Mục lục

GIỚI THIỆU

Phần mềm ngày càng lớn và phức tạp và đòi hỏi được cập nhật liên tục để đáp ứng những yêu cầu mới của người dùng. Phương pháp lập trình thủ tục truyền thống dần trở nên không đáp ứng được những đòi hỏi đó của ngành công nghiệp phần mềm. Lập trình hướng đối tượng đã ra đời trong bối cảnh như vậy để hỗ trợ sử dụng lại và phát triển các phần mềm qui mô lớn.

Giáo trình này cung cấp cho sinh viên các kiến thức từ cơ bản cho đến một số kỹ thuật nâng cao về phương pháp lập trình hướng đối tượng. Giáo trình dùng cho sinh viên ngành Công nghệ thông tin đã có kiến thức căn bản về lập trình. Giáo trình sử dụng ngôn ngữ lập trình Java để minh họa và đồng thời cũng giới thiệu một số kiến thức căn bản của ngôn ngữ này.

Các nội dung chính về phương pháp lập trình hướng đối tượng được trình bày trong giáo trình bao gồm lớp và đối tượng, đóng gói/che giấu thông tin, kế thừa và đa hình, xử lý ngoại lệ và lập trình tổng quát. Ngoài ra, giáo trình cũng trình bày các kiến thức về Java bao gồm các đặc trưng cơ bản của ngôn ngữ, các thư viện cơ bản và cách thức tổ chức vào/ra dữ liệu.

Thay vì cách trình bày theo tính hàn lâm về một chủ đề rộng, để thuận tiện cho giảng dạy, giáo trình chọn cách trình bày theo các bài học cụ thể được sắp xếp theo trình tự kiến thức từ cơ sở đến chuyên sâu. Mỗi chủ đề có thể được giảng dạy với thời lượng 2~3 giờ lý thuyết và giờ thực hành tương ứng. Ch-¬ng 2 và Ch-¬ng 6, với nội dung là các kiến thức cơ bản về ngôn ngữ lập trình Java, tuy cần thiết nhưng không phải nội dung trọng tâm của môn học Lập trình hướng đối tượng. Các chương này, do đó, nên để sinh viên tự học. Chương 9 và Chương 10 không nhất thiết phải được dạy thành những chủ đề độc lập mà có thể được tách rải rác các nội dung kiến thức và giới thiệu kèm theo các khái niệm hướng đối tượng có liên quan, hoặc yêu cầu sinh viên tự đọc khi cần đến các kiến thức này trong quá trình thực hành.

Tuy cuốn giáo trình này không trình bày sâu về lập trình Java, nhưng kiến thức về lập trình Java lại là cần thiết đối với sinh viên, ngay cả với mục đích thực hành môn học. Do đó, ngoài mục đích thực hành các nội dung liên quan đến lập trình hướng đối tượng, các bài tập thực hành của môn học này nên có thêm đóng vai trò định hướng và gợi ý giúp đỡ sinh viên tự học các chủ đề thuần túy Java mà giáo viên cho là cần thiết, chẳng hạn như học về vào ra dữ liệu đơn giản ngay từ tuần đầu tiên của môn học. Các định hướng này có thể được thể hiện ở những bài tập thực hành với những đoạn chương trình mẫu, hoặc yêu cầu tìm hiểu tài liệu API về một số lớp tiện ích. Một số bài tập cuối chương là ví dụ của dạng bài tập này.

Các thuật ngữ hướng đối tượng nguyên gốc tiếng Anh đã được chuyển sang tiếng Việt theo những cách khác nhau tùy các tác giả. Sinh viên cần biết thuật ngữ nguyên gốc tiếng Anh cũng như các cách dịch khác nhau đó để tiện cho việc sử dụng tài liệu tiếng Anh cũng như để liên hệ kiến thức giữa các tài liệu tiếng Việt. Vì lí do đó, giáo trình này cung cấp bảng thuật ngữ Anh-Việt với các cách dịch khác nhau tại Phụ lục C, bên cạnh Phụ lục A về công cụ lập trình JDK và Phụ lục B về tổ chức gói của ngôn ngữ Java.

Các tác giả chân thành cảm ơn PGS. TS. Nguyễn Đình Hóa, TS. Trương Anh Hoàng, TS. Cao Tuấn Dũng, TS. Đặng Đức Hạnh, cũng như các đồng nghiệp và sinh viên tại Khoa Công nghệ thông tin, Trường Đại học Công nghệ đã đọc bản thảo giáo trình và có các góp ý quí báu về nội dung chuyên môn cũng như cách thức trình bày. Tuy vậy, giáo trình vẫn còn nhiều khiếm khuyết, các tác giả mong tiếp tục nhận được góp ý để hoàn thiện trong tương lai.

Chương 1. MỞ ĐẦU

Lập trình là công đoạn quan trọng chủ chốt và không thể thiếu để tạo ra sản phẩm phần mềm. Phần mềm càng trở nên đa dạng và ngành công nghiệp phần mềm càng phát triển thì người ta càng thấy rõ tầm quan trọng của phương pháp lập trình. Phương pháp lập trình tốt không chỉ đảm bảo tạo ra phần mềm tốt mà còn hỗ trợ thiết kế phần mềm có tính mở và hỗ trợ khả năng sử dụng lại các mô đun. Nhờ đó chúng ta có thể dễ dàng bảo trì, nâng cấp phần mềm cũng như giảm chi phí phát triển phần mềm.

