người gọi xác định. Sự khác biệt là truyền tham trị thì tham số lưu giá trị thuộc kiểu dữ liệu nguyên thủy còn truyền tham chiếu thì tham số lưu giá trị là địa chỉ ô nhớ của đối tượng thuộc một kiểu nhất định.

### 2.5.8 Đa hình

Tính đa hình của một đối tượng được thể hiện qua kỹ thuật nạp chồng phương thức (overloaded method). Các phương thức nạp chồng là các phương thức nằm trong cùng một lớp có cùng tên nhưng khác nhau về danh sách tham số.

Ví dụ

```
class TinhToan  
{  
    public static void main(String[] args)  
    {  
        Tinh c = new Tinh();  
        c.add(10,20);  
        c.add(40.0f,35.65f);  
        c.add("Good ", "Morning");  
    }  
}  
class Tinh  
{
```

```

public void add(int a, int b)
{
    int c = a+b;
    System.out.println("Phep cong hai so nguyen :"+c);
}
public void add(float a, float b)
{
    float c = a+b;
    System.out.println("Phep cong hai so dau phay dong
                        :"+c);
}
public void add(String a, String b)
{
    String c = a+b;
    System.out.println("Phep cong hai xau :"+c);
}
};

```

**Kết quả:**

```

C:\MyJava\Baitap>java TinhToan
Phep cong hai so nguyen :30
Phep cong hai so dau phay dong :75.65
Phep cong hai xau :Good Morning

```

Giải thích: Trong chương trình trên phương thức add() là phương thức được nạp chồng. Có ba phương thức có cùng tên add() nhưng có các tham số khác nhau. Khi phương thức add được gọi, dựa vào danh sách tham số truyền vào, phương thức tương ứng sẽ được triệu gọi ra thực hiện.

### 2.5.9 Thành phần tĩnh

Các thành phần tĩnh là thành phần độc lập của lớp. Thành phần này có thể truy cập qua tên. Mọi đối tượng trong lớp đều có thể truy cập đến các thành phần này. Các thành phần tĩnh bao gồm: biến tĩnh, phương thức tĩnh, khối tĩnh.

Ví dụ về biến static

```

class StaticVariable {
    static int count=0;
}

```

```

        StaticVariable() {
            count++;
        }
        public static void main(String[] args)
        {
            //bien dem count se cho biet so doi tuong duoc tao ra
            StaticVariable c1=new StaticVariable();
            System.out.println("Bien dem count="+count);
            StaticVariable c2=new StaticVariable();
            System.out.println("Bien dem count="+count);
            StaticVariable c3=new StaticVariable();
            System.out.println("Bien dem count="+count);
        }
    }

```

Phương thức tĩnh chỉ sử dụng các thuộc tính hoặc phương thức của lớp chính nó thì các thuộc tính, phương thức này phải là tĩnh.

Ví dụ phương thức static

```

class StaticMethodDemo
{
    static String name="Phuong thuc tinh";
    public static void main(){
        System.out.println("Hello" + name);
    }
    public static void main(String[] args)
    {
        main();
        System.out.println("Hello World!");
    }
}

```

Khởi static được sử dụng để khởi tạo giá trị của các biến tĩnh. Khi lớp lần đầu tiên được tải vào bộ nhớ, các khởi static luôn được xử lý trước.

Ví dụ khởi tĩnh

```

class StaticDemo
{

```

```

static{
    System.out.println("Khoi static 1");
}

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

static {
    System.out.println("Khoi static 2");
}
}

```

Vì khối static luôn được xử lý trước nên kết quả in ra của chương trình trên sẽ là:

```

Khoi static 1
Hello World!
Khoi static 2

```

### 2.5.10 Các thành phần hằng (final)

Biến được khai báo với từ khóa final có nghĩa là biến có giá trị hằng, không thay đổi được nội dung.

Ví dụ.

```
final double PI=3.1416;
```

Các phương thức được khai báo là final không thể được viết đè ở lớp thừa kế.

Ví dụ

```

class A {
    final void method(){
    }
}

class B extends A{
    final void method(){ //lỗi
    }
}

```

Lớp khai báo với từ khóa final không thể được thừa kế.

Ví dụ

```
final class A {  
}
```

## 2.6 Lớp trừu tượng

Lớp trừu tượng là lớp có ít nhất một phương thức trừu tượng. Phương thức trừu tượng là phương thức chỉ khai báo khuôn dạng hàm mà không thể cài đặt chi tiết hàm.

Cú pháp khai báo phương thức trừu tượng:

```
[phạm vi] abstract <kieu du lieu> <tenphuongthuc>([ds tham so]);
```

Cú pháp khai báo lớp trừu tượng:

```
abstract class <ten lop>{  
    //có ít nhất một phương thức trừu tượng  
}
```

Ví dụ:

```
abstract class  Hình2D  
{  
    double a,b,r;  
    public abstract double dientich();  
    public abstract double chuvi();  
}  
class HìnhTron extends Hình2D  
{  
    public HìnhTron(double r)  
    {  
        this.r=r;  
    }  
    public double dientich()  
    {  
        return Math.PI*r*r;  
    }  
    public double chuvi()  
    {  
        return Math.PI*2*r;  
    }  
}
```

```

class HìnhChuNhat extends Hình2D
{
    public HìnhChuNhat(double a,double b)
    {
        this.a=a;
        this.b=b;
    }
    public double dientich()
    {
        return a*b;
    }
    public double chuvi()
    {
        return (a+b)*2;
    }
}
class AbstractDemo
{
    public static void main(String args[])
    {
        Hình2D ht=new HìnhTron(1);
        System.out.println("Dien tich hình tron ban kinh 1.0
                               la:"+ht.dientich());
        System.out.println("Chu vi hình tron ban kinh 1.0
                               la:"+ht.chuvi());
        Hình2D hcn=new HìnhChuNhat(3,4);
        System.out.println("Dien tich hình chu nhat
                               la:"+hcn.dientich());
        System.out.println("Chu vi hình chu nhat la
                               "+hcn.chuvi());
    }
};

```

**Kết quả thực hiện chương trình**

```
C:\MyJava>java AbstractDemo
```

```
Diện tích hình tròn bán kính 1.0 là:3.141592653589793
Chu vi hình tròn bán kính 1.0 là:6.283185307179586
Diện tích hình chu nhật là:12.0
Chu vi hình chu nhật là 14.0
```

## 2.7 Giao diện (Interface)

Giao diện là sự trừu tượng hóa cao độ, nghĩa là một lớp đối tượng chỉ bao gồm tập hợp các phương thức trừu tượng hoặc thuộc tính hằng. Giao diện được sử dụng trong cơ chế liên kết muộn. Giao diện cho phép đa thừa kế.

Cú pháp khai báo giao diện

```
public interface <tên giao diện> [extends <tên giao diện cha>]
{
    //Thân giao diện
}
```

Ví dụ định nghĩa một giao diện:

```
public interface CalculatorInterface
{
    public double add(double x, double y);
    public double sub(double x, double y);
    public double mul(double x, double y);
    public double div(double x, double y);
}
```

Cài đặt giao diện

Giao diện trình bày các phương thức chung của các lớp đối tượng cài đặt nó. Lớp đối tượng cài đặt giao diện có cú pháp như sau.

```
class <tên lớp> implements <tên lớp cha>{
    //Cài đặt các phương thức của giao diện
}
```

Ví dụ:

```
class CalculatorTest implements CalculatorInterface
{
    public double add(double x, double y)
    {
```



```

        return x+y;
    }
    public double sub(double x, double y)
    {
        return x-y;
    }
    public double mul(double x, double y)
    {
        return x*y;
    }
    public double div(double x, double y)
    {return x/y;

    }
    public static void main(String[] args) throws Exception
    {
        CalculatorInterface cal=new CalculatorTest();
        if(args.length!=2)
        {
            System.out.println("Cach chay chuong trinh: java
            CalculatorImpl so1 so2");
            return;
        }
        else
        {
            double x,y,z;
            x=Double.parseDouble(args[0]);
            y=Double.parseDouble(args[1]);
            System.out.println(x+" "+y+"="+cal.add(x,y));
            System.out.println(x+"-"+y+"="+cal.sub(x,y));
            System.out.println(x+"*"+y+"="+cal.mul(x,y));
            System.out.println(x+"/"+y+"="+cal.div(x,y));
        }
    }
}

```

Kết quả thực hiện chương trình là

```
C:\MyJava>java CalculatorTest 12 3
12.0+3.0=15.0
12.0-3.0=9.0
12.0*3.0=36.0
12.0/3.0=4.0
```

## 2.8 Gói (Package)

Các gói gồm có các thành phần là các lớp, các giao diện và các gói con có liên quan với nhau. Việc tổ chức thành các gói có một số lợi ích sau:

- Cho phép nhóm các thành phần cùng đặc trưng, chức năng thành một đơn vị, thuận tiện cho việc quản lý.
- Tránh xung đột tên.
- Cho phép qui định phạm vi truy cập các thành phần mở mức gói.

Cú pháp truy cập tới thành phần của gói

<Tên gói>.<Tên thành phần>

Cú pháp nạp các thành phần trong gói vào trong một chương trình.

```
import <Tên gói>.*; //tên gói
```

Cú pháp khai báo nạp các thành phần trong gói con như sau:

```
import <Tên gói>.<Tên gói con>.*;
```

Cách tạo ra các gói trong Java

Cú pháp khai báo để đưa một lớp đối tượng vào một gói

```
package <Tên gói>;
```

Ví dụ:

```
package mypackage;
public class Calculator
{
    public double cong(double a,double b)
    {
        return a+b;
    }
    public double nhan(double a, double b)
    {
```

```

        return a*b;
    }
    public double tru(double a,double b)
    {
        return a-b;
    }
    public double chia(double a,double b) throws Exception
    {
        return a/b;
    }
}

```

Biên dịch

```
C:\>javac -d C:\MyJava Calculator.java
```

*Lưu ý:* các thành phần của gói cần được khai báo với thuộc tính public, nếu cần truy xuất chúng từ bên ngoài.

## 2.9 Quản lý ngoại lệ (Exception Handling)

Khi lập một chương trình thường có các lỗi xảy ra. Trong đó, lỗi trong quá trình thực thi chương trình là một loại lỗi nghiêm trọng làm cho chương trình không hoạt động bình thường hoặc ngừng hoạt động. Các lỗi này là do chương trình không kiểm soát hết các tính huống dữ liệu đầu vào khác nhau, ví dụ như tên tập tin cung cấp không tìm thấy, địa chỉ mạng không hợp lệ, vượt kích thước mảng, ...)

Ngôn ngữ lập trình Java hỗ trợ cơ chế kiểm tra lỗi chặt chẽ. Cụ thể các lớp biểu diễn lỗi như `Throwable`, `Error`, `Exception`,... Các thành phần trong thư viện có sẵn của Java đều được cài đặt kiểm tra lỗi và đưa ra lớp biểu diễn lỗi phù hợp. Ngoài ra, Java còn cho phép tạo ra các lớp đối tượng có kiểm tra lỗi và phát sinh ra đối tượng lỗi tương ứng.

### 2.9.1 Lệnh kiểm soát ngoại lệ cơ bản

Một lệnh kiểm soát ngoại lệ cơ bản gồm 2 khối lệnh chính là khối `try` và khối `catch`. Khối `try` là nơi chứa đoạn lệnh có khả năng gây ra lỗi run-time. Khối `catch` là nơi chứa đoạn lệnh xử lý lỗi do khối `try` phát hiện. Một lệnh kiểm soát ngoại lệ có duy nhất một khối `try` và có thể có nhiều khối `catch`.

Ngoài ra còn khối tùy chọn là **finally**, là nơi chứa các lệnh luôn được thực thi dù có phát sinh exception hay không. Nếu trong khối **catch** có lệnh thoát khỏi phương thức, như **return**, thì lệnh trong khối **finally** vẫn được thực thi bình thường. Các lệnh này thường là những lệnh "dọn dẹp" bộ nhớ, đóng các stream,...

Cú pháp tổng quát:

```
try
{
    // Các câu lệnh có khả năng gây ra exception
}
catch (Exception1 E1)
{
    // Các câu lệnh xử lý cho exception thuộc loại Exception1
}
catch (Exception2 E2)
{
    // Các câu lệnh xử lý cho exception thuộc loại Exception2
}
...
...
finally
{
    // Các câu lệnh luôn được thực thi
}
```

Ví dụ cụ thể: Trong ví dụ này, ta sẽ cố tình phát sinh ra một ngoại lệ, đó là lỗi chia cho 0. Lỗi này chỉ được phát hiện lúc run-time.

```
public class Main01
{
    public static void main(String[] args)
    {
        int x=4,y=0;
        try
        {
            int z=x/y;
```

```

        // Câu lệnh này sẽ không được thực thi
        System.out.println("Hello!");
    }
    catch (Exception E)
    {
        System.out.println("Co loi run-time!");
    }
}
}

```

Và kết quả khi chạy chương trình:

```

Co loi run-time!
Press any key to continue...

```

Nếu không xử lý exception, ta sẽ được kết quả như sau:

```

Exception in thread "main" java.lang.ArithmeticException: / by
zero
    at Main01.main(Main01.java:8)
Press any key to continue...

```

Đó là exception do Java tự phát hiện và xử lý.

Ví dụ trên cho thấy rằng khi một câu lệnh phát sinh exception, các câu lệnh sau nó trong khối try sẽ không được thực thi. Khi đó, con trỏ lệnh sẽ nhảy ngay đến khối **catch** phù hợp (nếu có) để thực thi các lệnh trong đó.

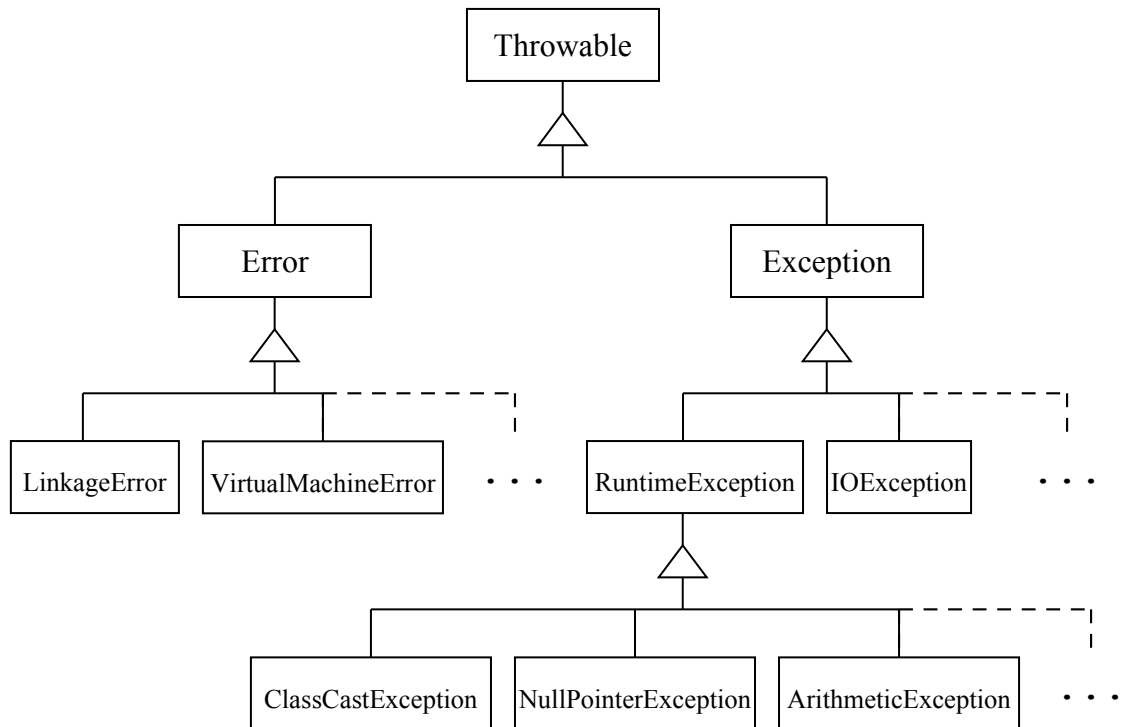
### 2.9.2 Tổ chức lớp biểu diễn ngoại lệ

Trong Java, các ngoại lệ được biểu diễn ở lớp cao nhất là `Throwable`. Kế thừa nó là hai lớp con `Exception` và `Error`. Tất cả các lớp biểu diễn ngoại lệ trong Java đều kế thừa từ hai lớp này.

Một đối tượng phải được dẫn xuất từ `Throwable` mới dùng được trong câu lệnh **throw** để phát sinh ngoại lệ.

Lớp `Exception` dùng để mô tả chung cho tất cả ngoại lệ trong Java, ... Trong đó, `RuntimeException` là một dẫn xuất quan trọng. Từ nó, các ngoại lệ như lỗi vượt phạm vi mảng, lỗi tính toán số học, lỗi ép kiểu, lỗi sử dụng đối tượng chưa khởi tạo,... được cài đặt.

Lớp `Error` dùng để mô tả các chung cho các loại ngoại lệ đặc biệt, không cần xử lý, ví dụ như lỗi của máy ảo, lỗi liên kết class,...



Sơ đồ tổ chức exception của Java

### 2.9.3 Phát sinh ngoại lệ

Các lỗi nếu có phát sinh sẽ được bắt bởi máy ảo Java và ném ra một đối tượng ngoại lệ chứa thông tin lỗi. Ngôn ngữ lập trình Java cho phép người lập trình ném ra đối tượng ngoại lệ bằng từ khóa `throw`.

```
public class Main02
{
    public static void main(String[] args)
    {
        Object o=null;
        if (o==null)
            // Phát sinh exception loại NullPointerException
            throw new NullPointerException();
    }
}
```

Trong đó, `NullPointerException` là một exception có sẵn của Java. Khi chạy chương trình, ta sẽ nhận được thông báo sau:

```
Exception in thread "main" java.lang.NullPointerException
    at Main02.main(Main02.java:8)
```

Để exception được rõ nghĩa, ta có thể throw một exception với hàm khởi tạo có tham số là chuỗi mô tả loại lỗi, sau đó dùng phương thức `getMessage()` để xuất câu thông báo ra màn hình.

```
public class Main02
{
    public static void main(String[] args)
    {
        try
        {
            String S="Lỗi: object có giá trị null";
            // Tạo ra exception bằng constructor có tham số
            throw new NullPointerException(S);
        }
        catch (NullPointerException E)
        {
            System.out.println(E.getMessage());
        }
    }
}
```

Khi đó, câu thông báo lỗi sẽ trở nên thân thiện hơn:

```
Lỗi: object có giá trị null
Press any key to continue...
```

Trong một phương thức có ném ra ngoại lệ thì phương thức đó phải được khai báo ném ngoại lệ bằng từ khóa `throws`.

ví dụ:

```
// Từ khóa throws cho biết phương thức này có phát sinh exception
// cho khối cao hơn xử lý
void methodD(int[]a) throws NullPointerException
```

```

{
    if (a == null)
        // Phát sinh exception
        throw new NullPointerException("Loi: Mang null");
    // ...
}

```

Khi đó, nếu phương thức `methodC()` gọi `methodD()` mà không đặt nó trong khối exception thì cũng phải có từ khóa `throws` trong phần khai báo. Khi đó exception của phương thức `methodC()` phải cùng loại hoặc là lớp cha của exception do `methodD()` phát sinh.

```

void methodC() throws NullPointerException // Hoặc throws
Exception,...
{
    int[] a=null;
    // ...
    methodD(a);
    // ...
}

```

Ví dụ bất nhiều ngoại lệ

```

Scanner sn=new Scanner(System.in);
System.out.print("Nhap gia tri n (0, 1, 2, ...): ");
int n=sn.nextInt();
try
{
    switch (n)
    {
        case 0 : break; // Không phát sinh exception

        case 1 : throw new NullPointerException();
        case 2 : throw new ClassCastException();
        case 3 : throw new ArrayIndexOutOfBoundsException();
        default:
            // Phát sinh exception IllegalArgumentException
            // Lưu ý: không có khối catch riêng cho exception này,
            // việc catch nó sẽ do khối catch Exception thực hiện
    }
}

```



```

        throw new IllegalArgumentException();
    }
}
catch (NullPointerException E)
{
    System.out.println("Loi doi tuong null!");
    // return sẽ ngăn cản việc thực thi khối lệnh ngoài finally,
    // nhưng không ngăn cản việc thực thi khối lệnh finally
    return;
}
catch (ClassCastException E)
{
    System.out.println("Loi ep kieu!");
}
catch (ArrayIndexOutOfBoundsException E)
{
    System.out.println("Loi chi so mang vuot pham vi!");
}
catch (Exception E)
{
    System.out.println("Co loi!");
}
finally
{
    // Khối lệnh này luôn được thực thi
    System.out.println("Khoi lenh finally.");
}
//Lệnh này có khi được thực thi, có khi không, tùy vào cách xử lý
// của các khối catch ở trên
System.out.println("Lenh ngoai khoi finally.");

```

Khi được thực thi, tùy vào giá trị của  $n$  mà đoạn code trên sẽ phát sinh exception tương ứng. Khi đó chỉ có chỉ có khối **catch** phù hợp nhất được thực thi.

Nhập  $n = 0$ : không phát sinh exception nên không thực thi khối catch nào, vẫn thực thi khối finally và lệnh ngoài khối finally.

```
Nhap gia tri n (0, 1, 2, ...): 0
Khoi lenh finally.
Lenh ngoai khoi finally.
Press any key to continue...
```

Nhập `n = 1`: phát sinh `NullPointerException`, khối `catch` tương ứng với exception này chứa lệnh `return` thoát ngay khỏi phương thức nên lệnh ngoài khối `finally` không được thực thi, tuy nhiên lệnh trong khối `finally` vẫn được thực thi bình thường.

```
Nhap gia tri n (0, 1, 2, ...): 1
Loi doi tuong null!
Khoi lenh finally.
Press any key to continue...
```

Nhập `n = 2`: phát sinh `ClassCastException`, khối `catch` tương ứng với exception này không thoát khỏi phương thức nên lệnh ngoài khối `finally` vẫn được thực thi, và dĩ nhiên lệnh trong khối `finally` vẫn được thực thi bình thường.

```
Nhap gia tri n (0, 1, 2, ...): 2
Loi ep kieu!
Khoi lenh finally.
Lenh ngoai khoi finally.
Press any key to continue...
```

Mỗi loại exception sẽ ứng với một đối tượng exception trong Java. Nếu ta "catch" một exception thì ta đã "catch" luôn các exception kế thừa từ nó. Nên cần lưu ý rằng Java không cho phép ta "catch" exception "cha" trước khi "catch" exception "con", vì như vậy thì khối lệnh `catch` của exception "con" sẽ không bao giờ được thực thi.

Trong khối `catch` cuối cùng, ta bắt lỗi với loại lỗi là `Exception`. Do mọi exception thông thường trong Java đều kế thừa từ class `Exception` nên khối `catch` này có khả năng bắt gần như mọi loại exception của Java. Do đó, như đã nói ở trên, khối `catch` của nó phải được đặt sau cùng (nếu có).

Dĩ nhiên là nếu muốn catch mọi loại exception trong Java, ta có thể `catch` với loại lỗi là `Throwable`:

```
catch (Throwable T)
{
```

```
        System.out.println("Co loi trong Java!");  
    }
```

#### 2.9.4 Sự lan truyền ngoại lệ

Khi một exception được phát sinh, nó sẽ lan truyền dần từ nơi phát sinh ra các cấp cao hơn cho tới khi được catch hoặc tới phương thức `main()`. Khi exception truyền tới `main()` mà vẫn không được catch, nó sẽ được thông báo cho người dùng.

Xem ví dụ sau:

```
public class Main04  
{  
    public static void main(String[] args)  
    {  
        methodA(null);  
    }  
    static void methodA(int[] a)  
    {  
        methodB(a);  
    }  
    static void methodB(int[] b)  
    {  
        System.out.println(b[0]);  
    }  
}
```

Và kết quả khi chạy ví dụ trên:

```
Exception in thread "main" java.lang.NullPointerException  
    at Main04.methodB(Main04.java:13)  
    at Main04.methodA(Main04.java:9)  
    at Main04.main(Main04.java:5)  
Press any key to continue...
```

Dựa vào thông báo, ta có thể thấy exception `NullPointerException` được phát sinh từ `methodB()`, sau đó được lan truyền về `methodA()` và sau cùng là `main()`.

### 2.9.5 Các phương thức chính của lớp Exception

Các phương thức dưới đây đều kế thừa từ Throwable.

<code>Exception()</code>	Khởi tạo một Exception mặc định, câu thông báo sẽ là <b>null</b> .
<code>Exception(String msg)</code>	Khởi tạo một Exception với câu thông báo <code>msg</code>
<code>String getMessage()</code>	Lấy câu thông báo của Exception.
<b>void</b> <code>printStackTrace()</code>	In ra stack lan truyền của Exception.
<code>Throwable</code> <code>fillInStackTrace()</code>	Ghi thông tin của Exception vào stack lan truyền, sau đó trả về chính Exception <b>this</b>
<code>StackTraceElement[]</code> <code>getStackTrace()</code>	Lấy ra mảng chứa các <code>StackTraceElement</code> , mỗi <code>StackTraceElement</code> là một đối tượng chứa thông tin về Exception được lan truyền (gồm dòng, file, lớp, phương thức gây lỗi).

Ví dụ:

```
public static void main(String[] args)
{
    StackTraceElement[] ste=null;
    try
    {
        methodA();
    }
    catch (Exception E)
    {
        // In ra câu thông báo lỗi
        System.out.println("Co loi: "+E.getMessage());
        // Lấy ra stack lưu thông tin về sự lan truyền
        // exception
        ste=E.getStackTrace();
    }
    for(int i=0; ste!=null && i<ste.length; i++)
        System.out.println(
            " Noi gay loi: Tap tin " + ste[i].getFileName() +
            " Dong " + ste[i].getLineNumber() +
            " Class " + ste[i].getClassName() +
```

```

        " Phương thức: " + ste[i].getMethodName()
        );
    }
    static void methodA() throws Exception
    {
        methodB();
    }
    static void methodB() throws Exception
    {
        throw new ArrayIndexOutOfBoundsException("Loi chi so mang.");
    }
}

```

Kết quả chạy đoạn ví dụ trên:

```

Co loi: Loi chi so mang.
  Noi gay loi: Tap tin Main05.java Dong 34 Class Main05 Phương
thức: methodB
  Noi gay loi: Tap tin Main05.java Dong 30 Class Main05 Phương
thức: methodA
  Noi gay loi: Tap tin Main05.java Dong 11 Class Main05 Phương
thức: main
Press any key to continue...

```

### 2.9.6 Lớp ngoại lệ tự định nghĩa

Ngoài việc dùng các lớp đối tượng ngoại lệ có sẵn của Java, ta có thể tự tạo các exception cho phù hợp với chương trình của mình. Để tạo một lớp ngoại lệ, ta chỉ cần tạo một class kế thừa từ Throwable (hoặc kế thừa từ các lớp con của Throwable), sau đó cài đặt các phương thức phù hợp.

Ví dụ: cài đặt exception `LoiDiemSo` cho điểm trung bình của học sinh: nếu nhập điểm nhỏ hơn 0 hoặc lớn hơn 10 thì chương trình sẽ phát sinh exception này.

```

// Khai báo exception LoiDiemSo
// Exception này sẽ được phát sinh khi nhập điểm không hợp lệ
public class LoiDiemSo extends Exception
{
    public LoiDiemSo()

```

```

    {
        super("Diem phai tu 0 toi 10");
    }
    public LoiDiemSo(String s)
    {
        super(s);
    }
}

```

Sau khi đã có lớp ngoại lệ `LoiDiemSo`, ta có thể dùng như sau:

```

// ...
public void Nhap() throws LoiDiemSo
{
    Scanner sn=new Scanner(System.in);

    System.out.print(" + Nhap ten hoc sinh: ");
    HoTen=sn.nextLine();

    System.out.print(" + Nhap dien trung binh : ");
    DiemTrungBinh=sn.nextFloat();

    // Neu diem khong hop le thi phat sinh exception LoiDiemSo
    if (DiemTrungBinh<0 || DiemTrungBinh>10)
        throw new LoiDiemSo();
}

```

Trong phương thức `main`, khi nhập học sinh, ta sẽ bắt lỗi và xuất ra câu thông báo:

```

public static void main(String[] args)
{
    HocSinh hs=new HocSinh();
    try
    {
        hs.Nhap();
    }
    catch (LoiDiemSo L)
    {

```

```

        System.out.println(L.getMessage());
        return;
    }
    System.out.println(hs);
}

```

### 2.9.7 Overriding với exception

Khi một lớp con override phương thức của lớp cha, nếu lớp cha có ném ra lỗi E (qua từ khóa throws) thì lớp con cũng phải ném ra lỗi E hoặc lỗi con của E.

```

class LopCha
{
    public void Method1() throws Exception
    {
    }

    public void Method2() throws RuntimeException
    {
    }

    public void Method3() throws ClassCastException
    {
    }
}

class LopCon extends LopCha
{
    // OK, vì NullPointerException là con của Exception
    public void Method1() throws NullPointerException
    {
    }

    // OK, vì cha và con ném ra cùng loại lỗi là RuntimeException
    public void Method2() throws RuntimeException
    {
    }

    // Phương thức này không hợp lệ
}

```

```
// vì Exception là cha của ClassCastException
/*
public void Method3()throws Exception
{
}
*/
}
```

## 2.10 Bài tập

3. Cần quản lý một danh sách nhân viên của 1 cơ quan gồm 2 loại: Nhân viên biên chế và nhân viên hợp đồng.

- Các thông tin chung: họ tên, phòng.
- NVBC: hệ số lương, số năm CT.
- NVHD: lương hợp đồng, Loại HĐ(NH,DH).

Các thao tác trên danh sách nhân viên:

- Tạo lập và lưu các nhân viên
- Liệt kê danh sách nhân viên.
- Liệt kê danh sách nhân viên theo loại: HĐ, BC
- Tính tổng lương toàn bộ nhân viên.
- Liệt kê danh sách nhân viên hợp đồng dài hạn.

4. Xây dựng các lớp tính diện tích các hình: tròn, tam giác, chữ nhật. Chương trình minh họa gồm một mảng các đối tượng và tính tổng diện tích của các hình trong mảng.

5. Một thư viện gồm các loại tài liệu sau:

- Sách(Mã sách, Tên Sách, Tác giả, NXB, Năm XB, Vị trí)
- Tạp chí (Mã tạp chí, Tên tạp chí, Chuyên ngành, Số, Năm, Vị trí)
- CD(Mã CD, Tên CD, Số thứ tự, Nội dung, Vị trí)

Hãy tổ chức các lớp sao cho có thể lập trình để thực hiện được các chức năng sau:

- Lưu danh sách các tài liệu có trong thư viện.
- Liệt kê toàn bộ tài liệu có trong thư viện.
- Liệt kê từng loại tài liệu có trong thư viện.
- Xem thông tin về tài liệu khi biết mã tài liệu.



- Tìm kiếm một tài liệu theo: Tên và tác giả đối với sách; Tên tạp chí, Chuyên ngành, số, năm đối với tạp chí, Tên CD, Số thứ tự và nội dung đối với CD.

## CHƯƠNG 3 . QUẢN LÝ CÁC LUỒNG VÀO RA

### 3.1 Giới thiệu

Một chương trình máy tính cần phải lấy dữ liệu từ bên ngoài vào chương trình để xử lý. Sau khi chương trình xử lý xong sẽ xuất kết quả ra ngoài cho thành phần khác để tiếp tục xử lý hoặc lưu trữ. Dữ liệu được truyền từ chương trình này sang chương trình khác bằng các luồng dữ liệu. Để cài đặt việc truyền dữ liệu này, Java cung cấp package `java.io` với rất nhiều lớp và các tính năng phong phú.

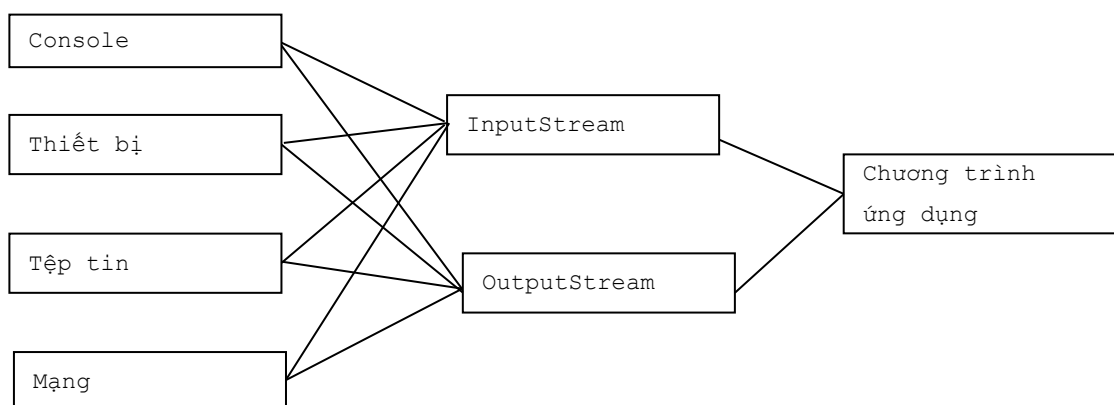
Xét theo loại dữ liệu, Java chia các luồng thành hai loại là luồng byte (byte stream) và luồng ký tự (character stream). Giữa hai loại dữ liệu này có các lớp trung gian để phục vụ cho việc chuyển đổi.

Xét theo thao tác, Java chia các luồng thành hai loại là input (đọc byte) và output (ghi byte) hoặc reader (đọc character) và writer (ghi character).

	Byte streams	Character streams
Source streams	InputStream	Reader
Sink streams	OutputStream	Writer

Các stream cơ bản trong Java

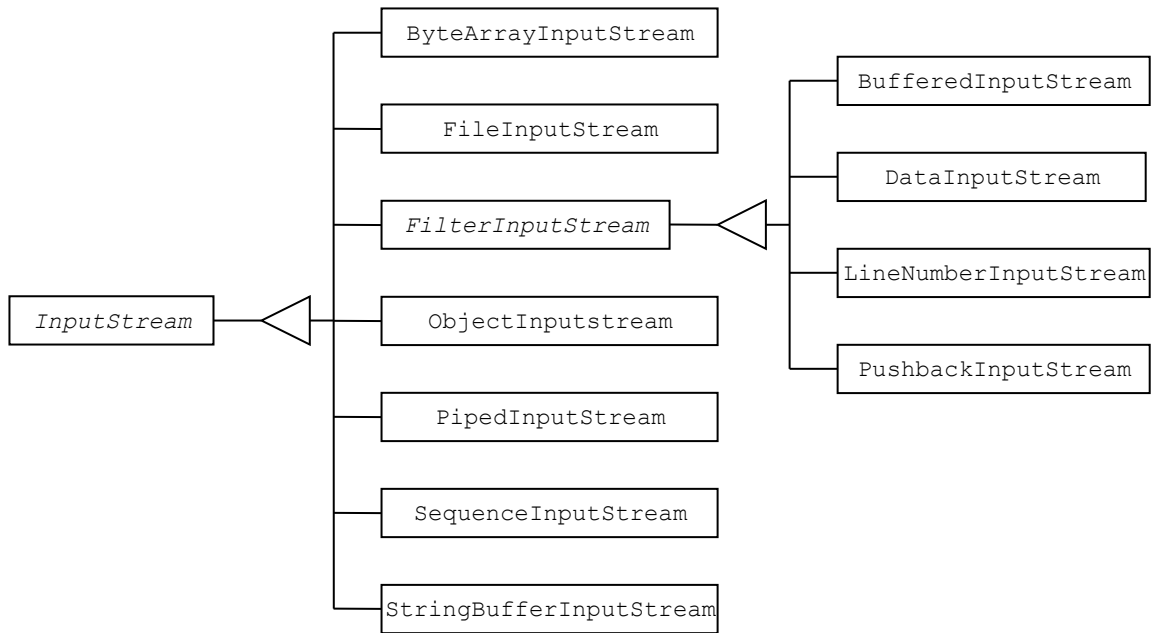
Ngoài ra, Java còn hỗ trợ rất nhiều luồng hỗ trợ cho việc đọc ghi khác.



