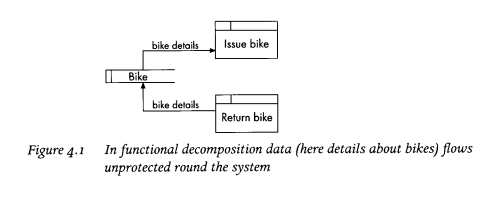
Giớithiệu  
Trong chương này, chúng tôi bắt đầu bằng cách xem xét các vấn đề liên quan đến các phương pháp truyền thống để phát triển phần mềm và cách thức hướng đối tượng giải quyết các vấn đề này. Chúng tôi giới thiệu đối tượng là khối xây dựng cơ bản của phần mềm hướng đối tượng, tức là làm cơ sở cho kiến trúc phần mềm. Chúng tôi tóm tắt ngắn gọn lý do tại sao cộng đồng hướng đối tượng dựa trên phân rã phần mềm của họ trên đối tượng chứ không phải phân rã chức năng từ trên xuống truyền thống. Chúng tôi giải thích khái niệm về một lớp và mối quan hệ giữa các đối tượng và các lớp. Cuối cùng, chúng tôi kiểm tra các mối quan hệ khác nhau có thể tồn tại giữa các đối tượng và giải thích ý nghĩa của chúng trong bối cảnh của hệ thống đang phát triển.

Tại sao cần một phương pháp phát triển mới

Các vấn đề với cấu trúc để phát triển phần mềm

Phần này xem xét cách thức phát triển hệ thống trước hướng đối tượng và tại một số vấn đề dẫn đến từ việc sử dụng phương pháp truyền thống.

*Funtional decomposition(phân rã chức năng).* Trong nhiều năm, các hệ thống phần mềm đã được phát triển bằng cách sử dụng một cách tiếp cận có cấu trúc dựa trên sự phân rã chức năng. Điều này có nghĩa là các nhà phát triển đã phân tách và sau đó xây dựng hệ thống theo các lĩnh vực hoạt động chính - nói cách khác, các hệ thống con được xác định tương ứng trực tiếp với các nhiệm vụ mà hệ thống phải thực hiện. Ví dụ, trong hệ thống cho thuê xe đạp, chẳng hạn như hệ thống được sử dụng trong cuốn sách này, hệ thống có thể sẽ dựa trên các hệ thống con hoặc quy trình xử lý việc cấp xe đạp, trả lại xe đạp, duy trì hồ sơ cho xe đạp và cho khách hàng, v.v. các quy trình sẽ thực hiện một chức năng riêng biệt và dữ liệu về xe đạp, giao dịch và khách hàng sẽ được chuyển tự do giữa chúng. Trong phân tách chức năng, có một sự tách biệt rõ ràng giữa dữ liệu và quy trình và các mục dữ liệu có thể thường xuyên được truy cập bởi bất kỳ phần nào của chương trình. Hình 4.1 cho thấy dữ liệu về xe đạp được chuyển giữa một cửa hàng dữ liệu lưu giữ hồ sơ về xe đạp và các quy trình xử lý việc phát hành và trả lại xe đạp cho khách hàng. Đây là một nguồn có thể có vấn đề, bởi vì các chi tiết xe đạp có thể truy cập được vào các bộ phận khác của hệ thống mà không có sự bảo vệ khỏi các quy trình có thể truy cập và sửa đổi chúng do lỗi.



Tại sao phải có những vấn đề bảo trì lớn?

**Bảo trì**. Phần mềm có cấu trúc được thiết kế sử dụng phân rã chức năng mang đến những vấn đề bảo trì lớn. Thay đổi mã thường xuyên được giới thiệu các lỗi mới; một sự thay đổi ở một nơi gây ra tác dụng phụ không mong muốn ở nơi khác: 'hiệu ứng gợn' khét tiếng. Các nhà phát triển phần mềm nhận thức được rằng nếu dữ liệu trong hệ thống có thể truy cập được vào tất cả các phần của mã, thì nó có thể vô tình bị hỏng và đây là nguyên nhân gây ra hiệu ứng gợn. Các nhà phát triển phần mềm nhận ra rằng để ngăn chặn điều này, dữ liệu cần phải được đóng gói (bảo vệ khỏi truy cập trái phép).

**Mô-đun kém.** Tuy nhiên, một hệ thống bị phân rã chức năng, điều rất quan trọng là mỗi thành phần hoặc mô-đun riêng biệt phải khép kín, với mục đích được xác định rõ và độc lập nhất có thể với các mô-đun khác. Thật không may, phân tách một hệ thống sử dụng phân rã chức năng đã không dẫn đến các mô-đun với các đặc điểm này. Một mô-đun trong một hệ thống dựa trên phân rã chức năng thường không gắn kết bên trong và phụ thuộc nhiều vào các mô-đun khác trong chương trình.

Tại sao quá trình lại diễn ra chậm?

Với phân rã chức năng, thực hiện thay đổi mã, để sửa lỗi phần mềm hoặc thực hiện các thay đổi trong yêu cầu của người dùng, là một quá trình rất chậm. Để hiểu đủ mã để thực hiện thay đổi, tốt nhất là không gây ra tác dụng phụ, lập trình viên bảo trì phải đọc rất nhiều mã. Điều này là do tính mô-đun kém, các mô-đun mã không đủ độc lập, chúng không tự chủ. Lập trình viên bảo trì không thể hiểu chúng một cách cô lập, anh ta phải đọc nhiều mô-đun ngoại vi để hiểu được cái mà anh ta đang thay đổi. Điều này ngụ ý rằng có các phụ thuộc ẩn giữa các mô-đun. Điều cần thiết là các mô-đun gắn kết (kết hợp nội bộ) có thể hoạt động độc lập hoặc thông qua các giao diện được xác định rõ với các mô-đun khác.

**Kiểm tra**. Các hệ thống phần mềm dựa trên phân rã chức năng vẫn đang được cung cấp với các lỗi trong mã; họ được coi là không đáng tin cậy và không được kiểm tra đầy đủ. Người ta cho rằng điều này có thể là một phần vì khó kiểm tra các đơn vị trong sự cô lập và quản lý kiểm tra tích hợp nghiêm ngặt. Điều này một lần nữa một phần vì các đơn vị mã quá phụ thuộc lẫn nhau. Nhà phát triển phần mềm muốn các đơn vị phần mềm có thể dễ dàng được kiểm tra và được kết hợp một cách hợp lý vào lịch kiểm thử tích hợp.

Tại sao các mô-đun cần phải độc lập?

**Tái sử dụng phần mềm.** Việc tái sử dụng phần mềm chưa bao giờ được thực hiện như dự kiến. Một trong những lý do chính cho điều này là một cấu trúc phần mềm phù hợp đã không tồn tại. Một mô-đun phụ thuộc nhiều vào các mô-đun phần mềm khác không thể dễ dàng được đưa vào thư viện phần mềm - tất cả các bit mà nó phụ thuộc cũng phải có trong thư viện. Để được sử dụng lại, các mô-đun cần phải độc lập. Các mô-đun thư viện hữu ích cũng phải có một mục đích rõ ràng đủ chung để có ích trong nhiều hệ thống.

**Dữ liệu so với chức năng**. Cách tiếp cận có cấu trúc đã xây dựng các hệ thống dựa trên chức năng của chúng - những gì hệ thống phải làm, các nhiệm vụ mà nó phải thực hiện. Vấn đề với điều này là thông thường chức năng của một hệ thống có nhiều biến động và có thể thay đổi so với dữ liệu của nó. Các nghiên cứu mở rộng đã chỉ ra rằng những thay đổi phổ biến nhất trong yêu cầu của người dùng là chức năng của hệ thống. Chúng ta hãy xem xét, ví dụ, một hệ thống phân bổ bệnh nhân và nhân viên đến các phòng trong bệnh viện. Chúng ta có thể tưởng tượng rằng theo thời gian, cách thức mà bệnh nhân được chỉ định nằm trên giường trong phòng bệnh và cách thức mà các y tá làm việc có thể thay đổi. Ngoài ra, có khả năng hệ thống sẽ được yêu cầu tạo ra nhiều báo cáo khác nhau và khác nhau cho người dùng. Dù những thay đổi này có thể là gì, tuy nhiên, hệ thống vẫn sẽ xử lý bệnh nhân, giường, y tá và phường - nói cách khác, dữ liệu vẫn giữ nguyên. Dữ liệu ổn định hơn nhiều so với chức năng và là cơ sở hợp lý hơn để xây dựng một hệ thống. Trong phân rã chức năng, kiến ​​trúc phần mềm (cách phân chia hệ thống) dựa trên chức năng của nó và theo thời gian, toàn bộ cấu trúc của hệ thống cuối cùng trở nên không ổn định. Tuy nhiên, dữ liệu trong hệ thống vẫn không thay đổi theo thời gian sử dụng của hệ thống; cuối cùng đã trở nên rõ ràng đối với cộng đồng phát triển phần mềm rằng một hệ thống dựa trên dữ liệu sẽ mạnh mẽ hơn.