Trong những thập kỷ 1970, 1980, phương pháp phát triển phần mềm chủ yếu là lập trình có cấu trúc (*structured programming*). Cách tiếp cận cấu trúc đối với việc thiết kế chương trình dựa trên chiến lược chia để trị: Để giải một bài toán lớn, chúng ta tìm cách chia nó thành vài bài toán nhỏ hơn và giải riêng từng bài; để giải mỗi bài, hãy coi nó như một bài toán mới và có thể tiếp tục chia nó thành các bài toán nhỏ hơn; cuối cùng, ta sẽ đi đến những bài toán có thể giải ngay được mà không cần phải chia tiếp. Cách tiếp cận này được gọi là lập trình từ trên xuống (*top-down programming*).

Lập trình từ trên xuống là một phương pháp tốt và đã được áp dụng thành công cho phát triển rất nhiều phần mềm. Tuy nhiên, cùng với sự đa dạng và phức tạp của phần mềm, phương pháp này bộc lộ những hạn chế. Trước hết, nó hầu như chỉ đáp ứng việc tạo ra các lệnh hay là các quy trình để giải quyết một bài toán. Dần dần, người ta nhận ra rằng thiết kế các cấu trúc dữ liệu cho một chương trình có tầm quan trọng không kém việc thiết kế các hàm/thủ tục và các cấu trúc điều khiển. Lập trình từ trên xuống không quan tâm đủ đến dữ liệu mà chương trình cần xử lý.

Thứ hai, với lập trình từ trên xuống, chúng ta khó có thể tái sử dụng các phần của chương trình này cho các chương trình khác. Bằng việc xuất phát từ một bài toán cụ thể và chia nó thành các mảnh sao cho thuận, cách tiếp cận này có xu hướng tạo ra một thiết kế đặc thù cho chính bài toán đó. Chúng ta khó có khả năng lấy một đoạn mã lớn từ một chương trình cũ lắp vào một dự án mới mà không phải sửa đổi lớn. Việc xây dựng các chương trình chất lượng cao là khó khăn và tốn kém, do đó những nhà phát triển phần mềm luôn luôn muốn tái sử dụng các sản phẩm cũ.

Thứ ba, môi trường hoạt động trong thực tế của các ứng dụng luôn thay đổi. Dẫn đến việc yêu cầu phần mềm cũng phải liên tục thay đổi theo để đáp ứng nhu cầu của người dùng nếu không muốn phần mềm bị đào thải. Do đó, một thiết kế linh hoạt mềm dẻo là cái mà các nhà phát triển phần mềm mong muốn. Phương pháp tiếp cận từ dưới lên (bottom-up) hỗ trợ tốt hơn cho tính linh hoạt mềm dẻo đó.

Trong thực tế, thiết kế và lập trình từ trên xuống thường được kết hợp với thiết kế và lập trình từ dưới lên. Trong tiếp cận từ dưới lên, từ các vấn đề mà ta đã biết

cách giải và có thể đã có sẵn các thành phần tái sử dụng được chúng ta xây dựng dần theo hướng lên trên, hướng đến một giải pháp cho bài toán tổng.

Các thành phần tái sử dụng được nên có tính mô-đun hóa cao nhất có thể. Mỗi mô-đun là một thành phần của một hệ thống lớn hơn, nó tương tác với phần còn lại của hệ thống theo một cách đơn giản và được quy ước chặt chẽ. Ý tưởng ở đây là một mô-đun có thể được "lắp vào" một hệ thống. Chi tiết về những gì xảy ra bên trong mô-đun không cần được xét đến đối với hệ thống nói chung, miễn là mô-đun đó hoàn thành tốt vai trò được giao. Đây gọi là **che giấu thông tin** (*information hiding*), một trong những nguyên lý quan trọng nhất của công nghệ phần mềm.

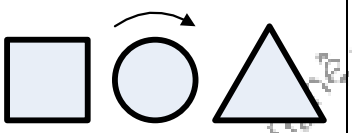
Một dạng thường thấy của các mô-đun phần mềm là nó chứa một số dữ liệu kèm theo một số hàm/thủ tục để xử lý dữ liệu đó. Ví dụ, một mô-đun sổ địa chỉ có thể chứa một danh sách các tên và địa chỉ, kèm theo là các hàm/thủ tục để thêm một mục tên mới, in nhãn địa chỉ…Với cách này, dữ liệu được bảo vệ vì nó chỉ được xử lý theo các cách đã được biết trước và được định nghĩa chặt chẽ. Ngoài ra, nó cũng tạo thuận lợi cho các chương trình sử dụng mô-đun này, vì các chương trình đó không phải quan tâm đến chi tiết biểu diễn dữ liệu bên trong mô-đun. Thông tin về biểu diễn dữ liệu được che giấu.

Các mô-đun hỗ trợ dạng che giấu thông tin này bắt đầu trở nên phổ biến trong các ngôn ngữ lập trình đầu thập kỷ 1980. Từ đó, một hình thức tiên tiến hơn của chính ý tưởng đó đã lan rộng trong ngành công nghệ phần mềm. Cách tiếp cận đó được gọi là **lập trình hướng đối tượng** (*object-oriented programming*), thường được gọi tắt là **OOP**.

Câu chuyện tưởng tượng sau đây[1] minh họa phần nào sự khác biệt giữa lập trình thủ tục và lập trình hướng đối tượng trong thực tế của ngành công nghệ phàn mềm. Có hai lập trình viên nhận được cùng một đặc tả hệ thống và được yêu cầu xây dựng hệ thống đó, thi xem ai là người hoàn thành sớm nhất. Dậu là người chuyên dùng phương pháp lập trình thủ tục, còn Tuất quen dùng lập trình hướng đối tượng. Cả Dậu và Tuất đều cho rằng đây là nhiệm vụ đơn giản.

