

ĐOÀN VĂN BAN

Lập trình
hướng
đối
tượng
với

JAVA



NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT

ĐOÀN VĂN BAN

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI JAVA

(Tái bản lần thứ nhất: có hiệu chỉnh và bổ sung)



NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT
HÀ NỘI - 2005

LỜI NÓI ĐẦU

Các phương pháp hướng đối tượng, đặc biệt là *lập trình hướng đối tượng* được xây dựng dựa trên nhiều khái niệm mới và được hỗ trợ bởi nhiều công cụ, ngôn ngữ lập trình ([2], [8]) rất mạnh giúp cho việc tạo ra những phần mềm ứng dụng có chất lượng cao, ngày càng đáp ứng tốt hơn yêu cầu của người sử dụng.

Ngôn ngữ Java do gãnh Sun (<http://java.sun.com>) phát triển từ đầu những năm 90 đã trở thành ngôn ngữ lập trình hướng đối tượng rất được ưa chuộng trong những năm gần đây nhờ một số đặc điểm hết sức thích hợp với mạng Internet, hiện đã và đang được dùng phổ biến trên toàn thế giới nhằm đáp ứng các yêu cầu phát triển các ứng dụng trên mạng phục vụ cho nhiều người sử dụng với những môi trường thực hiện phần mềm khác nhau, Java là một ngôn ngữ lập trình hoàn chỉnh được thiết kế theo cách tiếp cận hướng đối tượng và kế thừa, sử dụng lại có nâng cấp của những ngôn ngữ lập trình trước nó.

Về mặt cú pháp, Java rất giống với C++, một ngôn ngữ lập trình hướng đối tượng dùng phổ biến nhất hiện nay, nhưng loại đi một số tính khả dụng (*facilities*) quá mạnh nhưng khó và ít dùng, hoặc thừa về mặt ngôn ngữ [1] như: loại bỏ kế thừa nhiều lớp, vì chúng có thể tạo ra những lược đồ dữ liệu dạng đồ thị và là nguyên nhân chính có thể gây ra sự phức tạp và không đảm bảo tính nhất quán, tính đúng đắn trong quan hệ thông tin ([10], [11]); Java không cho phép thao tác số học trên kiểu con trỏ vì đây là nguồn gốc của những “con bọ” rất khó phát hiện ra khi biên dịch, v.v... Mục đích chính của Java là: *đơn giản, thân thiện, hướng đối tượng và cách tân nhằm tạo ra những phần mềm ứng dụng độc lập với môi trường sử dụng* ([7], [8]).

Cuốn sách này giới thiệu về lập trình hướng đối tượng sử dụng ngôn ngữ lập trình Java. Nội dung chính của cuốn sách được trình bày trong mươi chương.

Chương I trình bày khái quát cách tiếp cận hướng chức năng và lập trình hướng đối tượng. Ngôn ngữ mô hình hóa hệ thống UML [2] được sử dụng để đặc tả các khái niệm cơ bản của lập trình hướng đối tượng. Chương II giới thiệu chu trình phát triển của các chương trình Java, quá trình dịch, thông dịch với JVM (Java Virtual Machine) và các thể loại chương trình ứng dụng, nhất là ứng dụng nhúng (*applet*) của Java. Chương III và chương IV trình bày những khái niệm cơ sở nhất của một ngôn ngữ lập trình và nêu cách xây dựng, tổ chức lớp các đối tượng trong các chương trình ứng dụng. Các lệnh điều khiển dòng thực hiện chương trình, đặc biệt là cơ chế xử lý ngoại lệ hỗ trợ để tạo ra những chương trình hoạt động tốt trong mọi tình huống, thích ứng được với mọi điều kiện trên cơ sở kiểm soát được các lỗi, các tình

huống có thể xảy ra, được giới thiệu chi tiết ở chương V. Chương VI đề cập đến một số lớp cơ sở nhất của Java và các kiểu cấu trúc dữ liệu phổ dụng như kiểu tuyển tập (*Collection*), kiểu tập hợp (*Set*), kiểu danh sách (*List*), v.v. Vấn đề phát triển những ứng dụng *applet* và sử dụng giao diện đồ họa của Windows (AWT) dưới dạng các trang Web với nhiều ví dụ minh họa được giới thiệu ở chương VII. Chương VIII giới thiệu các lớp xử lý các luồng dữ liệu vào/ra chuẩn và những vấn đề có tổ chức, đọc, ghi lên các loại tệp dữ liệu. Chương IX trình bày vấn đề kết nối các cơ sở dữ liệu với JDBC nhằm tạo ra những hệ thống phần mềm tích hợp từ nhiều loại hệ thống thông tin khác nhau trên mạng. Lần tái bản này được bổ sung thêm chương X giới thiệu các thành phần của Swing, cho phép tạo ra những phần mềm mà bạn “thấy và cảm nhận được”. Trong các chương có nhiều ví dụ là những chương trình hoàn chỉnh, minh họa cho cách sử dụng những khái niệm đã nêu ở trên.

Cuốn sách được biên soạn dựa trên kinh nghiệm giảng dạy giáo trình phân tích, thiết kế và lập trình hướng đối tượng của tác giả trong nhiều năm tại các khóa cao học, đại học của ĐH Quốc Gia Hà Nội, ĐH Bách Khoa Hà Nội, ĐH Khoa Học Huế, v.v. Cuốn sách có thể dùng làm giáo trình học tập, tài liệu tham khảo cho sinh viên các hệ kỹ sư, cử nhân, học viên cao học CNTT và các bạn quan tâm đến vấn đề lập trình hướng đối tượng để phát triển những ứng dụng độc lập với môi trường, hay để xây dựng các Web Site trên mạng.

Tác giả bày tỏ lòng biết ơn chân thành tới các bạn đồng nghiệp trong Phòng các Hệ thống phần mềm tích hợp, đặc biệt cảm ơn TS. Đặng Thành Phu, Viện Công nghệ Thông tin, TT KHTN & CNQG, PTS.TS. Đỗ Đức Giáo, Khoa Công nghệ, ĐH QGHN đã động viên, góp ý và giúp đỡ để hoàn chỉnh nội dung cuốn sách này. Xin cảm ơn Nhà xuất bản Khoa học và Kỹ thuật đã hỗ trợ và tạo điều kiện để cuốn sách được tái bản.

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

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

Hà Nội, tháng 5 năm 2005

Tác giả

CHƯƠNG I

GIỚI THIỆU VỀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

1.1. CÁC CÁCH TIẾP CẬN TRONG LẬP TRÌNH

Phương pháp lập trình truyền thống chúng ta vẫn áp dụng đó là *lập trình có cấu trúc*. Phương pháp lập trình này thực hiện theo cách tiếp cận *hướng chức năng* dựa chủ yếu vào việc phân tách các chức năng chính của bài toán thành những chức năng đơn giản hơn và thực hiện làm mịn dần từ trên xuống (*top-down*). Theo cách tiếp cận hướng chức năng, chương trình được xem như là một tập các hàm chức năng, trong đó dữ liệu và các hàm là tách rời nhau. Đối với những hệ thống lớn, phức hợp, độ phức tạp của chương trình tăng lên, sự phụ thuộc của nó vào các kiểu dữ mà nó xử lý cũng tăng theo. Các kiểu dữ liệu được xử lý trong nhiều chức năng bên trong chương trình có cấu trúc, và khi có sự thay đổi về kiểu dữ liệu thì cũng phải thực hiện thay đổi ở mọi nơi mà dữ liệu đó được sử dụng. Một nhược điểm nữa của lập trình hướng chức năng là khi có nhiều người tham gia xây dựng chương trình, mỗi người được giao viết một số chức năng (hàm) riêng biệt nhưng lại phải sử dụng chung dữ liệu. Khi có nhu cầu cần thay đổi dữ liệu sẽ ảnh hưởng rất lớn đến công việc của nhiều người, v.v.

