LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

PGS.TS. Trần Đình Quế
KS. Nguyễn Mạnh Hùng

Các khái niệm cơ bản của Lập trình hướng đối tượng
Lập trình hướng đối tượng với Java

GIỚI THIỆU

Trong những năm gần đây, lập trình hướng đối tượng đã trở nên gần gũi nhờ sự ra đời liên tiếp của các ngôn ngữ lập trình hướng đối tượng. Sức mạnh của phương pháp lập trình hướng đối tượng thể hiện ở chỗ khả năng mô hình hoá hệ thống dựa trên các đối tượng thực tế, khả năng đóng gói và bảo vệ an toàn dữ liệu, khả năng sử dụng lại mã nguồn để tiết kiệm chi phí và tài nguyên; đặc biệt là khả năng chia sẽ mã nguồn trong cộng đồng lập trình viên chuyên nghiệp. Những điểm mạnh này hứa hẹn sẽ thúc đẩy phát triển một môi trường lập trình tiên tiến cùng với nền công nghiệp lắp ráp phần mềm với các thư viện thành phần có sẵn.

Tài liệu này nhằm giới thiệu cho các sinh viên một cái nhìn tổng quan về phương pháp lập trình hướng đối tượng cùng cung cấp những kiến thức, các kỹ thuật cơ bản cho phát triển các ứng dụng của mình dựa trên ngôn ngữ lập trình Java - một trong những ngôn ngữ lập trình hướng đối tượng thông dụng nhất hiện nay.

Nội dung của tài liệu này bao gồm hai phần chính:

- Phần thứ nhất trình bày những khái niệm và các vấn đề cơ bản của lập trình hướng đối tượng bao gồm tổng quan về cách tiếp cận hướng đối tượng và các khái niệm đối tượng, lớp, kế thừa, đóng gói, đa hình…
- Phần thứ hai trình bày chi tiết phương pháp lập trình hướng đối tượng với ngôn ngữ lập trình Java.

Nội dung của tài liệu bao gồm 6 chương:

Chương 1: Tổng quan về cách tiếp cận hướng đối tượng. Trình bày sự tiến hoá của cách tiếp cận từ lập trình truyền thống đến cách tiếp cận của lập trình hướng đối tượng và xu hướng phát triển của lập trình hướng đối tượng hiện nay.

Chương 2: Những khái niệm cơ bản của lập trình hướng đối tượng. Trình bày các khái niệm cơ bản như: đối tượng, lớp đối tượng với các thuộc tính và phương thức, tính kế thừa và đa hình, tính đóng gói của lập trình hướng đối tượng. Chương này cũng giới thiệu tổng quan một số ngôn ngữ lập trình hướng đối tượng thông dụng hiện nay.

Chương 3: Ngôn ngữ Java. Giới thiệu những khái niệm và những quy ước ban đầu của ngôn ngữ lập trình Java: Cấu trúc chương trình, cách biên dịch, cách đặt tên biến, kiểu dữ liệu, các toán tử và cấu trúc lệnh của ngôn ngữ Java.

Chương 4: Kế thừa và đa hình trên Java. Trình bày các kỹ thuật lập trình hướng đối tượng dựa trên ngôn ngữ Java: Khai báo lớp, các thuộc tính và phương thức của lớp; kỹ thuật thừa kế, các lớp trừu tượng, cài đặt nạp chồng và đa hình trên Java.

Chương 5: Biểu diễn và cài đặt các cấu trúc dữ liệu trừu tượng trên Java. Trình bày kỹ thuật cài đặt và sử dụng một số cấu trúc dữ liệu quen thuộc trong Java: ngăn xếp, hàng đợi, danh sách liên kết, cây nhị phân và đồ thị.

Chương 6: Lập trình giao diện trên Java. Trình bày các kỹ thuật lập trình giao diện trên Java: Lập trình với các giao diện cơ bản trong thư viện AWT, lập trình giao diện với Applet và HTML, lập trình giao diện nâng cao với thư viện SWING.

Tài liệu này được viết nhằm phục vụ môn học "**Lập trình hướng đối tượng**" giảng dạy tiếp theo sau môn học Ngôn ngữ lập trình C++ và như vậy khi học môn học này sinh viên sẽ dễ nắm bắt được những đặc trưng khác biệt của ngôn ngữ Java so với C++.

Cuốn sách này còn có kèm theo một đĩa CD chứa toàn bộ mã các chương trình cài đặt làm ví dụ và bài tập trong cuốn sách.

Mặc dù các tác giả đã có nhiều cố gắng trong quá trình biên soạn tài liệu này, song không thể tránh khỏi những thiếu sót. Rất mong nhận được sự đóng góp ý kiến của sinh viên và các bạn đồng nghiệp.

PHẦN 1

NHỮNG KHÁI NIỆM CƠ BẢN CỦA LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

CHƯƠNG 1
TỔNG QUAN VỀ CÁCH TIẾP CẬN HƯỚNG ĐỐI TƯỢNG

Nội dung chương này nhằm giới thiệu một cách tổng quan về cách tiếp cận hướng đối tượng. Nội dung trình bày bao gồm:

- Giới thiệu về cách tiếp cận của lập trình truyền thống.
- Giới thiệu cách tiếp cận của lập trình hướng đối tượng.
- So sánh sự khác biệt giữa hai cách tiếp cận này.
- Xu hướng hiện nay của lập trình hướng đối tượng

1.1 PHƯƠNG PHÁP TIẾP CẬN CỦA LẬP TRÌNH TRUYỀN THỐNG

Lập trình truyền thống đã trải qua hai giai đoạn:

- Giai đoạn sơ khai, khi khái niệm lập trình mới ra đời, là lập trình tuyến tính.
- Giai đoạn tiếp theo, là lập trình hướng cấu trúc.

1.1.1 Lập trình tuyến tính

Đặc trưng cơ bản của lập trình tuyến tính là tư duy theo lối tuần tự. Chương trình sẽ được thực hiện tuần tự từ đầu đến cuối, lệnh này kế tiếp lệnh kia cho đến khi kết thúc chương trình.

Đặc trưng

Lập trình tuyến tính có hai đặc trưng:

- **Đơn giản**: chương trình được tiến hành đơn giản theo lối tuần tự, không phức tạp.
- **Đơn luồng**: chỉ có một luồng công việc duy nhất, và các công việc được thực hiện tuần tự trong luồng đó.

Tính chất

- **Ưu điểm**: Do tính đơn giản, lập trình tuyến tính có ưu điểm là chương trình đơn giản, dễ hiểu. Lập trình tuyến tính được ứng dụng cho các chương trình đơn giản.
- **Nhược điểm**: Với các ứng dụng phức tạp, người ta không thể dùng lập trình tuyến tính để giải quyết.

Ngày nay, lập trình tuyến tính chỉ tồn tại trong phạm vi các modul nhỏ nhất của các phương pháp lập trình khác. Ví dụ trong một chương trình con của lập trình cấu trúc, các lệnh cũng được thực hiện theo tuần tự từ đầu đến cuối chương trình con.

1.1.2 Lập trình cấu trúc

Trong lập trình hướng cấu trúc, chương trình chính được chia nhỏ thành các chương trình con và mỗi chương trình con thực hiện một công việc xác định. Chương trình chính sẽ gọi đến chương trình con theo một giải thuật, hoặc một cấu trúc được xác định trong chương trình chính.

Các ngôn ngữ lập trình cấu trúc phổ biến là Pascal, C và C++. Riêng C++ ngoài việc có đặc trưng của lập trình cấu trúc do kế thừa từ C, còn có đặc trưng của lập trình hướng đối tượng. Cho nên C++ còn được gọi là ngôn ngữ lập trình nửa cấu trúc, nửa hướng đối tượng.

Đặc trưng

Đặc trưng cơ bản nhất của lập trình cấu trúc thể hiện ở mối quan hệ:

Chương trình = Cấu trúc dữ liệu + Giải thuật

Trong đó:

- **Cấu trúc dữ liệu** là cách tổ chức dữ liệu, cách mô tả bài toán dưới dạng ngôn ngữ lập trình
- **Giải thuật** là một quy trình để thực hiện một công việc xác định

Trong chương trình, giải thuật có quan hệ phụ thuộc vào cấu trúc dữ liệu:

- Một cấu trúc dữ liệu chỉ phù hợp với một số hạn chế các giải thuật.
- Nếu thay đổi cấu trúc dữ liệu thì phải thay đổi giải thuật cho phù hợp.
- Một giải thuật thường phải đi kèm với một cấu trúc dữ liệu nhất định.

Tính chất

- Mỗi chương trình con có thể được gọi thực hiện nhiều lần trong một chương trình chính.
- Các chương trình con có thể được gọi đến để thực hiện theo một thứ tự bất kì, tuỳ thuộc vào giải thuật trong chương trình chính mà không phụ thuộc vào thứ tự khai báo của các chương trình con.
- Các ngôn ngữ lập trình cấu trúc cung cấp một số cấu trúc lệnh điều khiển chương trình.

Ưu điểm

- Chương trình sáng sủa, dễ hiểu, dễ theo dõi.
- Tư duy giải thuật rõ ràng.

Nhược điểm

- Lập trình cấu trúc không hỗ trợ việc sử dụng lại mã nguồn: Giải thuật luôn phụ thuộc chặt chẽ vào cấu trúc dữ liệu, do đó, khi thay đổi cấu trúc dữ liệu, phải thay đổi giải thuật, nghĩa là phải viết lại chương trình.
- Không phù hợp với các phần mềm lớn: tư duy cấu trúc với các giải thuật chỉ phù hợp với các bài toán nhỏ, nằm trong phạm vi một modul của chương trình. Với dự án phần mềm lớn, lập trình cấu trúc tỏ ra không hiệu quả trong việc giải quyết mối quan hệ vĩ mô giữa các modul của phần mềm.

Vấn đề

Vấn đề cơ bản của lập trình cấu trúc là bằng cách nào để phân chia chương trình chính thành các chương trình con cho phù hợp với yêu cầu, chức năng và mục đích của mỗi bài toán.

Thông thường, để phân rã bài toán trong lập trình cấu trúc, người ta sử dụng phương pháp thiết kế trên xuống (top-down).

Phương pháp thiết kế trên xuống (top-down)

Phương pháp thiết kế top-down tiếp cận bài toán theo hướng từ trên xuống dưới, từ tổng quan đến chi tiết. Theo đó, một bài toán được chia thành các bài toán con nhỏ hơn. Mỗi bài toán con lại được chia nhỏ tiếp, nếu có thể, thành các bài toán con nhỏ hơn nữa.

Quá trình này còn được gọi là quá trình làm mịn dần. Quá trình làm mịn dần sẽ dừng lại khi các bài toán con không cần chia nhỏ thêm nữa. Nghĩa là khi mỗi bài toán con đều có thể giải quyết bằng một chương trình con với một giải thuật đơn giản.

Ví dụ, sử dụng phương pháp top-down để giải quyết bài toán là xây một căn nhà mới. Khi đó, ta có thể phân rã bài toán theo các bước như sau:

- Ở mức thứ nhất, chia bài toán xây nhà thành các bài toán nhỏ hơn như: làm móng, đổ cột, đổ trần, xây tường, lợp mái.
- Ở mức thứ hai, phân rã các công việc ở mức thứ nhất: việc làm móng nhà có thể phân rã tiếp thành các công việc: đào móng, gia cố nền, làm khung sắt, đổ bê tông. Công việc đổ cột được phần rã thành …
- Ở mức thứ ba, phân rã các công việc của mức thứ hai: việc đào móng có thể phân chia tiếp thành các công việc: đo đạc, cắm mốc, chăng dây, đào và kiểm tra móng. Việc gia cố nền được phân rã thành …

Quá trình phân rã có thể dừng ở mức này, bởi vì các công việc con thu được là: đo đạc, cắm mốc, chăng dây, đào… có thể thực hiện được ngay, không cần chia nhỏ thêm nữa.

Lưu ý:

- Cùng sử dụng phương pháp top-down với cùng một bài toán, nhưng có thể cho ra nhiều kết quả khác nhau. Nguyên nhân là do sự khác nhau trong tiêu chí để phân rã một bài toán thành các bài toán con.

Ví dụ, vẫn áp dụng phương pháp top-down để giải quyết bài toán xây nhà, nhưng nếu sử dụng một cách khác để phân chia bài toán, ta có thể thu được kết quả khác biệt so với phương pháp ban đầu:

- Ở mức thứ nhất, chia bài toán xây nhà thành các bài toán nhỏ hơn như: làm phần gỗ, làm phần sắt, làm phần bê tông và làm phần gạch.
- Ở mức thứ hai, phân rã các công việc ở mức thứ nhất: việc làm gỗ có thể chia thành các công việc như: xẻ gỗ, gia công gỗ, tạo khung, lắp vào nhà. Việc làm sắt có thể chia nhỏ thành…

Rõ ràng, với cách làm mịn thế này, ta sẽ thu được một kết quả khác hẳn với cách thức đã thực hiện ở phần trên.

1.2 PHƯƠNG PHÁP TIẾP CẬN HƯỚNG ĐỐI TƯỢNG

1.2.1 Phương pháp lập trình hướng đối tượng

Xuất phát từ hai hạn chế chính của phương pháp lập trình cấu trúc:

- Không quản lí được sự thay đổi dữ liệu khi có nhiều chương trình cùng thay đổi một biến chung. Vấn đề này đặc biệt nghiêm trọng khi các ứng dụng ngày càng lớn, người ta không thể kiểm soát được sự truy nhập đến các biến dữ liệu chung.

- Không tiết kiệm được tài nguyên con người: Giải thuật gắn liền với cấu trúc dữ liệu, nếu thay đổi cấu trúc dữ liệu, sẽ phải thay đổi giải thuật, và do đó, phải viết lại mã chương trình từ đầu.

Để khắc phục được hai hạn chế này khi giải quyết các bài toán lớn, người ta xây dựng một phương pháp tiếp cận mới, là phương pháp lập trình hướng đối tượng, với hai mục đích chính:

- Đóng gói dữ liệu để hạn chế sự truy nhập tự do vào dữ liệu, không quản lí được.
- Cho phép sử dụng lại mã nguồn, hạn chế việc phải viết lại mã từ đầu cho các chương trình.

Việc đóng gói dữ liệu được thực hiện theo phương pháp trừu tượng hoá đối tượng thành lớp từ thấp lên cao như sau:

- Thu thập các thuộc tính của mỗi đối tượng, gắn các thuộc tính vào đối tượng tương ứng.
- Nhóm các đối tượng có các thuộc tính tương tự nhau thành nhóm, loại bỏ bớt các thuộc tính cá biệt, chỉ giữ lại các thuộc tính chung nhất. Đây được gọi là quá trình trừu tượng hoá đối tượng thành lớp.
- Đóng gói dữ liệu của các đối tượng vào lớp tương ứng. Mỗi thuộc tính của đối tượng trở thành một thuộc tính của lớp tương ứng.
- Việc truy nhập dữ liệu được thực hiện thông qua các phương thức được trang bị cho lớp. Không được truy nhập tự do trực tiếp đến dữ liệu.
- Khi có thay đổi trong dữ liệu của đối tượng, ta chỉ cần thay đổi các phương thức truy nhập thuộc tính của lớp, mà không cần phải thay đổi mã nguồn của các chương trình sử dụng lớp tương ứng.

Việc cho phép sử dụng lại mã nguồn được thực hiện thông qua cơ chế kế thừa trong lập trình hướng đối tượng. Theo đó:

- Các lớp có thể được kế thừa nhau để tận dụng các thuộc tính, các phương thức của nhau.
- Trong lớp dẫn xuất (lớp được kế thừa) có thể sử dụng lại các phương thức của lớp cơ sở (lớp bị lớp khác kế thừa) mà không cần thiết phải cài đặt lại mã nguồn.
- Ngay cả khi lớp dẫn xuất định nghĩa lại các phương thức cho mình, lớp cơ sở cũng không bị ảnh hưởng và không phải sửa lại bất kì một đoạn mã nguồn nào.

Ngôn ngữ lập trình hướng đối tượng phổ biến hiện nay là Java và C++. Tuy nhiên, C++ mặc dù cũng có những đặc trưng cơ bản của lập trình hướng đối tượng nhưng vẫn không phải là ngôn ngữ lập trình thuần hướng đối tượng. Java thật sự là một ngôn ngữ lập trình thuần hướng đối tượng.

Đặc trưng

Lập trình hướng đối tượng có hai đặc trưng cơ bản:

- **Đóng gói dữ liệu**: dữ liệu luôn được tổ chức thành các thuộc tính của lớp đối tượng. Việc truy nhập đến dữ liệu phải thông qua các phương thức của đối tượng lớp.
- **Sử dụng lại mã nguồn**: việc sử dụng lại mã nguồn được thể hiện thông qua cơ chế kế thừa. Cơ chế này cho phép các lớp đối tượng có thể kế thừa từ các lớp đối tượng khác. Khi đó, trong các lớp kế thừa, có thể sử dụng các phương thức (mã nguồn) của các lớp bị kế thừa, mà không cần phải định nghĩa lại.

Ưu điểm

Lập trình hướng đối tượng có một số ưu điểm nổi bật:

- Không còn nguy cơ dữ liệu bị thay đổi tự do trong chương trình. Vì dữ liệu đã được đóng gói vào các đối tượng. Nếu muốn truy nhập vào dữ liệu phải thông qua các phương thức cho phép của đối tượng.
- Khi thay đổi cấu trúc dữ liệu của một đối tượng, không cần thay đổi các đổi mã nguồn của các đối tượng khác, mà chỉ cần thay đổi một số hàm thành phần của đối tượng bị thay đổi. Điều này hạn chế sự ảnh hưởng xấu của việc thay đổi dữ liệu đến các đối tượng khác trong chương trình.
- Có thể sử dụng lại mã nguồn, tiết kiệm tài nguyên. Vì nguyên tắc kế thừa cho phép các lớp kế thừa sử dụng các phương thức được kế thừa từ lớp khác như những phương thức của chính nó, mà không cần thiết phải định nghĩa lại.
- Phù hợp với các dự án phần mềm lớn, phức tạp.

1.2.2 Phương pháp phân tích và thiết kế hướng đối tượng

Một vấn đề cơ bản đặt ra cho phương pháp hướng đối tượng là từ một bài toán ban đầu, làm sao để thu được một tập các đối tượng, với các chức năng được phối hợp với nhau, đáp ứng được yêu cầu của bài toán đặt ra?

Phương pháp phân tích thiết kế hướng đối tượng ra đời nhằm trả lời cho câu hỏi này. Mục đích là xây dựng một tập các lớp đối tượng tương ứng với mỗi bài toán, phương pháp này tiến hành theo hai pha chính:

Pha phân tích: Chuyển đổi yêu cầu bài toán từ ngôn ngữ tự nhiên sang ngôn ngữ mô hình.

Pha thiết kế: Chuyển đổi đặc tả bài toán dưới dạng ngôn ngữ mô hình sang một mô hình cụ thể có thể cài đặt được.

Hai pha phân tích và thiết kế này bao gồm nhiều bước khác nhau:

- Mô tả bài toán
- Đặc tả yêu cầu
- Trích chọn đối tượng
- Mô hình hoá lớp đối tượng
- Thiết kế tổng quan
- Thiết kế chi tiết.

Bước 1: Mô tả bài toán

Bài toán ban đầu được phát biểu dưới dạng ngôn ngữ tự nhiên, bao gồm:

- Mục đích, chức năng chung
- Các yêu cầu về thông tin dữ liệu
- Các yêu cầu về chức năng thực hiện

Bước 2: Đặc tả yêu cầu

Các yêu cầu được hình thức hoá lên một mức cao hơn bằng cách sử dụng ngôn ngữ kiểu kịch bản (scenario) để mô tả. Trong một kịch bản, mỗi chức năng, mỗi hoạt động được mô tả bằng một kịch bản, bao gồm:

- Các tác nhân tham gia vào kịch bản.
- Vai trò của mỗi tác nhân trong kịch bản.
- Thứ tự các hành động mà mỗi tác nhân thực hiện: khi nào thực hiện, tác động vào tác nhân nào, thông tin nào được trao đổi.

Quá trình trên được tiến hành với tất cả các chức năng yêu cầu của hệ thống.

Bước 3: Trích chọn đối tượng

Bước này sẽ tiến hành đề xuất các đối tượng có thể có mặt trong hệ thống:

- Dựa vào các kịch bản được mô tả trong bước hai, chọn ra các tác nhân có xuất hiện để đề xuất thành các đối tượng.
- Lựa chọn các đối tượng bằng cách loại bỏ các tác nhân bên ngoài hệ thống, các tác nhân trùng lặp.
- Cuối cùng, ta thu được tập các đối tượng của hệ thống.

Bước 4: Mô hình hoá lớp đối tượng

Bước này tiến hành trừu tượng hoá đối tượng thành các lớp:

- Thu thập tất cả các thuộc tính của mỗi đối tương vừa thu thập được, dựa vào yêu cầu về thông tin trong yêu cầu hệ thống (từ bước 1).
- Thu thập các hành động mà mỗi đối tượng cần thực hiện, dựa vào các kịch bản mà đối tượng tương ứng có tham gia (trong bước 2).
- Nhóm các đối tượng tương tự nhau, hoặc có nhiều thuộc tính gần giống nhau.
- Loại bỏ một số thuộc tính cá biệt, riêng tư của một số đối tượng trong nhóm.
- Mô hình mỗi nhóm đối tượng còn lại thành lớp: Các thuộc tính chung của các đối tượng thành thuộc tính của lớp, các hành động của các đối tượng thành phương thức của lớp.

Kết quả thu được một tập các lớp đối tượng ban đầu của hệ thống.