Tại sao dễ dàng điều chỉnh để phù hợp với yêu cầu mới hoặc thay đổi?

Ưu điểm của phát triển hướng đối tượng

Để giải quyết các vấn đề liên quan đến phân rã chức năng, các nhà phát triển cần một cấu trúc phần mềm:

* Tự chủ, tức là không phụ thuộc nhiều vào các mô-đun khác một cách rõ ràng hoặc theo cách tối nghĩa
* Gắn kết với một mục đích duy nhất, được xác định rõ
* Dễ hiểu
* Dễ dàng điều chỉnh để phù hợp với yêu cầu mới hoặc thay đổi
* Dựa trên dữ liệu.

Ngoài các đặc điểm này, cấu trúc phần mềm cũng cần có:

* Dữ liệu đóng gói
* một giao diện công cộng được xác định rõ.

Hướng đối tượng sẽ dễ dàng hơn để kiểm tra kỹ lưỡng cả trong sự cô lập và là một phần của kế hoạch kiểm tra tích hợp. Nó nhiều khả năng sẽ được tái sử dụng và có thể được bao gồm trong một thư viện phần mềm.

Ý đầu slide 2. Cấu trúc phân rã phần mềm là gì?

Cộng đồng hướng đối tượng đề xuất đối tượng là một cấu trúc phần mềm có các đặc điểm được liệt kê ở trên. Các đối tượng, mặc dù chúng cung cấp chức năng, dựa trên dữ liệu trong hệ thống. Điều này cung cấp một cấu trúc ít có thể thay đổi. Như chúng ta sẽ thấy sau trong chương này, một đối tượng bảo vệ (hoặc đóng gói) dữ liệu của nó bằng cách làm cho dữ liệu chỉ có thể truy cập bằng cách sử dụng đối tượng ~ các hoạt động được khai báo công khai. Các đối tượng được thiết kế tốt là độc lập và tự chủ, giúp chúng dễ dàng kiểm tra độc lập trước khi được tích hợp vào hệ thống. Một đối tượng tốt có tính gắn kết, tức là liên quan đến một ý tưởng duy nhất, điều này làm cho nó phù hợp để đưa vào thư viện phần mềm để tái sử dụng.

*Sử dụng phân tách trường hợp và phân rã hướng đối tượng*

Mặc dù các đối tượng dựa trên dữ liệu trong hệ thống, điều này không có nghĩa là phần mềm hướng đối tượng bỏ qua chức năng cần thiết. Chúng ta đã thấy trong Chương 3 rằng mô hình use case sử dụng chỉ định, từ quan điểm của người dùng, hệ thống phải làm gì. Các đối tượng chúng tôi xác định phải có khả năng cung cấp chức năng đó. Mô hình use case sử dụng cung cấp một phân tách của hệ thống, dựa trên chức năng được yêu cầu. Tuy nhiên, cấu trúc phần mềm của một hệ thống hướng đối tượng dựa trên đối tượng. Nói cách khác, khi chúng ta đến để xây dựng phần mềm và chia nó thành các phần riêng biệt, chúng ta không chia nó ra theo các trường hợp sử dụng mà theo các đối tượng và nhóm đối tượng.

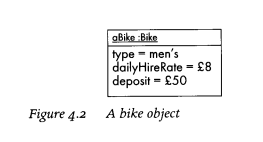
Một ưu điểm khác của phương pháp hướng đối tượng là nó cung cấp một quy trình phát triển liền mạch. Các đối tượng chúng tôi xác định là một phần của giai đoạn phân tích vẫn tồn tại qua các giai đoạn phát triển cho mã. Chuỗi liên tục này có nghĩa là các đối tượng phần mềm có khả năng dự đoán nhất định - ví dụ, chúng tôi có những kỳ vọng về việc một đối tượng đại diện cho khách hàng sẽ trông như thế nào và nó sẽ làm gì. Trong mã hướng đối tượng được thiết kế tốt, kỳ vọng của chúng tôi sẽ không bị thất vọng. Điều này làm cho mã dễ hiểu hơn và do đó dễ bảo trì hơn.

Sự phát triển liền mạch cũng cung cấp khả năng truy nguyên nguồn gốc của các yêu cầu của người dùng từ nhận dạng ban đầu cho đến mã. Trong cách tiếp cận có cấu trúc truyền thống để phát triển, một số mô hình được sử dụng để nắm bắt các yêu cầu của người dùng (như sơ đồ luồng dữ liệu) đã bị bỏ qua ở giai đoạn thiết kế nơi sử dụng một kỹ thuật mô hình mới để xác định cấu trúc của mã (ví dụ: biểu đồ cấu trúc). Điều này khiến cho việc đảm bảo rằng tất cả các yêu cầu của người dùng được kết hợp trong quá trình thực hiện cuối cùng là khó khăn: rất dễ mất các tính năng trong việc thay đổi mô hình. Trong cách tiếp cận hướng đối tượng, chúng tôi xác định các đối tượng đại diện cho những thứ trong miền vấn đề và về việc chúng tôi cần lưu trữ thông tin. Sau này trong quá trình phát triển, chúng tôi thêm các đối tượng để cung cấp cấu trúc phần mềm mà chúng tôi muốn triển khai (ví dụ: các đối tượng điều khiển và ranh giới) và các đối tượng sẽ làm với cách phần mềm sẽ hoạt động (nút, cửa sổ, trình nghe chuột, v.v.). Tuy nhiên, các đối tượng ban đầu, mặc dù chúng có thể thu thập một số tính năng bổ sung trong quá trình phát triển, vẫn sẽ được nhận dạng trong mã.

**Một đối tượng là gì?**

Đối tượng là khái niệm quan trọng nhất trong phát triển phần mềm hướng đối tượng; hệ thống hướng đối tượng được dựa trên đối tượng. Đơn giản nhất, một đối tượng là một đại diện của một cái gì đó trong khu vực ứng dụng mà chúng ta cần lưu trữ dữ liệu để cho phép hệ thống thực hiện những gì người dùng muốn. Ví dụ, trong hệ thống Wheels, chúng tôi chắc chắn sẽ muốn lưu trữ dữ liệu về xe đạp. Do đó, xe đạp sẽ là đối tượng trong mô hình hệ thống Wheels của chúng tôi và cuối cùng là mã. Các bit dữ liệu chúng ta cần lưu trữ về những chiếc xe đạp này, chẳng hạn như loại, tỷ lệ thuê hàng ngày và tiền đặt cọc, được gọi là các thuộc tính của đối tượng xe đạp.

Trong UML, một đối tượng được biểu diễn như trong Hình 4.2 dưới dạng hình chữ nhật có hai phần. Phần trên cùng là tên của đối tượng và phần thứ hai được sử dụng cho các giá trị thuộc tính đối tượng.



Trong trường hợp của Hình 4.2, tên của đối tượng là aBike: Bike. Tên đối tượng luôn được gạch chân và có thể có hai phần, một trong hai phần có thể được sử dụng riêng. Phần đầu tiên của tên, aBike, nhãn đối tượng; phần thứ hai ,: Bike xác định nó là một đối tượng của lớp Bike. Có một số cách mà chúng ta có thể đề cập đến một đối tượng: chúng ta có thể nói về 'một đối tượng của lớp Xe đạp', 'đối tượng Xe đạp' hoặc ': Xe đạp' - tất cả các tên này đều có cùng một ý nghĩa trong điều này bối cảnh.

Mỗi đối tượng thuộc về một lớp là một mẫu hoặc nhà máy để sản xuất các đối tượng. Tất cả các đối tượng của một lớp có cùng các thuộc tính, cùng một hành vi và các mối quan hệ giống nhau. Các lớp học sẽ được thảo luận chi tiết sau trong chương này.