Phương pháp lập trình mà ngày nay chúng ta nghe nói tới nhiều đó là *lập trình hướng đối tượng*. Lập trình hướng đối tượng dựa trên nền tảng là các *đối tượng*. Đối tượng (thực thể) được xây dựng trên cơ sở gắn dữ liệu với các phép toán sẽ thể hiện được đúng cách mà chúng ta suy nghĩ, bao quát về chúng trong thế giới thực [3], [5]. Chẳng hạn, ô tô có bánh xe, di chuyển được và hướng của nó thay đổi được bằng cách thay đổi tay lái. Tương tự, cây là loại thực vật có thân gỗ và lá. Cây không phải là ô tô và những gì thực hiện được với ô tô sẽ không làm được với cây.

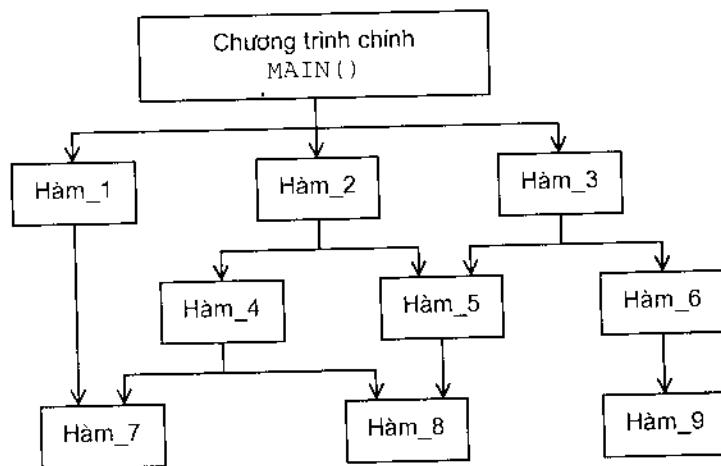
Lập trình hướng đối tượng cho phép chúng ta kết hợp những tri thức bao quát về các quá trình thực tế với những khái niệm trừu tượng được sử dụng trong máy tính. Chương trình hay hệ thống hướng đối tượng được xem như là tập các lớp đối tượng tương tác với nhau để thực hiện các yêu cầu của bài toán đặt ra.

Như vậy, hiện nay chúng ta có hai cách tiếp cận cơ bản để phát triển các hệ thống phần mềm: đó là cách tiếp cận hướng chức năng (*Function-Oriented*) và hướng đối tượng (*Object-Oriented*). Cả hai cách tiếp cận này đều áp dụng một nguyên lý chung để mô hình hóa hệ thống (để hiểu hệ thống) là thực hiện “*chia để trị*”, phân chia bài toán thành những đơn vị tương đối đơn giản mà có thể dễ dàng quản lý được chúng một cách có hiệu quả.

Để hiểu rõ và áp dụng hiệu quả những phương pháp lập trình cần lựa chọn, chúng ta cần phân biệt những đặc trưng cơ bản nhất, đánh giá những mặt mạnh, mặt yếu của chúng.

1.1.1. LẬP TRÌNH HƯỚNG CHỨC NĂNG (THỦ TỤC)

Những ngôn ngữ lập trình bậc cao truyền thống như COBOL, FORTRAN, PASCAL, C v.v..., được gọi chung là ngôn ngữ lập trình *hướng chức năng*. Theo cách tiếp cận hướng chức năng thì một hệ thống phần mềm được xem như là dây các công việc (chức năng) cần thực hiện như nhập dữ liệu, tính toán, xử lý, lập báo cáo và in ấn kết quả v.v... Mỗi công việc đó sẽ được thực hiện bởi một số hàm nhất định. Như vậy trọng tâm của cách tiếp cận này là các *hàm chức năng*. Cấu trúc của chương trình được xây dựng theo cách tiếp cận hướng chức năng có dạng như hình 1.1.

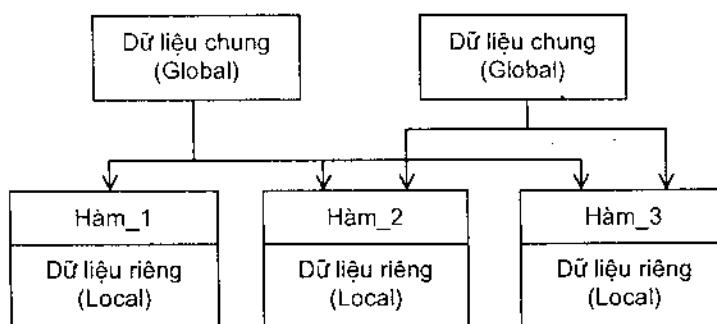


Hình 1.1. Cấu trúc của chương trình hướng chức năng.

Lập trình hướng chức năng (LTHCN) sử dụng kỹ thuật *phân rã các chức năng (hàm)* theo cách tiếp cận *top-down* để tạo ra cấu trúc phân cấp. Chương trình được xây dựng theo cách tiếp cận hướng chức năng thực chất là *tập các chương trình con* (có thể xem như là các hàm) mà theo đó máy tính cần thực hiện để hoàn thành những nhiệm vụ đặt ra của hệ thống.

Khi đặt trọng tâm vào các hàm thì dữ liệu, những cái mà các hàm sử dụng để thực hiện công việc của mình lại trở thành thứ yếu. Cái gì sẽ xảy ra đối với dữ liệu và gán

dữ liệu với các hàm như thế nào? cùng nhiều vấn đề khác cần phải giải quyết khi chúng ta muốn xây dựng các phương pháp phù hợp để phát triển những hệ thống phần mềm giải quyết những yêu cầu đặt ra của thực tế. Hơn nữa, hệ thống luôn là một thể thống nhất, do vậy *các hàm trong một chương trình phải có liên hệ, trao đổi được với nhau*. Như chúng ta đã biết, các hàm (các bộ phận chức năng) chỉ có thể trao đổi được với nhau *qua các tham số*, nghĩa là phải sử dụng biến chung (*global*). Mỗi hàm có thể có vùng dữ liệu riêng còn gọi là *dữ liệu cục bộ (local)*. Mối quan hệ giữa dữ liệu và hàm trong chương trình hướng chức năng được mô tả trong hình 1.2.



Hình 1.2. Quan hệ giữa dữ liệu và hàm trong LTHCN.