Bước 5: Thiết kế tổng quát

Bước này sẽ tiến hành thiết kế vĩ mô, nghĩa là thiết kế mối quan hệ giữa các lớp trong hệ thống:

- Xác định sơ đồ thừa kế, nếu có, giữa các lớp: Nếu hai lớp có một số thuộc tính chung, thì tách các thuộc tính chung làm thành một lớp cơ sở, và hai lớp ban đầu đều dẫn xuất từ lớp cơ sở đó. Thông thường, lớp các trừu tượng (chung nhất) sẽ làm lớp cơ sở, lớp càng cụ thể, càng chi tiết thì làm lớp dẫn xuất (lớp con, cháu).
- Xác định tương tác, nếu có, giữa các lớp: Dựa vào các kịch bản được mô tả trong bước 2, hai tác nhân có tương tác với nhau thì hai lớp tương ứng ở bước này cũng có tương tác với nhau.

Kết quả thu được của bước này là một sơ đồ quan hệ bên ngoài giữa các lớp trong hệ thống.

Bước 6: Thiết kế chi tiết

Bước này sẽ thực hiện thiết kế ở mức vi mô, nghĩa là thiết kế kiến trúc bên trong của mỗi lớp đối tượng:

- Tổ chức dữ liệu của lớp theo các thuộc tính. Qui định phạm vi truy nhập cho từng thuộc tính.
- Thiết kế chi tiết cách cư xử của lớp đối tượng thông qua các phương thức của lớp: Xác định kiểu dữ liệu trả về, kiểu tham số của phương thức, mô tả thuật toán chi tiết cho từng phương thức, nếu cần.

Kết quả thu được của bước này là một tập các lớp với thiết kế chi tiết kiến trúc bên trong.

Sau các bước phân tích thiết kế hướng đối tượng từ một yêu cầu của bài toán ban đầu, ta thu được một mô hình hệ thống hướng đối tượng chi tiết:

- Có cái nhìn tổng quan, vĩ mô về hệ thống bằng mô hình thiết kế tổng quan, chỉ rõ số lượng các lớp đối tượng, mối quan hệ kế thừa và quan hệ tương tác giữa các lớp đối tượng trong hệ thống.
- Có cái nhìn chi tiết, vi mô về hệ thống bằng mô hình thiết kế chi tiết. Mô hình này chỉ rõ bên trong mỗi lớp đối tương: các thuộc tính, các phương thức với kiểu trả về và kiểu tham số, thuật toán chi tiết cho mỗi phương thức.

Sau pha phân tích và thiết kế hướng đối tượng, ta thu được đặc tả hệ thống dưới dạng mô hình các lớp: quan hệ giữa các lớp và kiến trúc bên trong của mỗi lớp. Đây sẽ là đầu vào cho pha tiếp theo, pha lập trình hướng đối tượng, như chúng ta đã biết.

1.3 SO SÁNH HAI CÁCH TIẾP CẬN

Phương pháp tiếp cận hướng đối tượng có bản chất hoàn toàn khác với phương pháp tiếp cận truyền thống (phương pháp tiếp cận hướng cấu trúc) trên nhiều mặt:

- Phương pháp mô hình bài toán khác nhau.
- Đặc trưng khác nhau về đóng gói
- Ưu / nhược điểm khác nhau.
- Lĩnh vực ứng dụng khác nhau.

Khác nhau về phương pháp mô hình

Hai phương pháp này khác nhau hoàn toàn ở cách tiếp cận và mô hình bài toán, phương pháp hướng đối tượng tiến hành theo phương pháp từ dưới lên trên, từ thấp lên cao, từ cụ thể đến trừu tượng. Trong khi đó, phương pháp cấu trúc tiếp cận theo phương pháp từ trên xuống dưới, từ tổng quan đến chi tiết:

- Phương pháp hướng đối tượng bắt đầu bằng những đối tượng cụ thể, tập hợp các thuộc tính của từng đối tượng. Sau đó, nhóm các đối tượng tương tự nhau thành nhóm, loại bỏ các thuộc tính quá cá biệt, chỉ giữ lại các thuộc tính chung nhất, nhóm thành lớp. Cho nên, quá trình hình thành lớp là quá trình đi từ thấp lên cao, từ cụ thể ở mức thấp đến trừu tượng hoá ở mức cao.

- Trong khi đó, phương pháp hướng cấu trúc lại đi theo chiều ngược lại. Phương pháp này bắt đầu từ một bài toán tổng quan, ở mức khái quát cao, chia nhỏ dần và làm mịn dần cho đến khi thu được một tập các bài toán con, nhỏ hơn, cụ thể hơn, chi tiết hơn.

Khác nhau về đặc trưng đóng gói

Hai phương pháp tiếp cận này cũng có những đặc trưng hoàn toàn khác nhau:

- Phương pháp hướng đối tượng có đặc trưng là dữ liệu được đóng gói để hạn chế truy nhập tự do trực tiếp vào dữ liệu. Thứ hai là cho phép sử dụng lại mã nguồn để tiết kiệm tài nguyên và công sức lập trình.
- Trong khi đó, đặc trưng của phương pháp cấu trúc là cấu trúc dữ liệu và giải thuật và mối quan hệ phụ thuộc chặt nhẽ của giải thuật vào cấu trúc dữ liệu.

Khác nhau về ưu nhược điểm

Hai phương pháp này cũng có những ưu nhược điểm trái ngược nhau:

- Phương pháp hướng đối tượng có ưu điểm là bảo vệ được dữ liệu tránh bị truy nhập trực tiếp tự do từ bên ngoài, tiết kiệm được tài nguyên và công sức lập trình do có thể dùng lại mã nguồn. Tuy nhiên, phương pháp này lại khá phức tạp, khó theo dõi được luồng dữ liệu và hơn nữa, giải thuật không phải là vấn đề trọng tâm của phương pháp này.
- Trái lại, phương pháp hướng cấu trúc lại có ưu điểm là tư duy giải thuật rõ ràng, dễ theo dõi luồng dữ liệu, chương trình đơn giản và dễ hiểu. Tuy nhiên, không bảo về được an toàn dữ liệu trong chương trình. Hơn nữa, hạn chế lớn nhất là sự phụ thuộc chặt chẽ của giải thuật vào cấu trúc dữ liệu, khiến cho khi thay đổi cấu trúc dữ liệu, thường phải thay đổi giải thuật, và do đó, phải viết lại mã cho chương trình.

Khác nhau về lĩnh vực áp dụng

Do sự khác nhau về các đặc trưng và sự khác nhau về ưu nhược điểm, cho nên hai phương pháp này cũng có sự khác nhau đáng kể trong lĩnh vực áp dụng:

- Phương pháp hướng đối tượng thường được áp dụng cho các bài toán lớn, phức tạp, có nhiều luồng dữ liệu khác nhau, không thể quản lí được bằng phương pháp cấu trúc. Khi đó, người ta dùng phương pháp hướng đối tượng để tận dụng khả năng bảo vệ dữ liệu tránh bị truy nhập tự do. Hơn nữa, tận dụng khả năng dùng lại mã nguồn của phương pháp này để tiết kiệm tài nguyên và công sức.
- Trong khi đó, phương pháp cấu trúc thường phù hợp với các bài toán nhỏ, có luồng dữ liệu rõ ràng, cần phải tư duy giải thuật rõ ràng và người lập trình vẫn có khả năng tự quản lí được mọi truy nhập đến các dữ liệu của chương trình.

1.4 XU HƯỚNG PHÁT TRIỂN CỦA LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Lập trình hướng thành phần

Xuất phát từ lập trình hướng đối tượng, tư duy lập trình hướng thành phần (component-oriented programming) theo ý tưởng:

- Giải quyết bài toán bằng cách xây dựng một tập các thành phần (component) có tính độc lập tương đối với nhau. Mỗi thành phần đảm nhiệm một phần công việc nhất định.

- Sau đó, người ta ghép các thành phần với nhau để thu được một phần mềm thoả mãn một tập các yêu cầu xác định.

Với lập trình hướng thành phần, người ta có thể tiến hành lập trình theo phương pháp sau:

- Xây dựng một thư viện các thành phần, mỗi thành phần thực hiện một công việc xác định.
- Khi cần phát triển một phần mềm cụ thể, người ta chỉ cần chọn những thành phần có sẵn trong thư viện để ghép lại với nhau. Người lập trình chỉ phải phát triển thêm các thành phần mình cần mà chưa có trong thư viện.

Phương pháp này có những ưu điểm rất lớn:

- Lập trình viên có thể chia sẻ với nhau những thành phần mình đã xây dựng cho nhiều người khác dùng chung.
- Khi cần, lập trình viên có thể lắp ghép các thành phần có sẵn khác nhau để tạo thành các chương trình có chức năng khác nhau. Tất cả chỉ cần dựa trên công nghệ lắp ghép thành phần, tiết kiệm được rất nhiều công sức lập trình.

Trong xu hướng lập trình hướng thành phần, một số phương pháp lập trình khác đã nảy sinh và đang phát triển mạnh mẽ:

- Lập trình hướng agent (agent-oriented programming)
- Lập trình hướng aspect (aspect-oriented programming)

Lập trình hướng agent

Lập trình hướng agent có thể xem là một mức trừu tượng cao hơn của lập trình hướng thành phần. Trong đó, các agent là các thành phần có khả năng hoạt động độc lập, tự chủ để hoàn thành công việc của mình. Hơn nữa, các agent có khả năng chủ động liên lạc với các agent khác để có thể phối hợp, cộng tác hay cạnh tranh nhau để hoàn thành nhiệm vụ.

Lập trình hướng agent có hai đặc trưng cơ bản:

- Thứ nhất là khả năng tự chủ của mỗi agent để hoàn thành nhiệm vụ riêng của nó.
- Thứ hai là tính tổ chức xã hội giữa các agent, cho phép các agent phối hợp, cộng tác, cạnh tranh nhau để hoàn thành nhiệm vụ chung của toàn hệ thống.

Lập trình hướng aspect

Lập trình hướng aspect cũng là một xu hướng của lập trình hướng thành phần. Theo đó, mỗi thành phần có nhiệm vụ hoàn thành theo một luồng công việc hoặc một khía cạnh của vấn đề. Sau đó, tổng hợp các thành phần của các luồng khác nhau, ta thu được giải pháp cho bài toán của mình.

Lập trình hướng aspect có đặc trưng cơ bản:

- Tính đóng gói theo luồng công việc, hoặc đóng gói theo khía cạnh của vấn đề.
- Tính đơn điệu theo luồng, trong một luồng công việc, các nhiệm vụ được thực hiện liên tiếp nhau, tuần tự như trong lập trình tuyến tính.

TỔNG KẾT CHƯƠNG 1

Nội dung chương 1 đã trình bày các vấn đề tổng quan liên quan đến phương pháp tiếp cận hướng đối tượng trong lập trình:

- Các phương pháp tiếp cận truyền thống: lập trình tuyến tính và lập trình cấu trúc.

- Phương pháp tiếp cận hướng đối tượng với hai đặc trưng cơ bản: Đóng gói dữ liệu và sử dụng lại mã nguồn.
- Lập trình hướng đối tượng, phương pháp phân tích và thiết kế hệ thống hướng đối tượng.
- So sánh sự khác biệt của phương pháp hướng đối tượng với các phương pháp truyền thống trên các khía cạnh: Cách tiếp cận bài toán, đặc trưng, ưu nhược điểm và lĩnh vực áp dụng của mỗi phương pháp.
- Hiện nay, lập trình hướng thành phần, lập trình hướng agent và lập trình hướng aspect tiến hoá từ lập trình hướng đối tượng đang là xu hướng phát triển mạnh mẽ.

CHƯƠNG 2
NHỮNG KHÁI NIỆM CƠ BẢN CỦA LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Nội dung chương này tập trung trình bày các khái niệm cơ bản của lập trình hướng đối tượng:

- Đối tượng
- Lớp đối tượng
- Việc trừu tượng hoá đối tượng theo chức năng
- Trừu tượng hoá đối tượng theo dữ liệu
- Kế thừa
- Đóng gói
- Đa hình
- Phương pháp cài đặt lớp đối tượng
- Giới thiệu một số ngôn ngữ lập trình hướng đối tượng thông dụng hiện nay.

2.1 CÁC KHÁI NIỆM CƠ BẢN

2.1.1 Đối tượng

Trong lập trình hướng đối tượng, tất cả các thực thể trong hệ thống đều được coi là các đối tượng cụ thể. Đối tượng là một thực thể hoạt động khi chương trình đang chạy.

Ví dụ:

1. Trong bài toán quản lí buôn bán xe hơi của một cửa hàng kinh doanh, mỗi chiếc xe đang có mặt trong cửa hàng được coi là một đối tượng. Chẳng hạn, một chiếc xe nhãn hiệu "Ford", màu trắng, giá 5000$ là một đối tượng.
2. Trong bài toán quản lí nhân viên của một văn phòng, mỗi nhân viên trong văn phòng được coi là một đối tượng. Chẳng hạn, nhân viên tên là "Vinh", 25 tuổi làm ở phòng hành chính là một đối tượng.

Một đối tượng là một thực thể đang tồn tại trong hệ thống và được xác định bằng ba yếu tố:

- Định danh đối tượng: xác định duy nhất cho mỗi đối tượng trong hệ thống, nhằm phân biệt các đối tượng với nhau.
- Trạng thái của đối tượng: là sự tổ hợp của các giá trị của các thuộc tính mà đối tượng đang có.
- Hoạt động của đối tượng: là các hành động mà đối tượng có khả năng thực hiện được.

Trạng thái hiện tại của đối tượng qui định tính chất đặc trưng của đối tượng. Ví dụ, nhân viên trong ví dụ trên có trạng thái là:

- Tên là Vinh
- Tuổi là 25
- Vị trí làm việc là phòng hành chính.

Trong khi đó, trạng thái của chiếc xe trong cửa hàng là:

- Nhãn hiệu xe là Ford
- Màu sơn xe là trắng
- Giá bán xe là 5000$

Mỗi đối tượng sẽ thực hiện một số hành động. Ví dụ, đối tượng xe hơi có khả năng thực hiện những hành động sau:

- Khởi động.
- Dừng lại.
- Chạy.

Để biểu diễn đối tượng trong lập trình hướng đối tượng, người ta trừu tượng hoá đối tượng để tạo nên khái niệm lớp đối tượng.

2.1.2 Lớp đối tượng

Trong lập trình hướng đối tượng, đối tượng là một thực thể cụ thể, tồn tại trong hệ thống. Trong khi đó, lớp là một khái niệm trừu tượng, dùng để chỉ một tập hợp các đối tượng có mặt trong hệ thống.

Ví dụ:

1. Trong bài toán quản lí buôn bán xe hơi của một cửa hàng kinh doanh, mỗi chiếc xe đang có mặt trong cửa hàng được coi là một đối tượng. Nhưng khái niệm "Xe hơi" là một lớp đối tượng dùng để chỉ tất cả các loại xe hơi của của hàng.
2. Trong bài toán quản lí nhân viên của một văn phòng, mỗi nhân viên trong văn phòng được coi là một đối tượng. Nhưng khái niệm "Nhân viên" là một lớp đối tượng dùng để chỉ chung chung các nhân viên của văn phòng.

Lưu ý:

- Lớp là một khái niệm, mang tính trừu tượng, dùng để biểu diễn một tập các đối tượng.
- Đối tượng là một thể hiện cụ thể của lớp, là một thực thể tồn tại trong hệ thống.

Lớp được dùng để biểu diễn đối tượng, cho nên lớp cũng có thuộc tính và phương thức:

- Thuộc tính của lớp tương ứng với thuộc tính của các đối tượng.
- Phương thức của lớp tương ứng với các hành động của đối tượng.

Ví dụ, lớp xe ô tô được mô tả bằng các thuộc tính và phương thức:

Lớp Xe ô tô

Thuộc tính:

Nhãn hiệu xe
Màu xe
Giá xe
Công suất xe (mã lực)

Phương thức:

Khởi động xe
Chạy xe

```
Dừng xe
Tắt máy
```

Lưu ý:

Một lớp có thể có một trong các khả năng sau:

- Hoặc chỉ có thuộc tính, không có phương thức.
- Hoặc chỉ có phương thức, không có thuộc tính.
- Hoặc có cả thuộc tính và phương thức, trường hợp này là phổ biến nhất.
- Đặc biệt, lớp không có thuộc tính và phương thức nào là các lớp trừu tượng. Các lớp này không có đối tượng tương ứng.

Lớp và Đối tượng

Lớp và đối tượng, mặc dù có mối liên hệ tương ứng lẫn nhau, nhưng bản chất lại khác nhau:

- Lớp là sự trừu tượng hoá của các đối tượng. Trong khi đó, đối tượng là một thể hiện của lớp.
- Đối tượng là một thực thể cụ thể, có thực, tồn tại trong hệ thống. Trong khi đó, lớp là một khái niệm trừu tượng, chỉ tồn tại ở dạng khái niệm để mô tả các đặc tính chung của một số đối tượng.
- Tất cả các đối tượng thuộc về cùng một lớp có cùng các thuộc tính và các phương thức.
- Một lớp là một nguyên mẫu của một đối tượng. Nó xác định các hành động khả thi và các thuộc tính cần thiết cho một nhóm các đối tượng cụ thể.

Nói chung, lớp là khái niệm tồn tại khi phát triển hệ thống, mang tính khái niệm, trừu tượng. Trong khi đó, đối tượng là một thực thể cụ thể tồn tại khi hệ thống đang hoạt động.

2.1.3 Trừu tượng hoá đối tượng theo chức năng

Trừu tượng hoá đối tượng theo chức năng chính là quá trình mô hình hoá phương thức của lớp dựa trên các hành động của các đối tượng. Quá trình này được tiến hành như sau:

- Tập hợp tất cả các hành động có thể có của các đối tượng.
- Nhóm các đối tượng có các hoạt động tương tự nhau, loại bỏ bớt các hoạt động cá biệt, tạo thành một nhóm chung.
- Mỗi nhóm đối tượng đề xuất một lớp tương ứng.
- Các hành động chung của nhóm đối tượng sẽ cấu thành các phương thức của lớp tương ứng.

Ví dụ, trong bài toán quản lí cửa hàng bán ô tô. Mỗi ô tô có mặt trong của hàng là một đối tượng. Mặc dù mỗi chiếc xe có một số đặc điểm khác nhau về nhãn hiệu, giá xe, màu sắc… nhưng có chung các hành động của một chiếc xe ô tô là:

- Có thể khởi động máy.
- Có thể chạy.
- Có thể dừng lại.
- Có thể tắt máy.

Ngoài ra, một số ít xe có thể thực hiện một số hành động cá biệt như:

- Có thể giấu đèn pha
- Có thể tự bật đèn pha
- Có thể tự động phát tín hiệu báo động.

Tuy nhiên, không phải xe nào cũng thực hiện được các hành động này. Cho nên ta loại bỏ các hành động cá biệt của một số xe, chỉ giữ lại các hành động chung nhất, để mô hình thành các phương thức của đối tượng xe ô tô tương ứng với các hành động chung nhất của các xe ô tô.

Lớp Xe ô tô

Phương thức:

```
Khởi động xe
Chạy xe
Dừng xe
Tắt máy
```

2.1.4 Trừu tượng hoá đối tượng theo dữ liệu

Trừu tượng hoá đối tượng theo dữ liệu chính là quá trình mô hình hoá các thuộc tính của lớp dựa trên các thuộc tính của các đối tượng tương ứng. Quá trình này được tiến hành như sau:

- Tập hợp tất cả các thuộc tính có thể có của các đối tượng.
- Nhóm các đối tượng có các thuộc tính tương tự nhau, loại bỏ bớt các thuộc tính cá biệt, tạo thành một nhóm chung.
- Mỗi nhóm đối tượng đề xuất một lớp tương ứng.
- Các thuộc tính chung của nhóm đối tượng sẽ cấu thành các thuộc tính tương ứng của lớp được đề xuất.

Ví dụ, trong bài toán quản lí cửa hàng bán ô tô. Mỗi ô tô có mặt trong của hàng là một đối tượng. Mặc dù mỗi chiếc xe có một số đặc điểm khác nhau về nhãn hiệu, giá xe, màu sắc… nhưng có chung các thuộc tính của một chiếc xe ô tô là:

- Các xe đều có nhãn hiệu.
- Các xe đều có màu sắc
- Các xe đều có giá bán
- Các xe đều có công suất động cơ

Ngoài ra, một số ít xe có thể có thêm các thuộc tính:

- Có xe có thể có dàn nghe nhạc
- Có xe có thể có màn hình xem ti vi
- Có xe có lắp kính chống nắng, chống đạn…

Tuy nhiên, đây là các thuộc tính cá biệt của một số đối tượng xe, nên không được đề xuất thành thuộc tính của lớp ô tô. Do đó, ta mô hình lớp ô tô với các thuộc tính chung nhất của các ô tô.

Lớp Xe ô tô
Thuộc tính:
Nhãn hiệu xe
Màu xe
Giá xe
Công suất xe (mã lực)

Ưu điểm của việc trừu tượng hóa

Những ưu điểm của việc trừu tượng hóa là:

- Tập trung vào vấn đề cần quan tâm
- Xác định những đặc tính thiết yếu và những hành động cần thiết
- Giảm thiểu những chi tiết không cần thiết

Việc trừu tượng hóa dữ liệu là cần thiết, bởi vì không thể mô tả tất cả các hành động và các thuộc tính của một thực thể. Vấn đề mấu chốt là tập trung đến những hành vi cốt yếu và áp dụng chúng trong ứng dụng.

2.1.5 Khái niệm kế thừa

Xét trường hợp bài toán quản lí nhân sự và sinh viên của một trường đại học. Khi đó, ta có hai lớp đối tượng chính là lớp Nhân viên và lớp Sinh viên:

Lớp Nhân viên	*Lớp Sinh viên*
Thuộc tính:	**Thuộc tính:**
Tên	Tên
Ngày sinh	Ngày sinh
Giới tính	Giới tính
Lương	Lớp
Phương thức:	**Phương thức:**
Nhập/xem tên	Nhập/xem tên
Nhập/xem ngày sinh	Nhập/xem ngày sinh
Nhập/xem giới tính	Nhập/xem giới tính
Nhập/xem lương	Nhập/xem lớp

Ta nhận thấy rằng hai lớp này có một số thuộc tính và phương thức chung: tên, ngày sinh, giới tính. Tuy nhiên, không thể loại bỏ các thuộc tính cá biệt để gộp chúng thành một lớp duy nhất, vì các thuộc tính lương nhân viên và lớp của sinh viên là cần thiết cho việc quản lí. Vấn đề nảy sinh như sau:

- Ta phải viết mã trùng nhau đến hai lần cho các phương thức: nhập/xem tên, nhập/xem ngày sinh, nhập/xem giới tính. Rõ rang điều này rất tốn công sức.
- Nếu khi có sự thay đổi về kiểu dữ liệu, chẳng hạn kiểu ngày sinh được quản lí trong hệ thống, ta phải sửa lại chương trình hai lần.