Đặc tả như sau:

Một phần mềm có giao diện đồ họa, nó vẽ một hình vuông, một hình tròn, và một hình tam giác. Khi người dùng click vào một hình, hình đó sẽ xoay 360° theo chiều kim đồng hồ và chương trình sẽ chơi một đoạn âm thanh AIF đặc thù của hình đó.



[1] Nguồn: Head First Java, 2nd Edition.

Dậu tính toán, "Chương trình này phải *làm* những gì? Ta cần đến những thủ tục nào?" Anh tự trả lời, "**xoay** và **chơi nhạc**." Và anh bắt tay vào viết các thủ tục đó. Chương trình không phải là một loạt các thủ tục thì nó là cái gì?

Trong khi đó, Tuất nghĩ, "Trong chương trình này có những *thứ* gì...đâu là những *nhân tố* chính?" Đầu tiên, anh ta nghĩ đến những **Hình vẽ**. Ngoài ra, anh còn nghĩ đến những đối tượng khác như *người dùng*, *âm thanh*, và sự kiện *click chuột*. Nhưng anh đã có sẵn thư viện mã cho mấy đối tượng đó, nên anh tập trung vào việc xây dựng các *Hình vẽ*.

Dậu đã quá thạo với công việc kiểu này rồi, anh ra bắt tay vào viết các thủ tục quan trọng và nhanh chóng hoàn thành hai thủ tục **xoay** (rotate) và **chơi nhạc** (playSound):

```
rotate(shapeNum) {
    // cho hình xoay 360º
}
playSound(shapeNum) {
     // dùng shapeNum để tra xem cần chơi file AIF nào
     // và chơi file đó
}
```

Còn Tuất ngồi viết ba lớp, mỗi lớp dành cho một hình.

Square

rotate() {
  // xoay hình vuông
}
playSound() {
  // chơi file AIF cho
  // hình vuông
}

Circle

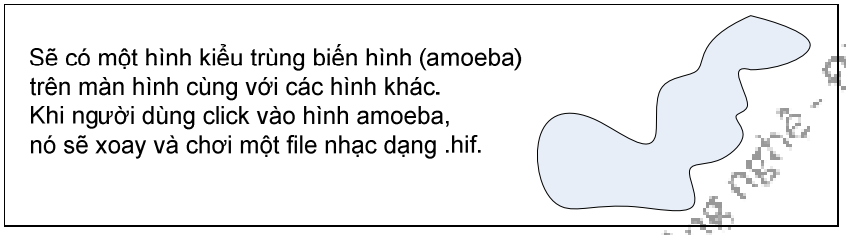
rotate() {
  // xoay hình tròn
}
playSound() {
  // chơi file AIF cho
  // hình tròn
}

Triangle

rotate() {
  // xoay hình tam giác
}
playSound() {
  // chơi file AIF cho
  // hình tam giác
}

Dậu vừa nghĩ rằng mình đã thắng cuộc thì sếp nói "Về mặt kĩ thuật thì Dậu xong trước, nhưng ta phải bổ sung một chút xíu nữa vào chương trình." Hai người đã quá quen với chuyện đặc tả thay đổi – chuyện thường ngày trong ngành.

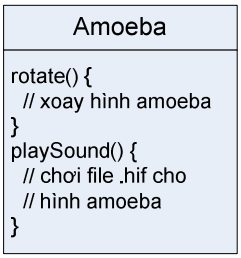
Đặc tả được bổ sung nội dung sau:



Đối với Dậu, thủ tục rotate vẫn ổn, mã dùng một bảng tra cứu để khớp giá trị shapeNum với một hình đồ họa cụ thể. Nhưng ***playSound thì phải sửa***.

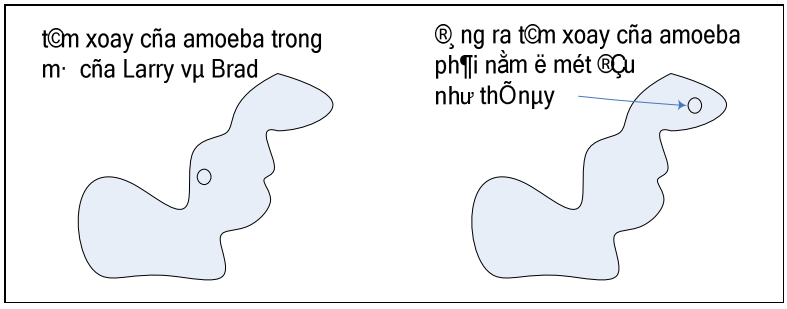
Rốt cục không phải sửa nghiêm trọng, nhưng Dậu vẫn thấy ***không thoải mái khi phải động vào sửa phần mã đã được test xong từ trước***. Anh biết, dù quản lý dự án có nói gì đi chăng nữa, ***đặc tả thay đổi suốt***.

Còn Tuất thì thản nhiên vừa nhâm nhi cà phê vừa viết một lớp mới. Điều anh thích nhất về OOP là anh không phải sửa gì ở phần mã đã được test và bàn giao. Anh nghĩ về những ích lợi của OOP và lẩm bẩm "Tính linh hoạt, khả năng mở rộng,...".



Dậu cũng vừa kịp hoàn thành chỉ một lát trước Tuất. Nhưng nụ cười của anh vụt tắt khi nhìn thấy bộ mặt của sếp và nghe thấy giọng sếp vẻ thất vọng "không được rồi, amoeba thực ra không xoay kiểu này..."