Nhiều hàm có thể truy nhập, sử dụng dữ liệu chung, làm thay đổi giá trị của chúng và vì vậy rất khó kiểm soát. Nhất là đối với các chương trình lớn, phức tạp thì vấn đề càng trở nên khó khăn hơn. Khi chúng ta muốn thay đổi, bổ sung cấu trúc dữ liệu dùng chung cho một số hàm thì chúng ta phải thay đổi hầu như tất cả các hàm liên quan đến dữ liệu đó. Nhất là đối với những chương trình, dự án tin học lớn, phức tạp đòi hỏi nhiều người, nhiều nhóm tham gia thì những thay đổi của những biến dữ liệu chung sẽ ảnh hưởng tới tất cả những ai có liên quan tới chúng.

Một đặc tính nữa của cách tiếp cận hướng chức năng dễ nhận thấy là *tính mở (open) của hệ thống kém*. Thứ nhất, vì *dựa chính vào chức năng mà trong thực tế thì nhiệm vụ của hệ thống lại hay thay đổi* nên khi đó muốn cho hệ thống đáp ứng các yêu cầu thì phải thay đổi lại cấu trúc của hệ thống, nghĩa là phải thiết kế, lập trình lại hệ thống. Thứ hai, việc sử dụng các biến dữ liệu chung trong chương trình làm cho các nhóm chức năng phụ thuộc vào nhau về cấu trúc dữ liệu nên cũng hạn chế tính mở của hệ thống. Trong thực tế, cơ cấu tổ chức của mọi tổ chức thường ít thay đổi hơn là chức năng, nhiệm vụ phải thực hiện.

Mặt khác, cách tiếp cận hướng chức năng lại *tách dữ liệu khỏi chức năng xử lý* nên vấn đề che giấu, bảo vệ thông tin trong hệ thống là kém, nghĩa là vấn đề *an toàn, an ninh dữ liệu* là rất phức tạp.

Ngoài ra cách tiếp cận hướng chức năng cũng không hỗ trợ việc *sử dụng lại và kế*

thừa nên chất lượng và giá thành của các phần mềm rất khó được cải thiện. Những trở ngại mà chúng ta đã nêu ở trên sẽ làm cho mô hình được xây dựng theo cách tiếp cận hướng chức năng không mô tả được đầy đủ, trung thực hệ thống trong thực tế.

1.1.2. LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Ngược lại, lập trình hướng đối tượng (LTHĐT) *đặt trọng tâm vào các đối tượng*, yếu tố quan trọng trong quá trình phát triển chương trình và nó không cho phép dữ liệu tách biệt, chuyển động tự do trong hệ thống. Dữ liệu được gắn chặt với các hàm thành phần và chúng được tổ chức, quản lý truy nhập theo nhiều mức khác nhau.

LTHĐT cho phép chúng ta phân tích bài toán thành *tập các thực thể* được gọi là các *lớp đối tượng*, sau đó xây dựng các dữ liệu thành phần cùng với các hàm thành phần thao tác trên các dữ liệu đó và trao đổi với những đối tượng khác để thực hiện những nhiệm vụ được giao.

Một chương trình, một hệ thống được xem như là một tập các lớp đối tượng và các đối tượng đó trao đổi với nhau thông qua việc gửi và nhận các thông điệp (message), do vậy một chương trình hướng đối tượng thực sự có thể hoàn toàn không cần sử dụng biến chung [1], [3].

Lập trình hướng đối tượng *dựa chủ yếu vào các đối tượng*, nên khi có nhu cầu thay đổi thì chỉ cần thay đổi ở một số lớp có liên quan, hoặc có thể bổ sung một số lớp mới trên cơ sở kế thừa và sử dụng lại nhiều nhất có thể. Mặt khác, như trên đã phân tích, một chương trình hướng đối tượng có thể không sử dụng hoặc hạn chế sử dụng các biến chung, do vậy dễ dàng tạo ra được những hệ thống có *tính mở* cao hơn.

Cơ chế bao bọc, che giấu thông tin của phương pháp hướng đối tượng giúp tạo ra được những *hệ thống an toàn, an ninh* cao hơn cách tiếp cận hướng chức năng.

Hơn nữa, phương pháp hướng đối tượng còn hỗ trợ rất mạnh nguyên lý *sử dụng lại* *nhiều nhất có thể* và *tạo ra mọi khả năng* để kế thừa những lớp đã được thiết kế, lập trình tốt để nhanh chóng tạo ra được những phần mềm có chất lượng, giá thành rẻ hơn và đáp ứng các yêu cầu của người sử dụng.

1.2. NHỮNG KHÁI NIỆM CƠ BẢN CỦA LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Để tiện lợi và phù hợp với xu thế phát triển hiện nay của công nghệ thông tin, chúng ta sẽ sử dụng UML (*Unified Modelling Language* [2], [7]) để đặc tả những khái niệm cơ bản của lập trình hướng đối tượng thông qua *Rational Rose* (<http://www.rational.com>). UML là *ngôn ngữ mô hình hóa hình thức, thống nhất* và

trực quan. Phần lớn các thông tin trong mô hình được thể hiện bởi các ký hiệu đồ họa, biểu đồ thể hiện mối quan hệ giữa các thành phần của hệ thống một cách thống nhất và có logic chặt chẽ.

UML được sử dụng để *đặc tả, xây dựng và làm tài liệu cho các tác phẩm (Artifacts), các kết quả của các phân tích, thiết kế và lập trình hướng đối tượng* dưới dạng các biểu đồ, bản mẫu hay các trang Web.

UML là ngôn ngữ chuẩn công nghiệp để lập kế hoạch chi tiết phát triển phần mềm. Hiện nay nhiều hãng sản xuất phần mềm lớn như: Microsoft, IBM, HP, Oracle, Digital Equipment Corp., Texas Instruments, Rational Software, v.v., sử dụng UML như là chuẩn cho ngôn ngữ mô hình hóa hệ thống phần mềm.

Phương pháp hướng đối tượng được xây dựng dựa trên một tập các khái niệm cơ sở:

1. Đối tượng (object),
2. Lớp đối tượng (class),
3. Trừu tượng hóa dữ liệu (Data Abstraction),
4. Bao bọc và che giấu thông tin (Encapsulation and Information Hiding),
5. Mở rộng, kế thừa giữa các lớp (Inheritance),
6. Đa xạ và nạp chồng (Polymorphism and Overloading),
7. Liên kết động (Dynamic Binding),
8. Truyền thông điệp (Message Passing).

1.2.1. ĐỐI TƯỢNG

Đối tượng là khái niệm cơ sở, quan trọng nhất của cách tiếp cận hướng đối tượng. Đối tượng là thực thể của hệ thống, của CSDL và được xác định thông qua định danh ID (*IDentifier*) của chúng. Mỗi *đối tượng* có *tập các đặc trưng bao gồm* cả các phần tài sản thường là *các dữ liệu thành phần* hay các thuộc tính mô tả các tính chất và *các phương thức, các thao tác trên các dữ liệu* để xác định hành vi của đối tượng đó.

Đối tượng là những thực thể được xác định trong thời gian hệ thống hướng đối tượng hoạt động. Như vậy đối tượng có thể biểu diễn cho người, vật, hay một bảng dữ liệu hoặc bất kỳ một hạng thức nào đó cần xử lý trong chương trình. Đối tượng cũng có thể là các dữ liệu được định nghĩa bởi người sử dụng (người lập trình) như *vector*, danh sách, các *record* v.v... Nhiệm vụ của LTHĐT là phân tích bài toán thành các đối tượng và xác định được bản chất của sự trao đổi thông tin giữa chúng. Đối tượng trong chương trình cần phải được chọn sao cho nó thể hiện được một cách gần nhất với những thực thể có trong hệ thống thực.