Để tránh rắc rối do các vấn đề nảy sinh như vậy, lập trình hướng đối tượng sử dụng kỹ thuật kế thừa nhằm nhóm các phần giống nhau của các lớp thành một lớp mới, sau đó cho các lớp ban đầu kế thừa lại lớp được tạo ra. Như vậy, mỗi lớp thừa kế (lớp dẫn xuất, lớp con) đều có các thuộc tính và phương thức của lớp bị thừa kế (lớp cơ sở, lớp cha).

Quay lại với bài toán quản lí trường đại học, các thuộc tính và phương thức chung giữa lớp Nhân viên và lớp Sinh viên là:

- Tên,
- Ngày sinh,
- Giới tính,
- Nhập/xem tên,
- Nhập/xem ngày sinh
- Nhập/xem giới tính.

Ta tách phần chung này thành một lớp ở mức trừu tượng cao hơn, lớp Người. Lớp Người sẽ làm lớp cha của lớp Nhân viên và lớp Sinh viên. Khi đó, các lớp trở thành:

<table>
<tr><td colspan="2">Lớp Người
Thuộc tính:
Tên
Ngày sinh
Giới tính
Phương thức:
Nhập/xem tên
Nhập/xem ngày sinh
Nhập/xem giới tính</td></tr>
<tr><td>Lớp Nhân viên kế thừa từ lớp Người
Thuộc tính:
Lương
Phương thức:
Nhập/xem lương</td><td>Lớp Sinh viên kế thừa từ lớp Người
Thuộc tính:
Lớp
Phương thức:
Nhập/xem lớp</td></tr>
</table>

Như vậy, sự kế thừa trong lập trình hướng đối tượng:

- Cho phép lớp dẫn xuất có thể sử dụng các thuộc tính và phương thức của lớp cơ sở tương tự như sử dụng các thuộc tính và phương thức của mình.
- Cho phép việc chỉ cần cài đặt phương thức ở một lớp cơ sở, mà có thể sử dụng được ở tất cả các lớp dẫn xuất.
- Cho phép tránh sự cài đặt trùng lặp mã nguồn của chương trình.
- Cho phép chỉ phải thay đổi một lần khi cần phải thay đổi dữ liệu của các lớp.

2.1.6 Khái niệm đóng gói

Xét ví dụ bài toán quản lí nhân viên văn phòng với lớp Nhân viên như sau:

Lớp Nhân viên

Thuộc tính:

Tên

Ngày sinh

Giới tính

Phòng ban

Hệ số lương

Phương thức:

Tính lương nhân viên

Khi đó, cách tính lương cho nhân viên là khác nhau đối với mỗi người:

```
<Tiền lương> = <Hệ số lương> * <Lương cơ bản> * <Tỉ lệ phần trăm>
```

Trong đó, tỉ lệ phần trăm là khác nhau cho mỗi phòng ban, ví dụ:

- Phòng kế hoạch là 105%
- Phòng hành chính là 100%
- Phòng nhân sự là 110%

Khi đó, tuỳ vào thuộc tính phòng ban khác nhau mà ta phải dùng công thức tỉ lệ khác nhau để tính lương cho mỗi nhân viên.

Tuy nhiên, cách tính cụ thể này là công việc bên trong của phương thức tính tiền lương của lớp Nhân viên. Với mỗi ứng dụng, khi tạo một đối tượng cụ thể của lớp nhân viên, ta chỉ cần truyền các tham số thuộc tính cho đối tượng, sau đó gọi phương thức tính tiền lương cho đối tượng nhân viên đó, ta sẽ biết được tiền lương của nhân viên. Cách gọi phương thức tính tiền lương là hoàn toàn giống nhau cho tất cả các đối tượng nhân viên của văn phòng.

Sự giống nhau về cách sử dụng phương thức cho các đối tượng của cùng một lớp, mặc dù bên trong phương thức có các cách tính toán khác nhau với các đối tương khác nhau, được gọi là tính đóng gói dữ liệu của lập trình hướng đối tượng. Như vậy, tính đóng gói dữ liệu của lập trình hướng đối tượng:

- Cho phép che dấu sự cài đặt chi tiết bên trong của phương thức. Khi sử dụng chỉ cần gọi các phương thức theo một cách thống nhất, mặc dù các phương thức có thể được cài đặt khác nhau cho các trường hợp khác nhau.
- Cho phép che dấu dữ liệu bên trong của đối tượng. Khi sử dụng, ta không biết được thực sự bên trong đối tượng có những gì, ta chỉ thấy được những gì đối tượng cho phép truy nhập vào.
- Cho phép hạn chế tối đa việc sửa lại mã chương trình. Khi phải thay đổi công thức tính toán của một phương thức, ta chỉ cần thay đổi mã bên trong của phương thức, mà không phải thay đổi các chương trình gọi đến phương thức bị thay đổi.

2.1.7 Khái niệm đa hình

Trở lại với ví dụ về quản lí trường đại học, với hai lớp Nhân viên và lớp Sinh viên, đều kế thừa từ lớp Người. Khi đó, ta thêm vào mỗi lớp một phương thức show():

- Phương thức show của lớp Người sẽ giới thiệu tên và tuổi của người đó.

- Phương thức show của lớp Nhân viên sẽ giới thiệu nhân viên đó có tiền lương là bao nhiêu
- Phương thức show của lớp Sinh viên sẽ giới thiệu là sinh viên đó đang học ở lớp nào.

<table>
<tr><td colspan="2">Lớp Người
Thuộc tính:
Tên
Ngày sinh
Giới tính
Phương thức:
Nhập/xem tên
Nhập/xem ngày sinh
Nhập/xem giới tính
show</td></tr>
<tr><td>Lớp Nhân viên kế thừa từ lớp Người
Thuộc tính:
Lương
Phương thức:
Nhập/xem lương
show</td><td>Lớp Sinh viên kế thừa từ lớp Người
Thuộc tính:
Lớp
Phương thức:
Nhập/xem lớp
show</td></tr>
</table>

Khi đó, nếu trong hệ thống có các đối tượng cụ thể tương ứng với ba lớp, thì:

- Khi ta gọi hàm show từ đối tượng của lớp Người, sẽ nhận được tên và tuổi của người đó.
- Khi ta gọi phương thức show từ đối tượng của lớp Nhân viên, sẽ nhận được số tiền lương của nhân viên đó.
- Khi ta gọi phương thức show từ đối tượng của lớp Sinh viên, ta sẽ biết được lớp học của sinh viên đó.

Việc chỉ cần gọi cùng một phương thức, nhưng từ các đối tượng khác nhau, sẽ cho kế quả khác nhau được gọi là tính đa hình trong lập trình hướng đối tượng. Như vậy, tính đa hình trong lập trình hướng đối tượng:

- Cho phép các lớp được định nghĩa các phương thức trùng nhau: cùng tên, cùng số lượng và kiểu tham số, cùng kiểu trả về. Việc định nghĩa phương thức trùng nhau của các lớp kế thừa nhau còn được gọi là sự nạp chồng phương thức.
- Khi gọi các phương thức trùng tên, dựa vào đối tượng đang gọi mà chương trình sẽ thực hiện phương thức của lớp tương ứng, và do đó, sẽ cho các kết quả khác nhau.

2.2 SO SÁNH LỚP VÀ CẤU TRÚC

Trong phần này, chúng ta sẽ tiến hành so sánh Class (Lớp) và Structure (Cấu trúc) trên nhiều khía cạnh khác nhau:

- Mức khái niệm
- Mục đích và chức năng
- Về ưu và nhược điểm

So sánh ở mức khái niệm

Ở mức khái niệm, Lớp và cấu trúc hoàn toàn khác nhau:

- Lớp là khái niệm chỉ có trong lập trình hướng đối tượng; nó được dùng để biểu diễn một tập các đối tượng tương tự nhau.
- Trong khi đó, Cấu trúc là khái niệm chỉ tồn tại trong lập trình cấu trúc, không phải là một khái niệm của lập trình hướng đối tượng. Vì trong lập trình hướng đối tượng, tất cả các thực thể đều được coi là một đối tượng, nghĩa là nó phải là một thể hiện cụ thể của một lớp nào đó. Do đó, trong lập trình hướng đối tượng, không có khái niệm Cấu trúc.

So sánh về mục đích và chức năng

Về mục đích, Lớp và Cấu trúc đều có chung một mục đích ban đầu, đó là nhóm một tập hợp các dữ liệu lại với nhau để xử lí đồng bộ và thống nhất: Cấu trúc nhóm các dữ liệu hay phải đi kèm với nhau lại thành một nhóm cho dễ xử lí. Tương tự, Lớp là tập hợp một số thuộc tính chung của đối tượng để xử lí.

Tuy nhiên, Lớp và Cấu trúc cũng có một số khác biệt trên khía cạnh này:

- Lớp ngoài mục đích nhóm các thuộc tính dữ liệu của đối tượng, còn nhóm các hoạt động của đối tượng thành các phương thức của Lớp.
- Trong khi đó, mặc dù cũng có thể cung cấp các hàm trong Cấu trúc, nhưng mục đích chính của Cấu trúc chỉ là nhóm dữ liệu thành cấu trúc cho dễ xử lí.

So sánh về ưu nhược điểm

Vì có cùng mục đích là nhóm các dữ liệu lại với nhau để xử lí, cho nên Lớp cà Cấu trúc có cùng ưu điểm là làm chương trình gọn gàng, xử lí đồng bộ và thống nhất.

Tuy nhiên, Lớp còn có một số ưu điểm mà Cấu trúc không có:

- Lớp có khả năng bảo vệ dữ liệu tránh bị truy nhập tự do từ bên ngoài. Các chương trình bên ngoài chỉ có thể truy nhập vào dữ liệu của đối tượng thông qua các phương thức do Lớp cung cấp, không thể tự do truy nhập. Trong khi đó, Cấu trúc mặc dầu đã nhóm dữ liệu với nhau nhưng không có khả năng bảo vệ dữ liệu: Các chương trình bên ngoài vẫn có thể truy nhập tự do vào các thành phần của Cấu trúc.
- Lớp có khả năng đóng gói để hạn chế tối đa thay đổi khi phải sửa lại mã chương trình. Khi có sự thay đổi, chỉ cần thay đổi mã của một phương thức, các chương trình bên ngoài sử dụng phương thức đó đều không phải thay đổi. Trong khi đó, nếu thay đổi một thành phần của Cấu trúc, ta phải thay đổi mã của tất cả các chương trình sử dụng thành phần đó của Cấu trúc.
- Lớp có thể được kế thừa bởi một Lớp khác, điều này làm tăng khả năng sử dụng lại mã nguồn của chương trình. Trong khi đó, Cấu trúc hoàn toàn không có cơ chế kế thừa, cho nên nhiều khi phải viết lại những đoạn mã giống nhau nhiều lần. Điều này vừa tốn công sức, vừa không an toàn khi có sự thay đổi một trong những đoạn mã giống nhau đó.

2.3 THÀNH PHẦN PRIVATE VÀ PUBLIC CỦA LỚP

Để bảo vệ dữ liệu tránh bị truy nhập tự do từ bên ngoài, lập trình hướng đối tượng sử dụng các từ khoá quy định phạm vi truy nhập các thuộc tính và phương thức của lớp. Một cách tổng quát, lập trình hướng đối tượng chia ra hai mức truy nhập các thành phần lớp:

- Private: Truy nhập trong nội bộ lớp.
- Protected: Thành phần được bảo vệ, được hạn chế truy nhập như thành phần private (sẽ được trình bày sau).
- Public: Truy nhập tự do từ bên ngoài.

Thành phần private

Thành phần private là khu vực dành riêng cho lớp, không chia sẻ với bất kì lớp khác từ bên ngoài. Thành phần private chỉ cho phép truy nhập trong phạm vi nội bộ lớp: Từ phương thức vào các thuộc tính hoặc giữa các phương thức của lớp với nhau. Các thành phần private không thể truy nhập từ bên ngoài lớp, cũng như từ đối tượng khác.

Trong một lớp, thông thường các thành phần sau sẽ được đặt vào khu vực private của lớp:

- Tất cả các thuộc tính dữ liệu của lớp. Các thuộc tính dữ liệu của lớp được đặt vào vùng private nhằm bảo vệ chúng, tránh sự truy nhập tự do từ bên ngoài.
- Các phương thức trung gian, được sử dụng như các bước tính toán đệm cho các phương thức khác. Các phương thức trung gian được đặt vào vùng private để thực hiện việc đóng gói trong lập trình hướng đối tượng: Các đối tượng, chương trình bên ngoài không cần, và không thể biết cách tính toán cụ thể bên trong của lớp.

Thành phần public

Thành phần public là khu vực mà Lớp có thể chia sẻ với tất cả các chương trình và đối tượng bên ngoài. Thành phần public có thể được truy nhập từ bên trong lẫn bên ngoài lớp:

- Bên trong lớp: từ phương thức lớp vào các thuộc tính dữ liệu của lớp, hoặc giữa các phương thức của lớp với nhau.
- Bên ngoài lớp: Từ chương trình bên ngoài hoặc các đối tượng khác vào các phương thức của lớp.

Trong một lớp, thông thường các thành phần sau sẽ được đặt vào vùng chia sẻ public của lớp:

- Các phương thức để nhập/xem (set/get) các thuộc tính dữ liệu của lớp. Các phương thức này sẽ cho phép các đối tượng bên ngoài truy nhập vào các thuộc tính dữ liệu của lớp một cách gián tiếp.
- Các phương thức cung cấp chức năng hoạt động, cách cư xử của đối tượng đối với môi trường bên ngoài. Các phương thức này thể hiện chức năng của các đối tượng lớp.

2.4 MỘT SỐ NGÔN NGỮ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Nội dung phần này sẽ trình bày một số ngôn ngữ lập trình hướng đối tượng thông dụng hiện nay:

- Ngôn ngữ lập trình C++
- Ngôn ngữ lập trình ASP.NET và C#.NET

- Ngôn ngữ lập trình Java

2.4.1 C++

C++, ra đời vào giữa những năm 1980, là một ngôn ngữ lập trình hướng đối tượng được mở rộng từ ngôn ngữ lập trình cấu trúc C. Cho nên, C++ là ngôn ngữ lập trình nửa hướng đối tượng, nửa hướng cấu trúc.

Những đặc trưng hướng đối tượng của C++

- Cho phép định nghĩa lớp đối tượng.
- Cho phép đóng gói dữ liệu vào các lớp đối tượng. Cho phép định nghĩa phạm vi truy nhập dữ liệu của lớp bằng các từ khoá phạm vi.
- Cho phép kế thừa lớp với các kiểu kế thừa khác nhau tuỳ vào từ khoá dẫn xuất.
- Cho phép lớp kế thừa sử dụng các phương thức của lớp bị kế thừa (trong phạm vi quy định).
- Cho phép định nghĩa chồng phương thức trong lớp kế thừa.

Những vi phạm hướng đối tượng của C++

Những vi phạm này là do kết quả kế thừa từ ngôn ngữ C, một ngôn ngữ lập trình thuần cấu trúc.

- Cho phép định nghĩa và sử dụng các biến dữ liệu tự do.
- Cho phép định nghĩa và sử dụng các hàm tự do.
- Ngay cả khi dữ liệu được đóng gói vào lớp, dữ liệu vẫn có thể truy nhập trực tiếp như dữ liệu tự do bởi các hàm bạn, lớp bạn (friend) trong C++.

2.4.2 ASP.NET và C#.NET

Các ngôn ngữ lập trình .NET (còn được gọi là .NET Frameworks) của MicroSoft ra đời vào cuối những năm 1990 để cạnh tranh với ngôn ngữ lập trình Java. .NET là một ngôn ngữ hoàn toàn hướng đối tượng, hơn nữa, nó còn cung cấp một giao diện lập trình đồ hoạ thân thiện và đẹp mắt với truyền thống lập trình kéo thả của MicroSoft.

Một số đặc điểm của ngôn ngữ .NET:

- Là một ngôn ngữ hoàn toàn hướng đối tượng: Tất cả các thành phần, các thực thể trong chương trình đều được mô hình dưới dạng một lớp nhất định. Không có dữ liệu tự do và hàm tự do trong chương trình.
- Cung cấp giao diện lập trình đồ hoạ: lập trình viên chỉ cần kéo và thả các đối tượng đồ hoạ cho ứng dụng của mình.
- Cho phép lập trình viên tự tạo ra các thư viện UserControl của mình. Đây là một thư viện bao gồm các thành phần được người dùng tự thiết kế giao diện, viết mã nguồn, đóng gói và có thể sử dụng lại trong nhiều ứng dụng khác nhau, tuỳ theo chức năng của các thành phần.

2.4.3 Java

Java là một ngôn ngữ lập trình được Sun Microsystems giới thiệu vào tháng 6 năm 1995. Java được xây dựng trên nền tảng của C và C++: Java sử dụng cú pháp của C và đặc trưng hướng đối tượng của C++.

Một số đặc điểm của Java:

- Java là một ngôn ngữ lập trình hoàn toàn hướng đối tượng: Tất cả các thực thể đều được coi là một đối tượng, là một thể hiện cụ thể của một lớp xác định. Không có dữ liệu tự do và hàm tự do trong Java, tất cả đều được đóng gói vào các lớp xác định.
- Java là ngôn ngữ vừa biên dịch vừa thông dịch. Đầu tiên mã nguồn được biên dịch thành dạng bytecode; sau đó được thực thi trên từng loại máy nhờ trình thông dịch. Điều này tạo ra khả năng hoạt động độc lập với nền tảng phần cứng của các ứng dụng Java.
- Java cho phép người dùng tự tạo các đối tượng thư viện JavaBeans của mình (tương tự như các thành phần UserControl của .NET). Các đối tượng Bean sẽ được sử dụng lại như các thành phần có sẵn trong các ứng dụng khác. Điều này mở ra khả năng to lớn để tiết kiệm công sức viết mã nguồn và khả năng xây dựng các kỹ thuật cho một nền công nghiệp lắp ráp phần mềm.

Ngôn ngữ lập trình hướng đối tượng Java sẽ được trình bày chi tiết trong toàn bộ phần 2 của giáo trình này.

TỔNG KẾT CHƯƠNG 2

Nội dung chương 2 đã trình bày một số khái niệm cơ bản của lập trình hướng đối tượng:

- Khái niệm đối tượng, dùng để chỉ các thực thể tồn tại thực tế trong các ứng dụng hướng đối tượng.
- Khái niệm lớp, một sự trừu tượng hoá của đối tượng, dùng để biểu diễn đối tượng trong lập trình hướng đối tượng. Các thành phần của lớp là thuộc tính (dữ liệu) và phương thức (hành động).
- Quá trình trừu tượng hoá theo chức năng để hình thành các phương thức của lớp, thể hiện các hoạt động của đối tượng.
- Quá trình trừu tượng hoá theo dữ liệu để hình thành các thuộc tính của lớp, biểu diễn các thuộc tính tương ứng của đối tượng.
- Khái niệm kế thừa trong lập trình hướng đối tượng, nhằm hạn chế việc trùng lặp mã nguồn và tăng khả năng sử dụng lại mã nguồn của chương trình.
- Khái niệm đóng gói trong lập trình hướng đối tượng, nhằm hạn chế tối đa sự thay đổi mã nguồn. Chỉ cần thay đổi trong phương thức, các chương trình bên ngoài có sử dụng phương thức đó không cần phải thay đổi.
- Khái niệm đa hình, cho phép gọi cùng một phương thức, nhưng với các đối tượng khác nhau sẽ có hiệu quả khác nhau.
- So sánh Lớp và Cấu trúc trên các khía cạnh khác nhau: khái niệm, mục đích, chức năng và ưu nhược điểm.
- Mô tả các thành phần nằm trong các vùng khác nhau của Lớp: private và public.

- Giới thiệu một số ngôn ngữ lập trình hướng đối tượng thông dụng hiện nay: C++, .NET, Java.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 2

1. Trong số các nhận định sau, cái nào đúng, cái nào sai:
   a. Đối tượng là một thực thể cụ thể, tồn tại thực tế trong các ứng dụng.
   b. Đối tượng là một thể hiện cụ thể của Lớp.
   c. Lớp là một khái niệm trừu tượng dùng để biểu diễn các Đối tượng.
   d. Lớp là một sự trừu tượng hoá của Đối tượng.
   e. Lớp và Đối tượng có bản chất giống nhau.
   f. Trừu tượng hoá đối tượng theo chức năng tạo ra các thuộc tính của lớp.
   g. Trừu tượng hoá đối tượng theo chức năng tạo ra các phương thức của lớp.
   h. Trừu tượng hoá đối tượng theo dữ liệu tạo ra các thuộc tính của lớp.
   i. Trừu tượng hoá đối tượng theo dữ liệu tạo ra các phương thức của lớp.
   j. Kế thừa cho phép hạn chế sự trùng lặp mã nguồn.
   k. Kế thừa cho phép tăng khả năng sử dụng lại mã nguồn.
   l. Đóng gói hạn chế khả năng truy nhập dữ liệu.
   m. Đóng gói hạn chế việc phải sửa đổi mã nguồn.
   n. Đa hình cho phép thực hiện cùng một thao tác trên nhiều đối tượng khác nhau.
2. Liệt kê tất cả các thuộc tính và hành động của đối tượng Xe ô tô. Đề xuất lớp Car (Ô tô).
3. Liệt kê tất cả các thuộc tính và hành động của đối tượng Xe buýt. Đề xuất lớp Bus.
4. Từ hai lớp Car và Bus của bài 2 và bài 3. Đề xuất một lớp Động cơ (Engine) cho hai lớp trên kế thừa, để tránh trùng lặp dữ liệu giữa hai lớp Car và Bus.