Thì ra cả hai lập trình viên đều đã viết đoạn xoay hình theo cách: (1) xác định hình chữ nhật bao hình; (2) xác định tâm của hình chữ nhật đó và xoay hình quanh điểm đó. Nhưng hình trùng biến hình thì lại cần xoay quanh một điểm ở một *đầu mút*, như kiểu kim đồng hồ.



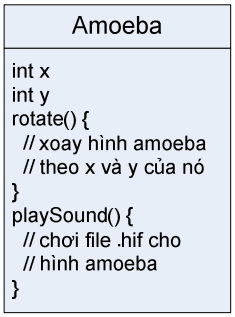
"Mình tèo rồi." Dậu ngán ngẩm. "Tuy là, ừm, có thể thêm một lệnh if/else nữa vào thủ tục rotate, rồi hard-code tâm xoay cho amoeba. Làm vậy chắc là sẽ không làm hỏng đoạn nào khác." Nhưng một giọng nói trong đầu Dậu thì thào, "*Nhầm to! Cậu có chắc là đặc tả sẽ không thay đổi lần nữa không đấy?*"

Cuối cùng Dậu chọn cách bổ sung tham số về tâm xoay vào cho thủ tục rotate. ***Rất nhiều đoạn mã đã bị ảnh hưởng.*** Phải test lại, dịch lại cả đống mã. Có những đoạn trước chạy tốt thì nay không chạy được nữa.

```
rotate(shapeNum, xPt, yPt) {
```

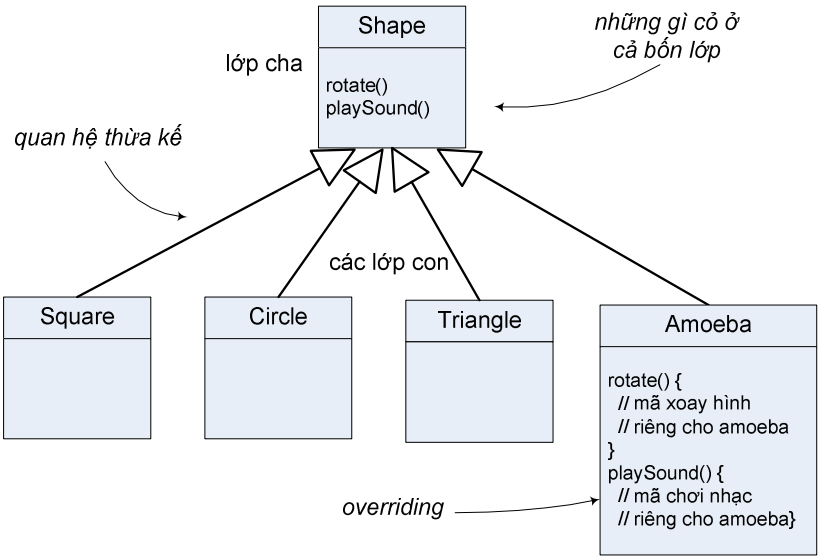
```
    //nếu hình không phải amoeba,
      // tính tâm xoay
      // dựa trên một hình chữ nhật
      // rồi xoay hình
    //nếu không
      // dựng xPt và yPt làm offset tâm xoay
      // rồi xoay hình
}
```

Còn Tuất, không chần chừ chút nào, anh sửa luôn phương thức rotate, nhưng chỉ sửa ở lớp Amoeba mà thôi. Tuất ***không hề động đến các đoạn mã đã dịch, đã chạy và đã test*** tại các phần khác trong chương trình. Để cho Amoeba một tâm xoay, anh thêm một **thuộc tính** mà tất cả các hình trùng biến hình sẽ có. Anh nhanh chóng sửa, test, và bàn giao mã cho sếp.



"Không nhanh thế được!" Dậu tìm thấy một nhược điểm trong cách tiếp cận của Tuất, và anh chắc mẩm nó sẽ giúp anh chuyển bại thành thắng. Dậu thấy mã của Tuất bị lặp, rotate có mặt ở cả bốn thứ hình, thiết kế này có gì hay ho khi phải bảo trì cả bốn phương thức rotate khác nhau?

Tuất giải thích: Dậu chưa nhìn thấy đặc điểm quan trọng của thiết kế, đó là quan hệ thừa kế. Bốn lớp có những đặc điểm chung, những đặc điểm đó được tách ra và đặt trong một lớp mới tên là Shape. Các lớp kia, mỗi lớp đều được xem là "thừa kế từ lớp Shape". Nói cách khác, nếu lớp Shape có những chức năng gì thì các lớp kia tự động có các chức năng đó.



Tuy nhiên, Amoeba có tâm xoay khác và chơi file nhạc khác. Lớp Amoeba **cài đè** các hoạt động rotate và playSound đã được thừa kế từ Shape bằng cách định nghĩa lại các thủ tục này. Và khi chạy, hệ thống tự biết là cần dùng phiên bản được viết tại Amoeba thay vì dùng phiên bản thừa kế từ Shape. Đó là đặc điểm thú vị của phương pháp hướng đối tượng.

Khi ta cần yêu cầu một hình nào đó xoay, tam giác hay amoeba, ta chỉ việc gọi phương thức rotate cho đối tượng đó, và hệ thống sẽ tự biết phải làm gì, trong khi phần còn lại của chương trình không biết hoặc không quan tâm đến việc đối tượng đó xoay kiểu gì. Và khi ta cần bổ sung một cái gì đó mới vào chương trình, ta chỉ phải viết một lớp mới cho loại đối tượng mới, từ đó, các đối tượng mới sẽ có cách hành xử của riêng chúng.