Như vậy, đối tượng được định nghĩa một cách trừu tượng như là một khái niệm, một

cấu trúc gộp chung cả phần dữ liệu (thuộc tính) với các hàm (phương thức) thao tác trên những dữ liệu đó và có thể trao đổi với những đối tượng khác. Theo quan điểm của người lập trình [3], *đối tượng được xem như là vùng bộ nhớ được phân chia trong máy tính để lưu trữ dữ liệu và tập các hàm tác động trên dữ liệu gắn với chúng*. Bởi vì các vùng phân hoạch bộ nhớ là độc lập với nhau nên các đối tượng có thể sử dụng bởi nhiều chương trình khác nhau mà không ảnh hưởng lẫn nhau.

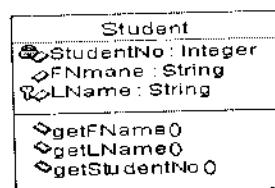
1.2.2. LỚP ĐỐI TƯỢNG

Lớp là bản mẫu hay một kiểu chung cho tất cả những đối tượng có những *đặc trưng giống nhau*, nghĩa là có các thuộc tính và hành vi giống nhau. Đối tượng chính là thể hiện (cá thể) của một lớp xác định. Trong lập trình hướng đối tượng, lớp được xem là đồng nhất với kiểu dữ liệu trừu tượng (ADT - Abstract Data Type) được Barbara đề xuất vào những năm 70).

Phương pháp lập trình *hướng đối tượng* là cách phân chia chương trình thành các đơn thể (các lớp) bằng cách tạo ra các vùng bộ nhớ cho cả dữ liệu lẫn hàm và chúng sẽ được sử dụng như các mẫu để tạo ra bản sao từng đối tượng khi chúng được tạo ra trong hệ thống.

Nhiệm vụ của người lập trình là tìm cách xác định đầy đủ, chính xác danh sách các lớp đại diện cho tất cả các thực thể trong hệ thống và mối quan hệ giữa các lớp đối tượng đó [3], [5].

Như vậy, lớp chính là tập các đối tượng có cùng các thuộc tính và hành vi giống nhau. Các thành phần của lớp có thể chia thành ba vùng quản lý chính: công khai (*public*), được bảo vệ (*protected*) và vùng riêng (*private*). Trong ngôn ngữ mô hình hóa thống nhất UML, cấu trúc của lớp thường được đặc tả bởi: tên của lớp, tập các thuộc tính và tập các hàm.



Hình 1.3. Lớp Student trong UML.

Lớp có tên là *Student*. Lớp có ba thuộc tính: *StudentNo* có kiểu *Integer* là sở hữu riêng (*private*), bị khóa; thuộc tính *FName* có kiểu *String* là công khai (*public*) và thuộc tính *LName* là được bảo vệ (*protected*). Lớp *Student* có ba hàm : *getStudentNo()*, *getFName()*, *getLName()* đều là công khai.

1.2.3. TRỪU TƯỢNG HÓA DỮ LIỆU

Trừu tượng hóa là cách biểu diễn những đặc tính chính và bỏ qua những chi tiết. *Trừu tượng hóa* là sự mở rộng khái niệm kiểu dữ liệu và cho phép định nghĩa những phép toán trừu tượng trên các dữ liệu trừu tượng.

Để xây dựng các lớp, chúng ta phải sử dụng khái niệm trừu tượng hóa. Ví dụ, chúng ta có thể định nghĩa một lớp là danh sách các thuộc tính trừu tượng như là kích thước, hình dáng, màu và các hàm xác định trên các thuộc tính này để mô tả các đối tượng trong không gian hình học. Trong lập trình, lớp được sử dụng như *kiểu dữ liệu trừu tượng*.

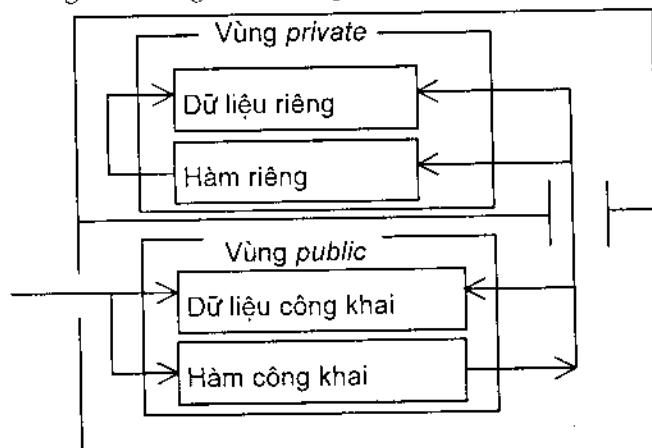
Khi xây dựng phần mềm thì nguyên lý trừu tượng hóa được thực hiện thông qua việc trừu tượng hóa các chức năng, hàm (thuật toán) và trừu tượng hóa các kiểu dữ liệu nguyên thủy.

1.2.4. BAO BỌC VÀ CHE GIẤU THÔNG TIN

Việc đóng gói dữ liệu và các hàm vào một đơn vị cấu trúc (gọi là lớp) được xem như một nguyên tắc . Kỹ thuật này cho phép xác định các vùng đặc trưng riêng, công khai hay được bảo vệ bao gồm cả dữ liệu và các câu lệnh nhằm điều khiển hoặc hạn chế những truy nhập tùy tiện của những đối tượng khác. Dữ liệu được tổ chức sao cho thế giới bên ngoài (các đối tượng ở lớp khác) không truy nhập được vào những thuộc tính riêng và chỉ cho phép các hàm trong cùng lớp hoặc trong những lớp có quan hệ kế thừa với nhau được quyền truy nhập đến vùng được bảo vệ. Vùng công khai của lớp thì cho phép mọi đối tượng được phép truy nhập.

Chính các hàm thành phần công khai của lớp sẽ đóng vai trò như là giao diện giữa các đối tượng và với phần còn lại của hệ thống. Nguyên tắc bao bọc dữ liệu để ngăn cấm sự truy nhập trực tiếp trong lập trình được gọi là *sự che giấu thông tin*.

Nguyên lý bao bọc che giấu thông tin của lớp đối tượng được mô tả như trong hình 1.4.



Hình 1.4. *Bao bọc và che giấu thông tin của lớp đối tượng*.

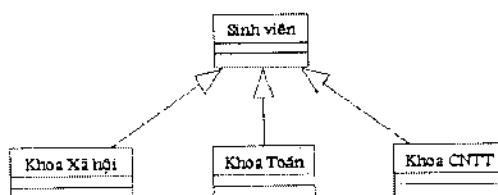
1.2.5. SỰ MỞ RỘNG, KẾ THỪA GIỮA CÁC LỚP

Nguyên lý kế thừa cho phép các đối tượng của lớp này được quyền sử dụng một số tính chất (cả dữ liệu và các hàm thành phần) của các lớp khác. Một lớp có thể là *lớp con* (*lớp dẫn xuất*) của một lớp khác, nghĩa là có thể bổ sung thêm một số tính chất để thu hẹp phạm vi xác định các đối tượng trong lớp mới cho phù hợp với ngữ cảnh trong thực tế.