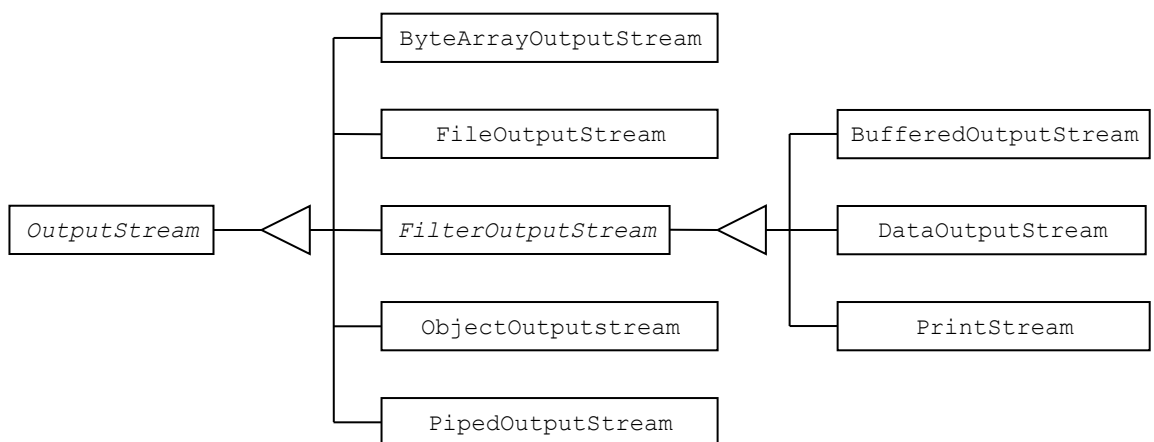
Minh họa các nguồn dữ liệu

### 3.2 Các luồng Byte

Các luồng byte có chức năng đọc hoặc ghi dữ liệu dạng byte. Dựa vào thao tác, các luồng byte được chia ra hai loại chính: nhóm input được đại diện bởi lớp `InputStream` và nhóm output được đại diện bởi `OutputStream`. Các lớp xử lý trên luồng byte đều được kế thừa từ hai lớp này.



Sơ đồ tổ chức của các stream đọc byte



Sơ đồ tổ chức của các stream ghi byte

### 3.2.1 Các luồng byte tổng quát

Gồm các lớp `InputStream` và `OutputStream`. Đây hai là lớp trừu tượng, nó quy định các phương thức đọc và ghi dữ liệu dạng byte và một số phương thức hỗ trợ khác:

Phương thức	Ý nghĩa
<b>Lớp <code>InputStream</code></b>	
<code>int available()</code>	Trả về số byte còn lại trong stream.
<code>void close()</code>	Đóng stream.
<code>void mark(int readlimit)</code>	Đánh dấu vị trí hiện tại. Nếu sau khi mark ta đọc quá <code>readlimit</code> byte thì chỗ đánh dấu không còn hiệu lực.
<code>boolean markSupported()</code>	Trả về <code>true</code> nếu stream hỗ trợ mark và reset.
<code>abstract int read()</code>	Đọc byte kế tiếp trong stream.  Quy ước: Trả về -1 nếu hết stream.  Đây là phương thức trừu tượng.
<code>int read(byte[] buffer)</code>	Đọc một dãy byte và lưu kết quả trong mảng <code>buffer</code> . Số byte đọc được tối đa là sức chứa ( <code>length</code> ) của <code>buffer</code> . Nếu <code>buffer</code> có sức chứa là 0 thì sẽ không đọc được gì.  Sau khi đọc xong sẽ trả về số byte đọc được thật sự (nếu đang đọc mà hết stream thì số byte đọc được thật sự sẽ nhỏ hơn sức chứa của <code>buffer</code> ).  Nếu stream đã hết mà vẫn đọc nữa thì kết quả trả về là -1.
<code>int read(byte[] buffer, int offset, int len)</code>	Đọc một dãy byte và lưu kết quả trong mảng <code>buffer</code> kể từ byte thứ <code>offset</code> . Số byte đọc được tối đa là <code>len</code> . Nếu <code>len</code> là 0 thì sẽ không đọc được gì.  Sau khi đọc xong sẽ trả về số byte đọc được thật sự (nếu đang đọc mà hết stream thì số byte đọc được thật sự sẽ nhỏ hơn <code>len</code> ).  Nếu stream đã hết mà vẫn đọc nữa thì kết quả trả về là

	-1.
<code>void reset()</code>	Trở lại chỗ đã đánh dấu bằng phương thức <code>mark()</code> .
<code>long skip(long n)</code>	Bỏ qua <code>n</code> byte (để đọc các byte tiếp sau <code>n</code> byte đó).
<b>Lớp <code>OutputStream</code></b>	
<code>void close()</code>	Đóng stream.
<code>void flush()</code>	Buộc stream ghi hết dữ liệu trong vùng đệm ra ngoài.
<code>void write(byte[] buffer)</code>	Ghi một dãy byte vào stream. Số byte được ghi sẽ là <code>buffer.length</code> .
<code>int write(byte[] buffer, int offset, int len)</code>	Ghi một dãy byte vào stream. Bắt đầu ghi từ byte thứ <code>offset</code> , và ghi tổng cộng <code>len</code> byte.
<code>abstract void write(int b)</code>	Ghi một byte vào stream.  Đây là phương thức trừu tượng.

Các phương thức cơ bản của các byte stream tổng quát

Lưu ý: Các phương thức đọc/ghi sẽ phát sinh `IOException` nếu có lỗi xảy ra.

### 3.2.2 Các luồng đọc byte hiện thực

Để sử dụng đọc/ghi luồng dữ liệu dạng byte, các lớp con hiện thực của `InputStream` và `OutputStream` thường được sử dụng gồm:

- `FileInputStream`: đọc (các) byte từ tập tin.
- `FileOutputStream`: ghi (các) byte vào tập tin.
- `ByteArrayInputStream`: chứa bộ đệm là mảng byte để đọc dữ liệu.
- `ByteArrayOutputStream`: chứa bộ đệm là mảng byte để ghi dữ liệu.
- `PipedInputStream`: đọc (các) byte từ một piped output stream.
- `PipedOutputStream`: ghi (các) byte vào một piped input stream.

Nhìn chung các lớp này đều có những chức năng chính tương tự nhau. Dưới đây chỉ trình bày những phương thức đặc thù của chúng.

Stream	Phương thức	Ý nghĩa
<b>File Input Stream</b>	<code>FileInputStream(File file)</code>	Tạo <code>FileInputStream</code> để đọc dữ liệu từ một tập tin liên kết tới đối tượng <code>File</code> .
	<code>FileInputStream(String name)</code>	Tạo <code>FileInputStream</code> để đọc dữ liệu từ

		một tập tin có tên là <code>name</code> .
<b>File Output Stream</b>	<code>FileOutputStream(File file)</code>	Tạo <code>FileOutputStream</code> để ghi dữ liệu vào một tập tin liên kết tới đối tượng <code>File</code> .
	<code>FileOutputStream(File file, boolean append)</code>	Tạo <code>FileOutputStream</code> để ghi dữ liệu tập tin liên kết tới đối tượng <code>File</code> . Nếu <code>append</code> là <code>true</code> thì sẽ ghi tiếp vào cuối tập tin, ngược lại thì ghi đè lên tập tin.
	<code>FileOutputStream(String name)</code>	Tạo <code>FileOutputStream</code> để ghi dữ liệu vào một tập tin có tên là <code>name</code> .
	<code>FileOutputStream(String name, boolean append)</code>	Tạo <code>FileOutputStream</code> để ghi dữ liệu vào một tập tin có tên là <code>name</code> . Nếu <code>append</code> là <code>true</code> thì sẽ ghi tiếp vào cuối tập tin, ngược lại thì ghi đè lên tập tin.
<b>Byte Array Input Stream</b>	<code>ByteArrayInputStream(byte[] buf)</code>	Tạo ra <code>ByteArrayInputStream</code> và dùng <code>buf</code> để làm vùng đệm.
	<code>ByteArrayInputStream(byte[] buf, int off, int len)</code>	Tạo ra <code>ByteArrayInputStream</code> và dùng một phần của <code>buf</code> để làm vùng đệm.
<b>Byte Array Output Stream</b>	<code>ByteArrayOutputStream()</code>	Tạo ra một <code>ByteArrayOutputStream</code> với vùng đệm 32 byte và có thể tăng nếu cần.
	<code>ByteArrayOutputStream(int size)</code>	Tạo ra một <code>ByteArrayOutputStream</code> với vùng đệm <code>size</code> byte.
	<code>byte[] toByteArray()</code>	Tạo ra mảng byte là bản sao của vùng đệm của <code>this</code> .
	<code>String toString()</code>	Tạo ra chuỗi là bản sao của vùng đệm của <code>this</code> với các byte được đổi thành ký tự tương ứng.
	<code>void writeTo(OutputStream out)</code>	Ghi dữ liệu trong vùng đệm vào một

		output stream khác.
<b>Piped Input Stream</b>	<code>PipedInputStream()</code>	Tạo ra một piped input stream. Stream này chưa được kết nối với piped output stream nào.
	<code>PipedInputStream (PipedOutputStream src)</code>	Tạo ra một piped input stream kết nối với piped output stream <code>src</code> .
	<code>connect (PipedOutputStream src)</code>	Kết nối tới piped output stream <code>src</code> .
<b>Piped Output Stream</b>	<code>PipedOutputStream()</code>	Tạo ra một piped output stream. Stream này chưa được kết nối với piped input stream nào.
	<code>PipedOutputStream (PipedInputStream snk)</code>	Tạo ra một piped output stream kết nối với piped input stream <code>snk</code> .
	<code>connect (PipedInputStream snk)</code>	Kết nối tới piped input stream <code>snk</code> .

Bảng 8: Những phương thức đặc trưng của các byte stream cụ thể

### 3.2.3 Các ví dụ

Để dùng các stream, trước hết cần import package `java.io`.

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

Dưới đây sẽ trình bày các ví dụ về file stream và piped stream.

File stream:

```
String fileName="C:\\TestB.txt";
// Phát sinh và xuất mảng ngẫu nhiên
byte[]a=new byte[10];
System.out.print("Du lieu phat sinh: ");
for(int i=0;i<a.length;i++)
{
    a[i]=(byte)Math.round(Math.random()*20);
    System.out.print(a[i]+" ");
}
System.out.println();
// Ghi mảng a vào tập tin
```

```

FileOutputStream fo=new FileOutputStream(fileName);
fo.write(a);
fo.close();

// Đọc tập tin vào mảng b
FileInputStream fi=new FileInputStream (fileName);
byte[]b= new byte[fi.available()];
fi.read(b);
fi.close();
// Xuất mảng b
System.out.print("Du lieu doc duoc : ");
for(int i=0;i<b.length;i++)
{
    System.out.print(b[i]+" ");
}
System.out.println();

```

Kết quả thực thi (các con số sẽ khác nhau với mỗi lần chạy):

```

Du lieu phat sinh: 4 14 0 16 2 19 10 9 9 15
Du lieu doc duoc : 4 14 0 16 2 19 10 9 9 15
Press any key to continue...

```

Ví dụ về luồng mảng: Tạo một mảng gồm 100 byte rồi gắn vào mảng này một luồng `ByteArrayInputStream` để lấy dữ liệu ra.

```

import java.io.*;
public class LuongNhapMang
{
    public static void main(String[] args)
    {
        byte[] b = new byte[100];
        for(byte i=0;i<b.length;i++) b[i]=i;
        try{
            InputStream is = new ByteArrayInputStream(b);
            for(byte i=0;i<b.length;i++)
                System.out.print(is.read()+" ");
        }
    }
}

```



```

        catch(IOException e)
        {
            System.err.println(e);
        }
    }
}

```

### Kết quả thực hiện chương trình

```

C:\MyJava\Baitap>java LuongNhapMang
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68
69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99

```

Ví dụ luồng viết mảng: Viết chương trình tạo lập một luồng xuất mảng (ByteArrayOutputStream) 100 byte. Ghi vào luồng xuất mảng 100 phần tử từ 0 đến 99. Đổ dữ liệu từ luồng xuất mảng vào mảng b. In dữ liệu từ mảng b ra màn hình.

```

import java.io.*;
class LuongXuatMang
{
    public static void main(String[] args)
    {
        try{
            //Tao mot luong xuat mang 100 byte
            ByteArrayOutputStream os = new ByteArrayOutputStream(100);
            //Ghi du lieu vao luong
            for(byte i=0;i<100;i++) os.write(i);
            //Doc du lieu tu luong vao mang
            byte[] b = os.toByteArray();
            for(byte i=0;i<100;i++) System.out.print(b[i]+" ");
            os.close();
        }catch(IOException e)
        {
            System.err.println(e);
        }
    }
}

```

```
}  
}
```

Kết quả thực hiện chương trình:

```
C:\MyJava\Baitap>java LuongXuatMang  
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24  
25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46  
47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68  
69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90  
91 92 93 94 95 96 97 98 99
```

Ví dụ về Piped stream:

```
PipedInputStream pi= new PipedInputStream();  
PipedOutputStream po= new PipedOutputStream(pi);  
  
// Ghi các số nguyên ngẫu nhiên bằng PipedOutputStream  
System.out.print("Du lieu ghi duoc: ");  
for(int i=0;i<10;i++)  
{  
    int b=(int)Math.round(Math.random()*20);  
    po.write(b);  
    System.out.print(b+" ");  
}  
po.flush();  
System.out.println();  
  
// Đọc các số nguyên từ PipedOutputStream bằng PipedInputStream  
System.out.print("Du lieu doc duoc: ");  
while (pi.available()>0)  
{  
    int b=pi.read();  
    System.out.print(b+" ");  
}  
System.out.println();  
  
// Đóng các stream
```

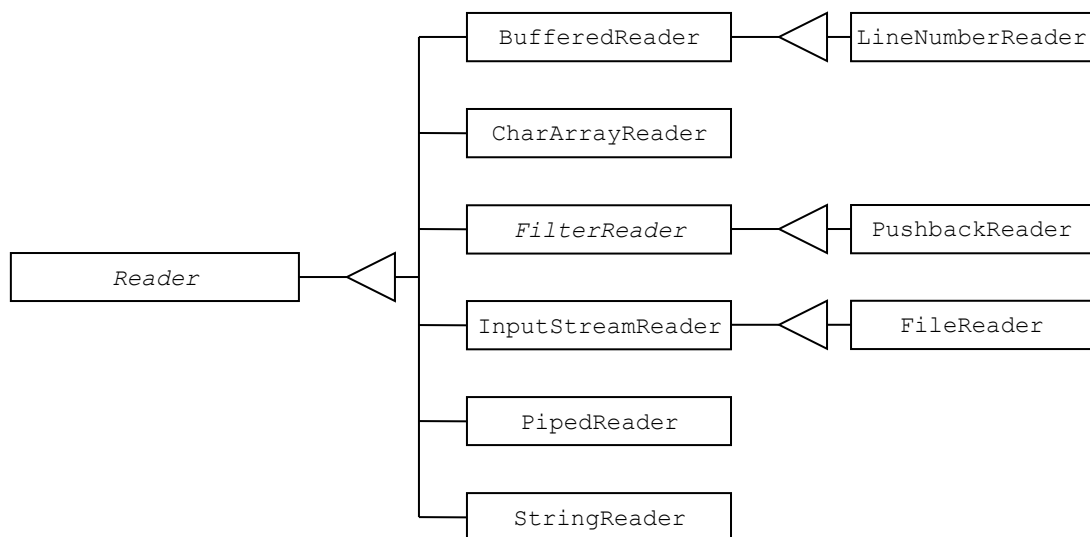
```
pi.close();  
po.close();
```

Kết quả thực thi (các con số sẽ khác nhau với mỗi lần chạy):

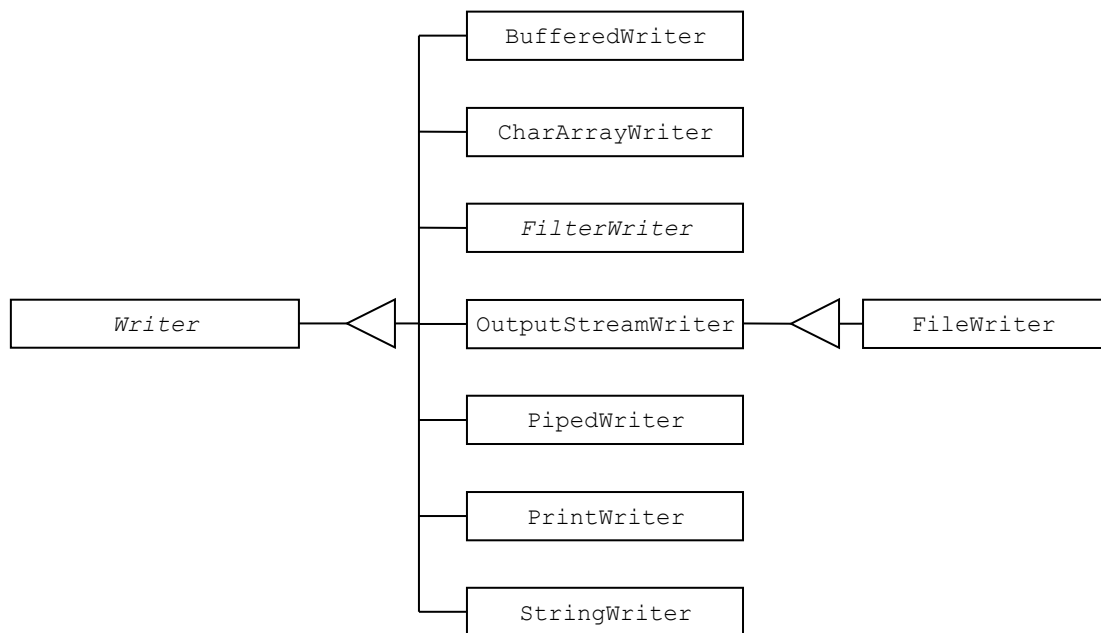
```
Du lieu ghi duoc: 16 19 9 11 19 3 2 13 18 16  
Du lieu doc duoc: 16 19 9 11 19 3 2 13 18 16  
Press any key to continue...
```

### 3.3 Các luồng ký tự

Các luồng ký tự (character stream) có chức năng đọc hoặc ghi dữ liệu dạng ký tự. Dựa vào thao tác, các character stream được chia ra hai loại chính: nhóm input được đại diện bởi lớp `Reader` và nhóm output được đại diện bởi `Writer`. Các lớp xử lý trên character stream đều được kế thừa từ hai lớp này.



Hình 4: Sơ đồ tổ chức của các stream đọc ký tự



Hình 5: Sơ đồ tổ chức của các stream ghi ký tự

### 3.3.1 Các luồng ký tự tổng quát

Gồm các lớp `Reader` và `Writer`. Đây hai là lớp trừu tượng, nó quy định các phương thức đọc và ghi dữ liệu dạng character và một số phương thức hỗ trợ khác:

Phương thức	Ý nghĩa
Lớp <code>Reader</code>	
<code>void close()</code>	Đóng stream.
<code>void mark(int readlimit)</code>	Đánh dấu vị trí hiện tại. Nếu sau khi đánh dấu ta đọc quá <code>readlimit</code> ký tự thì chỗ đánh dấu không còn hiệu lực.
<code>boolean markSupported()</code>	Trả về <code>true</code> nếu stream hỗ trợ <code>mark</code> và <code>reset</code> .
<code>abstract int read()</code>	Đọc ký tự kế tiếp trong stream. Quy ước: Trả về -1 nếu hết stream. Đây là phương thức trừu tượng.
<code>int read(char[] cbuf)</code>	Đọc một dãy ký tự và lưu kết quả trong mảng <code>cbuf</code> .  Sau khi đọc xong sẽ trả về số ký tự đọc được thật sự (nếu đang đọc mà hết stream thì số ký tự đọc được thật

	<p>sự sẽ nhỏ hơn sức chứa của cbuf).</p> <p>Nếu stream đã hết mà vẫn đọc nữa thì kết quả trả về là -1.</p>
<pre>int read(char[] cbuf, int offset, int len)</pre>	<p>Đọc một dãy ký tự và lưu kết quả trong mảng cbuf kể từ ký tự thứ offset.</p> <p>Sau khi đọc xong sẽ trả về số ký tự đọc được thật sự (nếu đang đọc mà hết stream thì số ký tự đọc được thật sự sẽ nhỏ hơn len).</p> <p>Nếu stream đã hết mà vẫn đọc nữa thì kết quả trả về là -1.</p>
<pre>void read(CharBuffer target)</pre>	<p>Đọc một dãy ký tự và lưu kết quả trong target.</p>
<pre>boolean ready()</pre>	<p>Trả về true nếu stream sẵn sàng để đọc.</p>
<pre>void reset()</pre>	<p>Trở lại chỗ đã đánh dấu bằng phương thức mark().</p>
<pre>long skip(long n)</pre>	<p>Bỏ qua n ký tự (để đọc các ký tự tiếp sau).</p>
<b>Lớp writer</b>	
<pre>Writer append(char c)</pre>	<p>Nối đuôi ký tự c vào stream.</p>
<pre>Writer append(CharSequence csq)</pre>	<p>Nối đuôi dãy ký tự csq vào stream.</p>
<pre>Writer append(CharSequence csq, int start, int end)</pre>	<p>Nối đuôi một phần của dãy ký tự csq vào stream.</p>
<pre>void close()</pre>	<p>Đóng stream.</p>
<pre>void flush()</pre>	<p>Buộc stream ghi hết dữ liệu trong vùng đệm ra ngoài.</p>
<pre>void write(char[] cbuf)</pre>	<p>Ghi một mảng ký tự vào stream. Số ký tự được ghi sẽ là cbuffer.length.</p>
<pre>abstract void write(int c)</pre>	<p>Ghi một ký tự vào stream.</p> <p>Đây là phương thức trừu tượng.</p>
<pre>void write(String c)</pre>	<p>Ghi một chuỗi vào stream.</p>
<pre>void write(String str, int off, int len)</pre>	<p>Ghi một phần của chuỗi vào stream.</p>

### Các phương thức cơ bản của character stream tổng quát

Lưu ý: Các phương thức đọc/ghi sẽ phát sinh `IOException` nếu có lỗi xảy ra.

#### 3.3.2 Các luồng ký tự hiện thực

Để sử dụng đọc/ghi ký tự, ta phải dùng các lớp con của `Reader` và `Writer`. Các lớp thường dùng gồm:

- `FileReader`: đọc (các) ký tự từ tập tin.
- `FileWriter`: ghi (các) ký tự vào tập tin.
- `CharArrayReader`: chứa bộ đệm là mảng ký tự để đọc dữ liệu.
- `CharArrayWriter`: chứa bộ đệm là mảng ký tự để ghi dữ liệu.
- `PipedReader`: đọc (các) ký tự từ một piped writer.
- `PipedWriter`: ghi (các) ký tự vào một piped reader.
- `StringReader`: đọc chuỗi ký tự.
- `StringWriter`: ghi chuỗi ký tự.

Nhìn chung các lớp này đều có những chức năng chính tương tự nhau. Dưới đây chỉ trình bày những phương thức đặc thù của chúng.

Stream	Phương thức	Ý nghĩa
<b>File Reader</b>	<code>FileReader(File file)</code>	Tạo <code>FileReader</code> để đọc dữ liệu từ một tập tin liên kết tới đối tượng <code>File</code> .
	<code>FileReader(String name)</code>	Tạo <code>FileReader</code> để đọc dữ liệu từ một tập tin có tên là <code>name</code> .
<b>File Writer</b>	<code>FileWriter(File file)</code>	Tạo <code>FileWriter</code> để ghi dữ liệu vào một tập tin liên kết tới đối tượng <code>File</code> .
	<code>FileWriter(File file, boolean append)</code>	Tạo <code>FileWriter</code> để ghi dữ liệu tập tin liên kết tới đối tượng <code>File</code> . Nếu <code>append</code> là <code>true</code> thì sẽ ghi tiếp vào cuối tập tin, ngược lại thì ghi đè lên tập tin.
	<code>FileWriter(String name)</code>	Tạo <code>FileWriter</code> để ghi dữ liệu vào một tập tin có tên là <code>name</code> .
	<code>FileWriter(String name, boolean append)</code>	Tạo <code>FileWriter</code> để ghi dữ liệu vào một

		tập tin có tên là <code>name</code> .  Nếu <code>append</code> là <code>true</code> thì sẽ ghi tiếp vào cuối tập tin, ngược lại thì ghi đè lên tập tin.
<b>Char Array Reader</b>	<code>CharArrayReader(char[] buf)</code>	Tạo ra <code>CharArrayReader</code> và dùng <code>buf</code> để làm vùng đệm.
	<code>CharArrayReader(char[] buf, int off, int len)</code>	Tạo ra <code>CharArrayReader</code> và dùng một phần của <code>buf</code> để làm vùng đệm.
<b>Char Array Writer</b>	<code>CharArrayWriter()</code>	Tạo ra một <code>CharArrayWriter</code> .
	<code>CharArrayWriter(int size)</code>	Tạo ra một <code>CharArrayWriter</code> với vùng đệm <code>size</code> ký tự.
	<code>char[]toCharArray()</code>	Tạo ra mảng ký tự là bản sao của vùng đệm của <code>this</code> .
	<code>String toString()</code>	Tạo ra chuỗi là bản sao của vùng đệm của <code>this</code> với các byte được đổi thành ký tự tương ứng.
	<code>void writeTo(Writer out)</code>	Ghi dữ liệu trong vùng đệm vào một writer khác.
<b>Piped Reader</b>	<code>PipedReader()</code>	Tạo ra một piped reader. Stream này chưa được kết nối với piped output stream nào.
	<code>PipedReader(PipedWriter src)</code>	Tạo ra một piped reader kết nối với piped output stream <code>src</code> .
	<code>connect(PipedWriter src)</code>	Kết nối tới piped writer <code>src</code> .
<b>Piped Writer</b>	<code>PipedWriter()</code>	Tạo ra một piped writer. Stream này chưa được kết nối với piped input stream nào.
	<code>PipedWriter(PipedReader snk)</code>	Tạo ra một piped writer kết nối với piped reader <code>snk</code> .
	<code>connect(PipedReader snk)</code>	Kết nối tới piped reader <code>snk</code> .
<b>String Reader</b>	<code>StringReader(String s)</code>	Tạo ra một <code>StringReader</code> .
<b>String Writer</b>	<code>StringWriter()</code>	Tạo ra một <code>StringReader</code> , dùng vùng đệm có kích thước mặc định.
	<code>StringWriter(int)</code>	Tạo ra một <code>StringReader</code> , dùng vùng đệm

	<code>initialSize)</code>	có kích thước là <code>initialSize</code> .
	<code>StringBuffer getBuffer()</code>	Trả về vùng đệm của stream.

Bảng 9: Những phương thức đặc trưng của các character stream cụ thể

### 3.3.3 Các ví dụ

Dùng file reader và file writer:

```
String fileName="C:\\TestC.txt";
String s="Hello File Reader/Writer!";
System.out.println("Du lieu ban dau : "+s);
// Ghi s vào tập tin
FileWriter fw=new FileWriter(fileName);
fw.write(s);
fw.close();
// Đọc tập tin vào chuỗi sb
FileReader fr=new FileReader(fileName);
StringBuffer sb=new StringBuffer();
char ca[]=new char[5];           // Đọc mỗi lần tối đa 5 ký tự
while (fr.ready())
{
    int len=fr.read(ca);         // len: số ký tự đọc được thật sự
    sb.append(ca,0,len);
}
fr.close();
// Xuất chuỗi sb
System.out.println("Du lieu doc duoc : "+sb);
```

Kết quả thực thi:

```
Du lieu ban dau : Hello File Reader/Writer!
Du lieu doc duoc : Hello File Reader/Writer!
Press any key to continue...
```

Dùng piped reader và piped writer:

```
PipedReader pr=new PipedReader();
PipedWriter pw=new PipedWriter(pr);
// PipedWriter gửi dữ liệu
```



```
String s="Hello Piped Reader/Writer!";
System.out.println("Du lieu ghi      : "+s);
pw.write(s);
pw.close();

// PipedReader nhận dữ liệu
StringBuffer sb=new StringBuffer();
char ca[]=new char[5];           // Đọc mỗi lần tối đa 5 ký tự
while (pr.ready())
{
    int len=pr.read(ca);         // len: số ký tự đọc được thật sự
    sb.append(ca,0,len);
}
pr.close();
// Kết quả
System.out.println("Du lieu doc duoc: "+sb);
```

Kết quả thực thi:

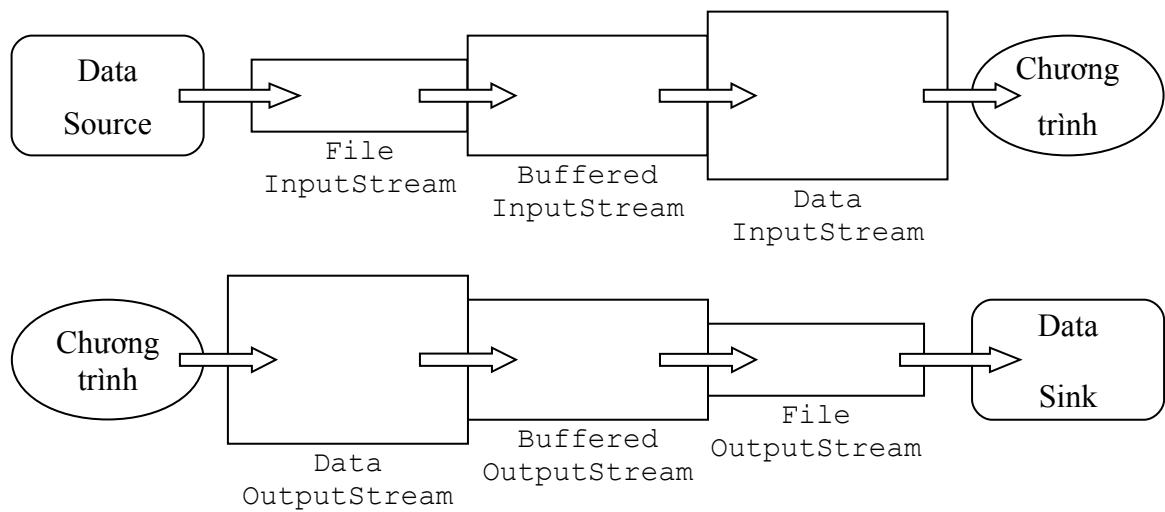
```
Du lieu ghi      : Hello Piped Reader/Writer!
Du lieu doc duoc: Hello Piped Reader/Writer!
Press any key to continue...
```

### 3.4 Các luồng lọc dữ liệu (Filter Stream)

Bởi vì các luồng dữ liệu được chuyển tải dạng byte hoặc ký tự, nên cần có một bộ biến đổi các luồng dữ liệu này sang những giá trị có định kiểu khác nhau. Java cung cấp thêm các luồng lọc dữ liệu. Khi các luồng này được tạo ra, nó phải được gắn với một luồng dữ liệu cụ thể để lọc nó. Do vậy, việc đọc dữ liệu từ một nguồn là sự phối hợp giữa các luồng tạo thành một dây chuyền xử lý trên các luồng khi có thao tác đọc/ghi xảy ra.

- Dây chuyền đọc bắt đầu bởi một luồng đọc cơ bản và kết thúc bởi một luồng lọc dữ liệu. Ở giữa có thể có thêm các luồng xử lý đọc trung gian.
- Dây chuyền ghi bắt đầu bởi một luồng xử lý ghi và kết thúc bởi một luồng ghi cơ bản. Ở giữa có thể có các luồng xử lý ghi trung gian.

Ví dụ:



Ví dụ về dây chuyền đọc/ghi dữ liệu

### 3.4.1 Các luồng lọc tổng quát

Trong Java, các luồng xử lý lọc tổng quát là các lớp trừu tượng sau:

- `FileInputStream`: luồng xử lý cho thao tác đọc dữ liệu dạng byte.
- `FileOutputStream`: luồng xử lý cho thao tác ghi dữ liệu dạng byte.
- `FileReader`: luồng xử lý cho thao tác đọc dữ liệu dạng ký tự.
- `FileWriter`: luồng xử lý cho thao tác ghi dữ liệu dạng ký tự.

### 3.4.2 Các luồng lọc hiện thực

Các luồng lọc hiện thực các luồng lọc tổng quát chịu trách nhiệm hiện thực các thao tác biến đổi dữ liệu có định kiểu thành luồng byte/ký tự hoặc ngược lại từ luồng byte/ký tự thành giá trị có định kiểu. Cụ thể, các luồng lọc này là `DataInputStream`, `DataOutputStream`. Các lớp này có những phương thức đặc trưng như sau:

Stream	Phương thức	Ý nghĩa
Data Input Stream	<code>DataInputStream</code> <code>(InputStream in)</code>	Tạo stream để đọc dữ liệu theo định dạng.
	<code>boolean readBoolean()</code>	Đọc một giá trị kiểu <b>boolean</b> . Các kiểu <b>byte</b> , <b>char</b> , <b>double</b> , <b>float</b> ,... cũng có những phương thức tương ứng.

<b>Data Output Stream</b>	<code>DataOutputStream (OutputStream out)</code>	Tạo stream để ghi dữ liệu theo định dạng.
	<code>int size()</code>	Trả về số byte mà stream đã ghi.
	<code>void writeBoolean (boolean v)</code>	Ghi một biến kiểu <b>boolean</b> . Các kiểu <b>byte</b> , <b>char</b> , <b>double</b> , <b>float</b> ,... cũng có những phương thức tương ứng.
	<code>void writeBytes(String s)</code>	Ghi chuỗi thành một dãy các byte.
	<code>void writeChars(String s)</code>	Ghi chuỗi thành một dãy các ký tự.