1.1. KHÁI NIỆM CƠ BẢN

Hướng đối tượng là kĩ thuật mô hình hóa một hệ thống thế giới thực trong phần mềm dựa trên các đối tượng. **Đối tượng** (*object*) là khái niệm trung tâm của OOP, nó là một mô hình của một thực thể hay khái niệm trong thế giới thực. Việc mô hình hóa này bao gồm xác định các đối tượng tham gia bài toán – những cái làm nhiệm vụ gì đó hoặc bị làm gì đó. Lập trình theo kiểu hướng đối tượng là hoạt động định nghĩa các thể loại của các đối tượng đó ở hình thức các khuôn mẫu để tạo ra chúng.

Trong thời gian chạy, một chương trình OOP chính là một tập các đối tượng gửi thông điệp cho nhau để yêu cầu dịch vụ và thực hiện dịch vụ khi được yêu cầu. Việc một đối tượng thực hiện một dịch vụ có thể dẫn đến việc nó thay đổi trạng thái của bản thân. Một ví dụ có tính chất gần với thế giới thực: ông A đến rút tiền tại máy ATM. Ta có các đối tượng: ông A, máy ATM, cơ sở dữ liệu ngân hàng, và tài khoản của ông A. Trình tự diễn ra như sau: Ông A cho thẻ ngân hàng vào khe máy ATM;

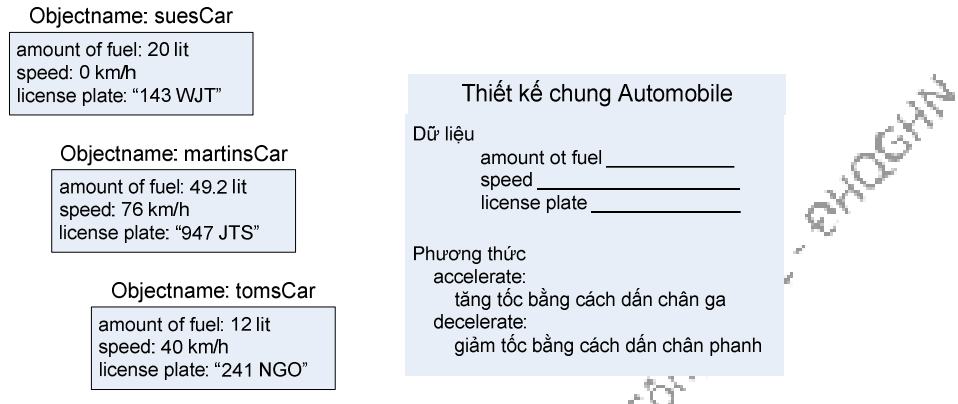
đối tượng ATM yêu cầu cơ sở dữ liệu ngân hàng cung cấp đối tượng tài khoản của ông A; ông A yêu cầu rút 100.000 đồng; đối tượng ATM yêu cầu đối tượng tài khoản trừ đi 100.000 đồng. Như vậy giao dịch này bao gồm chuỗi các yêu cầu dịch vụ và việc các đối tượng thực hiện các yêu cầu đó, đồng thời thay đổi trạng thái của mình (tài khoản ông A bị bớt tiền, ông A có thêm tiền, dữ liệu nhật trình ATM có thêm thông tin về một giao dịch).

1.2. ĐỐI TƯỢNG VÀ LỚP

Gần như bất cứ thứ gì cũng có thể được mô hình hóa bằng một đối tượng. Chẳng hạn, một màu, một hình vẽ, một cái nhiệt kế.

Mỗi đối tượng có một tập các **thuộc tính** (*attribute*) như các giá trị hay trạng thái để mô hình hóa đối tượng đó. Chẳng hạn, một cái nhiệt kế có thể có thuộc tính là vị trí hiện tại của nó và trạng thái hiện tại tắt hay bật, các thuộc tính một màu có thể là giá trị của ba thành phần RGB của nó. Một cái ô tô có các thuộc tính như: lượng xăng hiện có, tốc độ hiện tại, biển số.

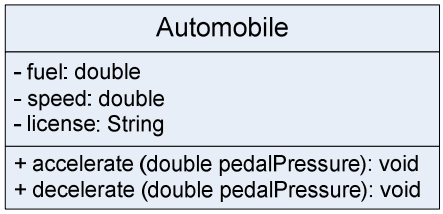
Mỗi đối tượng có một tập các trách nhiệm mà nó thực hiện bằng cách cung cấp dịch vụ cho các đối tượng khác. Các dịch vụ này có thể cho phép truy vấn thông tin hoặc làm thay đổi trạng thái của đối tượng. Ví dụ, nhiệt kế cho phép truy vấn về tình trạng tắt/bật của nó; đáp ứng các yêu cầu về nhiệt độ hiện hành mà nó đo được, yêu cầu tắt/bật. Một cái ô tô cho phép tăng ga, giảm ga để tăng/giảm tốc độ di chuyển. Đối với thiết kế tốt, các đối tượng bên ngoài không phải quan tâm xem một đối tượng nào đó cài đặt một dịch vụ như thế nào, mà chỉ cần biết đối tượng đó cung cấp những dịch vụ nào (hay nó có những trách nhiệm gì). Chẳng hạn, người lái xe không cần biết cơ chế chuyển đổi từ lực nhấn lên chân đạp ga sang sự thay đổi về tốc độ của ô tô.



Hình 1.1: Các đối tượng ô tô và đặc điểm chung của chúng.

