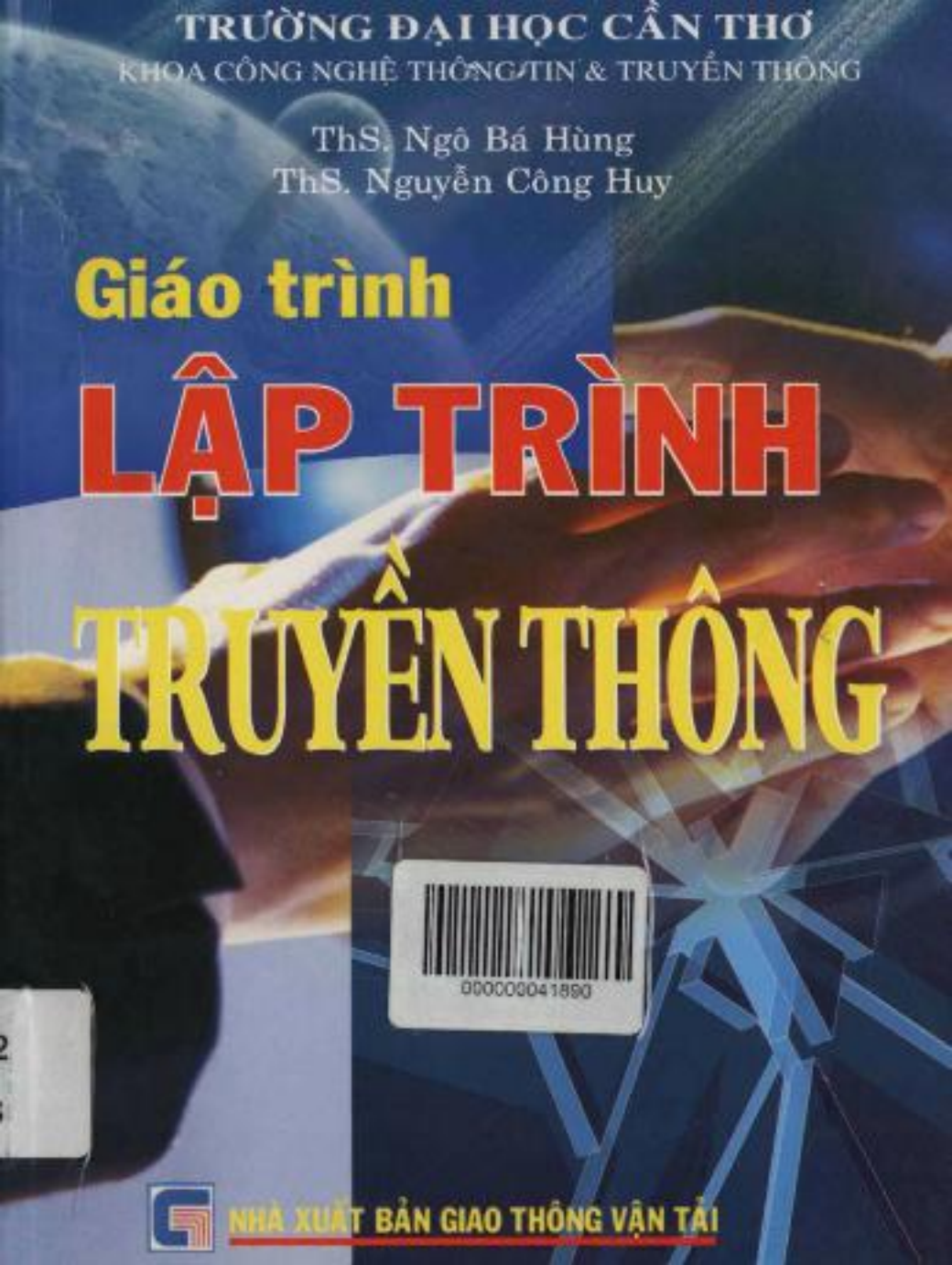
TRƯỜNG ĐẠI HỌC CẦN THƠ
KHOA CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG

ThS. Ngô Bá Hùng
ThS. Nguyễn Công Huy

Giáo trình
LẬP TRÌNH TRUYỀN THÔNG



NHÀ XUẤT BẢN GIAO THÔNG VẬN TẢI

Th.s Ngô Bá Hùng
Th.s Nguyễn Công Huy

GIÁO TRÌNH
LẬP TRÌNH TRUYỀN THÔNG

NHÀ XUẤT BẢN GIAO THÔNG VẬN TẢI

LỜI NÓI ĐẦU

Ngày nay Internet đã trở nên phổ biến đối với tất cả mọi người trong xã hội. Việc truy cập các trang báo điện tử, gửi nhận thư, điện hoa, tìm kiếm thông tin, tán gẫu, viết blog và chơi game trên mạng không còn là điều xa lạ với người dân. Trong các cơ quan xí nghiệp, những ứng dụng xử lý nghiệp vụ cũng được thiết kế và cài đặt theo xu hướng vận hành trên mạng.

Đối với những người trong cuộc, những người mà họ sẽ mang cả thế giới này lên mạng, đó chính là những sinh viên chuyên ngành tin học thì ít nhất một lần nào đó những câu hỏi đại loại như: "Làm thế nào để xây dựng được một web server?", "Làm thế nào để viết một chương trình chat?" hay "Làm thế nào để viết được các chương trình ứng dụng mà trong đó có sự giao tiếp giữa nhiều máy tính trên mạng ?",... đã xuất hiện trong đầu của họ. Cuộc tìm kiếm đã bắt đầu và không ít những người trong số họ đã bỏ cuộc vì đường đi đến câu trả lời còn quá gian nan.

Giáo trình này mong được làm một chiếc gậy nhỏ bé đồng hành với họ trên con đường đi tìm lời giải đáp.

Chúng tôi gọi việc xây dựng các chương trình ứng dụng mà trong đó có sự giao tiếp giữa các máy tính là Lập trình truyền thông. Một số sách hay một số tác giả khác gọi chúng là Lập trình mạng hay Phát triển ứng dụng phân tán.

Trong giáo trình này chúng tôi giới thiệu đến người đọc ba cơ chế chủ yếu là Pipe, Socket và RPC/RMI để xây dựng các loại ứng dụng mà trong đó có sự giao tiếp giữa các thành phần của chúng trong phạm vi một máy tính, hai máy tính hay nhiều hơn thế.

Giáo trình được thiết kế trên cơ sở người học đã có qua kiến thức về lập trình hướng đối tượng với C++. Chúng tôi chọn Java làm ngôn ngữ để minh họa vì nó có tính đa nền, đơn giản, thích hợp cho minh họa những ý tưởng về lập trình truyền thông. Với Java, người học không phải quá chú tâm vào những vấn đề về ngôn ngữ lập trình, nhờ đó giúp họ tập trung hơn vào những vấn đề liên quan đến kỹ thuật lập trình truyền thông. Thực tế, những ý tưởng, những cơ chế được giới thiệu trong giáo trình này có thể dễ dàng được cài đặt bằng các ngôn ngữ lập trình khác.

Mặc dù đã có nhiều cố gắng nhưng chắc chắn vẫn còn nhiều điểm khiếm khuyết trong giáo trình. Nhóm biên soạn mong nhận được nhiều ý kiến đóng góp và xây dựng từ quí đọc giả và đồng nghiệp. Xin chân thành cám ơn.

Cần Thơ, tháng 09 năm 2008
Nhóm biên soạn

GIỚI THIỆU TỔNG QUAN
GIÁO TRÌNH LẬP TRÌNH TRUYỀN THÔNG

A. MỤC ĐÍCH

Giáo trình này nhằm giới thiệu với người đọc những nội dung chủ yếu sau:

- Giới thiệu về các loại kiến trúc chương trình phổ biến hiện nay.
- Giới thiệu về Pipe, Socket, RPC là các cơ chế giao tiếp liên quá trình (IPC- InterProcess Communication) thường được dùng trong việc xây dựng các ứng dụng client-server, các ứng dụng phân tán.
- Giới thiệu về cách sử dụng các tiện ích về Pipe, Socket, RMI của ngôn ngữ Java để xây dựng các ứng dụng Client-Server, ứng dụng phân tán.

B. YÊU CẦU

Sau khi học xong môn này, người học phải có được những khả năng sau:

- Giải thích được cơ chế giao tiếp liên quá trình là gì.
- Trình bày được các kiểu kiến trúc chương trình

trong việc thiết kế một ứng dụng trên mạng.

- Trình bày được đặc điểm của các tiện ích: Pipe, Socket và RPC.
- Hiểu rõ ý nghĩa về giao thức (Protocol).
- Thiết kế và cài đặt một ứng dụng theo một giao thức đã có.
- Xây dựng được giao thức mới để giải quyết các vấn đề đặt ra.
- Xây dựng được các ứng dụng Client-Server, ứng dụng phân tán bằng ngôn ngữ Java với các tiện ích như Pipe, Socket và RMI.

C. NỘI DUNG CỐT LÕI

Giáo trình được chia thành 5 chương với nội dung chủ yếu như sau:

- Chương 1: Giới thiệu tổng quan những vấn đề có liên quan đến lập trình truyền thông.
- Chương 2: Giới thiệu sơ lược về ngôn ngữ Java nhằm giúp người đọc có thể đọc hiểu các ví dụ minh họa và thực hiện được các bài tập yêu cầu trong các chương còn lại.
- Chương 3: Giới thiệu về Pipe, cơ chế giao tiếp liên quá trình cổ điển, điển hình cho cơ chế giao tiếp liên quá trình trên phạm vi một máy tính.

- Chương 4: Giới thiệu về Socket, cơ chế giao tiếp liên quá trình trên hệ thống mạng TCP/IP, cho phép xây dựng các ứng dụng theo mô hình Client-Server.
- Chương 5: Giới thiệu về RPC và RMI cơ chế giao tiếp liên quá trình cho phép xây dựng các ứng dụng phân tán, các ứng dụng đối tượng phân tán.

D. KIẾN THỨC TIÊN QUYẾT

- Kiến thức về Lập trình hướng đối tượng với C++
- Kiến thức về Mạng máy tính

E. TÀI LIỆU THAM KHẢO

1. [ELLIOTTE RUSTY HAROLD],
   Java network programming,
   O'REILLY, 06/1997
2. [PATRICK NAUGHTON & HERNERT SCHILDT],
   Java: The complete Reference,
   Osborne McGraw-Hill,1997
3. [JERRY R. JACKSON & ALAN L. MC CLELLAN],
   Java By Example,

Sunsoft Press A. Prentice Hall Titlem 1997

4. Website http://www.javasoft.com

F. PHƯƠNG PHÁP HỌC TẬP

- Sinh viên đọc tài liệu, làm các bài tập ví dụ, các bài tập mẫu, các bài tập gợi ý và các bài tập kiểm tra theo đúng tiến độ.
- Tham khảo thêm về lập trình hướng đối tượng bằng ngôn ngữ Java.
- Tham khảo thư viện hàm API (Application Programming Interface) của Java về các lớp, các phương thức có liên quan.

Chương 1
TỔNG QUAN VỀ LẬP TRÌNH TRUYỀN THÔNG

Mục đích

Chương này nhằm cung cấp cho các bạn một cái nhìn tổng quan về các vấn đề có liên quan trong lập trình truyền thông.

Yêu cầu

Sau khi hoàn tất chương này, bạn có thể:

- Giải thích được Cơ chế giao tiếp liên quá trình (Inter-Process Communication) là gì.
- Mô tả chức năng, nhiệm vụ của các tầng trong mô hình OSI.
- Định nghĩa về giao thức và biện luận được sự cần thiết của giao thức trong truyền thông.
- Mô tả về bộ giao thức TCP/IP.
- Định nghĩa mô hình Client – Server.
- Phân biệt được 2 chế độ giao tiếp: Nghẽn và Không nghẽn.
- Phân biệt được các kiểu kiến trúc chương trình.

1.1 CƠ CHẾ GIAO TIẾP LIÊN QUÁ TRÌNH LÀ GÌ?

Truyền thông là một khái niệm dùng để chỉ sự giao tiếp, trao đổi thông tin giữa hai hay nhiều thực thể trong một hệ thống nào đó. Nếu hệ thống mà chúng ta xem xét là xã hội loài người, thì truyền thông có thể là quá trình trao đổi thông tin giữa người với người trong cuộc sống thông qua các phương tiện truyền tải thông tin khác nhau như không khí (trong trò chuyện trực tiếp), hệ thống điện thoại, sách, báo, các phương tiện nghe nhìn, mạng máy tính...

Nếu hệ thống mà chúng ta xem xét là một hệ thống máy tính hay một hệ thống mạng thì truyền thông có thể được phân thành hai mức:

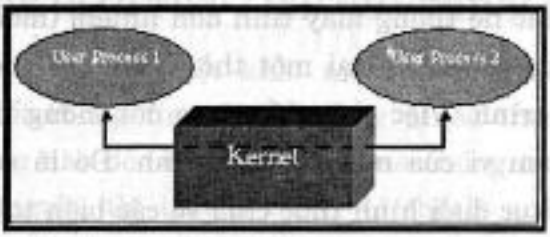
- Mức phần cứng: là sự giao tiếp, trao đổi thông tin giữa các bộ phận vật lý cấu thành nên hệ thống máy tính như CPU, bộ nhớ, thiết bị vào ra, card giao tiếp mạng, nhờ vào các phương tiện truyền thông như hệ thống BUS nội, hệ thống BUS vào ra hay các dây cáp mạng...
- Mức phần mềm: là sự giao tiếp, trao đổi thông tin giữa các thành phần bên trong của một chương trình hay giữa các chương trình với nhau thông qua các cơ chế truyền thông được hỗ trợ bởi các hệ điều hành, hệ điều hành mạng.

Trong các hệ thống máy tính đơn nhiệm (mono-tasking) cổ điển, ví dụ MS-DOS, tại một thời điểm chỉ cho phép tồn tại một quá trình. Việc giao tiếp, trao đổi thông tin chỉ diễn ra trong phạm vi của một chương trình. Đó là sự giao tiếp giữa các thủ tục dưới hình thức chia sẻ các biến toàn cục, hay bằng cách truyền các tham số khi gọi hàm, thủ tục hay bằng giá trị trả về của một hàm.. Ngược lại, trong các hệ thống đa nhiệm (multi-tasking) tại 1 thời điểm có nhiều quá trình tồn tại song song nhau, mỗi quá trình được thực hiện trong một không gian địa chỉ (address space) riêng biệt. Việc giao tiếp giữa các quá trình muốn thực hiện được đòi hỏi phải có những tiện ích hỗ trợ bởi hệ điều hành, hệ điều hành mạng. Các tiện ích này thường được gọi với cái tên là **Cơ chế giao tiếp liên quá trình** (IPC - Inter-Process Communication).

1.2 PHÂN LOẠI CƠ CHẾ GIAO TIẾP LIÊN QUÁ TRÌNH

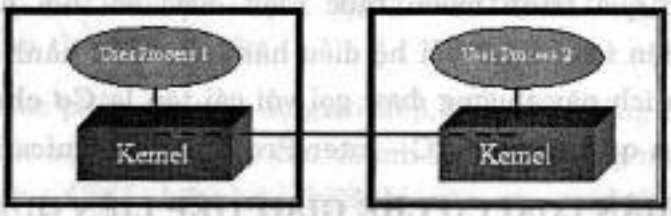
Các cơ chế giao tiếp liên quá trình được hỗ trợ bởi các hệ điều hành đa nhiệm, hệ điều hành mạng được chia ra làm hai loại:

- Loại 1: Cơ chế giao tiếp liên quá trình hỗ trợ giao tiếp giữa các quá trình trên cùng một máy tính (mô tả như trong hình 1.1).



Hình 1.1 - Cơ chế giao tiếp liên quá trình loại 1

- Loại 2: Cơ chế giao tiếp liên quá trình hỗ trợ giao tiếp giữa các quá trình nằm trên các máy tính khác nhau (mô tả trong Hình 1.2).



Hình 1.2 - Cơ chế giao tiếp liên quá trình loại 2

Trong cơ chế giao tiếp liên quá trình trên cùng một máy tính, dữ liệu trao đổi qua lại giữa các quá trình phải đi xuyên qua hạt nhân (kernel) của hệ điều hành. Đó có thể là một vùng nhớ dùng chung cho các quá trình đã được qui định trước bởi hệ điều hành, hay một tập tin trên đĩa được quản lý bởi hệ điều hành, trong đó một quá trình sẽ ghi dữ liệu vào, quá trình khác đọc dữ liệu ra.

Trong cơ chế giao tiếp liên quá trình trên các máy tính khác nhau, dữ liệu trao đổi giữa các quá trình không những phải đi qua hạt nhân như cơ chế giao tiếp liên quá trình trên một máy tính mà hơn thế các hạt nhân của các máy có liên quan phải hiểu nhau. Nói cách khác, các hạt nhân phải thoả thuận trước với nhau về các quy tắc trao đổi thông tin giữa chúng. Thông thường, ta gọi các qui tắc này là các giao thức (protocol).

1.3 MÔ HÌNH THAM KHẢO OSI

Để dễ dàng cho việc nối kết và trao đổi thông tin giữa các máy tính với nhau, vào năm 1983, Tổ chức tiêu chuẩn thế giới ISO đã phát triển một mô hình cho phép hai máy tính có thể gởi và nhận dữ liệu cho nhau. Mô hình này dựa trên tiếp cận phân tầng (lớp), với mỗi tầng đảm nhiệm một số các chức năng cơ bản nào đó và được gọi là mô hình OSI.

Để hai máy tính có thể trao đổi thông tin được với nhau cần có rất nhiều vấn đề liên quan. Ví dụ như cần có card mạng, dây cáp mạng, điện thế tín hiệu trên cáp mạng, cách thức đóng gói dữ liệu, điều khiển lỗi trên đường truyền ... Bằng cách phân chia các chức năng này vào những tầng riêng biệt nhau, việc viết các phần mềm để thực hiện chúng trở nên dễ dàng hơn. Mô hình OSI giúp đồng nhất các hệ thống máy tính khác biệt nhau khi chúng trao đổi thông tin. Mô hình này gồm có 7 tầng:

7. Tầng ứng dụng (Application Layer)

Đây là tầng trên cùng, cung cấp các ứng dụng truy xuất đến các dịch vụ mạng. Nó bao gồm các ứng dụng của người dùng, ví dụ như các Web Browser (Netscape Navigator, Internet Explorer, Mozilla Firefox, ...), các Mail User Agent (Outlook Express, Netscape Messenger, ...) hay các chương trình làm server cung cấp các dịch vụ mạng như các Web Server (Netscape Enterprise, Internet Information Service, Apache, ...), các FTP Server, các Mail server (Send mail, MDeamon). Người dùng mạng giao tiếp trực tiếp với tầng này.