Các sơ đồ đối tượng có thể được vẽ bằng các thuộc tính và các giá trị của chúng được hiển thị như trong Hình 4.2 hoặc không có các thuộc tính (như trong Hình 4.6). Trong Hình 4.2, các thuộc tính phổ biến cho tất cả các đối tượng Xe đạp là loại, DailyHireRate và tiền gửi và các giá trị cho đối tượng cụ thể này là 'men ~', '£8' và '£50'.

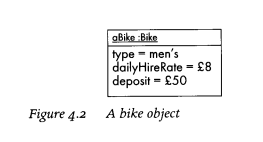
Tuy nhiên, có nhiều thứ cho một đối tượng hơn khả năng ghi dữ liệu của nó. Để hoàn thiện, bây giờ chúng ta sẽ đưa ra một định nghĩa chi tiết hơn về một đối tượng và thảo luận về ý nghĩa của các thuật ngữ được sử dụng trong định nghĩa. Một số cuộc thảo luận này sẽ liên quan đến các khái niệm mà chúng tôi chưa gặp, vì vậy đừng lo lắng nếu bạn không hiểu hết về nó vào lúc này; bạn có thể thấy hữu ích khi quay lại phần này sau khi bạn đã đọc về các lớp học.

Một đối tượng là một khái niệm, trừu tượng hoặc điều có ranh giới rõ ràng và ý nghĩa trong khu vực ứng dụng hiện tại. Chúng tôi sử dụng cả hai đối tượng để mô hình hóa các đặc điểm trong thế giới thực của khu vực ứng dụng và để cung cấp cho chúng tôi cơ sở để thực hiện máy tính.

Một đối tượng có thể đại diện cho một cái gì đó cụ thể trong thế giới thực, chẳng hạn như chiếc ghế này, chiếc xe đạp màu đỏ, xe của Ben hoặc một khái niệm như giao dịch tài chính (ví dụ: trả tiền cho một con tem), một đơn đặt hàng của khách hàng, giao dịch mượn sách từ thư viện v.v ... Một đối tượng luôn luôn là một sự trừu tượng (xem Chương 1) bởi vì mặc dù chúng ta muốn nó đại diện cho một thế giới thực, chúng ta chỉ quan tâm đến các khía cạnh nhất định của thế giới thực. Ví dụ: một đối tượng thành viên thư viện sẽ lưu trữ thông tin quan tâm trong hệ thống thư viện: tên và địa chỉ của thành viên nhưng không phải số hộ chiếu hoặc sở thích của cô ấy trong giày - chúng tôi bỏ qua các chi tiết hiện không liên quan. Chúng tôi chỉ bao gồm các thuộc tính của thế giới thực có ý nghĩa cho vấn đề trong tay.

Mỗi đối tượng trong một hệ thống có ba đặc điểm: behaviour, state and identity (hành vi, trạng thái và bản sắc.)

**Behaviour**.Cho đến nay, khái niệm về một đối tượng sẽ có vẻ khá quen thuộc với bất kỳ ai sử dụng cho các thực thể trong mô hình hóa mối quan hệ thực thể (xem Howe, 200l). Tuy nhiên, không giống như các thực thể, các đối tượng không chỉ lưu trữ thông tin, chúng có hành vi. Trong lịch sử, lý do mà các đối tượng có hành vi là ban đầu chúng được sử dụng trong mô phỏng máy tính. Một chiếc máy bay trong một mô phỏng máy tính lưu trữ thông tin: nó biết nó cao bao nhiêu, bao nhiêu nhiên liệu trong xe tăng của nó, nó bay nhanh như thế nào. Nó cũng có thể làm một số việc: nó có thể cất cánh, bay lên, bay về phía trước, rẽ và hạ cánh. Các đối tượng trong một hệ thống thông tin thường không được mô phỏng theo cùng một cách (hệ thống của chúng tôi sẽ không có những chiếc xe đạp chạy vòng quanh màn hình) nhưng chúng biết một số điều nhất định và chúng có thể làm một số điều nhất định. Một đối tượng xe đạp biết (cửa hàng) loại của nó, tỷ lệ thuê hàng ngày và tiền gửi. Khi chúng tôi thiết kế đối tượng xe đạp, chúng tôi quyết định những gì nó có thể làm. Hệ thống chắc chắn sẽ cần có khả năng lưu trữ, cập nhật và hiển thị các giá trị của các thuộc tính của từng đối tượng xe đạp. Chúng tôi cũng sẽ cần có thể tính ra tổng chi phí thuê trong một số ngày nhất định. Trong phát triển hướng đối tượng, các quy trình hoạt động trên các mục dữ liệu liên quan được gói cùng với dữ liệu. Trong thực tế, một đối tượng ít hơn một bó các mục dữ liệu và các quy trình thao túng chúng. Các quy trình này được gọi là hoạt động. Hành vi của một đối tượng được chia thành các hoạt động, mỗi hành vi đại diện cho điều gì đó mà đối tượng có thể làm (ví dụ: cập nhật DailyHireRate, hiển thị tiền gửi). Hành vi của một đối tượng được kích hoạt để phản hồi lại một thông điệp được gửi từ một đối tượng khác yêu cầu nó thực hiện một trong các hoạt động của nó. Chúng ta sẽ thảo luận về các thông điệp sau trong chương này.

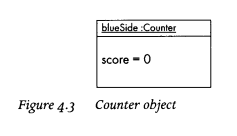


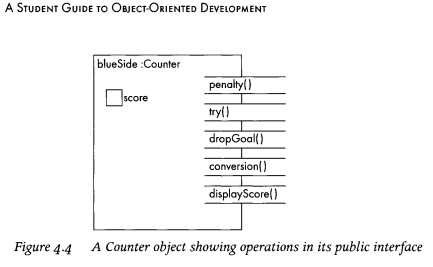
**State.** Hầu hết các đối tượng có thuộc tính; ví dụ. Type, DailyHireRate và tiềngửi (xem Hình 4.2). Các thuộc tính này có giá trị, ví dụ: Type = nam, DailyHireRate = £8 và tiềngửi = £50. Thông thường các giá trị thuộc tính có thể thay đổi - DailyHireRate có thể tăng hoặc chúng tôi có thể thay đổi số tiền chúng tôi yêu cầu khách hàng để lại dưới dạng tiền gửi. Trạng thái của một đối tượng được xác định bởi các giá trị của các thuộc tính và các liên kết của nó với các đối tượng khác. Ví dụ, chúng ta có thể biết nếu một đối tượng tài khoản ngân hàng ở trạng thái rút tiền bằng cách xem giá trị của số dư thuộc tính. Lý do chúng tôi quan tâm đến trạng thái của một đối tượng là hành vi của nó có thể thay đổi tùy thuộc vào trạng thái của nó. Một ví dụ rõ ràng về điều này là loại tài khoản ngân hàng nơi chủ tài khoản không được phép rút tiền. Trong trường hợp này, cách tài khoản phản hồi khi yêu cầu rút tiền phụ thuộc vào số tiền trong tài khoản. Nếu có đủ tiền trong tài khoản, việc rút tiền được cho phép. Nếu không, việc rút tiền không được phép. Trạng thái và ảnh hưởng của nó đối với hành vi của đối tượng được thảo luận đầy đủ trong Chương 7.

**Identity.**. Khi chúng ta nói rằng một đối tượng có danh tính, chúng ta có nghĩa là mỗi đối tượng là duy nhất, nó có một sự tồn tại riêng biệt và cuối cùng là một không gian riêng trong bộ nhớ máy tính từ mọi đối tượng khác. Mỗi chiếc xe đạp trong hệ thống Bánh xe sẽ được đại diện bởi một đối tượng riêng biệt trong mã. Ngay cả hai đối tượng có thuộc tính có giá trị giống hệt nhau hoàn toàn khác biệt với nhau. Trong triển khai máy tính, các đối tượng được xác định bởi một tham chiếu do máy tính tạo ra (id đối tượng) là mã nội bộ thường không hiển thị cho người lập trình; đại khái là nó tương ứng với một vị trí bộ nhớ. Điều này khá tách biệt với bất kỳ số nhận dạng nào liên quan đến kinh doanh, chẳng hạn như số khách hàng hoặc số xe đạp hoặc bất kỳ tên nào do lập trình viên tạo cho một đối tượng. Tất cả xe đạp của wheels đều có số được phân bổ bởi cửa hàng xe đạp cho mục đích kinh doanh. Những con số này được ghi lại là một trong những thuộc tính của đối tượng xe đạp, nhưng chúng không phải là cách phần mềm máy tính xác định đối tượng xe đạp. Theo như lập trình viên hoặc các phần khác của mã, các đối tượng phải được đặt tên và giải quyết bằng tên của chúng. Ví dụ: nếu danh sách 600 xe đạp của Wheels được lưu trữ trong một **wheelsBikes[6oo](bánh xe mảng [6oo])** và chúng tôi muốn biết tỷ lệ thuê hàng ngày của một trong những chiếc xe đạp trong mảng, chúng tôi sẽ gửi một thông điệp gửi đến chiếc xe đạp cụ thể đó (giả sử số lo 5 trong mảng) yêu cầu nó hiển thị tỷ lệ thuê hàng ngày của nó -wheelsBike [xos].showDailyHireRate().wheelBike[xos] là tên của đối tượng xe đạp mà chúng ta quan tâm, showDailyHireRate() là hoạt động mà chúng ta muốn nó thực hiện.