Lưu ý:

- Khi đọc các biến, phải đọc đúng theo thứ tự đã ghi.
- Để ghi chuỗi hoặc các đối tượng khác, ta phải dùng chức năng đọc/ghi object.
- Ta có thể kết hợp đọc có định dạng với vùng đệm.

Ví dụ: đọc/ghi các biến trên tập tin

```
String name="C:\\TestD.txt";
// Ghi dữ liệu
DataOutputStream fo=new DataOutputStream(new
FileOutputStream(name));
fo.writeBoolean(true);
fo.writeInt(786);
fo.writeFloat(0.123f);
fo.close();
// Đọc dữ liệu
DataInputStream fi=new DataInputStream(new FileInputStream
(name));
boolean b=fi.readBoolean();
int i=fi.readInt();
float f=fi.readFloat();
fo.close();
// Xuất dữ liệu đọc được ra màn hình
System.out.println("Du lieu doc duoc: ");
System.out.println(String.valueOf(b));
```

```
System.out.println(i);  
System.out.println(String.valueOf(f));
```

Kết quả thực thi:

```
Du lieu doc duoc:  
true  
786  
0.123  
Press any key to continue...
```

### 3.5 Các luồng đệm dữ liệu

Để tăng tốc độ đọc/ghi, nhất là đối với thao tác đọc/ghi trên bộ nhớ phụ, người ta dùng kỹ thuật vùng nhớ đệm.

- Vùng đệm đọc: dữ liệu sẽ được đọc vào vùng đệm thành từng khối, sau đó lấy ra từng từ từ, khối hết thì sẽ đọc tiếp khối kế → giảm số thao tác đọc.
- Vùng đệm ghi: dữ liệu cần ghi sẽ được gom lại đến khi đủ số lượng cần thiết thì sẽ được ghi một lần → giảm số thao tác ghi.

Như vậy, vùng đệm càng lớn thì càng tăng tốc độ, nhưng sẽ càng dễ xảy ra mất dữ liệu khi chương trình bị chấm dứt đột ngột. Ví dụ: ta yêu cầu ghi nhưng do chưa đủ dữ liệu nên chương trình chưa ghi thật sự vào đĩa, sau đó bị cúp điện → các dữ liệu được ta yêu cầu ghi đó sẽ mất.

Trong Java, các lớp luồng đệm hỗ trợ thao tác đọc/ghi như:

- `BufferedInputStream`: đọc dữ liệu dạng byte, có dùng vùng đệm
- `BufferedOutputStream`: ghi dữ liệu dạng byte, có dùng vùng đệm
- `BufferedReader`: đọc dữ liệu dạng ký tự, có dùng vùng đệm
- `BufferedWriter`: ghi dữ liệu dạng ký tự, có dùng vùng đệm

Khi tạo ra một đối tượng luồng đệm, nó phải gắn với một đối tượng luồng dữ liệu khác. Nhìn chung các luồng đệm có những phương thức giống như luồng thông thường, nhưng có khác biệt về cách khởi tạo:

Stream	Phương thức	Ý nghĩa
<b>Buffered Input Stream</b>	<code>BufferedInputStream (InputStream in)</code>	Tạo một stream vùng đệm để đọc dữ liệu dạng byte.
	<code>BufferedInputStream (InputStream in, <b>int</b> size)</code>	Tạo một stream với vùng đệm kích thước <code>size</code> để đọc dữ liệu dạng byte.
<b>Buffered Output Stream</b>	<code>BufferedOutputStream (OutputStream out)</code>	Tạo một stream vùng đệm để ghi dữ liệu dạng byte.
	<code>BufferedOutputStream (OutputStream out, <b>int</b> sz)</code>	Tạo một stream với vùng đệm kích thước <code>sz</code> byte để ghi dữ liệu dạng byte.
<b>Buffered Reader</b>	<code>BufferedReader (Reader in)</code>	Tạo một stream vùng đệm để đọc dữ liệu dạng ký tự.
	<code>BufferedReader (Reader in, <b>int</b> sz)</code>	Tạo một stream với vùng đệm kích thước <code>sz</code> để đọc dữ liệu dạng ký tự.
<b>Buffered Writer</b>	<code>BufferedWriter (Writer out)</code>	Tạo một stream vùng đệm để ghi dữ liệu dạng ký tự.
	<code>BufferedWriter (Writer out, <b>int</b> sz)</code>	Tạo một stream với vùng đệm kích thước <code>sz</code> byte để ghi dữ liệu dạng byte.
	<code><b>void</b> newLine()</code>	Ghi ký tự xuống dòng vào stream.

Bảng 10: Những phương thức đặc trưng của các stream vùng đệm

Ví dụ:

```
String fileName="C:\\TestC.txt";
String s="Hello Buffered File Reader/Writer!";
System.out.println("Du lieu ban dau : "+s);

// Ghi s vào tập tin
BufferedWriter bw=new BufferedWriter(new FileWriter(fileName));
bw.write(s);
bw.close();

// Đọc tập tin vào chuỗi sb
BufferedReader br=new BufferedReader(new FileReader(fileName));
```

```

StringBuffer sb=new StringBuffer();
char ca[]=new char[5];           // Đọc mỗi lần tối đa 5 ký
tự
while (br.ready())
{
    int len=br.read(ca);         // len: số ký tự đọc được thật sự
    sb.append(ca,0,len);
}
br.close();

// Xuất chuỗi sb
System.out.println("Du lieu doc duoc : "+sb);

```

Kết quả:

```

Du lieu ban dau : Hello Buffered File Reader/Writer!
Du lieu doc duoc : Hello Buffered File Reader/Writer!
Press any key to continue...

```

Nhìn vào ví dụ trên, ta thấy dùng stream vùng đệm và không dùng cho kết quả không khác gì nhau. Vậy đâu là ưu điểm của stream vùng đệm? Hãy xem ví dụ kế tiếp.

Ví dụ: dưới đây là ví dụ so sánh tốc độ khi dùng và không dùng stream vùng đệm:

```

String fileName="C:\\TestB.txt";
long n=50000;
// Ghi với stream vùng đệm
{
    long t=System.currentTimeMillis();

    FileOutputStream fo=new FileOutputStream(fileName);
    BufferedOutputStream bo=new BufferedOutputStream(fo);
    for(int i=0;i<n;i++)
        bo.write(i);
    bo.close();

    t=System.currentTimeMillis()-t;
    System.out.println("Ghi co vung dem : mat "+t+"ms.");
}

```

```

}
// Ghi không có stream vùng đệm
{
    long t=System.currentTimeMillis();

    FileOutputStream fo=new FileOutputStream(fileName);
    for(int i=0;i<n;i++)
        fo.write(i);
    fo.close();

    t=System.currentTimeMillis()-t;
    System.out.println("Ghi khong vung dem: mat "+t+"ms.");
}

```

Kết quả thực thi: thời gian chạy rất chênh lệch (kết quả có thể khác nhau tùy máy)

```

Ghi co vung dem    : mat 15ms.
Ghi khong vung dem: mat 1703ms.
Press any key to continue...

```

Ví dụ: kết hợp đọc/ghi có định dạng trên tập tin, có dùng vùng đệm

```

String fileName="C:\\TestB.txt";
// Ghi các số từ 0 tới 9 và căn bậc 2 tương ứng của chúng
// Tạo các stream
FileOutputStream fo=new FileOutputStream(fileName);
BufferedOutputStream bo=new BufferedOutputStream(fo);
DataOutputStream dos=new DataOutputStream(bo);
// Ghi các số
for(int i=0;i<10;i++)
{
    dos.writeInt(i);
    dos.writeDouble(Math.sqrt(i));
}
dos.close();
/* Đọc các số nguyên và căn bậc hai tương ứng */
// Tạo các stream
FileInputStream fi=new FileInputStream(fileName);

```

```

BufferedInputStream bi=new BufferedInputStream(fi);
DataInputStream dis=new DataInputStream(bi);
// Đọc các số và xuất ra màn hình
while(dis.available()>0)
{
    int i=dis.readInt();
    double sinI=dis.readDouble();
    System.out.println("sqrt("+i+")"+" = "+sinI);
}
dis.close();

```

Kết quả thực thi:

```

sqrt(0) = 0.0
sqrt(1) = 1.0
sqrt(2) = 1.4142135623730951
sqrt(3) = 1.7320508075688772
sqrt(4) = 2.0
sqrt(5) = 2.23606797749979
sqrt(6) = 2.449489742783178
sqrt(7) = 2.6457513110645907
sqrt(8) = 2.8284271247461903
sqrt(9) = 3.0
Press any key to continue...

```

### 3.6 Các lớp định dạng luồng dữ liệu nhập/xuất

Java hỗ trợ hai lớp đối tượng Scanner (thuộc gói java.util.\*) và PrintWriter (thuộc gói java.io.\*) rất hiệu quả cho việc định dạng dữ liệu của một luồng nhập/xuất. Bảng dưới đây trình bày một số phương thức thông dụng của hai lớp đối tượng này.

Phương thức	Ý nghĩa
<b>Lớp Scanner</b>	
Scanner( <b>InputStream</b> source)	Khởi tạo đối tượng Scanner lấy dữ liệu từ một luồng nhập.
Scanner( <b>ReadableByteChannel</b> source)	Khởi tạo đối tượng Scanner lấy dữ liệu từ một kênh nhập.



<code>void close()</code>	Đóng đối tượng Scanner.
<code>bool nextBoolean()</code>	Đọc một khối dữ liệu trong luồng và ép nó sang kiểu logic bool.
<code>int nextInt()</code>	Đọc một khối dữ liệu trong luồng và ép nó sang kiểu số nguyên int.
<code>long nextLong()</code>	Đọc một khối dữ liệu trong luồng và ép nó sang kiểu số nguyên long.
<code>float nextFloat()</code>	Đọc một khối dữ liệu trong luồng và ép nó sang kiểu số thực float.
<code>double nextDouble()</code>	Đọc một khối dữ liệu trong luồng và ép nó sang kiểu số thực double.
<code>String nextLine()</code>	Đọc các khối dữ liệu trong luồng và ép nó sang kiểu chuỗi String.
<code>bool hasNext()</code>	kiểm tra luồng còn khối dữ liệu nào không.
<b>Lớp PrintWriter</b>	
<code>PrintWriter(OutputStream out)</code>	Khởi tạo đối tượng PrintWriter để viết dữ liệu ra luồng xuất byte out.
<code>PrintWriter(Writer out)</code>	Khởi tạo đối tượng PrintWriter để viết dữ liệu ra luồng xuất ký tự out.
<code>void close()</code>	Đóng đối tượng PrintWriter.
<code>void print(String s)</code>	Ghi một chuỗi ra luồng.
<code>void print(bool b)</code>	Ghi giá trị bool ra luồng.
<code>void println()</code>	Ghi ký hiệu kết thúc dòng ra luồng
<code>void println(int i)</code>	Ghi một số nguyên i và ký hiệu kết thúc dòng ra luồng.
<code>void println(String s)</code>	Ghi một chuỗi s và ký hiệu kết thúc dòng ra luồng.
<code>void flush()</code>	Ghi toàn bộ dữ liệu trong PrintWriter ra luồng.

Ví dụ: Đọc và ghi dữ liệu có định dạng từ hai luồng nhập xuất cơ bản System.in và System.out

```
import java.util.*;
import java.io.*;
public class NhapXuat {
```

```

public static void main(String[] args) {
    //hien thi day so vua nhap len man hinh su dung PrintWriter
    PrintWriter pw=new PrintWriter(System.out);
    //Tao doi tuong Scanner de dinh dang du lieu cho luong
    //nhap ban phim
    Scanner sc=new Scanner(System.in);
    //doc so luong phan tu day so can nhap
    pw.println("nhap so phan tu n:"); pw.flush();
    int n =sc.nextInt();
    //doc vao mot day so n nguyen luu vao mang
    int a[] =new int[n];
    for(int i=0;i<n;i++){
        pw.println("a[" + i+"]=");pw.flush();
        a[i]=sc.nextInt();
    }
    pw.println("Hien thi day so su dung doi tuong Printer");
    for(int i=0;i<n;i++)
        pw.print(a[i]);
    pw.close();
}
}

```

### Kết quả thực hiện

```

nhap so phan tu n:
3
a[0]=1
a[1]=2
a[2]=3
Hien thi day so su dung doi tuong Printer
1
2
3

```

### 3.7 Tuần tự hóa đối tượng (object serialization)

Object serialization là khả năng biến đổi một đối tượng thành một dãy byte để lưu trữ trên bộ nhớ phụ hoặc truyền đi nơi khác. Trong Java, object serialization đã được tự động hóa hoàn toàn và hầu như lập trình viên không phải làm gì thêm.

Để một đối tượng có thể được "serialize", ta chỉ cần cho nó implement interface `Serializable`. Mọi việc sau đó như đọc/ghi/truyền/nhận đều do Java thực hiện.

Lưu ý:

- Chỉ có các thuộc tính (dữ liệu) của đối tượng mới được serialize.
- Các thuộc tính được đánh dấu bằng từ khóa **`transient`** (nghĩa là có tính tạm thời) sẽ không được serialize.
- Sau khi serialize, trạng thái của đối tượng được lưu trữ trên bộ nhớ ngoài được gọi là persistence (nghĩa là được giữ lại một cách bền vững).

Ví dụ: tạo ra lớp học sinh có thể được serialize:

```
public class HocSinh implements Serializable
{
    protected String hoTen;
    protected int namSinh;
    protected float diemVan, diemToan;
    protected transient float diemTrungBinh;
    // ...
}
```

### 3.8 Luồng viết đối tượng

Trong Java, các lớp đảm nhận việc đọc/ghi đối tượng gồm:

- `ObjectInputStream`: đọc dãy byte và chuyển thành đối tượng phù hợp.
- `ObjectOutputStream`: chuyển đối tượng thành dãy byte và ghi.

Các đối tượng đọc/ghi này cũng không thể hoạt động độc lập mà phải được liên kết với stream khác. Chúng có các phương thức chính như sau:

Stream	Phương thức	Ý nghĩa
Object Input Stream	<code>ObjectInputStream (InputStream in)</code>	Tạo stream để đọc đối tượng.
	<code>Object readObject()</code>	Đọc một đối tượng. Nếu đối tượng không được khai báo trong chương trình thì sẽ phát sinh <code>ClassNotFoundException</code> .
Object Output Stream	<code>ObjectOutputStream (OutputStream out)</code>	Tạo stream để ghi đối tượng.
	<code>void reset()</code>	Phục hồi lại stream và hủy bỏ thông tin về các đối tượng mà stream đã ghi.
	<code>writeObject(Object obj)</code>	Ghi một đối tượng.

Bảng 11: Những thao tác đặc trưng của các object stream

Ví dụ

Tạo ra lớp phân số có thể được serialize:

```
public class PhanSo implements Serializable
{
    protected int tu, mau;
    private transient float val=0;
    public PhanSo()
    {
        tu=0;
        mau=1;
    }
    public PhanSo(int tu, int mau)
    {
        this.tu=tu;
        this.mau=mau;
        val=(float) tu/mau;
    }
    public String toString()
    {
        return tu+"/"+mau+" (" +val+" )";
    }
}
```

```
}
```

Ghi và đọc các đối tượng phân số (có dùng stream vùng đệm):

```
// Ghi dữ liệu
FileOutputStream fo=new FileOutputStream("C:\\Test0.bin");
BufferedOutputStream bo=new BufferedOutputStream(fo);
ObjectOutputStream oo=new ObjectOutputStream(bo);

System.out.println("Du lieu ghi duoc:");
for(int i=0;i<4;i++)
{
    int tu =(int) (Math.random()*10);
    int mau=(int) (Math.random()*9)+1;
    PhanSo ps=new PhanSo(tu,mau);
    oo.writeObject(ps);
    System.out.println(ps);
}
oo.close();

// Đọc dữ liệu
FileInputStream fi=new FileInputStream("C:\\Test0.bin");
BufferedInputStream bi=new BufferedInputStream(fi);
ObjectInputStream oi=new ObjectInputStream(bi);

System.out.println("\nDu lieu doc duoc:");
while(bi.available(>0))
{
    PhanSo ps=(PhanSo)oi.readObject();
    System.out.println(ps);
}
oi.close();
```

Kết quả thực thi có dạng như sau (kết quả thực tế sẽ khác nhau với mỗi lần chạy):

Du lieu ghi duoc:

2/2 (1.0)

6/8 (0.75)

```
5/6 (0.8333333)
3/5 (0.6)

Du lieu doc duoc:
2/2 (0.0)
6/8 (0.0)
5/6 (0.0)
3/5 (0.0)
Press any key to continue...
```

Ta thấy thuộc tính `val` không được ghi do nó là **transient**. Nếu trong lớp phân số, ta bỏ từ khóa **transient** thì kết quả sẽ có dạng như sau:

```
Du lieu ghi duoc:
2/6 (0.33333334)
3/7 (0.42857143)
2/1 (2.0)
2/4 (0.5)

Du lieu doc duoc:
2/6 (0.33333334)
3/7 (0.42857143)
2/1 (2.0)
2/4 (0.5)
Press any key to continue...
```

### 3.9 Bài tập

6. Viết chương trình cài đặt tất cả các ví dụ chong chương.
7. Cho biết các loại lớp biểu diễn luồng dữ liệu trong Java. Nêu tính năng chính của từng loại.

## CHƯƠNG 4 . LẬP TRÌNH ĐA TUYẾN

### 4.1 Giới thiệu

Thông thường, một số chương trình ứng dụng cần xử lý nhiều yêu cầu cùng một lúc (ví dụ chương trình Webserver phải đáp trả nhiều yêu cầu của nhiều máy khách cùng một lúc hoặc một cửa sổ vừa có hình ảnh động vừa cho phép người sử dụng nhập liệu bình thường, chương trình soạn thảo văn bản sẽ vừa cho phép bạn gõ văn bản vừa chạy chức năng kiểm lỗi chính tả ...). Ngôn ngữ Java cho phép lập trình đa tuyến trình. Do vậy, các ứng dụng mạng thực hiện theo cơ chế khách/chủ được cài đặt dễ dàng.

#### 4.1.1 Đơn tuyến trình

Một chương trình thực hiện đơn tiến trình thì các câu lệnh của chương trình sẽ được thực hiện tuần tự. Nếu một lệnh không hoàn thành thì lệnh tiếp theo sẽ không được xử lý. Do vậy, trạng thái của chương trình tại bất kỳ có thể dự đoán được đang ở thời điểm nào cho trước.

#### 4.1.2 Đa tiến trình

Một chương trình có thực hiện đa tiến trình cho phép nhiều tiến trình thực hiện cùng một lúc. Mỗi tiến trình được xử lý bởi một CPU riêng.

Với máy có nhiều CPU (hoặc CPU đa nhân) thì việc xử lý đồng thời là thật nhưng dĩ nhiên số lệnh chạy cùng lúc không thể vượt quá số CPU. Khi đó, việc điều phối xem tiến trình chạy trên CPU nào thông thường sẽ do HĐH quy định.

Với CPU (đơn nhân) thì chỉ có thể thực thi lệnh một cách tuần tự, nghĩa là một CPU chỉ chạy một lệnh trong một thời điểm. Vậy làm sao chương trình đa tiến trình có thể chạy trên máy chỉ có 1 CPU ? Câu trả lời chính là phương pháp xoay vòng. Theo đó, hệ điều hành sẽ cấp cho mỗi tiến trình một CPU ảo và một vùng nhớ ảo, khi đó mỗi tiến trình đều "tưởng" rằng chỉ có một mình nó dùng CPU và bộ nhớ. Còn về thực chất, mỗi tiến trình sẽ dùng CPU trong một khoảng thời gian ngắn rồi trả CPU lại cho tiến trình khác sử dụng. Cách làm này thực chất không phải là đồng thời nhưng do việc "mượn" và "trả" CPU

diễn ra rất nhanh và hoàn toàn tự động nên người dùng sẽ không cảm thấy điều gì bất thường.

## 4.2 Tiến trình

Một tuyến trình đoạn có thể ở một trong bốn trạng thái sau trong suốt vòng đời của nó:

- Running: tiến trình đang thực thi.
- Suspended: việc thực thi tiến trình bị tạm dừng và có thể phục hồi để chạy tiếp.
- Blocked: tiến trình cần dùng tài nguyên nhưng tài nguyên đang được tiến trình khác dùng nên nó phải chờ.
- Terminated: việc thực thi của tiến trình bị ngừng hẳn và không thể phục hồi.

## 4.3 Lớp Thread

Để lập trình đa tiến trình, lớp dẫn xuất từ lớp `Thread` được cài đặt sau đó override phương thức `run()`. Khi tiến trình cần chạy thì phương thức `start()` của tiến trình được gọi thực hiện.

Xem ví dụ sau (file `Main01.java`):

```
public class Main01
{
    public static void main(String[] args)
    {
        MyThread t1=new MyThread(); // Tạo tiến trình
        MyThread t2=new MyThread();
        t1.start();                  // Gọi thực thi tiến trình
        t2.start();
    }
}
class MyThread extends Thread    // Kế thừa lớp Thread
{
    public void run()              // Override phương thức run
    {
        int i=0;
```



```

        while (i<200)
        {
            System.out.println(getName()+" "+i);
            i++;
        }
    }
}

```

## 4.4 Quản lý Thread

Quản lý thread gồm: tạo thread, chỉnh độ ưu tiên, chạy thread, và dừng thread.

### 4.4.1 Tạo Thread

Dùng các constructor sau:

Constructor	Ý nghĩa
Thread()	Tạo Thread với giá trị mặc định
Thread(String name)	Tạo Thread với tên cho trước
Thread(Runnable target)	Tạo thread liên kết với đối tượng Runnable
Thread(Runnable target, String name)	Tạo Thread liên kết với đối tượng Runnable và có tên cho trước

Nếu không dùng Runnable, ta buộc phải tạo class kế thừa từ class Thread có sẵn rồi override phương thức `run()`. Khi thread được kích hoạt, nó sẽ chạy các lệnh trong phương thức `run()` này. Do đó trong thực tế, ta dùng các constructor của lớp dẫn xuất chứ không dùng các constructor của thread một cách trực tiếp.

Ví dụ:

```

class MyThread extends Thread    // Kế thừa lớp Thread
{
    // ...
    public void run()             // Override phương thức run
    {
        // ...
    }
    // ...
}

```

```
}
```

Rồi dùng như sau:

```
MyThread t=new MyThread(); // Tạo tiểu trình  
t.start(); // Gọi thực thi tiểu trình
```

#### 4.4.2 *Chỉnh độ ưu tiên*

Trong Java, thread có 10 mức ưu tiên, đánh số từ 1 (độ ưu tiên thấp nhất) tới 10 (độ ưu tiên cao nhất). Khi thread được tạo ra, nó sẽ có độ ưu tiên mặc định là 5. ta có thể chỉnh lại độ ưu tiên của thread bằng phương thức **void** `setPriority(int newPriority)`.

Ví dụ:

```
t.setPriority(8);
```

Lưu ý:

- Nếu truyền vào độ ưu tiên không thuộc giá trị từ 1 tới 10, phương thức sẽ phát sinh `IllegalArgumentException`.
- Độ ưu tiên thấp nhất và cao nhất được lưu trong hai hằng (kiểu số nguyên) là `Thread.MIN_PRIORITY` và `Thread.MAX_PRIORITY`.

#### 4.4.3 *Thực thi thread*

Để thực thi thread, ta gọi phương thức **void** `start()`. Khi được thực thi, thread sẽ chạy các lệnh trong phương thức `run()`.

Ví dụ:

```
t.start();
```

Lưu ý:

- Nếu thread đã `start()` rồi, việc gọi `start()` lần nữa sẽ bị phát sinh exception `IllegalThreadStateException`.
- Để `start()` lại thread, ta cần tạo mới thread.

#### 4.4.4 *Dừng thread*

Để dừng thread đang chạy, ta dùng phương thức **void** `interrupt()`.

```
t.interrupt();
```

Nếu thread đang ở trạng thái sleep, việc dừng nó sẽ làm phát sinh InterruptedException.

#### 4.4.5 Một số phương thức quản lý khác của lớp Thread

Phương thức	Ý nghĩa
<pre>public final void wait(long timeout) throws InterruptedException</pre>	Tuyến trình hiện thời chờ cho tới khi được cảnh báo hoặc một khoảng thời gian timeout nhất định. Nếu timeout bằng 0 thì phương thức sẽ chỉ chờ cho tới khi có cảnh báo về sự kiện.
<pre>public final void notify()</pre>	Cảnh báo ít nhất một tuyến trình đang chờ một điều kiện nào đó thay đổi. Các tuyến trình phải chờ một điều kiện thay đổi trước khi có thể gọi phương thức wait nào đó.
<pre>public final void notifyAll()</pre>	Phương thức này cảnh báo tất cả các tuyến trình đang chờ một điều kiện thay đổi. Các tuyến trình đang chờ thường chờ một tuyến trình khác thay đổi điều kiện nào đó. Trong số các tuyến trình đã được cảnh báo, tuyến trình nào có độ ưu tiên cao nhất thì sẽ chạy trước tiên.
<pre>public static void sleep(long ms) throws InterruptedException</pre>	Phương thức này đưa tiến trình hiện hành vào trạng thái nghỉ tối thiểu là ms (mili giây).
<pre>public static void yeild()</pre>	Phương này giành lấy quyền thực thi của tuyến trình hiện hành cho một trong các tuyến trình khác.

## 4.5 Interface Runnable

Nếu không thể tạo class kế thừa từ Thread, ta có thể cho nó implement interface Runnable. Khi đó, class phải hiện thực phương thức **void** run() của interface này.

Sau đó, ta tạo thread liên kết với `Runnable` thông qua constructor tương ứng của lớp `Thread`. Khi thread được start, nó sẽ gọi thực thi lệnh trong phương thức `run()` của `Runnable` liên kết với nó.

Ví dụ:

```
public class Main09
{
    public static void main(String[] args)
    {
        Runnable r1=new MyRunnable();    // Tạo runnable
        Runnable r2=new MyRunnable();

        Thread t1=new Thread(r1);        // Tạo thread liên kết
        Thread t2=new Thread(r2);        // với runnable

        t1.start();
        t2.start();
    }
}

class MyRunnable implements Runnable
{
    public void run()    // Cài đặt phương thức run của Runnable
    {
        int i=0;
        while(i<200)
        {
            System.out.println(getName()+" "+i);
            i++;
        }
    }
}
```

Kết quả thực thi tương tự khi dùng cách kế thừa lớp `Thread`.

Việc quản lý Runnable, được tiến hành thông qua Thread liên kết với nó như đã trình bày trong phần trước.

## 4.6 Đồng bộ

### 4.6.1 Đồng bộ hóa sử dụng cho phương thức

Nếu nhiều thread cùng truy cập vào một vùng nhớ, lỗi có thể xảy ra. Để chỉ cho phép trong một lúc chỉ có tối đa một thread truy cập vào vùng nhớ, ta cần đồng bộ hóa bằng cách dùng từ khóa **synchronized**.

Xem ví dụ sau:

```
public class Main08 extends Thread
{
    public static void main(String[] args)
    {
        MyThread[] T=new MyThread[3];
        for(int i=0;i<T.length;i++)
        {
            T[i]=new MyThread();
            T[i].start();
        }
    }
}

class MyThread extends Thread
{
    public static int x=5;
    public static boolean stop=false;

    public static /*synchronized*/ void Trans()
    {
        if (x>0)
        {
            try
            {

```

```

        sleep(100);
    }
    catch (Exception E)
    {
        return;
    }
    x--;

}
}
public void run()
{
    do
    {
        Trans();
        System.out.println("x = "+x);
    }
    while (x>0);
}
}

```

Theo đoạn code thì các tiểu trình sẽ chỉ chạy khi  $x > 0$ .

Kết quả thực thi khi không đồng bộ:

```

x = 4
x = 3
x = 2
x = 1
x = 0
x = -1
x = -2
Press any key to continue...

```

Kết quả thực thi khi có đồng bộ:

```

x = 4
x = 3

```

```
x = 2
x = 1
x = 0
x = 0
x = 0
Press any key to continue...
```

Như vậy, trong thực tế nếu không dùng đồng bộ hóa, thì lại xảy ra trường hợp  $x < 0$ . Nguyên nhân như sau:

- Khi không đồng bộ, lúc  $x = 1$ , một tiểu trình nào đó kiểm tra thấy thỏa yêu cầu  $> 0$  và giảm  $x$ . Nhưng trong lúc  $x$  chưa kịp bị giảm thì tiểu trình khác cũng kiểm tra và thấy  $x$  vẫn thỏa nên lại giảm  $x$ . Kết quả là  $x$  sẽ bị giảm nhiều lần và sau cùng  $x$  sẽ có giá trị âm.
- Khi thêm từ khóa **synchronized** vào, trong một thời điểm chỉ có một tiểu trình được truy cập vào phương thức `Trans()` nên sẽ không bị lỗi nữa.

Một điều cần lưu ý là khi đồng bộ hóa, tốc độ xử lý sẽ chậm hơn khi không đồng bộ, nên ta không được làm dụng việc đồng bộ mà chỉ dùng khi cần thiết.

#### 4.6.2 Lệnh *synchronized*

Lệnh `synchronized` cho phép đồng bộ hóa một đối tượng mà không cần yêu cầu bạn tác động một phương thức `synchronized` trên đối tượng đó.

Cú pháp

```
synchronized (expr)
    statement
```

Khi có được khóa, `statement` được xử lý giống như nó là một phương thức `synchronized` trên đối tượng đó.

Ví dụ: Chuyển các phần tử trong mảng thành các số không âm

```
public static void abs(int[] v)
{
    synchronized(v)
    {
        for(int i=0;i<v.length;i++)
        {
```

```

        if (v[i]<0) v[i]=-v[i];
    }
}
}

```

#### 4.7 Trao đổi dữ liệu giữa các thread

Để trao đổi dữ liệu giữa các thread, ta dùng các lớp `PipedOutputStream` (xuất dữ liệu) và `PipedInputStream` (nhận dữ liệu). Lưu ý: cần import package `java.io`.

Để kết nối stream xuất và nhận, ta dùng phương thức `connect()` của các pipe. Sau khi đã kết nối, ta dùng các phương thức `read()` và `write()` để đọc / ghi dữ liệu.

Ví dụ:

```

import java.io.*;
public class Multi10
{
    public static void main(String[] args) throws Exception
    {
        Teacher t=new Teacher();
        Student s=new Student();
        t.pout.connect(s.pin);    // Kết nối hai Piped Stream
        //s.pin.connect(t.pout);  // Tương đương với lệnh trên
        t.start();
        s.start();
    }
}

class Teacher extends Thread
{
    public PipedOutputStream pout=new PipedOutputStream();
    // ..
    public void run()
    {
        // ..
    }
}

```



```

        // ..
    }

    class Student extends Thread
    {
        public PipedInputStream pin=new PipedInputStream();
        // ..
        public void run()
        {
            // ...
        }
        // ..
    }
}

```

Lưu ý:

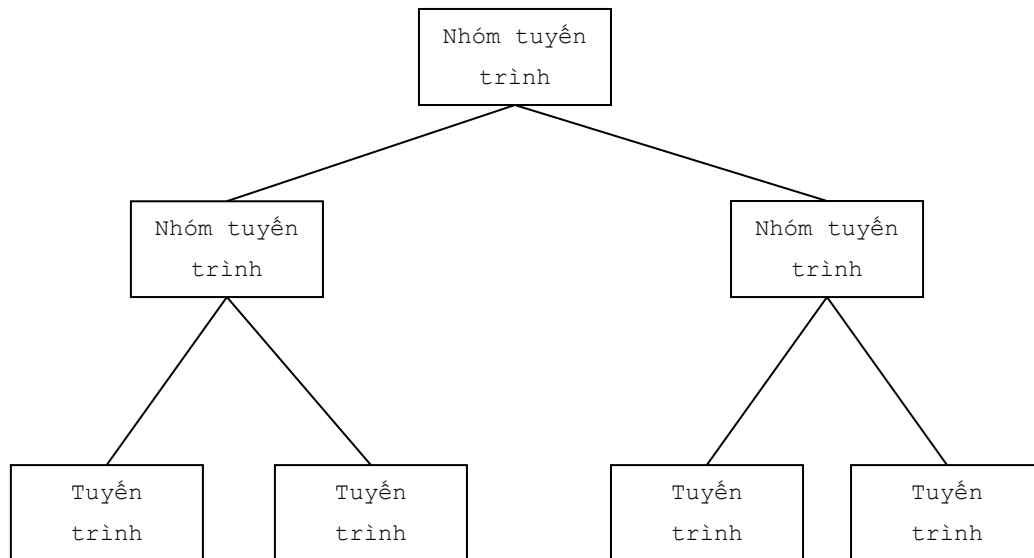
- Khi thread đang đọc dữ liệu từ stream mà stream không có dữ liệu, nó sẽ bị lock và không hoạt động cho đến khi đọc được dữ liệu hoặc stream đóng. Để tránh bị lock khi đang đọc, ta dùng phương thức `available()` của `PipedInputStream` để kiểm tra số byte dữ liệu đang có trong các stream. Nếu số byte thỏa yêu cầu thì mới tiến hành đọc.
- Để trao đổi dữ liệu dạng text, ta dùng `PipedReader` và `PipedWrite`. Cách dùng cũng tương tự `PipedInputStream` và `PipedOutputStream`.

#### 4.8 Nhóm các tuyến trình –ThreadGroup

Các tuyến trình có thể được tạo ra và được chạy riêng theo từng tuyến trình. Đôi khi, cần quản lý một nhóm các tuyến trình tại từng thời điểm. Đối với một danh sách các tuyến trình, cần quản lý các thao tác như tạm dừng, phục hồi, dừng hẳn hoặc ngắt. Khi đó, để thực hiện một thao tác, cần phải gọi từng tuyến trình ra xử lý riêng. Điều này thì phức tạp. Một giải pháp thay thế là nhóm các tuyến trình với nhau và áp dụng một thao tác trên nhóm.

Java API hỗ trợ khả năng làm việc với các nhóm tuyến trình bằng lớp `ThreadGroup`. Mục đích của lớp `ThreadGroup` là biểu diễn một tập hợp các tuyến trình, và cung cấp các phương thức tác động nhanh trên từng tuyến trình riêng trong nhóm. Một nhóm có thể bao

gồm nhiều tuyến trình. Một tiến trình có thể gia nhập nhóm. Tuy nhiên, không có cách nào để gỡ bỏ tuyến trình ra khỏi một nhóm. Một nhóm các tuyến trình có thể chứa các nhóm tuyến trình khác được. Các thao tác như dừng hoặc tạm dừng có thể được thực hiện trên các nhóm con hoặc nhóm cha. Khi một thao tác được áp dụng cho nhóm cha, thao tác sẽ được lan truyền tới nhóm con đó.