Trong mỗi ứng dụng, các đối tượng có đặc điểm tương tự nhau, chẳng hạn các tài khoản ngân hàng, các sinh viên, các máy ATM, những chiếc ô tô được xếp vào cùng một nhóm, đó là **lớp** (*class*). Mỗi lớp là đặc tả các đặc điểm của các đối tượng

thuộc lớp đó. Cụ thể, một định nghĩa lớp mô tả tất cả các thuộc tính của các đối tượng thành viên của lớp đó và các phương thức thực thi hành vi của các đối tượng đó. Ví dụ, ta có thể có nhiều đối tượng ô tô với thông số khác nhau về lượng xăng hiện có, tốc độ hiện tại, và biển số xe; định nghĩa lớp ô tô mô tả đặc điểm chung của các thông số đó cùng với các phương thức thực hiện các hoạt động tăng tốc, giảm tốc.



Hình 1.2: Lớp Automobile vẽ bằng kí pháp UML

Quan hệ giữa lớp và đối tượng gần giống như quan hệ giữa kiểu dữ liệu và các biến thuộc kiểu dữ liệu đó. Các đối tượng được tạo ra khi chương trình chạy, và lớp là khuôn mẫu mà từ đó có thể tạo ra các đối tượng thuộc lớp đó. Mỗi đối tượng được tạo ra từ một lớp được gọi là một **thực thể** (*instance*) của lớp đó. Một chương trình khi được viết là sự kết hợp của các lớp khác nhau. Còn khi chạy, nó là một tập hợp các đối tượng hoạt động và tương tác với nhau, các đối tượng này được sinh ra từ các lớp cấu thành nên chương trình đó.

Mỗi đối tượng đều có một **thời gian sống**. Trong khi chương trình chạy, đối tượng được tạo và khởi tạo giá trị theo yêu cầu. Ngay khi một đối tượng được tạo ra, hệ thống tự động gọi một **hàm khởi tạo** (*constructor*) để khởi tạo giá trị cho các thuộc tính của đối tượng. Kể từ đó, đối tượng bắt đầu tồn tại, nó gửi và nhận các thông điệp, và cuối cùng thì nó bị hủy đi khi không còn cần đến nữa. Trong khi đối tượng tồn tại, nó giữ định danh và trạng thái của mình. Mỗi đối tượng có một định danh riêng và có bộ thuộc tính riêng, độc lập với các đối tượng khác thuộc cùng một lớp. Trong thực tế, mỗi đối tượng có vị trí riêng trong bộ nhớ.

Các đối tượng dùng các **thông điệp** (*message*) để liên lạc với nhau. Nhìn từ phương diện lập trình, việc gửi một thông điệp tới một đối tượng chính là gọi một phương thức của đối tượng đó, còn việc một đối tượng nhận được một thông điệp chính là việc một phương thức của nó được một đối tượng khác gọi. Chương trình khi chạy là một tập các đối tượng, mỗi đối tượng gửi thông điệp cho các đối tượng khác trong hệ thống và đáp ứng các thông điệp mà mình nhận được. Thông thường, một thông điệp được gửi bằng một lời gọi phương thức trong chương trình. Tuy nhiên, các thông điệp có thể xuất phát từ hệ điều hành hoặc môi trường chạy chương trình. Chẳng hạn khi người dùng click chuột vào một nút bấm tại một cửa sổ chương trình, một thông điệp sẽ được gửi đến đối tượng điều khiển nút bấm đó thông báo rằng cái nút đó đã bị nhấn.

1.3. CÁC NGUYÊN TẮC TRỤ CỘT

Lập trình hướng đối tượng có ba nguyên tắc trụ cột: đóng gói, thừa kế và đa hình, còn trừu tượng hóa là khái niệm nền tảng.

Trừu tượng hóa (*abstraction*) là một cơ chế cho phép biểu diễn một tình huống phức tạp trong thế giới thực bằng một mô hình được đơn giản hóa. Nó bao gồm việc tập trung vào các tính chất quan trọng của một đối tượng khi phải làm việc với lượng lớn thông tin. Ví dụ, đối với một con mèo trong ngữ cảnh một cửa hàng bán thú cảnh, ta có thể tập trung vào giống mèo, màu lông, cân nặng, tuổi, đã tiêm phòng dại hay chưa, và bỏ qua các thông tin khác như dung tích phổi, nồng độ đường trong máu, huyết áp, còn đối với một con mèo trong ngữ cảnh bệnh viện thú y thì lại là một chuyện khác. Các đối tượng ta thiết kế trong chương trình OOP sẽ là các trừu tượng hóa theo nghĩa đó, ta bỏ qua nhiều đặc điểm của đối tượng thực và chỉ tập trung vào các thuộc tính quan trọng cho việc giải một bài toán cụ thể. Người ta gọi một trừu tượng hóa là một mô hình của một đối tượng hoặc khái niệm trong thế giới thực.

Trừu tượng hóa là một trong những công cụ cơ bản của tất cả các phương pháp lập trình, không chỉ lập trình hướng đối tượng. Khi viết một chương trình giải một bài toán của thế giới thực, trừu tượng hóa là một cách để mô hình hóa bài toán đó. Ví dụ, khi ta viết một chương trình quản lý sổ địa chỉ, ta sẽ dùng các trừu tượng hóa như tên, địa chỉ, số điện thoại, thứ tự bảng chữ cái, và các khái niệm liên quan tới một sổ địa chỉ. Ta sẽ định nghĩa các thao tác để xử lý dữ liệu chẳng hạn như thêm một mục tên mới hoặc sửa một địa chỉ. Trong ngữ cảnh lập trình, trừu tượng hóa là mô hình hóa thế giới thực theo cách mà nó có thể được cài đặt dưới dạng một chương trình máy tính.