6. Tầng trình bày (Presentation Layer)

Tầng này đảm bảo các máy tính có kiểu định dạng dữ liệu khác nhau vẫn có thể trao đổi thông tin cho nhau. Thông thường các máy tính sẽ thống nhất với nhau về một kiểu định dạng dữ liệu trung gian để trao đổi thông tin giữa các máy tính. Một dữ liệu cần gởi đi sẽ được tầng trình bày chuyển sang định dạng trung gian trước khi nó được truyền lên mạng. Ngược lại, khi nhận dữ liệu từ mạng, tầng trình bày sẽ chuyển dữ liệu sang định dạng riêng của nó.

5. Tầng giao dịch (Session Layer)

Tầng này cho phép các ứng dụng thiết lập, sử dụng và xóa các kênh giao tiếp giữa chúng (được gọi là giao dịch). Nó cung cấp cơ chế cho việc nhận biết tên và các chức năng về bảo mật thông tin khi truyền qua mạng.

4. Tầng vận chuyển (Transport Layer)

Tầng này đảm bảo truyền tải dữ liệu giữa các quá trình. Dữ liệu gởi đi được đảm bảo không có lỗi, theo đúng trình tự, không bị mất mát hay trùng lắp. Đối với các gói tin có kích thước lớn, tầng này sẽ phân chia chúng thành các phần nhỏ trước khi gởi đi, cũng như tập hợp chúng lại khi nhận được.

3. Tầng mạng (Network Layer)

Tầng này đảm bảo các gói dữ liệu (Packet) có thể truyền từ máy tính này đến máy tính kia cho dù không có đường truyền vật lý trực tiếp giữa chúng. Nó nhận nhiệm vụ tìm đường đi cho dữ liệu đến các đích khác nhau trong hệ thống mạng.

2. Tầng liên kết dữ liệu (Data-Link Layer)

Tầng này đảm bảo truyền tải các khung dữ liệu (Frame) giữa hai máy tính có đường truyền vật lý nối trực tiếp với nhau. Nó cài đặt cơ chế phát hiện và xử lý lỗi dữ liệu nhận.

1. Tầng vật lý (Physical Layer)

Điều khiển việc truyền tải thật sự các bit trên đường truyền vật lý. Nó định nghĩa các tín hiệu điện, trạng thái đường truyền, phương pháp mã hóa dữ liệu, các loại đầu nối được sử dụng, ...

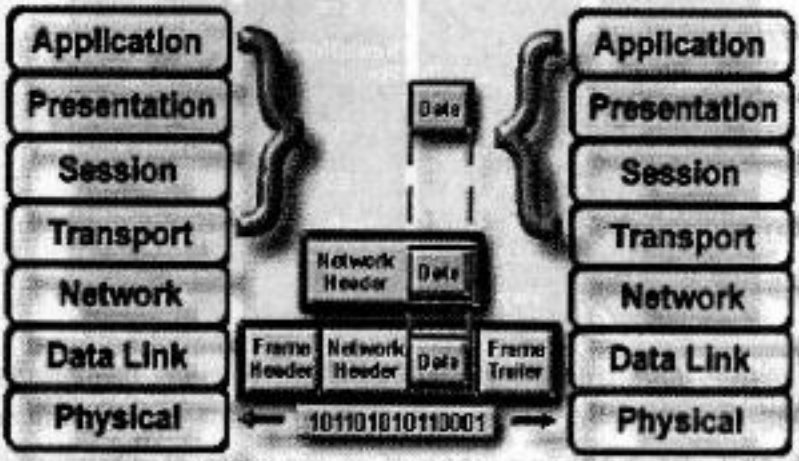
Về nguyên tắc, tầng n của một hệ thống chỉ giao tiếp, trao đổi thông tin với tầng n của hệ thống khác. Mỗi tầng sẽ có các đơn vị truyền dữ liệu riêng:

- Tầng vật lý: bit
- Tầng liên kết dữ liệu: Frame
- Tầng mạng: Packet
- Tầng vận chuyển: Segment

Trong thực tế, dữ liệu được gởi đi từ tầng trên xuống tầng dưới cho đến tầng thấp nhất của máy tính gởi. Ở đó, dữ liệu sẽ được truyền đi trên đường truyền vật lý. Mỗi khi dữ liệu được truyền xuống tầng phía dưới thì nó bị "gói" lại trong đơn vị dữ liệu của tầng dưới. Tại bên nhận, dữ liệu sẽ được truyền ngược lên các tầng cao dần. Mỗi lần qua một tầng, đơn vị dữ liệu tương ứng sẽ được "tháo" ra.

Đơn vị dữ liệu của mỗi tầng sẽ có một tiêu đề (header) riêng, được mô tả trong hình 1.3.

Hình 1.3 - Xử lý dữ liệu qua các tầng



OSI chỉ là mô hình tham khảo, mỗi nhà sản xuất khi phát minh ra hệ thống mạng của mình sẽ thực hiện các chức năng ở từng tầng theo những cách thức riêng. Các cách thức này thường được mô tả dưới dạng các chuẩn mạng hay các giao thức mạng. Như vậy dẫn đến trường hợp cùng một chức năng nhưng hai hệ thống mạng khác nhau sẽ không tương tác được với nhau. Hình 1.4 sẽ so sánh kiến trúc của các hệ điều hành mạng thông dụng với mô hình OSI.

Hình 1.4 - Kiến trúc của một số hệ điều hành mạng thông dụng

Để thực hiện các chức năng ở tầng 3 và tầng 4 trong mô hình OSI, mỗi hệ thống mạng sẽ có các protocol riêng:

- UNIX: Tầng 3 dùng giao thức IP, tầng 4 giao thức TCP/UDP
- Netware: Tầng 3 dùng giao thức IPX, tầng 4 giao thức SPX
- Windows NT: chỉ dùng 1 giao thức NETBEUI

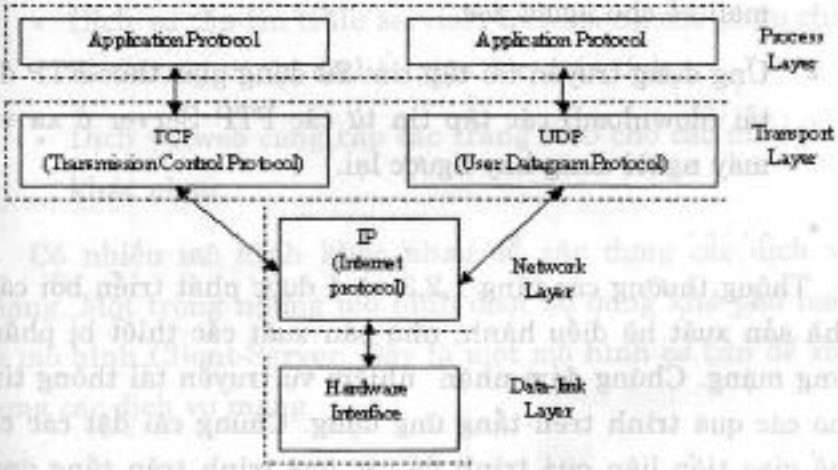
Nếu chỉ dừng lại ở đây thì các máy tính UNIX, Netware, NT sẽ không trao đổi thông tin được với nhau. Với sự lớn mạnh của mạng Internet, các máy tính cài đặt các hệ điều hành khác nhau đòi hỏi phải giao tiếp được với nhau, tức phải sử dụng chung một giao thức. Đó chính là bộ giao thức TCP/IP, giao thức của mạng Internet.

1.4 MẠNG TCP/IP

Đây là kiến trúc của mạng Internet, chỉ gồm 5 tầng như hình vẽ dưới đây:

Hình 1.5 - Kiến trúc mạng TCP/IP

Người ta còn gọi mô hình này là mô hình OSI đơn giản. Các giao thức được sử dụng trên mỗi tầng được qui định như sau:



Hình 1.6 – Bộ giao thức TCP/IP

Tầng 3 sử dụng giao thức IP, tầng 4 có thể sử dụng giao thức TCP ở chế độ có nối kết hoặc UDP ở chế độ không nối kết.

Tầng 5 là tầng của các ứng dụng. Mỗi loại ứng dụng phải định nghĩa một giao thức riêng để các thành phần trong ứng dụng trao đổi thông tin qua lại với nhau. Một số ứng dụng phổ biến đã trở thành chuẩn của mạng Internet như:

- Ứng dụng Web: Sử dụng giao thức HTTP để tải các trang web từ Web Server về Web Browser.
- Ứng dụng thư điện tử: Sử dụng giao thức SMTP để chuyển tiếp mail gởi đi đến Mail Server của người nhận và dùng giao thức POP3 hoặc IMAP để nhận mail về cho người đọc.
- Ứng dụng truyền tải tập tin: Sử dụng giao thức FTP để tải (download) các tập tin từ các FTP Server ở xa về máy người dùng hay ngược lại.
- .....

Thông thường các tầng 1,2,3 và 4 được phát triển bởi các nhà sản xuất hệ điều hành, nhà sản xuất các thiết bị phần cứng mạng. Chúng đảm nhận nhiệm vụ truyền tải thông tin cho các quá trình trên tầng ứng dụng. Chúng cài đặt các cơ chế giao tiếp liên quá trình để các quá trình trên tầng ứng

dụng có thể truy xuất đến dịch vụ truyền tải thông tin do chúng cung cấp. Trong khi đó, tầng 5 là nơi các nhà sản xuất phần mềm khai thác để tạo ra các ứng dụng giải quyết các vấn đề khác nhau của cuộc sống. Nó được xem như là tầng xử lý thông tin.

1.5 DỊCH VỤ MẠNG

Dịch vụ mạng (Net service) là một chương trình ứng dụng thực hiện một tác vụ nào đó trên hệ thống mạng.

Ví dụ:

- Dịch vụ in trên mạng cho phép nhiều máy tính cùng sử dụng một máy in.
- Dịch vụ tập tin (File service) trên mạng cho phép chia sẻ chương trình, dữ liệu giữa các máy tính.
- Dịch vụ web cung cấp các trang web cho các máy tính khác nhau.

Có nhiều mô hình khác nhau để xây dựng các dịch vụ mạng. Một trong những mô hình được sử dụng khá phổ biến là mô hình Client-Server. Đây là một mô hình cơ bản để xây dựng các dịch vụ mạng.

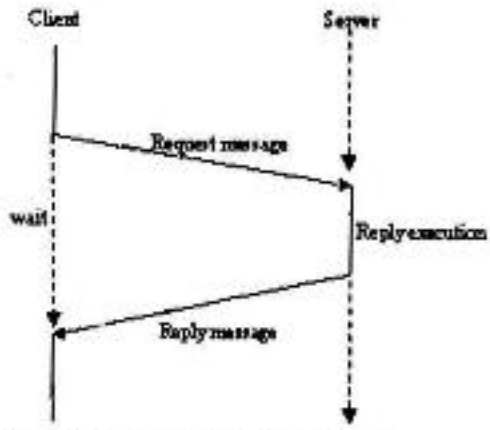
1.6 MÔ HÌNH CLIENT - SERVER

1.6.1 Giới thiệu

Trong mô hình này, chương trình ứng dụng được chia thành 2 thành phần:

- Quá trình chuyên cung cấp một số phục vụ nào đó, chẳng hạn: phục vụ tập tin, phục vụ máy in, phục vụ thư điện tử, phục vụ Web... Các quá trình này được gọi là các trình phục vụ hay **Server**.
- Một số quá trình khác có yêu cầu sử dụng các dịch vụ do các server cung cấp được gọi là các quá trình khách hàng hay **Client**.

Việc giao tiếp giữa client và server được thực hiện dưới hình thức trao đổi các thông điệp (Message). Để được phục vụ, client sẽ gởi một thông điệp yêu cầu (Request Message) mô tả về công việc muốn server thực hiện.



Hình 1.7 - Mô hình Client-Server

Khi nhận được

thông điệp yêu cầu, server tiến hành phân tích để xác định công việc cần phải thực thi. Nếu việc thực hiện yêu cầu này có sinh ra kết quả trả về, server sẽ gởi nó cho client trong một thông điệp trả lời (Reply Message). Dạng thức (format) và ý nghĩa của các thông điệp trao đổi giữa client và server được qui định rõ bởi giao thức (protocol) của ứng dụng.

1.6.2 Ví dụ về dịch vụ Web.

Dịch vụ web được tổ chức theo mô hình Client -Server, trong đó:

- Web server sẵn sàng cung cấp các trang web đang được lưu trữ trên đĩa cứng cục bộ của mình.
- Web Client, còn gọi là các Browser, có nhu cầu nhận các trang web từ các Web Server
- HTTP là giao thức trao đổi thông tin qua lại giữa Web client và Web Server.
- Thông điệp yêu cầu là một chuỗi có dạng sau:

```
Command URL HTTP/Ver \n\n
```

- Thông điệp trả lời có dạng sau:

```
<HEADER>\n\n
<CONTENT>
```

- Giả sử Client cần nhận trang Web ở địa chỉ

http://www.cit.ctu.edu.vn/, nó sẽ gởi đến Web Server có tên www.cit.ctu.edu.vn thông điệp yêu cầu sau:

GET www.cit.ctu.edu.vn HTTP/1.1\n\n

- Server sẽ gởi về nội dung sau:

HTTP/1.0 200 OK
Date: Mon, 24 Nov 2003 02:43:46 GMT
Server: Apache/1.3.23 (Unix) (Red-Hat/Linux)
mod_ssl/2.8.7 OpenSSL/0.9.6b DAV/1
.0.3 PHP/4.1.2 mod_perl/1.26
Last-Modified: Tue, 01 Jul 2003 08:08:52 GMT
ETag: "17f5d-2abb-3f014194"
Accept-Ranges: bytes
Content-Length: 10939
Content-Type: text/html
X-Cache: HIT from proxy.cit.ctu.edu.vn
Proxy-Connection: close

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

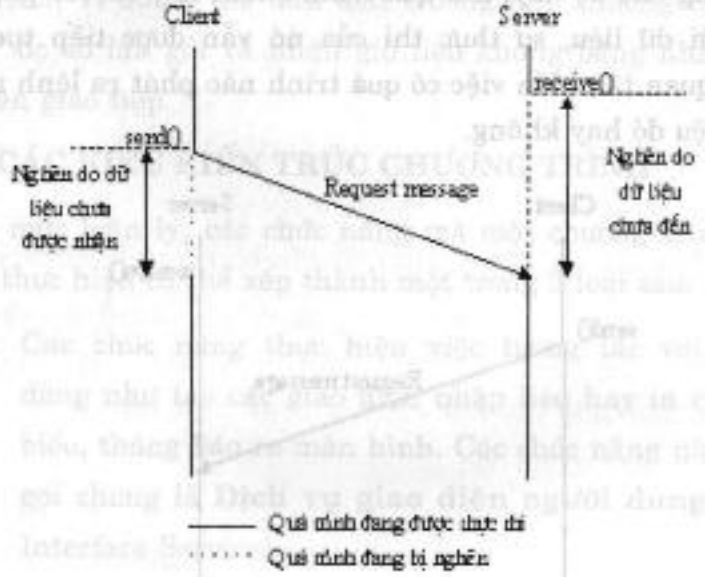
<HTML>
<HEAD>
<LINK href="favicon.ico" rel="SHORTCUT ICON">

.....

1.6.3 Các chế độ giao tiếp

Quá trình giao tiếp giữa client và server có thể diễn ra theo hai chế độ là nghẽn (blocked) hay không nghẽn (Non blocked).

1.6.3.1 Chế độ nghẽn:



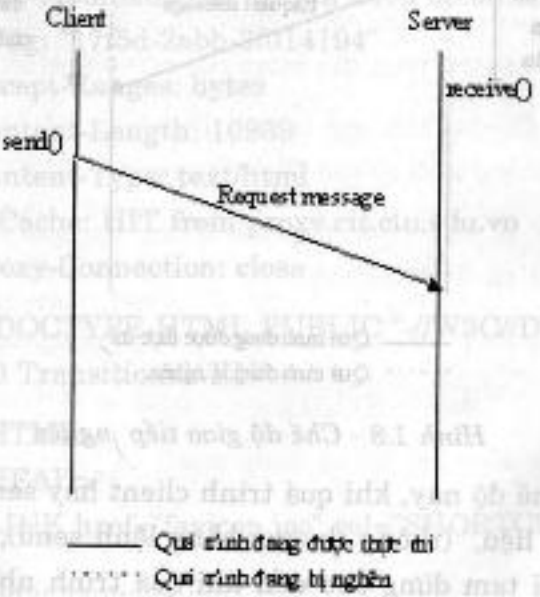
Hình 1.8 - Chế độ giao tiếp nghẽn

Trong chế độ này, khi quá trình client hay server phát ra lệnh gởi dữ liệu, (thông thường bằng lệnh send), sự thực thi của nó sẽ bị tạm dừng cho đến khi quá trình nhận phát ra lệnh nhận số dữ liệu đó (thường là lệnh receive).

Tương tự cho trường hợp nhận dữ liệu, nếu quá trình nào đó, client hay server, phát ra lệnh nhận dữ liệu, mà ở thời điểm đó chưa có dữ liệu gởi đến, sự thực thi của nó cũng tạm dừng cho đến khi có dữ liệu gởi đến.

1.6.3.2 Chế độ không nghẽn:

Trong chế độ này, khi quá trình client hay server phát ra lệnh gởi dữ liệu, sự thực thi của nó vẫn được tiếp tục mà không quan tâm đến việc có quá trình nào phát ra lệnh nhận số dữ liệu đó hay không.