PHẦN 2

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI JAVA

CHƯƠNG 3
GIỚI THIỆU VỀ JAVA

Nội dung của chương này tập trung trình bày các vấn đề chính về ngôn ngữ lập trình Java:

- Lịch sử ra đời và phát triển của Java
- Kiến trúc tổng quát một chương trình xây dựng trên Java
- Các toán tử và các cấu trúc dữ liệu cơ bản trên Java
- Các cấu trúc lệnh của Java

3.1 LỊCH SỬ PHÁT TRIỂN CỦA JAVA

3.1.1 Java

Năm 1991, một nhóm kỹ sư của Sun Microsystems muốn lập trình để điều khiển các thiết bị điện tử như tivi, máy giặt, lò nướng… Ban đầu, họ định dùng C và C++ nhưng trình biên dịch C/C++ lại phụ thuộc vào từng loại CPU. Do đó, họ đã bắt tay vào xây dựng một ngôn ngữ chạy nhanh, gọn, hiệu quả, độc lập thiết bị và ngôn ngữ "Oak" ra đời và vào năm 1995, sau đó được đổi tên thành Java.

Ngôn ngữ lập trình Java được Sun Microsystems đưa ra giới thiệu vào tháng 6 năm 1995 và đã nhanh chóng trở thành một ngôn ngữ lập trình của các lập trình viên chuyên nghiệp. Java được xây dựng dựa trên nền tảng của C và C++ nghĩa là Java sử dụng cú pháp của C và đặc trưng hướng đối tượng của C++. Java là ngôn ngữ vừa biên dịch vừa thông dịch. Đầu tiên mã nguồn được biên dịch thành dạng bytecode. Sau đó được thực thi trên từng loại máy nhờ trình thông dịch. Mục tiêu của các nhà thiết kế Java là cho phép người lập trình viết chương trình một lần nhưng có thể chạy trên các nền phần cứng khác nhau.

Ngày nay, Java được sử dụng rộng rãi, không chỉ để viết các ứng dụng trên máy cục bộ hay trên mạng mà còn để xây dựng các trình điều khiển thiết bị di động, PDA, …

3.1.2 Đặc trưng của ngôn ngữ Java

Ngôn ngữ Java có những đặc trưng cơ bản sau:

- Đơn giản
- Hướng đối tượng
- Độc lập phần cứng và hệ điều hành
- Mạnh mẽ
- Bảo mật
- Phân tán
- Đa luồng
- Linh động

Đơn giản

Những người thiết kế mong muốn phát triển một ngôn ngữ dễ học và quen thuộc với đa số người lập trình. Do vậy Java loại bỏ các đặc trưng phức tạp của C và C++ như:

- Loại bỏ thao tác con trỏ, thao tác định nghĩa chồng toán tử (operator overloading)…
- Không cho phép đa kế thừa (Multi-inheritance) mà sử dụng các giao diện (interface)
- Không sử dụng lệnh "goto" cũng như file header (.h).
- Loại bỏ cấu trúc "struct" và "union".

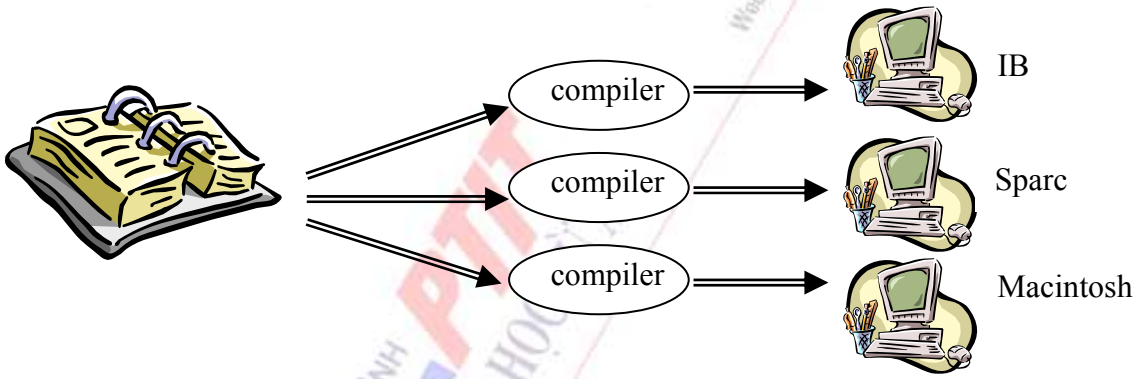
Hướng đối tượng

Java là ngôn ngữ lập trình hoàn toàn hướng đối tượng:

- Mọi thực thể trong hệ thống đều được coi là một đối tượng, tức là một thể hiện cụ thể của một lớp xác định.
- Tất cả các chương trình đều phải nằm trong một class nhất định.
- Không thể dùng Java để viết một chức năng mà không thuộc vào bất kì một lớp nào. Tức là Java không cho phép định nghĩa dữ liệu và hàm tự do trong chương trình.

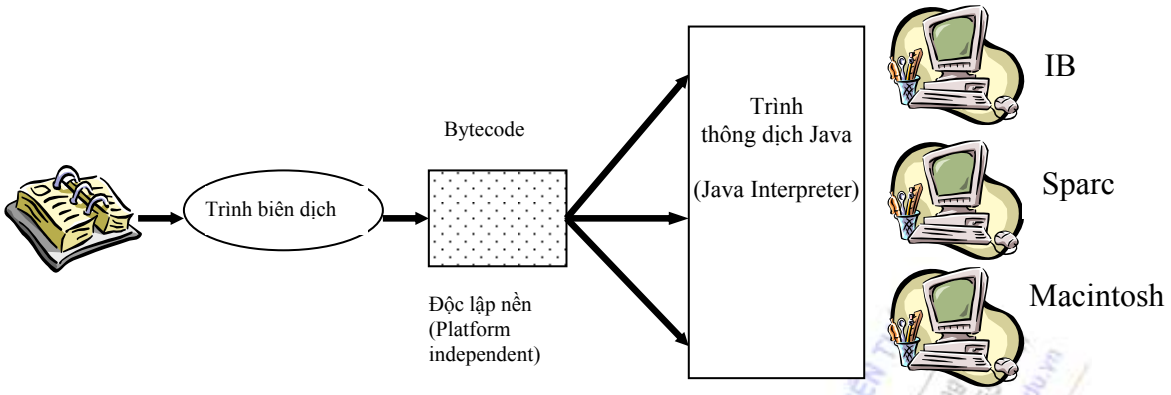
Độc lập phần cứng và hệ điều hành

Đối với các ngôn ngữ lập trình truyền thống như C/C++, phương pháp biên dịch được thực hiện như sau (Hình 3.1): Với mỗi một nền phần cứng khác nhau, có một trình biên dịch khác nhau để biên dịch mã nguồn chương trình cho phù hợp với nền phần cứng ấy. Do vậy, khi chạy trên một nền phần cứng khác, bắt buộc phải biên dịch lại mà nguồn.



Hình 3.1 Cách biên dịch truyền thống

Đối các chương trình viết bằng Java, trình biên dịch Javac sẽ biên dịch mã nguồn thành dạng bytecode. Sau đó, khi chạy chương trình trên các nền phần cứng khác nhau, máy ảo Java dùng trình thông dịch Java để chuyển mã bytecode thành dạng chạy được trên các nền phần cứng tương ứng. Do vậy, khi thay đổi nền phần cứng, không phải biên dịch lại mã nguồn Java. Hình 3.2 minh hoạ quá trình biên dịch và thông dịch mã nguồn Java.



Hình 3.2 Dịch chương trình Java

Mạnh mẽ

Java là ngôn ngữ yêu cầu chặt chẽ về kiểu dữ liệu:

- Kiểu dữ liệu phải được khai báo tường minh.
- Java không sử dụng con trỏ và các phép toán con trỏ.
- Java kiểm tra việc truy nhập đến mảng, chuỗi khi thực thi để đảm bảo rằng các truy nhập đó không ra ngoài giới hạn kích thước mảng.
- Quá trình cấp phát, giải phóng bộ nhớ cho biến được thực hiện tự động, nhờ dịch vụ thu nhặt những đối tượng không còn sử dụng nữa (garbage collection).
- Cơ chế bẫy lỗi của Java giúp đơn giản hóa qúa trình xử lý lỗi và hồi phục sau lỗi.

Bảo mật

Java cung cấp một môi trường quản lý thực thi chương trình với nhiều mức để kiểm soát tính an toàn:

- Ở mức thứ nhất, dữ liệu và các phương thức được đóng gói bên trong lớp. Chúng chỉ được truy xuất thông qua các giao diện mà lớp cung cấp.
- Ở mức thứ hai, trình biên dịch kiểm soát để đảm bảo mã là an toàn, và tuân theo các nguyên tắc của Java.
- Mức thứ ba được đảm bảo bởi trình thông dịch. Chúng kiểm tra xem bytecode có đảm bảo các qui tắc an toàn trước khi thực thi.
- Mức thứ tư kiểm soát việc nạp các lớp vào bộ nhớ để giám sát việc vi phạm giới hạn truy xuất trước khi nạp vào hệ thống.

Phân tán

Java được thiết kế để hỗ trợ các ứng dụng chạy trên mạng bằng các lớp Mạng (java.net). Hơn nữa, Java hỗ trợ nhiều nền chạy khác nhau nên chúng được sử dụng rộng rãi như là công cụ phát triển trên Internet, nơi sử dụng nhiều nền khác nhau.

Đa luồng

Chương trình Java cung cấp giải pháp đa luồng (Multithreading) để thực thi các công việc cùng đồng thời và đồng bộ giữa các luồng.

Linh động

Java được thiết kế như một ngôn ngữ động để đáp ứng cho những môi trường mở. Các chương trình Java chứa rất nhiều thông tin thực thi nhằm kiểm soát và truy nhập đối tượng lúc chạy. Điều này cho phép khả năng liên kết động mã.

3.1.3 Cài đặt Java

Quá trình cài đặt môi trường Java trên máy bao gồm ba bước:

- Copy bộ cài đặt
- Chạy chương trình cài đặt
- Cập nhật biến môi trường

Copy bộ cài đặt

Có thể copy từ đĩa CD hoặc tải xuống miễn phí tại địa chỉ web site của nhóm Java:

http://www.java.sun.com/downloads/

Chạy chương trình cài đặt

Chạy tập tin vừa copy (hoặc vừa tải về). Chọn thư mục cài đặt, thư mục mặc định là:

`C:\jdk1.5` (với phiên bản JDK1.5)

Cập nhật biến môi trường

Cập nhật biến môi trường (PATH) giúp việc thực thi và biên dịch mã Java có thể tiến hành ở bất cứ một thư mục nào. Để cập nhật biến PATH, cần thêm đường dẫn đầy đủ của thư mục java vừa cài đặt (`C:\jdk1.5\bin`) vào cuối của giá trị biến này.

- Đối với WindowsNT, WinXP khởi động `Control Panel`, chọn `System`, chọn `Environment` (hoặc click chuột phải vào `My Computer`, chọn `Properties`, chọn `Advanced`, click vào `Environment Variables`), click vào biến PATH trong phần `User Variables` và `System Variables`. Sau đó, thêm vào cuối nội dung biến hiện có dòng sau (phải có dấu chấm phẩy):

  ```
  ;C:\jdk1.5\bin
  ```

- Đối với Windows98/95, chọn START, chọn RUN, nhập dòng sysedit vào ô lệnh, nhấn OK, chọn cửa sổ của AUTOEXEC.BAT. Tìm dòng khai báo biến PATH, nếu không có, thêm vào một dòng mới theo mẫu: `SET PATH=C:\jdk1.5\bin`. Nếu có sẵn biến PATH, thêm vào cuối dòng này nội dung: `;C:\jdk1.5\bin`

3.2 KIẾN TRÚC CHƯƠNG TRÌNH XÂY DỰNG TRÊN JAVA

3.2.1 Kiến trúc chương trình Java

Dạng cơ bản của một tập tin mã nguồn Java có cấu trúc như sau :

```
package packageName; // Khai báo tên gói, nếu có
import java.awt.*;   // Khai báo tên thư viện sẵn có, nếu cần dùng

class className      // Khai báo tên lớp
{
   /* Đây là dòng ghi chú */
   int var;          // Khai báo biến

   public void methodName()// Khai báo tên phương thức
   {
      /* Phần thân của phương thức */
      statement (s); // Lệnh thực hiện
   }
}
```

Một tệp mã nguồn Java có thể có ba phần chính:

- Phần khai báo tên gói (khối) bằng từ khoá **package**.
- Phần khai báo thư viện tham khảo bằng từ khoá **import**.
- Phần khai báo nội dung lớp bằng từ khoá **class**.

Khai báo Package

Package được dùng để đóng gói các lớp trong chương trình lại với nhau thành một khối. Đây là một cách hữu hiệu để lưu trữ các lớp gần giống nhau hoặc có cùng một module thành một khối thống nhất.

Cú pháp khai báo tên gói bằng từ khoá **package**:

```
package <Tên gói>;
```

Để đặt tên package trong chương trình, người ta có thể tiến hành như đặt tên thư mục trên ổ đĩa. Nghĩa là bắt dầu bằng tên có phạm vi lớn, cho đến các tên có phạm vi nhỏ, cuối cùng là tên các gói trực tiếp chứa các lớp. Phạm vi đặt tên gói, trên thực tế, được tiến hành theo thứ tự phạm vi lớn đến nhỏ như sau:

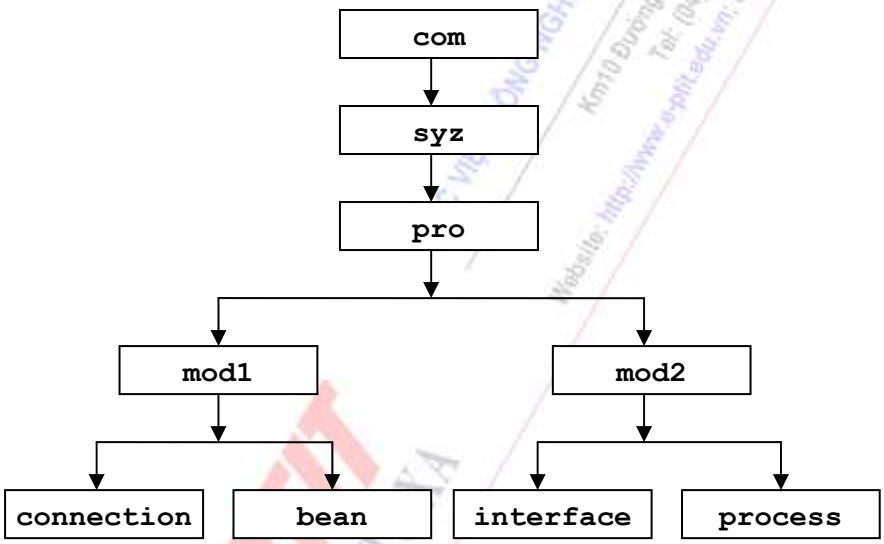
- Tên tổ chức
- Tên công ty
- Tên dự án

- Tên modul trong dự án
- Tên các chức năng trong modul.

Ví dụ:

- Tên miền của công ty là **syz.com**
- Tên dự án là **pro**
- Dự án có hai modul là **mod1** và **mod2**
- Modul mod1 có hai chức năng là kết nối cơ sở dữ liệu **connection** và biểu diễn dữ liệu **bean**.
- Modul mod2 có hai chức năng là giao tiếp **interface** và xử lí yêu cầu **process**.

Khi đó, cấu trúc khối của dự án được mô tả như hình 3.3



Hình 3.3: Kiến trúc khối của dự án

Khi đó, trong chức năng **bean** có lớp User, thì phải khai báo tên khối trong lớp này như sau:

```
package com.syz.pro.mod1.bean;
```

Ưu điểm của package:

- Cho phép nhóm các lớp vào với nhau thành các đơn vị nhỏ hơn. Việc thao tác trên các đơn vị khối sẽ gọn hơn thao tác trên một tập các lớp.
- Tránh việc xung đột khi đặt tên lớp. Vì các lớp không cùng package thì có thể đặt tên trùng nhau. Khi số lượng lớp của chương trình quá lớn ta có thể tránh phải đặt tên khác nhau cho các lớp bằng cách đặt chúng vào các package khác nhau.
- Cho phép bảo vệ các lớp. Khi chương trình lớn, việc chia nhỏ chương trình thành các package sẽ thuận lợi hơn cho việc quản lí và phát triển.
- Tên gói còn được dùng để định danh lớp trong ứng dụng.

Lưu ý:

- Dòng lệnh khai báo tên khối phải được đặt đầu tiên trong tệp tin mã chương trình.

- Chỉ được khai báo tối đa một tên khối đối với mỗi tệp mã nguồn Java.
- Các tệp tin của các lớp nằm cùng gói ứng dụng phải được lưu trong cùng một thư mục (tên thư mục là tên khối) theo cấu trúc khối của dự án.
- Tên khối nên đặt theo chữ thường vì tên khối sẽ là tên thư mục tương ứng trong ổ đĩa, tránh nhầm lẫn với tên các tệp tin là tên các lớp của chương trình.
- Khi không phân chia chương trình thành khối (chương trình đơn giản), không cần thiết phải khai báo tên khối ở đầu chương trình.

Khai báo thư viện

Khai báo thư viện để chỉ ra những thư viện đã được định nghĩa sẵn mà chương trình sẽ tham khảo tới. Cú pháp khai báo thư viện với từ khoá **import** như sau:

```
import <Tên thư viện>;
```

Java chuẩn cung cấp một số thư viện như sau:

- **java.lang**: cung cấp các hàm thao tác trên các kiểu dữ liệu cơ bản, xử lí lỗi và ngoại lệ, xử lí vào ra trên các thiết bị chuẩn như bàn phím và màn hình.
- **java.applet**: cung cấp các hàm cho xây dựng các applet (sẽ trình bày trong Chương 6).
- **java.awt**: cung cấp các hàm cho xây dựng các ứng dụng đồ hoạ với các thành phần giao diện multi media (sẽ trình bày chi tiết trong Chương 6).
- **java.io**: cung cấp các hàm xử lí vào/ra trên các thiết bị chuẩn và các thiết bị ngoại vi.
- **java.util**: cung cấp các hàm tiện ích trong xử lí liên quan đến các kiểu dữ liệu có cấu trúc như Date, Stack, Vector.

Ví dụ, nếu trong chương trình cần đến các thao tác chuyển kiểu đổi dữ liệu tường minh (từ kiểu string sang kiểu int), thì ta sẽ phải tham khảo thư viện **java.lang**:

```
import java.lang.*;
```

Lưu ý:

- Nếu muốn khai báo tham khảo nhiều thư viện, phải khai báo tham khảo mỗi thư viện với một từ khoá **import**.
- Nếu chỉ tham khảo một vài lớp trong một thư viện, nên chỉ rõ tham khảo lớp nào, thay vì phải khai báo tham khảo cả gói (bằng kí hiệu "*") vì tham khảo cả gói sẽ tăng kích cỡ tệp tin class sau khi biên dịch.
- Nếu không tham khảo thư viện nào, không cần thiết phải khai báo các tham khảo với từ khoá import.

Khai báo lớp

Phần thứ ba là phần khai báo lớp và nội dung của lớp, phần này luôn bắt buộc phải có đối với một tệp mã nguồn Java:

- Khai báo tên lớp với từ khoá **class**.
- Khái báo các thuộc tính của lớp.

- Khai báo các phương thức của lớp

Việc khai báo lớp với các thuộc tính và phương thức sẽ được trình bày chi tiết trong chương 4.

3.2.2 Chương trình Java đầu tiên

Chương trình sau đây cho phép hiển thị một thông điệp (Nằm trong tệp mã nguồn First.java):

Chương trình 3.1

```
package vidu.chuong3;
// Đây là chương trình "First.java"
class First
{
   public static void main(String args[])
   {
      System.out.println("Hello World");
   }
}
```

Để biên dịch mã nguồn, ta sử dụng trình biên dịch **javac**. Trình biên dịch xác định tên của file nguồn tại dòng lệnh như mô tả dưới đây (giả sử đang ở thư mục chứa package vidu và biến môi trường PATH đã được thiết lập đúng qui cách):

>javac vidu/chuong3/First.java

Trình dịch javac tạo ra file First.class chứa các mã "bytecodes". Những mã này chưa thể thực thi được. Để chương trình thực thi được ta cần dùng trình thông dịch "**java interpreter"** với lệnh **java.** Lệnh được thực hiện như sau:

>javac vidu.chuong3.First

Kết quả sẽ hiển thị trên màn hình như sau:

```
Hello World
```

3.2.3 Phân tích chương trình đầu tiên

Trong Java, tất cả mã lệnh đều phải được tổ chức vào trong một lớp nhất định. Do đó, mỗi tệp tin mã nguồn xác định ít nhất một lớp Java và tên tệp tin phải trùng với tên lớp. Java phân biệt chữ hoa và chữ thường, cho nên tên tập tin của chương trình trên phải trùng với tên lớp: First.java.

```
package vidu.chuong3;
```

Đây là dòng khai báo tên khối của chương trình, vì tên khối của chương trình được đặt theo hai mức:

- Mức thứ nhất là kiểu bài: ví dụ (vidu) hoặc bài tập (baitap).
- Mức thứ hai là tên của chương: chuong3, chuong4, chuong5, chuong6.

Vì đây là ví dụ, nằm ở chương 3 nên thuộc vào gói **vidu.chuong3**. Đồng thời, tệp tin First.java sẽ nằm trong thư mục: ../vidu/chuong3/.