#### 4.8.1 Tạo một nhóm tuyến trình

Dùng các constructor sau:

Constructor	Ý nghĩa
<code>public ThreadGroup(String name) throws java.lang.SecurityException</code>	Constructor này tạo ra một nhóm tuyến trình mới, nhóm này có tên được xác định bởi một xâu ký tự truyền vào như một tham số.
<code>ThreadGroup(ThreadGroup parentGroup, String name) throws java.lang.SecurityException</code>	Tạo ra một nhóm tuyến trình mới được định danh bởi tên name. Nó là con của một nhóm cho truyền vào.

#### 4.8.2 Sử dụng một nhóm tuyến trình

Mỗi khi được tạo ra, một nhóm tuyến trình có thể được sử dụng như một tuyến trình bình thường. Nhóm tuyến trình này có thể được tạm dừng, phục hồi, ngắt hoặc dừng bằng cách

gọi phương thức thích hợp. Để sử dụng nhóm tuyến trình hiệu quả thì tuyến trình phải khác rỗng .

Nếu các tuyến trình không được gắn với một nhóm cụ thể tại thời điểm tạo ra chúng thì sau đó, không thể gắn một tuyến trình cho một nhóm sau đó, hoặc chuyển một tuyến trình từ nhóm này sang nhóm khác. Có ba constructor để thực hiện điều này

```
Thread(ThreadGroup group, Runnable runnbale)
Thread(ThreadGroup group, Runnable name)
Thread(ThreadGroup group, Runnable runnbale, String name)
```

Mỗi khi tạo ra một nhóm các tuyến trình, các phương thức sau được sử dụng để tác động lên nhóm.

Phương thức	Ý nghĩa
<code>int activeCount()</code>	trả về số tuyến trình trong nhóm, và các nhóm con.
<code>int activeGroupCount()</code>	trả về số nhóm con các tuyến trình
<code>boolean allowThreadSuspension()</code>	chỉ ra tuyến trình bị tạm ngừng hay không.
<code>void destroy()</code>	hủy bỏ nhóm tuyến trình
<code>int enumerate(Thread[] threadList)</code>	đưa các tuyến trình trong nhóm vào một mảng các tuyến trình.
<code>int enumerate(Thread[] threadList, boolean subgroupFlag)</code>	đưa các tuyến trình trong nhóm vào một mảng các tuyến trình. Phương thức này đưa các tuyến trình trong nhóm bao gồm cả các tuyến trình nhóm con nếu biến logic subgroupFlag được thiết lập là true.
<code>int enumerate(ThreadGroup[] groupList)</code>	đặt tất cả các nhóm con vào mảng

#### 4.8.3 Minh họa về lớp ThreadGroup

```
public class GroupDemo implements Runnable{
    public static void main(String args[]) throws Exception
    {
        //tạo ra nhóm tuyến trình
        ThreadGroup parent = new ThreadGroup("parent");
        //tạo ra nhóm tuyến trình con
```

```

        ThreadGroup subGroup;
        subGroup = new ThreadGroup(parent, "subgroup");
        //tao ra một số tuyến trình trong nhóm tuyến trình cha
        //và nhóm tuyến trình con
        Thread t1 = new Thread ( parent, new GroupDemo() );
        t1.start();
        Thread t2 = new Thread ( parent, new GroupDemo() );
        t2.start();
        Thread t3 = new Thread ( subGroup, new GroupDemo() );
        t3.start();
        //hiển thị nội dung nhóm
        parent.list();
        //đợi người sử dụng kết thúc
        System.out.println ("Press enter to continue");
        System.in.read();
        System.exit(0);
    }
    public void run(){
        // không làm gì hết
        for(;;)
        {
            Thread.yield();
        }
    }
}

```

### Kết quả thực hiện chương trình

```

C:\MyJava>java GroupDemo
java.lang.ThreadGroup[name=parent,maxpri=10]
    Thread[Thread-0,5,parent]
    Thread[Thread-1,5,parent]
java.lang.ThreadGroup[name=subgroup,maxpri=10]
    Thread[Thread-2,5,subgroup]
Press enter to continue

```

#### **4.9 Bài tập**

8. Cho chương trình minh hoạt cơ chế deadlock có thể xảy ra khi lập trình nhiều tuyến trình khi sử dụng dữ liệu chia sẻ.
9. Tìm hiểu vai trò của lập trình đa tuyến trình trong các ứng dụng mạng.
10. Sử dụng tuyến trình trong việc viết chương trình hiển thị đồng hồ đếm giờ lên một cửa sổ.

## **CHƯƠNG 5 . QUẢN LÝ ĐỊA CHỈ KẾT NỐI MẠNG VỚI CÁC LỚP INETADDRESS, URL VÀ URLCONNECTION**

Mạng máy tính là sự kết nối giữa các thiết bị (hay còn gọi là các node như máy tính, router, ..) với nhau. Các node là máy tính được gọi là host. Các thiết bị được kết nối với mạng LAN có địa chỉ vật lý (MAC) duy nhất. Điều này giúp cho các máy khác trên mạng trong việc truyền các gói tin đến đúng vị trí. Những người lập trình mạng không cần phải quan tâm đến từng chi tiết dữ liệu được định tuyến như thế nào trong một mạng LAN. Hơn nữa, Java không cung cấp khả năng truy xuất tới các giao thức tầng liên kết dữ liệu mức thấp được sử dụng bởi LAN. Việc hỗ trợ như vậy là rất khó khăn. Vì mỗi kiểu giao thức sử dụng một kiểu địa chỉ khác nhau và có các đặc trưng khác nhau, chúng ta cần các chương trình khác nhau cho mỗi kiểu giao thức mạng khác nhau. Đối với mạng Internet, địa chỉ MAC không hỗ trợ được việc xác định vùng mạng. Do đó, gói tin không thể được định tuyến tới máy đích. Tuy nhiên, mỗi node có một địa chỉ duy nhất để nhận dạng, được gọi là địa chỉ Internet. Định dạng địa chỉ được sử dụng thông dụng là IPv4. Vì sự phát triển nhanh của số lượng máy tính trên mạng Internet, định dạng địa chỉ IPv6 được sử dụng để mở rộng phạm vi miền địa chỉ. Một ứng dụng mạng cần sử dụng địa chỉ Internet để định danh và tạo sự kết nối giữa các máy tính. Ngôn ngữ lập trình Java hỗ trợ giao thức TCP/IP, giao thức này có nhiệm vụ liên kết mạng. Lớp InetAddress, lớp URL và URLConnection được cung cấp để quản lý địa chỉ kết nối mạng Internet. Đặc biệt lớp InetAddress được sử dụng dụng bởi các lớp Socket, ServerSocket, URL, DatagramSocket, DatagramPacket, .. trong các chương trình ứng dụng mạng.

### **5.1 Lớp InetAddress**

Lớp InetAddress nằm trong gói java.net và biểu diễn một địa chỉ Internet bao gồm hai thuộc tính cơ bản: `hostname` (String) và `address` (int[]).

### 5.1.1 Tạo các đối tượng *InetAddress*

Lớp *InetAddress* không giống với các lớp khác, không có các constructor cho lớp *InetAddress*. Tuy nhiên, lớp *InetAddress* có ba phương thức tĩnh trả về các đối tượng *InetAddress*

Các phương thức trong lớp *InetAddress*

Phương thức	Ý nghĩa
<code>public static InetAddress InetAddress.getByName(String hostname)</code>	Cung cấp tên miền, và hàm trả về đối tượng <i>InetAddress</i> .
<code>public static InetAddress[] InetAddress.getAllByName(String hostname)</code>	Cung cấp tên miền và trả về tất cả địa chỉ <i>InetAddress</i> có thể có.
<code>public static InetAddress InetAddress.getLocalHost()</code>	trả về đối tượng <i>InetAddress</i> biểu diễn địa chỉ Internet của máy đang chạy chương trình.

Tất cả các phương thức này đều thực hiện kết nối tới server DNS để biết được các thông tin trong đối tượng *InetAddress*

Ví dụ:

```
try{  
    InetAddress dc =InetAddress.getByName("www.microsoft.com");  
    System.out.println(dc);  
}  
  
catch(UnknownHostException e)  
{  
    System.err.println(e);  
}
```

Ví dụ : Viết chương trình nhận tên miền từ đối dòng lệnh và in ra địa chỉ IP tương ứng với tên miền đó.

```
import java.net.*;  
public class TimDCIP  
{  
    public static void main(String[] args)  
    {
```

```

try{
    if (args.length!=1)
    {
        System.out.println("Cach su dung: java TimDCIP
                                <Hostname>");
    }
    InetAddress host = InetAddress.getByName(args[0]);
    String hostName = host.getHostName();
    System.out.println("Host name:"+hostName);
    System.out.println("Dia chi
                                IP:"+host.getHostAddress());
}catch(UnknownHostException e)
{
    System.out.println("Khong tim thay dia chi");
    return;
}
}
}

```

### 5.1.2 Các thao tác đối dữ liệu của một đối tượng InetAddress

Chỉ có các lớp trong gói java.net có quyền truy xuất tới các trường của lớp InetAddress. Các lớp trong gói này có thể đọc dữ liệu của một đối tượng InetAddress bằng cách gọi phương thức getHostName và getAddress().

Phương thức	Ý nghĩa
<pre>public String getHostName()</pre>	Phương thức này trả về một chuỗi biểu diễn hostname của một đối tượng InetAddress. Nếu máy không có hostname, thì nó sẽ trả về địa chỉ IP của máy này dưới dạng một chuỗi ký tự.
<pre>public byte[] getAddress()</pre>	Trả về một địa chỉ IP dưới dạng một mảng các byte.

Ví dụ :Viết chương trình nhập một tên miền từ đối dòng lệnh và in ra dòng thông báo cho biết địa chỉ IP tương ứng với địa chỉ IP đó thuộc lớp nào.

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



```

public class PhanLoaiDCIP
{
    public static void main(String[] args)
    {
        try{
            if(args.length!=1)
            {
                System.out.println("Cach su dung: java TimDCIP
                                   <Hostname>");
            }
            InetAddress host = InetAddress.getByName(args[0]);
            String hostName = host.getHostName();
            System.out.println("Host name:"+hostName);
            System.out.println("Dia chi
                               IP:"+host.getHostAddress());
            byte[] b=host.getAddress();
            int i=b[0]>=0?b[0]:256+b[0];
            if((i>=1)&(i<=126))
                System.out.println(host+" thuoc dia chi lop A");
            if((i<=191)&(i>=128))
                System.out.println(host+" thuoc dia chi lop B");
            if((i<=223)&(i>=192))
                System.out.println(host+" thuoc dia chi lop C");
        }catch(UnknownHostException e){
            System.out.println("Khong tim thay dia chi");
            return;
        }
    }
}

```

Chú ý: Dòng lệnh `int i=b[0]>=0?b[0]:256+b[0]` làm nhiệm vụ chuyển đổi số nguyên có dấu ở dạng bù 2 về dạng số nguyên không dấu.

### 5.1.3 Các phương thức kiểm tra địa chỉ IP

Phương thức	Ý nghĩa
-------------	---------

<pre>public boolean isReachable(int timeout) throws IOException</pre>	<p>Kiểm tra xem một liên kết mạng đã được thiết lập hay chưa. Nếu host đáp ứng trong khoảng thời gian timeout mili giây, các phương thức này trả về giá trị true nếu đến được, ngược lại nó trả về giá trị false.</p> <p><i>Chú ý. Các liên kết có thể bị phong tỏa vì nhiều nguyên nhân như firewall, các server ủy quyền, các router hoạt động sai chức năng, dây cáp bị đứt, hoặc host ở xa không bật.</i></p>
<pre>public boolean isReachable(NetworkInterface netif, int ttl, int timeout) throws IOException</pre>	<p>Kiểm tra có thể tạo kết nối tới địa chỉ mạng này hay không bằng cách thực hiện một ICMP ECHO REQUEST hoặc thiết lập kết nối TCP tới cổng 7 (Echo) của máy đích.</p> <p>netif – null hoặc giao diện bất kỳ.</p> <p>ttl – số lượng tối đa các hops, mặc định là 0.</p> <p>timeout – thời gian xem xét tối đa, mili giây.</p>
<pre>public boolean isMulticastAddress()</pre>	<p>Kiểm tra có là địa chỉ Multicast hay không, true nếu đúng, ngược lại là false.</p>
<pre>public boolean isLoopbackAddress()</pre>	<p>Kiểm tra địa chỉ này có là địa chỉ loopback không.</p>
<pre>public boolean isSiteLocalAddress()</pre>	<p>Kiểm tra địa chỉ này có là địa chỉ cục bộ không. Tiềm lợi trong việc định tuyến.</p>

## 5.2 Lớp URL

Ngôn ngữ Java hỗ trợ việc định vị và lấy dữ liệu thông qua địa chỉ URL (Uniform Resource Locator) bằng lớp đối tượng URL. Lớp đối tượng này cho phép chương trình giao tiếp với Server để lấy dữ liệu về dựa trên các giao thức chuẩn được hỗ trợ bởi Server. Việc xác định Server, giao thức, cũng như cổng dịch vụ, đường dẫn là dựa vào URL được cung cấp cho lớp đối tượng URL. Do vậy, lớp URL quản lý các thông tin liên quan đến việc định vị tài nguyên như sau: giao thức (protocol), tên miền (hostname), cổng (port), đường dẫn (path), tên tập tin (filename), mục tài liệu (document section). Các thông tin này có thể thiết lập độc lập.

### 5.2.1 Tạo các URL

Trong việc tạo đối tượng URL, có bốn hàm khởi tạo hỗ trợ ứng với các trường hợp khác nhau của việc cung cấp tham số về định vị nguồn tài nguyên URL. Tất cả các hàm khởi tạo này đều ném ra ngoại lệ `MalformedURLException` nếu URL không đúng khuôn dạng.

Hàm khởi tạo	Ý nghĩa
<code>public URL(String url)</code> throws <code>MalformedURLException</code>	Tạo đối tượng URL với chuỗi url.
<code>public URL(String protocol, String host, String file)</code> throws <code>MalformedURLException</code>	Tạo đối tượng URL với từng phần tách biệt: giao thức, tên miền, tên tập tin, (cổng sẽ thiết lập là -1, sử dụng cổng mặc định của giao thức đó) .
<code>public URL(String protocol, String host, int port, String file)</code> throws <code>MalformedURLException</code>	Tạo ra đối tượng URL với đầy đủ các thông tin như giao thức, tên miền, số hiệu cổng, tên tập tin.
<code>public URL(URL u, String s)</code> throws <code>MalformedURLException</code>	Tạo ra đối tượng URL từ một đối tượng URL khác.

Ví dụ

```
try{
    URL u = new URL("http://www.sun.com/index.html");
}
catch(MalformedURLException e)
{
    System.err.println(e);
}
```

Ví dụ

```
try{
    URL u = new URL("http", "/www.sun.com", "index.html");
}
catch(MalformedURLException e){
    System.err.println(e);
}
```

Ví dụ

```
try{
URL u = new URL("http", "/www.sun.com", 80, "index.html");
}
catch (MalformedURLException e) {
    System.err.println(e);
}
```

Ví dụ

```
URL u1,u2;
try{
    URL u1= new URL("http://www.macfaq.com/index.html");
    URL u2 = new URL(u1,"vendor.html");
}
catch (MalformedURLException e)
{
    System.err.println(e);
}
```

### 5.2.2 Nhận thông tin các thành phần của URL

URL có sáu trường thông tin trong lớp URL: tên miền, địa chỉ IP, giao thức, cổng, tập tin, mục tham chiếu tài liệu.

Các phương thức	Ý nghĩa
<code>public String getProtocol()</code>	trả về một chuỗi ký tự biểu diễn giao thức
<code>public String getHost()</code>	trả về một chuỗi ký tự biểu diễn tên miền
<code>public int getPort()</code>	trả về một số nguyên kiểu int biểu diễn số hiệu cổng.
<code>public int getDefaultPort()</code>	trả về số hiệu cổng mặc định cho giao thức.
<code>public String getFile()</code>	trả về một chuỗi ký tự chứa đường dẫn tập tin

Ví dụ: Viết chương trình nhập vào một URL từ đối dòng lệnh và hiển thị từng thành phần tạo nên URL lên màn hình.

```
//Chương trình lấy thông tin của URL với url nhập từ bàn phím
import java.net.*;
```

```

class GetURLParts{
    public static void main(String[] args)    {
        try{
            URL u = new URL(args[0]);
            System.out.println("URL is "+u);
            System.out.println("The protocol part is"+u.getProtocol());
            System.out.println("The host part is "+u.getHost());
            System.out.println("The file part is "+u.getFile());
        }catch(MalformedURLException e){
            System.err.println(e);
        }
    }
}

```

**Kết quả thực hiện chương trình như sau:**

```

javac GetURLParts.java
java GetURLParts http://it.hutech.edu.vn/KhoaCNTT/index.php
URL is http://it.hutech.edu.vn/KhoaCNTT/index.php
The protocol part is http
The host part is it.hutech.edu.vn
The file part is /KhoaCNTT/index.php

```

**Ví dụ: kiểm tra giao thức được hỗ trợ bởi lớp URL.**

```

/* kiểm tra kiểu giao thức nào được máy ao hỗ trợ */

import java.net.*;
public class ProtocolTester {
    public static void main(String[] args) {
        // hypertext transfer protocol
        testProtocol("http://www.adc.org");
        // secure http
        testProtocol("https://www.amazon.com/exec/obidos/order2/");
        // file transfer protocol
        testProtocol("ftp://metalab.unc.edu/pub/languages/java/javafaq/");
        // Simple Mail Transfer Protocol
        testProtocol("mailto:elharo@metalab.unc.edu");
    }
}

```

```

// telnet
testProtocol("telnet://dibner.poly.edu/");
// local file access
testProtocol("file:///etc/passwd");
// gopher
testProtocol("gopher://gopher.anc.org.za/");
// Lightweight Directory Access Protocol
testProtocol("ldap://ldap.itd.umich.edu/o=University +
              "%20of%20Michigan,c=US?postalAddress");
// JAR
testProtocol("jar:http://cafeaulait.org/books/javaio/" +
              "ioexamples/javaio.jar!/com/macfaq/io/StreamCopier.class");
// NFS, Network File System
testProtocol("nfs://utopia.poly.edu/usr/tmp/");
// a custom protocol for JDBC
testProtocol("jdbc:mysql://luna.metalab.unc.edu:3306/NEWS");
// rmi, a custom protocol for remote method invocation
testProtocol("rmi://metalab.unc.edu/RenderEngine");
// custom protocols for HotJava
testProtocol("doc:/UsersGuide/release.html");
testProtocol("netdoc:/UsersGuide/release.html");
}
private static void testProtocol(String url) {
    try {
        URL u = new URL(url);
        System.out.println(u.getProtocol() + " is supported");
    }
    catch (MalformedURLException ex) {
        String protocol = url.substring(0, url.indexOf(':'));
        System.out.println(protocol + " is not supported");
    }
}
}

```

**Kết quả thực hiện:**

```
java ProtocolTester
```

```

http is supported
https is supported
ftp is supported
mailto is supported
telnet is not supported
file is supported
gopher is supported
ldap is not supported
jar is supported
nfs is not supported
jdbc is not supported
rmi is not supported
doc is supported
netdoc is supported

```

### 5.2.3 Nhận dữ liệu từ máy đích trong URL

Sau khi xác định được thông tin tài nguyên của dịch vụ máy đích, đối tượng URL hỗ trợ các phương thức để lấy dữ liệu từ máy đích về.

Các phương thức	Ý nghĩa
<pre> public final InputStream openStream() throws java.io.IOException </pre>	<p>Phương thức này kết nối tới một tài nguyên được tham chiếu bởi một URL, thực hiện cơ chế bắt tay cần thiết giữa client và server, và trả về một luồng nhập InputStream. Ta sử dụng luồng này để đọc dữ liệu. Dữ liệu nhận từ luồng này là dữ liệu thô của một tệp tin mà URL tham chiếu (mã ASCII nếu đọc một tệp văn bản, mã HTML nếu đọc một tài liệu HTML, một ảnh nhị phân nếu ta đọc một file ảnh). Nó không có các thông tin header và các thông tin có liên quan đến giao thức</p>
<pre> public URLConnection openConnection() throws java.io.IOException </pre>	<p>Phương thức openConnection() mở một socket tới một URL xác định và trả về một đối tượng URL. Một đối tượng URLConnection biểu diễn một liên kết mở tới một tài nguyên mạng. Nếu lời gọi</p>

	phương thức thất bại nó đưa ra ngoại lệ IOException.
<pre>public final Object getContent() throws java.io.IOException</pre>	<p>Phương thức getContent() tìm kiếm dữ liệu được tham chiếu bởi một URL và chuyển nó thành một kiểu đối tượng nào đó. Nếu đối tượng tham chiếu tới một kiểu đối tượng văn bản nào đó như tệp tin ASCII hoặc tệp HTML, đối tượng được trả về thông thường sẽ là một kiểu luồng nhập InputStream nào đó. Nếu URL tham chiếu tới một đối tượng ảnh như ảnh GIF hoặc JPEG thì phương thức getContent() trả về đối tượng java.awt.ImageProducer</p>

Ví dụ:Viết chương trình nhập một URL từ bàn phím, kết nối với Internet và hiển thị mã nguồn của trang Web đó lên màn hình.

```
import java.net.*;
import java.io.*;
public class ViewSource
{
    public static void main(String[] args)
    {
        URL u;
        String thisLine;
        if(args.length>0){
            try{
                u =new URL(args[0]);
                try{
                    DataInputStream dis = new
                    DataInputStream(u.openStream());
                    while((thisLine=dis.readLine())!=null)
                        System.out.println(thisLine);
                }catch(IOException e)
                {
                    System.err.println(e);
                }
            }
        }
    }
}
```



```

    }
    } catch (MalformedURLException e) {
        System.err.println(e);
    }
}
}
}

```

### 5.3 Lớp URLConnection

URLConnection là một lớp trừu tượng biểu diễn một liên kết tới một tài nguyên được xác định bởi một url. Lớp URLConnection cung cấp nhiều khả năng điều khiển hơn so với lớp URL thông qua việc tương tác với một server. Hơn nữa, lớp đối tượng này cho phép gửi dữ liệu đến Server và đọc thông tin đáp trả từ Server (tùy thuộc giao thức mà Server đó có hỗ trợ). Việc tạo ra đối tượng hiện thực lớp URLConnection bằng phương thức openConnection của lớp URL. Để sử dụng lớp URLConnection, thực hiện các bước cơ bản sau:

- Xây dựng một đối tượng URL.
- Gọi phương thức `openConnection()` của đối tượng URL để tìm kiếm một đối tượng `URLConnection` cho URL đó.
- Cấu hình đối tượng URL.
- Đọc các trường header.
- Nhận một luồng nhập và đọc dữ liệu.
- Nhận một luồng xuất và ghi dữ liệu.
- Đóng liên kết.

Một số phương thức hỗ trợ cho việc thao tác với lớp `URLConnection`.

Các phương thức	Ý nghĩa
public InputStream getInputStream()	trả lại một luồng đọc dữ liệu từ máy đích.
public OutputStream getOutputStream()	trả về một luồng OutputStream trên đó bạn có thể ghi dữ liệu để truyền tới máy đích.

Ví dụ: Đọc dữ liệu là một từ web đáp trả từ một web Server ở máy đích.

```
import java.net.*;
import java.io.*;
public class ViewSource
{
    public static void main(String[] args)
    {
        String thisLine;
        URL u;
        URLConnection uc;
        if(args.length>0)
        {
            try{
                u =new URL(args[0]);
                try{
                    uc=u.openConnection();
                    DataInputStream theHtml;
                    theHtml = new
                        DataInputStream(uc.getInputStream());
                    try{
                        while((thisLine=theHtml.readLine())!=null)
                        {
                            System.out.println(thisLine);
                        }
                    }
                    catch(Exception e)
                    {
                        System.err.println(e);
                    }
                }
            }
            catch(Exception e)
            {
                System.err.println(e);
            }
        }
    }
}
```

```

        catch (MalformedURLException e)
        {
            System.err.println(args[0] + " is not a parseable URL");
            System.err.println(e);
        }
    }
}

```

Ví dụ: Gửi dữ liệu lên WebServer của máy đích

```

try{
    URL u = new URL("http://www.somehost.com/cgi-bin/acgi");
    URLConnection uc = u.openConnection();
    uc.setDoOutput(true);
    DataOutputStream dos;
    dos = new DataOutputStream(uc.getOutputStream());
    dos.writeByte("Herre is some data");
}catch (Exception e)
{
    System.err.println(e);
}

```

Ngoài ra lớp `URLConnection` còn hỗ trợ một số phương thức xem thông tin về tập tin và các phương thức cho phép nhận và phân tích thông tin Header của giao thức HTTP.

Các phương thức	Ý nghĩa
<code>public int getLength()</code>	trả về độ dài của trường header.
<code>public long getDate()</code>	trả về một số nguyên kiểu long cho biết tài liệu đã được gửi khi nào.
<code>public String getEncoding()</code>	Phương thức này trả về String cho ta biết cách thức mã hóa. Nếu nội dung được gửi không được mã hóa (như trong trường hợp của HTTP server), phương thức này trả về giá trị null.
<code>public long getLastModified()</code>	trả về ngày mà tài liệu được sửa đổi lần cuối
<code>public Map</code>	trả lại thông tin trường Header đáp trả bởi Server ở

<b>getHeaderFields ()</b>	máy đích.
public String getHeaderField(String name)	trả về giá trị của thành phần trường header ứng với tên name.

Ví dụ, để tìm giá trị của các trường header Content-type, Content-encoding của một đối tượng *URLConnection uc* bạn có thể viết:

```
uc.getHeaderField("content-type");
uc.getHeaderField("content-encoding");
```

Để nhận thông tin về các trường Date, Content-length, hoặc Expiration bạn cũng thực hiện tương tự:

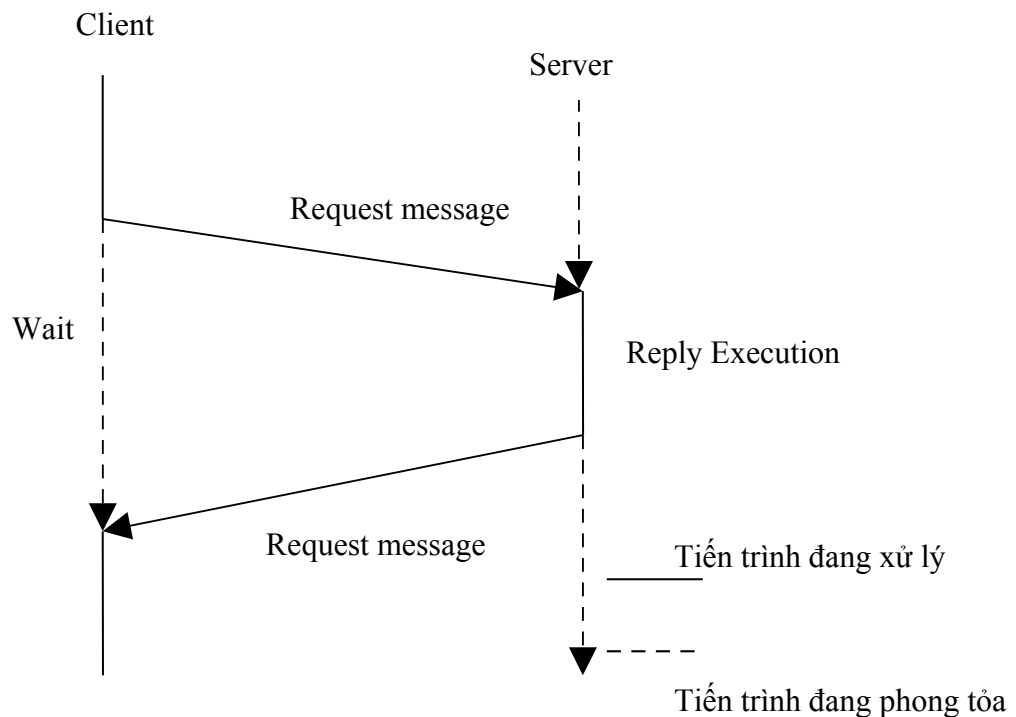
```
uc.getHeaderField("date");
uc.getHeaderField("expires");
uc.getHeaderField("Content-length");
```

## CHƯƠNG 6 . LẬP TRÌNH SOCKET CHO THỨC TCP

### 6.1 Mô hình client/server

Các ứng dụng mạng thường hoạt động theo mô hình client/server như thư điện tử, truyền nhận tập tin, game trên mạng, ... Mô hình này gồm có một chương trình đóng vai trò là client và một chương trình đóng vai trò là server. Hai chương trình này sẽ giao tiếp với nhau thông qua mạng. Chương trình server đóng vai trò cung cấp dịch vụ. Chương trình này luôn luôn lắng nghe các yêu cầu từ phía client, rồi tính toán và đáp trả kết quả tương ứng. Chương trình client cần một dịch vụ và gửi yêu cầu dịch vụ tới chương trình server và đợi đáp trả từ server. Như vậy, quá trình trao đổi dữ liệu giữa client/server bao gồm:

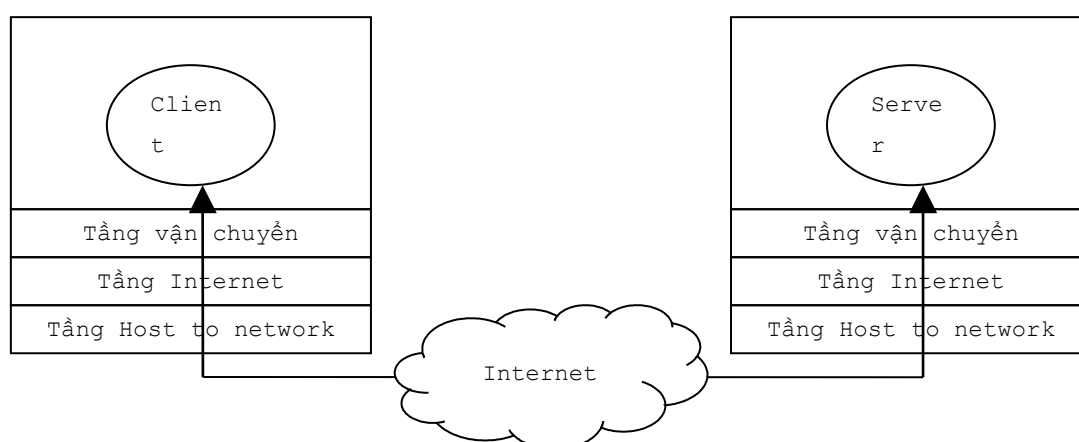
- Truyền một yêu cầu từ chương trình client tới chương trình server
- Yêu cầu được server xử lý
- Truyền đáp ứng cho client



Mô hình truyền tin này thực hiện truyền hai thông điệp qua lại giữa client và server một cách đồng bộ hóa. Chương trình server nhận được thông điệp từ client thì nó phát ra yêu cầu client chuyển sang trạng thái chờ (tạm dừng) cho tới khi client nhận được thông điệp đáp ứng do server gửi về. Mô hình client/server thường được cài đặt dựa trên các thao tác cơ bản là gửi (send) và nhận (receive).

## 6.2 Mô hình truyền tin socket

Chương trình client và server sử dụng giao thức vận chuyển để gửi và nhận dữ liệu. Một ví dụ là giao thức TCP/IP được sử dụng để giao tiếp qua mạng Internet.



TCP và UDP là các giao thức tầng giao vận để truyền dữ liệu. Mỗi giao thức có những ưu và nhược điểm riêng. Chẳng hạn, giao thức TCP có độ tin cậy truyền tin cao, nhưng tốc độ truyền tin bị hạn chế do phải có giai đoạn thiết lập và giải phóng liên kết khi truyền tin, khi gói tin có lỗi hay bị thất lạc thì giao thức TCP phải có trách nhiệm truyền lại,... Ngược lại, giao thức UDP có tốc độ truyền tin rất nhanh vì nó chỉ có một cơ chế truyền tin rất đơn giản: không cần phải thiết lập và giải phóng liên kết.

Dữ liệu được truyền trên mạng Internet dưới dạng các gói (packet) có kích thước hữu hạn được gọi là datagram. Mỗi datagram chứa một header và một payload. Header chứa địa chỉ và cổng cần truyền gói tin đến, cũng như địa chỉ và cổng xuất phát của gói tin, và các thông tin khác được sử dụng để đảm bảo độ tin cậy truyền tin, payload chứa dữ liệu. Tuy nhiên do các datagram có chiều dài hữu hạn nên thường phải phân chia dữ liệu thành nhiều gói và khôi phục lại dữ liệu ban đầu từ các gói ở nơi nhận. Trong quá trình truyền tin có thể có thể có một hay nhiều gói bị mất hay bị hỏng và cần phải truyền lại hoặc các gói tin đến không theo đúng trình tự. Để tránh những điều này, việc phân chia dữ liệu thành

các gói, tạo các header, phân tích header của các gói đến, quản lý danh sách các gói đã nhận được và các gói chưa nhận được, ... rất nhiều công việc cần phải thực hiện, và đòi hỏi rất nhiều phần mềm phức tạp.

Để giải quyết bài toán này, Đại học UC Berkeley đưa ra khái niệm Socket. Chúng cho phép người lập trình xem một liên kết mạng như là một luồng mà có thể đọc dữ liệu ra hay ghi dữ liệu vào từ luồng này. Các Socket che dấu người lập trình khỏi các chi tiết mức thấp của mạng như kiểu đường truyền, các kích thước gói, yêu cầu truyền lại gói, các địa chỉ mạng...

Một socket có thể thực hiện bảy thao tác cơ bản:

- Kết nối với một máy ở xa (ví dụ, chuẩn bị để gửi và nhận dữ liệu)
- Gửi dữ liệu
- Nhận dữ liệu
- Ngắt liên kết
- Gán cổng
- Nghe dữ liệu đến
- Chấp nhận liên kết từ các máy ở xa trên cổng đã được gán

Có nhiều kiểu Socket khác nhau tương ứng với mỗi kiểu giao thức được sử dụng để giao tiếp giữa hai máy trên mạng Internet. Đối với chồng giao thức TCP/IP, có hai kiểu Socket chính được sử dụng là stream socket và datagram socket. Stream socket sử dụng giao thức TCP để cung cấp dịch vụ gửi dữ liệu tin cậy. Datagram socket sử dụng giao thức UDP để cung cấp dịch vụ gói tin đơn giản. Ngôn ngữ lập trình Java hỗ trợ lớp Socket và ServerSocket cho kiểu stream socket và lớp DatagramPacket và DatagramSocket cho kiểu datagram socket.

Đối với kiểu stream socket, lớp Socket được sử dụng bởi cả client và server. Lớp Socket này có các phương thức tương ứng với bốn thao tác đầu tiên của thao tác cơ bản của socket trình bày ở trên. Ba thao tác cuối được hỗ trợ bởi server để chờ các client liên kết tới. Các thao tác này được cài đặt bởi lớp ServerSocket.