Hình 1.9 - Chế độ giao tiếp không nghẽn

Tượng tự cho trường hợp nhận dữ liệu, khi quá trình phát ra lệnh nhận dữ liệu, nó sẽ nhận được số lượng dữ liệu hiện có (bằng 0 nếu chưa có quá trình nào gởi dữ liệu đến). Sự thực thi của quá trình vẫn được tiếp tục.

Trong thực tế, cần chú ý đến chế độ giao tiếp nghẽn khi lập trình, vì nó có thể dẫn đến trường hợp chương trình bị "treo" do số lần gởi và nhận giữ liệu không bằng nhau giữa hai bên giao tiếp.

1.7 CÁC KIỂU KIẾN TRÚC CHƯƠNG TRÌNH

Ở mức luận lý, các chức năng mà một chương trình ứng dụng thực hiện có thể xếp thành một trong 3 loại sau:

1. Các chức năng thực hiện việc tương tác với người dùng như tạo các giao diện nhập liệu hay in các báo biểu, thông báo ra màn hình. Các chức năng này được gọi chung là **Dịch vụ giao diện người dùng** (User Interface Service).

2. Các chức năng tính toán các dữ liệu, xử lý thông tin theo những qui luật (rule), giải thuật được qui định bởi vấn đề mà ứng dụng giải quyết. Các chức năng này được gọi chung là **Dịch vụ nghiệp vụ** (Business Rule Service).

3. Trong quá trình tính toán, chương trình ứng dụng cần truy vấn đến các thông tin đã có, được lưu trên đĩa cứng hay trong các cơ sở dữ liệu. Ngoài ra, chương trình ứng dụng cũng cần phải lưu lại các kết quả tính toán được để sử dụng về sau. Các chức năng này được gọi chung là **Dịch vụ lưu trữ** (Data Storage Service).

Ở mức vật lý, các chức năng này có thể được cài đặt vào một hay nhiều tập tin thực thi hình thành các kiểu kiến trúc chương trình khác nhau. Cho đến thời điểm hiện nay, người ta chia kiến trúc của chương trình thành 3 loại được trình bày tiếp theo sau.

1.7.1 Kiến trúc đơn tầng (Single-tier Architecture)

Trong kiểu kiến trúc này, cả 3 thành phần của chương trình ứng dụng (User Interface, Business Rule, Data Storage) đều được tích hợp vào một chương trình thực thi.

Ví dụ: BKAV, D2, Winword, . . .

Các ứng dụng kiểu này chỉ được thực thi trên một máy tính.

Hình 1.10 - Kiến trúc chương trình đơn tầng

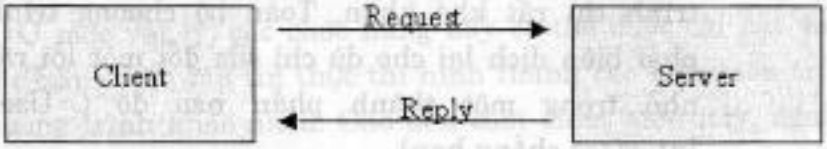
- **Ưu điểm:**
  - Dễ dàng trong thiết kế cài đặt ứng dụng kiểu này.
- **Nhược điểm:**
  - Bởi vì cả 3 thành phần được cài vào một tập tin thực thi, nên việc sửa lỗi hay nâng cấp chương trình thì rất khó khăn. Toàn bộ chương trình phải biên dịch lại cho dù chỉ sửa đổi một lỗi rất nhỏ trong một thành phần nào đó (User Interface chẳng hạn).
  - Việc bảo trì, nâng cấp ấn bản mới là một công việc cực kỳ nặng nề vì ta phải thực hiện việc cài đặt trên tất cả các máy tính.
  - Trong kiểu này, mỗi máy tính duy trì một cơ sở dữ liệu riêng cho nên rất khó trong việc trao đổi, tổng hợp dữ liệu.
  - Máy tính phải đủ mạnh để có thể thực hiện đồng thời cả 3 loại dịch vụ.

1.7.2 Kiến trúc hai tầng (Two - Tier Architecture)

Kiến trúc này còn được biết đến với tên kiến trúc Client-Server. Kiến trúc này gồm 2 chương trình thực thi: chương trình Client và chương trình Server. Cả hai chương trình có thể được thực thi trên cùng một máy tính hay trên hai máy

tính khác nhau.

Client và Server trao đổi thông tin với nhau dưới dạng các thông điệp (Message) . Thông điệp gởi từ Client sang Server gọi là các thông điệp yêu cầu (Request Message) mô tả công việc mà phần Client muốn Server thực hiện.



Hình 1.11 - Kiến trúc chương trình Client-Server

Mỗi khi Server nhận được một thông điệp yêu cầu, Server sẽ phân tích yêu cầu, thực thi công việc theo yêu cầu và gởi kết quả về client (nếu có) trong một thông điệp trả lời (Reply Message). Khi vận hành, một máy tính làm Server phục vụ cho nhiều máy tính Client.

Mỗi một ứng dụng Client-Server phải định nghĩa một **Giao thức** (Protocol) riêng cho sự trao đổi thông tin, phối hợp công việc giữa Client và Server. Protocol qui định một số vấn đề cơ bản sau:

- Khuôn dạng loại thông điệp.
- Số lượng và ý nghĩa của từng loại thông điệp.
- Cách thức bắt tay, đồng bộ hóa tiến trình truyền nhận

giữa Client và Server.

- ....

Thông thường phần client đảm nhận các chức năng về User Interface, như tạo các form nhập liệu, các thông báo, các báo biểu giao tiếp với người dùng.

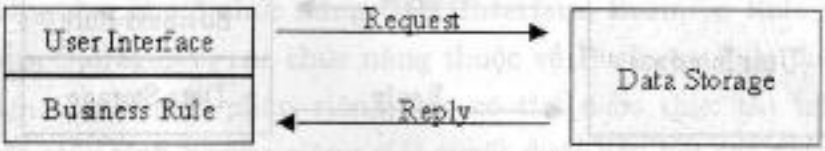
Phần Server đảm nhận các chức năng về Data Storage. Nhờ đó dễ dàng trong việc bảo trì, chia sẻ tổng hợp dữ liệu trong toàn bộ công ty hoặc tổ chức.

Các chức năng về Business Rule có thể được cài đặt ở phần client hoặc ở phần server tạo ra hai loại kiến trúc Client - Server là:

- Fat Client
- Fat Server.

1.7.2.1 Loại Fat Client

Trong loại này Business Rule được cài đặt bên phía Client. Phần Server chủ yếu thực hiện chức năng về truy vấn và lưu trữ thông tin.



Hình 1.12 - Kiến trúc chương trình Client – Server theo kiểu Fat Client

Ưu điểm

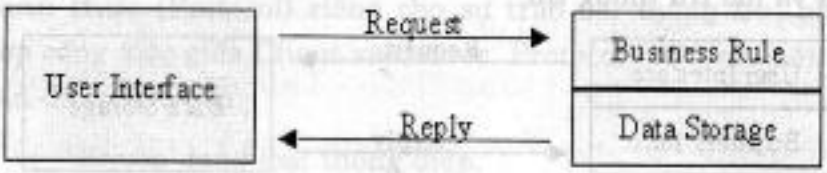
- Tạo ra ít giao thông trên mạng nhờ dữ liệu tạm thời trong quá trình tính toán được lưu tại Client.

Nhược điểm

- Vì Business Rule được cài đặt trên phía Client, đòi hỏi máy tính thực thi phần Client phải đủ mạnh, dẫn đến tốn kém trong chi phí đầu tư phần cứng cho các công ty xí nghiệp.
- Phải cài lại tất cả các máy tính Client khi nâng cấp chương trình

1.7.2.2 Loại Fat Server

Trong loại này, phần lớn các chức năng về Business Rule được đặt ở phần Server. Phần Client chỉ thực hiện một số chức năng nhỏ của Business Rule về kiểm tra tính hợp lệ của dữ liệu nhập bởi người dùng.



Hình 1.13 - Kiến trúc chương trình Client – Server theo kiểu Fat Server

Ưu điểm

- Vì Business Rule được đặt ở phần Server, các máy tính Client không cần phải có cấu hình mạnh.
- Việc nâng cấp chương trình khi Business Rule thay đổi trở nên nhẹ nhàng hơn vì chỉ phải cài đặt lại phần Server.

Nhược điểm

- Tạo ra nhiều thông điệp trao đổi giữa Client và Server làm tăng giao thông trên mạng.
- Tăng tải trên máy Server vì nó phải đồng thời thực hiện các chức năng của Business Rule và Data Storage làm giảm hiệu năng của chương trình.

1.7.3 Kiến trúc đa tầng (N-Tier Architecture)

Đây là kiến trúc cho các **Ứng dụng phân tán** (Distributed Application). Thông thường là kiến trúc 3 tầng. Chương trình ứng dụng được tách thành 3 phần riêng biệt tương ứng cho 3 chức năng User Interface, Business Rule và Data Storage. Vì các chức năng thuộc về Business Rule được tách thành một phần riêng, nó có thể được thực thi trên một máy tính Server riêng giải quyết được hầu hết các nhược điểm mắc phải của kiến trúc đơn tầng và kiến trúc hai tầng nói trên.

Hình 1.12 - Kiến trúc chương trình đa tầng

Kiến trúc này đáp ứng tốt với những thay đổi về qui luật xử lý dữ liệu của vấn đề mà ứng dụng giải quyết. Việc thay đổi chỉ ảnh hưởng trên tầng Business Rule mà không ảnh hưởng đến hai tầng còn lại.

Thông thường, người ta gọi tên các thành phần trên là:

Client – Application Server – Database Server

1.8 BÀI TẬP

1.8.1 Bài tập bắt buộc

Bài tập 1.1: Protocol HTTP

Tìm đọc và viết một báo cáo không quá 10 trang về giao thức HTTP.

Bài tập 1.2: Chat Protocol

Tìm hiểu về dịch vụ Chat trên mạng Internet. Viết một bảng báo cáo không qua 10 trang trình bày 2 nội dung sau:

- Một bảng mô tả các chức năng thường được hỗ trợ

trong một dịch vụ Chat.

- Xây dựng Chat Protocol riêng của bạn trong đó mô tả:
  - Các chức năng hỗ trợ bởi Chat Server.
  - Khuôn dạng (Format) và các loại thông điệp (Message) hỗ trợ bởi Protocol.
  - Sơ đồ trạng thái hoạt động của server và client (giải thuật).
  - Minh họa các kịch bản khác nhau cho từng chức năng của dịch vụ.

1.8.2 Bài tập gợi ý

Bài tập 1.3: POP3 Protocol

Tìm đọc và viết một báo cáo không quá 10 trang về giao thức POP3.

Chương 2
SƠ LƯỢC VỀ NGÔN NGỮ JAVA

Mục đích

Chương này nhằm giới thiệu sơ lược về ngôn ngữ Java cho các sinh viên đã có kiến thức căn bản về Lập trình hướng đối tượng với C++. Chương này sẽ không đề cập đến tất cả các vấn đề có trong Java mà chỉ giới thiệu những vấn đề cơ bản nhất về ngôn ngữ Java, đủ để các học viên có thể đọc hiểu các chương trình minh họa và làm được các bài tập ứng dụng ở các chương sau.

Yêu cầu

Sau khi hoàn tất chương này, bạn có thể:

- Trình bày được những vấn đề tổng quan về ngôn ngữ Java như:
  - Đặc điểm và khả năng của ngôn ngữ Java.
  - Khái niệm máy ảo của Java (JVM - Java Virtual Machine).
  - Vai trò của bộ phát triển ứng dụng JDK (Java Development Kit).

- Phân biệt được hai kiểu chương trình Applet và Application của Java.
- Các kiểu dữ liệu và các phép toán được hỗ trợ bởi Java.

- Biên soạn, biên dịch và thực thi thành công chương trình HelloWorld .
- Sử dụng thành thạo các cấu trúc điều khiển dưới Java như: if, switch, while, do-while, for.
- Biết cách nhận đối số của chương trình Java.
- Biết đổi qua lại các kiểu: chuỗi, số và mảng các byte trong Java.
- Sử dụng được cơ chế ngoại lệ của Java.
- Biết định nghĩa lớp mới, sử dụng một lớp đã có của Java.
- Giải thích được cơ chế vào ra với Stream trong Java.
- Sử dụng thành thạo các phương thức của hai lớp InputStream và OutputStream.
- Có thể nhập / xuất chuỗi trên một InputStream / OutputStream.
- Giải thích được cơ chế luồng (Thread).
- Cài đặt được các luồng trong Java.

2.1 GIỚI THIỆU VỀ NGÔN NGỮ JAVA

2.1.1 Lịch sử phát triển

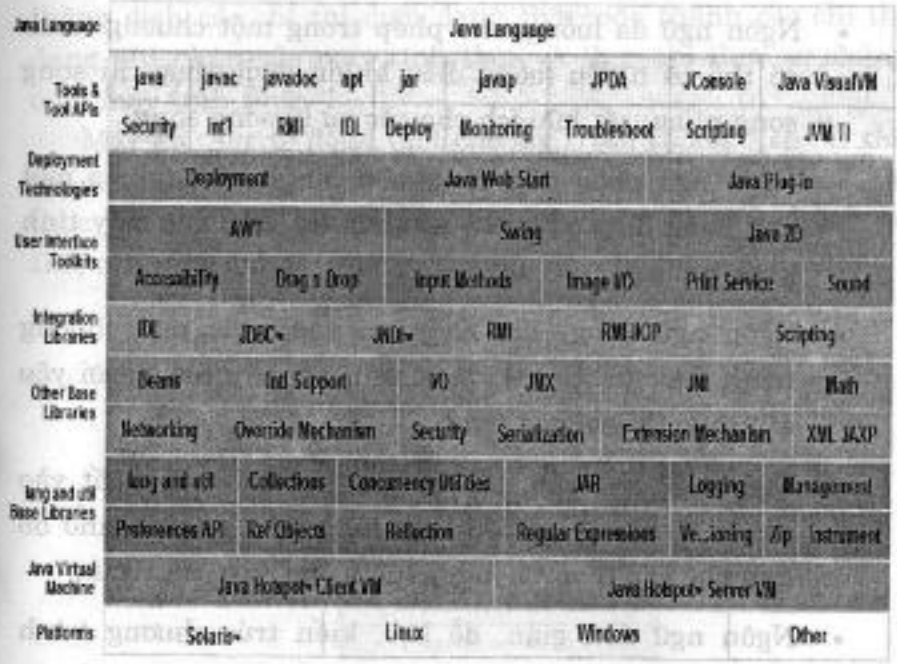
Năm 1990, Sun MicroSystems thực hiện dự án Green nhằm phát triển phần mềm trong các thiết bị dân dụng. James Gosling, chuyên gia lập trình đã tạo ra một ngôn ngữ lập trình mới có tên là Oak. Ngôn ngữ này có cú pháp gần giống như C++ nhưng bỏ qua các tính năng nguy hiểm của C++ như truy cập trực tiếp tài nguyên hệ thống, con trỏ, định nghĩa chồng các tác tử...

Khi ngôn ngữ Oak trưởng thành, WWW cũng đang vào thời kỳ phát triển mạnh mẽ, Sun cho rằng đây là một ngôn ngữ thích hợp cho Internet. Năm 1995, Oak đổi tên thành Java và sau đó đến 1996 Java đã được xem như một chuẩn công nghiệp cho Internet.

2.1.2 Khả năng của ngôn ngữ Java

- Là một ngôn ngữ bậc cao như C, C++, Perl, SmallTalk,.. nên có thể được dùng để tạo ra các ứng dụng để giải quyết các vấn đề về số, xử lý văn bản, tạo ra trò chơi, và nhiều thứ khác.
- Có các thư viện hàm hỗ trợ xây dựng giao diện đồ họa (GUI) cho ứng dụng như AWT, Swing.
- Có các môi trường lập trình đồ họa như Symantec Cafe, Borland JBuilder, NetBeans, Eclipse, ...
- Có khả năng truy cập dữ liệu từ xa thông qua cầu nối JDBC (Java DataBase Connectivity)

- Hỗ trợ các lớp hữu ích, tiện lợi trong lập trình các ứng dụng mạng (Socket) cũng như truy xuất Web.
- Hỗ trợ lập trình phân tán (RMI - Remote Method Invocation) cho phép một ứng dụng có thể được xử lý phân tán trên các máy tính khác nhau.
- Hỗ trợ lập trình trên các thiết bị cầm tay (J2ME).
- Hỗ trợ phát triển các ứng dụng trong môi trường xí nghiệp (J2EE).
- Java luôn được bổ sung các tính năng và tiện ích mới trong các phiên bản sau.



Hình 2.1 – Bộ phát triển ứng dụng J2SE 1.6

2.1.3 Những đặc điểm của ngôn ngữ Java

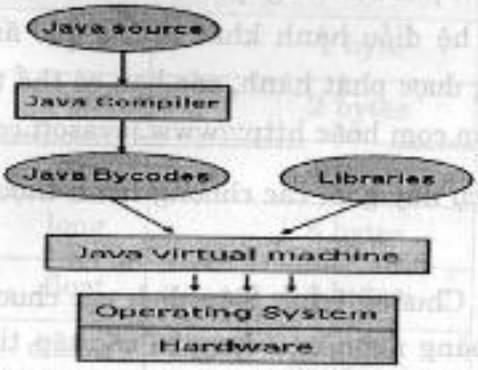
- Ngôn ngữ hoàn toàn hướng đối tượng (Pure Object Oriented Programming).
- Ngôn ngữ đa nền cho phép một chương trình có thể thực thi trên các hệ điều hành khác nhau (MS Windows, UNIX, Linux, ...) mà không phải biên dịch lại chương trình. Phương châm của Java là "**Viết một lần , chạy trên nhiều nền**" (Write Once, Run Anywhere).
- Ngôn ngữ đa luồng, cho phép trong một chương trình có thể có nhiều luồng điều khiển được thực thi song song nhau, rất hữu ích cho các xử lý song song.
- Ngôn ngữ phân tán, cho phép các đối tượng của một ứng dụng được phân bố và thực thi trên các máy tính khác nhau.
- Ngôn ngữ động, cho phép mã lệnh của một chương trình được tải từ một máy tính về máy của người yêu cầu thực thi chương trình.
- Ngôn ngữ an toàn, tất cả các thao tác truy xuất vào các thiết bị vào ra đều thực hiện trên máy ảo nhờ đó hạn chế các thao tác nguy hiểm cho máy tính thật.
- Ngôn ngữ đơn giản, dễ học, kiến trúc chương trình đơn giản, trong sáng.