Chương trình này không tham khảo thư viện nào nên không cần lệnh import nào.

```
// Đây là chương trình "First.java"
```

Ký hiệu "// " dùng để chú thích dòng lệnh. Trình biên dịch sẽ bỏ qua dòng chú thích này. Java hỗ trợ hai loại chú thích:

- Loại chú thích trên một dòng, dùng "//". Trình biên dịch sẽ bỏ qua nội dung bắt đầu từ kí hiệu "//" cho đến hết dòng lệnh chứa nó.
- Loại chú thích trên nhiều dòng có thể bắt đầu với "/*" và kết thúc với "*/". Trình biên dịch sẽ bỏ qua nội dung nằm giữa hai kí hiệu này.

Dòng kế tiếp khai báo lớp có tên **First**: Bắt đầu với từ khoá **class**, kế đến là tên lớp

```
class First
```

Một định nghĩa lớp nằm trọn vẹn giữa hai ngoặc móc mở "{" và đóng "}". Các ngoặc này đánh dấu bắt đầu và kết thúc một khối lệnh.

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

Đây là phương thức chính, từ đây chương trình bắt đầu việc thực thi của mình. Tất cả các ứng dụng java đều sử dụng một phương thức **main** này.

- Từ khoá **public** là một chỉ định truy xuất. Nó cho biết thành viên của lớp có thể được truy xuất từ bất cứ đâu trong chương trình.
- Từ khoá **static** cho phép **main** được gọi tới mà không cần tạo ra một thể hiện (instance) của lớp. Nó không phụ thuộc vào các thể hiện của lớp được tạo ra.
- Từ khoá **void** thông báo cho máy tính biết rằng phương thức sẽ không trả lại bất cứ giá trị nào khi thực thi chương trình.
- **String args[]** là tham số dùng trong phương thức **main**. Khi không có một thông tin nào được chuyển vào **main,** phương thức được thực hiện với các dữ liệu rỗng – không có gì trong dấu ngoặc đơn.
- **System.out.println("Hello World");** Dòng lệnh này hiển thị chuỗi "**Hello World**" trên màn hình. Lệnh **println()** cho phép hiển thị chuỗi được truyền vào lên màn hình.

Truyền đối số trong dòng lệnh

Chương trình 3.2 minh hoạ các tham số (argument) của các dòng lệnh được tiếp nhận như thế nào trong phương thức **main.**

Chương trình 3.2

```
package vidu.chuong3;
class  PassArgument{
   public static void main(String args[])
   {
      System.out.println("This is what the main method received");
      System.out.println(args[0]);
      System.out.println(args[1]);
      System.out.println(args[2]);
   }
}
```

Biên dịch chương trình:

```
>javac PassArgumet.java
```

Thực thi chương trình với dòng lệnh:

```
>java PassArgument A 123 B1
```

Sẽ thu được trên màn hình kết quả:

```
This is what the main method received
A
123
B1
```

3.3 CÁC KIỂU DỮ LIỆU VÀ TOÁN TỬ CƠ BẢN TRÊN JAVA

3.3.1 Khai báo biến

Cú pháp khai báo biến:

```
dataType varName;
```

Trong đó, dataType là kiểu dữ liệu của biến, varName là tên biến. Trong Java, việc đặt tên biến phải tuân theo các quy tắc sau:

- Chỉ được bắt đầu bằng một kí tự (chữ), hoặc một dấu gạch dưới , hoặc một kí tự dollar
- Không có khoảng trắng giữa tên
- Bắt đầu từ kí tự thứ hai, có thể dùng các kí tự (chữ), chữ số, dấu dollar, dấu gạch dưới
- Không trùng với các từ khoá
- Có phân biệt chữ hoa chữ thường

Phạm vi hoạt động của biến

Một biến có phạm vi hoạt động trong toàn bộ khối lệnh mà nó được khai báo. Một khối lệnh bắt đầu bằng dấu "{" và kết thúc bằng dấu "}":

- Nếu biến được khai báo trong một cấu trúc lệnh điều khiển, biến đó có phạm vi hoạt động trong khối lệnh tương ứng.
- Nếu biến được khai báo trong một phương thức (Không nằm trong khối lệnh nào), biến đó có phạm vi hoạt động trong phương thức tương ứng: có thể được sử dụng trong tất cả các khối lệnh của phương thức.
- Nếu biến được khai báo trong một lớp (Không nằm trong trong một phương thức nào), biến đó có phạm vi hoạt động trong toàn bộ lớp tương ứng: có thể được sử dụng trong tất cả các phương thức của lớp.

3.3.2 Kiểu dữ liệu

Trong Java, kiểu dữ liệu được chia thành hai loại:

- Các kiểu dữ liệu cơ bản
- Các kiểu dữ liệu đối tượng

Kiểu dữ liệu cơ bản

Java cung cấp các kiểu dữ liệu cơ bản như sau:

byte: Dùng để lưu dữ liệu kiểu số nguyên có kích thước một byte (8 bít). Phạm vi biểu diễn giá trị từ -128 đến 127. Giá trị mặc định là 0.

char: Dùng để lưu dữ liệu kiểu kí tự hoặc số nguyên không âm có kích thước 2 byte (16 bít). Phạm vi biểu diễn giá trị từ 0 đến u\ffff. Giá trị mặc định là 0.

boolean: Dùng để lưu dữ liệu chỉ có hai trạng thái đúng hoặc sai (độ lớn chỉ có 1 bít). Phạm vi biểu diễn giá trị là {"True", "False"}. Giá trị mặc định là False.

short: Dùng để lưu dữ liệu có kiểu số nguyên, kích cỡ 2 byte (16 bít). Phạm vi biểu diễn giá trị từ - 32768 đến 32767. Giá trị mặc định là 0.

int: Dùng để lưu dữ liệu có kiểu số nguyên, kích cỡ 4 byte (32 bít). Phạm vi biểu diễn giá trị từ -2,147,483,648 đến 2,147,483,647. Giá trị mặc định là 0.

float: Dùng để lưu dữ liệu có kiểu số thực, kích cỡ 4 byte (32 bít). Giá trị mặc định là 0.0f.

double: Dùng để lưu dữ liệu có kiểu số thực có kích thước lên đến 8 byte. Giá trị mặc định là 0.00d

long: Dùng để lưu dữ liệu có kiểu số nguyên có kích thước lên đến 8 byte. Giá trị mặc định là 0l.

Kiểu dữ liệu đối tượng

Trong Java, có 3 kiểu dữ liệu đối tượng:

Array: Một mảng của các dữ liệu cùng kiểu

class: Dữ liệu kiểu lớp đối tượng do người dùng định nghĩa. Chứa tập các thuộc tính và phương thức.

interface: Dữ liệu kiểu lớp giao tiếp do người dùng định nghĩa. Chứa các phương thức của giao tiếp.

Ép kiểu (Type casting)

Ví dụ, nhiều khi gặp tình huống cần cộng một biến có dạng **integer** với một biến có dạng **float.** Để xử lý tình huống này, Java sử dụng tính năng ép kiểu (type casting) của C/C++. Đoạn mã sau đây thực hiện phép cộng một giá trị dấu phẩy động (float) với một giá trị nguyên (integer).

```
float c = 35.8f;
int b = (int)c + 1;
```

Đầu tiên giá trị dấu phảy động **c** được đổi thành giá trị nguyên 35. Sau đó nó được cộng với 1 và kết quả là giá trị 36 được lưu vào **b.**

Trong Java có hai loại ép kiểu dữ liệu:

- Nới rộng (widening): quá trình làm tròn số từ kiểu dữ liệu có kích thước nhỏ hơn sang kiểu có kích thước lớn hơn. Kiểu biến đổi này không làm mất thông tin. Ví dụ chuyển từ **int** sang **float**. Chuyển kiểu loại này có thể được thực hiện ngầm định bởi trình biên dịch.
- Thu hẹp (narrowwing): quá trình làm tròn số từ kiểu dữ liệu có kích thước lớn hơn sang kiểu có kích thước nhỏ hơn. Kiểu biến đổi này có thể làm mất thông tin như ví dụ ở trên. Chuyển kiểu loại này không thể thực hiện ngầm định bởi trình biên dịch, người dùng phải thực hiện chuyển kiểu tường minh.

3.3.3 Các toán tử

Java cung cấp các dạng toán tử sau:

- Toán tử số học
- Toán tử bit
- Toán tử quan hệ
- Toán tử logic
- Toán tử điều kiện
- Toán tử gán

Toán tử số học

Các toán hạng của các toán tử số học phải ở dạng số. Các toán hạng kiểu boolean không sử dụng được, song các toán hạng ký tự cho phép sử dụng loại toán tử này. Một vài kiểu toán tử được liệt kê trong bảng dưới đây.

Toán tử	**Mô tả**
+	Cộng. Trả về giá trị tổng hai toán hạng
-	Trừ

	Trả về kết quả của phép trừ.
*	Nhân Trả về giá trị là tích hai toán hạng.
/	Chia Trả về giá trị là thương của phép chia
%	Phép lấy modul Giá trị trả về là phần dư của phép chia
++	Tăng dần Tăng giá trị của biến lên 1. Ví dụ a++ tương đương với a = a + 1
--	Giảm dần Giảm giá trị của biến 1 đơn vị. Ví dụ a-- tương đương với a = a - 1
+=	Cộng và gán giá trị Cộng các giá trị của toán hạng bên trái vào toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ c += a tương đương c = c + a
-=	Trừ và gán giá trị Trừ các giá trị của toán hạng bên trái vào toán toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ c -= a tương đương với c = c - a
*=	Nhân và gán Nhân các giá trị của toán hạng bên trái với toán toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ c *= a tương đương với c = c*a
/=	Chia và gán Chia giá trị của toán hạng bên trái cho toán toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ c /= a tương đương với c = c/a
%=	Lấy số dư và gán Chia giá trị của toán hạng bên trái cho toán toán hạng bên phải và gán giá trị **số dư** vào toán hạng bên trái. Ví dụ c %= a tương đương với c = c%a

Bảng 3.1 Các toán tử số học

Toán tử Bit

Các toán tử dạng bit cho phép ta thao tác trên từng bit riêng biệt trong các kiểu dữ liệu nguyên thuỷ.

Toán tử	**Mô tả**
~	Phủ định bit (NOT) Trả về giá trị phủ định của một bít.
&	Toán tử AND bít Trả về giá trị là 1 nếu các toán hạng là 1 và 0 trong các trường hợp khác
|	Toán tử OR bít Trả về giá trị là 1 nếu một trong các toán hạng là 1 và 0 trong các trường hợp khác.

^	Toán tử Exclusive OR bít Trả về giá trị là 1 **nếu chỉ một** trong các toán hạng là 1 và trả về 0 trong các trường hợp khác.
>>	Dịch sang phải bít Chuyển toàn bộ các bít cuả một số sang phải một vị trí, giữ nguyên dấu của số âm. Toán hạng bên trái là số bị dịch còn số bên phải chỉ số vị trí mà các bít cần dịch.
<<	Dịch sang trái bít Chuyển toàn bộ các bít cuả một số sang trái một vị trí, giữ nguyên dấu cuả số âm. Toán hạng bên trái là số bị dịch còn số bên phải chỉ số vị trí mà các bít cần dịch.

Bảng 3.2 Các toán tử Bit

Các toán tử quan hệ

Các toán tử quan hệ kiểm tra mối quan hệ giữa hai toán hạng. Kết quả của một biểu thức có dùng các toán tử quan hệ là những giá trị Boolean (logic "đúng" hoặc "sai"). Các toán tử quan hệ được sử dụng trong các cấu trúc điều khiển.

Toán tử	**Mô tả**
= =	So sánh bằng Toán tử này kiểm tra sự tương đương của hai toán hạng
!=	So sánh khác Kiểm tra sự khác nhau của hai toán hạng
>	Lớn hơn Kiểm tra giá trị của toán hạng bên phải lớn hơn toán hạng bên trái hay không
<	Nhỏ hơn Kiểm tra giá trị của toán hạng bên phải có nhỏ hơn toán hạng bên trái hay không
>=	Lớn hơn hoặc bằng Kiểm tra giá trị của toán hạng bên phải có lớn hơn hoặc bằng toán hạng bên trái hay không
<=	Nhỏ hơn hoặc bằng Kiểm tra giá trị của toán hạng bên phải có nhỏ hơn hoặc bằng toán hạng bên trái hay không

Bảng 3.3 Các toán tử quan hệ

Các toán tử logic

Các toán tử logic làm việc với các toán hạng Boolean. Một vài toán tử kiểu này được chỉ ra dưới đây

Toán tử	**Mô tả**
&&	Và (AND) Trả về một giá trị "Đúng" (True) nếu chỉ khi cả hai toán tử có giá trị "True"

||	Hoặc (OR) Trả về giá trị "True" nếu ít nhất một giá trị là True
^	XOR Trả về giá trị True nếu và chỉ nếu chỉ một trong các giá trị là True, các trường hợp còn lại cho giá trị False (sai)
!	Toán hạng đơn tử NOT. Chuyển giá trị từ True sang False và ngược lại.

Bảng 3.4 Các toán tử logic

Các toán tử điều kiện

Toán tử điều kiện là một loại toán tử đặc biệt vì nó bao gồm ba thành phần cấu thành biểu thức điều kiện. Cú pháp:

```
<biểu thức 1> ? <biểu thức 2> : <biểu thức 3>;
```

- **biểu thức 1:** Biểu thức logic. Trả trả về giá trị True hoặc False
- **biểu thức 2:** Là giá trị trả về nếu <biểu thức 1> xác định là True
- **biểu thức 3:** Là giá trị trả về nếu <biểu thức 1> xác định là False

Toán tử gán

Toán tử gán (=) dùng để gán một giá trị vào một biến và có thể gán nhiều giá trị cho nhiều biến cùng một lúc. Ví dụ đoạn lệnh sau gán một giá trị cho biến **var** và giá trị này lại được gán cho nhiều biến trên một dòng lệnh đơn.

```
int var = 20;
int p,q,r,s;
p=q=r=s=var;
```

Dòng lệnh cuối cùng được thực hiện từ phải qua trái. Đầu tiên giá trị ở biến var được gán cho 's', sau đó giá trị của 's' được gán cho 'r' và cứ tiếp như vậy.

Thứ tự ưu tiên của các toán tử

Các biểu thức được viết ra nói chung gồm nhiều toán tử. Thứ tự ưu tiên quyết định trật tự thực hiện các toán tử trên các biểu thức. Bảng dưới đây liệt kê thứ tự thực hiện các toán tử trong Java

Thứ tự	**Toán tử**
1.	Các toán tử đơn như +,-,++,--
2.	Các toán tử số học và các toán tử dịch như *,/,+,-,<<,>>
3.	Các toán tử quan hệ như >,<,>=,<=,= =,!=
4.	Các toán tử logic và Bit như &&,||,&,|,^
5.	Các toán tử gán như =,*=,/=,+=,-=

Bảng 3.5 Thứ tự ưu tiên các toán tử

Thay đổi thứ tự ưu tiên

Để thay đổi thứ tự ưu tiên trên một biểu thức, bạn có thể sử dụng dấu ngoặc đơn ():

- Phần được giới hạn trong ngoặc đơn được thực hiện trước.
- Nếu dùng nhiều ngoặc đơn lồng nhau thì toán tử nằm trong ngoặc đơn phía trong sẽ thực thi trước, sau đó đến các vòng phía ngoài.
- Trong phạm vi một cặp ngoặc đơn thì quy tắc thứ tự ưu tiên vẫn giữ nguyên tác dụng.

3.4 CÁC CẤU TRÚC LỆNH TRÊN JAVA

Java cung cấp hai loại cấu trúc điều khiển:

Điều khiển rẽ nhánh

- Mệnh đề if-else
- Mệnh đề swich-case

Vòng lặp (Loops)

- Vòng lặp while
- Vòng lặp do-while
- Vòng lặp for

3.4.1 Câu lệnh if-else

Câu lệnh **if-else** kiểm tra giá trị dạng boolean của điều kiện. Nếu giá trị điều kiện là **True** thì chỉ có khối lệnh sau **if** sẽ được thực hiện, nếu là **False** thì chỉ có khối lệnh sau **else** được thực hiện. Cú pháp:

```
if (conditon)
{
      action1 statements;
}
else
{
      action2 statements;
}
```

Condition: Biểu thức boolean như toán tử so sánh.

action 1: Khối lệnh được thực thi khi giá trị điều kiện là True

action 2: Khối lệnh được thực thi nếu điều kiện trả về giá trị False

Đoạn chương trình sau kiểm tra xem các số có chia hết cho 5 hay không.

Chương trình 3.3

```
package vidu.chuong3;
class CheckNumber
{
   public static void main(String args[])
```

```
    {
        int num = 10;
        if(num%5 == 0)
            System.out.println (num + " is divisable for 5!");
        else
            System.out.println (num + " is indivisable for 5!");
    }
}
```

Ở đoạn chương trình trên num được gán giá trị nguyên là 10. Trong câu lệnh **if-else** điều kiện **num** %5 trả về giá trị 0 và điều kiện thực hiện là True. Thông báo "10 is divisable for 5!" được in ra. Lưu ý rằng vì chỉ có một câu lệnh được viết trong đoạn "if" và "else", bởi vậy không cần thiết phải được đưa vào dấu ngoặc móc "{" và "}".

3.4.2 Câu lệnh switch-case

Khối lệnh switch-case có thể được sử dụng thay thế câu lệnh if-else trong trường hợp một biểu thức cho ra nhiều kết quả. Cú pháp:

```
swich (expression)
{
      case 'value1': action 1 statement;
                     break;
      case 'value2': action 2 statement;
                     break;
      ……
      case 'valueN': actionN statement;
                     break;
      default:   default_action statement;
}
```

expression - Biến chứa một giá trị xác định

value1,value 2,….valueN: Các giá trị hằng số phù hợp với giá trị trên biến **expression** .

action1,action2…actionN: Khối lệnh được thực thi khi trường hợp tương ứng có giá trị True

break: Từ khoá được sử dụng để bỏ qua tất cả các câu lệnh sau đó và giành quyền điều khiển cho cấu trúc bên ngoài **switch**

default: Từ khóa tuỳ chọn được sử dụng để chỉ rõ các câu lệnh nào được thực hiện chỉ khi tất cả các trường hợp nhận giá trị False

default - action: Khối lệnh được thực hiện chỉ khi tất cả các trường hợp nhận giá trị False

Đoạn chương trình sau xác định giá trị trong một biến nguyên và hiển thị ngày trong tuần được thể hiện dưới dạng chuỗi. Để kiểm tra các giá trị nằm trong khoảng từ 0 đến 6, chương trình sẽ thông báo lỗi nếu nằm ngoài phạm vi trên.

Chương trình 3.4

```
package vidu.chuong3;
class SwitchDemo
{
   public static void main(String agrs[])
   {
      int day = 2;
      switch(day)
      {
         case 0 : System.out.println("Sunday");
                  break;
         case 1 : System.out.println("Monday");
                  break;
         case 2 : System.out.println("Tuesday");
                  break;
         case 3 : System.out.println("Wednesday");
                  break;
         case 4 : System.out.println("Thursday");
                  break;
         case 5:  System.out.println("Friday");
                  break;
         case 6 : System.out.println("Satuday");
                  break;
         default:
                  System.out.println("Invalid day of week");
      }
   }
}
```

Nếu giá trị của biến **day** là 2, chương trình sẽ hiển thị **`Tuesday`**, và cứ tiếp như vậy .

3.4.3 Vòng lặp While

Vòng lặp **while** thực thi khối lệnh khi điều kiện thực thi vẫn là True và dừng lại khi điều kiện thực thi nhận giá trị False. Cú pháp:

```
while(condition)
{
      action statements;
}
```

condition: có giá trị bool; vòng lặp sẽ tiếp tục cho nếu điều kiện vẫn có giá trị True.

action statement: Khối lệnh được thực hiện nếu **condition** nhận giá trị True

Đoạn chương trình sau tính tổng của 5 số tự nhiên đầu tiên dùng cấu trúc while.

Chương trình 3.5

```
package vidu.chuong3;
class WhileDemo
{
   public static void main(String args[])
   {
      int a = 5, sum = 1;
      while  (a >= 1)
      {
         sum +=a;
         a--;
      }
      System.out.println("The sum is " + sum);
   }
}
```

Ở ví dụ trên, vòng lặp được thực thi cho đến khi điều kiện a>=1 là **True**. Biến **a** được khai báo bên ngoài vòng lặp và được gán giá trị là 5. Cuối mỗi vòng lặp, giá tri của **a** giảm đi 1. Sau năm vòng giá trị của **a** bằng 0. Điều kiện trả về giá trị False và vòng lặp kết thúc. Kết quả sẽ được hiển thị **" The sum is 15"**

3.4.4 Vòng lặp do-while

Vòng lặp do-while thực thi khối lệnh khi mà điều kiện là True, tương tự như vòng lặp while, ngoại trừ do-while thực hiện lệnh ít nhất một lần ngay cả khi điều kiện là False. Cú pháp:

```
do{
      action statements;
}while(condition);
```

condition: Biểu thức bool; vòng lặp sẽ tiếp tục khi mà điều kiện vẫn có giá trị True.

action statement: Khối lệnh luôn được thực hiện ở lần thứ nhất, từ vòng lặp thứ hai, chúng được thực hiện khi **condition** nhận giá trị True.

Ví dụ sau tính tổng của 5 số tự nhiên đầu tiên dùng cấu trúc do-while.

Chương trình 3.6

```
package vidu.chuong3;
class DoWhileDemo
{
   public static void main(String args[])
   {
      int a = 1, sum = 0;
      do{
```

```
            sum += a;
            a++;
        }while  (a <= 5);
        System.out.println("Sum of 1 to 5  is " + sum);
    }
}
```

Biến **a** được khởi tạo với giá trị 1, sau đó nó vừa được dùng làm biến chạy (tăng lên 1 sau mỗi lần lặp) vừa được dùng để cộng dồn vào biến **sum**. Tại thời điểm kết thúc, chương trình sẽ in ra **Sum of 1 to 5 is 15**.