### 6.3 TCP client - Socket

Client thiết lập giao tiếp với server và đợi sự đáp trả từ Server. Các socket cho client thường được sử dụng theo các bước sau:

- Tạo ra một Socket mới sử dụng hàm khởi tạo với địa chỉ IP và số hiệu cổng dịch vụ của máy đích. Socket cố gắng liên kết với socket server của máy đích.
- Sau khi liên kết được thiết lập, lấy luồng nhập và luồng xuất được tạo ra giữa socket ở máy gửi - client và socket ở máy đích – server. Các luồng này được sử dụng để gửi dữ liệu cho nhau. Kiểu liên kết này được gọi là song công (full-duplex) các host có thể nhận và gửi dữ liệu đồng thời. Ý nghĩa của dữ liệu phụ thuộc vào giao thức.
- Khi việc truyền dữ liệu hoàn thành, một hoặc cả hai phía ngắt liên kết. Một số giao thức, như HTTP, đòi hỏi mỗi liên kết phải bị đóng sau mỗi khi yêu cầu được phục vụ. Các giao thức khác, chẳng hạn FTP, cho phép nhiều yêu cầu được xử lý trong một liên kết đơn.

#### 6.3.1 Các hàm khởi tạo của lớp Socket

Hàm khởi tạo	Ý nghĩa
<code>public Socket(String host, int port) throws UnknownHostException, IOException</code>	Hàm này tạo một socket TCP với host và cổng xác định, và thực hiện liên kết với host ở xa.
<code>public Socket(InetAddress host, int port) throws IOException</code>	tạo một socket TCP với thông tin là địa chỉ của một host được xác định bởi một đối tượng <code>InetAddress</code> và số hiệu cổng port, sau đó nó thực hiện kết nối tới host. Nó đưa ra ngoại lệ <code>IOException</code> nhưng không đưa ra ngoại lệ <code>UnknownHostException</code> . Constructor đưa ra ngoại lệ trong trường hợp không kết nối được tới host.
<code>public Socket (String host, int port, InetAddress interface, int localPort) throws</code>	tạo ra một socket với thông tin là địa chỉ IP được biểu diễn bởi một đối tượng <code>String</code> và một số hiệu cổng và thực hiện kết nối tới host đó. Socket kết



IOException, UnknownHostException	nối tới host ở xa thông qua một giao tiếp mạng và số hiệu cổng cục bộ được xác định bởi hai tham số sau. Nếu localPort bằng 0 thì Java sẽ lựa chọn một cổng ngẫu nhiên có sẵn nằm trong khoảng từ 1024 đến 65535.
public Socket (InetAddress host, int port, InetAddress interface, int localPort) throws IOException, UnknownHostException	Constructor chỉ khác constructor trên ở chỗ địa chỉ của host lúc này được biểu diễn bởi một đối tượng InetAddress.

Ví dụ. Tạo ra một TCP socket kết nối tới webserver của Vietnamnet

```
try{
    //ket noi voi may chu web cua Vietnamnet
    Socket s = new Socket("www.vnn.vn",80);
}
catch(UnknownHostException e){
    System.err.println(e);
}
catch(IOException e){
    System.err.println(e);
}
```

Trong đoạn code trên, nếu máy đích với tên miền không xác định hoặc máy chủ tên miền không hoạt động thì hàm khởi tạo sẽ đưa ra ngoại lệ UnknownHostException. Nếu vì một lý do nào đó mà không thể mở được socket thì hàm khởi tạo sẽ đưa ra ngoại lệ IOException.

Ví dụ: Viết chương trình để kiểm tra trên 1024 cổng đầu tiên những cổng nào đang có server hoạt động

```
import java.net.*;
import java.io.*;
class PortScanner
{
    public static void main(String[] args)
    {
```

```

String host="localhost";
if (args.length>0) {
    host=args[0];
}
for (int i=0;i<1024;i++) {
try{
    Socket s=new Socket(host,i);
    System.out.println("Co mot server dang
                                hoat dong tren cong:"+i);
}
catch (UnknownHostException e) {
    System.err.println(e);
}
catch (IOException e) {
    System.err.println(e);
}
}
}

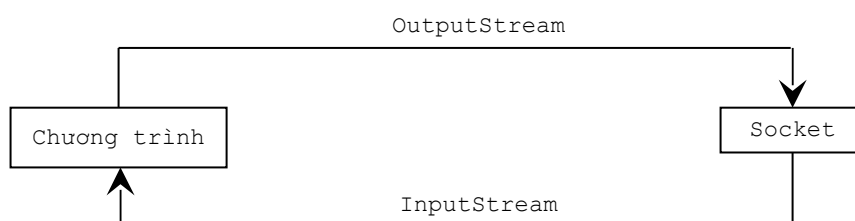
```

### 6.3.2 Các phương thức phục vụ giao tiếp giữa các Socket

Các phương thức	Ý nghĩa
public InetAddress getInetAddress()	Cho trước một đối tượng Socket, phương thức getInetAddress() cho trả lại địa chỉ máy đích bằng đối tượng InetAddress.
public int getPort()	Lấy số hiệu cổng dịch vụ của máy đích.
public int getLocalPort()	trả lại địa chỉ cổng sử dụng ở máy cục bộ.
public InetAddress getLocalAddress()	trả lại địa chỉ mạng của máy cục bộ dùng để giao tiếp với máy ở xa.
public InputStream getInputStream() throws IOException	trả về một luồng nhập để đọc dữ liệu từ một socket vào chương trình. Thông thường ta có thể gắn kết luồng nhập thô InputStream tới một luồng lọc hoặc một luồng ký tự nhằm đưa các chức năng tiện ích (chẳng hạn như các luồng InputStream, hoặc

	InputStreamReader). Để tăng cao hiệu năng, ta có thể đệm dữ liệu bằng cách gắn kết nó với luồng lọc BufferedInputStream hoặc BufferedReader.
<pre>public OutputStream getOutputStream() throws IOException</pre>	trả về một luồng xuất thô để ghi dữ liệu từ ứng dụng ra đầu cuối của một socket. Thông thường, ta sẽ gắn kết luồng này với một luồng tiện lợi hơn như lớp DataOutputStream hoặc OutputStreamWriter trước khi sử dụng nó. Để tăng hiệu quả ghi.

Hai phương thức `getInputStream()` và `getOutputStream()` là các phương thức cho phép lấy về các luồng dữ liệu nhập và xuất. Để nhận dữ liệu từ một máy ở xa, một luồng nhập từ socket được nhận và đọc dữ liệu từ luồng đó. Để ghi dữ liệu lên một máy ở xa, một luồng xuất được nhận từ socket và ghi dữ liệu lên luồng. Dưới đây là hình vẽ minh họa trực quan.



### 6.3.3 Các phương thức đóng Socket

Khi giao tiếp giữa client và server được thực hiện xong, cần thiết phải đóng kênh kết nối lại. Dưới đây là các phương thức hỗ trợ đóng kết nối.

Các phương thức	Ý nghĩa
<code>public void close() throws IOException</code>	Đóng luồng kết nối lại.
<code>public void shutdownInput() throws IOException</code>	đóng một luồng nhập.
<code>public void shutdownOutput() throws IOException</code>	đóng luồng xuất.
<code>public boolean isOutputShutdown()</code>	kiểm tra luồng xuất đóng chưa.
<code>public boolean</code>	kiểm tra luồng nhập đóng chưa.

<code>isInputShutdown()</code>	
--------------------------------	--

Các socket được đóng một cách tự động khi một trong hai luồng đóng lại, hoặc khi chương trình kết thúc, hoặc khi socket được thu hồi bởi gabbage collector. Tuy nhiên, thực tế cho thấy việc cho rằng hệ thống sẽ tự đóng socket là không tốt, đặc biệt là khi các chương trình chạy trong khoảng thời gian vô hạn. Mỗi khi một Socket đã bị đóng lại, các trường thông tin InetAddress, địa chỉ cục bộ, và số hiệu cổng cục bộ vẫn có thể truy xuất tới được thông qua các phương thức `getInetAddress()`, `getPort()`, `getLocalHost()`, và `getLocalPort()`. Tuy nhiên khi ta gọi các phương thức `getInputStream()` hoặc `getOutputStream()` để đọc dữ liệu từ luồng đọc `InputStream` hoặc ghi dữ liệu `OutputStream` thì ngoại lệ `IOException` được đưa ra.

#### 6.3.4 Các phương thức thiết lập các tùy chọn cho Socket

**TCP\_NODELAY:** Thiết lập giá trị `TCP_NODELAY` là true để đảm bảo rằng các gói tin được gửi đi nhanh nhất có thể mà không quan tâm đến kích thước của chúng. Thông thường, các gói tin nhỏ được kết hợp lại thành các gói tin lớn hơn trước khi được gửi đi. Trước khi gửi đi một gói tin khác, host cục bộ đợi để nhận các xác thực của gói tin trước đó từ hệ thống ở xa.

<pre>public void setTcpNoDelay(boolean on) throws SocketException public boolean getTcpNoDelay() throws SocketException</pre>
---

**SO\_LINGER:** Tùy chọn `SO_LINGER` xác định phải thực hiện công việc gì với datagram vẫn chưa được gửi đi khi một socket đã bị đóng lại. Ở chế độ mặc định, phương thức `close()` sẽ có hiệu lực ngay lập tức; nhưng hệ thống vẫn cố gắng để gửi phần dữ liệu còn lại. Nếu `SO_LINGER` được thiết lập bằng 0, các gói tin chưa được gửi đi bị phá hủy khi socket bị đóng lại. Nếu `SO_LINGER` lớn hơn 0, thì phương thức `close()` phong tỏa để chờ cho dữ liệu được gửi đi và nhận được xác thực từ phía nhận. Khi hết thời gian qui định, socket sẽ bị đóng lại và bất kỳ phần dữ liệu còn lại sẽ không được gửi đi.

<pre>public void setSoLinger(boolean on, int seconds) throws SocketException public int getSoLinger() throws SocketException</pre>
--

**SO\_TIMEOUT:** Thông thường khi ta đọc dữ liệu từ một socket, lời gọi phương thức phong tỏa cho tới khi nhận đủ số byte. Bằng cách thiết lập phương thức `SO_TIMEOUT`, ta sẽ

đảm bảo rằng lời gọi phương thức sẽ không phong tỏa trong khoảng thời gian quá số giây quy định.

```
public void setSoTimeout(int milliseconds) throws SocketException
public int getSoTimeout() throws SocketException
```

### 6.3.5 Một số ví dụ chương trình client giao tiếp trên một số dịch vụ chuẩn

Finger là giao thức đơn giản được miêu tả trong RFC 1288. Finger sử dụng giao thức TCP trên cổng dịch vụ 79. Giao thức này được sử dụng để xem thông tin người sử dụng đã đăng nhập hệ thống máy tính ở xa.

## 6.4 Lớp ServerSocket

Lớp ServerSocket có đủ mọi thứ ta cần để viết các server bằng Java. Nó có các constructor để tạo các đối tượng ServerSocket mới, các phương thức để lắng nghe các liên kết trên một cổng xác định, và các phương thức trả về một Socket khi liên kết được thiết lập, vì vậy ta có thể gửi và nhận dữ liệu.

Vòng đời của một server

1. Một ServerSocket mới được tạo ra trên một cổng xác định bằng cách sử dụng một constructor ServerSocket.
2. ServerSocket lắng nghe liên kết đến trên cổng đó bằng cách sử dụng phương thức accept(). Phương thức accept() phong tỏa cho tới khi một client thực hiện một liên kết, phương thức accept() trả về một đối tượng Socket mà liên kết giữa client và server.
3. Tùy thuộc vào kiểu server, hoặc phương thức getInputStream(), getOutputStream() hoặc cả hai được gọi để nhận các luồng vào ra để truyền tin với client.
4. server và client tương tác theo một giao thức thỏa thuận sẵn cho tới khi ngắt liên kết.
5. Server, client hoặc cả hai ngắt liên kết
6. Server trở về bước hai và đợi liên kết tiếp theo.

### 6.4.1 Các constructor

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

Constructor này tạo một socket cho server trên cổng xác định. Nếu port bằng 0, hệ thống chọn một cổng ngẫu nhiên cho ta. Cổng do hệ thống chọn đôi khi được gọi là cổng vô danh vì ta không biết số hiệu cổng. Với các server, các cổng vô danh không hữu ích lắm vì các client cần phải biết trước cổng nào mà nó nối tới (giống như người gọi điện thoại ngoài việc xác định cần gọi cho ai cần phải biết số điện thoại để liên lạc với người đó).

Ví dụ: Để tạo một server socket cho cổng 80

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

Constructor đưa ra ngoại lệ `IOException` nếu ta không thể tạo và gán Socket cho cổng được yêu cầu. Ngoại lệ `IOException` phát sinh khi:

- Cổng đã được sử dụng
- Không có quyền hoặc cố liên kết với một cổng nằm giữa 0 và 1023.

Ví dụ;

```
import java.net.*;
import java.io.*;
public class congLocalHost
{
    public static void main(String[] args)
    {
        ServerSocket ss;
        for(int i=0;i<=1024;i++)
        {
            try{
                ss= new ServerSocket(i);
                ss.close();
            }
        }
    }
}
```

```

    }
    catch(IOException e)
    {
        System.out.println("Co mot server tren cong "+i);
    }
}
}
}

```

- `public ServerSocket(int port, int queuelength, InetAddress bindAddress) throws IOException`

Constructor này tạo một đối tượng `ServerSocket` trên cổng xác định với chiều dài hàng đợi xác định. `ServerSocket` chỉ gán cho địa chỉ IP cục bộ xác định. Constructor này hữu ích cho các server chạy trên các hệ thống có nhiều địa chỉ IP.

#### 6.4.2 *Chấp nhận và ngắt liên kết*

Một đối tượng `ServerSocket` hoạt động trong một vòng lặp chấp nhận các liên kết. Mỗi lần lặp nó gọi phương thức `accept()`. Phương thức này trả về một đối tượng `Socket` biểu diễn liên kết giữa client và server. Tương tác giữa client và server được tiến hành thông qua socket này. Khi giao tác hoàn thành, server gọi phương thức `close()` của đối tượng socket. Nếu client ngắt liên kết trong khi server vẫn đang hoạt động, các luồng vào ra kết nối server với client sẽ đưa ra ngoại lệ `InterruptedException` trong lần lặp tiếp theo

- `public Socket accept() throws IOException`

Khi bước thiết lập liên kết hoàn thành, và ta sẵn sàng để chấp nhận liên kết, cần gọi phương thức `accept()` của lớp `ServerSocket`. Phương thức này phong tỏa; nó dừng quá trình xử lý và đợi cho tới khi client được kết nối. Khi client thực sự kết nối, phương thức `accept()` trả về đối tượng `Socket`. Ta sử dụng các phương thức `getInputStream()` và `getOutputStream()` để truyền tin với client.

Ví dụ:

```

try{
    ServerSocket theServer = new ServerSocket(5776);
    while(true)
    {

```

```

        Socket con = theServer.accept();
        PrintStream p = new PrintStream(con.getOutputStream());
        p.println("Ban da ket noi toi server nay. Bye-bye now.");
        con.close();
    }
}
catch(IOException e)
{
    System.err.println(e);
}

```

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

Nếu ta đã kết thúc làm việc với một đối tượng server socket thì cần phải đóng lại đối tượng này.

Ví dụ: Cài đặt một server daytime

```

import java.net.*;
import java.io.*;
import java.util.Date;
public class daytimeServer{
    public final static int daytimePort =13;

    public static void main(String[]args)
    {
        ServerSocket  theServer;
        Socket con;
        PrintStream p;
        try{
            theServer = new ServerSocket(daytimePort);
            try{
                p= new PrintStream(con.getOutputStream());
                p.println(new Date());
                con.close();
            }
        }
        catch(IOException e)
        {

```



```

        theServer.close();
        System.err.println(e);
    }
}
catch(IOException e)
{
    System.err.println(e);
}
}
}

```

- `public void close() throws IOException`

Nếu đã hoàn thành công việc với một `ServerSocket`, ta cần phải đóng nó lại, đặc biệt nếu chương trình của ta tiếp tục chạy. Điều này nhằm tạo điều kiện cho các chương trình khác muốn sử dụng nó. Đóng một `ServerSocket` không đồng nhất với việc đóng một `Socket`.

Lớp `ServerSocket` cung cấp một số phương thức cho ta biết địa chỉ cục bộ và cổng mà trên đó đối tượng server đang hoạt động. Các phương thức này hữu ích khi ta đã mở một đối tượng server socket trên một cổng vô danh và trên một giao tiếp mạng không

- `public InetAddress getInetAddress()`

Phương thức này trả về địa chỉ được sử dụng bởi server (localhost). Nếu localhost có địa chỉ IP, địa chỉ này được trả về bởi phương thức `InetAddress.getLocalHost()`

Ví dụ:

```

try{
    ServerSocket httpd = new ServerSocket(80);
    InetAddress ia = httpd.getInetAddress();
}
catch(IOException e)
{
}

```

- `public int getLocalHost()`

Các constructor `ServerSocket` cho phép ta nghe dữ liệu trên cổng không định trước bằng cách gán số 0 cho cổng. Phương thức này cho phép ta tìm ra cổng mà server đang nghe.

## 6.5 Các bước cài đặt chương trình phía Client bằng Java

Sau khi đã tìm hiểu các lớp và các phương thức cần thiết để cài đặt chương trình Socket. Ở mục 6 và mục 7 chúng ta sẽ đi vào các bước cụ thể để cài đặt các chương trình Client và Server.

Các bước để cài đặt Client

Bước 1:Tạo một đối tượng Socket

```
Socket client =new Socket("hostname",portName);
```

Bước 2:Tạo một luồng xuất để có thể sử dụng để gửi thông tin tới Socket

```
PrintWriter out=new PrintWriter(client.getOutputStream(),true);
```

Bước 3:Tạo một luồng nhập để đọc thông tin đáp ứng từ server

```
BufferedReader in=new BufferedReader (new  
InputStreamReader(client.getInputStream()));
```

Bước 4:Thực hiện các thao tác vào/ra với các luồng nhập và luồng xuất

Đối với các luồng xuất, PrintWriter, ta sử dụng các phương thức print và println, tương tự như System.out.println.

Đối với luồng nhập, BufferedReader, ta có thể sử dụng phương thức read() để đọc một ký tự, hoặc một mảng các ký tự, hoặc gọi phương thức readLine() để đọc vào một dòng ký tự. Cần chú ý rằng phương thức readLine() trả về null nếu kết thúc luồng.

Bước 5: Đóng socket khi hoàn thành quá trình truyền tin

Ví dụ: Viết chương trình client liên kết với một server. Người sử dụng nhập vào một dòng ký tự từ bàn phím và gửi dữ liệu cho server.

```
import java.net.*;  
import java.io.*;  
public class EchoClient1  
{  
    public static void main(String[] args)  
    {  
        String hostname="localhost";  
        if(args.length>0)  
        {  
            hostname=args[0];  
        }  
        PrintWriter pw=null;
```

```

BufferedReader br=null;
try{
    Socket s=new Socket(hostname,2007);
    br=newBufferedReader(new
InputStreamReader(s.getInputStream()));
    BufferedReader user;
    user=new BufferedReader(new InputStreamReader(System.in));
    pw=new PrintWriter(s.getOutputStream());
    System.out.println("Da ket noi duoc voi server...");
    while(true)
    {
        String st=user.readLine();
        if(st.equals("exit"))
        {
            break;
        }
        pw.println(st);
        pw.flush();
        System.out.println(br.readLine());
    }
}
catch(IOException e)
{
    System.err.println(e);
}
finally{
    try{
        if(br!=null)br.close();
        if(pw!=null)pw.close();
    }
    catch(IOException e)
    {
        System.err.println(e);
    }
}
}

```

```
}
```

Chương trình EchoClient đọc vào hostname từ đối dòng lệnh. Tiếp theo ta tạo một socket với hostname đã xác định trên cổng số 2007. Tất nhiên cổng này hoàn toàn do ta lựa chọn sao cho nó không trùng với cổng đã có dịch vụ hoạt động. Việc tạo socket thành công có nghĩa là ta đã liên kết được với server. Ta nhận luồng nhập từ socket thông qua phương thức `getInputStream()` và gắn kết nó với các luồng ký tự và luồng đệm nhờ lệnh:

```
br=new BufferedReader(new InputStreamReader(s.getInputStream()));
```

Tương tự ta lấy về luồng xuất thông qua phương thức `getOutputStream()` của socket. Sau đó gắn kết luồng này với luồng `PrintWriter` để gửi dữ liệu tới server

```
pw=new PrintWriter(s.getOutputStream());
```

Để đọc dữ liệu từ bàn phím ta gắn bàn phím với các luồng nhập nhờ câu lệnh:

```
BufferedReader user=new BufferedReader(new  
InputStreamReader(System.in));
```

Sau khi đã tạo được các luồng thì vấn đề nhận và gửi dữ liệu trở thành vấn đề đơn giản là đọc dữ liệu từ các luồng nhập `br`, `user` và ghi dữ liệu lên luồng xuất `pw`.

## 6.6 Các bước để cài đặt chương trình Server bằng Java

Để cài đặt chương trình Server bằng `ServerSocket` ta thực hiện các bước sau:

Bước 1

Tạo một đối tượng `ServerSocket`

```
ServerSocket ss=new ServerSocket(port)
```

Bước 2:

Tạo một đối tượng `Socket` bằng cách chấp nhận liên kết từ yêu cầu liên kết của client. Sau khi chấp nhận liên kết, phương thức `accept()` trả về đối tượng `Socket` thể hiện liên kết giữa Client và Server.

```
while(<btđieukien>)  
{  
    Socket s=ss.accept();  
    doSomething(s);  
}
```

Người ta khuyến cáo rằng chúng ta nên giao công việc xử lý đối tượng `s` cho một tuyến đoạn nào đó.

### Bước 3: Tạo một luồng nhập để đọc dữ liệu từ client

```
BufferedReader in=new BufferedReader(new  
InputStreamReader(s.getInputStream()));
```

### Bước 4: Tạo một luồng xuất để gửi dữ liệu trở lại cho server

```
PrintWriter pw=new PrintWriter(s.getOutputStream(),true);
```

Trong đó tham số true được sử dụng để xác định rằng luồng sẽ được tự động đẩy ra.

### Bước 5: Thực hiện các thao tác vào ra với các luồng nhập và luồng xuất

Bước 6: Đóng socket s khi đã truyền tin xong. Việc đóng socket cũng đồng nghĩa với việc đóng các luồng.

Ví dụ: Viết chương trình server EchoServer để phục vụ chương trình EchoClient1 đã viết ở bước 5

```
import java.net.*;  
import java.io.*;  
public class EchoServer1  
{  
    public final static int DEFAULT_PORT=2007;  
    public static void main(String[] args)  
    {  
        int port=DEFAULT_PORT;  
        try{  
            ServerSocket ss=new ServerSocket(port);  
            Socket s=null;  
            while(true)  
            {  
                try{  
                    s=ss.accept();  
                    PrintWriter pw; BufferedReader br;  
pw=new PrintWriter(new OutputStreamWriter(s.getOutputStream()));  
br=newBufferedReader(new InputStreamReader(s.getInputStream()));  
                    while(true){  
                        String line=br.readLine();  
                        if(line.equals("exit"))break;  
                        String upper=line.toUpperCase();  
                        pw.println(upper);  
                        pw.flush();  
                    }  
                }  
            }  
        }  
    }  
}
```

```

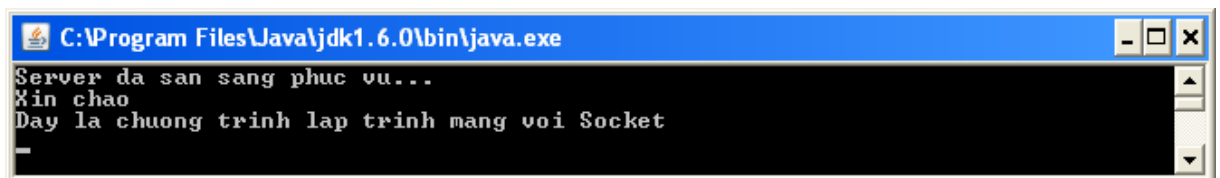
    }catch(IOException e){ }
    finally{
        try{
            if(s!=null){
                s.close();
            }
        }catch(IOException e){}
    }
    }
    }catch(IOException e){}
    }
}

```

Chương trình bắt đầu bằng việc tạo ra một đối tượng ServerSocket trên cổng xác định. Server lắng nghe các liên kết trong một vòng lặp vô hạn. Nó chấp nhận liên kết bằng cách gọi phương thức accept(). Phương thức accept() trả về một đối tượng Socket thể hiện mỗi liên kết giữa client và server. Ta cũng nhận về các luồng nhập và luồng xuất từ đối tượng Socket nhờ các phương thức getInputStream() và getOutputStream(). Việc nhận yêu cầu từ client sẽ thông qua các luồng nhập và việc gửi đáp ứng tới server sẽ thông qua luồng xuất.

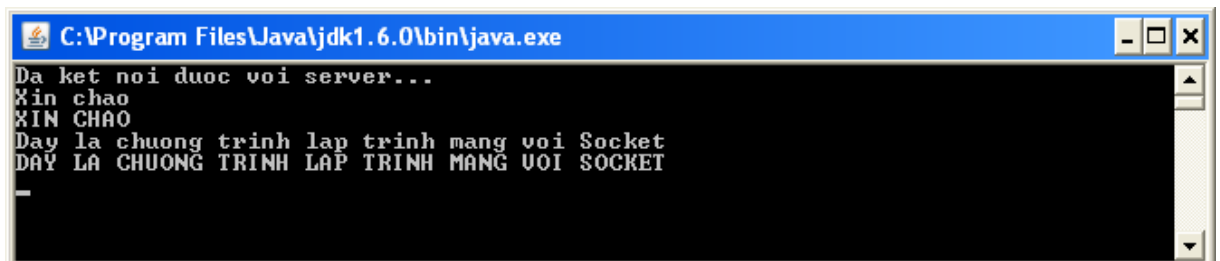
Khởi động chương trình server

```
start java EchoServer1
```



Khởi động client

```
C:\MyJava>start java EchoClient1
```



## 6.7 Ứng dụng đa tuyến đoạn trong lập trình Java

Các server như đã viết ở trên rất đơn giản nhưng nhược điểm của nó là bị hạn chế về mặt hiệu năng vì nó chỉ quản lý được một client tại một thời điểm. Khi khối lượng công việc mà server cần xử lý một yêu cầu của client là quá lớn và không biết trước được thời điểm hoàn thành công việc xử lý thì các server này là không thể chấp nhận được.

Để khắc phục điều này, người ta quản lý mỗi phiên của client bằng một tuyến đoạn riêng, cho phép các server làm việc với nhiều client đồng thời. Server này được gọi là server tương tranh (concurrent server)-server tạo ra một tuyến đoạn để quản lý từng yêu cầu, sau đó tiếp tục lắng nghe các client khác.

Chương trình client/server chúng ta đã xét mở mục 6 và mục 7 là chương trình client/server đơn tuyến đoạn. Các server đơn tuyến đoạn chỉ quản lý được một liên kết tại một thời điểm. Trong thực tế một server có thể phải quản lý nhiều liên kết cùng một lúc. Để thực hiện điều này server chấp nhận các liên kết và chuyển các liên kết này cho từng tuyến đoạn xử lý.

Trong phần dưới đây chúng ta sẽ xem xét cách tiến hành cài đặt một chương trình client/server đa tuyến đoạn.

Chương trình phía server

```
import java.io.*;
import java.net.*;
class EchoServe extends Thread
{
    private Socket socket;
    private BufferedReader in;
    private PrintWriter out;
    public EchoServe (Socket s) throws IOException
    {
        socket = s;
        System.out.println("Serving: "+socket);
        in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        // Cho phép auto-flush:
        out = new PrintWriter(new
OutputStreamWriter(
```

```

        socket.getOutputStream()))), true);
        // Nếu bất kỳ lời gọi nào ở trên đưa ra ngoại lệ
        // thì chương trình gọi có trách nhiệm đóng socket. Ngược
        lại tuyến đoạn sẽ
        // sẽ đóng socket
        start();
    }
    public void run()
    {
        try{
            while (true)
            {
                System.out.println("....Server is waiting...");
                String str = in.readLine();
                if (str.equals("exit") ) break;
                System.out.println("Received: " + str);
                System.out.println("From: "+ socket);
                String upper=str.toUpperCase();
                // gửi lại cho client
                out.println(upper);
            }
            System.out.println("Disconnected with.." + socket);
        } catch (IOException e) {}
        finally{
            try{
                socket.close();
            }
            catch(IOException e) {}
        }
    }
}

public class TCPServer1
{
    static int PORT=0; .
    public static void main(String[] args) throws IOException
    {

```



```

        if (args.length == 1)
        {
            PORT=Integer.parseInt(args[0]); // Nhập số hiệu cổng từ đối
dòng lệnh
        }
        // Tạo một đối tượng Server Socket
        ServerSocket s = new ServerSocket(PORT);
        InetAddress  addr= InetAddress.getLocalHost();
        System.out.println("TCP/Server running on : "+ addr + "
,Port "+s.getLocalPort());
        try {
            while(true) {
                // Phong tỏa cho tới khi có một liên kết đến
                Socket socket = s.accept();
                try {
                    new EchoServe(socket);
                    // Tạo một tuyến đoạn quản lý riêng từng liên kết
                } catch(IOException e) {
                    socket.close();
                }
            }
        }
        finally {
            s.close();
        }
    }
}

```

### Chương trình phía client

```

import java.net.*;
import java.io.*;
public class TCPClient1
{
    public static void main(String[] args) throws IOException
    {
        if (args.length != 2) {
            System.out.println("Sử dụng: java TCPClient  hostid port#");

```

```

        System.exit(0);
    }
    try{
        InetAddress addr = InetAddress.getByName(args[0]);
        Socket socket = new Socket(addr, Integer.parseInt(args[1]));
        try {
            System.out.println("socket = " + socket);
            BufferedReader in = new BufferedReader(new
InputStreamReader(
                socket.getInputStream()));
            // Output is automatically flushed by PrintWriter:
            PrintWriter out =new PrintWriter(new BufferedWriter(
                new OutputStreamWriter(socket.getOutputStream())),true);
            // Đọc dòng ký tự từ bàn phím
            DataInputStream myinput = new DataInputStream(new
BufferedInputStream(System.in));
            try
            {
                for(;;){
                    System.out.println("Type anything followed by RETURN,
or Exit to terminate the program.");
                    String strin=myinput.readLine();
                    // Quit if the user typed ctrl+D
                    if (strin.equals("exit")) break;
                    else
                        out.println(strin);           // Send the message
                    String strout = in.readLine();    // Recive it back
                    if ( strin.length()==strout.length())
                    { // Compare Both Strings
                        System.out.println("Received: "+strout);
                    } else
                        System.out.println("Echo bad -- string unequal"+
strout);
                    } // of for ;;
                }catch (IOException e){
                    e.printStackTrace();
                }
            }
        }
    }

```

```

        // User is exiting
    }
    finally
    {
        System.out.println("EOF...exit");
        socket.close();
    }
}
catch (UnknownHostException e)
{
    System.err.println("Can't find host");
    System.exit(1);
} catch (SocketException e)
{
    System.err.println("Can't open socket");
    e.printStackTrace();
    System.exit(1);
}
}
}

```

## 6.8 Kết luận

Chúng ta đã tìm hiểu cách lập trình mạng cho giao thức TCP. Các Socket còn được gọi là socket luồng vì để gửi và nhận dữ liệu đều được tiến hành thông qua việc đọc ghi các luồng. Ta đọc cũng đã tìm hiểu cơ chế hoạt động của socket và cách thức lập các chương trình server và client. Ngoài ra, chương này cũng đã giải thích tạo sao cần có cài đặt server đa tuyến đoạn và tìm hiểu cách thức để lập các chương trình client/server đa tuyến đoạn. Trong chương tiếp theo chúng ta sẽ học cách xây dựng một chương trình client/server cho giao thức UDP, một giao thức gần với giao thức TCP.

## **CHƯƠNG 7. LẬP TRÌNH ỨNG DỤNG CHO GIAO THỨC UDP**

### **7.1 Tổng quan về giao thức UDP**

TCP/IP là một họ các giao thức được gọi là họ giao thức IP, bao gồm bốn tầng. Cần nhớ rằng TCP/IP không phải là một giao thức mà thực sự là một họ các giao thức, và bao gồm các giao thức mức thấp khác như IP, TCP, và UDP. UDP nằm ở tầng giao vận, phía trên giao thức IP. Tầng giao vận cung cấp khả năng truyền tin giữa các mạng thông qua các gateway. Nó sử dụng các địa chỉ IP để gửi các gói tin trên Internet hoặc trên mạng thông

qua các trình điều khiển thiết bị khác nhau. TCP và UDP là một phần của họ giao thức TCP/IP; mỗi giao thức có những ưu và nhược điểm riêng của nó.

Giao thức UDP là giao thức đơn giản, phi liên kết và cung cấp dịch vụ trên tầng giao vận với tốc độ nhanh. Nó hỗ trợ liên kết một-nhiều và thường được sử dụng thường xuyên trong liên kết một-nhiều bằng cách sử dụng các datagram multicast và unicast.

Giao thức IP là giao thức cơ bản của Internet. TCP và UDP đều là hai giao thức tầng giao thức vận trên cơ sở của giao thức IP. Hình dưới đây chỉ ra cách ánh xạ mô hình OSI ánh xạ vào kiến trúc TCP/IP và họ giao thức TCP/IP.

	Các tầng OSI	Họ giao thức TCP	TCP/IP Stack				
7	Tầng ứng dụng	Tầng ứng dụng	HTTP	FTP	SMTP	RIP	DNS
6	Tầng trình diễn						
5	Tầng phiên						
4	Tầng giao vận	Tầng giao vận	TCP			UDP	
3	Tầng mạng	Tầng Internet	ICMP,IP, IGMP				
2	Tầng liên kết dữ liệu	Tầng mạng	Ethernet, ATM, Frame Relay,..				
1	Tầng vật lý						

### 7.1.1 Một số thuật ngữ UDP

Trước khi kiểm tra xem giao thức UDP hoạt động như thế nào, chúng ta cần làm quen với một số thuật ngữ. Trong phần dưới đây, chúng ta sẽ định nghĩa một số thuật ngữ cơ bản có liên quan đến giao thức UDP.

- Packet

Trong truyền số liệu, một packet là một dãy các số nhị phân, biểu diễn dữ liệu và các tín hiệu điều khiển, các gói tin này được chuyển đi và chuyển tới tới host. Trong gói tin, thông tin được sắp xếp theo một khuôn dạng cụ thể.

- Datagram

Một datagram là một gói tin độc lập, tự chứa, mang đầy đủ dữ liệu để định tuyến từ nguồn tới đích mà không cần thông tin thêm.