2.1.4 Máy ảo Java (JMV - Java Virtual Machine)

Để đảm bảo tính đa nền, Java sử dụng cơ chế **Máy ảo của Java**. ByteCode đó là ngôn ngữ máy của Máy ảo Java tương tự như các lệnh nhị phân của các máy tính thực. Một chương trình sau khi được viết bằng ngôn ngữ Java (có phần mở rộng là .java) phải được biên dịch thành tập tin thực thi được trên máy ảo Java (có phần mở rộng là .class). Tập tin thực thi này chứa các chỉ thị dưới dạng mã Bytecode mà máy ảo Java hiểu được phải làm gì.

Khi thực hiện một chương trình, máy ảo Java lần lượt thông dịch các chỉ thị dưới dạng Bytecode thành các chỉ thị dạng nhị phân của máy tính thực và thực thi thực sự chúng trên máy tính thực.

Máy ảo thực tế đó là một chương trình thông dịch. Vì thế các hệ điều hành khác nhau sẽ có các máy ảo khác nhau. Để thực thi một ứng dụng của Java trên một hệ điều hành cụ thể, cần phải cài đặt máy ảo tương ứng cho hệ điều hành đó.



Hình 2.2 – Cơ chế thực thi chương trình Java thông qua máy ảo

2.1.5 Hai kiểu ứng dụng dưới ngôn ngữ java

Khi bắt đầu thiết kế một ứng dụng dưới ngôn ngữ Java, bạn phải chọn kiểu cho nó là **Application** hay **Applet.**

- **Applet:** Là một chương trình ứng dụng được nhúng vào các trang web. Mã của chương trình được tải về máy người dùng từ Web server khi người dùng truy xuất đến trang web chứa nó.
- **Application:** Là một chương trình ứng dụng được thực thi trực tiếp trên các máy ảo của Java.

2.1.6 Bộ phát triển ứng dụng Java (JDK- Java Development Kit)

JDK là một bộ công cụ cho phép người lập trình phát triển và triển khai các ứng dụng bằng ngôn ngữ java được cung cấp miễn phí bởi công ty JavaSoft (hoặc Sun). Có các bộ Jdk cho các hệ điều hành khác nhau. Các ấn bản của JDK không ngừng được phát hành, các bạn có thể tải về từ địa chỉ http://java.sun.com hoặc http://www.javasoft.com

Bộ công cụ này gồm các chương trình thực thi đáng chú ý sau:

- javac: Chương trình biên dịch các chương trình nguồn viết bằng ngôn ngữ Java ra các tập tin thực thi được trên máy ảo Java.

- java: Đây là chương trình làm máy ảo của Java, thông dịch mã Bytecode của các chương trình kiểu application thành mã thực thi của máy thực.
- appletviewer: Bộ thông dịch, thực thi các chương trình kiểu applet.
- javadoc: Tạo tài liệu về chú thích chương trình nguồn một cách tự động.
- jdb: Trình gở rối.
- rmic: Tạo Stub cho ứng dụng kiểu RMI.
- rmiregistry: Phục vụ danh bạ (Name Server) trong hệ thống RMI

2.1.7 Kiểu dữ liệu cơ bản dưới Java

- Kiểu số

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

- **Kiểu ký tự char**

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

- **Kiểu chuỗi ký tự String**

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

- **Kiểu luận lý boolean**

  Nhận 2 giá trị là : true và false.

- **Kiểu mảng**
  - Khai báo:
    - int[] a ; float[] yt; String[] names;
    - int a[]; float yt[]; String names[];
    - int maTran[][]; float bangDiem[][];
  - Khởi tạo:
    - a = new int[3]; yt = new float[10]; names = new String[50];
    - maTran = int[10][10];
  - Sử dụng mảng:
    - int i = a[0]; float f = yt[9]; String str = names[20];
    - Int x = matran [2][5];

2.1.8 Các phép toán cơ bản

Các phép toán trong Java cũng tương tự như trong C++.

- Phép toán số học: +, - , *, / , % , =,++ , -- , += , - = , *= , /= , %=
- Phép toán logic ==, !=, && , | |, ! ,> , < , >= , <=
- Phép toán trên bit : & , | , ^ , << , >> , ~
- Phép toán điều kiện : ? :
- Phép chuyển đổi kiểu: (Kiểu Mới)

2.1.9 Qui cách đặt tên trong Java

Tên hằng, biến, lớp, phương thức, ... được đặt tên theo qui tắc bắt buộc sau:

- Tên phân biệt giữa chữ HOA và chữ thường.
- Dùng các chữ cái, ký tự số, ký tự _ và $.
- Không bắt đầu bằng ký tự số.
- Không có khoảng trắng trong tên.

Để chương trình nguồn dễ đọc, dễ theo dõi người ta còn sử dụng quy ước đặt tên sau (không bắt buộc):

- Tên lớp:
  - Các ký tự đầu tiên của một từ được viết hoa,

- Các ký tự còn lại viết thường.

Ví dụ: lớp Nguoi, SinhVien, MonHoc, String, InputStream, OutputStream. . .

- Tên biến, tên hằng, tên phương thức:
  - Từ đầu tiên viết thường.
  - Ký tự đầu tiên của từ thứ hai trở đi được viết hoa.

Ví dụ: ten, ngaySinh, diaChi, inTen(), inDiaChi(), getInputStream(), ...

- Vị trí đặt dấu { và } để bắt đầu và kết thúc các khối như sau:

```
if (condition)  {
   command 1;
   command 2;
 }
else {
   command 3;
   command 4;
 }
```

2.2 CHƯƠNG TRÌNH ỨNG DỤNG KIỂU APPLICATION

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

Lớp thực thi được này có các đặc điểm sau:

- Có tên lớp trùng với tên tập tin chứa nó.
- Phải khai báo phạm vi là public
- Có chứa phương thức:
  public static void main (String args[]){
  ...
  }
  là phương thức được thực thi đầu tiên.
- Nếu nhiều lớp được định nghĩa trong một tập tin, chỉ có một lớp được khai báo public.

2.2.1 Chương trình HelloWorld

Trong ví dụ này, chúng ta viết một chương trình ứng

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

```
public class HelloWorld {
    public static void main(String args[]) {
        System.out.print("Hello World! \n");
    }
}
```

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

Ta có thể dùng bất kỳ chương trình soạn thảo văn bản nào để biên soạn chương trình. Nhưng bắt buộc phải ghi lại với phần mở rộng là .java.

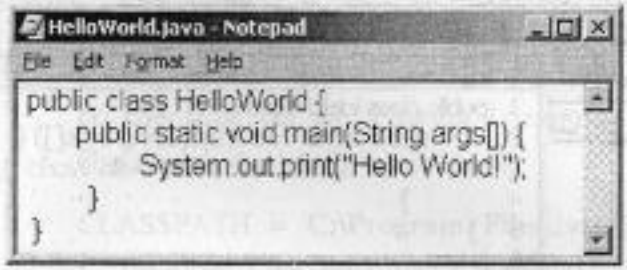
2.2.2 Biên soạn chương trình

a) Sử dụng phần mềm Notepad của MS-Windows

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

- Chạy chương trình Notepad:
  - Chọn menu Start \ Programs \ Accessories \ Notepad

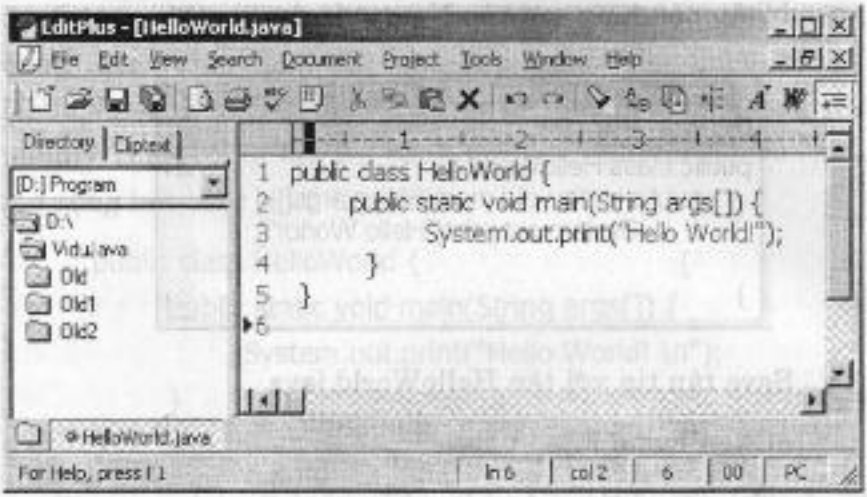
- Nhập nội dung sau vào Notepad



- Save tập tin với tên HelloWorld.java
  - Chọn menu File \ Save
  - Tại cửa sổ Save As hãy nhập vào:
    - Save in: Thư mục nơi sẽ lưu tập tin
    - File Name: HelloWorld.java
    - Save as type: All Files
    - Nhấp vào nút Save

b) Sử dụng phần mềm hỗ trợ soạn thảo EditPlus

- Chạy chương trình EditPlus
- Chọn File / New / Java
- Nhập nội dung vào EditPlus
- Save tập tin với tên: HelloWorld.java



Ngoài các cách trên, còn có rất nhiều các IDE mạnh (chẳng hạn như Jcreator, JBuilder, NetBeans, Eclipse) dùng để phát triển các ứng dụng viết bằng Java có tính năng hỗ trợ soạn thảo chương trình rất tiện lợi.

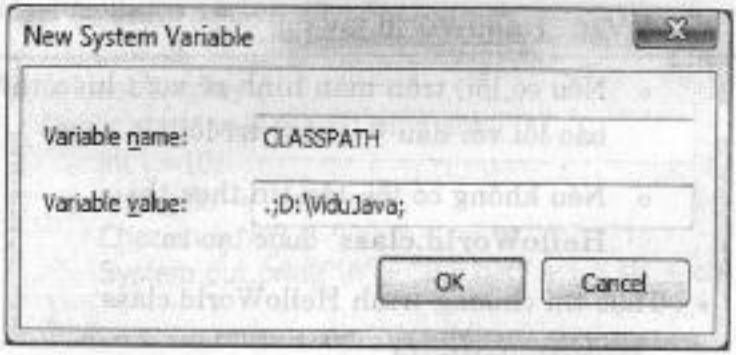
2.2.3 Cài đặt và cấu hình bộ phát triển ứng dụng JDK

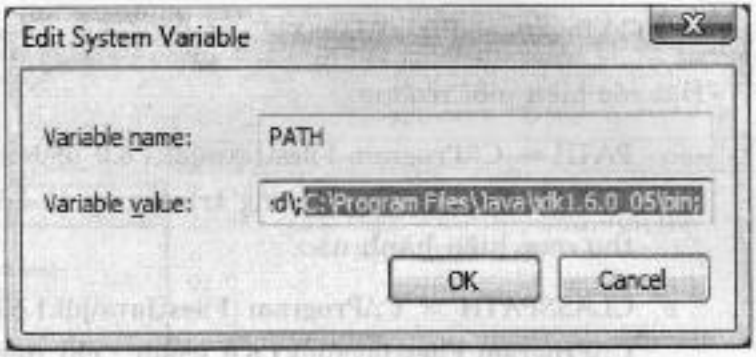
- Chuẩn bị bộ nguồn cài đặt JDK phù hợp với hệ điều hành sử dụng (Giả sử Windows Vista): chẳng hạn file **jdk-6u5-windows-i586-p.exe**
- Click đúp vào tập tin trên để tiến hành cài đặt.
- Thư mục mặc định:

C:\Program Files\Java\jdk1.6.0_05

- Đặt các biến môi trường:
  - PATH = **C:\Program Files\Java\jdk1.6.0_05\bin;** để có thể thực thi các chương trình này từ bất kỳ thư mục hiện hành nào.
  - CLASSPATH = **C:\Program Files\Java\jdk1.6.0_05; C:\Program Files\Java\jdk1.6.0_05\lib;.;** chỉ đến các lớp thư viện của Java trong thư mục **C:\Program Files\Java\jdk1.6.0_05**, thư mục con **lib** và các lớp tại thư mục hiện hành, thể hiện bằng dấu chấm(.).

Trong Windows, ta đặt các biến môi trường này trong mục Environment Variables của System Properties của hệ thống.





2.2.4 Biên dịch và thực thi chương trình

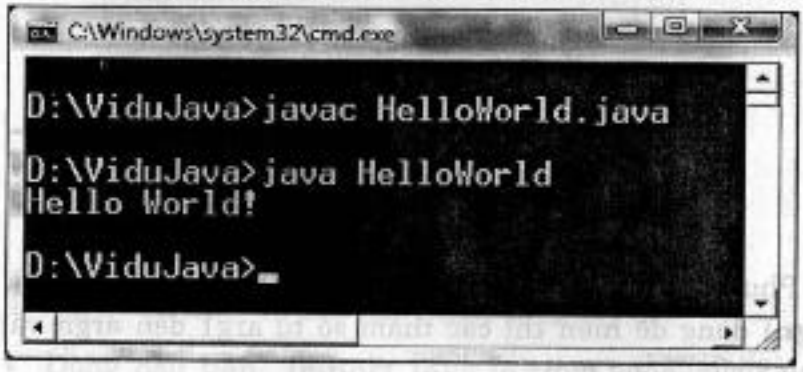
- Mở cửa sổ MS-DOS: Chọn menu Start \ Programs \ Accessories \ Command Prompt (Chọn chọn Start / Run, gõ lệnh **cmd**)
- Chuyển vào thư mục chứa tập tin HelloWorld.java
- Dùng chương trình javac để biên dịch tập tin HelloWorld.java

```
javac HelloWorld.java
```

  - Nếu có lỗi, trên màn hình sẽ xuất hiện thông báo lỗi với dấu ^ chỉ vị trí lỗi.
  - Nếu không có lỗi, tập tin thực thi **HelloWorld.class** được tạo ra.
- Thực thi chương trình HelloWorld.class

```
java HelloWorld
```

Trên màn hình sẽ xuất hiện dòng chữ Hello World!



2.2.5 Một số kỹ thuật

2.2.5.1 Hiển thị thông tin ra màn hình

Để in thông tin ra màn hình, ta sẽ dùng cách thức sau:

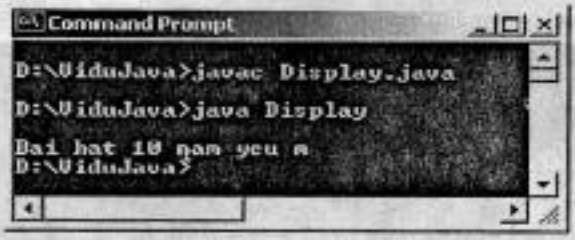
System.out.print(arg1 + arg2 + .. + argn)

Java sẽ tự động định dạng dữ liệu cho các tham số arg1, arg2, ..., argn tùy theo kiểu của chúng.

Hãy lưu chương trình sau vào tập tin Display.java:

```
Public class Display {
    Public static void main(String args[]) {
        int i =10;
        String str = "nam yeu";
        Char ch = 'm';
        System.out.print('\n' + "Bai hat" + i + str + ch);
    }
}
```

Biên dịch và thực thi ta có kết quả :



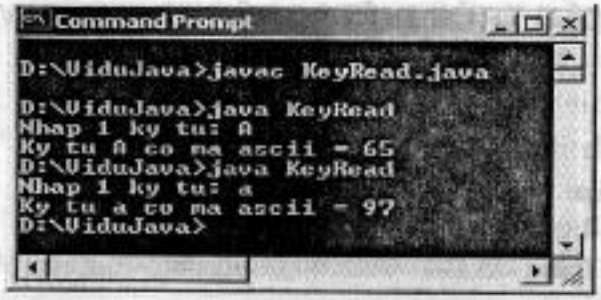
Phương thức **System.out.println(arg1 + arg2 + .. + argn)** dùng để hiển thị các tham số từ arg1 đến argn và tự động xuống dòng mới.

2.2.5.2 Đọc 1 ký tự từ bàn phím

Phương thức **int System.int.read()** trả một số nguyên là thứ tự trong bảng mã ASCII của ký tự vừa nhập từ bàn phím.

Hãy lưu chương trình sau vào tập tin KeyRead.java

```
import java.io.*;
public class KeyRead {
    public static void main(String args[]) {
        try {
            System.out.print("Nhap 1 ky  tu:");
            int ch = System.in.read();
            System.out.print("Ky tu" + (char)ch + "co ma ascii
        =" + ch);
            } catch(IOException ie) {
              System.out.print("Error" + ie);
            }
    }
}
```



Trong ví dụ trên, ta cần lưu ý một số điểm sau:

- Dòng đầu tiên **import java.io.*;** là cơ chế để khai báo với trình biên dịch các lớp thư viện của Java mà chương trình có sử dụng đến. Trong trường hợp này chương trình khai báo sử dụng tất cả các lớp trong gói (package) java.io. Thực tế chương trình trên chỉ sử dụng lớp IOException của gói java.io mà thôi, vì thế ta có thể thay thế dòng **import java.io.*;** bằng **import java.io.IOException;**
- Cơ chế ngoại lệ (Exception) của java:

```
try {
      ....
    }
catch(IOException ie) {
      ....
    }
```

sẽ được giải thích rõ ở phần sau.

2.3 CÁC CẤU TRÚC ĐIỀU KHIỂN TRONG JAVA

Java có tất cả các cấu trúc điều khiển giống hệt như cấu trúc điều khiển trong C++.