3.4.5 Vòng lặp for

Vòng lặp for cung cấp một dạng kết hợp tất cả các đặc điểm chung của tất cả các loại vòng lặp: giá trị khởi tạo của biến chạy, điều kiện dừng của vòng lặp và lệnh thay đổi giá trị của biến chạy. Cú pháp:

```
for(initialization statements; condition; increment statements)
{
      action statements;
}
```

initialization statements: khởi tạo giá trị ban đầu cho các biến chạy, các lệnh khởi tạo được phân cách nhau bởi dấu phẩy và chỉ thực hiện duy nhất một lần vào thời điểm bắt đầu của vòng lặp.

condition: Biểu thức bool; vòng lặp sẽ tiếp tục cho đến khi nào điều kiện có giá trị False.

increment statements: Các câu lệnh thay đổi giá trị của biến chạy. Các lệnh này luôn được thực hiện sau mỗi lần thực hiện khối lệnh trong vòng lặp. Các lệnh phận biệt nhau bởi dấu phẩy.

Đoạn chương trình sau hiển thi tổng của 5 số đầu tiên dùng vòng lặp for.

Chương trình 3.7

```
package vidu.chuong3;
class ForDemo
{
   public static void main(String args[])
   {
      int sum = 0;
      for (int i=1; i<=5; i++)
         sum += i;
      System.out.println ("The sum is " + sum);
   }
}
```

Ở ví dụ trên, i và sum là hai biến được gán các giá trị đầu là 1 và 0 tương ứng. Điều kiện được kiểm tra và khi nó còn nhận giá trị True, câu lệnh tác động trong vòng lặp được thực hiện. Tiếp theo giá trị của i được tăng lên 2 để tạo ra số chẵn tiếp theo. Một lần nữa, điều kiện lại được kiểm tra và câu lệnh tác động lại được thực hiện. Sau năm vòng, i tăng lên 6, điều kiện trả về giá trị False và vòng lặp kết thúc. Thông báo: **The sum is 15** được hiển thị.

3.5 CASE STUDY I

Bây giờ, áp dụng các nội dung đã học trong chương này để viết một chương trình tính chu vi và diện tích của một hình chữ nhật có kích thước x,y với yêu cầu:

- Kích thước x, y nhập từ tham số dòng lệnh.
- Phải kiểm tra x, y là các số nguyên dương hay không trước khi tính toán.
- In kết quả tính toán ra màn hình

Đây là đoạn chương trình thực hiện bài toán này.

Chương trình 3.8

```
package vidu.chuong3;

import java.awt.*;
import java.lang.*;

class RectangleDemo
{
   public static void main(String args[])
   {
      //khai báo các biến lưu giữ kích thước của hình chữ nhật
      int x = 0, y = 0;

      /*đọc các kích thước từ tham số dòng lệnh*/
      //nếu truyền đủ hai tham số thì mới tính tiếp
      if(args.length >= 2)
      {
         //chuyển kiểu từ String sang integer
         x = Integer.parseInt(args[0]);
         y = Integer.parseInt(args[1]);
      }

      /*Tinh chu vi và diện tích hình chữ nhật*/
      //nếu cả hai tham số đều dương thì mới tính
      if(x>0 && y>0)
      {
         //tính chu vi
         int chuvi = 2*(x + y);
               System.out.println ("Chu vi là " + chuvi);
         //tính diện tích
         int dientich = x*y;
         System.out.println ("Diện tích là " + dientich);
      }
      else
         System.out.println ("Các tham số không đúng!");
   }
}
```

Sau khi biên dịch chương trình3.8 (tệp tin có tên RectangleDemo.java), ta chạy từ cửa sổ dòng lệnh:

>java RectangleDemo 10 20

Sẽ thu được kết quả:

```
Chu vi là: 60
Diện tích là: 200
```

Nếu chỉ gõ ở cửa sổ dòng lệnh:

>java RectangleDemo

Thì sẽ nhận được một thông báo lỗi:

```
Các tham số không đúng!
```

TỔNG KẾT CHƯƠNG 3

Nội dung chương 3 đã trình bày các nội dung cơ bản về cú pháp ngôn ngữ lập trình Java:

- Tất cả các lệnh của java phải được tổ chức vào trong một lớp nhất định. Tên tập tin mã nguồn phải trùng với tên lớp.
- Lệnh **package** được dùng để khai báo tên gói của lớp.
- Lệnh **import** được sử dụng trong chương trình để truy cập các gói thư viện Java.
- Lệnh class được dùng để khai báo tên lớp
- Tên lớp, tên phương thức, tên hằng và tên biến trong java phải tuân theo quy tắc đặt tên của java.
- Ứng dụng Java có một lớp chứa phương thức main. Các tham số có thể được truyền vào phương thức main nhờ các tham số lệnh (command line parameters).
- Java cung cấp các dạng toán tử:
    - Các toán tử số học
    - Các toán tử bit
    - Các toán tử quan hệ
    - Các toán tử logic
    - Toán tử điều kiện
    - Toán tử gán
- Java cung cấp các cấu trúc điều khiển lệnh:
    - if-else
    - switch
    - for
    - while
    - do while

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 3

1. Trong các tên sau, tên nào có thể dùng làm tên biến trong java:
    a. _123
    b. a$

c. 1abc
d. class
e. vi du
f. $123

2. Muốn lưu giữ một biến số nguyên dương mà có giá trị lớn nhất là một triệu thì dùng kiểu dữ liệu nào là tiết kiệm bộ nhớ nhất?
3. Muốn lưu giữ một biến số nguyên âm mà có giá trị nhỏ nhất là âm một tỉ thì dùng kiểu dữ liệu nào là tiết kiệm bộ nhớ nhất?
4. Trong cấu trúc lệnh if-else đơn (1 if và 1 else) thì có ít nhất một khối lệnh (của if hoặc của else) được thực hiện. Đúng hay sai?
5. Trong cấu trúc lệnh switch-case, khi không dùng "default" thì có ít nhất một khối lệnh được thực hiện. Đúng hay sai?
6. Trong cấu trúc lệnh switch-case, khi dùng "default" thì có ít nhất một khối lệnh được thực hiện. Đúng hay sai?
7. Trong cấu trúc lệnh while, khối lệnh được thực hiện ít nhất một lần ngay cả khi điều kiện có giá trị False. Đúng hay sai?
8. Trong cấu trúc lệnh do-while, khối lệnh được thực hiện ít nhất một lần ngay cả khi điều kiện có giá trị False. Đúng hay sai?
9. Trong cấu trúc lệnh for, khối lệnh được thực hiện ít nhất một lần ngay cả khi điều kiện có giá trị False. Đúng hay sai?
10. Cho biết kết quả thu đươc khi thực hiện đoạn chương trình sau?

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

11. Cho biết đoạn chương trình sau thực hiện vòng lặp bao nhiêu lần và kết quả in ra là gì?

```
class me{
    public static void main(String args[]){
       int i = 0;
       int sum = 0;
       do{
          sum += i;
          i++;
       }while(i <= 10);
       System.out.println(sum);
       }
}
```

12. Cho biết đoạn chương trình sau thực hiện vòng lặp bao nhiêu lần và kết quả in ra là gì?

```
class me{
```

```
    public static void main(String args[]){
        int i = 5;
        int sum = 0;
        do{
            sum += i;
            i++;
        }while(i < 5);
        System.out.println(sum);
        }
}
```

13. Cho biết hai đoạn chương trình sau in ra kết quả giống hay khác nhau?

```
class me1{
    public static void main(String args[]){
        int i = 0;
        int sum = 0;
        for(i=0; i<5; i++){
            sum += i;
        }
        System.out.println(sum);
        }
}
```

và:

```
class me2{
    public static void main(String args[]){
        int i = 0;
        int sum = 0;
        for( ; i<5; i++){
            sum += i;
        }
        System.out.println(sum);
        }
}
```

14. Viết chương trình tính tổng các số chẵn nằm trong khoảng 1 đến 100.
15. Viết chương trình hiển thị tổng các bội số của 7 nằm giữa 1 và 100.
16. Viết chương trình tìm giai thừa của n (n>0), n nhập từ tham số dòng lệnh.
17. Viết chương trình tìm bội số chung nhỏ nhất của m và n (m,n>0), m và n được nhập từ tham số dòng lệnh.
18. Viết chương trình tìm ước số chung lớn nhất của m và n (m,n>0), m và n được nhập từ tham số dòng lệnh.
19. Viết chương trình tìm số Fibonaci thứ n (n>2), n nhập từ tham số dòng lệnh. Biết rằng số Fibonaci được tính theo công thức: F(n) = F(n-1) + F(n-2) với n>=2 và F(0) = F(1) = 1.

CHƯƠNG 4
KẾ THỪA VÀ ĐA HÌNH TRÊN JAVA

Nội dung của chương này tập trung trình bày các đặc trưng hướng đối tượng của ngôn ngữ Java:

- Kế thừa đơn
- Kế thừa kép
- Các lớp trừu tượng
- Đa hình

4.1 KẾ THỪA ĐƠN

4.1.1 Lớp

Java coi lớp là một khuôn mẫu (Template) của một đối tượng, trong đó lớp chứa các thuộc tính và các phương thức hoạt động của đối tượng.

Khai báo lớp

Một lớp được khai báo với cú pháp:

```
<tính chất> class <tên lớp>
{
}
```

Lớp trong java có ba tính chất đặc trưng bởi ba từ khoá:

- **public**: Lớp thông thường, có thể được truy cập từ các gói (package) khác. **public** là giá trị mặc định cho tính chất của lớp.
- **final**: Khai báo lớp hằng, lớp này không thể tạo dẫn xuất. Tức là không có lớp nào kế thừa được từ các lớp có tính chất final.
- **abstract**: Khai báo lớp trừu tượng, lớp này chỉ được phép chứa các phương thức trừu tượng. Hơn nữa, không thể tạo các thể hiện (Instance) của các lớp trừu tượng bằng toán tử **new** như các lớp thông thường.

Chương trình 4.1 khai báo một lớp thông thường với kiểu mặc định là public với dòng khai báo.

Chương trình 4.1

```
package vidu.chuong4;
class Person
{
}
```

Sử dụng lớp

Lớp được sử dụng khi chương trình cần một đối tượng có kiểu của lớp đó. Khi đó, đối tượng được khai báo dựa vào toán tử **new**:

```
<tên lớp> <tên đối tượng> = new <tên lớp>();
```

Ví dụ, muốn tạo một đối tượng có kiểu là lớp Person trong chương trình 4.1, ta dùng lệnh sau:

```
Person myClass = new Person();
```

Khai báo thuộc tính của lớp

Thuộc tính của lớp được khai báo theo cú pháp:

```
<tính chất> <kiểu dữ liệu> <tên thuộc tính>;
```

- **Kiểu dữ liệu**: có thể là các kiểu dữ liệu cơ bản sẵn có của java, có thể là các lớp do người dùng tự định nghĩa.
- **Tên thuộc tính**: được đặt tên theo quy tắc đặt tên biến của java.
- **Tính chất**: Các thuộc tính và phương thức của lớp có các tính chất được đặc trưng bởi các từ khoá sau (giá trị mặc định là **public**):
  - **public**: có thể được truy cập từ bên ngoài lớp định nghĩa.
  - **protected**: chỉ được truy cập từ lớp định nghĩa và các lớp kế thừa từ lớp đó.
  - **private**: chỉ được truy cập trong phạm vi bản thân lớp định nghĩa.
  - **static**: được dùng chung cho một thể hiện của lớp, có thể được truy cập trực tiếp bằng <tên lớp>.<tên thuộc tính> mà không cần khởi tạo một thể hiện của lớp.
  - **abstract**: định nghĩa một thuộc tính trừu tượng. Thuộc tính này không thể truy nhập trong lớp nhưng có thể bị định nghĩa chồng ở các lớp kế thừa.
  - **final**: một thuộc tính hằng, không bị định nghĩa chồng ở các lớp kế thừa.
  - **native**: dùng cho phương thức khi cài đặt phụ thuộc môi trường trong một ngôn ngữ khác, như C hay hợp ngữ.
  - **synchronized**: dùng cho phương thức tới hạn, nhằm ngăn các tác động của các đối tượng khác khi phương thức đang được thực hiện.

Chương trình 4.2 minh hoạ việc khai báo hai thuộc tính là tên và tuổi của lớp Người (Person).

Chương trình 4.2

```
package vidu.chuong4;
class Person
{
   public String  name;
   public int     age;
}
```

Khai báo phương thức của lớp

Phương thức của lớp được khai báo theo cú pháp

```
<tính chất> <kiểu trả về> <tên phương thức> ([<các tham số>])
      [throws <các ngoại lệ>]
{
}
```

- **Tính chất**: đặc trưng bởi các từ khoá tương tự như tính chất của thuộc tính. Giá trị mặc định là **public**.
- **Kiểu trả về**: Kiểu dữ liệu trả về của phương thức, có thể là kiểu dữ liệu sẵn có của java hoặc là kiểu do người dùng tự định nghĩa.
- **Tên phương thức**: tuân theo qui tắc đặt tên biến của java.
- **Các ngoại lệ**: là một đối tượng đặc biệt được tạo ra khi chương trình gặp lỗi. Java sẽ trả lại cho chương trình ngoại lệ này theo từ khoá **throws**. Các ngoại lệ, nếu có, được phân cách nhau bởi dấu phẩy.
- **Các tham số**: các tham số của phương thức, được liệt kê theo cặp `<kiểu tham số> <tên tham số>`, các tham số được phân biệt bởi dấu phẩy.

Chương trình 4.3 mô tả việc khai báo phương thức show() để hiển thị thông tin cá nhân của lớp Person.

Chương trình 4.3

```
package vidu.chuong4;
class Person
{
   public String  name;
   public int     age;

   public void show()
   {
      System.out.println( name + " is " + age + " years old!");
   }
}
```

Phương thức khởi tạo của lớp

Phương thức khởi tạo (Constructor) được dùng để khởi tạo một thể hiện cụ thể của một lớp, nghĩa là gán các giá trị khởi đầu cho các thuộc tính, nếu có, và tạo ra một đối tượng cụ thể. Phương thức khởi tạo phải cùng tên với lớp.

Lưu ý:

- Phương thức khởi tạo phải có tên trùng với tên của lớp
- Phương thức khởi tạo không có giá trị trả về
- Phương thức khởi tạo có tính chất public
- Có thể có nhiều phương thức khởi tạo của cùng một lớp

Chương trình 4.4a minh hoạ một phương thức khởi tạo của lớp Person bằng cách gán giá trị cho các thuộc tính tên và tuổi.

Chương trình 4.4a

```
package vidu.chuong4;
```

```
class Person
{
   public String  name;
   public int     age;

   // Phương thức khởi dựng
   public Person(String name1, int age1)
   {
      name = name1;
      age = age1;
   }

   public void show()
   {
      System.out.println( name + " is " + age + " years old!");
   }
}
```

Chương trình 4.4b minh hoạ cách dùng lớp Person mà chúng ta vừa định nghĩa trong chương trình 4.4a. Chương trình này sẽ tạo ra một đối tượng myPerson của lớp Person với các thuộc tính có giá trị khởi tạo: name = "Minh" và age = "21". Sau đó, chương trình sử dụng phương thức show() của đối tượng myPerson để in ra dòng thông báo "Minh is 21 years old!".

Chương trình 4.4b

```
package vidu.chuong4;
class PersonDemo
{
   public static void main(String args[])
   {
      Person myPerson = new Person("Minh", 21);
      myPerson.show();
   }
}
```

Biến this

Biến **this** là một biến ẩn đặc biệt luôn tồn tại trong các lớp java: một lớp có đúng một biến ẩn **this**. Biến này được sử dụng trong khi chạy và nó trỏ đến bản thân lớp chứa nó. Biến **this** thường được sử dụng trong các hàm khởi tạo của lớp.

Chương trình 4.4c khai báo một lớp hoàn toàn giống với lớp được khai báo trong chương trình 4.4a, nhưng chỉ khác là có dùng biến **this** trong hàm khởi tạo của lớp.

Chương trình 4.4c

```
package vidu.chuong4;
class Person
{
   public String  name;
   public int     age;

   // Phương thức khởi dựng
   public Person(String name, int age)
   {
      this.name = name;
      this.age = age;
   }

   public void show()
   {
      System.out.println( name + " is " + age + " years old!");
   }
}
```

Trong chương trình 4.4c, ta chú ý đến hàm khởi tạo của lớp, hàm này có hai biến cục bộ là name và age, trùng với các biến của lớp. Do đó, trong phạm vi hàm này, biến this.name và this.age sẽ chỉ các biến của lớp, còn các biến name và age sẽ chỉ các biến cục bộ của hàm. Cho nên, các lệnh gán vẫn thực thi như trong chương trình 4.4a.

4.1.2 Sự kế thừa

Sự kế thừa được sử dụng khi muốn tạo một lớp mới từ một lớp đã biết. Khi đó, tất cả các thuộc tính và phương thức của lớp cũ đều trở thành thuộc tính và phương thức của lớp mới. Lớp cũ được gọi là lớp cha, lớp mới được gọi là lớp con.

Khai báo lớp kế thừa

Khai báo lớp kế thừa được thực hiện bởi từ khoá **extends**:

```
<thuộc tính> <tên lớp con> extends <tên lớp cha>
{
}
```

Chương trình 4.5a minh hoạ việc tạo một lớp Nhân viên (Employee) được kế thừa từ lớp Person đã được xây dựng trong phần 4.1.1.

Chương trình 4.5a

```
package vidu.chuong4;
class Employee extends Person
{
```

```
    public float salary;

    // Phương thức khởi dựng
    public Employee(String name, int age, float salary)
    {
        super(name, age);
        this.salary = salary;
    }
}
```

Khi đó, đoạn chương trình của chương trình 4.5b vẫn in ra dòng thông báo "Minh is 21 years old!" vì khi đó đối tượng myEmployee gọi đến phương thức show() được kế thừa từ lớp Person.

Chương trình 4.5b

```
package vidu.chuong4;
class EmployeeDemo1
{
   public static void main(String args[])
   {
      Employee myEmployee = new Employee("Minh", 21, 300f);
      myEmployee.show();
   }
}
```

Khai báo phương thức nạp chồng

Khi muốn thay đổi nội dung của các phương thức được kế thừa từ lớp cha, ta dùng cách khai báo phương thức nạp chồng. Thực ra là khai báo lại một phương thức mới có cùng tên và kiểu với một phương thức đã có trong lớp cha.

Chương trình 4.6a sẽ khai báo nạp chồng phương thức show() của lớp Employee mà không dùng lại phương thức show() của lớp Person nữa.

Chương trình 4.6a

```
package vidu.chuong4;
class Employee extends Person
{
   public float salary;

   // Phương thức khởi dựng
   public Employee(String name, int age, float salary)
   {
      super(name, age);
```

```
        this.salary = salary;
    }

    // Khai báo nạp chồng
    public void show()
    {
        System.out.println( name + "has a salary of"
            + salary + "$/month");
    }
}
```

Khi đó, đoạn chương trình 4.6b sẽ in ra dòng thông báo "Minh has a salary of 300$/month" thay vì dòng thông báo "Minh is 21 years old!" như trong chương trình 4.5b. Lí do là lúc này, đối tượng myEmployee sẽ gọi phương thức show() của lớp Employee mà không gọi phương thức show() của lớp Person nữa.

Chương trình 4.6b

```
package vidu.chuong4;
class EmployeeDemo2
{
   public static void main(String args[])
   {
      Employee myEmployee = new Employee("Minh", 21, 300f);
      myEmployee.show();
   }
}
```

Quy tắc truy nhập trong kế thừa

Các quy tắc này quy định khả năng truy nhập của lớp con đối với các thuộc tính và phương thức của lớp cha:

- **private**: chỉ được truy nhập trong phạm vi lớp cha, lớp con không truy nhập được. Tất cả các lớp ngoài lớp cha đều không truy nhập được.
- **protected**: lớp con có thể truy nhập được. Tất cả các lớp không kế thừa từ lớp cha đều không truy nhập được.
- **final**: lớp con có thể sử dụng được nhưng không thể khai báo nạp chồng được.
- **public**: lớp con có thể sử dụng và nạp chồng được. Tất cả các lớp bên ngoài đều sử dụng được.

4.2 KẾ THỪA BỘI

Nhằm tránh những nhập nhằng của tính chất đa kế thừa của C++, Java không cho phép kế thừa trực tiếp từ nhiều hơn một lớp cha. Nghĩa là Java không cho phép đa kế thừa trực tiếp, nhưng cho

phép cài đặt nhiều giao tiếp (Interface) để có thể thừa hưởng thêm các thuộc tính và phương thức của các giao tiếp đó.

4.2.1 Giao tiếp

Khai báo giao tiếp

Cú pháp khai báo một giao tiếp như sau:

```
[public] interface <tên giao tiếp> [extends <danh sách giao tiếp>]
{
}
```

- **Tính chất**: tính chất của một giao tiếp luôn là **public**. Nếu không khai báo tường minh thì giá trị mặc định cũng là **public**.
- **Tên giao tiếp**: tuân thủ theo quy tắc đặt tên biến của java.
- **Danh sách các giao tiếp**: danh sách các giao tiếp cha đã được định nghĩa để kế thừa, các giao tiếp cha được phân cách nhau bởi dấu phẩy. (Phần trong ngoặc vuông "[]" là tuỳ chọn).

Lưu ý:

- Một giao tiếp chỉ có thể kế thừa từ các giao tiếp khác mà không thể được kế thừa từ các lớp sẵn có.

Chương trình 4.7 minh hoạ việc khai báo một giao tiếp không kế thừa từ bất kì một giao tiếp nào.

Chương trình 4.7

```
package vidu.chuong4;
public interface Product
{
}
```

Khai báo phương thức của giao tiếp

Cú pháp khai báo một phương thức của giao tiếp như sau:

```
[public] <kiểu giá trị trả về> <tên phương thức> ([<các tham số>])
      [throws <danh sách ngoại lệ>];
```

- **Tính chất**: tính chất của một thuộc tính hay phương thức của giao tiếp luôn là **public**. Nếu không khai báo tường minh thì giá trị mặc định cũng là **public**. Đối với thuộc tính, thì chất chất luôn phải thêm là hằng (**final**) và tĩnh (**static**).
- **Kiểu giá trị trả về**: có thể là các kiểu cơ bản của java, cũng có thể là kiểu do người dùng tự định nghĩa (kiểu đối tượng).
- **Tên phương thức**: tuân thủ theo quy tắc đặt tên phương thức của lớp
- **Các tham số**: nếu có thì mỗi tham số được xác định bằng một cặp <kiểu tham số> <tên tham số>. Các tham số được phân cách nhau bởi dấu phẩy.
- **Các ngoại lệ**: nếu có thì mỗi ngoại lệ được phân cách nhau bởi dấu phẩy.