- MTU

MTU là viết tắt của Maximum Transmission Unit. MTU là một đặc trưng của tầng liên kết mô tả số byte dữ liệu tối đa có thể truyền trong một gói tin. Mặt khác, MTU là gói dữ liệu

lớn nhất mà môi trường mạng cho trước có thể truyền. Ví dụ, Ethernet có MTU cố định là 1500 byte. Trong UDP, nếu kích thước của một datagram lớn hơn MTU, IP sẽ thực hiện phân đoạn, chia datagram thành các phần nhỏ hơn (các đoạn), vì vậy mỗi đoạn nhỏ có kích thước nhỏ hơn MTU.

- Port

UDP sử dụng các cổng để ánh xạ dữ liệu đến vào một tiến trình cụ thể đang chạy trên một máy tính. UDP định đường đi cho packet tại vị trí xác định bằng cách sử dụng số hiệu cổng được xác định trong header của datagram. Các cổng được biểu diễn bởi các số 16-bit, vì thế các cổng nằm trong dải từ 0 đến 65535. Các cổng cũng được xem như là các điểm cuối của các liên kết logic, và được chia thành ba loại sau:

- Các cổng phổ biến: Từ 0 đến 1023
- Các cổng đã đăng ký: 1024 đến 49151
- Các cổng động/dành riêng 49152 đến 65535

Chú ý rằng các cổng UDP có thể nhận nhiều hơn một thông điệp ở một thời điểm. Trong một số trường hợp, các dịch vụ TCP và UDP có thể sử dụng cùng một số hiệu cổng, như 7 (Echo) hoặc trên cổng 23 (Telnet).

UDP có các cổng thông dụng sau:

Cổng UDP	Mô tả
15	Netstat- Network Status-Tình trạng mạng
53	DNS-Domain Name Server
69	TFTP-Trivial File Transfer Protocol Giao thức truyền tệp thông thường
137	NetBIOS Name Service
138	Dịch vụ Datagram NetBIOS
161	SNMP

Bảng 7.2

- TTL (Time To Live)

Giá trị TTL cho phép chúng ta thiết lập một giới hạn trên của các router mà một datagram có thể đi qua. Giá trị TTL ngăn ngừa các gói tin khỏi bị kẹt trong các vòng lặp định tuyến vô hạn. TTL được khởi tạo bởi phía gửi và giá trị được giảm đi bởi mỗi router quản lý datagram. Khi TTL bằng 0, datagram bị loại bỏ.

- Multicasting

Multicasting là phương pháp dựa trên chuẩn có tính chất mở để phân phối các thông tin giống nhau đến nhiều người dùng. Multicasting là một đặc trưng chính của giao thức UDP. Multicasting cho phép chúng ta truyền tin theo kiểu một nhiều, ví dụ gửi tin hoặc thư điện tử tới nhiều người nhận, đài phát thanh trên Internet, hoặc các chương trình demo trực tuyến.

### *7.1.2 Hoạt động của giao thức UDP*

Khi một ứng dụng dựa trên giao thức UDP gửi dữ liệu tới một host khác trên mạng, UDP thêm vào một header có độ dài 8 byte chứa các số hiệu cổng nguồn và đích, cùng với tổng chiều dài dữ liệu và thông tin checksum. IP thêm vào header của riêng nó vào đầu mỗi datagram UDP để tạo lên một datagram IP:

### *7.1.3 Các nhược điểm của giao thức UDP*

So với giao thức TCP, UDP có những nhược điểm sau:

- Thiếu các tín hiệu bắt tay. Trước khi gửi một đoạn, UDP không gửi các tín hiệu bắt tay giữa bên gửi và bên nhận. Vì thế phía gửi không có cách nào để biết datagram đã đến đích hay chưa. Do vậy, UDP không đảm bảo việc dữ liệu đã đến đích hay chưa.
- Sử dụng các phiên. Để TCP là hướng liên kết, các phiên được duy trì giữa các host. TCP sử dụng các chỉ số phiên (session ID) để duy trì các liên kết giữa hai host. UDP không hỗ trợ bất kỳ phiên nào do bản chất phi liên kết của nó.
- Độ tin cậy. UDP không đảm bảo rằng chỉ có một bản sao dữ liệu tới đích. Để gửi dữ liệu tới các hệ thống cuối, UDP phân chia dữ liệu thành các đoạn nhỏ. UDP không đảm bảo rằng các đoạn này sẽ đến đích đúng thứ tự như chúng đã được tạo ra ở nguồn. Ngược lại, TCP sử dụng các số thứ tự cùng với số hiệu cổng và các gói tin xác thực thường xuyên, điều này đảm bảo rằng các gói tin đến đích đúng thứ tự mà nó đã được tạo ra.
- Bảo mật. TCP có tính bảo mật cao hơn UDP. Trong nhiều tổ chức, firewall và router cấm các gói tin UDP, điều này là vì các hacker thường sử dụng các cổng UDP.

- Kiểm soát luồng. UDP không có kiểm soát luồng; kết quả là, một ứng dụng UDP được thiết kế tồi có thể làm giảm băng thông của mạng.

#### 7.1.4 Các ưu điểm của UDP

- Không cần thiết lập liên kết. UDP là giao thức phi liên kết, vì thế không cần phải thiết lập liên kết. Vì UDP không sử dụng các tín hiệu handshaking, nên có thể tránh được thời gian trễ. Đó chính là lý do tại sao DNS thường sử dụng giao thức UDP hơn là TCP-DNS sẽ chậm hơn rất nhiều khi dùng TCP.
- Tốc độ. UDP nhanh hơn so với TCP. Bởi vì điều này, nhiều ứng dụng thường được cài đặt trên giao thức UDP hơn so với giao thức TCP.
- Hỗ trợ hình trạng (Topology). UDP hỗ trợ các liên kết 1-1, 1-n, ngược lại TCP chỉ hỗ trợ liên kết 1-1.
- Kích thước header. UDP chỉ có 8 byte header cho mỗi đoạn, ngược lại TCP cần các header 20 byte, vì vậy sử dụng băng thông ít hơn.

Bảng dưới đây tổng kết những sự khác nhau giữa hai giao thức TCP và UDP:

Các đặc trưng	UDP	TCP
Hướng liên kết	Không	Có
Sử dụng phiên	Không	Có
Độ tin cậy	Không	Có
Xác thực	Không	Có
Đánh thứ tự	Không	Có
Điều khiển luồng	Không	Có
Bảo mật	Ít	Nhiều hơn

#### 7.1.5 Khi nào thì nên sử dụng UDP

Rất nhiều ứng dụng trên Internet sử dụng UDP. Dựa trên các ưu và nhược điểm của UDP chúng ta có thể kết luận UDP có ích khi:

- Sử dụng cho các phương thức truyền broadcasting và multicasting khi chúng ta muốn truyền tin với nhiều host.
- Kích thước datagram nhỏ và trình tự đoạn là không quan trọng



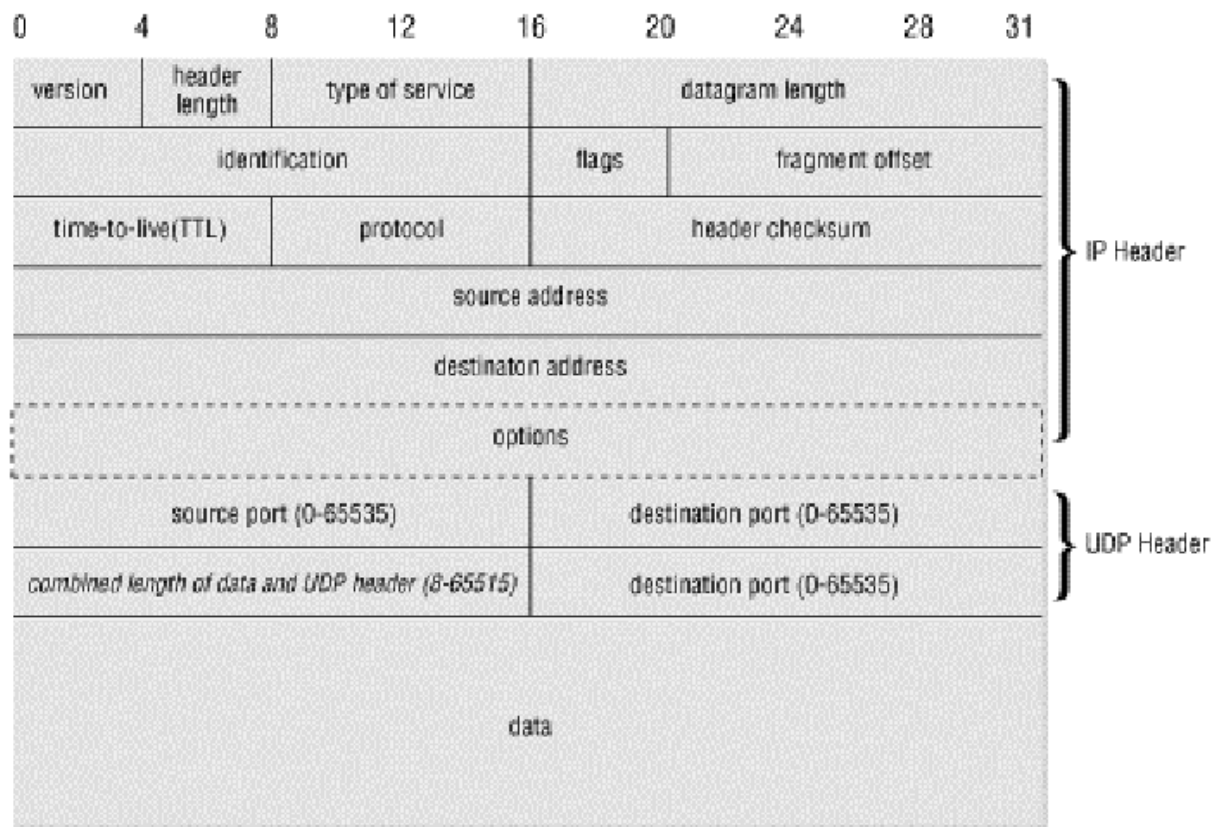
- Không cần thiết lập liên kết
- Ứng dụng không gửi các dữ liệu quan trọng
- Không cần truyền lại các gói tin
- Bảng thông của mạng đóng vai trò quan trọng

Việc cài đặt ứng dụng UDP trong Java cần có hai lớp là `DatagramPacket` và `DatagramSocket`. `DatagramPacket` đóng gói các byte dữ liệu vào các gói tin UDP được gọi là datagram và cho phép ta mở các datagram khi nhận được. Một `DatagramSocket` đồng thời thực hiện cả hai nhiệm vụ nhận và gửi gói tin. Để gửi dữ liệu, ta đặt dữ liệu trong một `DatagramPacket` và gửi gói tin bằng cách sử dụng `DatagramSocket`. Để nhận dữ liệu, ta nhận một đối tượng `DatagramPacket` từ `DatagramSocket` và sau đó đọc nội dung của gói tin.

UDP không có bất kỳ khái niệm nào về liên kết giữa hai host. Một socket gửi tất cả dữ liệu tới một cổng hoặc nhận tất cả dữ liệu từ một cổng mà không cần quan tâm host nào gửi. Một `DatagramSocket` có thể gửi dữ liệu tới nhiều host độc lập hoặc nhận dữ liệu từ nhiều host độc lập. Socket không dành riêng cho một liên kết cụ thể thể nào cả như trong giao thức TCP. Các socket TCP xem liên kết mạng như là một luồng: ta gửi và nhận dữ liệu với các luồng nhập và luồng xuất nhận được từ socket. UDP không cho phép điều này; ta phải làm việc với từng gói tin. Tất cả dữ liệu được đặt trong datagram được gửi đi dưới dạng một gói tin. Gói tin này cũng có thể nhận được bởi một nhóm hoặc cũng có thể bị mất. Một gói tin không nhất thiết phải liên quan đến gói tin tiếp theo. Cho trước hai gói tin, không có cách nào để biết được gói tin nào được gửi trước và gói tin nào được gửi sau.

## 7.2 Lớp `DatagramPacket`

Các datagram UDP đưa rất ít thông tin vào datagram IP. Header UDP chỉ đưa tám byte vào header IP. Header UDP bao gồm số hiệu cổng nguồn và đích, chiều dài của dữ liệu và header UDP, tiếp đến là một checksum tùy chọn. Vì mỗi cổng được biểu diễn bằng hai byte nên tổng số cổng UDP trên một host sẽ là 65536. Chiều dài cũng được biểu diễn bằng hai byte nên số byte trong datagram tối đa sẽ là 65536 trừ đi tám 8 byte dành cho phần thông tin header.



Trong Java, một datagram UDP được biểu diễn bởi lớp DatagramPacket:

- `public final class DatagramPacket extends Object`

Lớp này cung cấp các phương thức để nhận và thiết lập các địa chỉ nguồn, đích từ header IP, nhận và thiết lập các thông tin về cổng nguồn và đích, nhận và thiết lập độ dài dữ liệu. Các trường thông tin còn lại không thể truy nhập được từ mã Java thuần túy.

DatagramPacket sử dụng các constructor khác nhau tùy thuộc vào gói tin được sử dụng để gửi hay nhận dữ liệu.

### 7.2.1 Các constructor để nhận datagram

Hai constructor tạo ra các đối tượng DatagramSocket mới để nhận dữ liệu từ mạng:

- `public DatagramPacket(byte[] b, int length)`
- `public DatagramPacket(byte[] b, int offset, int length)`

Khi một socket nhận một datagram, nó lưu trữ phần dữ liệu của datagram ở trong vùng đệm b bắt đầu tại vị trí b[0] và tiếp tục cho tới khi gói tin được lưu trữ hoàn toàn hoặc cho tới khi lưu trữ hết length byte. Nếu sử dụng constructor thứ hai, thì dữ liệu được lưu trữ bắt

đầu từ vị trí `b[offset]`. Chiều dài của `b` phải nhỏ hơn hoặc bằng `b.length - offset`. Nếu ta xây dựng một `DatagramPacket` có chiều dài vượt quá chiều dài của vùng đệm thì constructor sẽ đưa ra ngoại lệ `IllegalArgumentException`. Đây là kiểu ngoại lệ `RuntimeException` nên chương trình của ta không cần thiết phải đón bắt ngoại lệ này.

Ví dụ, xây dựng một `DatagramPacket` để nhận dữ liệu có kích thước lên tới 8912 byte

```
byte b[]=new byte[8912];  
DatagramPacket dp=new DatagramPacket(b,b.length);
```

### 7.2.2 Constructor để gửi các datagram

Bốn constructor tạo các đối tượng `DatagramPacket` mới để gửi dữ liệu trên mạng:

- `public DatagramPacket(byte[] b, int length, InetAddress dc, int port)`
- `public DatagramPacket(byte[] b, int offset, int length, InetAddress dc, int port)`
- `public DatagramPacket(byte[] b, int length, SocketAddress dc, int port)`
- `public DatagramPacket(byte[] b, int offset, int length, SocketAddress dc, int port)`

Mỗi constructor tạo ra một `DatagramPacket` mới để được gửi đi tới một host khác. Gói tin được điền đầy dữ liệu với chiều dài là `length` byte bắt đầu từ vị trí `offset` hoặc vị trí 0 nếu `offset` không được sử dụng.

Ví dụ để gửi đi một chuỗi ký tự đến một host khác như sau:

```
String s="This is an example of UDP Programming";  
byte[] b= s.getBytes();  
try{  
    InetAddress dc=InetAddress.getByName("www.vnn.vn");  
    int port =7;  
    DatagramPacket dp=new DatagramPacket(b,b.length,dc,port);  
    //Gửi gói tin  
}  
catch(IOException e){  
    System.err.println(e);  
}
```

Công việc khó khăn nhất trong việc tạo ra một đối tượng `DatagramPacket` chính là việc chuyển đổi dữ liệu thành một mảng byte. Đoạn mã trên chuyển đổi một chuỗi ký tự thành một mảng byte để gửi dữ liệu đi

### 7.2.3 Các phương thức nhận các thông tin từ DatagramPacket

DatagramPacket có sáu phương thức để tìm các phần khác nhau của một datagram: dữ liệu thực sự cộng với một số trường header. Các phương thức này thường được sử dụng cho các datagram nhận được từ mạng.

- `public InetAddress getAddress()`

Phương thức `getAddress()` trả về một đối tượng `InetAddress` chứa địa chỉ IP của host ở xa. Nếu datagram được nhận từ Internet, địa chỉ trả về chính là địa chỉ của máy đã gửi datagram (địa chỉ nguồn). Mặt khác nếu datagram được tạo cục bộ để được gửi tới máy ở xa, phương thức này trả về địa chỉ của host mà datagram được đánh địa chỉ.

- `public int getPort()`

Phương thức `getPort()` trả về một số nguyên xác định cổng trên host ở xa. Nếu datagram được nhận từ Internet thì cổng này là cổng trên host đã gửi gói tin đi.

- `public SocketAddress()`

Phương thức này trả về một đối tượng `SocketAddress` chứa địa chỉ IP và số hiệu cổng của host ở xa.

- `public byte[] getData()`

Phương thức `getData()` trả về một mảng byte chứa dữ liệu từ datagram. Thông thường cần phải chuyển các byte này thành một dạng dữ liệu khác trước khi chương trình xử lý dữ liệu. Một cách để thực hiện điều này là chuyển đổi mảng byte thành một đối tượng `String` sử dụng constructor sau đây:

- `public String(byte[] buffer,String encoding)`

Tham số đầu tiên, `buffer`, là mảng các byte chứa dữ liệu từ datagram. Tham số thứ hai cho biết cách thức mã hóa xâu ký tự. Cho trước một `DatagramPacket dp` được nhận từ mạng, ta có thể chuyển đổi nó thành xâu ký tự như sau:

```
String s=new String(dp.getData(),"ASCII");
```

Nếu datagram không chứa văn bản, việc chuyển đổi nó thành dữ liệu Java khó khăn hơn nhiều. Một cách tiếp cận là chuyển đổi mảng byte được trả về bởi phương thức `getData()` thành luồng `ByteArrayInputStream` bằng cách sử dụng constructor này:

- `public ByteArrayInputStream(byte[] b, int offset, int length)`

b là mảng byte được sử dụng như là một luồng nhập InputStream

- `public int getLength()`

Phương thức `getLength()` trả về số bytes dữ liệu có trong một datagram.

- `public getOffset()`

Phương thức này trả về vị trí trong mảng được trả về bởi phương thức `getData()` mà từ đó dữ liệu trong datagram xuất phát.

Các phương thức thiết lập giá trị cho các trường thông tin

Sáu constructor ở trên là đủ để tạo lập ra các datagram. Tuy nhiên, Java cung cấp một số phương thức để thay đổi dữ liệu, địa chỉ của máy ở xa, và cổng trên máy ở xa sau khi datagram đã được tạo ra. Trong một số trường hợp việc sử dụng lại các `DatagramPacket` đã có sẵn sẽ nhanh hơn việc tạo mới các đối tượng này.

- `public void setData(byte[] b)`: Phương thức này thay đổi dữ liệu của datagram
- `public void setData(byte[] b, int offset, int length)`

Phương thức này đưa ra giải pháp để gửi một khối lượng dữ liệu lớn. Thay vì gửi toàn bộ dữ liệu trong mảng, ta có thể gửi dữ liệu trong từng đoạn của mảng tại mỗi thời điểm.

Ví dụ đoạn mã sau đây sẽ gửi dữ liệu theo từng đoạn 512 byte:

```
int offset=0;
DatagramPacket dp=new DatagramPacket(b,offset,512);
int bytesSent=0;
while(bytesSent<b.length)
{
    ds.send(dp);
    bytesSent+=dp.getLength();
    int bytesToSend=b.length-bytesSent;
    int size=(bytesToSend>512):512:bytesToSend;
    dp.setData(b,bytesSent,512);
}
```

- `public void setAddress(InetAddress dc)`

Phương thức `setAddress()` thay đổi địa chỉ của máy mà ta sẽ gửi gói tin tới. Điều này sẽ cho phép ta gửi cùng một datagram đến nhiều nơi nhận.

- `public void setPort(int port)`

Phương thức này thay đổi số hiệu cổng gửi tới của gói tin.

- public void setAddress(SocketAddress sa)
- public void setLength(int length)

Phương thức này thay đổi số byte dữ liệu có thể đặt trong vùng đệm.

### 7.3 Lớp DatagramSocket

Để gửi hoặc nhận một DatagramPacket, bạn phải mở một DatagramSocket. Trong Java, một datagram socket được tạo ra và được truy xuất thông qua đối tượng DatagramSocket

```
public class DatagramSocket extends Object
```

Tất cả các datagram được gắn với một cổng cục bộ, cổng này được sử dụng để lắng nghe các datagram đến hoặc được đặt trên các header của các datagram sẽ gửi đi. Nếu ta viết một client thì không cần phải quan tâm đến số hiệu cổng cục bộ là bao nhiêu

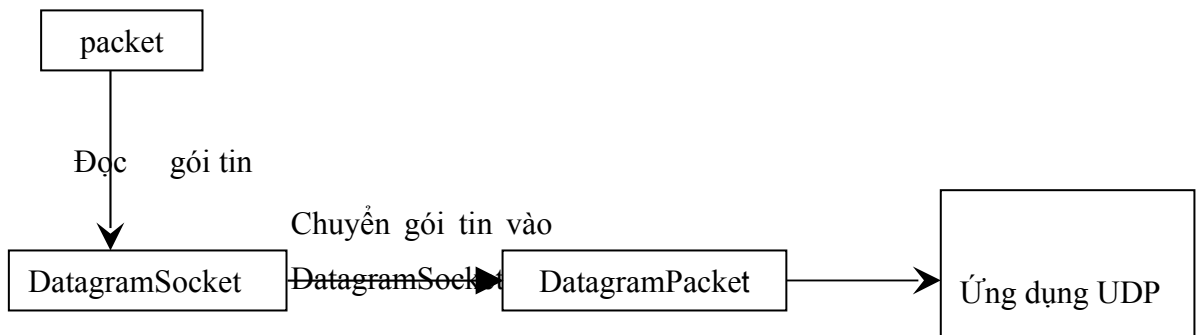
DatagramSocket được sử dụng để gửi và nhận các gói tin UDP. Nó cung cấp các phương thức để gửi và nhận các gói tin, cũng như xác định một giá trị timeout khi sử dụng phương pháp vào ra không phong tỏa (non blocking I/O), kiểm tra và sửa đổi kích thước tối đa của gói tin UDP, đóng socket.

Các phương thức

- void close(): đóng một liên kết và giải phóng nó khỏi cổng cục bộ.
- void connect(InetAddress remote\_address, int remote\_port)-
- InetAddress getInetAddress(): phương thức này trả về địa chỉ remote mà socket kết nối tới, hoặc giá trị null nếu không tồn tại liên kết.
- InetAddress getLocalAddress(): trả về địa chỉ cục bộ
- int getSoTimeout() trả về giá trị tùy chọn timeout của socket. Giá trị này xác định thời gian mà thao tác đọc sẽ phong tỏa trước khi nó đưa ra ngoại lệ InterruptedException. Ở chế độ mặc định, giá trị này bằng 0, chỉ ra rằng vào ra không phong tỏa được sử dụng.
- void receive(DatagramPacket dp) throws IOException: phương thức đọc một gói tin UDP và lưu nội dung trong packet xác định.
- void send(DatagramSocket dp) throws IOException: phương thức gửi một gói tin

- void setSoTimeOut(int timeout): thiết lập giá trị tùy chọn của socket.

#### 7.4 Nhận các gói tin



Trước khi một ứng dụng có thể đọc các gói tin UDP được gửi bởi các máy ở xa, nó phải gán một socket với một cổng UDP bằng cách sử dụng DatagramSocket, và tạo ra một DatagramPacket sẽ đóng vai trò như là một bộ chứa cho dữ liệu của gói tin UDP. Hình vẽ dưới đây chỉ ra mối quan hệ giữa một gói tin UDP với các lớp Java khác nhau được sử dụng để xử lý nó và các ứng dụng thực tế.

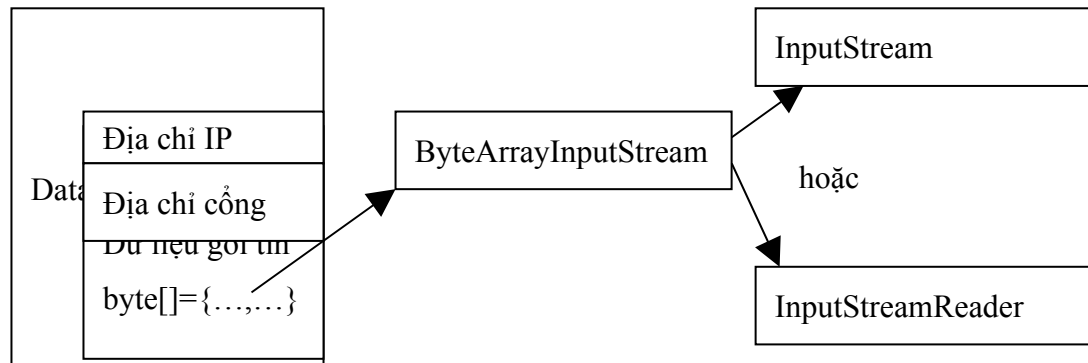
Khi một ứng dụng muốn đọc các gói tin UDP, nó gọi phương thức DatagramSocket.receive(), phương thức này sao chép gói tin UDP vào một DatagramPacket xác định. Xử lý nội dung nói tin và tiến trình lặp lại khi cần

```

DatagramPacket dp=new DatagramPacket(new byte[256],256);
DatagramSocket ds=new DatagramSocket(2000);
boolean finished=false;
while(!finished)
{
    ds.receive(dp);
    //Xử lý gói tin
}
ds.close();
  
```

Khi xử lý gói tin ứng dụng phải làm việc trực tiếp với một mảng byte. Tuy nhiên nếu ứng dụng là đọc văn bản thì ta có thể sử dụng các lớp từ gói vào ra để chuyển đổi giữa mảng byte và luồng stream và reader. Bằng cách gắn kết luồng nhập ByteArrayInputStream với nội dung của một datagram và sau đó kết nối với một kiểu luồng khác, khi đó bạn có thể truy xuất tới nội dung của gói UDP một cách dễ dàng. Rất nhiều người lập trình thích dùng

các luồng vào ra I/O để xử lý dữ liệu, bằng cách sử dụng luồng `DataInputStream` hoặc `BufferedReader` để truy xuất tới nội dung của các mảng byte.



Ví dụ, để gắn kết một luồng `DataInputStream` với nội dung của một `DatagramPacket`, ta sử dụng đoạn mã sau:

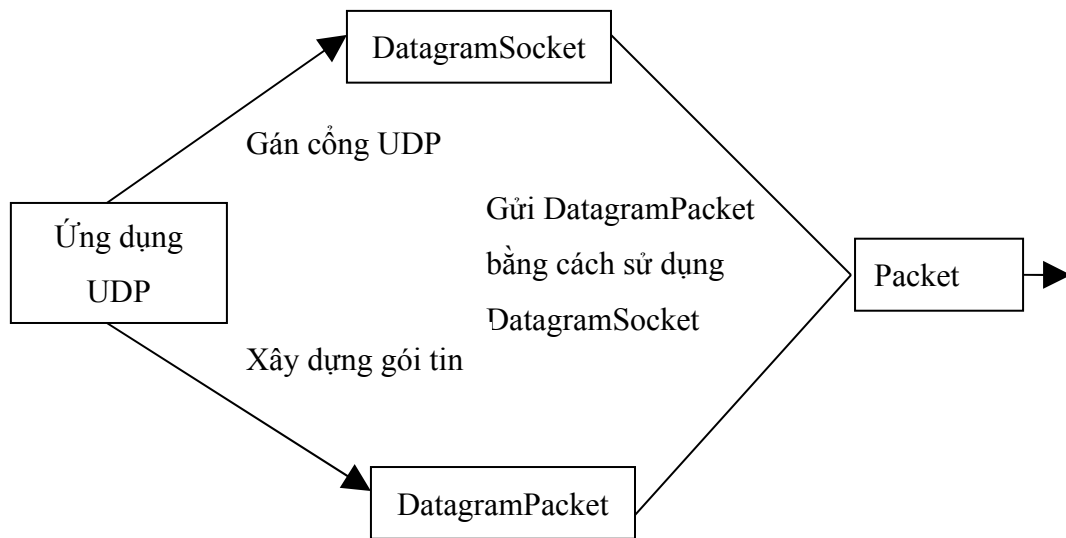
```

ByteArrayInputStream bis=new ByteArrayInputStream(dp.getData());
DataInputStream dis=new DataInputStream(bis);
//đọc nội dung của gói tin UDP
  
```

## 7.5 Gửi các gói tin

Lớp `DatagramSocket` cũng được sử dụng để gửi các gói tin. Khi gửi gói tin, ứng dụng phải tạo ra một `DatagramPacket`, thiết lập địa chỉ và thông tin cổng, và ghi dữ liệu cần truyền vào mảng byte. Nếu muốn gửi thông tin phúc đáp thì ta cũng đã biết địa chỉ và số hiệu cổng của gói tin nhận được. Mỗi khi gói tin sẵn sàng để gửi, ta sử dụng phương thức `send()` của lớp `DatagramSocket` để gửi gói tin đi.





```

//Socket lắng nghe các gói tin đến trên cổng 2000
DatagramSocket socket = new DatagramSocket(2000);
DatagramPacket packet = new DatagramPacket (new byte[256], 256);
packet.setAddress ( InetAddress.getByName ( somehost ) );
packet.setPort ( 2000 );
boolean finished = false;
while !finished )
{
    // Ghi dữ liệu vào vùng đệm buffer
    .....
    socket.send (packet);
    // Thực hiện hành động nào đó, chẳng hạn như đọc gói tin khác
    hoặc kiểm tra xemor
    // còn gói tin nào cần gửi đi hay không
    .....
}
socket.close();

```

## 7.6 Ví dụ minh họa giao thức UDP

Để minh họa các gói tin UDP được gửi và nhận như thế nào, chúng ta sẽ viết, biên dịch và chạy ứng dụng sau.

Viết chương trình theo mô hình Client/Server để:

Client thực hiện các thao tác sau đây:

- Client gửi một chuỗi ký tự do người dùng nhập từ bàn phím cho server
- Client nhận thông tin phản hồi trở lại từ Server và hiển thị thông tin đó trên màn hình

Server thực hiện các thao tác sau:

- Server nhận chuỗi ký tự do client gửi tới và in lên màn hình
- Server biến đổi chuỗi ký tự thành chữ hoa và gửi trở lại cho Client

```
import java.net.*;
import java.io.*;
public class UDPClient
{
    public final static int CONG_MAC_DINH=9;
    public static void main(String args[])
    {
        String hostname;
        int port=CONG_MAC_DINH;
        if(args.length>0)
        {
            hostname=args[0];
            try{
            }
            catch(Exception e){
                port =Integer.parseInt(args[1]);
            }
        }
        else
        {
            hostname="127.0.0.1";
        }
        try{
            InetAddress dc=InetAddress.getByName(hostname);
```

```

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

        DatagramSocket ds =new DatagramSocket(port);
        while(true){
            String line=userInput.readLine();
            if(line.equals("exit"))break;
            byte[] data=line.getBytes();
            DatagramPacket                                dp=new
DatagramPacket(data,data.length,dc,port);
            ds.send(dp);
            dp.setLength(65507);
            ds.receive(dp);
            ByteArrayInputStream                                bis                                =new
ByteArrayInputStream(dp.getData());
            BufferedReader dis =new    BufferedReader (new
InputStreamReader(bis));
            System.out.println(dis.readLine());

        }
    }
    catch(UnknownHostException e)
    {
        System.err.println(e);
    }
    catch(IOException e)
    {
        System.err.println(e);
    }
}
}

```

```

import java.net.*;
import java.io.*;
public class UDPServer
{
    public final static int CONG_MAC_DINH=9;

```

```

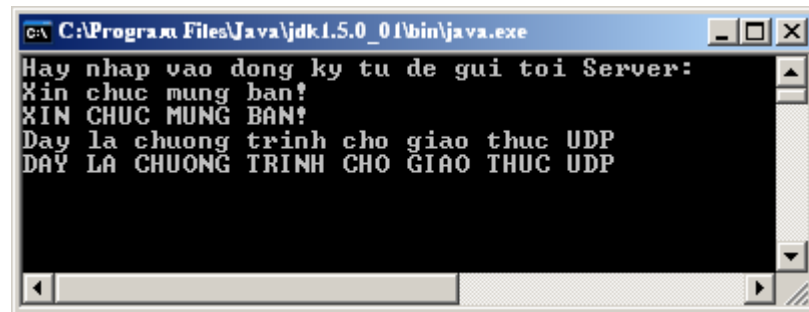
public static void main(String args[])
{
    int port=CONG_MAC_DINH;
    try{
    }
    catch(Exception e){
        port =Integer.parseInt(args[1]);
    }
    try{
        DatagramSocket ds =new DatagramSocket(port);
        DatagramPacket dp=new DatagramPacket(new
byte[65507],65507);
        while(true){
            ds.receive(dp);
            ByteArrayInputStream bis =new
ByteArrayInputStream(dp.getData());
            BufferedReader dis;
            dis =new BufferedReader(new InputStreamReader(bis));
            String s=dis.readLine();
            System.out.println(s);
            s.toUpperCase();
            dp.setData(s.getBytes());
            dp.setLength(s.length());
            dp.setAddress(dp.getAddress());
            dp.setPort(dp.getPort());
            ds.send(dp);
        }
    }
    catch(UnknownHostException e)
    {
        System.err.println(e);
    }
    catch(IOException e)
    {
        System.err.println(e);
    }
}

```

```
}  
  
}
```

```
C:\>start java UDPServer
```

```
C:\>start java UDPClient
```



### Chương trình Client/Server sử dụng đa tuyến đoạn

```
import java.net.*;  
import java.io.*;  
public abstract class UDPServer extends Thread  
{  
    private int bufferSize;  
    protected DatagramSocket ds;  
  
    public UDPServer(int port, int bufferSize)  
                                                throws SocketException  
    {  
        this.bufferSize=bufferSize;  
        this.ds=new DatagramSocket(port);  
    }  
    public UDPServer(int port) throws SocketException  
    {  
        this(port, 8192);  
    }  
    public void run()  
    {  
        byte[] buffer=new byte[bufferSize];  
        while(true)
```

```

        {
            DatagramPacket dp;
            dp=new DatagramPacket(buffer,buffer.length);
            try{
                ds.receive(dp);
                this.respond(dp);
            }
            catch(IOException e)
            {
                System.err.println(e);
            }
        }
    }
    public abstract void respond(DatagramPacket req);
}

```

## Server Echo

```

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

public class UDPEchoServer extends UDPServer
{
    public final static int DEFAULT_PORT=7;

    public UDPEchoServer()throws SocketException
    {
        super(DEFAULT_PORT);
    }

    public void respond(DatagramPacket dp)
    {
        try{
            DatagramPacket outdp=new
            DatagramPacket(dp.getData(),dp.getLength(),dp.getAddress(),dp.getPort());
        }
    }
}

```

```

        ds.send(outdp);
    }
    catch(IOException e)
    {
        System.err.println(e);
    }
}
public static void main(String[] args)
{
    try
    {
        UDPServer server=new UDPEchoServer();
        server.start();
        System.out.println("Server dang da san sang lang nghe
lien ket...");

    }
    catch(SocketException e)
    {
        System.err.println(e);
    }
}
}

```

## Client

```

import java.net.*;
import java.io.*;
public class ReceiverThread extends Thread
{
    private DatagramSocket ds;
    private boolean stopped=false;
    public ReceiverThread(DatagramSocket ds)
                                                                    throws SocketException
    {
        this.ds=ds;
    }
}

```

```

    }

    public void halt(){
        this.stopped=true;
    }

    public void run()
    {
        byte buffer[]=new byte[65507];

        while(true)
        {
            if(stopped) return;
            DatagramPacket dp;
            dp=new DatagramPacket(buffer,buffer.length);
            try{
                ds.receive(dp);
                String s;
                s=new String(dp.getData(),0,dp.getLength());
                System.out.println(s);
                Thread.yield();
            }
            catch(IOException e)
            {
                System.err.println(e);
            }
        }
    }
}

```

```

import java.net.*;
import java.io.*;
public class SenderThread extends Thread
{
    private InetAddress server;
    private DatagramSocket ds;

```



```

private boolean stopped=false;
private int port;
public SenderThread(InetAddress address, int port)
                                throws SocketException
{
    this.server=address;
    this.port=port;
    this.ds=new DatagramSocket();
    this.ds.connect(server,port);
}
public void halt(){
    this.stopped=true;
}
public DatagramSocket getSocket()
{
    return this.ds;
}
public void run()
{
    try{
        BufferedReader userInput;
userInput=new BufferedReader(new InputStreamReader(System.in));
        while(true)
        {
            if(stopped) return;
            String line=userInput.readLine();
            if(line.equals("exit"))break;
            byte[] data=line.getBytes();
            DatagramPacket dp;
            dp = new DatagramPacket(data,data.length,server,port);
            ds.send(dp);
            Thread.yield();
        }
    }
    catch(IOException e)

```

```

        {
            System.err.println(e);
        }
    }
}

```

## Client Echo

```

import java.net.*;
import java.io.*;
public class UDPEchoClient
{
    public final static int DEFAULT_PORT=7;
    public static void main(String[] args)
    {
        String hostname="localhost";
        int port= DEFAULT_PORT;
        if(args.length>0)
        {
            hostname=args[0];
        }
        try{
            InetAddress ia=InetAddress.getByName(args[0]);
            SenderThread sender=new SenderThread(ia,DEFAULT_PORT);
            sender.start();
            ReceiverThread receiver;
            receiver=new ReceiverThread(sender.getSocket());
            receiver.start();
        }
        catch(UnknownHostException e)
        {
            System.err.println(e);
        }
        catch(SocketException e)
        {
            System.err.println(e);
        }
    }
}

```

```
}  
}
```

## 7.7 Kết luận

Trong chương này, chúng ta đã thảo luận những khái niệm căn bản về giao thức UDP và so sánh nó với giao thức TCP. Chúng ta đã đề cập tới việc cài đặt các chương trình UDP trong Java bằng cách sử dụng hai lớp DatagramPacket và DatagramSocket. Một số chương trình mẫu cũng được giới thiệu để bạn đọc tham khảo và giúp hiểu sâu hơn về các vấn đề lý thuyết.