*Đóng gói và ẩn dữ liệu*

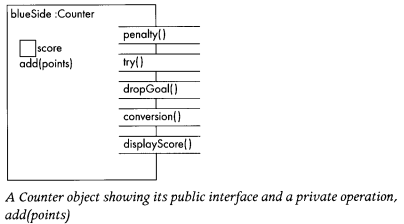
Một trong những lợi thế của phần mềm hướng đối tượng là các đối tượng đóng gói dữ liệu. Dữ liệu có thể được ẩn bên trong một đối tượng theo cách nó được bảo vệ và không thể được truy cập trực tiếp bởi các phần khác của chương trình. Ưu điểm lớn của việc này là nó không thể vô tình bị hỏng. (Tìm ví dụ khác) Ví dụ, chúng ta hãy thiết kế một bộ đếm để theo dõi điểm số trong trò chơi bóng bầu dục. Chúng ta có một đối tượng được gọi là blueSide: Counter với điểm thuộc tính như trong Hình 4.3 - blueSide là tên của đối tượng, Counter là lớp của nó.





Hệ thống tính điểm trong một trận bóng bầu dục là một bên được 3 điểm cho một quả phạt đền, 3 cho một mục tiêu thả 5 cho một lần thử và 2 cho một chuyển đổi. Nếu bạn không quen với trò chơi bóng bầu dục thì đừng lo lắng về các điều khoản này. Tất cả những gì chúng ta cần hiểu là chúng ta chỉ có thể tăng thuộc tính điểm lên 2, 3 hoặc 5. Trong Hình 4.4, chúng ta đã mô tả điểm thuộc tính bên trong bức tường bảo vệ của một hình chữ nhật đại diện cho đối tượng blueSide: Counter. 2 Điểm duy nhất có thể được cập nhật là bằng cách sử dụng một trong các thao tác chúng tôi đã xác định cho đối tượng này: penalty(), try(), dropGoal() hoặc convert(). Điều này có nghĩa là không có phần nào khác của chương trình có thể vô tình (hoặc cố tình!) Thay đổi điểm số bằng cách sử dụng số học số nguyên thông thường, ví dụ để trừ lo hoặc nhân với 4. Dữ liệu được bảo vệ bởi các hoạt động đóng gói nó. Trong thực tế, phần còn lại của chương trình thậm chí không cần biết rằng chúng tôi đã triển khai nó như một số nguyên. Tất cả phần còn lại của chương trình cần biết là tên của đối tượng và tên của các hoạt động mà nó có thể sử dụng để cập nhật hoặc hiển thị chính xác điểm số của blueSide. Điều này ngay lập tức được gọi là ẩn dữ liệu.

*Public interface and messages.* Trong ví dụ trong Hình 4.4, các hình phạt hoạt động (), try(), dropGoal(), convert() và displayScore() là những gì được gọi là đối tượng Counter ~ giao diện công cộng - tất cả các hoạt động này được định nghĩa là công khai, tức là có sẵn để sử dụng bởi các phần khác của chương trình. Mỗi hoạt động có một chữ ký, ví dụ displayScore(), tạo thành giao diện chung của hoạt động và phải được sử dụng khi thao tác được gọi.

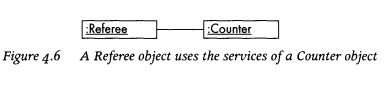


Chữ ký bao gồm hoạt động tên, danh sách tham số, loại kết quả và một dấu hiệu cho biết đó là hoạt động riêng tư hay công khai. Giao diện chung của một đối tượng là điều duy nhất phần còn lại của chương trình biết về đối tượng ngoài tên của nó. Giao diện công cộng cung cấp các dịch vụ mà nó cung cấp cho các đối tượng khác. Nếu một đối tượng khác muốn cập nhật điểm số của blueSide - bởi vì bên màu xanh đã ghi điểm - nó không thể thực hiện trực tiếp, nên nó phải làm điều đó bằng cách gửi tin nhắn đến đối tượng blueSide, yêu cầu nó cập nhật điểm số. Thông báo phải xác định chính xác đối tượng được gửi và chữ ký của hoạt động. Nó phải ở một định dạng cụ thể: bắt đầu bằng tên của đối tượng, theo sau là dừng hoàn toàn, theo sau là chữ ký của thao tác sẽ được gọi, ví dụ: blueSide.try (). Để thông báo hoạt động, nó phải giải quyết chính xác đối tượng theo tên và thao tác mà nó chỉ định phải là một phần của giao diện chung của đối tượng đó.

Bộ đếm có thể có các hoạt động khác được thiết kế để sử dụng nội bộ của riêng nó, ví dụ: nó có thể có add(points) được sử dụng bởi hầu hết các hoạt động trong giao diện công cộng, để thêm một số điểm được chỉ định vào điểm thuộc tính (xem Hình 4.5 ). Các hoạt động nội bộ như vậy được gọi là các hoạt động riêng tư và không phải là một phần của các dịch vụ mà đối tượng Counter cung cấp cho các đối tượng khác; chúng tạo thành một giao diện riêng cho một đối tượng để gửi tin nhắn đến chính nó.

*Dependencies(phụ thuộc).* Như chúng ta đã thấy, một ưu điểm của việc sử dụng giao diện công cộng được xác định rõ ràng là nó đóng gói dữ liệu, một ưu điểm khác là nó làm rõ sự phụ thuộc giữa các cấu trúc phần mềm và hạn chế thiệt hại có thể được thực hiện bởi các phụ thuộc. Hai mô-đun được cho là phụ thuộc lẫn nhau nếu một mô đun sử dụng dịch vụ của nhau. Điều này được gọi là mối quan hệ máy khách-máy chủ.

Sự phụ thuộc giữa các mô-đun có thể gây ra vấn đề với bảo trì nếu chúng không được kiểm soát cẩn thận. Ngay khi một sự phụ thuộc được thiết lập, ít nhất là về mặt lý thuyết, việc thay đổi thành một mô-đun có thể ảnh hưởng đến mô-đun kia. Ví dụ: nếu chúng ta đã thiết kế ví dụ truy cập ở trên theo cách mà điểm số có thể nhìn thấy được đối với các mô-đun khác và có thể bị thao túng trực tiếp bởi các quy trình trong các mô-đun khác, chúng ta sẽ phải rất cẩn thận nếu thay đổi điểm số. Nếu chúng tôi quyết định thực hiện điểm số như một thực tế hoặc một



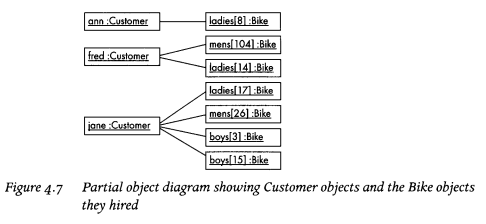
số tự nhiên, tất cả các mô-đun biết rằng điểm số là một số nguyên sẽ phải được thay đổi. Tuy nhiên, nếu các mô-đun giao tiếp với nhau bằng thông điệp truyền qua giao diện công cộng, như các đối tượng thực hiện, thì thay đổi thành mã thực hiện các hoạt động không ảnh hưởng đến các mô-đun máy khách; miễn là giao diện không bị thay đổi, chúng không bị ảnh hưởng. Điều này đúng với phần thân mã thực tế cũng như các cấu trúc dữ liệu liên quan. Ví dụ: nếu chúng tôi quyết định thay đổi mã thực hiện một thao tác để làm cho nó thực thi nhanh hơn, miễn là chúng tôi không thay đổi giao diện, sẽ không có đối tượng khách hàng nào bị ảnh hưởng. Việc sử dụng giao diện công cộng không chỉ che giấu dữ liệu mà còn ẩn mã.