2.3.1 Lệnh if – else

Cú pháp:

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

Ví dụ:

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

2.3.2 Phép toán?

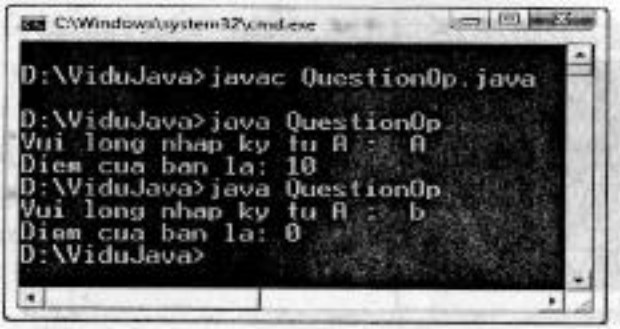
Cú pháp:

(condition) ? Operation1 : Operation2;

Nếu điều kiện condition có giá trị là true lệnh sẽ trả về giá trị của biểu thức Operation1, ngược lại sẽ trả về giá trị của biểu thức Operation2.

Ví dụ:

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



2.3.3 Lệnh switch

Cú pháp

```
switch ( variable ) {
    case value1 : {
      Task 1;
// Các tác vụ sẽ được thực thi nếu giá trị của variable là value1
      break;
    }
    case value2 : {
      Task 2;
// Các tác vụ sẽ được thực thi nếu giá trị của variable là value2
      break;
    }
    ...
    default:
      Task n;
// Tác vụ sẽ được thực thi nếu giá trị của variable không là các giá trị phía trên
}
```

Ví dụ

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

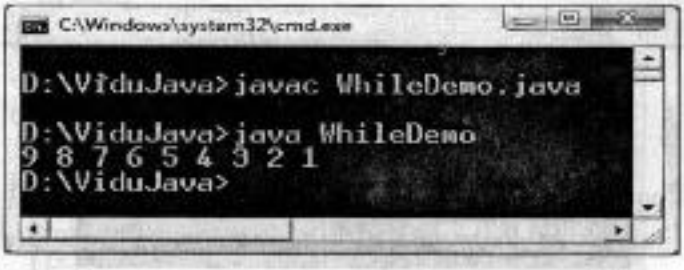
2.3.4 Lệnh while

Cú pháp

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

Ví dụ

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



2.3.5 Lệnh do - while

Cú pháp

do {

// Lặp lại các tác vụ ở đây cho đến khi điều kiện condition có giá trị là false

}

while (condition)

Ví dụ:

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

2.3.6 Lệnh for

Cú pháp

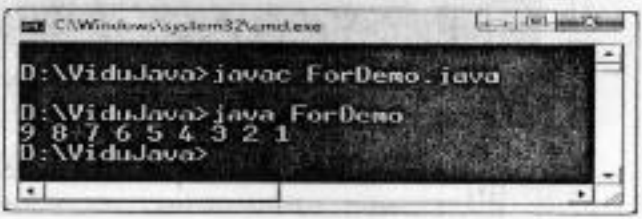
```
for (operation1; condition; operation2){
    // Các tác vụ được lặp lại
}
```

Tương đương như cấu trúc sau:

```
operation1;
while (condition) {
    // Các tác vụ được lặp lại
    operation2;
}
```

Ví dụ

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



2.3.7 Lệnh break

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

Ví dụ

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

Chương trình trên đổi đối số thứ nhất của nó (lưu trong args[0]) thành số bằng lệnh

Integer.valueOf(args[0]).inValue() và tìm số lớn nhất

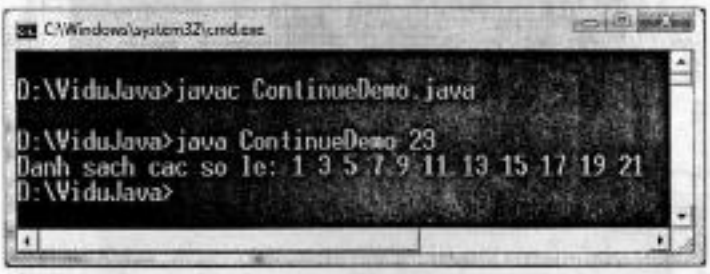
chia hết số này.

2.3.8 Lệnh continue

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

Ví dụ:

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



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

2.3.9 Một số kỹ thuật khác

2.3.9.1 Đọc đối số của chương trình

Khi thực thi chương trình, ta có thể nhập vào các đối số từ dòng lệnh theo cú pháp sau:

java ClassName arg1 arg2 arg3 argn

Các đối số cách nhau khoảng trắng. Để đón nhận các đối số này, phương thức main bắt buộc phải khai báo một tham số có kiểu mảng các chuỗi.

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

Các đối số lần lượt được đặt vào các phần tử của mảng này. Số lượng đối số có thể xác định được bằng cách truy xuất thuộc tính **args.length** của mảng.

Ví dụ

```
public class PrintArgs {
public static void main(String args[]) {
    for (int i = 0; i < args.length; i ++) {
        if (i % 2 = = 0) continue;
            System.out.println(args[i]);
        }
    }
}
```

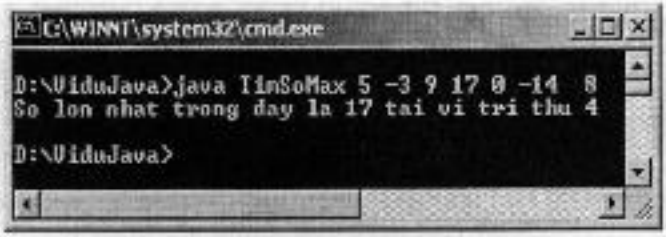
2.3.9.2 Đổi chuỗi thành số

Khi thực thi chương trình, đôi khi ta cần chuyển đổi các kiểu dữ liệu với nhau, sau đây là một số cách có thể dùng để chuyển đổi các chuỗi (chẳng hạn chuỗi str) thành số:

```
int    i = Integer.valueOf(str).intValue();
long   l = Long.valueOf(str).longValue();
float  f = Float.valueOf(str).floatValue();
```

Ví dụ

```
public class TimSoMax {
    public static void main(String args[]) {
        int max = Integer.valueOf(args[0].intValue();
        int vitri = 0;
        for (int i = 1; i < args.length; i ++) {
            int x = Integer. valueOf(args[i].intValue();
            if(max <x)
            {max = x; vitri =i;}
        }
        System.out.print("So lon nhat trong day la" + max);
        System.out.println("Tai vi tri thu" + (vitri + 1));
    }
}
```



2.3.9.3 Đổi số thành chuỗi

Có nhiều cách để đổi số thành chuỗi. Cách đơn giản nhất là lấy 1 chuỗi rỗng ("") cộng với số đó để được chuỗi kết quả.

Ví dụ

```
int x = 15; float y = 3.14;
String str1 = "" + x;
String str2 = "" + y;
```

2.3.9.4 Đổi chuỗi thành mảng các byte

Để đổi 1 chuỗi thành mảng byte, ta sẽ dùng hàm getBytes() trên chuỗi đó.

Ví dụ

```
public class ChuoiVaMang {
    public static void main(String args[]) {
        String str1 = "Day la chuoi can doi";
        byte b[] = str1.getBytes();
        System.out.print("Cac ky tu gom:");
        For(int i=0; i<b.length; i++)
            System.out.print((char)b[i]+ " ");
    }
}
```

2.3.9.5 Đổi mảng các byte thành chuỗi

Để tạo ra 1 chuỗi từ mảng các byte, ta sẽ dùng 2 phương thức sau:

- String(byte[] b): tạo chuỗi từ tất cả các phần tử trong mảng b[]
- String(byte[] b, int offset, int len): Tạo chuỗi từ các phần tử trong mảng b[], bắt đầu tại vị trí offset và chiều dài len.
- Ví dụ

```
public class ChuoiVaMang1 {
    public static void main(String args[]) {
        byte b[] = new byte[50]
        int i=0;
        For(byte x = 'A'; x <= 'a'; x++)
            B[i++]=x;
        String str = new String(b,0,i);
        System.out.println(str + " – Chieu dai =" + str.length());
    }
}
```

2.4 NGOẠI LỆ (EXCEPTION)

Trong chương trình, có một số các "thao tác không chắc chắn", chẳng hạn như các thao tác vào/ra: đĩa mềm chưa sẵn sàng, máy in có lỗi, mở 1 file không tồn tại, nối kết mạng không thực hiện được ... sẽ dẫn đến lỗi thực thi chương trình. Các lỗi này sẽ làm kết thúc chương trình ngay tại vị trí đó và đưa ra câu thông báo lỗi của hệ thống. Điều đó, đôi khi gây ra phiền toái cho người sử dụng và thậm chí mất mát dữ liệu đã tính toán được trước đó.

Java gọi các lỗi sinh ra từ "thao tác không chắc chắn" đó là Ngoại lệ (Exception).

Ngoại lệ tức là một sự kiện xảy ra ngoài dự tính của chương trình. Nếu không xử lý sẽ làm cho chương trình chuyển sang trạng thái không còn kiểm soát được. Ví dụ điều gì sẽ xảy ra nếu chương trình truy xuất đến phần tử thứ 11 của một mảng 10 phần tử ? Một số ngôn ngữ như C, C++ sẽ không báo lỗi gì cả, chương trình vẫn tiếp tục vận hành nhưng kết quả thì không thể xác định được.

Để hạn chế những lỗi như thế, Java bắt buộc các lệnh có thể dẫn đến các lỗi truy xuất (ngoại lệ) phải có các đoạn mã xử lý phòng hờ khi ngoại lệ xảy ra (nắm bắt ngoại lệ) theo cú pháp sau:

```
try {
        Các thao tác vào ra có thể sinh ra các
ngoại lệ.
 }
catch (KieuNgoaiLe_01   bien) {
        Ứng xử khi ngoại lệ KieuNgoaiLe_01
sinh ra
 }
catch (KieuNgoaiLe_02  bien) {
        Ứng xử khi ngoại lệ KieuNgoaiLe_02
sinh ra
 }
finally {
    Công việc luôn luôn được thực hiện
 }
```

Trong cơ chế này, các lệnh có thể tạo ra ngoại lệ sẽ được đưa vào trong khối bao bọc bởi từ khóa **try** {}. Tiếp theo đó là một loạt các khối **catch**{}. Một lệnh có thể sinh ra một hoặc nhiều loại ngoại lệ. Ứng với mỗi loại ngoại lệ sẽ có một khối **catch**{} để xử lý cho loại ngoại lệ đó. Tham số của catch

chỉ ra loại ngoại lệ mà nó có trách nhiệm xử lý. Khi thực thi chương trình, nếu một lệnh nào đó nằm trong khối **try{}** tạo ra ngoại lệ, điều khiển sẽ được chuyển sang các lệnh nằm trong các khối **catch{}** tương ứng với loại ngoại lệ đó. Các lệnh phía sau lệnh tạo ra ngoại lệ trong khối **try{}** sẽ bị bỏ qua. Các lệnh nằm trong khối **finally{}** thì luôn luôn được thực hiện cho dù có xảy ra ngoại lệ haykhông. Khối lệnh **finally{}** là tùy chọn nên có những trường hợp có thể không cần.

Ngoại lệ có loại bắt buộc phải xử lý, tức phải có **try{}**, có **catch{}** khi sử dụng lệnh đó. Ví dụ như lệnh đọc từ bàn phím. Trình biên dịch của Java sẽ báo lỗi nếu chúng ta không xử lý chúng.

Ngược lại, có loại ngoại lệ không bắt buộc phải xử lý, chẳng hạn như truy xuất đến phần tử bên ngoài chỉ số mảng, đổi chuỗi thành số, ...

Thông thường, mỗi 1 gói (package) trong Java đều khai báo 1 số các ngoại lệ sẽ sử dụng trong gói đó. Ta có thể tra cứu tài liệu đặc tả các API của Java để biết được các ngoại lệ nào có thể được tạo ra từ một phương thức cụ thể. Tài liệu này được đóng gói vào file document đi chung của bộ JDK và có thể được download tại website java.sun.com.

Ví dụ:

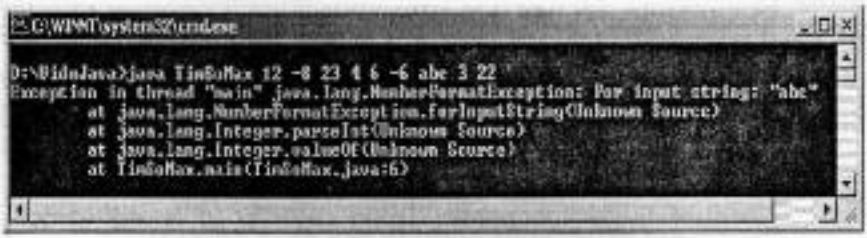
Trở lại với ví dụ tìm số lớn nhất trong dãy dối số nhập

vào chương trình.:

```
public class TimSoMax {
    public static void main(String args[]) {
        int max = Integer.valueOf(args[0].intValue();
        int vitri = 0;
        for (int i = 1; i < args.length; i ++) {
            int x = Integer. valueOf(args[i].intValue();
            if(max <x)
            {max = x; vitri =i;}
        }
        System.out.print("So lon nhat trong day la" + max);
        System.out.println("Tai vi tri thu" + (vitri + 1));
    }
}
```

Chương trình sẽ đưa ra thông báo lỗi (Exception) nếu ta không nhập vào đối số dòng lệnh giá trị nào.

Chương trình cũng sẽ đưa ra thông báo lỗi nếu ta nhập vào giá trị của đối số không phải là số nguyên.



Để khắc phục việc này, ta sẽ dùng cách bắt ngoại lệ (Exception) như sau:

```
public class TimSoMax {
    public static void main(String args[]) {
        int vitri = -1; max=Integer.MIN_VALUE;
        try {
            max = Integer. valueOf(args[0].intValue();
            vitri = 0;
            }
            catch(ArrayIndexOutOfBoundsException e)
            {System.out.println("Chua nhap tham so"); return;}
             catch(NumberFormatException e)
                 {System.out.println("Sai dinh dang tai vi tri so thu 1");}
             for(int i=1; i<args.length; i++) {
                 try {
                     int x = Integer.valueOf(args[i].intValue();
                 if(vitri ==-1) // chua gan duoc so max ban dau
                     { max = x, vitri = i; }
                 else
                     if(max<x)   // neu van co 1 so > max
                         { max = x; vitri =i; }
                 }
                 catch(NumberFormatException e)
                  { System.out.println("Sai dinh dang tai vi tri so thu" + (i
                  + 1)); }
            . } // for
```

```
            if( vitri = -1) {
                { System.out.print ("So lon nhat trong day la" + max);
                { System.out.println ("Tai vi tri thu" + (vitri + 1));
            }
            Else
                { System.out.println("Trong day doi so khong co so
nguyen");
        }
}
```

2.5 MỘT SỐ VẤN ĐỀ LIÊN QUAN ĐẾN LỚP TRONG JAVA

2.5.1 Định nghĩa lớp mới

Ngoài các lớp được định nghĩa sẵn trong thư viện chuẩn của java, các lập trình viên có thể định nghĩa thêm các lớp của mình theo cú pháp sau:

```
class ClassName {
// Danh sách các thuộc tính thuộc lớp
    DataType01 attribute1, attribute2, . .;
    DataType02 attribute3, attribute4, . .;
// Danh sách các phương thức thuộc lớp
    ClassName([DataType parameter, DataType
parameter]) {
        //    Constructor
        ...
    }
    void method01() {
        . . .
    }
    DataType method02( . . .) {
        ...
        return xx;
    }
}
```

ClassName là tên lớp mới đang được định nghĩa.

Tạo đối tượng tên obj thuộc lớp ClassName.

```
ClassName  obj = new ClassName();
```

Ví dụ: Định nghĩa một lớp có:

- Tên là Person
- Hai thuộc tính là name và address
- Phương thức khởi tạo có hai tham số để gán giá trị ban đầu cho hai thuộc tính.

- Phương thức void display() cho biết người đó tên là gì, địa chỉ ở đâu.
- Phương thức main() tạo ra một đối tượng tên là tom thuộc lớp Person

```
public class Person {
    String name;      // Thuoc tinh
    String address;   // Thuoc tinh
    Person(String n, String address) {   // Ham xay dung
        name = n;
        this.address = address;
    }
    void display() {   // Hien thi ten va dia chi ra man hinh
        System.out.print(name + "is at" + address);
    }
    public static void main(String args[]) {
        Person tom = new Person("Tom", "Disney Land");
        tom.display();
    }
}
```

2.5.2 Phạm vi nhìn thấy của một lớp

Một lớp được định nghĩa và cài đặt bên trong một tập tin. Một tập tin có thể chứa một hoặc nhiều lớp. Trong một

tập tin, chỉ có một lớp được khai báo là **public** (phía trước từ khóa class), các lớp còn lại mặc nhiên là package-private. Một lớp được khai báo là public sẽ được nhìn thấy bởi các lớp khác ở bất kỳ ở đâu. Ngược lại, các lớp package-private chỉ được nhìn thấy bởi các lớp nằm cùng tập tin hoặc trong cùng package (có thể hiểu như cùng thư mục) với nó mà thôi.

Ví dụ: Trong ví dụ này, chúng ta tách phương thức main() ra khỏi lớp Person và đưa nó vào lớp mới MultiClass. Lưu hai lớp này vào trong cùng một tập tin tên là MultiClass.java, với lớp MultiClass được khai báo là public, lớp Person không được khai báo public.

```
public class MultiClass {
   public static void main(String args[])
      Person tom = new Person("Tom", "Disney Land");
      tom.display();
      }
}
Class Person{
      String name;
      String address;

      Person(String n, String address) {
         name = n;
         this.address = address;
      }
      void display() {
         System.out.println(name + "is at" + address);
      }
}
```

2.5.3 Tính thừa kế

Thừa kế trong Java có ý nghĩa như thừa kế trong C++, tuy nhiên chúng ta lưu ý một số vấn đề trong thừa kế của Java như sau:

- Một lớp chỉ có thể có một lớp cha (thừa kế đơn).
- Lớp cha được tham khảo từ lớp con bởi từ khóa **super**.
- Dùng từ khóa **extends** để khai báo thừa kế.