Lưu ý:

- Các phương thức của giao tiếp chỉ được khai báo dưới dạng mẫu mà không có cài đặt chi tiết (có dấu chấm phẩy ngay sau khai báo và không có phần cài đặt trong dấu "{}"). Phần cài đặt chi tiết của các phương thức chỉ được thực hiện trong các lớp (class) sử dụng giao tiếp đó.
- Các thuộc tính của giao tiếp luôn có tính chất là hằng (final), tĩnh (static) và public. Do đó, cần gán giá trị khởi đầu ngay khi khai báo thuộc tính của giao tiếp.

Chương trình 4.8 minh hoạ việc khai báo một thuộc tính và một phương thức của giao tiếp Product đã được khai báo trong chương trình 4.7: thuộc tính lưu nhãn hiệu của nhà sản xuất sản phẩm; phương thức dùng để truy xuất giá bán của sản phẩm.

Chương trình 4.8

```
package vidu.chuong4;
public interface Product
{
   public static final String MARK = "Adidas";
   public float getCost();
}
```

4.2.2 Sử dụng giao tiếp

Vì giao tiếp chỉ được khai báo dưới dạng các phương thức mẫu và các thuộc tính hằng nên việc sử dụng giao tiếp phải thông qua một lớp có cài đặt giao tiếp đó. Việc khai báo một lớp có cài đặt giao tiếp được thực hiện thông qua từ khoá **implements** như sau:

```
<tính chất> class <tên lớp> implements <các giao tiếp>
{
}
```

- **Tính chất** và **tên lớp** được sử dụng như trong khai báo lớp thông thường.
- **Các giao tiếp**: một lớp có thể cài đặt nhiều giao tiếp. Các giao tiếp được phân cách nhau bởi dấu phẩy. Khi đó, lớp phải cài đặt cụ thể tất cả các phương thức của tất cả các giao tiếp mà nó sử dụng.

Lưu ý:

- Một phương thức được khai báo trong giao tiếp phải được cài đặt cụ thể trong lớp có cài đặt giao tiếp nhưng không được phép khai báo chồng. Nghĩa là số lượng các tham số của phương thức trong giao tiếp phải được giữ nguyên khi cài đặt cụ thể trong lớp.

Chương trình 4.9 minh hoạ việc cài đặt một lớp giày (Shoe) cài đặt giao tiếp Product với các thuộc tính và phương thức đã được khai báo trong chương trình 4.8.

Chương trình 4.9

```
package vidu.chuong4;
public class Shoe implements Product
{
```

```
    // Cài đặt phương thức được khai báo trong giao tiếp
    public float getCost()
    {
        return 10f;
    }

    // Phương thức truy nhập nhãn hiệu sản phẩm
    public String getMark()
    {
        return MARK;
    }

    // Phương thức main
    public static void main(String args[])
    {
        Shoe myShoe = new Shoe();
        System.out.println("This shoe is " + myShoe.getMark() +
                    " having a cost of $" + myShoe.getCost());
    }
}
```

Chương trình 4.9 sẽ in ra dòng: "This shoe is Adidas having a cost of $10". Hàm getMark() sẽ trả về nhãn hiệu của sản phẩm, là thuộc tính đã được khai báo trong giao tiếp. Hàm getCost() là cài đặt riêng của lớp Shoe đối với phương thức đã được khai báo trong giao tiếp Product mà nó sử dụng, cài đặt này trả về giá trị 10 đối với lớp Shoe.

4.3 LỚP TRỪU TƯỢNG

Lớp trừu tượng là một dạng lớp đặc biệt, trong đó các phương thức chỉ được khai báo ở dạng khuôn mẫu (template) mà không được cài đặt chi tiết. Việc cài đặt chi tiết các phương thức chỉ được thực hiện ở các lớp con kế thừa lớp trừu tượng đó.

Lớp trừu tượng được sử dụng khi muốn định nghĩa một lớp mà không thể biết và định nghĩa ngay được các thuộc tính và phương thức của nó.

4.3.1 Khai báo

Khai báo lớp trừu tượng

Lớp trừu tượng được khái báo như cách khai báo các lớp thông thường, ngoại trừ có thêm từ khoá **abstract** trong phần tính chất:

```
[public] abstract class <tên lớp>
{
}
```

- **Tính chất**: mặc định là **public**, bắt buộc phải có từ khoá **abstract** để xác định đây là một lớp trừu tượng.

- **Tên lớp**: tuân thủ theo quy tắc đặt tên lớp thông thường của java.

Lưu ý:

- Lớp trừu tượng cũng có thể kế thừa một lớp khác, nhưng lớp cha cũng phải là một lớp trừu tượng. (Khai báo kế thừa thông qua từ khoá **extends** như khai báo kế thừa thông thường).

Chương trình 4.10 khai báo một lớp trừu tượng là lớp động vật (Animal) một cách chung chung.

Chương trình 4.10

```
package vidu.chuong4;
abstract class Animal
{
}
```

Khai báo phương thức của lớp trừu tượng

Tất cả các thuộc tính và phương thức của lớp trừu tượng đều phải khai báo là trừu tượng. Hơn nữa, các phương thức của lớp trừu tượng chỉ được khai báo ở dạng khuôn mẫu mà không có phần khai báo chi tiết. Cú pháp khai báo phương thức của lớp trừu tượng:

```
[public] abstract <kiểu dữ liệu trả về> <tên phương thức>
      ([<các tham số>]) [throws <các ngoại lệ>];
```

- **Tính chất**: tính chất của một thuộc tính hay phương thức của lớp trừu tượng luôn là **public**. Nếu không khai báo tường minh thì giá trị mặc định cũng là **public**.
- **Kiểu dữ liệu trả về**: có thể là các kiểu cơ bản của java, cũng có thể là kiểu do người dùng tự định nghĩa (kiểu đối tượng).
- **Tên phương thức**: tuân thủ theo quy tắc đặt tên phương thức của lớp
- **Các tham số**: nếu có thì mỗi tham số được xác định bằng một cặp <kiểu tham số> <tên tham số>. Các tham số được phân cách nhau bởi dấu phẩy.
- **Các ngoại lệ**: nếu có thì mỗi ngoại lệ được phân cách nhau bởi dấu phẩy.

Lưu ý:

- Tính chất của phương thức trừu tượng không được là **private** hay **static**. Vì phương thức trừu tượng chỉ được khai báo chi tiết (nạp chồng) trong các lớp dẫn xuất (lớp kế thừa) của lớp trừu tượng. Do đó, nếu phương thức là **private** thì không thể nạp chồng, nếu phương thức là **static** thì không thể thay đổi trong lớp dẫn xuất.
- Phương thức trừu tượng chỉ được khai báo dưới dạng khuôn mẫu nên không có phần dấu móc "{}" mà kết thúc bằng dấu chấm phẩy "**;**".

Chương trình 4.11 khái báo hai phương thức của lớp trừu tượng Animal trong chương trình 4.10: Phương thức getName() trả về tên loài động vật, dù chưa biết tên cụ thể loài nào nhưng kiểu trả về là String. Phương thức getFeet() trả về số chân của loài động vật, cũng chưa biết cụ thể là bao nhiêu chân nhưng kiểu trả về là int.

Chương trình 4.11

```
package vidu.chuong4;
abstract class Animal
{
   abstract String getName();
   abstract int getFeet();
}
```

4.3.2 Sử dụng lớp trừu tượng

Lớp trừu tượng được sử dụng thông qua các lớp dẫn xuất của nó. Vì chỉ có các lớp dẫn xuất mới cài đặt cụ thể các phương thức được khai báo trong lớp trừu tượng.

Chương trình 4.12a khai báo lớp về loài chim (Bird) kế thừa từ lớp Animal trong chương trình 4.11. Lớp này cài đặt chi tiết hai phương thức đã được khai báo trong lớp Animal: phương thức getName() sẽ trả về tên loài là "Bird"; phương thức getFeet() trả về số chân của loài chim là 2.

Chương trình 4.12a

```
package vidu.chuong4;
public class Bird extends Animal
{
   // Trả về tên loài chim
   public String getName()
   {
      return "Bird";
   }

   // Trả về số chân của loài chim
   public int getFeet()
   {
      return 2;
   }
}
```

Chương trình 4.12b khai báo lớp về loài mèo (Cat) cũng kế thừa từ lớp Animal trong chương trình 4.11. Lớp này cài đặt chi tiết hai phương thức đã được khai báo trong lớp Animal: phương thức getName() sẽ trả về tên loài là "Cat"; phương thức getFeet() trả về số chân của loài mèo là 4.

Chương trình 4.12b

```
package vidu.chuong4;
public class Cat extends Animal
{
   // Trả về tên loài mèo
```

```
    public String getName()
    {
        return "Cat";
    }

    // Trả về số chân của loài mèo
    public int getFeet()
    {
        return 4;
    }
}
```

Chương trình 4.12c sử dụng lại hai lớp Bird và Cat trong các chương trình 4.12a và 4.12b. Chương trình này sẽ hiển thị hai dòng thông báo:

```
The Bird has 2 feets
The Cat has 4 feets
```

Chương trình 4.12c

```
package vidu.chuong4;
public class AnimalDemo
{
    public static void main(String args[])
    {
        Bird myBird = new Bird();
        System.out.println("The " + myBird.getName() + " has "
                    + myBird.getFeet() + " feets");
        Cat myCat = new Cat();
        System.out.println("The " + myCat.getName() + " has "
                    + myCat.getFeet() + " feets");
    }
}
```

4.4 ĐA HÌNH

4.4.1 Nạp chồng

Java cho phép trong cùng một lớp, có thể khai báo nhiều phương thức có cùng tên. Nạp chồng là hiện tượng các phương thức có cùng tên. Có hai kiểu nạp chồng trong Java:

- Các phương thức của cùng một lớp có cùng tên. Khi hai phương thức của một lớp có cùng tên thì bắt buộc chúng phải có:
  - Hoặc danh sách các tham số khác nhau
  - Hoặc kiểu trả về khác nhau

- Hoặc kết hợp hai điều kiện trên.

Nếu không, java sẽ không phân biệt được chúng. Ví dụ nếu trong cùng một lớp:

```
// Chấp nhận được
public int add(int x, int y){…}
public float add(float x, int y){…}

// Không chấp nhận được
public int add(int x, int y){…}
public int add(int x, int y){…}
```

- Phương thức của lớp con có cùng tên với phương thức của lớp cha. Trong trường hợp này, các phương thức nạp chồng có thể có cùng danh sách tham số và có cùng kiểu trả về (xem lại phần khai báo phương thức nạp chồng trong mục kế thừa 4.1.2).

4.4.2 Đa hình

Đa hình là việc triệu gọi đến các phương thức nạp chồng của đối tượng. Khi một phương thức nạp chồng được gọi, chương trình sẽ dựa vào kiểu các tham số và kiểu trả về để gọi phương thức của đối tượng cho phù hợp.

Chương trình 4.13 minh hoạ việc khai báo nhiều hàm add() để cộng hai số hoặc cộng hai xâu kí tự.

Chương trình 4.13

```
package vidu.chuong4;
public class Operator
{
   // Cộng hai số nguyên
   public int add(int x, int y)
   {
      return (x + y);
   }

   // Cộng hai số thực
   public float add(float x, float y)
   {
      return (x + y);
   }

   // Cộng hai chuỗi kí tự
   public String add(String a, String b)
   {
      return (a + b);
   }

   // Hàm main
```

```
    public static void main(String args[])
    {
        Operator myOperator = new Operator();
        System.out.println("The (5+19) is " + myOperator.add(5, 19));
        System.out.println("The (\"ab\" + \"cd\") is \""
              + myOperator.add("ab", "cd") + "\"");
    }
}
```

Chương trình 4.13 sẽ hiển thị ra hai dòng thông báo:

```
The (5+19) is 24
The ('ab' + 'cd') is 'abcd'
```

Trong lớp Operator có hai phương thức cùng tên và cùng có hai tham số đầu vào là add(). Khi chương trình thực thi lệnh myOperator.add(5, 19), chương trình sẽ tự đối chiếu các kiểu tham số, thấy 5 và 19 có dạng gần với kiểu int nhất, nên phương thức add(int, int) sẽ được gọi và trả về giá trị là 24.

Khi chương trình thực thi lệnh myOperator.add("ab", "cd"), chương trình sẽ tự đối chiếu các kiểu tham số, thấy 'ab' và 'cd' có dạng gần với kiểu String nhất, nên phương thức add(String, String) sẽ được gọi và trả về giá trị là 'abcd'.

Lưu ý:

- Khi gọi hàm với các kiểu dữ liệu khác với các hàm đã được khai báo, sẽ có sự chuyển đổi kiểu ngầm định diễn ra. Khi không thể thực hiện chuyển đổi kiểu ngầm định, java sẽ phát sinh một thông báo lỗi.

Chẳng hạn, trong chương trình 4.13, nếu ta thực thi lệnh myOperator.add(4.0f, 5) có dạng add(float, int), chương trình sẽ chuyển ngầm định số nguyên 5 thành float (chuyển từ kiểu int sang float thuộc diện nới rộng kiểu, là kiểu chuyển ngầm định trong java) để có thể sử dụng dạng được khai báo add(float, float) và kết quả sẽ là 9.0f.

Nếu ta thực thi lệnh myOperator.add('ab', 5) có dạng add(String, int), vì int không thể chuyển ngầm định thành String nên lệnh này sẽ phát sinh lỗi. Để tránh lỗi này, phải chuyển đổi kiểu tường minh cho số 5 thành kiểu String bằng một trong các cách sau:

```
myOperator.add("ab", (new Int(5)).toString());
myOperator.add("ab", 5 + "");
```

4.5 CASE STUDY II

Trong phần này, chúng ta sẽ viết một chương trình quản lí nhân viên của một công ty. Bao gồm các lớp chính:

- Lớp Human là một lớp trừu tượng, chỉ có một phương thức duy nhất là show().
- Lớp Person là lớp kế thừa từ lớp Human, có hai thuộc tính là tên (name) và tuổi (age). Để đóng gói dữ liệu các thuộc tính này có dạng private và các phương thức truy nhập chúng (get và set). Ngoài ra lớp này còn cài đặt phương thức show() kế thừa từ lớp trừu tượng Human.

- Lớp Employee là lớp kế thừa từ lớp Person, có thêm thuộc tính là lương (salary). Thuộc tính này cũng có dạng private để đóng gói dữ liệu và cần các phương thức truy nhập get/set. Lớp này cài đặt lại phương thức show(). Hơn nữa, lớp Employee còn có thêm hai phương thức addSalary() và addSalary(float) để tính tăng lương cho nhân viên: một phương thức tăng lương theo tỉ lệ mặc định là 10% (không cần tham số), và một phương thức tăng theo giá trị cụ thể đưa vào (cần tham số).

Các phần tiếp theo sẽ trình bày phần cài đặt chi tiết cho các lớp này.

4.5.1 Lớp Human

Lớp Human là một lớp trừu tượng, chỉ có một phương thức duy nhất là show(). Đây là nội dung tập tin Human.java.

Chương trình 4.14a

```
package vidu.chuong4;
abstract class Human
{
   abstract void show();
}
```

4.5.2 Lớp Person

Lớp Person là lớp kế thừa từ lớp Human:

- Có hai thuộc tính là tên (name) và tuổi (age) có dạng private
- Các phương thức truy nhập các thuộc tính name (getName() và setName(String)) và age (getAge() và setAge(int)).
- Cài đặt chồng phương thức show() kế thừa từ lớp trừu tượng Human.

Đây là nội dung tập tin Person.java.

Chương trình 4.14b

```
package vidu.chuong4;
class Person extends Human
{
   private String name;
   private int age;

   // Phương thức khởi dựng không có tham số
   public Person()
   {
      super();
      name = "";
      age = 0;
   }
```

```
    // Phương thức khởi dựng có tham số
    public Person(String name, int age)
    {
        this.name = name;
        this.age = age;
    }

    /* Phương thức truy nhập thuộc tính name */
    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    /* Phương thức truy nhập thuộc tính age */
    public int getAge()
    {
        return age;
    }

    public void setAge(int age)
    {
        this.age = age;
    }

    // Khai báo nạp chồng
    public void show()
    {
        System.out.println( name + " is " + age + " years old!");
    }
}
```


4.5.3 Lớp Employee

Lớp Employee là lớp kế thừa từ lớp Person:

- Có thêm thuộc tính là lương (salary) cũng có dạng private để đóng gói dữ liệu và cần các phương thức truy nhập get/set.
- Lớp này cài đặt lại phương thức show().

- Có thêm hai phương thức addSalary() và addSalary(float) để tính tăng lương cho nhân viên: phương thức addSalary() tăng lương theo tỉ lệ mặc định là 10% (không cần tham số), phương thức addSalary(float) tăng theo giá trị cụ thể đưa vào (cần tham số).

Sau đây là nội dung tập tin Employee.java

Chương trình 4.14c

```
package vidu.chuong4;
class Employee extends Person
{
   private float salary;

   // Phương thức khởi dựng không có tham số
   public Employee()
   {
      super();
      salary = 0f;
   }

   // Phương thức khởi dựng có tham số
   public Employee(String name, int age, float salary)
   {
      super(name, age);
      this.salary = salary;
   }

   /* Phương thức truy nhập thuộc tính salarry */
   public float getSalary()
   {
      return salary;
   }

   public void setSalary(float salary)
   {
      this.salary = salary;
   }

   // Khai báo nạp chồng
   public void show()
   {
      System.out.println( getName() + " is " + getAge()
                  + " years old having a salary of $"
                  + salary + "/month!");
   }
```

```
   /* Phương thức tăng lương */
   public void addSalary()
   {
      salary = salary*1.1f;
   }

   public void addSalary(float addition)
   {
      salary += addition;
   }
}
```

Lưu ý: Trong phương thức nạp chồng show() của lớp Employee, ta phải dùng các phương thức public được kế thừa từ lớp Person là getName() và getAge() để truy nhập đến thuộc tính name và age mà không thể truy xuất trực tiếp. Lí do là các thuộc tính name và age có dạng private trong lớp Person nên không thể truy xuất trực tiếp trong các lớp dẫn xuất. Do đó, ta phải truy xuất chúng thông qua các phương thức truy nhập public của lớp Person.

4.5.4 Chương trình demo

Chương trình 4.14d chứa hàm main để chạy minh hoạ việc sử dụng các lớp Person và Employee. Đây là nội dung của tập tin Casestudy2.java

Chương trình 4.14d

```
package vidu.chuong4;
public class Casestudy2
{
   // Hàm main
   public static void main(String args[])
   {
      // Sử dụng lớp Person
      Person myPerson = new Person("Vinh", 25);
      myPerson.show();

      // Sử dụng lớp Employee
      Employee myEmployee = new Employee("Vinh", 25, 300f);
      myEmployee.show();
      // Tăng lương theo mặc định
      myEmployee.addSalary();
      myEmployee.show();
      // Tăng lương lên $50
      myEmployee.addSalary(50f);
      myEmployee.show();
```

```
    }
}
```

Chương trình 4.14d sẽ hiển thị nội dung như sau:

```
Vinh is 25 years old!
Vinh is 25 years old having a salary of  $300/month!
Vinh is 25 years old having a salary of  $330/month!
Vinh is 25 years old having a salary of  $380/month!
```

Dòng thứ nhất in ra dòng thông báo theo phương thức show() của lớp Person. Dòng thứ hai cũng hiển thị thông báo theo phương thức show() của lớp Employee đã được khai báo nạp chồng. Dòng thứ ba hiển thị thông báo sau khi đã tăng lương theo mặc định (10%) từ 300$ lên 330$. Dòng thứ tư hiển thị thông báo sau khi tăng lương thêm một lần nữa với lượng tăng là 50$ từ 330$ lên 380$.