Sự phụ thuộc cũng có thể làm phức tạp việc tái sử dụng các mô-đun. Ví dụ: chúng ta giả sử đối tượng Counter được sử dụng trong mô phỏng trò chơi bóng bầu dục trong đó đối tượng Trọng tài sử dụng đối tượng Counter để cập nhật điểm số như trong Hình 4.6. Đối tượng Trọng tài sử dụng các dịch vụ của, tức là khách hàng của, đối tượng Counter.

Điều này có nghĩa là, nếu chúng tôi muốn sử dụng lại đối tượng Trọng tài trong một chương trình khác, chúng tôi cũng sẽ phải sử dụng đối tượng Counter. Trong trường hợp này không thực sự có vấn đề, bởi vì chúng tôi biết về sự phụ thuộc, nó được tuyên bố rõ ràng. Nếu chúng tôi muốn lưu trữ Trọng tài để tái sử dụng, chúng tôi có thể lưu trữ gói cùng với Counter trong một gói. Bạn có thể đọc về các gói trong Chương 5.

Các vấn đề phát sinh nếu hai mô-đun phụ thuộc vào nhau theo những cách không được khai báo. Trong các chương trình được viết bằng ngôn ngữ lập trình sớm, có thể nhảy mà không bị giới hạn ở phần khác của chương trình hoặc mô-đun khác, thực thi một vài dòng mã hoặc truy cập một số dữ liệu khi bạn đến đó, sau đó nhảy trở lại. Những bước nhảy như vậy làm cho rất khó đọc qua mã và hiểu những gì đang xảy ra. Nếu một lập trình viên bảo trì phải thay đổi một phần mã, thật khó để biết phần còn lại của chương trình sẽ bị ảnh hưởng như thế nào. Trừ khi anh ta đọc đúng tất cả các mã, anh ta không thể thấy phần nào của chương trình có thể sử dụng phần anh ta vừa thay đổi: đây là vấn đề về tác dụng phụ. Khai báo một giao diện công cộng và sử dụng ngôn ngữ lập trình để thực thi nó (như Java hoặc C ++) do đó có hai lợi thế lớn để bảo trì.

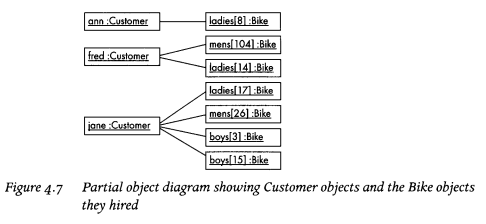
* Nó làm cho nó rõ ràng hơn nhiều đối với một người đọc mã mà mối quan hệ máy khách-máy chủ tồn tại. Điều đó làm cho công việc thay đổi mã nhanh hơn, bởi vì nó dễ dàng hơn để hiểu và bởi vì người bảo trì không phải đọc toàn bộ chương trình để kiểm tra các phụ thuộc, chỉ các mô-đun bị ảnh hưởng.



* Trừ khi giao diện được thay đổi, mã bên trong của mô-đun có thể được thay đổi mà không ảnh hưởng đến bất kỳ mô-đun máy khách nào. Theo nghĩa này, mã nội bộ của một mô-đun (cũng như bất kỳ dữ liệu nào) được gói gọn, nó được ẩn khỏi phần còn lại của chương trình.

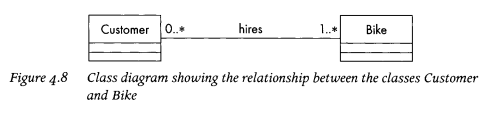
**What is a class?**

Cho đến nay cuộc thảo luận của chúng tôi đã hoàn toàn về các đối tượng và sơ đồ đối tượng. Trên sơ đồ đối tượng, mỗi đối tượng được biểu thị bằng một hình chữ nhật, như trong Hình 4-7.

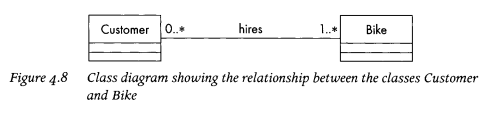


Hình 4.7 đại diện cho ba khách hàng và những chiếc xe đạp họ thuê; điều này có thể mô hình hóa các giao dịch thuê diễn ra trong một giờ tại cửa hàng xe đạp Wheels. Trong sơ đồ này, chúng tôi đang sử dụng các chuỗi đơn giản (ví dụ: ann) cho tên đối tượng Khách hàng và tham chiếu mảng (ví dụ: ladies[8]) cho tên của các đối tượng Xe đạp.

Chúng ta có thể thấy rằng, bằng cách sử dụng sơ đồ đối tượng như thế này, chúng ta sẽ nhanh chóng hết dung lượng nếu chúng ta cố gắng mô hình hóa tất cả các giao dịch thuê diễn ra trong một tuần; để mô hình hóa tất cả các giao dịch diễn ra trong một năm sẽ không còn nữa. Vấn đề với sơ đồ đối tượng là chúng chiếm rất nhiều không gian. Chúng rất hữu ích để khám phá các khu vực phức tạp hoặc lẫn lộn của hệ thống, nhưng chúng không phải là một cách hiệu quả để mô hình hóa bộ đối tượng hoàn chỉnh trong một hệ thống. Wheels có 600 xe đạp; nếu họ thuê họ tới 5000 khách hàng trong suốt một năm, chúng tôi sẽ không cố gắng mô hình hóa tất cả các thuê này trên một sơ đồ đối tượng. Nó là hiệu quả và thuận tiện hơn để suy nghĩ về các lớp của các đối tượng.



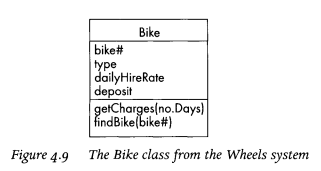
Một lớp các đối tượng là một nhóm các đối tượng có cùng một tập các thuộc tính, cùng các mối quan hệ và cùng một hành vi. Khi chúng ta nghiên cứu một miền vấn đề, có lẽ chúng ta sẽ bắt đầu bằng cách tìm các đối tượng trong đó và, khi chúng ta đã thực hiện điều đó, hãy tìm ra những gì chúng ta cần. Trong hệ thống Wheels, chúng tôi biết chúng tôi có 600 đối tượng xe đạp để đại diện. Bước tiếp theo là đồng ý về một tập hợp các thuộc tính mà những chiếc xe đạp có điểm chung. Sau đó, chúng ta cần tìm ra những gì chúng ta muốn các đối tượng xe đạp của chúng ta có thể làm được. Một khi chúng ta đã thực hiện rằng một lớp Bike có thể được xác định với tập các thuộc tính đã được thống nhất và các hoạt động có thể cung cấp hành vi cần thiết. Sau đó chúng ta có thể suy nghĩ và mô hình hóa về các lớp đối tượng và mối quan hệ của chúng với nhau. Để thể hiện việc thuê 600 chiếc xe đạp của Wheels bởi 5000 khách hàng, thay vì có sơ đồ đối tượng với vài nghìn đối tượng trên đó, chúng ta có thể tóm tắt thông tin trên sơ đồ lớp như trong Hình 4.8. Ký hiệu được giải thích sau trong chương này.



Mặc dù chúng ta bắt đầu bằng cách xác định các đối tượng, nhưng đó là lớp xác định cấu trúc và hành vi của các đối tượng. Chúng ta có thể so sánh một lớp học với một máy cắt bánh nướng, hoặc một trong những máy cắt có hình động vật mà trẻ em sử dụng để cắt các hình vẽ ra khỏi bột nhão. Một máy cắt có thể được sử dụng để tạo ra vô số bánh nướng hoặc hình động vật. Một lớp được sử dụng, như máy cắt scone, làm mẫu để tạo các đối tượng ở dạng của nó. Sản xuất đồ vật là vai trò chính của lớp học trong cuộc sống; một lớp là một nhà máy sản xuất đối tượng, nó có thể tạo ra hàng trăm đối tượng, tất cả đều có cùng cấu trúc và hành vi. Khi chúng ta định nghĩa một lớp, chúng ta định nghĩa cấu trúc và giao diện chung cho tất cả các đối tượng của lớp đó.