Theo nguyên lý chung của kế thừa thì chỉ những thuộc tính, hàm thành phần được bảo vệ và công khai là được quyền kế thừa, còn những thuộc tính, hàm thành phần riêng là không được phép kế thừa. Những thuộc tính và hàm được kế thừa từ *lớp cha* (*lớp cơ sở*) được xem như là “tài sản” của chính các đối tượng con cháu, chúng có quyền sử dụng mà không cần phải khai báo hay định nghĩa lại. Như vậy, nguyên lý kế thừa trong lập trình hướng đối tượng hoàn toàn đồng nhất với nguyên lý kế thừa gia sản trong xã hội loài người.

Phương pháp hướng đối tượng nói chung hỗ trợ hai nguyên lý kế thừa: *kế thừa đơn* và *kế thừa bội*. Kế thừa đơn là một lớp có thể kế thừa từ một lớp cơ sở, còn kế thừa bội là một lớp có thể kế thừa từ nhiều hơn một lớp cơ sở. Ngôn ngữ C++ hỗ trợ cả hai nguyên lý kế thừa, nhưng Java chỉ hỗ trợ thực hiện *kế thừa đơn* [8].

Nguyên lý kế thừa đơn hỗ trợ cho việc tạo ra cấu trúc cây phân cấp các lớp. Ví dụ, một trường đại học đào tạo sinh viên có ba khoa: Khoa Xã hội, Khoa Công nghệ Thông tin và Khoa Toán. Chúng ta có thể xây dựng lớp *Sinh viên* là lớp cơ sở để từ đó xây dựng tiếp ba lớp kế thừa là: *Khoa Xã hội*, *Khoa Toán* và *Khoa CNTT*. Hệ thống sẽ được thiết kế thành các lớp kế thừa và được mô tả trong UML như sau:



Hình 1.5. Cấu trúc phân cấp các lớp theo quan hệ kế thừa trong UML.

Trong LTHDT, khái niệm kế thừa kéo theo ý tưởng sử dụng lại. Nghĩa là từ một lớp đã được xây dựng chúng ta có thể bổ sung thêm một số tính chất tạo ra một lớp mới kế thừa lớp cũ mà không làm thay đổi những cái đã có.

Khái niệm kế thừa được hiểu như cơ chế sao chép ảo không đơn điệu. Trong thực tế, mọi việc xảy ra tựa như những lớp cơ sở đều được sao vào trong lớp con (lớp dẫn xuất) mặc dù điều này không được cài đặt tường minh (nên gọi là sao chép ảo) và việc sao chép chỉ thực hiện đối với những thông tin chưa được xác định trong các lớp

cơ sở (sao chép không đơn diệu). Do vậy, có thể diễn đạt cơ chế kế thừa như sau:

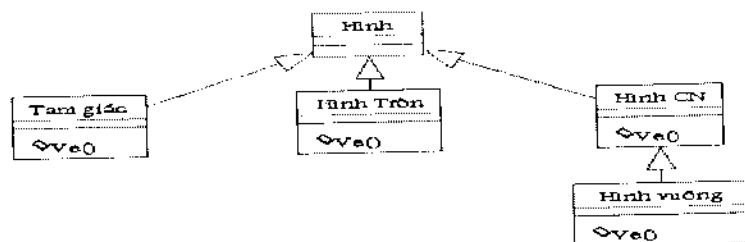
1. Lớp A kế thừa lớp B sẽ có (không tường minh) tất cả các thuộc tính, hàm đã được xác định trong B, ở những vùng được phép kế thừa (được bảo vệ, công khai),
2. Bổ sung thêm một số thuộc tính, hàm để mô tả được đúng các hành vi của những đối tượng mà lớp A quản lý.

1.2.6. ĐA XẠ (TƯƠNG ỨNG BỘI) VÀ NẠP CHỒNG

Một khái niệm quan trọng nữa trong LTHDT là khái niệm *đa xạ* tương tự như trong toán học. *Đa xạ* là kỹ thuật được sử dụng để mô tả *khả năng gửi một thông điệp chung tới nhiều đối tượng mà mỗi đối tượng lại có cách xử lý riêng theo ngữ cảnh của mình*.

Đa xạ đóng một vai trò quan trọng trong việc tạo ra các đối tượng có cấu trúc với những nội dung thực hiện khác nhau mà lại có khả năng sử dụng chung một giao diện (cùng một tên gọi). Theo một nghĩa nào đó, *đa xạ* là sự mở rộng khái niệm sử dụng lại trong nguyên lý kế thừa. Hình 1.6 cho chúng ta thấy hàm có tên là *Ve()* có thể sử dụng để vẽ các hình khác nhau phụ thuộc vào đối tượng là các hình khi gọi để thực hiện.

Tam giác, *Hình tròn* và *Hình chữ nhật* là các lớp con của lớp *Hình*. Hàm *Ve()* là hàm đa xạ và nó được xác định tùy theo ngữ cảnh khi sử dụng. Khi gọi thực hiện đối tượng là *Tam giác* thì sẽ vẽ tam giác, nếu đối tượng là *hình tròn* thì vẽ hình tròn, v.v.



Hình 1.6. Tương ứng bội của hàm *Ve()*.

Nạp chồng (Overloading) là một trường hợp của đa xạ. *Nạp chồng* là khả năng của một khái niệm (như các phép toán chẳng hạn) có thể được sử dụng với nhiều nội dung thực hiện khác nhau tùy theo ngữ cảnh, cụ thể là tùy thuộc vào kiểu và số các tham số của chúng. Ví dụ, hàm *Cong()* có thể được nạp chồng để cộng các số nguyên (*int*), số thực (*float*), số phức (*Complex*) hoặc ghép các xâu ký tự (*String*), v.v.

int Cong(int, int); // Cộng hai số nguyên

```

float Cong(float, float);           // Cộng hai số thực
Complex Cong(Complex, Complex);   // Cộng hai số phức
String Cong(String, String);      // Ghép hai xâu
String Cong(String, int);         // Ghép một xâu với một số nguyên

```

1.2.7. LIÊN KẾT ĐỘNG

Liên kết thông thường (liên kết tĩnh) là dạng liên kết được xác định ngay khi dịch chương trình. Hầu hết các chương trình được viết bằng Pascal, C đều là dạng liên kết tĩnh. *Liên kết động* là dạng liên kết các hàm, chức năng khi chương trình thực hiện các lời gọi các hàm, chức năng đó. Như vậy trong liên kết động, nội dung của đoạn chương trình ứng với chức năng, hàm sẽ không được xác định cho đến khi thực hiện lời gọi tới chức năng, hàm đó. Liên kết động liên quan chặt chẽ với những khái niệm đa xạ và kế thừa trong lập trình hướng đối tượng. Chính nhờ khái niệm liên kết động mà nhiều khái niệm của lập trình hướng đối tượng như khái niệm đa xạ thực hiện được.