Cú pháp:

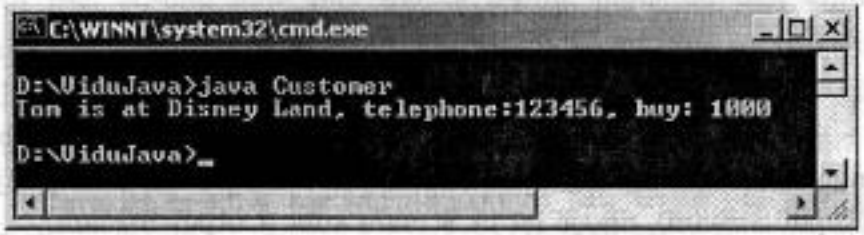
```
class A extends B { // Khai báo A thừa kế
từ B
   ...
}
```

Ví dụ: Định nghĩa lớp Customer có các đặc điểm sau:

- Thừa kế từ lớp Person (phía trước).
- Có thêm thuộc tính: telephone và buy (lượng hàng mua).
- Có phương thức khởi tạo.

- Định nghĩa lại phương thức void display() của lớp cha.

```
public class Customer extends Person {
   int telephone;
   long buy;
   public Customer(String n, String a, int t, long b) {
      super(n,a);
      telephone=t;
      buy = b;
   }
   public void display() {
      super.display();
      System.out.println(",telephone:" + telephone+",buy:" + buy);
   }
   Public static void main(string args[]) {
      Customer tom = new Customer("Tom","Disley
Land",123456,1000);
      tom.display();
   }
}
```



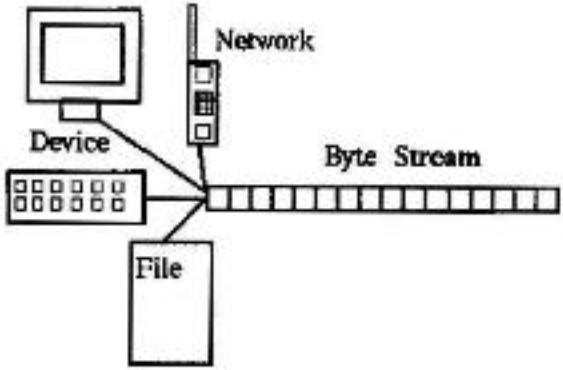
Lưu ý:

Java chỉ cho phép 1 lớp thừa kế từ 1 lớp nhưng cho

phép implements (một dạng tương tự như thừa kế) từ nhiều interface (lớp ảo – abstract class).

2.6 NHẬP/XUẤT VỚI STREAM

Stream là một dòng liên tục, có thứ tự các bytes dữ liệu chảy giữa chương trình và các thiết bị ngoại vi. Nó là khái niệm trừu tượng giúp giảm bớt các thao tác vào ra phức tạp đối với người lập trình. Nó cho phép nối kết nhiều loại thiết bị ngoại vi khác nhau với chương trình.



Hình 2.3 – Nhập xuất với stream

Nếu dòng dữ liệu trong Stream có hướng chảy từ thiết bị ngoại vi vào chương trình thì ta nói đây là Stream nhập (Input Stream), ngược lại là Stream xuất (Output Stream).

Đối với Java, các thiết bị chỉ nhập, như bàn phím, sẽ có các Stream nhập nối với nó, các thiết bị chỉ xuất, như màn

hình, sẽ có các stream xuất nối với nó , các thiết bị vừa xuất, vừa nhập, như đĩa từ, thì có cả stream nhập và xuất nối với nó.

Để giao tiếp với các thiết bị ngoại vi, chương trình trước tiên phải lấy được các stream nhập / xuất gắn với thiết bị ngoại vi này. Sau đó, chương trình có thể gởi dữ liệu ra ngoại vi bằng thao tác ghi (write) vào Stream xuất của ngoại vi. Ngược lại, chương trình có thể nhận dữ liệu từ ngoại vi bằng thao tác đọc (read) stream nhập của ngoại vi đó.

Như vậy, chương trình chỉ làm việc trên các stream nhập và stream xuất, mà không quan tâm đến đặc điểm riêng biệt của thiết bị ngoại vi nối với Stream. Điều này giúp chương trình giao tiếp với hệ thống mạng cũng dễ dàng như giao tiếp với màn hình, bàn phím hay đĩa từ.

Một điểm khác cần lưu ý là stream bao gồm những byte rời rạc. Những byte này mô tả những dạng dữ liệu khác nhau. Ví dụ một số integer khi viết vào stream sẽ chuyển thành 4 byte. Vì thế cần phải có các thao tác chuyển đổi dữ liệu nhận và gởi giữa chương trình và stream.

Java hỗ trợ hai các lớp Stream cơ bản trong gói java.io là:

- java.io.InputStream: Stream nhập
- java.io.OutputStream: Stream xuất

Ngoài ra, còn có các lớp Stream thừa kế từ hai lớp trên nhằm mục đích cung cấp các tiện ích cho các loại thiết bị vào ra chuyên biệt như: FileInputStream, FileOutputStream, PipedInputStream, PipedOutputStream, . . .

2.6.1 Lớp java.io.InputStream

Là loại stream cho phép chương trình nhận dữ liệu từ ngoại vi. Có các phương thức cơ bản sau:

public int read() throws IOException

Đọc 1 byte từ Stream

- Return 0-255 : Mã ASCII của byte nhận được từ ngoại vi
- -1 : Stream đã kết thúc, không còn dữ liệu.

public int read(byte b[]) throws IOException

Đọc tất cả các byte hiện có trong Stream vào mảng byte b[].

- Return 0-255: Số lượng byte đọc được.
- -1 : Stream đã kết thúc, không còn dữ liệu.

public int read(byte b[], int offset, int len) throws IOException

Đọc len byte từ Stream hiện tại, lưu vào trong mảng b bắt đầu từ vị trí offset

- Return: số lượng byte đọc được.
- -1 : Stream đã kết thúc.

Các phương thức trên khi thực thi sẽ bị nghẽn (block) cho đến khi có dữ liệu hoặc kết trúc Stream hay một ngoại lệ xuất hiện.

public int available() throws IOException

Trả về số lượng byte hiện có trong Stream mà không làm nghẽn chương trình.

public long skip(long n) throws IOException

Bỏ qua n byte hiện có trong stream.

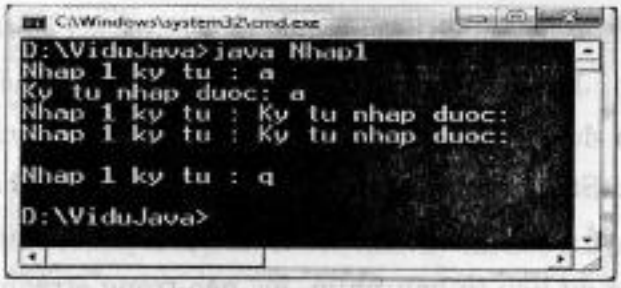
public void close() throws IOException

Đóng stream và giải phóng các tài nguyên đang liên kết với stream.

Đối với Java, **System.in** là một InputStream nối kết với bàn phím được tạo sẵn bởi hệ thống. Chương trình có thể dùng InputStream này để nhận các ký tự nhập từ bàn phím. Một vấn đề khi sử dụng đối tượng **System.in** là sau khi nhập xong dữ liệu từ bàn phím, lúc nào trong stream cũng sẽ có thêm 2 ký tự kết thúc là '\r' và '\n' (phím Enter). Do đó, để sử dụng đúng dữ liệu, ta phải loại bỏ đi 2 ký tự này.

Ví dụ 1: Nhập từng ký tự từ bàn phím và hiển thị ra màn hình

```
import java.io.*;
public class Nhap1  {
   public static void main(String args[]) {
      InputStream is = System.in;  // Ban phim
      while(true) {
      try {
         System.out.print("Nhap 1 ky tu:");
         int ch = is.read();
         if (ch == -1 || ch == 'q') break;
         System.out.println("Ky tu nhap duoc:"+ (char)ch);
      }
      Catch(IOException e)
         System.out.println("Co loi ve nhap xuat"); }
      }
   }
}
```



Trong ví dụ trên, khi ta nhập ký tự a từ bàn phím, ta sẽ nhận được 3 ký tự: a,'\r' và '\n' (có số thứ tự trong bảng mã là 97, 13 và 10). Do đó, sau khi nhận ký tự đầu tiên là a vào chương trình, trong vùng đệm của stream nhập đã có sẵn 2 ký tự '\r' và '\n' nên sẽ nhận tiếp để xử lý mà không yêu cầu

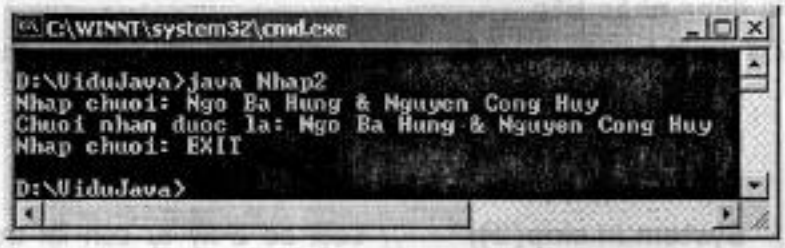
người dùng nhập tiếp.

Để xử lý loại bỏ 2 ký tự này, ta sẽ chỉnh sửa lại như sau:

```
System.out.print("Nhap 1 ky tu : ");
int ch = is.read();
if (ch==-1 || ch=='q') break;
System.in.skip(2);      // Loại bỏ 2 ký tự còn lại trong stream
System.out.println("Ky tu nhap duoc: "+ (char)ch);
```

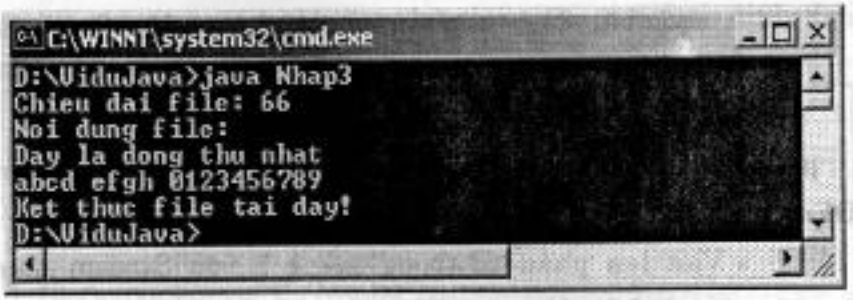
Ví dụ 2: Nhập 1 chuỗi từ bàn phím, hiển thị chuỗi vừa nhập ra màn hình.

```
import java.io.*;
public class Nhap2  {
   public static void main(String args[]) {
      while (true) {
         System.out.print("Nhap chuoi:");
         byte b[] = new byte[100];  //Tao vung dem de nhap chuoi
         try {
            int n = System.in.read(b);          // Nhap n ky tu
            String str = new String(b,0,n-2): // Doi byte[] -> String
            if (str.equals("EXIT")) break;  // Kiem tra Dk de thoat
            System.out.print("Chuoi nhan duoc la:" + str);
      }
      Catch (IOException ie) {
            System.out.print("Error:" + ie);
            }
      }
   }
}
```



Ví dụ 3: Đọc nội dung 1 file vào vùng đệm, hiển thị nội dung file đó.

```
import java.io.*;
public class Nhap3  {
    public static void main(String args[]) {
        try {
            FileInputStream          f1          =          new
FileInputStream("C:/Test.txt");
            int len = f1.available();
            System.out.println("Chieu dai file:" + len);
            System.out.println("Noi dung file:");
            For(int i = 0; i < len; i ++)
                System.out.print((char)f1.read());
            f1.close();
            }
        catch (IOException ie)
            { System.out.println("Loi khi truy xuat file"; }
    }
}
```



Do lớp FileInputStream thừa kế từ InputStream nên có thể sử dụng các phương thức read(), read(byte[]), read(byte[], int, int) tương tự như trên InputStream.

2.6.2 Lớp java.io.OutputStream

Là loại stream cho phép chương trình xuất dữ liệu ra ngoại vi. Có các phương thức cơ bản sau:

public void write(int b) throws IOException

- Viết byte b vào Stream hiện tại,
- Return : void

public void write (byte[] b) throws IOException

- Viết tất cả các phần tử của mảng b vào Stream hiện tại
- Return : void

public void write (byte[] b, int offset, int len) throws IOException

- Viết len phần tử trong mảng b vào Stream hiện

tại, bắt đầu từ phần tử có chỉ số là offset của mảng.

- Return : void

public void write (byte[] b, int offset, int len) throws IOException

- Viết len phần tử trong mảng b vào Stream hiện tại, bắt đầu từ phần tử có chỉ số là offset của mảng.
- Return : void

public void flush () throws IOException

Đẩy các byte được lưu trong vùng đệm của stream ra thẳng thiết bị ngoại vi. Phương thức này chỉ sử dụng cho 1 số các đối tượng đặc biệt thừa kế từ OutputStream (chẳng hạn PrintWriter).

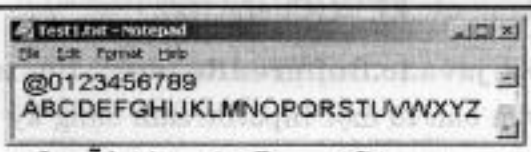
public void close () throws IOException

Đóng stream và giải phóng các tài nguyên đang liên kết với stream.

Đối với Java, **System.out** là một PrintStream nối kết với màn hình được tạo sẵn bởi hệ thống. Do PrintStream cũng thừa kế từ OutputStream nên chương trình có thể dùng các phương thức của OutputStream để gởi các ký tự ra màn hình.

Ví dụ: Minh họa ghi dữ liệu cho 1 file từ nhiều định dạng khác nhau.

```
import java.io.*;
public class Ghi {
    public static void main(String args[]) {
        try {
            FileOutputStream f1 = new
FileOutputStream("C:/Test.txt");
            int ch = '@';f1.write(ch);
            byte b1[] = new byte[10];
            int m=0;
            for(int i = '0'; i <= '9'; i ++)    b1[m++]=(byte)i;
            f1.write(b1); m=0; f1.write('\r'); f1.write('\n');
            byte b2[] = new byte[50];
            for(int j = 'A'; j <= 'Z'; j ++)     b2[m++] = (byte)j;
            f1,write(b2,0,m);
            f1.close();
            }
      catch (IOException ie)
            { System.out.println("Loi khi truy xuat file"; }
    }
}
```



2.6.3 Nhập chuỗi từ một InputStream

InputStream là Stream nhập gồm chuỗi các bytes. Nó chỉ cung cấp các phương thức cho việc đọc byte và mảng các bytes. Do đó, nếu muốn nhập 1 chuỗi (String) ta phải nhập

các ký tự đó vào mảng các byte[], sau đó dùng các cách chuyển đổi kiểu để đổi từ mảng byte[] sang chuỗi (String). Lưu ý rằng khi nhập chuỗi từ bàn phím thì phải bỏ đi 2 ký tự kết thúc là '\r' và '\n' cuối cùng.

Ví dụ: cách nhập 1 chuỗi từ InputStream.

```
byte b[] = new byte[100]; // Tạo vùng đệm cho mảng, dự kiến là không quá 100 ký tự
int n = System.in.read(b); // Nhập dữ liệu từ bàn phím và lưu vào mảng b[]
String str = new String(b, 0, n-2); // Chuyển đổi từ mảng b[] sang chuỗi
```

Để có thể đọc được 1 chuỗi dễ dàng hơn ta có thể sử dụng thêm các lớp sau:

- Lớp **java.io.InputStreamReader**: Là cầu nối để chuyển InputStream dạng byte sang InputStream dạng các ký tự (Character).
- Lớp **java.io.BufferedReader**: Hỗ trợ việc đọc văn bản từ một InputStream dạng ký tự.

Phương thức **String readLine() throws IOException** của **BufferedReader** cho phép đọc dòng văn bản kế tiếp trong InputStream. Một dòng kết thúc bởi cặp ký tự '\r"\n' hoặc kết thúc Stream.

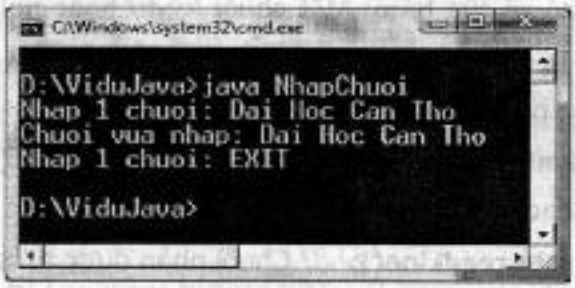
Kết quả trả về của hàm: Một chuỗi ký tự hoặc null.

Giả sử is là một đối tượng thuộc lớp InputStream. Để đọc chuỗi từ is ta thực hiện các thao tác sau:

```
InputStreamReader isr = new InputStreamReader(is);
BufferedReader br = new BufferedReader (isr);
String str = br.readLine();  // Chuỗi nhận được từ is
```

Ví dụ: Chương trình nhập 1 chuỗi từ bàn phím, hiển thị ra màn hình chuỗi đó.

```
import java.io.*;
public class NhapChuoi {
    public static void main(String args[]) {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        while (true) {
            try {
                System.out.print("Nhap 1 chuoi:");
                String line = br.readline();
                If (line.equals("EXIT")) break;
                System.out.println("Chuoi vua nhap:" + line);
            } catch (IOException ie) {
                System.out.print("Error:" + ie);
            }
        }
    }
}
```



2.6.4 Xuất chuỗi ra một OutputStream

OutputStream là Stream xuất gồm chuỗi các bytes. Nó chỉ cung cấp các phương thức cho việc viết byte và mảng các bytes. Nếu muốn ghi 1 chuỗi, ta phải chuyển chuỗi đó thành mảng byte[] (thông qua phương thức getBytes()), sau đó gọi các phương thức write(), write(byte[]) và write(byte[], int, int) của OutputStream.

Ví dụ: Đây là đoạn chương trình ghi 1 chuỗi str vào 1 file (OutputStream)

```
FileOutputStream f2 = new FileOutputStream("C:/Test2.txt");
String str = "Day la chuoi thu 1 can ghi vao file \r\n";
byte b[] = str.getBytes();  // Doi chuoi thanh mang byte[]
f2.write(b);
f2.close();
```

Để có thể dễ dàng hơn trong việc gửi 1 chuỗi (String) ra một OutputStream ta nên sử dụng lớp **java.io.PrintWriter**. Lớp PrintWriter có định nghĩa thêm các hàm như print(...) hay println(...) cho phép ghi nhiều loại dữ liệu khác nhau như số nguyên, số thực, chuỗi, ... ra các thiết bị ngoại vi. Lưu ý rằng, sau khi ghi vào stream khi dùng PrintWriter ta phải gọi thêm lệnh flush() để đẩy dữ liệu từ vùng đệm của OutputStream vào thiết bị ngoại vi đó.