Trong phần mềm hướng đối tượng, tất cả các đối tượng thuộc về một lớp. Mã hướng đối tượng được thiết kế tốt bao gồm toàn bộ các đối tượng (và các lớp của chúng); tất cả các chức năng hệ thống được tạo ra bởi các hoạt động mà chúng ta xác định cho các lớp. Trong phần mềm, mã thực hiện các hoạt động được đặt trong lớp. Các đối tượng của lớp biết và truy cập các hoạt động này, nhưng không mang theo bản sao của chúng. Ví dụ, một đối tượng Bike biết rằng nó có thể hiển thị số của nó và tính ra chi phí thuê nó (hay đúng hơn là chiếc xe đạp mà nó đại diện) trong một số ngày nhất định. Tuy nhiên, mã cho các hoạt động này là một phần của lớp Bike.

Các đối tượng đôi khi được gọi là các thể hiện của các lớp; quá trình tạo một đối tượng mới thuộc về một lớp được gọi là khởi tạo (instantiation).



Mỗi đối tượng của một lớp nhất định sẽ có cùng một tập các thuộc tính và cùng một tập hợp các hoạt động. Tuy nhiên, mặc dù chúng có cùng cấu trúc thông tin và cùng một tập hợp các hành vi có thể, mỗi hành vi sẽ có một bộ giá trị riêng cho các thuộc tính của mình và mỗi loại sẽ có bản sắc riêng. Trong hệ thống Wheels, mỗi chiếc xe đạp thuộc sở hữu của cửa hàng và mọi khách hàng thuê xe đạp sẽ được đại diện bởi một đối tượng trong phần mềm.

Hình 4.9 cho thấy lớp Bike từ hệ thống Wheels.

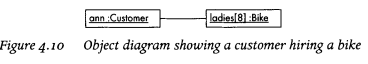
Biểu tượng UML cho một lớp là một hình chữ nhật được chia thành ba phần. Phần trên cùng được sử dụng cho tên lớp, phần giữa cho các thuộc tính và phần dưới cùng cho các hoạt động. UML không có bất kỳ quy tắc đặt tên nào; trong cuốn sách này, chúng tôi sử dụng một quy ước đặt tên, được sử dụng bởi hầu hết các học viên hướng đối tượng, dựa trên phong cách lập trình hướng đối tượng. Tên lớp bắt đầu bằng chữ in hoa với phần còn lại của tên bằng chữ thường. Nếu tên lớp bao gồm hai hoặc nhiều từ, các từ được chạy cùng nhau mà không có khoảng trắng và chữ cái đầu tiên của mỗi từ được viết hoa, ví dụ: BikeList, FlowerArrangingTalk. Tên lớp luôn là số ít.

Lưu ý rằng tên thuộc tính được viết bằng chữ thường. Nếu tên thuộc tính bao gồm hai hoặc nhiều từ, các từ được chạy cùng nhau không có khoảng trắng và chữ cái đầu tiên của mỗi từ, ngoại trừ từ đầu tiên, được viết hoa, ví dụ: hàng ngàyHireRate. Các hoạt động sử dụng cùng một ký hiệu, ví dụ: getCharges ().

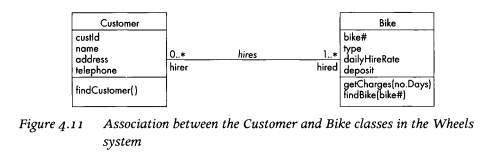
**Relationships between classes (Mối quan hệ giữa các lớp)**

Có ba loại mối quan hệ giữa các lớp: liên kết, tổng hợp và kế thừa. (association, aggregation and inheritance.)

*Association.* Một trong những đặc điểm chính của một hệ thống hướng đối tượng là các đối tượng của nó hợp tác để đạt được các chức năng cần thiết. Để điều này xảy ra, họ phải có khả năng giao tiếp với nhau. Họ làm điều này bằng cách nhắn tin đi qua. Các đối tượng sẽ không thể truyền tin nhắn cho nhau trừ khi chúng ta xây dựng các liên kết giữa các lớp, một tuyến đường để chúng nói chuyện với nhau - một con đường có thể điều hướng.



Hình 4.10 là sơ đồ đối tượng hiển thị: Khách hàng được liên kết với "Xe đạp. Nó mô hình mối quan hệ ngoài đời thực của khách hàng thuê xe đạp. Ngược lại, trong sơ đồ lớp (giống như trong Hình 4.11), mối quan hệ liên kết đơn giản được xây dựng vào mô hình khả năng các đối tượng được liên kết, để "Khách hàng có thể được liên kết với" Xe đạp theo yêu cầu. Hiệp hội không cho chúng tôi biết 9 Khách hàng nào thực sự được liên kết với "Xe đạp nào, như sơ đồ đối tượng trong Hình 4.7 không, chỉ là họ có thể được liên kết. Một liên kết đại diện cho một nhóm các liên kết giữa các đối tượng theo cách tương tự như một lớp đại diện cho một nhóm các đối tượng.

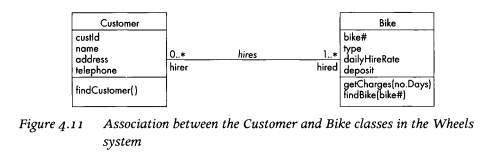


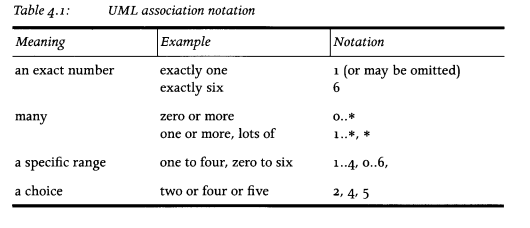
Trong giai đoạn đầu của mô hình hóa, chúng ta chưa biết chi tiết cách các đối tượng sẽ cần giao tiếp. Ở giai đoạn này, khi chúng ta mô hình hóa một liên kết giữa các lớp đối tượng, chúng ta không nói nhiều hơn là kết nối thực tế tồn tại giữa các đối tượng này và có thể được sử dụng bởi các đối tượng khi cần.

Hình 4.11 cho thấy mối quan hệ liên kết giữa các lớp Khách hàng và Xe đạp.

Chúng ta có thể thấy trong hình 4.11 rằng một hiệp hội có thể được đặt tên là "một khách hàng thuê một chiếc xe đạp. Một liên kết có hai đầu, mỗi đầu được gắn vào một lớp. Mỗi đầu kết hợp có thể có một tên vai trò: trên liên kết giữa Khách hàng và Xe đạp kết thúc liên kết bên cạnh Khách hàng là người thuê, bên cạnh Xe đạp được thuê. Tên mặc định cho kết thúc liên kết là tên của lớp được đính kèm, ví dụ, trong Hình 4.1x, tên mặc định cho kết thúc liên kết được gắn với Khách hàng sẽ là khách hàng Trong thực tế, cả tên hiệp hội và tên vai trò đều bị bỏ qua, trừ khi chúng hỗ trợ đáng kể cho sự hiểu biết.

Association kết thúc cũng có bội. Đa bội được biểu thị bằng số và dấu sao trên dòng. Tính đa bội của một hiệp hội biểu thị các giới hạn về số lượng đối tượng được phép tham gia





mối quan hệ. Chúng ta phải đọc bội số riêng cho mỗi kết thúc liên kết. Để diễn giải sự liên kết giữa Khách hàng và Xe đạp trong Hình 4.1x, trước tiên chúng tôi xem xét số lượng: Xe đạp a: Khách hàng có thể thuê, sau đó là bao nhiêu: Khách hàng a: Xe đạp có thể được thuê. Để biết được có bao nhiêu: Xe đạp một: Khách hàng có thể thuê chúng tôi xem xét sự liên kết từ đầu lớp Khách hàng đến số được chỉ định ở cuối lớp Xe đạp. Trong trường hợp này, số là x .... Điều này có nghĩa là một khách hàng có thể thuê x hoặc nhiều xe đạp. Trong các thuật ngữ đối tượng, sơ đồ chỉ định rằng một: Khách hàng có thể được liên kết với x hoặc nhiều: Xe đạp. Giải thích tính đa bội theo hướng khác, sơ đồ chỉ định rằng bất kỳ ai: Xe đạp có thể được thuê bởi o, x hoặc nhiều (o .. \*): Khách hàng. Chúng tôi giả định điều này có nghĩa là trong một khoảng thời gian. Tính đa bội luôn được chỉ định theo quan điểm của một đối tượng.