Chúng ta hãy xét hàm *Ve()* trong hình 1.6. Theo nguyên lý kế thừa thì mọi đối tượng đều có thể sử dụng hàm này để vẽ các hình cụ thể khi hệ thống thực hiện. Hàm *Ve()* được định nghĩa lại ở trong các lớp dẫn xuất để vẽ tam giác, hình tròn hay hình chữ nhật theo các thuật toán tương ứng. Khi thực hiện, ví dụ: nếu đối tượng *Hình* là *Hình tròn* thì hệ thống sẽ liên kết với hàm *Ve()* được định nghĩa trong lớp *Hình tròn* để vẽ hình tròn.

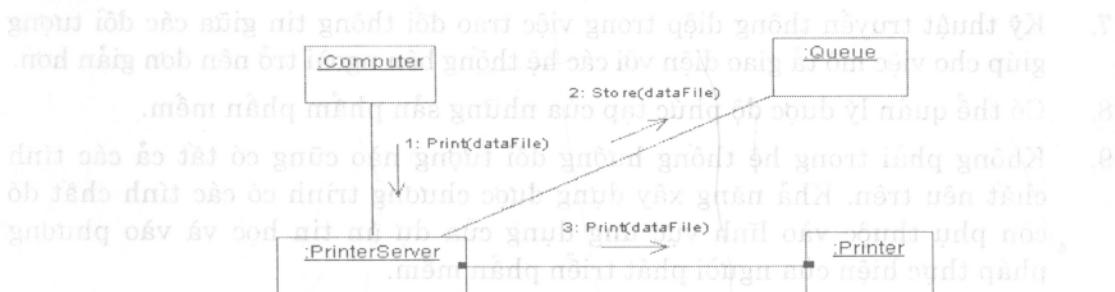
1.2.8. TRUYỀN THÔNG ĐIỆP

Chương trình hướng đối tượng (được thiết kế và lập trình theo phương pháp hướng đối tượng) bao gồm một tập các đối tượng và mối quan hệ giữa các đối tượng đó với nhau. Vì vậy, lập trình trong ngôn ngữ hướng đối tượng bao gồm các bước sau:

1. Tạo ra các lớp đối tượng và mô tả hành vi của chúng;
2. Tạo ra các đối tượng theo định nghĩa của các lớp;
3. Xác định sự trao đổi thông tin giữa các đối tượng trong hệ thống.

Các đối tượng gửi và nhận thông tin trong hệ thống hướng đối tượng cũng giống như con người trao đổi với nhau trong xã hội. Chính nguyên lý trao đổi thông tin bằng cách truyền thông điệp cho phép chúng ta dễ dàng xây dựng được hệ thống mô phỏng gần hơn với thực tế. Truyền thông điệp cho một đối tượng tức là báo cho nó phải thực hiện một việc, một yêu cầu (thỉnh cầu) nào đó. Cách ứng xử của đối tượng sẽ được mô tả ở trong lớp thông qua các hàm công khai (hay còn được gọi là lớp dịch vụ).

Trong chương trình, thông điệp gửi đến cho một đối tượng chính là để yêu cầu thực hiện một công việc cụ thể, nghĩa là sử dụng những hàm tương ứng để xử lý dữ liệu đã được khai báo trong lớp đối tượng đó. Vì vậy, trong thông điệp phải chỉ ra được hàm cần thực hiện của đối tượng nhận thông điệp. Hơn thế nữa, thông điệp truyền đi phải xác định tên đối tượng, tên hàm (thông điệp) và thông tin truyền đi. Ví dụ, khi hệ thống máy tính muốn in một tệp *dataFile* thì máy tính hiện thời (*:Computer*) gửi đến cho đối tượng *:PrinterServer* một yêu cầu *Print(dataFile)*. Bộ phận dịch vụ máy in sẽ kiểm tra xem nếu máy in đang bận thì lưu yêu cầu đó vào hàng đợi bằng cách gửi cho (*:Queue*) thông điệp là *Store(dataFile)*, ngược lại gửi yêu cầu in *Print(dataFile)* cho đối tượng *:Printer*. Hoạt động trao đổi thông điệp giữa các đối tượng trên có thể mô tả trong UML như hình 1.7.



Hình 1.7. Truyền thông điệp giữa các đối tượng.

Mỗi đối tượng chỉ tồn tại trong thời gian nhất định. Đối tượng được tạo ra khi nó được khai báo và sẽ bị hủy bỏ khi chương trình ra khỏi miền xác định của đối tượng đó. Sự trao đổi thông tin chỉ có thể thực hiện trong thời gian tồn tại của đối tượng.

1.2.9. CÁC ƯU ĐIỂM CỦA LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Như trên đã phân tích, lập trình hướng đối tượng đem lại một số lợi thế cho cả người thiết kế lẫn người lập trình. Cách tiếp cận hướng đối tượng giải quyết được nhiều vấn đề tồn tại trong quá trình phát triển phần mềm và tạo ra được những sản phẩm phần mềm có chất lượng cao. Những phương pháp này mở ra một triển vọng to lớn cho những người lập trình. Hy vọng sẽ có nhiều sản phẩm phần mềm tốt hơn, đáp ứng được những tính chất về sản phẩm chất lượng cao trong Công nghệ phần mềm và nhất là bảo trì hệ thống ít tổn kém hơn. Những ưu điểm chính của LTHDT là:

1. Thông qua nguyên lý kế thừa, chúng ta có thể loại bỏ được những đoạn chương trình lặp lại, dư thừa trong quá trình mô tả các lớp và mở rộng khả năng sử dụng các lớp đã được xây dựng.
2. Chương trình được xây dựng từ những đơn thể (đối tượng) trao đổi với nhau nên việc thiết kế và lập trình sẽ được thực hiện theo quy trình nhất định chứ không phải dựa vào kinh nghiệm và kỹ thuật như trước. Điều

này đảm bảo rút ngắn được thời gian xây dựng hệ thống và tăng năng suất lao động.

3. Nguyên lý che giấu thông tin giúp người lập trình tạo ra được những chương trình an toàn không bị thay đổi bởi những đoạn chương trình khác một cách tuỳ tiện.
4. Có thể xây dựng được ánh xạ các đối tượng của bài toán vào đối tượng của chương trình.
5. Cách tiếp cận thiết kế đặt trọng tâm vào dữ liệu, giúp chúng ta xây dựng được mô hình chi tiết và phù hợp với thực tế.
6. Những hệ thống hướng đối tượng dễ mở rộng, nâng cấp thành những hệ lớn hơn.
7. Kỹ thuật truyền thông điệp trong việc trao đổi thông tin giữa các đối tượng giúp cho việc mô tả giao diện với các hệ thống bên ngoài trở nên đơn giản hơn.
8. Có thể quản lý được độ phức tạp của những sản phẩm phần mềm.
9. Không phải trong hệ thống hướng đối tượng nào cũng có tất cả các tính chất nêu trên. Khả năng xây dựng được chương trình có các tính chất đó còn phụ thuộc vào lĩnh vực ứng dụng của dự án tin học và vào phương pháp thực hiện của người phát triển phần mềm.

1.3. CÁC NGÔN NGỮ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Lập trình hướng đối tượng không là đặc quyền của một ngôn ngữ nào đặc biệt. Cũng giống như lập trình có cấu trúc, những khái niệm trong lập trình hướng đối tượng có thể cài đặt trong những ngôn ngữ lập trình như C hoặc Pascal. Tuy nhiên, đối với những chương trình lớn, phức hợp thì vấn đề lập trình sẽ trở nên phức tạp, nếu sử dụng những ngôn ngữ không phải là ngôn ngữ hướng đối tượng thì phải thực hiện nhiều thỏa hiệp. Những ngôn ngữ được thiết kế đặc biệt, hỗ trợ cho việc mô tả, cài đặt các khái niệm của phương pháp hướng đối tượng được gọi chung là ngôn ngữ hướng đối tượng.