Phương pháp hướng đối tượng trừu tượng hóa thế giới thực thành các đối tượng và tương tác giữa chúng với các đối tượng khác. Việc mô hình hóa trở thành mô hình hóa các đối tượng tham gia bài toán – một cái nhiệt kế, một người chủ tài khoản ngân hàng, một sổ địa chỉ… mỗi đối tượng cần có đủ các thuộc tính và phương thức để thực hiện được tất cả các dịch vụ mà nó được yêu cầu.

Đóng gói (*encapsulation*): Các trừu tượng hóa của những gì có liên quan đến nhau được đóng gói vào trong một đơn vị duy nhất. Các trạng thái và hành vi của các trừu tượng hóa được bọc lại trong một khối gọi là lớp. Cụ thể, sau khi đã xác định được các đối tượng, rồi đến các thuộc tính và hành động của mỗi đối tượng, mục tiêu là đóng gói trong mỗi đối tượng các tính năng cần thiết để nó có thể thực hiện được vai trò của mình trong chương trình. Thí dụ, một đối tượng nhiệt kế cần có những gì cần thiết để có thể đo nhiệt độ, lưu trữ số liệu của các lần đo nhiệt độ trước và cho phép truy vấn các số liệu này.

Định nghĩa lớp là công cụ lập trình chính yếu cho việc thực hiện nguyên tắc đóng gói. Một lớp là mô tả về một tập hợp các đối tượng có cùng các thuộc tính, hành vi.

Thuộc tính (*attribute*) dùng để lưu trữ thông tin trạng thái của một đối tượng. Một thuộc tính có thể chỉ đơn giản là một biến Boolean lưu trữ trạng thái tắt hoặc bật, hay phức tạp hơn khi chính nó lại là một đối tượng khác. Các thuộc tính được khai báo trong định nghĩa lớp và được gọi là các **biến của thực thể** (*instance variable*), gọi tắt là **biến thực thể**. Chúng còn được gọi là các **thành viên dữ liệu** (*data member*), hay **trường** (*field*).

Trạng thái (*state*) phản ánh các giá trị hiện tại của các thuộc tính của một đối tượng và là kết quả của hành vi của đối tượng đó theo thời gian.

Hành vi (*behavior*) là hoạt động của một đối tượng mà có thể nhìn thấy được từ bên ngoài. Trong đó có việc đối tượng thay đổi trạng thái ra sao hoặc việc nó trả về thông tin trạng thái khi nó được thông điệp yêu cầu.

Phương thức (*method*) là một thao tác hay dịch vụ được thực hiện đối với đối tượng khi nó nhận thông điệp tương ứng. Các phương thức cài đặt hành vi của đối tượng và được định nghĩa trong định nghĩa lớp. Phương thức còn được gọi bằng các cái tên khác như: **hàm thành viên** (*member function*) – gọi tắt là 'hàm', **thao tác** (*operation*), **dịch vụ** (*service*).

Khái niệm đóng gói còn đi kèm với khái niệm **che giấu thông tin** (*information hiding*) nghĩa là che giấu các chi tiết bên trong của một đối tượng khỏi thế giới bên ngoài. Chẳng hạn khi dùng một cái cầu dao điện, đối với người sử dụng, nó chỉ là một cái hộp mà khi gạt cần sẽ có tác dụng ngắt và nối điện và cái hộp có khả năng tự ngắt điện khi quá tải. Người dùng không biết và không cần biết các mạch điện bên trong được thiết kế ra sao, cơ chế phát hiện quá tải như thế nào. Những chi tiết đó được giấu bên trong, còn từ bên ngoài ta chỉ nhìn thấy cầu dao là một cái hộp có cần gạt.

Nói theo phương diện lập trình, nhìn từ bên ngoài một mô-đun chỉ thấy được các giao diện. Các lập trình viên tự do cài đặt chi tiết bên trong, với ràng buộc duy nhất là tuân theo giao diện đã được quy ước từ trước. Ta có thể thực hiện nguyên tắc đóng gói với tất cả các ngôn ngữ lập trình hướng đối tượng cũng như các ngôn ngữ thủ tục. Tuy nhiên, chỉ các ngôn ngữ hướng đối tượng mới cung cấp cơ chế cho phép che giấu thông tin, ngăn không cho bên ngoài truy nhập vào chi tiết bên trong của mô-đun.

Thừa kế (*inheritance*) là quan hệ mang tính phân cấp mà trong đó các thành viên của một lớp được kế thừa bởi các lớp được dẫn xuất trực tiếp hoặc gián tiếp từ lớp đó. Đây là cơ chế cho phép định nghĩa một lớp mới dựa trên định nghĩa của một lớp có sẵn, sao cho tất cả các thành viên của lớp "cũ" (lớp cơ sở hay lớp cha) cũng có mặt trong lớp mới (lớp dẫn xuất hay lớp con) và các đối tượng thuộc lớp mới có thể được sử dụng thay cho đối tượng của lớp cũ ở bất cứ đâu. Thừa kế là một hình thức tái sử dụng phần mềm, trong đó một lớp mới được xây dựng bằng cách hấp thụ các thành viên của một lớp có sẵn và bổ sung những tính năng mới hoặc sửa tính năng có sẵn. Nói cách khác, xuất phát từ một lớp mô hình hóa một khái niệm tổng quát hơn,

chẳng hạn Shape, ta có thể dùng quan hệ thừa kế để xây dựng các lớp mô hình hóa các khái niệm cụ thể hơn, chẳng hạn Circle, Triangle. Bằng cách này, ta có thể sử dụng giao diện cũng như cài đặt của lớp cũ cho lớp mới.