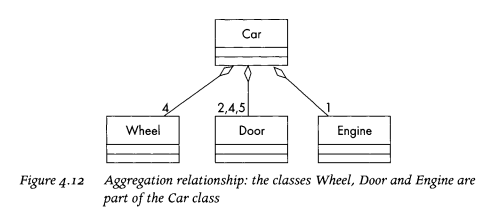
Ký hiệu bội số UML cho phép chúng ta chỉ định các mức độ liên kết như trong Bảng 4.1

*Aggregation* thường được coi là một hình thức liên kết chặt chẽ hơn; nó mô hình mối quan hệ toàn bộ giữa các lớp, ví dụ: bánh xe, cửa ra vào và động cơ là một phần của xe hơi. Một mối quan hệ tổng hợp có thể được xác định:

* Nếu cụm từ như 'bao gồm', 'có một' hoặc 'là một phần của' được sử dụng để mô tả mối quan hệ
* Nếu một lớp trong mối quan hệ (toàn bộ) quan trọng hơn lớp kia (phần)
* Nếu một số hoạt động áp dụng cho toàn bộ và các bộ phận của nó.

Hình 4.12 cho thấy mối quan hệ kết hợp giữa một chiếc xe và các bộ phận của nó: bánh xe, cửa và động cơ.

Trong ký hiệu UML, tập hợp được hiển thị dưới dạng một đường nối hai lớp với một viên kim cương bên cạnh cả lớp, xem Hình 4.12. Lưu ý rằng bội số có thể được chỉ định ở phần cuối của



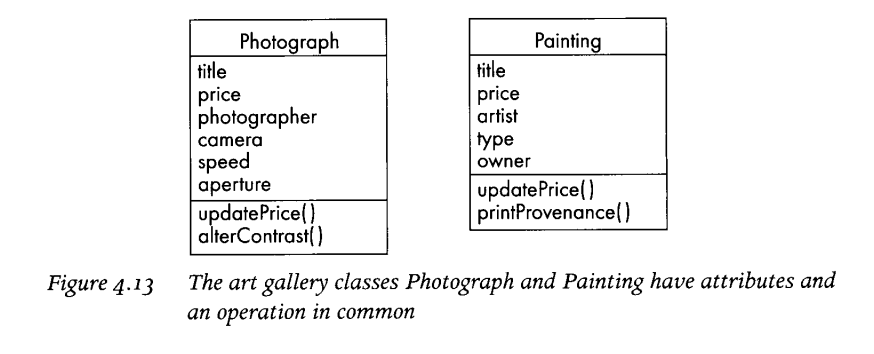
mối quan hệ, sử dụng các ký hiệu tương tự như đối với hiệp hội. Toàn bộ kết thúc luôn được coi là một, một đối tượng bánh xe, ví dụ, chỉ là một phần của một chiếc xe. Trong hình 4.12, một chiếc xe có bốn bánh, hai, bốn hoặc năm cửa và một động cơ.

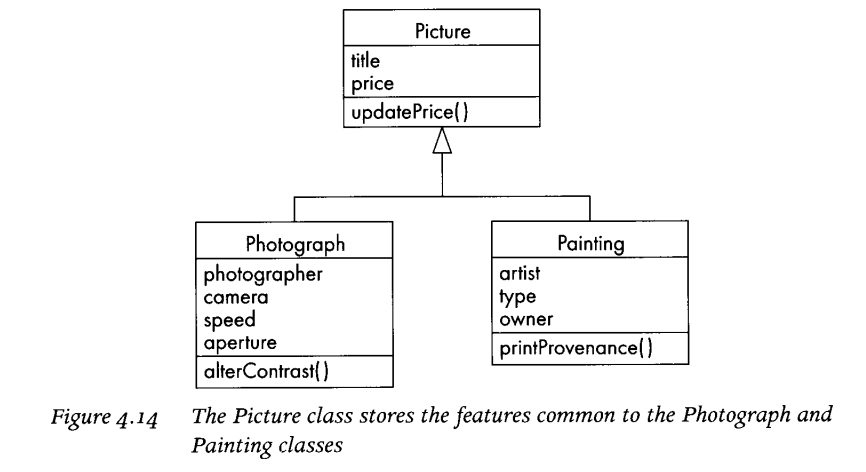
Trong những ngày trước khi UML được xuất bản, đã có nhiều cuộc tranh luận sôi nổi về tầm quan trọng và ý nghĩa của tập hợp. UML bao gồm tập hợp, nhưng không có định nghĩa chính xác. Có rất ít để phân biệt một mối quan hệ kết hợp với một mối quan hệ kết hợp. Vì đây là trường hợp, chúng tôi cảm thấy rằng tập hợp thường thêm ít ý nghĩa của một mô hình và không phải đưa vào. UML bao gồm một hình thức tổng hợp mạnh hơn, được gọi là thành phần. Thành phần là hữu ích vì nó có một ý nghĩa được xác định chính xác; nó được thảo luận trong phần Điểm kỹ thuật của chương này.

*Inheritance and generalization(kế thừa và khái quát hóa).* Nếu chúng ta nhận thấy, trong khi mô hình hóa các lớp, một số trong số chúng chia sẻ một số thuộc tính và hoạt động chung, việc giới thiệu một lớp mới cho các bit được chia sẻ, chỉ để lại các tính năng phân biệt trong các lớp gốc. Quá trình này được gọi là tổng quát hóa. Ví dụ, trong Hình 4.13, chúng ta có hai lớp mà chúng ta có thể tìm thấy trong một hệ thống cho một phòng trưng bày nghệ thuật: Chụp ảnh và Vẽ tranh. Cả hai lớp này đều có tiêu đề thuộc tính và giá cả và hoạt động updateprice().

Chúng ta có thể tạo một lớp chung mới, Ảnh, trong đó chúng ta có thể đặt các tính năng phổ biến này; điều này được thể hiện trong hình 4.14. Ảnh và Tranh giữ lại những nét đặc sắc của chúng và chia sẻ những hình ảnh.

Trong cấu trúc trong Hình 4.14, các lớp Chụp ảnh và Vẽ tranh là các chuyên ngành của lớp học chung, Vẽ tranh. Mối quan hệ giữa một lớp chung và các chuyên ngành của nó được gọi là mối quan hệ thừa kế. Cơ chế kế thừa cho phép các lớp chuyên biệt chia sẻ hoặc kế thừa các tính năng của lớp chung. Ký hiệu UML cho mối quan hệ thừa kế là một mũi tên có đầu mở chỉ từ lớp chuyên biệt sang lớp chung.

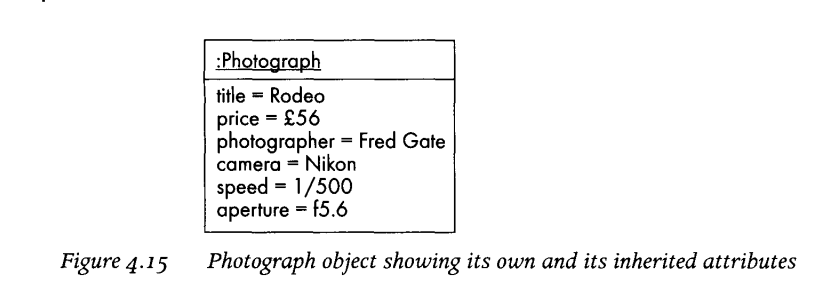




Có nhiều cách khác nhau để mô tả mối quan hệ thừa kế:

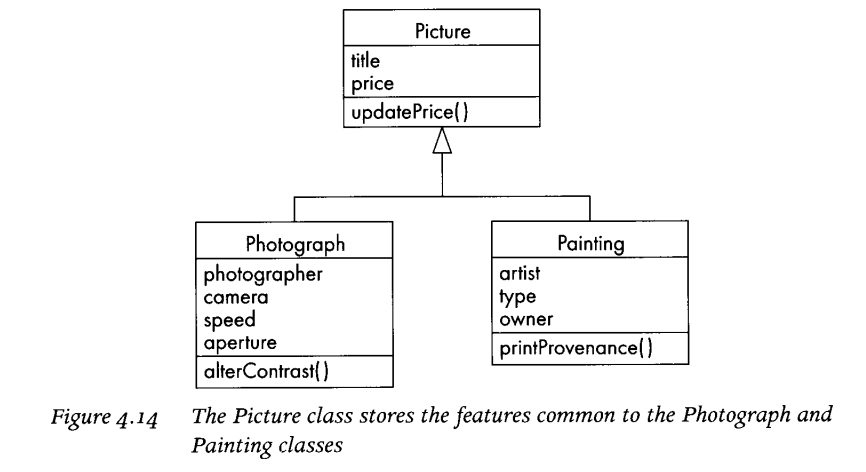
* Một lớp chuyên môn kế thừa từ một lớp chung.
* Một lớp con kế thừa từ một siêu lớp.
* Một lớp con kế thừa từ một lớp cha.
* Một lớp dẫn xuất kế thừa từ một lớp cơ sở.