Dựa vào khả năng đáp ứng các khái niệm về hướng đối tượng, chúng ta có thể chia ra làm hai loại:

1. Ngôn ngữ lập trình dựa trên đối tượng (object-based),
2. Ngôn ngữ lập trình hướng đối tượng (object-oriented).

Lập trình dựa trên đối tượng là kiểu lập trình hỗ trợ chính cho việc bao bọc, che giấu thông tin và định danh các đối tượng. Lập trình dựa trên đối tượng có những đặc tính sau:

- Bao bọc dữ liệu,
- Cơ chế che giấu và hạn chế truy nhập dữ liệu,

- Tự động tạo lập và hủy bỏ các đối tượng,
- Tính đa xạ.

Ngôn ngữ hỗ trợ cho kiểu lập trình trên được gọi là ngôn ngữ lập trình dựa trên đối tượng. Ngôn ngữ trong lớp này không hỗ trợ cho việc thực hiện kế thừa và liên kết động. Ada là ngôn ngữ lập trình dựa trên đối tượng.

Lập trình hướng đối tượng là kiểu lập trình dựa trên đối tượng và bổ sung thêm nhiều cấu trúc để cài đặt những quan hệ về kế thừa và liên kết động. Vì vậy đặc tính của LTHĐT có thể viết một cách ngắn gọn như sau:

Các đặc tính dựa trên đối tượng + kế thừa + liên kết động.

Ngôn ngữ hỗ trợ cho những đặc tính trên được gọi là ngôn ngữ LTHĐT, ví dụ như *Java, C++, Smalltalk, Object Pascal hay Eiffel*, v.v...

Việc chọn một ngôn ngữ để cài đặt phần mềm phụ thuộc nhiều vào các đặc tính và yêu cầu của bài toán ứng dụng, vào khả năng sử dụng lại của những chương trình đã có và vào tổ chức của nhóm tham gia xây dựng phần mềm. Một trong những ngôn ngữ lập trình hướng đối tượng thực sự được sử dụng phổ biến hiện nay là Java [8] sẽ được giới thiệu chi tiết ở các chương sau.

BÀI TẬP

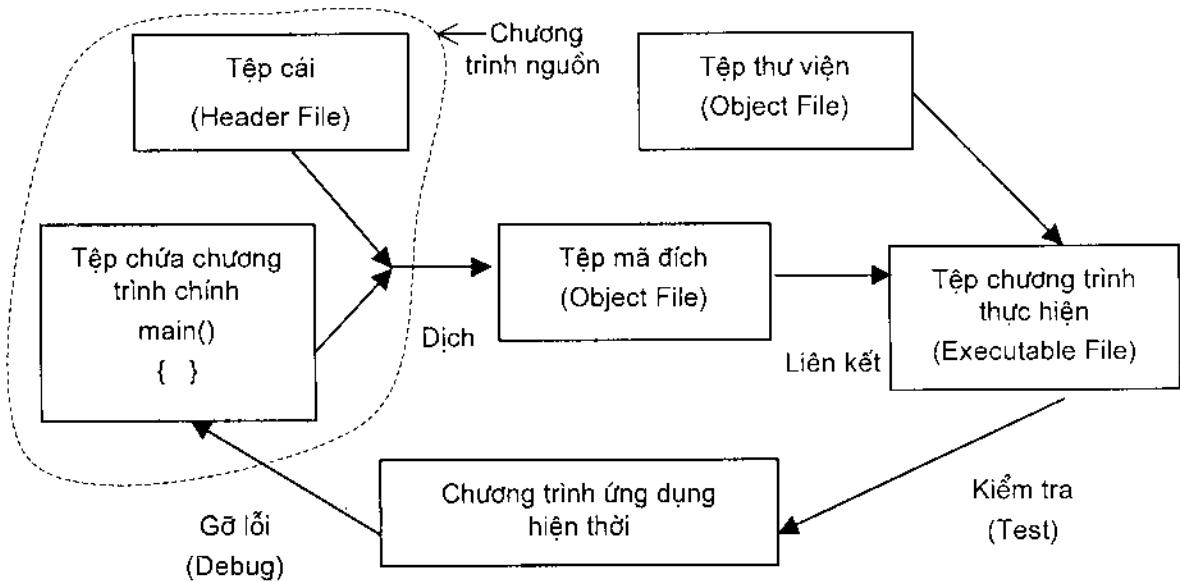
- 1.1. Nêu các đặc trưng cơ bản của cách tiếp cận lập trình hướng chức năng và lập trình hướng đối tượng.
- 1.2. Tại sao lại gọi khái niệm lớp là kiểu dữ liệu trừu tượng trong lập trình hướng đối tượng.
- 1.3. Nêu nguyên lý hoạt động của khái niệm đa xạ, tương ứng bội trong lập trình hướng đối tượng, khái niệm này thực hiện được trong lập trình hướng chức năng hay không, tại sao?
- 1.4. Khái niệm kế thừa và sử dụng lại trong lập trình hướng đối tượng là gì?, ngôn ngữ lập trình C++ và Java hỗ trợ quan hệ kế thừa như thế nào?.

CHƯƠNG II

GIỚI THIỆU VỀ LẬP TRÌNH VỚI JAVA

2.1. GIỚI THIỆU CHUNG

Các chương trình dịch của các ngôn ngữ lập trình truyền thống như C / C++, Pascal thường dịch các tệp chương trình nguồn sang các câu lệnh đặc biệt mà máy tính của bạn hiểu được. Những ngôn ngữ như C/C++ được xây dựng trên cơ sở tạo ra những chương trình dịch tối ưu với mã máy thực hiện hiệu quả, nhanh và linh hoạt. Chu trình phát triển và thực hiện của chương trình viết bằng những ngôn ngữ truyền thống như C/C++ có thể được mô tả như sau



Hình 2.1. Chu trình phát triển và thực hiện của chương trình C/C++.

Tuy nhiên để tối ưu hóa được chương trình dịch thì nó phải được thực hiện dựa *trên một kiến trúc (tập các lệnh) xác định*, nghĩa là phụ thuộc vào cấu hình của máy đích. Để tạo ra được sự độc lập tương đối thì chương trình của bạn phải dịch lại để phù

hợp với kiến trúc máy mới mỗi khi bạn thay đổi môi trường thực hiện chương trình.