Giả sử: **os** là một OutputStream, **str** là chuỗi cần viết vào **os**.

Ta thực hiện các thao tác sau:

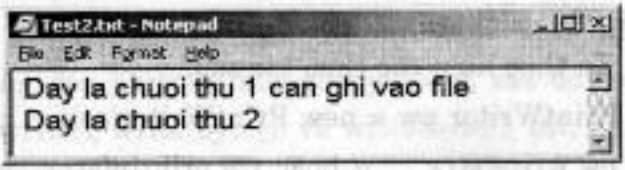
```
PrintWriter pw = new PrintWriter(os);
pw.write(str);    // hoặc pw.println(str); nếu muốn có ký tự xuống dòng
pw.flush()        // Đẩy dữ liệu từ buffer ra ngoại vi
```

Ví dụ:

```
import java.io.*;
public class Ghi1 {
    public static void main(String args[]) {
        try {
            FileOutputStream f1 = new FileOutputStream("C:/Test.txt");
```

```
            PrintWrite pw = new PrintWrite(f2);
            String str1 = "Day la chuoi thu 1 can ghi vao file";
            String str2 = "Day la chuoi thu 2";
            pw.println(str1);  // Ghi chuoi str1, sau do xuong dong
            pw.print(str2);    // Ghi chuoi str2 khong xuong dong
            pw.flush();            // Day du lieu tu Stream vao ngoai vi
            f2.close();
            }
        catch (IOException ie)
            { System.out.println("Loi khi truy xuat file"; }
    }
}
```

File kết quả:

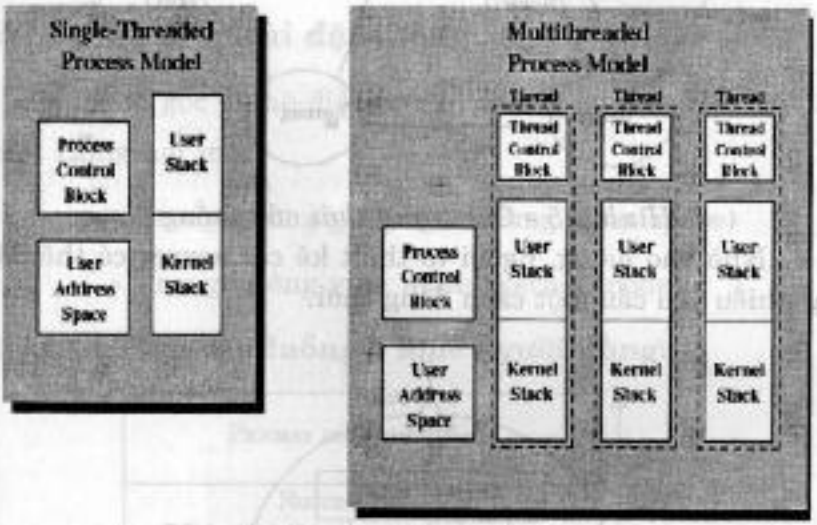


2.7 LUỒNG (THREAD)

Luồng là một cách thông dụng để nâng cao năng lực xử lý của các ứng dụng nhờ vào cơ chế song song. Trong một hệ điều hành cổ điển, đơn vị cơ bản sử dụng CPU là một quá trình. Mỗi quá trình có một Thanh ghi bộ đếm chương trình (PC-Program Counter), Thanh ghi trạng thái (Status Register), ngăn xếp (Stack) và không gian địa chỉ riêng (Address Space).

Ngược lại, trong một hệ điều hành có hỗ trợ tiện ích

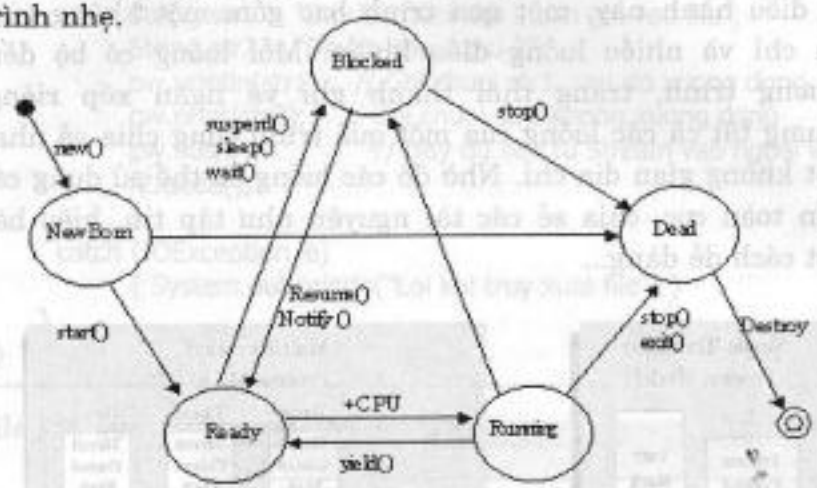
luồng, đơn vị cơ bản sử dụng CPU là một luồng. Trong những hệ điều hành này, một quá trình bao gồm một không gian địa chỉ và nhiều luồng điều khiển. Mỗi luồng có bộ đếm chương trình, trạng thái thanh ghi và ngăn xếp riêng. Nhưng tất cả các luồng của một quá trình cùng chia sẻ nhau một không gian địa chỉ. Nhờ đó các luồng có thể sử dụng các biến toàn cục, chia sẻ các tài nguyên như tập tin, hiệu báo một cách dễ dàng...



Hình 2.4 – Quá trình (Process) và luồng (Thread)

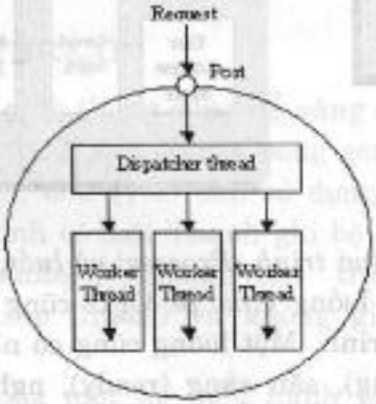
Cách thức các luồng chia sẻ CPU cũng giống như cách thức của các quá trình. Một luồng cũng có những trạng thái: đang chạy (running), sẵn sàng (ready), nghẽn (blocked) và

kết thúc (Dead). Một luồng thì được xem như là một quá trình nhẹ.



Hình 2.5 - Các trạng thái của luồng

Nhờ vào luồng, người ta thiết kế các server có thể đáp ứng nhiều yêu cầu một cách đồng thời.



Hình 2.6 - Sử dụng luồng cho các server

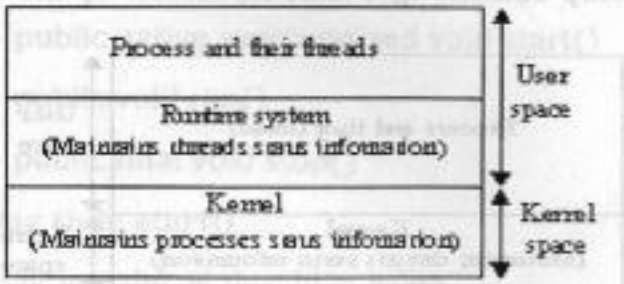
Trong mô hình này, Server có một **Luồng phân phát** (Dispatcher thread) và nhiều **Luồng thực hiện** (Worker thread). Luồng phân phát tiếp nhận các yêu cầu nối kết từ các Client, rồi chuyển chúng đến các luồng thực hiện còn rảnh để xử lý. Những luồng thực hiện hoạt động song song nhau và song song với cả luồng phân phát, nhờ đó, Server có thể phục vụ nhiều Client một cách đồng thời.

2.7.1 Các mức cài đặt luồng

Nhìn từ góc độ hệ điều hành , Luồng có thể được cài đặt ở một trong hai mức:

- Trong không gian người dùng (user space)
- Trong không gian nhân (kernel mode):

2.7.1.1 Tiếp cận luồng ở mức người dùng:

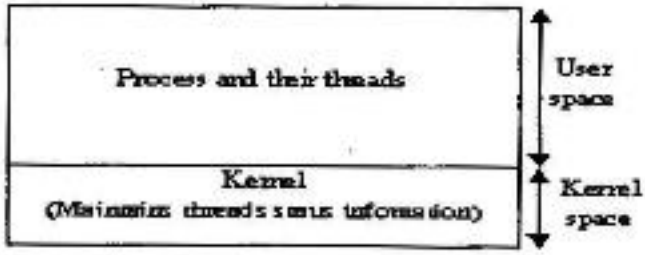


Hình 2.7 - Kiến trúc luồng cài đặt ở mức người dùng

Không gian người dùng bao gồm một hệ thống runtime mà nó tập hợp những thủ tục quản lý luồng. Các luồng chạy trong không gian nằm bên trên hệ thống runtime thì được quản lý bởi hệ thống này. Hệ thống runtime cũng lưu giữ một bảng tin trạng thái để theo dõi trạng thái hiện hành của mỗi luồng. Tương ứng với mỗi luồng sẽ có một mục từ trong bảng, bao gồm các thông tin về trạng thái, giá trị thanh ghi, độ ưu tiên và các thông tin khác về luồng.

Tiếp cận này có hai mức định thời biểu (scheduling): bộ định thời biểu cho các quá trình nặng và bộ định thời biểu trong hệ thống runtime. Bộ lập biểu của hệ thống runtime chia thời gian sử dụng CPU được cấp cho một quá trình thành những khoảng nhỏ hơn để cấp cho các luồng trong quá trình đó. Như vậy, việc kết thúc một luồng thì vượt ra ngoài tầm kiểm soát của kernel hệ thống.

2.7.1.2 Tiếp cận luồng ở mức hạt nhân hệ điều hành



Hình 2.8 - Kiến trúc luồng cài đặt ở mức hệ thống

Trong tiếp cận này không có hệ thống runtime và các luồng thì được quản lý bởi kernel của hệ điều hành. Vì vậy, bảng thông tin trạng thái của tất cả các luồng thì được lưu trữ bởi kernel. Tất cả những lời gọi mà nó làm nghẽn luồng sẽ được bẫy (TRAP) đến kernel. Khi một luồng bị nghẽn, kernel chọn luồng khác cho thực thi. Luồng được chọn có thể cùng một quá trình với luồng bị nghẽn hoặc thuộc một quá trình khác. Vì vậy, sự tồn tại của một luồng thì được biết bởi kernel và chỉ có một mức lập biểu trong hệ thống.

2.7.2 Luồng trong java

Trong Java, luồng là 1 đối tượng thuộc lớp **java.lang.Thread**. Một chương trình trong Java có thể cài đặt luồng bằng cách tạo ra một lớp con của lớp java.lang.Thread.

Lớp này có 3 phương thức cơ bản để điều khiển luồng là:

- **public native synchronized void start()**
- **public void run()**
- **public final void stop()**

Phương thức start()

Chuẩn bị mọi thứ để thực hiện luồng.

Phương thức run()

Thực hiện công việc thực sự của luồng, () sẽ được kích hoạt một cách tự động bởi phương thức start().

Phương thức stop()

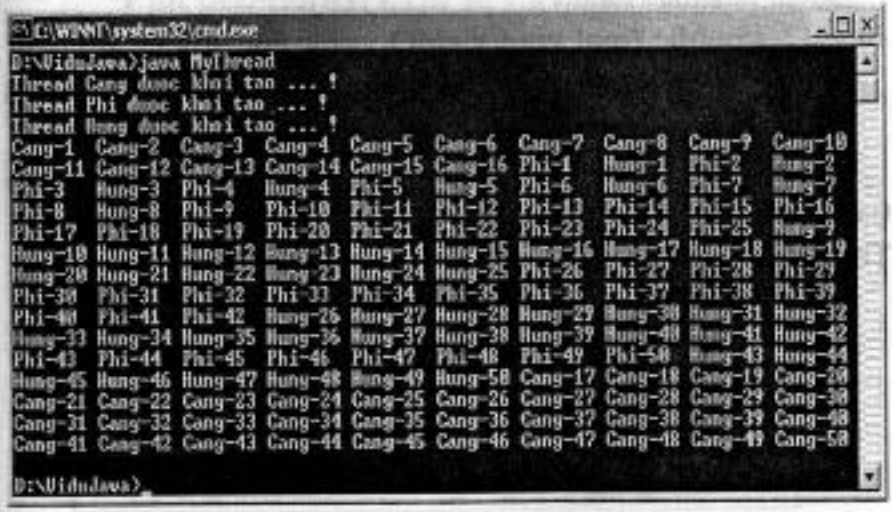
Kết thúc luồng.

Luồng sẽ "chết" khi tất cả các công việc trong phương thức run() được thực hiện hoặc khi phương thức stop() được kích hoạt.

Ví dụ: Định nghĩa lớp MyThread là một Thread có:

- Thuộc tính:
  - name: tên của thread
- Các phương thức:
  - MyThread(String ten):
    Là phương thức khởi tạo, có nhiệm vụ gán giá trị cho thuộc tính và gọi phương thức start() để cho thread hoạt động (start() tự động gọi run()).
  - void run()
    In 50 lần dòng thông báo ra màn hình rồi kết thúc thread.
  - void main()
    Tạo ra 3 thread thuộc lớp MyThread lần lượt có tên là Cang, Phi, Hung. Cho 3 thread đó thực thi song song.

```
public class MyThread extends Thread {
    String name;
    public MyThread(String ten) {
        name = ten;
        System.out.println("Thread" + name + "duoc khoi tao ... !");
    }
    public void run() {
        for(int i = 1; i <=50; i ++ );
            System.out.print(name + "-" + i + "\t");
    }
    public static void main(String args[]) {
            MyThread t1 = new MyThread("Cang");
            MyThread t2 = new MyThread("Phi");
            MyThread t3 = new MyThread("Hung");
            t1.start(); t2.start(); t3.start();
        }
}
```



Ta nhận thấy, 3 luồng dùng để in dãy số được thực thi xen kẻ và song song với nhau. Thread khởi tạo đầu tiên lại kết thúc sau cùng. Việc định thời biểu thực thi (scheduling) là do hệ điều hành quyết định.

Lớp Thread còn có 1 số các phương thức khác :

- **public static void sleep(long m) throws InterruptedException:** làm cho Thread bị nghẽn (Blocked) một khoảng thời gian m miligiây xác định.
- **public final void suspend():**Chuyển Thread từ trạng thái sẵn sàng (Ready) sang trạng thái nghẽn (Blocked).
- **public final void resume():**Chuyển Thread từ trạng thái nghẽn (Blocked) sang trạng thái sẵn sàng (Ready).
- **public final void yield()** : Chuyển Thread từ trạng thái đang chạy (Running) sang trạng thái sẵn sàng (Ready).

Ngoài ra, còn có một số hàm của lớp khác (java.lang.Object) có thể dùng điều khiển trạng thái của Thread là:

- **public void walt(long m) throws InterruptedException**: dừng Thread trong 1 khoảng thời gian mili giây xác định.
- **public final void notify():** đánh thức 1 thread trong hàng đợi.

- public void void notifyAll(): đánh thức tất cả các thread đang có trong hàng đợi.
- Độ ưu tiên của luồng

Độ ưu tiên của các luồng xác định mức độ ưu tiên trong việc phân phối CPU giữa các luồng với nhau. Khi có nhiều luồng đang ở trạng thái "Ready", luồng có độ ưu tiên cao nhất sẽ được thực thi (chuyển đến trạng thái "running").

Độ ưu tiên trong Java được định nghĩa bằng các hằng số nguyên theo thứ tự giảm dần như sau:

- Thread.MAX_PRIORITY (có giá trị là 10)
- Thread.NORM_PRIORITY (có giá trị là 5)
- Thread.MIN_PRIORITY (có giá trị là 1)

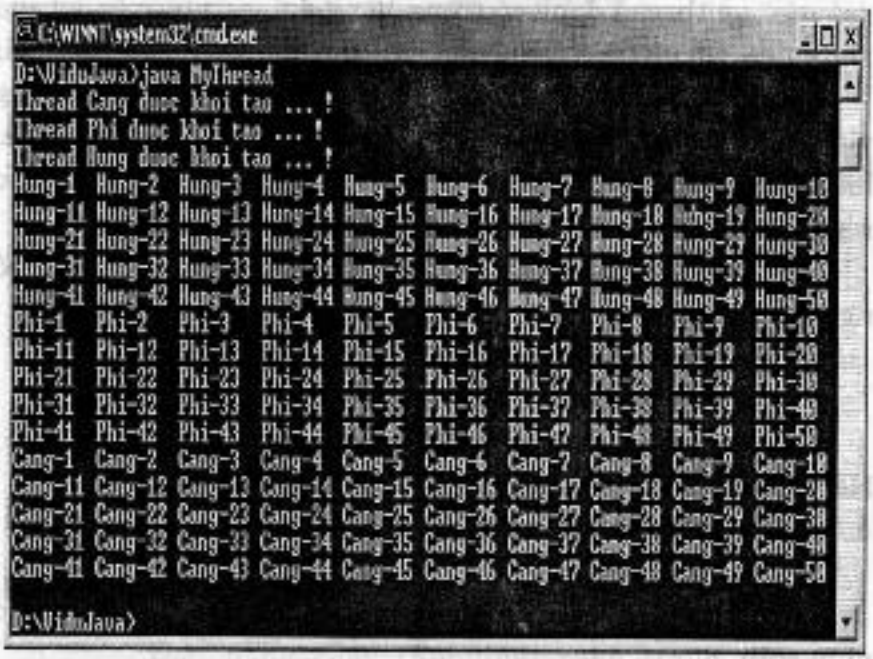
Hai phương thức liên quan đến độ ưu tiên của luồng là:

- **setPriority(int x)**: Đặt độ ưu tiên của luồng là x
- **int getPriority()**:Trả về giá trị ưu tiên của luồng

Ví dụ: Chỉnh sửa lại hàm main() trong lớp MyThread phía trên.

```
public static void main(String args[]){
        MyThread t1 = new MyThread("Cang");
        MyThread t2 = new MyThread("Phi");
        MyThread t3 = new MyThread("Hung");
        t1.setPriority(Thread.MIN_PRIORITY);
        t2.setPriority(Thread.NORM_ PRIORITY);
        t3.setPriority(Thread.MAX_ PRIORITY);
        t1.start(); t2.start(); t3.start();
    }
}
```

Kết quả đạt được như sau:



Thread thứ 3 (tên Hung) có độ ưu tiên cao nhất nên được ưu tiên thực thi trước, thread thứ 1 (tên Cang) có độ ưu tiên thấp nhất nên kết thúc sau cùng.