TỔNG KẾT CHƯƠNG 4

Nội dung chương 4 đã tập trung trình bày các vấn đề cơ bản liên quan đến kỹ thuật lập trình hướng đối tượng trong Java:

- Một lớp trong java được khai báo với từ khoá class, với các tính chất có thể có là public, final hoặc abstract.
- Khi khai báo một lớp là trừu tượng thì các phương thức của nó cũng được khai báo là trừu tượng và chỉ khai báo dạng khuôn mẫu mà không được khai báo chi tiết.
- Không thể tạo ra một đối tương từ một lớp trừu tượng.
- Có thể khai báo một lớp kế thừa từ một lớp khác thông qua từ khoá extends. Một lớp java chỉ có thể được kế thừa từ nhiều nhất một lớp cha mà thôi.
- Trong kế thừa, lớp con có thể định nghĩa lại (nạp chồng) các phương thức được kế thừa từ lớp cha.
- Một giao tiếp trong java được khai báo thông qua từ khoá interface. Các thuộc tính của giao tiếp phải là thuộc tính hằng (final) và static. Các phương thức chỉ được khai báo dưới dạng khuôn mẫu mà không được cài đặt chi tiết.
- Có thể khai báo một giao tiếp kế thừa từ một giao tiếp khác cũng bằng từ khoá extends. Một giao tiếp trong java có thể được kế thừa từ nhiều giao tiếp khác.
- Một giao tiếp trong java chỉ được sử dụng thông qua một lớp java cài đặt cụ thể giao tiếp đó. Khi đó, lớp java phải cài đặt chi tiết tất cả các phương thức được khai báo trong giao tiếp.
- Trong một lớp java, có thể khai báo nhiều phương thức có cùng tên. Khi thực thi, chương trình sẽ tự động chọn phương thức có khuôn mẫu kiểu tham số trùng với lệnh để thực hiện lệnh tương ứng.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 4

1. Trong các khai báo lớp sau, khai báo nào là đúng:

   a. `class myClass`

   b. `Class myClass`

c. `public class MyClass`

d. `Public class MyClass`

2. Trong các khai báo phương thức của một lớp (không trừu tượng) sau, khai báo nào là đúng:

a. `String getName{return name;}`

b. `String getName(){return name;}`

c. `String getName(){return name;};`

d. `String getName();`

3. Trong các khai báo phương thức của một lớp trừu tượng sau, khai báo nào là đúng:

a. `abstract String getName{return name;}`

b. `abstract String getName(){return name;}`

c. `abstract String getName(){return name;};`

d. `abstract String getName();`

4. Trong các khai báo một giao tiếp sau, khai báo nào là đúng:

a. `public abstract MyInterface{…}`

b. `public class MyInterface{…}`

c. `public interface MyInterface{…}`

d. `public interface MyInterface();`

5. Trong các khai báo kế thừa lớp sau, giả sử Aclass và Bclass là các lớp có sẵn, khai báo nào là đúng:

a. `public class MyClass extends Aclass{…}`

b. `public class MyClass Extends Aclass{…}`

c. `public class MyClass extends Aclass, Bclass{…}`

d. `public class MyClass extends Aclass, extends Bclass{…}`

6. Trong các khai báo kế thừa giao tiếp sau, giả sử Ainterface và Binterface là các giao tiếp có sẵn, khai báo nào là đúng:

a. `public interface MyInterface extends Ainterface{…}`

b. `public interface MyInterface Extends Ainterface{…}`

c. `public interface MyInterface extends Ainterface, Binterface{…}`

d. `public interface MyInterface extends Ainterface, extends Binterface{…}`

7. Trong các khai báo một lớp sử dụng giao tiếp sau, giả sử Ainterface và Binterface là các giao tiếp có sẵn, khai báo nào là đúng:

a. `public class MyClass extends Ainterface{…}`

b. `public class MyClass implements Ainterface{…}`

c. `public class MyClass implements Ainterface, Binterface{…}`

d. `public class MyClass implements Ainterface, implements Binterface{…}`

8. Xét đoạn chương trình cài đặt một lớp sử dụng một giao tiếp sau:

```
public interface MyInterface{
public void show(String a);
```

```
}
// Khai báo lớp sử dụng giao tiếp
public class MyClass implements MyInterface{
…
}
```

Khi đó, lớp MyClass chỉ có duy nhất một phương thức cài đặt chi tiết phương thức show của giao tiếp MyInterface. Trong các cách cài đặt sau, cách nào là đúng:

a. `public int show(String a){ System.out.println(a);}`

b. `public void show(){ System.out.println("Show!");}`

c. `public void show(String a){ System.out.println(a);}`

d. `public void show(String a, String b){ System.out.println(a+b);}`

9. Xét đoạn chương trình cài đặt phép toán cộng với các số hạng có kiểu khác nhau:

```
public class MyOperator {
      public static int add(int a, int b){return a+b;}
      public static float add(float a, float b){return a+b;}
      public static double add(double a, double b){return a+b;}
      public static String add(String a, String b){return a+b;}
}
```

Khi ta gọi lệnh MyOperator.add(-5, 100), chương trình sẽ gọi phương thức nào?

a. `add(int, int)`

b. `add(float, float)`

c. `add(double, double)`

d. `add(String, String)`

e. `Thông báo lỗi`

10. Cũng với lớp MyOperator trong bài 9, khi ta gọi lệnh MyOperator.add(5.0d, 100f), chương trình sẽ gọi phương thức nào?

a. `add(int, int)`

b. `add(float, float)`

c. `add(double, double)`

d. `add(String, String)`

e. `Thông báo lỗi`

11. Cũng với lớp MyOperator trong bài 9, khi ta gọi lệnh MyOperator.add("abcd", 100f), chương trình sẽ gọi phương thức nào?

a. `add(int, int)`

b. `add(float, float)`

c. `add(double, double)`

d. `add(String, String)`

e. `Thông báo lỗi`

12. Về sự khác nhau giữa lớp và giao tiếp trong java, những nhận định nào sau đây là đúng:

a. Lớp được kế thừa tối đa từ một lớp khác, giao tiếp có thể kế thừa từ nhiều giao tiếp khác.

b. Lớp có thể kế thừa từ giao tiếp, nhưng giao tiếp không được kế thừa từ các lớp.

c. Thuộc tính của lớp có thể có tính chất bất kì, thuộc tính của giao tiếp phải là hằng số và tĩnh.

d. Phương thức của lớp có thể được cài đặt chi tiết, phương thức của giao tiếp chỉ được khai báo ở dạng khuôn mẫu.

13. Xét đoạn chương trình khai báo hai lớp kế thừa nhau như sau:

```
public class Person {
public void show(){System.out.println("This is a person!");}
}
```

và:

```
public class Employee extends Person {
public void show(int x){
            System.out.println("This is the employee " + x);
      }
}
```

Khi đó, đoạn chương trình sau sẽ hiển thị thông báo nào?

```
public class Demo1 {
public static void main(String args[]){
            Employee myEmployee = new Employee();
            myEmployee.show();
      }
}
```

a. This is a person!.

b. This is the employee 0.

c. Thông báo lỗi.

d. Không hiển thi gì cả.

14. Cũng với hai lớp Person và Employee trong bài 14. Đoạn chương trình sau sẽ hiển thị thông báo nào?

```
public class Demo2 {
public static void main(String args[]){
            Employee myEmployee = new Employee();
            myEmployee.show(10);
      }
}
```

a. This is a person!.

b. This is the employee 10.

c. Thông báo lỗi.

d. Không hiển thi gì cả.

15. Viết một chương trình khai báo một lớp về các hình chữ nhật Rectangle. Lớp này có hai thuộc tính cục bộ là chiều rộng và chiều dài của hình. Viết hai phương thức khởi dựng tường minh cho lớp này: Một khởi dựng với một tham số kiểu int, khi đó, chiều rộng và chiều dài được thiết lập thành giá trị tham số đưa vào (hình vuông). Một khởi dựng với hai tham số kiểu int tương ứng là chiều rộng và chiều dài.
16. Viết một giao tiếp khai báo các phép toán cộng hai số, cộng hai xâu, cộng xâu và số. Sau đó viết một lớp cài đặt giao tiếp đó.
17. Viết một lớp trừu tượng về các hình phẳng, trong đó khai báo các phương thức tính chu vi và diện tích của hình. Sau đó viết ba lớp kế thừa từ lớp trừu tượng đó là: lớp hình vuông, lớp hình chữ nhật và lớp hình tròn. Tự thiết lập các thuộc tính cục bộ cần thiết cho mỗi lớp con và khai báo nạp chồng hai phương thức ban đầu trong mỗi lớp con.

CHƯƠNG 5
BIỂU DIỄN VÀ CÀI ĐẶT
CÁC CẤU TRÚC DỮ LIỆU TRỪU TƯỢNG TRÊN JAVA

Nội dung chương này tập trung trình bày việc cài đặt một số giải thuật và các cấu trúc dữ liệu trừu tượng trên java. Các giải thuật bao gồm:

- Phương pháp duyệt và đệ qui
- Phương pháp sắp xếp và tìm kiếm

Các cấu trúc dữ liệu trừu tượng bao gồm:

- Ngăn xếp và hàng đợi
- Danh sách liên kết
- Cây nhị phân
- Đồ thị

5.1 PHƯƠNG PHÁP DUYỆT VÀ ĐỆ QUI

5.1.1 Các phương pháp duyệt

Các phương pháp duyệt thường xuất hiện trong bài toán tổ hợp nhằm tìm kiếm tất cả các trạng thái (tổ hợp) có thể có của một tập các trạng thái, hoặc tìm ra một tổ hợp thoả mãn tốt nhất một số yêu cầu xác định trong số các trạng thái của tập hợp. Các phương pháp duyệt thông thường bao gồm:

- *Phương pháp duyệt toàn bộ:* Duyệt qua tất cả các trạng thái có thể có của tập hợp. Phương pháp này phù hợp với bài toán liệt kê và với các tập hợp có số lượng các trạng thái là đủ nhỏ để không tốn thời gian duyệt.
- *Phương pháp duyệt chọn lọc:* Thường áp dụng khi số lượng trạng thái của tổ hợp là rất lớn, không thể sử dụng phương pháp duyệt toàn bộ. Phương pháp này chỉ dùng cho các bài toán tìm kiếm một (hoặc một số) trạng thái tốt nhất (theo một số tiêu chí xác định) của tổ hợp bằng cách chỉ duyệt theo một số trạng thái được cho là có khả năng tìm được kết quả cao nhất. Một số thuật toán duyệt theo phương pháp này thường được áp dụng trong lĩnh vực trí tuệ nhân tạo như thuật toán A*, thuật toán nhánh và biên, thuật toán α / β .

Ngoài ra, tuỳ thuộc vào cách thức biểu diễn dữ liệu của trạng thái (cấu trúc dữ liệu), người ta sử dụng các phương pháp duyệt khác nhau:

- Duyệt trên mảng
- Duyệt trên cây
- Duyệt trên đồ thị

Các phương pháp duyệt này sẽ được trình bày trong các mục tương ứng về các đối tượng danh sách tuyến tính (duyệt trên mảng), cây nhị phân (duyệt trên cây), đồ thị (duyệt trên đồ thị).

5.1.2 Phương pháp đệ qui

Phương pháp đệ qui: được định nghĩa theo qui nạp toán học, một đối tượng được biểu diễn qua chính nó với một phạm vi nhỏ hơn.

Ví dụ, định nghĩa số Fibonacy thứ n:

```
F(n) = F(n-1) + F(n-2) với n>2
F(1) = F(2) = 1
```

là một định nghĩa mang tính đệ qui.

Giải thuật đệ qui: là phương pháp giải bài toán bằng cách rút gọn bài toán thành một (hoặc một số) bài toán con tương tự như vậy nhưng với dữ liệu nhỏ hơn với trạng thái dừng tồn tại.

Ví dụ, thủ tục tính số Fibonacy thứ n:

```
int Fibo(int n){
   if(n == 1 || n == 2) return 1;
   return (Fibo(n-1) + Fibo(n-2));
}
```

là một giải thuật đệ qui. Nó tính số Fibonacy thứ n thông qua việc tính hai số nhỏ hơn trước nó là số thứ n-1 và n-2. Điều kiện dừng là số thứ 1 và số thứ 2 là 1.

5.2 PHƯƠNG PHÁP SẮP XẾP VÀ TÌM KIẾM

5.2.1 Các phương pháp sắp xếp

Hiện nay, có rất nhiều phương pháp sắp xếp khác nhau:

- Sắp xếp nổi bọt (bubble sort)
- Sắp xếp chèn (insertion sort)
- Sắp xếp chọn (selection sort)
- Sắp xếp vun đống (heap sort)
- Sắp xếp trộn (merge sort)
- Sắp xếp nhanh (quick sort)

Nội dung phần này sẽ trình bày việc cài đặt giải thuật sắp xếp nhanh (quick sort). Vì việc sắp xếp thường gắn liền với một mảng các phần tử có thể so sánh được, cho nên giải thuật này sẽ được cài đặt trong lớp mảng các phần tử (Array). Việc cài đặt các giải thuật sắp xếp còn lại được coi như một bài tập của phần này.

Lớp mảng các phần tử Array

Lớp này có thuộc tính là một mảng các phần tử. Các phần tử của mảng có thể có kiểu bất kì, nhưng phải thoả mãn điều kiện là có thể so sánh được, khi đó, mảng có thể sắp xếp được. Trong phần này, ta sẽ cài đặt mảng các phần tử có kiểu int.

```
class Array{
   private int *elements;
}
```

Giải thuật sắp xếp nhanh Quick Sort

Ý tưởng của giải thuật này là chọn một phần tử đóng vai trò là khoá chốt (còn gọi là điểm mốc), các phần tử nhỏ hơn khoá chốt sẽ phải chuyển lên đứng trước khoá chốt. Các phần tử lớn hơn khoá chốt thì phải chuyển xuống đứng sau khoá chốt. Các bước cụ thể như sau:

- Chọn một phần tử (bất kì) làm khoá chốt.
- Đi từ đầu mảng đến cuối mảng, tìm phần tử đầu tiên lớn hơn khoá chốt, đánh dấu nó là phần tử thứ i.
- Đi từ cuối mảng lên đầu mảng, tìm phần tử đầu tiên nhỏ hơn khoá chốt, đánh dấu nó là phần tử thứ j.
- Nếu i<j, đổi chỗ phần tử thứ i và thứ j.
- Sau đó, đi tiếp theo hai chiều, và đổi chỗ, nếu có, cho đến khi i=j.
- Đổi chỗ phần tử thứ i (cũng là j vào thời điểm này) với phần tử chốt. Khi đó, phần tử chốt là đúng vị trí của nó trong mảng sắp xếp.
- Lặp lại giải thuật trên với hai đoạn của mảng: đoạn trước chốt và đoạn sau chốt.
- Quá trình sẽ dừng khi mỗi đoạn chỉ còn hai phần tử.

Chương trình 5.1a cài đặt thủ tục sắp xếp nhanh của lớp Array.

Chương trình 5.1a

```
package vidu.chuong5;
class Array{
   private int[] elements;

   /* Phương thức truy nhập các phần tử của mảng */
   public int[] get(){
      return elements;
   }
   public void set(int[] elements){
      this.elements = elements;
   }

   /* Phương thức săp xếp */
   public void sort(){
      quick(0, elements.length-1);
   }

   /* Phương thức sắp xếp nhanh */
   private void quick(int left, int right){
      int i=left, j=right;
      int pivot=(left+right)/2, tmp;
      do{
         while(elements[i]<elements[pivot] && i<right)i++;   // Quét xuôi
```

```
            while(elements[j]>elements[pivot] && j>left)j++;    // Quét ngược
            if(i<=j){                          // Đổi chỗ hai phần tử
               tmp = elements[i];
               elements[i] = elements[j];
               elements[j] = tmp;
            }
         }while (i<=j);
         if(left < j)    quick(left, j);       // Sắp xếp đoạn trước chốt
         if(i < right)   quick(i, right);      // Sắp xếp đoạn sau chốt
      }
   }
```

5.2.2 Các phương pháp tìm kiếm

Phương pháp tìm kiếm sẽ trả về chỉ số của một phần tử nếu nó có mặt trong mảng tìm kiếm, trả về -1 nếu không có phần tử đó trong mảng. Các phương pháp tìm kiếm cơ bản bao gồm:

- Tìm kiếm tuần tự (tuyến tính)
- Tìm kiếm nhị phân
- Tìm kiếm trên cây

Nội dung phần này sẽ trình bày phương pháp tìm kiếm nhị phân. Các phương pháp còn lại được coi như là bài tập của phần này. Phương pháp tìm kiếm nhị phân được thực hiện trên mảng đã sắp xếp. Các bước tiến hành như sau:

- Lấy khoá cần tìm so sánh với phần tử ở giữa mảng đã sắp xếp. Nếu bằng, kết thúc tìm kiếm.
- Nếu nhỏ hơn, tìm kiếm khoá đó trong nửa đầu của mảng (vẫn theo kiểu nhị phân).
- Nếu lớn hơn, tìm kiếm khoá đó trong nửa sau của mảng (vẫn theo kiểu nhị phân).
- Quá trình kết thúc khi khoá bằng phần tử giữa mảng, hoặc các đoạn chỉ còn một phần tử.

Chương trình 5.1b cài đặt thủ tục tìm kiếm nhị phân trên lớp mảng Array với các phần tử có kiểu int (khoá tìm kiếm cũng có kiểu int).

Chương trình 5.1b

```
package vidu.chuong5;
class Array{
   private int[] elements;

   /* Phương thức truy nhập các phần tử của mảng */
   public int[] get(){
      return elements;
   }
   public void set(int[] elements){
      this.elements = elements;
```

```
    }

    /* Phương thức tìm kiếm */
    public int search(int key){
        quick(0, elements.length-1);          // Sắp xếp mảng, dùng quicksort
        int low=0, hight=elements.length-1, mid;
        while(low <= hight){
            mid = (low + hight)/2;            // Chỉ số giữa
            if(key > elements[mid])
                low = mid+1;                  // Tìm nửa sau
            else if(key < elements[mid])
                hight= mid-1;                 // Tìm nửa đầu
            else return mid;                  // Tìm thấy
        }
        return -1;                            // Không tìm thấy
    }

    /* Phương thức săp xếp */
    public void sort(){
        quick(0, elements.length-1);
    }

    /* Phương thức sắp xếp nhanh */
    private void quick(int left, int right){
        int i=left, j=right;
        int pivot=(left+right)/2, tmp;
        do{
            while(elements[i]<elements[pivot] && i<right)i++;   // Quét xuôi
            while(elements[j]>elements[pivot] && j>left)j++;    // Quét ngược
            if(i<=j){                         // Đổi chỗ hai phần tử
                tmp = elements[i];
                elements[i] = elements[j];
                elements[j] = tmp;
            }
        }while (i<=j);
        if(left < j)   quick(left, j);        // Sắp xếp đoạn trước chốt
        if(i < right)  quick(i, right);       // Sắp xếp đoạn sau chốt
    }
}
```

5.3 NGĂN XẾP VÀ HÀNG ĐỢI

5.3.1 Ngăn xếp

Ngăn xếp (stack) có các thuộc tính cục bộ:

- Mảng lưu các nút của ngăn xếp

Các thao tác đối với ngăn xếp:

- Thêm vào một nút
- Lấy ra một nút

Định nghĩa một nút

Để đơn giản, ta chỉ định nghĩa một nút có giá trị kiểu int:

Chương trình 5.2a

```
package vidu.chuong5;
public class Node{
   private int value;

   /* Các phương thức khởi dựng */
   public Node(){
      value = 0;
   }

   public Node(int value){
      this.value = value;
   }

   /* Phương thức truy nhập thuộc tính value */
   public int getValue(){
      return value;
   }
   public void setValue(int value){
      this.value = value;
   }
}
```

Cài đặt ngăn xếp

- Ta coi đỉnh ngăn xếp là cuối mảng lưu giữ các nút. Do đó, các thao tác thêm vào và lấy ra sẽ thêm vào cuối mảng hoặc lấy nút ở cuối mảng ra.
- Mảng các giá trị được khai báo động để tiết kiệm bộ nhớ.

Chương trình 5.2b

```
package vidu.chuong5;
public class MyStack{
   private Node[] values;

   /* Các phương thức khởi dựng */
   public MyStack(){}

   public MyStack(Node[] values){
      this.values = values;
   }

   /* Phương thức lấy ra một node từ stack */
   public Node pop(){
      Node result = null;
      if((values != null)&&(values.length > 0)){
         result = values[values.length - 1];
         // Loại bỏ node cuối cùng
         Node[] tmpNode = new Node[values.length - 1];
         for(int i=0; i<values.length - 1; i++)
            tmpNode[i] = values[i];
         this.values = tmpNode;
      }
      return result;
   }

   /* Phương thức thêm một node vào stack */
   public void push(Node node){
      if(values == null){ // Ngăn xếp đang rỗng
         values = new Node[1];
         values[0] = node;
      }else{      // Ngăn xếp đã có dữ liệu
         Node[] tmpNode = new Node[values.length + 1];
         for(int i=0; i<values.length; i++)
            tmpNode[i] = values[i];
         tmpNode[values.length] = node;
         this.values = tmpNode;
      }
   }
}
```

5.3.2 Hàng đợi

Hàng đợi (queue) có các thuộc tính cục bộ:

- Mảng các giá trị trong hàng đợi

Các thao tác với hàng đợi:

- Thêm vào một nút vào cuối hàng đợi
- Lấy ra một nút từ đầu hàng đợi

Chương trình 5.3 cài đặt lớp hàng đợi.

Chương trình 5.3

```
package vidu.chuong5;
public class MyQueu{
   private Node[] values;

   /* Các phương thức khởi dựng */
   public MyQueu(){}

   public MyQueu(Node[] values){
      this.values = values;
   }

   /* Phương thức lấy ra một node từ đầu queu */
   public Node remove(){
      Node result = null;
      if((values != null)&&(values.length > 0)){
         result = values[0];
         // Loại bỏ node đầu hàng đợi
         Node[] tmpNode = new Node[values.length - 1];
         for(int i=0; i<values.length - 1; i++)
            tmpNode[i] = values[i+1];
         this.values = tmpNode;
      }
      return result;
   }

   /* Phương thức thêm một node vào cuối queu */
   public void insert(Node node){
      if(values == null){ // Hàng đợi đang rỗng
         values = new Node[1];
         values[0] = node;
      }else{      // Hàng đợi đã có dữ liệu
         Node[] tmpNode = new Node[values.length + 1];
         for(int i=0; i<values.length; i++)
```

```
            tmpNode[i] = values[i];
         tmpNode[values.length] = node;
         this.values = tmpNode;
      }
   }
}
```

5.4 DANH SÁCH LIÊN KẾT

Nội dung phần này tập trung cài đặt hai loại danh sách liên kết cơ bản:

- Danh sách liên kết đơn
- Danh sách liên kết kép

5.4.1 Danh sách liên kết đơn

Định nghĩa một nút của danh sách liên kết đơn

Một nút của danh sách liên kết đơn bao gồm:

- Giá trị của nút, có dạng là một đối tượng kiểu Node đã được định nghĩa trong chương trình 5.2a
- Nút tiếp theo của nút đó.

Một nút của danh sách liên kết đơn được cài đặt trong chương trình 5.4a.

Chương trình 5.4a

```
package vidu.chuong5;
public class SimpleNode{
   private Node value;     // Giá trị của node là một đối tượng kiểu Node
   private SimpleNode next;// Node tiếp theo của danh sách liên kết

   /* Các phương thức khởi dựng */
   public SimpleNode(){
      value = new Node();
      next = null;
   }
   public SimpleNode(Node value){
      this.value = value;
      next = null;
   }

   /* Phương thức truy nhập thuộc tính value */
   public Node getValue(){
      return value;
   }
```