## CHƯƠNG 8 . TUẦN TỰ HÓA ĐỐI TƯỢNG VÀ ỨNG DỤNG TRONG LẬP TRÌNH MẠNG

### 8.1 Tuần tự hóa đối tượng

#### 8.1.1 Khái niệm

Tuần tự hóa là quá trình chuyển tập hợp các thể hiện đối tượng chứa các tham chiếu tới các đối tượng khác thành một luồng byte tuyến tính, luồng này có thể được gửi đi qua một *Socket*, được lưu vào tệp tin hoặc được xử lý dưới dạng một luồng dữ liệu. Tuần tự hóa là cơ chế được sử dụng bởi RMI để truyền các đối tượng giữa các máy ảo JVM hoặc dưới dạng các tham số trong lời gọi phương thức từ client tới server hoặc là các giá trị trả về từ một lời gọi phương thức.

Tuần tự hóa là một cơ chế đã được xây dựng và được đưa vào các lớp thư viện Java căn bản để chuyển một đồ thị các đối tượng thành các luồng dữ liệu. Luồng dữ liệu này sau đó có thể được xử lý bằng cách lập trình và ta có thể tạo lại các bản sao của đối tượng ban đầu nhờ quá trình ngược lại được gọi là giải tuần tự hóa.

Tuần tự hóa có ba mục đích chính sau

- Cơ chế ổn định: Nếu luồng được sử dụng là *FileOutputStream*, thì dữ liệu sẽ được tự động ghi vào tệp.
- Cơ chế sao chép: Nếu luồng được sử dụng là *ByteArrayObjectOutput*, thì dữ liệu sẽ được ghi vào một mảng byte trong bộ nhớ. Mảng byte này sau đó có thể được sử dụng để tạo ra các bản sao của các đối tượng ban đầu.
- Nếu luồng đang được sử dụng xuất phát từ một *Socket* thì dữ liệu sẽ được tự động gửi đi tới *Socket* nhận, khi đó một chương trình khác sẽ quyết định phải làm gì đối với dữ liệu nhận được.

Một điều quan trọng khác cần chú ý là việc sử dụng tuần tự hóa độc lập với thuật toán tuần tự hóa.

### 8.1.2 1.2. Khả tuần tự (Serializable)

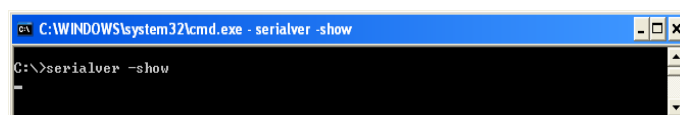
Chỉ có đối tượng thực thi giao diện *Serializable* mới có thể được ghi lại và được phục hồi bởi các tiện ích tuần tự hóa. Giao diện *Serializable* không định nghĩa các thành phần. Nếu một lớp thực thi giao diện *Serializable* thì lớp đó có khả năng tuần tự hóa. Một lớp là khả tuần tự thì tất cả các lớp con của nó cũng là khả tuần tự.

Giao diện *ObjectOutput* thừa kế từ giao diện *DataOutput* và hỗ trợ tuần tự hóa đối tượng. Lớp *ObjectOutputStream* là lớp con của lớp *ObjectOutput* và thực thi giao diện *ObjectOutput*. Nó có nhiệm vụ ghi các đối tượng vào một luồng bằng cách sử dụng phương thức *writeObject(Object obj)*.

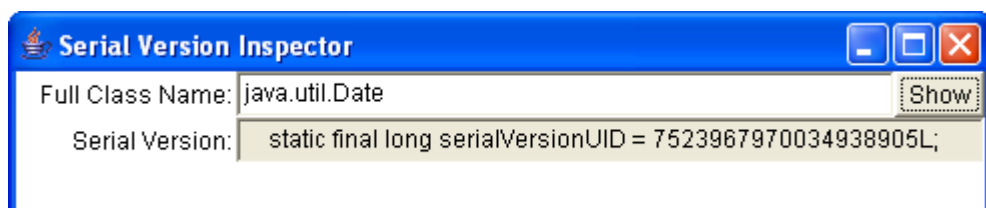
*ObjectInput* thừa kế giao diện *DataInput* và định nghĩa các phương thức. Nó hỗ trợ cho việc tuần tự hóa đối tượng. Phương thức *readObject()* được gọi để giải tuần tự hóa một đối tượng.

*ObjectInputStream* được định nghĩa trong gói *java.io* là một luồng cài đặt cơ chế đọc trạng thái của luồng nhập đối tượng.

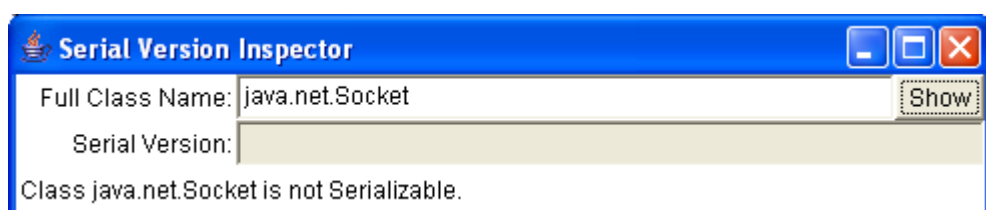
Một vấn đề đặt ra là: liệu mọi lớp trong Java đều có khả năng tuần tự hóa? Câu trả lời là không, bởi vì không cần thiết hoặc sẽ không có ý nghĩa khi tuần tự hóa một số lớp nhất định. Để xác định xem một lớp có khả tuần tự hay không ta sử dụng công cụ serialver có trong bộ JDK.



```
C:\WINDOWS\system32\cmd.exe - serialver -show
C:\>serialver -show
```



Với kết quả trên cho ta thấy lớp này là khả tuần tự. Nhưng không phải mọi lớp trong Java đều khả tuần tự chẳng hạn ta thử kiểm tra với lớp *java.net.Socket*



Khi đó kết quả hiển thị là Class java.net.Socket is not Serializable (Lớp java.net.Socket không khả tuần tự).

### 8.1.3 Xây dựng lớp một lớp khả tuần tự

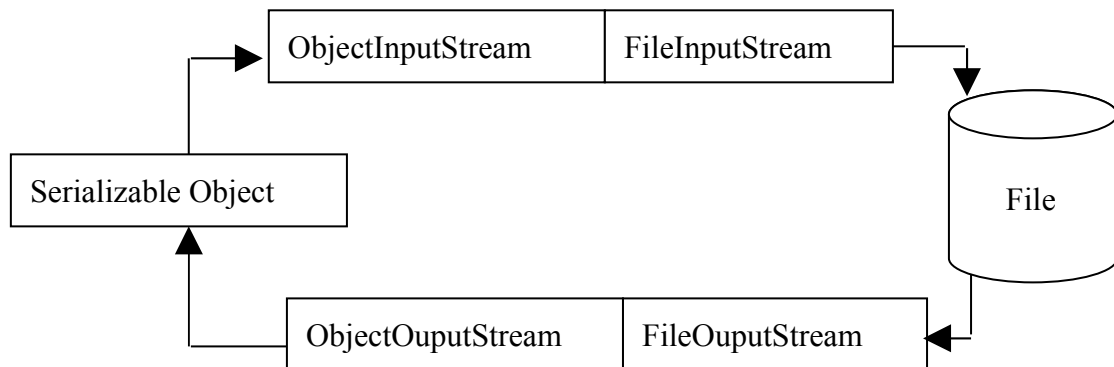
Đối với các lớp do người lập trình định nghĩa ta phải khai báo để báo hiệu cho hệ thống biết nó có khả tuần tự hay không. Một lớp do người dùng định nghĩa có khả năng tuần tự hóa khi lớp đó thực thi giao diện *Serializable*. Trong ví dụ dưới đây ta định nghĩa lớp *Point* để lớp này có khả năng tuần tự hóa.

```
public class Point implements Serializable
{
    private double x,y;
    public Point(double x,double y){
        this.x=x;
        this.y=y;
    }
    public double getX(){
        return x;
    }
    public double getY(){
        return y;
    }
    public void move(double dx,double dy){
        x+=dx;
        y+=dy;
    }
    public void print(){
        System.out.println("Toa do cua diem la:");
        System.out.println("Toa do x="+x);
        System.out.println("Toa do y="+y);
    }
}
```

### 8.1.4 Cơ chế đọc và ghi đối tượng trên thiết bị lưu trữ ngoài

Chúng ta đều biết rằng tất cả các thao tác nhập và xuất dữ liệu trong Java thực chất là việc đọc và ghi trên các luồng dữ liệu vào và luồng dữ liệu ra. Việc đọc và ghi đối tượng trên

thiết bị lưu trữ ngoài cũng không phải là một ngoại lệ. Chúng ta có thể thấy được cơ chế này qua hình 4.



Giả sử đối tượng obj là một đối tượng khả tuần tự. Bản thân đối tượng này có thể đã là khả tuần tự hoặc do người lập trình định nghĩa nên thuộc tính khả tuần tự cho nó.

Cơ chế ghi đối tượng được tiến hành rất đơn giản: Trước tiên ta tạo ra một tệp để ghi thông tin, thực chất là tạo ra đối tượng *FileOutputStream*, sau đó ta tạo ra một luồng ghi đối tượng *ObjectOutputStream* gắn với luồng ghi tệp và gắn kết hai luồng với nhau. Việc ghi đối tượng được thực hiện bởi *phương thức writeObject()*.

```
FileOutputStream fos=new FileOutputStream("date.out");
ObjectOutputStream oos=new ObjectOutputStream(fos);
Date d=new Date();
oos.writeObject(d);
```

Quá trình trên được gọi là quá trình tuần tự hóa.

Chúng ta nhận thấy rằng để phục hồi lại trạng thái của một đối tượng ta phải mở một tệp để đọc dữ liệu. Nhưng ta không thể đọc được trực tiếp mà phải thông qua luồng nhập đối tượng *ObjectInputStream* gắn với luồng nhập tệp tin *FileInputStream*. Việc đọc lại trạng thái đối tượng được tiến hành nhờ *phương thức readObject()*

```
FileInputStream fis=new FileInputStream("date.out");
ObjectInputStream ois=new ObjectInputStream(fis);
Date d=(Date)ois.readObject();
```

Quá trình trên còn được gọi là giải tuần tự hóa

Công việc đọc và ghi trạng thái của đối tượng khả tuần tự do người lập trình định nghĩa được tiến hành hoàn toàn tương tự như trên.

## 8.2 Truyền các đối tượng thông qua *Socket*

Chúng ta đã biết cách ghi và đọc các đối tượng từ các luồng vào ra trong một tiến trình đơn, bây giờ chúng ta sẽ xem xét cách truyền đối tượng thông qua *Socket*.

Mô hình lập trình *Socket* cho giao thức TCP là mô hình rất phổ biến trong lập trình mạng. Để lập chương trình client/server trong Java ta cần hai lớp *Socket* và *ServerSocket*.

### 8.2.1 Lớp *Socket*

Lớp *Socket* của Java được sử dụng bởi cả client và server, nó có các phương thức tương ứng với bốn thao tác đầu tiên. Ba thao tác cuối chỉ cần cho server để chờ các client liên kết với chúng. Các thao tác này được cài đặt bởi lớp *ServerSocket*. Các *Socket* cho client thường được sử dụng theo mô hình sau:

1. Một *Socket* mới được tạo ra bằng cách sử dụng hàm dựng *Socket()*.
2. *Socket* cố gắng liên kết với một host ở xa.
3. Mỗi khi liên kết được thiết lập, các host ở xa nhận các luồng vào và luồng ra từ *Socket*, và sử dụng các luồng này để gửi dữ liệu cho nhau. Kiểu liên kết này được gọi là song công (full-duplex), các host có thể nhận và gửi dữ liệu đồng thời. Ý nghĩa của dữ liệu phụ thuộc vào từng giao thức.
4. Khi việc truyền dữ liệu hoàn thành, một hoặc cả hai phía ngắt liên kết. Một số giao thức, như HTTP, đòi hỏi mỗi liên kết phải bị đóng sau mỗi khi yêu cầu được phục vụ. Các giao thức khác, chẳng hạn như FTP, cho phép nhiều yêu cầu được xử lý trong một liên kết đơn.

### 8.2.2 Lớp *ServerSocket*

Lớp *ServerSocket* có đủ mọi thứ ta cần để viết các server bằng Java. Nó có các constructor để tạo các đối tượng *ServerSocket* mới, các phương thức để lắng nghe các liên kết trên một cổng xác định và các phương thức trả về một *Socket* khi liên kết được thiết lập, vì vậy ta có thể gửi và nhận dữ liệu.

Vòng đời của một server

1. Một *ServerSocket* mới được tạo ra trên một cổng xác định bằng cách sử dụng một constructor *ServerSocket*.
2. *ServerSocket* lắng nghe liên kết đến trên cổng đó bằng cách sử dụng phương thức *accept()*. Phương thức *accept()* phong tỏa cho tới khi một client thực hiện

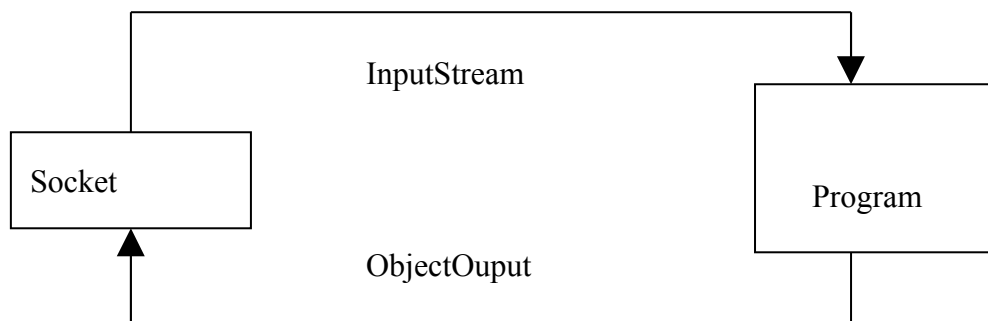


một liên kết, phương thức `accept()` trả về một đối tượng *Socket* biểu diễn liên kết giữa client và server.

3. Tùy thuộc vào kiểu server, hoặc phương thức `getInputStream()`, `getOutputStream()` hoặc cả hai được gọi để nhận các luồng vào ra phục vụ cho việc truyền tin với client.
4. Server và client tương tác theo một giao thức thỏa thuận sẵn cho tới khi ngắt liên kết.
5. Server, client hoặc cả hai ngắt liên kết  
Server trở về bước hai và đợi liên kết tiếp theo.

### 8.2.3 Truyền và nhận dữ liệu trong mô hình lập trình Socket

Việc truyền và nhận dữ liệu thực chất là các thao tác đọc và ghi dữ trên *Socket*. Ta có thể thấy điều này qua sơ đồ dưới đây:



Giả sử `s` là một đối tượng *Socket*. Nếu chương trình nhận dữ liệu thì ta sẽ lấy dữ liệu từ luồng nhập đến từ *Socket*:

```
InputStream is=s.getInputStream();
```

Để phục hồi trạng thái đối tượng ta gắn kết luồng nhập thô lấy được từ *Socket* với luồng đọc đối tượng *ObjectInputStream*:

```
ObjectInputStream ois=new ObjectInputStream(is);
```

Khi đó đối tượng được phục hồi lại trạng thái bởi câu lệnh:

```
Object obj=(Object)ois.readObject();
```

Nếu chương trình gửi dữ liệu thì ta sẽ lấy dữ liệu từ luồng xuất đến từ *Socket*:

```
ObjectOutput os=s.getOutputStream();
```

Để tiến hành ghi đối tượng ta gắn kết luồng xuất thô lấy được từ *Socket* với luồng xuất đối tượng *ObjectOutputStream*:

```
ObjectOutputStream oos=new ObjectOutputStream(os);
```

Việc truyền đối tượng lúc này trở thành một công việc rất đơn giản:

```
oos.writeObject(obj);  
oos.flush();
```

#### 8.2.4 Ví dụ minh họa

Để minh họa kỹ thuật chúng ta viết một server thực hiện phép nhân hai mảng số nguyên với nhau. Client gửi hai đối tượng, mỗi đối tượng biểu diễn một mảng nguyên; server nhận các đối tượng này, thực hiện lời gọi phương nhân hai mảng số nguyên với nhau và gửi kết quả trả về cho client.

Trước tiên chúng ta định nghĩa đối tượng để có thể sử dụng trong việc truyền các đối tượng.

```
public class ArrayObject implements java.io.Serializable{  
    private int[] a=null;  
    public ArrayObject(){  
    }  
    public void setArray(int a[]){  
        this.a=a;  
    }  
    public int[] getArray(){  
        return a;  
    }  
}
```

Lớp *ArrayObject* là lớp được xây dựng để đóng gói các mảng số nguyên và có khả năng truyền đi qua lại trên mạng. Cấu trúc lớp như sau: trường thông tin là một mảng số nguyên *a[]*; phương thức *setArray()* thiết lập giá trị cho mảng. Phương thức *getArray()* trả về một mảng số nguyên từ đối tượng *ArrayObject*.

Mô hình client/server tối thiểu phải có hai modul client và server. Trong ví dụ này cũng vậy ta sẽ xây dựng một số modul chương trình như sau:

Đầu tiên chúng ta phát triển client. Client tạo ra hai thể hiện của các đối tượng *ArrayObject* và ghi chúng ra luồng xuất (thực chất là gửi tới server).

```

public class ArrayClient{
    public static void main(String[] args)throws Exception{
        ObjectOuputStream oos=null;
        ObjectInputStream ois=null;
        int dat1[]={3,3,3,3,3,3,3};
        int dat2[]={5,5,5,5,5,5,5};
        Socket s=new Socket("localhost",1234);
        oos=new ObjectOuputStream(s.getObjectOuput());
        ois=new ObjectInputStream(s.getInputStream());
        ArrayObject a1=new ArrayObject();
        a1.setArray(dat1);
        ArrayObject a2=new ArrayObject();
        a2.setArray(dat2);
        ArrayObject res=null;
        int r[]=new int[7];
        oos.writeObject(a1);
        oos.writeObject(a2);
        oos.flush();
        res=(ArrayObject)ois.readObject();
        r=res.getArray();
        System.out.println("The result received from server...");
        System.out.println();
        for(int i=0;i<r.length;i++)System.out.print(r[i]+" ");
    }
}

```

Bước tiếp theo chúng ta phát triển server. Server là một chương trình cung cấp dịch vụ phục vụ các yêu cầu của client. Server nhận hai đối tượng *ArrayObject* và nhận về hai mảng từ hai đối tượng này và sau đó đem nhân chúng với nhau và gửi kết quả trở lại cho client.

```

public class ArrayServer extends Thread {
    private ServerSocket ss;
    public static void main(String args[])throws Exception
    {
        new ArrayServer();
    }
}

```

```

public ArrayServer() throws Exception{
    ss=new ServerSocket(1234);
    System.out.println("Server running on port "+1234);
    this.start();
}
public void run(){
    while(true){
        try{
            System.out.println("Waiting for client...");
            Socket s=ss.accept();
            System.out.println("Accepting a connection
                                from:"+s.getInetAddress());
            Connect c=new Connect(s);
        }catch(Exception e){
            System.out.println(e);
        }
    }
}
}

```

Trong mô hình client/server tại một thời điểm server có thể phục vụ các yêu cầu đến từ nhiều client, điều này có thể dẫn đến các vấn đề tương tranh. Chính vì lý do này mà lớp *ArrayServer* thừa kế lớp *Thread* để giải quyết vấn đề trên. Ngoài ra để nâng cao hiệu suất của chương trình thì sau khi đã chấp nhận liên kết từ một client nào đó, việc xử lý dữ liệu sẽ được dành riêng cho một tuyến đoạn để server có thể tiếp tục chấp nhận các yêu cầu khác. Hay nói cách khác, mỗi một yêu cầu của client được xử lý trong một tuyến đoạn riêng biệt.

```

class Connect extends Thread{
    private Socket client=null;
    private ObjectInputStream ois;
    private ObjectOutputStream oos;
    public Connect(){
    }
    public Connect(Socket client){
        this.client=client;
    }
}

```

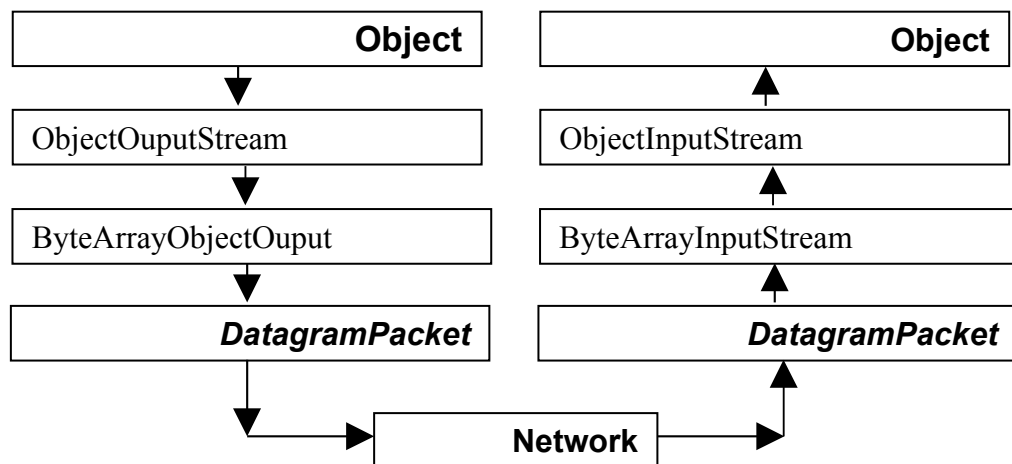
```

        try{
            ois=new ObjectInputStream(client.getInputStream());
            oos=new ObjectOuputStream(client.getObjectOuput());
        }
        catch(Exception e){
            System.err.println(e);
        }
        this.start();
    }
    public void run(){
        ArrayObject x=null;
        ArrayObject y=null;
        int a1[]=new int[7];
        int a2[]=new int[7];
        int r[]=new int[7];
        try{
            x=(ArrayObject)ois.readObject();
            y=(ArrayObject)ois.readObject();
            a1=x.getArray();
            a2=y.getArray();
            for(int i=0;i<a1.length;i++)r[i]=a1[i]*a2[i];
            ArrayObject res=new ArrayObject();
            res.setArray(r);
            oos.writeObject(res);
            oos.flush();
            ois.close();
            client.close();
        }
        catch(Exception e){
        }
    }
}

```

### 8.3 Truyền các đối tượng thông qua giao thức UDP

Một giao thức gần với giao thức TCP là giao thức UDP. Java hỗ trợ cho kiểu ứng dụng truyền tin phi liên kết trên giao thức UDP thông qua lớp *DatagramSocket* và *DatagramPacket*. Liệu chúng ta có thể viết được các chương trình nhập và xuất đối tượng bằng truyền tin datagram? Thực hiện điều này không thể tiến hành trực tiếp như với luồng *Socket*. Vấn đề là *DatagramSocket* không được gắn với bất kỳ luồng nào; mà nó sử dụng một tham số mảng byte để gửi và nhận dữ liệu.



Có thể thấy rằng để xây dựng một gói tin datagram, đối tượng phải được chuyển thành một mảng byte. Việc chuyển đổi này rất khó để thực hiện nếu bản thân đối tượng có liên quan đến một số đối tượng phức tạp trong đồ thị đối tượng.

Hình 6 minh họa dòng luân chuyển dữ liệu khi truyền một đối tượng thông qua một datagram. Dưới đây là bảy bước ta cần thực hiện để cài đặt mô hình truyền dữ liệu cho giao thức UDP

- Bước 1. Chuẩn bị: Tạo đối tượng cần truyền đi, giả sử đối tượng này là obj, làm cho nó khả tuần tự bằng cách thực thi giao tiếp *Serializable*.
- Bước 2. Tạo một luồng *ByteArrayOutputStream* và đặt tên cho nó là baos.
- Bước 3. Xây dựng đối tượng *ObjectOutputStream* và đặt tên cho nó là oos. Tham số cho cấu tử *ObjectOutputStream* là baos
- Bước 4. Ghi đối tượng obj vào luồng baos bằng cách sử dụng phương thức *writeObject()* của oos.

- Bước 5. Tìm kiếm vùng đệm dữ liệu mảng byte từ bảng cách sử dụng phương thức *toByteArray()*.
- Bước 6. Xây dựng đối tượng *DatagramPacket* và đặt tên là dp với dữ liệu đầu vào là vùng đệm dữ liệu đã tìm được ở bước 5.
- Bước 7. Gửi dp thông qua *DatagramSocket* bằng cách gọi phương thức *send()* của nó.

Ví dụ minh họa chi tiết quá trình gửi một đối tượng

```
InetAddress ia=InetAddress.getByName("localhost");
Student st=new Student("Peter",7,8,9);
DatagramSocket ds=new DatagramSocket();
ByteArrayOutputStream baos=new ByteArrayOutputStream(5000);
ObjectOutputStream oos=new ObjectOutputStream(
                                new BufferedOutputStream(baos));

oos.flush();
oos.writeObject(st);
oos.flush();
byte[] b=baos.toByteArray();
DatagramPacket dp=new DatagramPacket(b,b.length,ia,1234);
ds.send(dp);
oos.close();
```

Để nhận một đối tượng ta cũng tiến hành các bước như trên nhưng theo thứ tự ngược lại, thay thế luồng *ObjectOutputStream* bằng *ObjectInputStream* và *ByteArrayOutputStream* bằng *ByteArrayInputStream*.

Ví dụ dưới đây minh họa chi tiết quá trình nhận một đối tượng

```
DatagramSocket ds=new DatagramSocket(1234);
while(true){
    byte b[]=new byte[5000];
    DatagramPacket dp=new DatagramPacket(b,b.length);
    ds.receive(dp);
    ByteArrayInputStream bais;
    bais=new ByteArrayInputStream(new BufferedInputStream(b));
    ObjectInputStream ois =new ObjectInputStream(bais);
    Student st=(Student)ois.readObject();
```

```
st.computeAverage();  
st.print();  
ois.close();  
bais.close();  
}
```

## 8.4 Kết luận

Qua bài báo này tôi đã giới thiệu tổng quan về tuần tự hóa đối tượng. Thông qua các ví dụ chúng ta thấy không quá khó để làm việc với tuần tự hóa đối tượng và điều quan trọng hơn là chúng ta đã biết cách để truyền đi các đối tượng có cấu trúc phức tạp thông qua các *Socket*.

Ngoài ra, bài báo cũng đã đề cập tới cách truyền đối tượng bằng cách sử dụng các gói tin datagram. Nhờ những ưu điểm của tiện ích tuần tự hóa đối tượng, tôi đã minh họa một cách truyền các đối tượng bằng cách sử dụng các gói tin datagram. Như chúng ta đã thấy, mặc dù trong giao thức này không hỗ trợ xử lý theo luồng dữ liệu nhưng tôi đã “luồng hóa” các đối tượng để đưa các đối tượng vào các mảng byte.

Sự lựa chọn giữa việc sử dụng RMI hay giải pháp *Socket* kết hợp với tuần tự hóa phụ thuộc vào từng dự án và các yêu cầu của nó. Sự lựa chọn giải pháp nào chính là sự thỏa hiệp giữa các đặc trưng của mỗi giải pháp: nếu đối với RMI thì đó là tính đơn giản khi triển khai, ngược lại với *Socket* kết hợp với tuần tự hóa đối tượng thì đó lại là ưu thế về mặt hiệu năng. Nếu vấn đề hiệu năng có tầm quan trọng thì giải pháp lập trình *Socket* kết hợp tuần tự hóa đối tượng là giải pháp tốt hơn so với RMI.



## CHƯƠNG 9 . PHÂN TÁN ĐỐI TƯỢNG TRONG JAVA

### BẰNG RMI

#### 9.1 Tổng quan

RMI là một cơ chế cho phép một đối tượng đang chạy trên một máy ảo Java này ( Java Virtual Machine) gọi các phương thức của một đối tượng đang tồn tại trên một máy ảo Java khác (JVM).

Thực chất RMI là một cơ chế gọi phương thức từ xa đã được thực hiện và tích hợp trong ngôn ngữ Java. Vì Java là một ngôn ngữ lập trình hướng đối tượng, nên phương pháp lập trình trong RMI là phương pháp hướng đối tượng do đó các thao tác hay các lời gọi phương thức đều liên quan đến đối tượng. Ngoài ra, RMI còn cho phép một Client có thể gửi tới một đối tượng đến cho Server xử lý, và đối tượng này cũng có thể được xem là tham số cho lời gọi hàm từ xa, đối tượng này cũng có những dữ liệu bên trong và các hành vi như một đối tượng thực sự.

So sánh giữa gọi phương thức từ xa với các lời gọi thủ tục từ xa

Gọi phương thức từ xa không phải là một khái niệm mới. Thậm chí trước khi ra đời lập trình hướng đối tượng phần mềm đã có thể gọi các hàm và các thủ tục từ xa. Các hệ thống như RPC đã được sử dụng trong nhiều năm và hiện nay vẫn được sử dụng.

Trước hết, Java là một ngôn ngữ độc lập với nền và cho phép các ứng dụng Java truyền tin với các ứng dụng Java đang chạy trên bất kỳ phần cứng và hệ điều hành nào có hỗ trợ JVM. Sự khác biệt chính giữa hai mục tiêu là RPC hỗ trợ đa ngôn ngữ, ngược lại RMI chỉ hỗ trợ các ứng dụng được viết bằng Java.

Ngoài vấn đề về ngôn ngữ và hệ thống, có một số sự khác biệt căn bản giữa RPC và RMI. Gọi phương thức từ xa làm việc với các đối tượng, cho phép các phương thức chấp nhận và trả về các đối tượng Java cũng như các kiểu dữ liệu nguyên tố (primitive type). Ngược lại gọi thủ tục từ xa không hỗ trợ khái niệm đối tượng. Các thông điệp gửi cho một dịch vụ RPC (Remote Procedure Calling) được biểu diễn bởi ngôn ngữ XDR (External Data Representation): dạng thức biểu diễn dữ liệu ngoài. Chỉ có các kiểu dữ liệu có thể được định nghĩa bởi XDR mới có thể truyền đi.

## 9.2 Mục đích của RMI

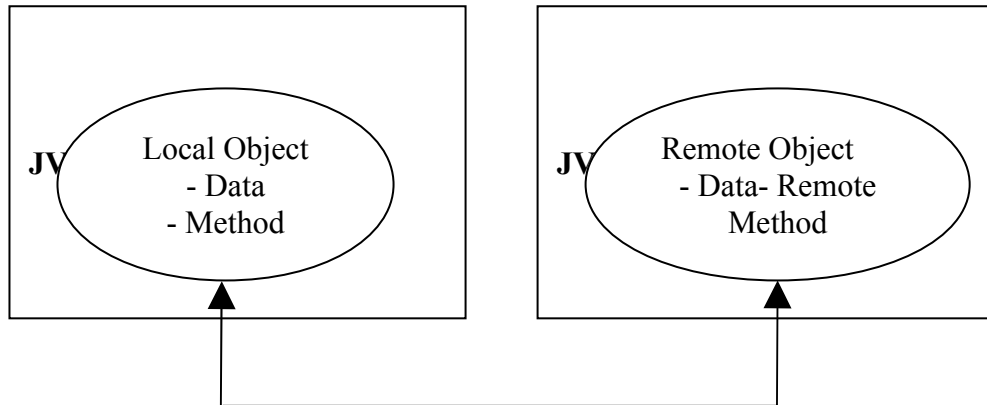
- Hỗ trợ gọi phương thức từ xa trên các đối tượng trong các máy ảo khác nhau
- Hỗ trợ gọi ngược phương thức ngược từ server tới các applet
- Tích hợp mô hình đối tượng phân tán vào ngôn ngữ lập trình Java theo một cách tự nhiên trong khi vẫn duy trì các ngữ cảnh đối tượng của ngôn ngữ lập trình Java
- Làm cho sự khác biệt giữa mô hình đối tượng phân tán và mô hình đối tượng cục bộ không có sự khác biệt.
- Tạo ra các ứng dụng phân tán có độ tin cậy một cách dễ dàng
- Duy trì sự an toàn kiểu được cung cấp bởi môi trường thời gian chạy của nền tảng Java
- Hỗ trợ các ngữ cảnh tham chiếu khác nhau cho các đối tượng từ xa
- Duy trì môi trường an toàn của Java bằng các trình bảo an và các trình nạp lớp.