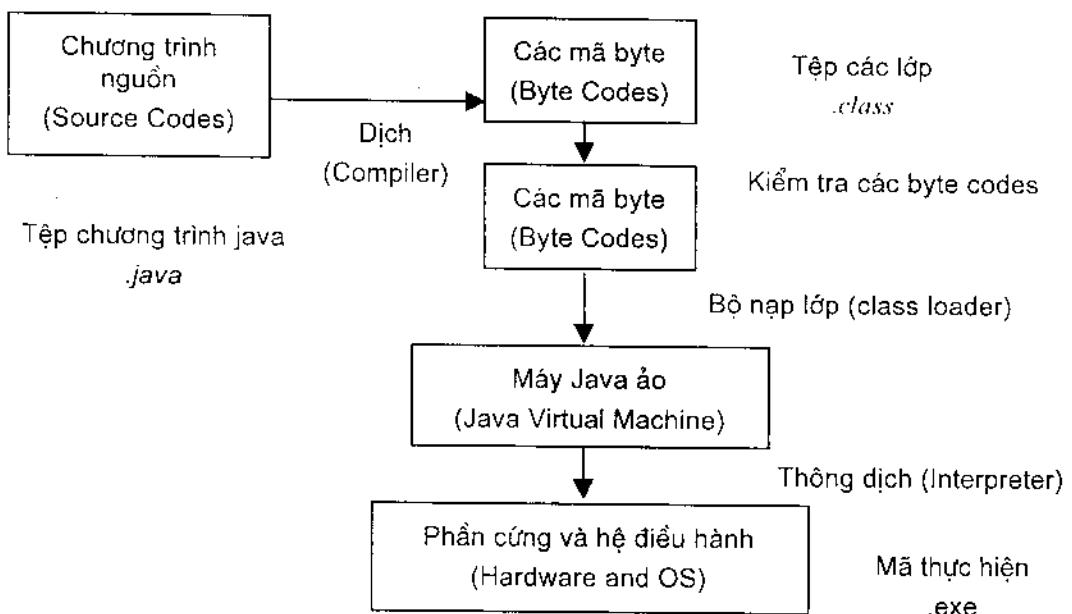
Hiện nay có nhiều loại máy tính với nhiều cấu hình khác nhau. Thông thường với mỗi loại ứng dụng chúng ta chọn một loại máy tính và cấu hình hợp lý để phát triển chương trình cho hiệu quả. Tuy nhiên, khi điều kiện thực hiện chương trình thay đổi thường gây rất nhiều trở ngại cho người sử dụng. Do đó những người phát triển chương trình ứng dụng (người lập trình) muốn phát kiến những phương thức làm việc mới có thể độc lập được với những thay đổi của môi trường và mong muốn có sự hỗ trợ của các ngôn ngữ lập trình.

Đặc biệt, *mạng Internet* được xem như là mạng của nhiều mạng khác nhau về nhiều lĩnh vực, cả phần cứng lẫn phần mềm. Mạng cung cấp nhiều dịch vụ tiện lợi cho nhiều ứng dụng khác nhau, nhất là các ứng dụng trên mạng của công nghệ Web. Để đưa được các yếu tố lập trình lên mạng và kết hợp với Web thì không nên sử dụng các chương trình viết bằng C/C++, bởi vì:

1. Vấn đề an toàn, an ninh dữ liệu trên mạng không đảm bảo,
2. Quan trọng hơn là các chương trình C/C++ được dịch sang một mã máy có cấu hình cố định, vì vậy khi được nạp từ trên mạng xuống một máy có cấu hình khác sẽ không thực hiện được. Ví dụ, một chương trình đã được dịch ở Macintosh thì sẽ không thực hiện được ở Windows và ngược lại.

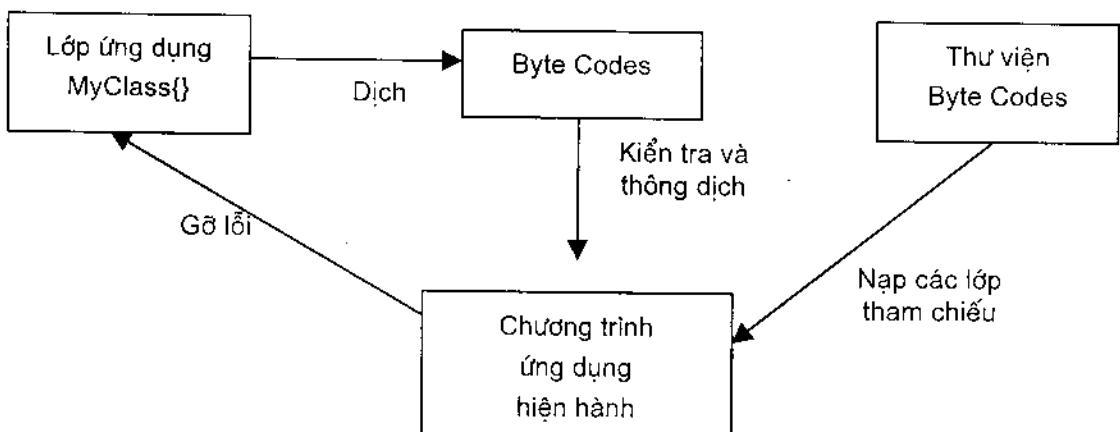
Điều mà chúng ta muốn nhất hiện nay là cần có một *ngôn ngữ thông dịch* thật mạnh để khắc phục những nhược điểm trên, đặc biệt để phát triển được dễ dàng các ứng dụng với Web trên mạng. Nhưng theo nguyên lý của chương trình thông dịch (*Interpreter*) thì nó thực hiện khá chậm. Nếu chương trình thông dịch phải thực hiện phân tích chương trình nguồn mỗi khi thực hiện thì các chương trình ứng dụng sẽ rất chậm.

Java vượt qua được các nhược điểm trên bằng cách dịch các chương trình nguồn sang ngôn ngữ máy ảo không phụ thuộc vào *chip (hệ lệnh cụ thể)* nào cả và sau đó khi cần thực hiện sẽ thông dịch sang hệ máy cụ thể. Kết quả của chương trình dịch không phải là mã đích (*Object Code*) như kết quả của các chương trình dịch truyền thống mà là chuỗi các bytes cơ sở bao gồm các mã lệnh thực hiện (*Opcode*) và các tham số của máy lý thuyết (máy ảo). Máy này được gọi là máy Java ảo (*JVM - Java Virtual Machine*). JVM có thể nhúng vào bất kỳ một biểu diễn, một môi trường cụ thể mà bạn có. Khi cài đặt trên một môi trường xác định (*máy có cấu hình cố định*) thì phần mềm được thông dịch và JVM trở thành máy cụ thể để có thể thực thi ứng dụng của bạn. Chương trình Java được thực hiện như sau:



Hình 2.2. Quá trình dịch và thông dịch chương trình Java.

Công nghệ Java giải quyết vấn đề tốc độ bằng cách dịch chương trình nguồn sang các *mã byte* (*byte codes*). Khi JVM thực hiện, nó sẽ tìm các đối tượng cần tham chiếu của các lớp trong chương trình chính (*chương trình ứng dụng*) ở thời điểm thực hiện để nạp chúng xuống. Quá trình phát triển và thực hiện chương trình Java thực hiện như sau:



Hình 2.3. Quá trình phát triển chương trình Java.

Như vậy, Java là *ngôn ngữ động* không yêu cầu mọi thứ phải sẵn sàng (xác định) ở lúc dịch mà JVM có thể nạp các lớp (các *đối tượng*) cần thiết (*tệp .class*) để làm việc khi thực hiện chương trình ứng dụng. Thông thường, mỗi lớp trong chương trình sẽ được dịch sang một tệp có đuôi (phần mở rộng) là *.class*. Theo cách tổ chức của Java