2.7.3 Đồng bộ hóa giữa các luồng

Tất cả các luồng của một quá trình thì được thực thi song song và độc lập nhau nhưng lại cùng chia sẻ nhau một không gian địa chỉ của quá trình. Chính vì vậy có thể dẫn đến khả năng đụng độ trong việc cập nhật các dữ liệu dùng chung của

chương trình (biến, các tập tin được mở) khi một luồng ghi lên một dữ liệu trong khi một luồng khác đang đọc dữ liệu này.

Trong trường hợp đó, cần phải sử dụng cơ chế đồng bộ hóa của Java. Có nhiều mức đồng bộ hóa như: trên một biến, trên một câu lệnh, trên một khối lệnh hay trên một phương thức.

Cách thức dùng để đồng bộ hóa thông dụng nhất trong Java là khai báo thuộc tính **synchronized** cho phương thức đó. Trong 1 thời điểm, chỉ có 1 Thread thực thi được phương thức có thuộc tính là **synchronized**.

Ví dụ:

```
class TaiKhoan {
    private String sotaikhoan;
    private double sodu;   // so du trong tai khoan
    public TaiKhoan(String sotk)
    { sotaikhoan = sotk; }
    public synchronized void GanSoDu(double s) {
        sodu = s;
    }
    public synchronized double LaySoDu() {
        return sodu;
    }
```

```
    synchronized boolean CapNhatSoDu(double sotien){
        if(LaySoDu() + sotien >=0)
          { GanSoDu(LaySoDu() + sotien); return true; }
        else
          return false;
    }
}
```

Các phương thức GanSoDu(), CapNhatSoDu(), ... cần phải có thuộc tính là **synchronized** để trong 1 thời điểm chỉ có 1 Thread gọi phương thức đó, tránh không đồng nhất về giá trị tài khoản.

2.8 BÀI TẬP ÁP DỤNG

Chủ đề 1: Cơ bản về Java

- **Mục đích:**
  - Sinh viên làm quen với ngôn ngữ Java, viết 1 số chương trình đơn giản bằng Java.
  - Thực tập cách nhập / xuất thông tin qua Java.
  - Thiết kế lớp đơn giản qua Java.
- **Yêu cầu**

Sinh viên thực hiện các bài tập sau

- **Bài 1** : Cài đặt bộ JDK trên hệ thống máy tính đang thực tập. Đặt các biến môi trường PATH và CLASSPATH đến các vị trí thích hợp.
- **Bài 2** : Viết chương trình hiển thị ra màn hình câu " Hello Java"
- **Bài 3** : Viết chương trình nhập vào 1 chuỗi ký tự. Đổi thành chữ Hoa và in ra màn hình.
- **Bài 4** : Viết chương trình nhập vào 1 số nguyên. Kiểm tra xem số đó có phải là số nguyên tố hay không và thông báo ra màn hình.
- **Bài 5** : Viết chương trình giải phương trình bậc 2.
- **Bài 6** : Viết chương trình tính tổng của dãy số từ 1 đến n (Với n được nhập từ bàn phím).
- **Bài 7** : Nhập vào 1 dãy số thực, tính tổng của các số dương trong dãy đó.

Chủ đề 2: Thiết kế lớp trong Java

- **Mục đích**:
  - Thiết kế lớp dưới Java.
- **Yêu cầu**

Sinh viên thực hiện các bài tập sau

- **Bài 1** : Thiết kế lớp Diem (Điểm trong không gian 2 chiều) gồm :
  - Thành phần dữ liệu : x,y kiểu int.
  - Các hàm thành viên gồm : các phương thức khởi tạo, phương thức gán tọa độ cho 1 điểm, phương thức nhập tọa độ cho 1 điểm, phương thức in ra màn hình tọa độ điểm theo dạng (x,y), phương thức tính khoảng cách từ điểm đó đến gốc tọa độ.
  - Viết hàm main() khai thác lớp vừa định nghĩa.
- **Bài 2** : Thiết kế lớp PhanSo (Phân số) có:
  - 2 thuộc tính tử số và mẫu số thuộc kiểu số nguyên
  - Các phương thức: Phương thức khởi tạo, phương thức in phân số, phương thức nghịch đảo phân số, phương thức trả về giá trị thực của phân số, hàm cộng, trừ, nhân, chia 2 phân số.
  - Phương thức main() sử dụng lớp PhanSo.

Chủ đề 3: Thread

- Mục đích:
  - Tìm hiểu về luồng (Thread), cách lập trình luồng, lập trình song song.
- Yêu cầu: Sinh viên thực bài tập sau:
  - **Bài 1** : Viết chương trình mô phỏng bài toán "Người sản xuất - Người tiêu dùng", trong đó Người sản xuất sẽ sản xuất ra một số lượng ngẫu nhiên n sản phẩm nào đó rồi yêu cầu nhập kho. Người tiêu dùng sẽ yêu cầu xuất kho một số lượng ngẫu nhiên m sản phẩm nào đó từ kho. Yêu cầu nhập kho chỉ được chấp nhận nếu số lượng hàng hóa đưa vào không vượt quá sức chứa của kho, nếu không, phải chờ cho đến khi có đủ chổ trống trong kho. Yêu cầu xuất kho chỉ được chấp nhận khi còn đủ hàng trong kho nếu không cũng phải chờ.

Gợi ý : Thiết kế các lớp sau:

- Lớp Kho: Có thuộc tính là sức chứa, phương thức khởi tạo gán giá trị cho sức chứa, các phương thức xem số lượng hàng tồn, phương thức nhập kho, phương thức xuất kho. In thông báo mỗi khi nhập kho hay xuất kho thành công

- Lớp Người Sản Xuất là một Thread: Có thuộc tính là kho để nhập hàng. Phương thức khởi tạo gán giá trị cho kho nhập hàng. Phương thức sản xuất lặp lại công việc là tạo ra n sản phẩm ngẫu nhiên và chờ để nhập vào kho.
- Lớp Người Tiêu Dùng là một Thread: Có thuộc tính là kho để xuất hàng. Phương thức khởi tạo gán giá trị cho kho để xuất hàng. Phương thức tiêu dùng lặp lại công việc là chờ để yêu cầu xuất m sản phẩm từ kho.
- Lớp Demo tạo ra một kho và 2 người sản xuất, 2 người tiêu dùng thực hiện việc nhập xuất trên cùng một kho.

Chương 3
ỐNG DẪN (PIPE)

Mục đích

Chương này nhằm giới thiệu cơ chế giao tiếp liên quá trình đầu tiên là Pipe và cách sử dụng nó trong Java để làm phương tiện giao tiếp giữa các Thread trong một chương trình.

Yêu cầu

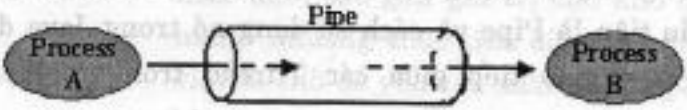
Sau khi hoàn tất chương này, bạn có thể:

- Trình bày được các đặc điểm của Pipe.
- Biết cách tạo Pipe và xuất/ nhập dữ liệu trên Pipe trong Java.
- Giải thích được chức năng của dịch vụ phản hồi thông tin (Echo Service).
- Xây dựng, biên dịch và thực thi thành công chương trình PipedEcho.

3.1 GIỚI THIỆU VỀ ỐNG DẪN

Ống dẫn là một tiện ích được hỗ trợ trong hầu hết các ngôn ngữ lập trình vận hành trên các hệ thống đa nhiệm. Ống dẫn cho phép hai quá trình nằm trên cùng một máy có thể trao đổi dữ liệu với nhau.

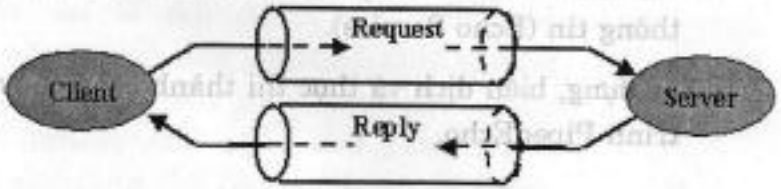
Dữ liệu đi trên ống dẫn theo một chiều nhất định. Khi sử dụng ống dẫn, người ta dùng một đầu cho việc viết dữ liệu vào và một đầu còn lại cho việc đọc dữ liệu ra.



Hình 3.1 Mô hình ống dẫn

Ống dẫn thích hợp cho trường hợp dữ liệu tạo ra của quá trình này sẽ là dữ liệu đầu vào cho quá trình kia.

Tuy nhiên ta cũng có thể sử dụng ống dẫn để xây dựng các ứng dụng theo kiến trúc Client- Server bằng cách sử dụng hai ống dẫn: một ống dẫn để truyền các yêu cầu (request), một ống dẫn để truyền các trả lời (reply).



Hình 3.2 – Dùng ống dẫn trong chương trình Client -Server

Có hai loại ống dẫn:

- Ống dẫn bình thương (Normal Pipe): Giới hạn trong phạm vi không gian địa chỉ của một quá trình mà thôi. Nó chỉ cho phép giao tiếp giữa quá trình cha với các quá trình con hay giữa các quá trình con của một quá trình với nhau. Java hỗ trợ ống dẫn loại này. Trong đó các quá trình con được thay thế bởi các luồng.
- Ống dẫn có tên (Named Pipe): Loại này có thể cho phép hai quá trình có không gian địa chỉ khác nhau (trên cùng một máy) giao tiếp với nhau. Thực chất nó giống như một tập tin với qui định rằng dữ liệu sẽ được lấy ra ở đầu tập tin và được thêm vào ở cuối tập tin.

3.2 ỐNG DẪN TRONG JAVA

3.2.1 Giới thiệu

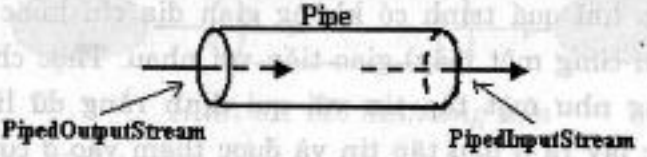
Java hỗ trợ tiên ích ống dẫn thông qua hai lớp **java.io.PipedInputStream** và **java.io.PipedOutputStream**. Chúng là hai đầu của một ống dẫn. Trong đó PipedInputStream là đầu đọc dữ liệu và PipedOutputStream là đầu ghi dữ liệu của ống dẫn.

PipedInputStream là lớp con của InputStream nên nó có tất cả các thuộc tính và phương thức của InputStream.

PipedOutputStream là lớp con của OutputStream nên nó có tất cả các thuộc tính và phương thức của OutputStream.

3.2.2 Các cách tạo ống dẫn

Để tạo một ống dẫn, ta chỉ cần tạo ra hai đối tượng thuộc lớp PipedInputStream và PipedOutputStream và nối chúng lại với nhau. Khi đó dữ liệu được ghi vào PipedOutputStream sẽ được đọc ra ở đầu PipedInputStream:



Hình 3.3 - Tạo ống dẫn trong Java

Cách 1

1. Tạo đầu đọc:
   - PipedInputStream readId = new PipedInputStream();
2. Tạo đầu ghi:
   - PipedOutputStream writeId = new PipedOutputStream();
3. Nối đầu đọc với đầu ghi hay ngược lại
   - readId.connect(writeId);
   - // hoặc writeId.connect(readId);

Cách 2

1. Tạo đầu đọc:
   - PipedInputStream readId = new PipedInputStream();
2. Tạo đầu ghi và nối vào đầu đọc đã có:
   - PipedOutputStream writeId = new PipedOutputStream(readId);

Hoặc: Ta có thể tạo đầu ghi trước rồi tạo đầu đọc sau.

Lưu ý: Các phương thức khởi tạo của PipedInputStream và PipedOutputStream sử dụng ở trên đòi hỏi phải "bắt" (catch) IOException.

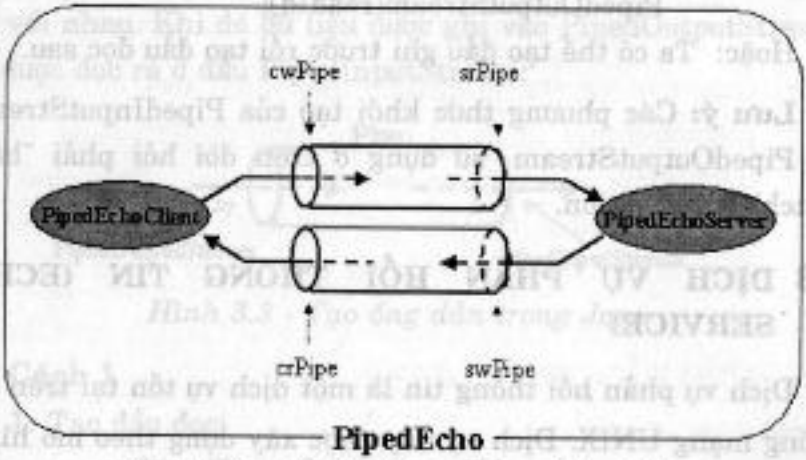
3.3 DỊCH VỤ PHẢN HỒI THÔNG TIN (ECHO SERVICE)

Dịch vụ phản hồi thông tin là một dịch vụ tồn tại trên hệ thống mạng UNIX. Dịch vụ này được xây dựng theo mô hình Client - Server, cơ chế hoạt động như sau:

- Phần Server: Chờ nhận các byte gởi đến từ Client. Với mỗi byte nhận được, Server sẽ gởi về đúng byte đã nhận trở về Client.
- Phần Client: Gởi các byte sang Server, chờ nhận các byte gởi về từ Server.

3.4 GIẢ LẬP DỊCH VỤ PHẢN HỒI THÔNG TIN BẰNG PIPE

Phần kế tiếp ta xây dựng một ứng dụng có tên là PipeEcho mô phỏng dịch vụ phản hồi thông tin để minh họa cách sử dụng Pipe làm phương tiện giao tiếp giữa các Thread trong một ứng dụng.



Hình 3.4 – Mô hình ứng dụng PipeEcho

Trong ứng dụng này Client và Server là hai Thread thuộc lớp PipedEchoClient và PipedEchoServer. Việc trao đổi thông tin giữa client và server được thực hiện thông qua 2 Pipe (cwPipe-srPipe) và (swPipe-crPipe).

PipedEchoClient nhận các ký tự từ bàn phím, gởi chúng sang PipedEchoServer và chờ nhận các ký tự gởi về từ PipedEchoServer để in ra màn hình. PipedEchoServer chờ

nhận từng ký tự từ PipedEchoClient, **đổi ký tự nhận được thành ký tự HOA** và gởi ngược về PipedEchoClient.

3.4.1 Lớp PipedEchoServer

```
import java.io.*;
public class PipedEchoServer extends Thread {
   PipedInputStream readPipe;
   PipedOutputStream writePipe;
   PipedEchoServer(PipedInputStream readPipe, PipedOutputStream
writePipe) {
        this.readPipe = readPipe;
        this.writePipe = writePipe;
        System.out.println("Server is starting ...");
        start();
    }
    public void run() {
        while(true) {
            try {
                int ch = readPipe.read();
                ch = Character.toUpperCase((char)ch);
                writePipe.write(ch);
            }
            catch(IOException ie) { System.out.println("Echo Server
      Error:" +              ie); }
        }
    }
}
```

3.4.2 Lớp PipedEchoClient

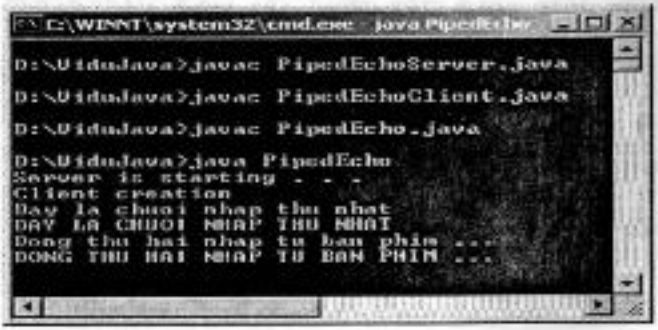
```
import java.io.*;
public class PipedEchoClient extends Thread {
      PipedInputStream readPipe;
      PipedOutputStream writePipe;
    PipedEchoClient(PipedInputStream                    readPipe,
PipedOutputStream writePipe) {
      this.readPipe = readPipe;
      this.writePipe = writePipe;
      System.out.println("Client creation");
      start();
    }
    piblic void run() {
      while (true) {
          try {
                int ch = System.in.read();
                writePipe.write(ch);
                ch = readPipe.read();
                System.out.print((char)ch);
          }
          catch(IOException ie){
                System.out.println("Echo Client Error:" + ie);
          }
      }
    }
}
```

3.4.3 Lớp PipedEcho

```
import java.io.*;
public class PipedEcho {
    public static void main(String argv[]) {
        try {
            PipedOutputStream cwPipe = new PipedOutputStream();
            PipedInputStream crPipe = new PipedInputStream();
            PipedOutputStream swPipe = new
PipedOutputStream(crPipe);
            PipedInputStream srPipe = new PipedInputStream(cwPipe);
            PipedEchoServer server = new
PipedEchoServer(srPipe,swPipe);
            PipedEchoClient client = new PipedEchoClient(crPipe, cwPipe);
        } catch(IOException ie){
            System.out.println("Pipe Echo Error:" + ie);
        }
    }
}
```

3.4.4 Biên dịch và thực thi chương trình

Biên dịch và thực thi chương trình theo cách sau:



Nhập vào bàn phím chuỗi ký tự mà bạn muốn rồi nhấp phím Enter. Bạn sẽ thấy chuỗi ký tự in hoa của chuỗi vừa nhập xuất hiện trên màn hình.