## 9.3 Một số thuật ngữ

Cũng như tất cả các chương trình khác trong Java, chương trình RMI cũng được xây dựng bởi các giao tiếp và lớp. Giao tiếp định nghĩa các phương thức và các lớp thực thi các phương thức đó. Ngoài ra lớp còn thực hiện một vài phương thức khác. Nhưng chỉ có những phương thức khai báo trong giao tiếp thừa kế từ giao tiếp Remote hoặc các lớp con của nó mới được Client gọi từ JVM khác. Trong mục này ta nêu một số thuật ngữ thường xuyên được sử dụng trong phần này:

- Giao tiếp Remote: Một giao tiếp khai báo các phương thức cho phép gọi từ xa. Trong Java giao tiếp Remote có các đặc điểm sau:
  - Thừa kế giao tiếp có sẵn: `java.rmi.Remote`.
  - Mỗi phương thức trong giao tiếp Remote phải được khai báo để đưa ra ngoại lệ `RemoteException` nhờ mệnh đề `throws java.rmi.RemoteException` và có thể có các ngoại lệ khác.
- Đối tượng Remote: một đối tượng được tạo ra để cho phép những đối tượng khác trên một máy JVM khác gọi tới nó.

- Phương thức Remote: Đối tượng Remote chứa một số các phương thức, những phương thức này có thể được gọi từ xa bởi các đối tượng trong JVM khác.



#### 9.4 Các lớp trung gian Stub và Skeleton

Trong kỹ thuật lập trình phân tán RMI, để các đối tượng trên các máy Java ảo khác nhau có thể truyền tin với nhau thông qua các lớp trung gian: Stub và Skeleton.

Vai trò của lớp trung gian: Lớp trung gian tồn tại cả ở hai phía client (nơi gọi phương thức của các đối tượng ở xa) và server (nơi đối tượng thật sự được cài đặt để thực thi mã lệnh của phương thức). Trong Java trình biên dịch `rmic.exe` được sử dụng để tạo ra lớp trung gian này. Phía client lớp trung gian này gọi là Stub (lớp móc), phía server lớp trung gian này gọi là Skeleton (lớp nối) chúng giống như các lớp môi giới giúp các lớp ở xa truyền tin với nhau.

#### 9.5 Cơ chế hoạt động của RMI

Các hệ thống RMI phục vụ cho việc truyền tin thường được chia thành hai loại: client và server. Một server cung cấp dịch vụ RMI, và client gọi các phương thức trên đối tượng của dịch vụ này.

Server RMI phải đăng ký với một dịch vụ tra tìm và đăng ký tên. Dịch vụ này cho phép các client truy tìm chúng, hoặc chúng có thể tham chiếu tới dịch vụ trong một mô hình khác. Một chương trình đóng vai trò như vậy có tên là `rmiregistry`, chương trình này chạy như một tiến trình độc lập và cho phép các ứng dụng đăng ký dịch vụ RMI hoặc nhận một tham chiếu tới dịch vụ được đặt tên. Mỗi khi server được đăng ký, nó sẽ chờ các yêu cầu RMI từ các client. Gắn với mỗi đăng ký dịch vụ là một tên được biểu diễn bằng một chuỗi ký tự để

cho phép các client lựa chọn dịch vụ thích hợp. Nếu một dịch vụ chuyển từ server này sang một server khác, client chỉ cần tra tìm trình đăng ký để tìm ra vị trí mới. Điều này làm cho hệ thống có khả năng dung thứ lỗi-nếu một dịch vụ không khả dụng do một máy bị sập, người quản trị hệ thống có thể tạo ra một thể hiện mới của dịch vụ trên một hệ thống khác và đăng ký nó với trình đăng ký RMI.

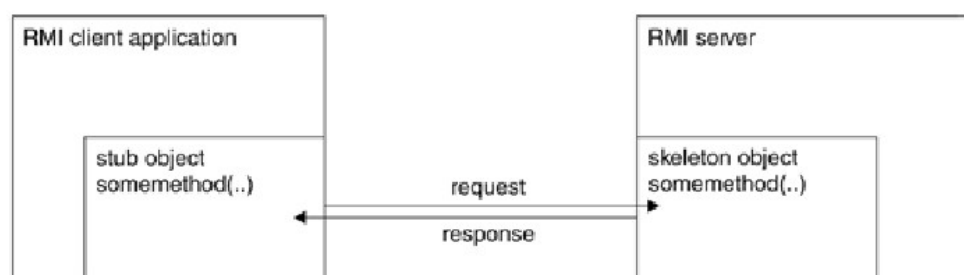
Các client RMI sẽ gửi các thông điệp RMI để gọi một phương thức trên một đối tượng từ xa. Trước khi thực hiện gọi phương thức từ xa, client phải nhận được một tham chiếu từ xa. Tham chiếu này thường có được bằng cách tra tìm một dịch vụ trong trình đăng ký RMI. Ứng dụng client yêu cầu một tên dịch vụ cụ thể, và nhận một URL trỏ tới tài nguyên từ xa. Khuôn dạng dưới đây được sử dụng để biểu diễn một tham chiếu đối tượng từ xa:

```
rmi://hostname:port/servicename
```

Trong đó *hostname* là tên của máy chủ hoặc một địa chỉ IP, *port* xác định dịch vụ, và *servicename* là một chuỗi ký tự mô tả dịch vụ.

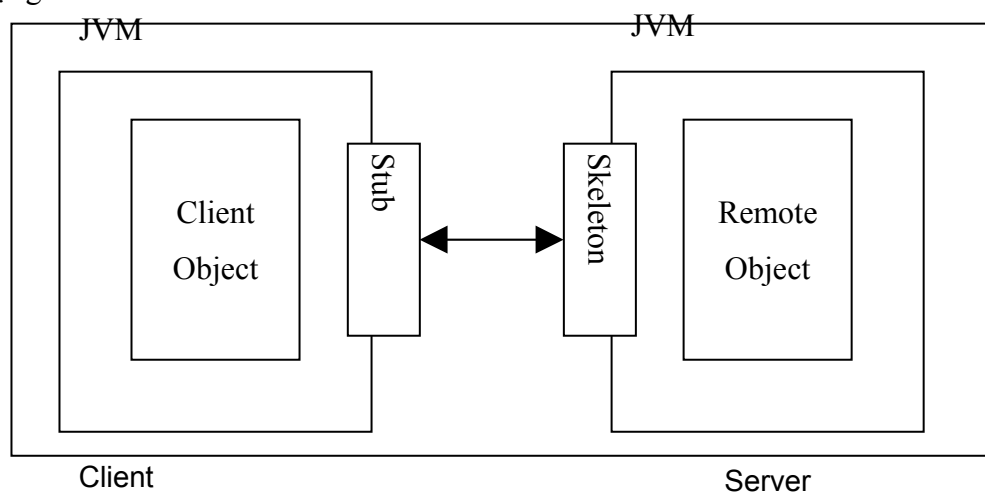
Mỗi khi có được một tham chiếu, client có thể tương tác với dịch vụ từ xa. Các chi tiết liên quan đến mạng hoàn toàn được che dấu đối với những người phát triển ứng dụng-làm việc với các đối tượng từ xa đơn giản như làm việc với các đối tượng cục bộ. Điều này có thể có được thông qua sự phân chia hệ thống RMI thành hai thành phần, stub và skeleton.

Đối tượng stub là một đối tượng ủy quyền, truyền tải yêu cầu đối tượng tới server RMI. Cần nhớ rằng mỗi dịch vụ RMI được định nghĩa như là một giao tiếp, chứ không phải là một chương trình cài đặt, các ứng dụng client giống như các chương trình hướng đối tượng khác. Tuy nhiên ngoài việc thực hiện công việc của chính nó, stub còn truyền một thông điệp tới một dịch vụ RMI ở xa, chờ đáp ứng, và trả về đáp ứng cho phương thức gọi. Người phát triển ứng dụng không cần quan tâm đến tài nguyên RMI nằm ở đâu, nó đang chạy trên nền nào, nó đáp ứng đầy đủ yêu cầu như thế nào. Client RMI đơn giản gọi một phương thức trên đối tượng ủy quyền, đối tượng này quản lý tất cả các chi tiết cài đặt.

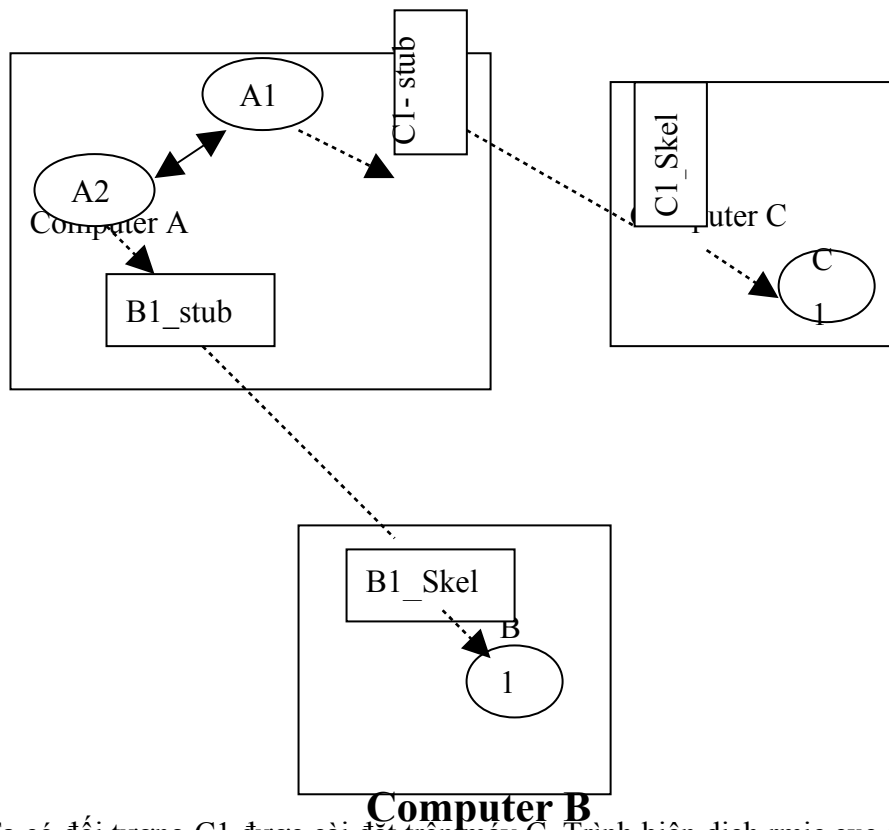


Tại phía server, đối tượng skeleton có nhiệm vụ lắng nghe các yêu cầu RMI đến và truyền các yêu cầu này tới dịch vụ RMI. Đối tượng skeleton không cung cấp bản cài đặt của dịch vụ RMI. Nó chỉ đóng vai trò như là chương trình nhận các yêu cầu, và truyền các yêu cầu. Sau khi người phát triển tạo ra một giao tiếp RMI, thì anh ta phải cung cấp một phiên bản cài đặt cụ thể của giao tiếp. Đối tượng cài đặt này được gọi là đối tượng skeleton, đối tượng này gọi phương thức tương ứng và truyền các kết quả cho đối tượng stub trong client RMI. Mô hình này làm cho việc lập trình trở nên đơn giản, vì skeleton được tách biệt với cài đặt thực tế của dịch vụ. Tất cả những gì mà người phát triển dịch vụ cần quan tâm là mã lệnh khởi tạo (để đăng ký dịch vụ và chấp nhận dịch vụ), và cung cấp chương trình cài đặt của giao tiếp dịch vụ RMI.

Với câu hỏi các thông điệp được truyền như thế nào, câu trả lời tương đối đơn giản. Việc truyền tin diễn ra giữa các đối tượng stub và skeleton bằng cách sử dụng các socket TCP. Mỗi khi được tạo ra, skeleton lắng nghe các yêu cầu đến được phát ra bởi các đối tượng stub. Các tham số trong hệ thống RMI không chỉ hạn chế đối với các kiểu dữ liệu nguyên tố-bất kỳ đối tượng nào có khả năng tuần tự hóa đều có thể được truyền như một tham số hoặc được trả về từ phương thức từ xa. Khi một stub truyền một yêu cầu tới một đối tượng skeleton, nó phải đóng gói các tham số (hoặc là các kiểu dữ liệu nguyên tố, các đối tượng hoặc cả hai) để truyền đi, quá trình này được gọi là marshalling. Tại phía skeleton các tham số được khôi phục lại để tạo nên các kiểu dữ liệu nguyên tố và các đối tượng, quá trình này còn được gọi là unmarshaling. Để thực hiện nhiệm vụ này, các lớp con của các lớp *ObjectOutputStream* và *ObjectInputStream* được sử dụng để đọc và ghi nội dung của các đối tượng.



Sơ đồ gọi phương thức của các đối tượng ở xa thông qua lớp trung gian được cụ thể hoá như sau:



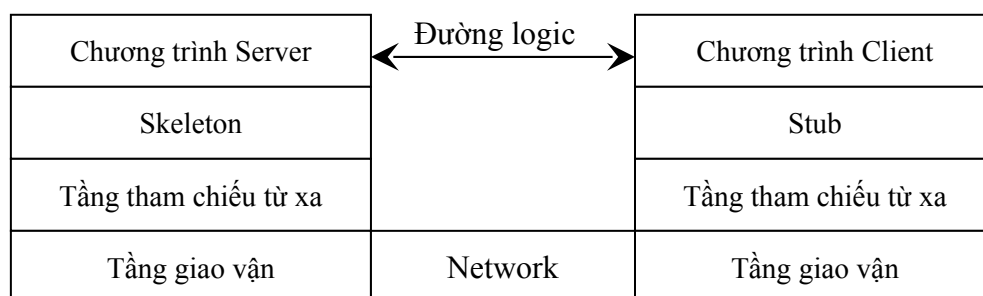
- Ta có đối tượng C1 được cài đặt trên máy C. Trình biên dịch rmic.exe sẽ tạo ra hai lớp trung gian C1\_Skel và C1\_Stub. Lớp C1\_Stub sẽ được đem về máy A. Khi A1 trên máy A gọi C1 nó sẽ chuyển lời gọi đến lớp C1\_Stub, C1\_Stub chịu trách nhiệm đóng gói tham số, chuyển vào không gian địa chỉ tương thích với đối tượng C1 sau đó gọi phương thức tương ứng.
- Nếu có phương thức của đối tượng C1 trả về sẽ được lớp C1\_Skel đóng gói trả ngược về cho C1\_Stub chuyển giao kết quả cuối cùng lại cho A1. Nếu khi kết nối mạng gặp sự cố thì lớp trung gian Stub sẽ thông báo lỗi đến đối tượng A1. Theo cơ chế này A1 luôn nghĩ rằng nó đang hoạt động trực tiếp với đối tượng C1 trên máy cục bộ.
- Trên thực tế, C1\_Stub trên máy A 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 này, đối tượng C1 cần cung cấp một giao diện tương ứng với các phương thức cho phép đối tượng A1 gọi nó trên máy A.

## 9.6 Kiến trúc RMI

Sự khác biệt căn bản giữa các đối tượng từ xa và các đối tượng cục bộ là các đối tượng từ xa nằm trên một máy ảo khác. Thông thường, các tham số đối tượng được truyền cho các phương thức và các giá trị đối tượng được trả về từ các phương thức thông qua cách truyền theo tham chiếu. Tuy nhiên cách này không làm việc khi các phương thức gọi và các phương thức được gọi không cùng nằm trên một máy ảo.

Vì vậy, có ba cơ chế khác nhau được sử dụng để truyền các tham số cho các phương thức từ xa và nhận các kết quả trả về từ các phương thức ở xa. Các kiểu nguyên tố (int, boolean, double,...) được truyền theo tham trị. Các tham chiếu tới các đối tượng từ xa được truyền dưới dạng các tham chiếu cho phép tất cả phía nhận gọi các phương thức trên các đối tượng từ xa. Các đối tượng không thực thi giao tiếp từ xa (nghĩa là các đối tượng không thực thi giao tiếp Remote) được truyền theo tham trị; nghĩa là các bản sao đầy đủ được truyền đi bằng cách sử dụng cơ chế tuần tự hóa đối tượng. Các đối tượng không có khả năng tuần tự hóa thì không thể được truyền đi tới các phương thức ở xa. Các đối tượng ở xa chạy trên server nhưng có thể được gọi bởi các đối tượng đang chạy trên client. Các đối tượng không phải ở xa, các đối tượng khả tuần tự chạy trên các hệ thống client.

Để quá trình truyền tin là trong suốt với người lập trình, truyền tin giữa client và server được cài đặt theo mô hình phân tầng như hình vẽ dưới đây



Đối với người lập trình, client dường như truyền tin trực tiếp với server. Thực tế, chương trình client chỉ truyền tin với đối tượng stub là đối tượng ủy quyền của đối tượng thực sự nằm trên hệ thống từ xa. Stub chuyển cuộc đàm thoại cho tầng tham chiếu, tầng này truyền tin trực tiếp với tầng giao vận. Tầng giao vận trên client truyền dữ liệu đi trên mạng máy tính tới tầng giao vận bên phía server. Tầng giao vận bên phía server truyền tin với tầng tham chiếu, tầng này truyền tin một phần của phần mềm server được gọi là skeleton.

Skeleton truyền tin với chính server. Theo hướng khác từ server đến client thì luồng truyền tin được đi theo chiều ngược lại.

Cách tiếp cận có vẻ phức tạp nhưng ta không cần quan tâm đến vấn đề này. Tất cả đều được che dấu đi, người lập trình chỉ quan tâm đến việc lập các chương trình có khả năng gọi phương thức từ xa giống như đối với chương trình cục bộ.

Trước khi có thể gọi một phương thức trên một đối tượng ở xa, ta cần một tham chiếu tới đối tượng đó. Để nhận được tham chiếu này, ta yêu cầu một trình đăng ký tên `rmiregistry` cung cấp tên của tham chiếu. Trình đăng ký đóng vai trò như là một DNS nhỏ cho các đối tượng từ xa. Một client kết nối với trình đăng ký và cung cấp cho nó một URL của đối tượng từ xa. Trình đăng ký cung cấp một tham chiếu tới đối tượng đó và client sử dụng tham chiếu này để gọi các phương thức trên server.

Trong thực tế, client chỉ gọi các phương thức cục bộ trên trong stub. Stub là một đối tượng cục bộ thực thi các giao tiếp từ xa của các đối tượng từ xa.

Tầng tham chiếu từ xa thực thi giao thức tầng tham chiếu từ xa cụ thể. Tầng này độc lập với các đối tượng stub và skeleton cụ thể. Tầng tham chiếu từ xa có nhiệm vụ hiểu tầng tham chiếu từ xa có ý nghĩa như thế nào. Đôi khi tầng tham chiếu từ xa có thể tham chiếu tới nhiều máy ảo trên nhiều host.

Tầng giao vận gửi các lời gọi trên Internet. Phía server, tầng giao vận lắng nghe các liên kết đến. Trên cơ sở nhận lời gọi phương thức, tầng giao vận chuyển lời gọi cho tầng tham chiếu trên server. Tầng tham chiếu chuyển đổi các tham chiếu được gửi bởi client thành các tham chiếu cho các máy ảo cục bộ. Sau đó nó chuyển yêu cầu cho skeleton. Skeleton đọc tham số và truyền dữ liệu cho chương trình server, chương trình server sẽ thực hiện lời gọi phương thức thực sự. Nếu lời gọi phương thức trả về giá trị, giá trị được gửi xuống cho skeleton, tầng tham chiếu ở xa, và tầng giao vận trên phía server, thông qua Internet và sau đó chuyển lên cho tầng giao vận, tầng tham chiếu ở xa, stub trên phía client.

## **9.7 Cài đặt chương trình**

Để lập một hệ thống client/server bằng RMI ta sử dụng ba gói cơ bản sau: `java.rmi`, `java.rmi.server`, `java.rmi.registry`. Gói `java.rmi` bao gồm các giao tiếp, các lớp và các ngoại lệ được sử dụng để lập trình cho phía client. Gói `java.rmi.server` cung cấp các giao tiếp, các lớp và các ngoại lệ được sử dụng để lập trình cho phía server. Gói `java.rmi.registry` có các giao tiếp, các lớp và các ngoại lệ được sử dụng để định vị và đặt tên các đối tượng.



### 9.7.1 Cài đặt chương trình phía Server

Để minh họa cho kỹ thuật lập trình RMI ở đây tác giả xin giới thiệu cách lập một chương trình FileServer đơn giản cho phép client tải về một tệp tin.

- **Bước 1:** Đặc tả giao tiếp Remote

```
import java.rmi.*;

public interface FileInterface extends Remote
{
    public byte[] downloadFile(String fileName) throws
    RemoteException;
}
```

- **Bước 2:** Viết lớp thực thi giao tiếp

```
import java.rmi.*;
import java.rmi.server.*;
import java.io.*;

public class FileImpl extends UnicastRemoteObject implements
FileInterface
{
    private String name;
    public FileImpl(String s) throws RemoteException
    {
        super();
        name=s;
    }
    public byte[] downloadFile(String fileName) throws
    RemoteException
    {
        try{
            File file=new File(fileName);
            //Tạo một mảng b để lưu nội dung của tệp
            byte b[]=new byte[(int)file.length()];
            BufferedInputStream bis=new BufferedInputStream(new
            FileInputStream(fileName));
            bis.read(b,0,b.length);
            bis.close();
            return b;
        }
    }
}
```

```

    }
    catch(Exception e)
    {
        System.err.println(e);
        return null;
    }
}
}

```

- **Bước 3: Viết chương trình phía server**

```

import java.io.*;
import java.rmi.*;
import java.net.*;
public class FileServer
{
    public static void main(String[] args) throws Exception
    {
        FileInterface fi=new FileImpl("FileServer");
        InetAddress dc=InetAddress.getLocalHost();
        Naming.rebind("rmi://" +dc.getHostAddress()
+" /FileServer",fi);
        System.out.println("Server ready for client requests...");
    }
}

```

- **Bước 4: Cài đặt client**

```

import java.rmi.*;
import java.io.*;
public class FileClient
{
    public static void main(String[] args) throws Exception
    {
        if(args.length!=2)
        {
            System.out.println("Sử dụng:java FileClient fileName
machineName ");

```

```

        System.exit(0);
    }
    String name="rmi://" + args[1] + "/FileServer";
    FileInterface fi=(FileInterface)Naming.lookup(name);
    byte[] filedata=fi.downloadFile(args[0]);
    File file =new File(args[0]);
    BufferedOutputStream bos=new BufferedOutputStream(new
    FileOutputStream(file.getName()));
    bos.write(filedata,0,filedata.length);
    bos.flush();
    bos.close();
}
}

```

## 9.8 Triển khai ứng dụng

Để triển khai ứng dụng RMI ta cần thực hiện các bước sau:

- **Bước 1:** Biên dịch các tệp chương trình

```

C:\MyJava>javac FileInterface.java
C:\MyJava>javac FileImpl.java
C:\MyJava>javac FileServer.java
C:\MyJava>javac FileClient.java

```

Ta sẽ thu được các lớp sau:

FileInterface.class, FileImpl.class, FileServer.class, FileClient.class

Để tạo ra các lớp trung gian ta dùng lệnh sau:

```

C:\MyJava>rmic FileImpl

```

Sau khi biên dịch ta sẽ thu được hai lớp trung gian là FileImpl\_Stub.class và FileImpl\_Skel.class.

- **Bước 2:** Tổ chức chương trình

Ta tổ chức chương trình trên hai máy client và server như sau:

Phía Server	Phía Client
FileInterface.class	FileInterface.class
FileImpl.class	FileImpl_Stub.class
FileImpl_Skel.class	FileClient.class

FileServer.class	
------------------	--

- **Bước 3:** Khởi động chương trình

Ở đây ta giả lập chương trình trên cùng một máy. Việc triển khai trên mạng không có gì khó khăn ngoài việc cung cấp hostname hoặc địa chỉ IP của server cung cấp dịch vụ

Khởi động trình đăng ký:

```
C:\MyJava>start rmiregistry
```

Khởi động server

```
C:\MyJava>start java FileServer
```

Khởi động client

```
C:\MyJava>java FileClient D:\RapidGet.exe localhost
```

## 9.9 Các lớp và các giao tiếp trong gói java.rmi

Khi viết một applet hay một ứng dụng sử dụng các đối tượng ở xa, người lập trình cần nhận thức rằng các giao tiếp và các lớp cần dùng cho phía client nằm ở trong gói java.rmi

### 9.9.1 *Giao tiếp Remote*

Giao tiếp này không khai báo bất kỳ phương thức nào. Các phương thức được khai báo trong phương thức này là các giao tiếp có thể được gọi từ xa.

### 9.9.2 *Lớp Naming*

Lớp java.rmi.Naming truyền tin trực tiếp với một trình đăng ký đang chạy trên server để ánh xạ các URL rmi://hostname/myObject thành các đối tượng từ xa cụ thể trên host xác định. Ta có thể xem trình đăng ký như là một DNS cho các đối tượng ở xa. Mỗi điểm vào trong trình đăng ký bao gồm một tên và một tham chiếu đối tượng. Các client cung cấp tên và nhận về một tham chiếu tới URL.

URL rmi giống như URL http ngoại trừ phần giao thức được thay thế bằng rmi. Phần đường dẫn của URL là tên gắn với đối tượng từ xa trên server chứ không phải là tên một tệp tin.

Lớp Naming cung cấp các phương thức sau:

- `Public static String[] list(String url) throws RemoteException`

Phương thức này trả về một mảng các chuỗi ký tự, mỗi chuỗi là một URL đã được gắn với một tham chiếu. Tham số url là URL của trình đăng ký Naming.

- `Public static Remote lookup(String url)throws RemoteException, NotBoundException, AccessException, MalformedURLException`

Client sử dụng phương thức này để tìm kiếm một đối tượng từ xa gắn liền với tên đối tượng.

Phương thức này đưa ra ngoại lệ `NotBoundException` nếu server ở xa không nhận ra tên của nó. Nó đưa ra ngoại lệ `RemoteException` nếu trình không thể liên lạc được với trình đăng ký. Nó đưa ra ngoại lệ `AccessException` nếu server từ chối tra tìm tên cho host cụ thể. Cuối cùng nếu URL không đúng cú pháp nó sẽ đưa ra ngoại lệ `MalformedURLException`.

- `Public static void bind(String url, Remote object)throws RemoteException, AlreadyBoundException, MalformedURLException, AccessException`

Server sử dụng phương thức `bind()` để liên kết một tên với một đối tượng ở xa. Nếu việc gán là thành công thì client có thể tìm kiếm đối tượng stub từ trình đăng ký.

Có rất nhiều tình huống có thể xảy ra trong quá trình gán tên. Phương thức này đưa ra ngoại lệ `MalformedURLException` nếu url không đúng cú pháp. Nó đưa ra ngoại lệ `RemoteException` nếu không thể liên lạc được với trình đăng ký. Nó đưa ra ngoại lệ `AccessException` nếu client không được phép gán các đối tượng trong trình đăng ký. Nếu đối tượng URL đã gắn với một đối tượng cục bộ nó sẽ đưa ra ngoại lệ `AlreadyBoundException`.

- `Public static void rebind(String url, Remote obj)throws RemoteException, AccessException, MalformedURLException`

Phương thức này giống như phương thức `bind()` ngoại trừ việc là nó gán URL cho đối tượng ngay cả khi URL đã được gán.

## 9.10 Các lớp và các giao tiếp trong gói `java.rmi.registry`

Làm thế nào để nhận được một tham chiếu tới đối tượng? Client tìm ra các đối tượng ở xa hiện có bằng cách thực hiện truy vấn với trình đăng ký của server. Trình đăng ký cho ta biết những đối tượng nào đang khả dụng bằng cách truy vấn trình đăng ký của server. Ta đã biết lớp `java.rmi.Naming` cho phép chương trình giao tiếp với trình đăng ký.

Giao tiếp Registry và lớp `LocateRegistry` cho phép các client tìm kiếm các đối tượng ở xa trên một server theo tên. `RegistryImpl` là lớp con của lớp `RemoteObject`, lớp này liên kết

các tên với các đối tượng RemoteObject. Client sử dụng lớp LocateRegistry để tìm kiếm RegistryImpl cho một host và cổng cụ thể.

#### 9.10.1 Giao tiếp Registry

Giao tiếp này cung cấp năm phương thức:

- Bind() để gán một tên với một đối tượng từ xa cụ thể
- List() liệt kê tất cả các tên đã được đăng ký với trình đăng ký
- Lookup() tìm một đối tượng từ xa cụ thể với một URL cho trước gắn với nó
- Rebind() gán một tên với một đối tượng ở xa khác
- Unbind() loại bỏ một tên đã được gán cho một đối tượng ở xa trong trình đăng ký

Registry.REGISTRY\_PORT là cổng mặc định để lắng nghe các yêu cầu. Giá trị mặc định là 1099.

#### 9.10.2 Lớp LocateRegistry

Lớp java.rmi.registry.LocateRegistry cho phép client tìm trong trình đăng ký trước tiên.

- Public static Registry getRegistry() throws RemoteException
- Public static Registry getRegistry(int port) throws RemoteException
- Public static Registry getRegistry(String host) throws RemoteException
- Public static Registry getRegistry(String host, int port) throws RemoteException
- Public static Registry getRegistry(String host, int port, RMIClientSocketFactory factory) throws RemoteException

Mỗi phương thức trên trả về một đối tượng Registry được sử dụng để nhận các đối tượng từ xa thông qua tên.

Ví dụ để client có tìm thấy đối tượng ta có đăng ký đối tượng đó với trình đăng ký thông qua lớp Registry:

```
Registry r=LocateRegistry.getRegistry();  
r.bind("MyName",this);
```

Client muốn gọi phương thức trên đối tượng từ xa có thể dùng các lệnh sau:

```
Registry r=LocateRegistry.getRegistry(www.somehose.com);
```

```
RemoteObjectInterface  
obj=(RemoteObjectInterface) r.lookup("MyName");  
Obj.invokeRemoteMethod();
```

Ví dụ dưới đây minh họa cách tạo ra một trình đăng ký ngay trong server

```
import java.io.*;  
import java.rmi.*;  
import java.net.*;  
import java.rmi.registry.*;  
public class FileServer  
{  
    public static void main(String[] args) throws Exception  
    {  
        FileInterface fi=new FileImpl("FileServer");  
        InetAddress dc=InetAddress.getLocalHost();  
        LocateRegistry.createRegistry(1099);  
        Naming.bind("rmi://" +dc.getHostAddress()+"/FileServer",fi);  
        System.out.println("Server ready for client requests...");  
    }  
}
```

Như vậy khi thực thi chương trình ta không cần phải khởi động trình đăng ký vì việc tạo ra trình đăng ký và khởi động nó đã được tiến hành ở ngay trong chương trình phía server.

## 9.11 Các lớp và các giao tiếp trong gói java.rmi.server

### 9.11.1 Lớp RemoteObject

Về mặt kỹ thuật đối tượng từ xa không phải là một thể hiện của lớp RemoteObject. Thực tế, phần lớn các đối tượng từ xa là thể hiện của các lớp con của lớp RemoteObject.

### 9.11.2 Lớp RemoteServer

Lớp này là lớp con của lớp RemoteObject; nó là lớp cha của lớp UnicastRemoteObject.

Lớp này có các constructor này:

- Protected RemoteServer()
- Protected RemoteServer(RemoteRef r)

Nhận các thông tin về client

Lớp RemoteServer có một phương thức để xác định thông tin về client mà ta đang truyền tin với nó:

- `public static String getClientHost() throws ServerNotActiveException`

Phương thức này trả về hostname của client mà gọi phương thức từ xa.

### *9.11.3 Lớp UnicastRemoteObject*

Lớp UnicastRemoteObject là một lớp con cụ thể của lớp RemoteServer. Để tạo ra một đối tượng ở xa, ta phải thừa kế lớp UnicastRemoteServer và khai báo lớp này thực thi giao tiếp Remote.

## **9.12 Kết luận**

RMI là một công nghệ phân tán cho phép các phương thức trên các máy ảo Java được gọi từ xa. Đây là cách đơn giản để truyền tin giữa một ứng dụng này với ứng dụng khác so với truyền tin trực tiếp với TCP socket, cách truyền tin này đòi hỏi cả hai phía đều sử dụng cùng một giao thức. Thay vì viết các chương trình cài đặt giao thức, những người phát triển có thể tương tác với các phương thức đối tượng được định nghĩa bởi một giao tiếp dịch vụ RMI. Mỗi khi có được một tham chiếu tới đối tượng từ xa, tham chiếu này có thể được xem như là một đối tượng cục bộ, đây là cách trực quan để phát triển các ứng dụng mạng.



## TÀI LIỆU THAM KHẢO

### Tiếng Việt

- [1] Elliott Rusty Harold, Java Network Programming
- [2] Nguyễn Phương Lan- Hoàng Đức Hải, Java lập trình mạng, Nhà xuất bản Giáo dục
- [3] Darrel Ince & Adam Freemat, Programming the Internet with Java, Addison-Wesley
- [4] Mary Campione&Kathy Walrath&Alison Huml, Java™ Tutorial, Third Edition: A Short Course on the Basics, Addison Wesley The Complete Java.
- [5] Nguyễn Thúc Hải, Mạng máy tính và các hệ thống mở, Nhà xuất bản Giáo dục
- [6] Đoàn Văn Ban, Lập trình hướng đối tượng với Java, Nhà xuất bản Khoa học và Kỹ thuật

### Tiếng Anh

- [7] Douglas E.Comer, David L.Stevens, Client-Server Programming And Applications. In book: Internetworking with TCP/IPVolume III, Pearson Education, Singapore, 2004.
- [8] Herbert Schildt, Java™ 2: The Complete Reference Fifth Edition, Tata McGraw-Hill Publishing Company Limited, India, 2002.
- [9] Elliott Rusty Harold, Java™ Network Programming, Third Edition, Oreilly, 2005.
- [10] Qusay H. Mahmoud, Advanced Socket Programming, <http://java.sun.com>, December 2001
- [11] Shengxi Zhou, Transport Java objects over the network with datagram packets, <http://www.javaworld.com>, 2006