Đa hình (*polymorphism*), theo nghĩa tổng quát, là khả năng tồn tại ở nhiều hình thức. Trong hướng đối tượng, đa hình đi kèm với quan hệ thừa kế và nó có nghĩa rằng cùng một cái tên có thể được hiểu theo các cách khác nhau tùy từng tình huống. Các đối tượng thuộc các lớp dẫn xuất khác nhau có thể được đối xử như nhau, như thể chúng là các đối tượng thuộc lớp cơ sở, chẳng hạn có thể đặt các đối tượng Triangle và Circle trong cùng một cấu trúc dữ liệu dành cho Shape, hoặc dùng cùng một lời gọi hàm rotate cho các đối tượng Triangle hay Circle. Và khi nhận được cùng một thông điệp đó, các đối tượng thuộc các lớp khác nhau hiểu nó theo những cách khác nhau. Ví dụ, khi nhận được thông điệp "rotate", các đối tượng Triangle và Amoeba thực hiện các phương thức rotate() khác nhau.

Bài tập

1. Điền từ thích hợp vào chỗ trống trong mỗi câu sau:

    a) Quan hệ giữa một ngôi nhà và một bản thiết kế tương tự như quan hệ giữa một _____ với một lớp.

    b) Khi mỗi đối tượng của một lớp giữ một bản riêng của một thuộc tính, trường dữ liệu đại diện cho thuộc tính đó được gọi là _____

2. Chú trọng đến các tính chất quan trọng trong khi bỏ qua các chi tiết ít quan trọng được gọi là

    A. Trừu tượng hóa

    B. Đa hình

    C. Đóng gói

    D. Che giấu thông tin

3. "Cùng một thông điệp được hiểu theo các cách khác nhau tùy theo đối tượng nhận được thông điệp đó thuộc lớp nào" là đặc điểm của khái niệm nào?

    A. Đóng gói

    B. Đa hình

    C. Thừa kế

    D. Tái sử dụng

4. "Đối tượng thuộc lớp con có thể được đối xử như đối tượng thuộc lớp cha" là đặc điểm của khái niệm nào?

    A. Trừu tượng hóa

    B. Đa hình

    C. Đóng gói

    D. Che giấu thông tin

    E. Thừa kế

5. "Che đi các chi tiết cài đặt và chỉ cho thấy giao diện của mô-đun" là đặc điểm của khái niệm nào?

    A. Trừu tượng hóa

    B. Đa hình

    C. Đóng gói

    D. Tái sử dụng

Chương 2. NGÔN NGỮ LẬP TRÌNH JAVA

Java được hãng Sun Microsystems[2] thiết kế năm 1991 như là một ngôn ngữ dành cho các chương trình nhúng (*embedded program*) chạy trên các thiết bị điện tử gia dụng như lò vi sóng và các hệ thống an ninh gia đình. Tuy nhiên, sự phát triển và lan rộng của Internet và World Wide Web (WWW) đã khiến Sun chuyển hướng Java từ một ngôn ngữ cho lập trình nhúng sang ngôn ngữ lập trình ứng dụng Web. Đến nay, Java đã trở thành ngôn ngữ lập trình ứng dụng phổ thông và là một trong những ngôn ngữ quan trọng nhất để phát triển các ứng dụng Web và Internet.

2.1. ĐẶC TÍNH CỦA JAVA

Java là ngôn ngữ hướng đối tượng. Các ngôn ngữ hướng đối tượng chia chương trình thành các mô-đun riêng biệt, được gọi là các đối tượng, chúng đóng gói dữ liệu và các thao tác của chương trình. Các khái niệm lập trình hướng đối tượng và thiết kế hướng đối tượng nói về phong cách tổ chức chương trình đang ngày càng được lựa chọn cho việc xây dựng các hệ thống phần mềm phức tạp. Không như ngôn ngữ C++, trong đó các đặc điểm hướng đối tượng được gắn thêm vào ngôn ngữ C, ngay từ đầu Java được thiết kế là một ngôn ngữ hướng đối tượng.

Java là ngôn ngữ có tính chắc chắn. Không như nhiều ngôn ngữ lập trình khác, lỗi trong các chương trình Java không gây sự cố hệ thống (*system crash*). Một số đặc tính của ngôn ngữ còn cho phép phát hiện nhiều lỗi tiềm tàng trước khi chương trình chạy.

Java có tính độc lập nền tảng (*platform independent*). Một nền tảng (*platform*) ở đây có nghĩa một hệ thống máy tính với hệ điều hành cụ thể, chẳng hạn như một hệ thống Windows hay Macintosh. Thương hiệu của Java là "Write once, run anywhere" (*Viết một lần, chạy bất cứ đâu*). Có nghĩa là một chương trình Java có thể chạy trên các nền tảng khác nhau mà không phải dịch lại. Một số ngôn ngữ bậc cao khác không có được đặc tính này. Tính khả chuyển, hay khả năng chạy trên hầu như tất cả các nền tảng, còn là nguyên do cho việc Java rất phù hợp cho các ứng dụng Web.

Java là ngôn ngữ phân tán. Các chương trình có thể được thiết kế để chạy trên mạng máy tính, một chương trình bao gồm những lớp đặt rải rác tại các máy khác nhau trong mạng. Bên cạnh ngôn ngữ, Java còn có một bộ sưu tập phong phú các thư viện mã đã được thiết kế để dùng trực tiếp cho các loại ứng dụng cụ thể, tạo

[2] Sun Microsystems đã nhập vào hãng Oracle từ năm 2010.