Nhìn vào quy trình theo cách khác, nó cũng hữu ích để có thể tạo các lớp mới từ các lớp hiện có; thay vì bắt đầu từ đầu, chúng ta có thể tinh chỉnh những cái chúng ta đã có. Điều này cho phép tái sử dụng các lớp từ một hệ thống khác hoặc có thể từ một thư viện lớp. Lớp chuyên biệt có thể điều chỉnh lớp chung hơn cho phù hợp với hệ thống mới bằng cách thêm các thuộc tính hoặc thao tác. Các hoạt động được kế thừa có thể được ghi đè, tức là mã được kế thừa có thể được thay thế bằng mã mới thực hiện thao tác theo một cách khác.

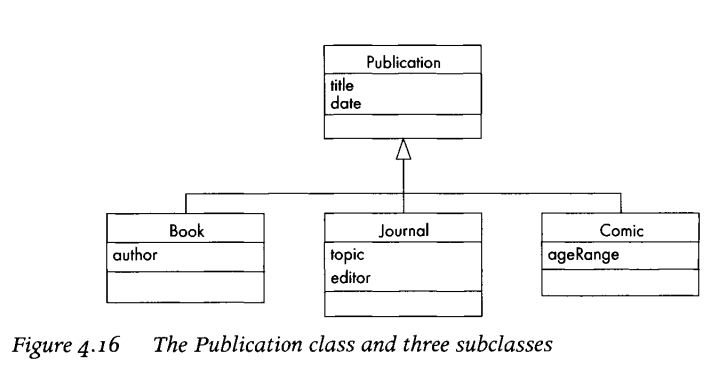


Khi chúng ta tạo một lớp chuyên biệt, nó kế thừa tất cả các thuộc tính, hoạt động và mối quan hệ của lớp cha. Trong Hình 4.14, các lớp chuyên biệt Chụp ảnh và Vẽ tranh kế thừa tiêu đề và giá thuộc tính và hoạt động updateprice() từ lớp Ảnh; chúng cũng có các thuộc tính và hoạt động chỉ liên quan đến các đối tượng của lớp chuyên biệt của chúng. Lưu ý rằng các đặc điểm được kế thừa không được hiển thị trong các lớp con; đây là những tính năng được chia sẻ để chứng minh sự khái quát hóa. Nền kinh tế đại diện này đơn giản hóa sơ đồ. Tuy nhiên, các tính năng được kế thừa là một phần của cấu trúc của đối tượng kế thừa. Một đối tượng Chụp ảnh sẽ có các thuộc tính: tiêu đề và giá cả (được kế thừa từ Ảnh) cũng như nhiếp ảnh gia, máy ảnh, tốc độ và khẩu độ (xem Hình 4.15). Nó sẽ biết về bản cập nhật hoạt động được cập nhậtpriceprice () cũng như hoạt động notifyContrast().

Khi chúng ta tạo một tập hợp các lớp con, chúng ta phải có một số cơ sở để phân biệt các lớp con với nhau; điều này được gọi là phân biệt đối xử. Trong các lớp thư viện nghệ thuật trong Hình 4.14, cơ sở cho sự khác biệt là loại hình ảnh. Thông thường sẽ có một lớp con cho mỗi giá trị có thể có của bộ phân biệt đối xử. Một ví dụ khác, trong Hình4.16, bộ phân biệt đối xử là loại ấn phẩm; trong trường hợp này có ba loại xuất bản, do đó có ba lớp con.’



Nó rất dễ lạm dụng thừa kế; nó chỉ là một kỹ thuật hữu ích nếu được sử dụng đúng cách Trước đây, các lập trình viên đã từng cố gắng giới thiệu một lớp từ một hệ thống khác và chuyên môn hóa nó một cách đơn giản để họ có thể sử dụng một lớp nếu phương thức của nó. 4 Lớp được sử dụng lại có thể không có gì chung với hệ thống mới ngoại trừ một thuật toán hữu ích. Điều này có thể gây nhầm lẫn cho những người đọc mã; kỳ vọng của họ về mối quan hệ thừa kế bị nhầm lẫn. Các lớp không nên được liên kết bằng sự kế thừa trừ khi có một mối quan hệ giữa chúng và giữa chúng: trong phòng trưng bày nghệ thuật, ví dụ, bức ảnh và tranh vẽ đều là một loại hình.



Một mối quan hệ tổng quát / kế thừa có thể được xác định:

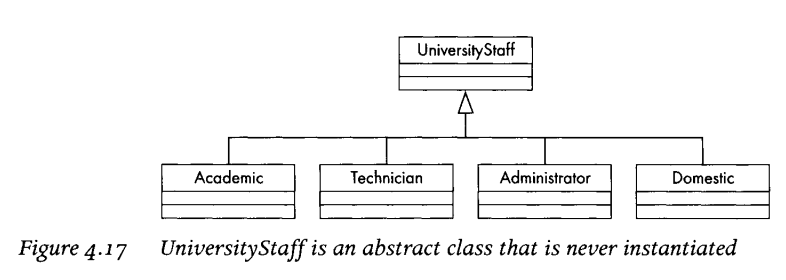
* Nếu các cụm từ như 'is-a' hoặc 'is-a-kind-of' có thể được sử dụng để mô tả mối quan hệ giữa các lớp, ví dụ: một tạp chí là một loại ấn phẩm, một con ngựa là 1 l loại động vật có vú
* Trong đó một hoặc nhiều lớp có các thuộc tính và hoạt động rất giống nhau và việc giới thiệu một lớp chung sẽ đơn giản hóa mô hình.

Về mặt lý thuyết, không có giới hạn về số lượng mức cho phép trong hệ thống phân cấp thừa kế; trong thực tế, người ta đã thấy rằng một hệ thống phân cấp với hơn sáu cấp độ trở nên không thể quản lý được.

Khái quát hóa và kế thừa là các kỹ thuật hữu ích bởi vì, như chúng ta đã thấy, chúng cho phép tái sử dụng các lớp hiện có. Phân loại các lớp thành một hệ thống phân cấp thừa kế cũng có nghĩa là chúng ta tránh lặp lại mã. Các hoạt động được kế thừa nằm trong lớp cha; các lớp con không cần phải mang theo phiên bản riêng của chúng, trừ khi chúng sẽ chuyên môn hóa hoạt động (xem đa hình bên dưới). Điều này có nghĩa là nếu chúng ta thay đổi mã thực hiện một thao tác, chúng ta chỉ cần thay đổi nó một lần, trong lớp cha.

Phân loại các lớp thành một hệ thống phân cấp thừa kế cũng giúp tổ chức và đơn giản hóa sự hiểu biết của chúng ta về các lớp trong hệ thống. Các lớp phù hợp để phân loại như vậy có cả điểm tương đồng và khác biệt; một hệ thống phân cấp thừa kế nhấn mạnh cả hai. Để tránh sự lặp lại không cần thiết và làm rõ sự hiểu biết của chúng tôi về các lớp, các thuộc tính và hoạt động được xác định ở mức áp dụng cao nhất trong hệ thống phân cấp. Sự khác biệt được phản ánh trong các thuộc tính và hoạt động được thêm hoặc xác định lại là các tính năng phân biệt của các lớp chuyên biệt.

*Abstract classes*. Kế thừa là mối quan hệ giữa các lớp, không phải giữa các đối tượng, nó là một cơ chế để tổ chức và đơn giản hóa các lớp trong hệ thống và các mối quan hệ giữa chúng. Một số lớp chỉ được sử dụng cho mục đích phân loại và không bao giờ

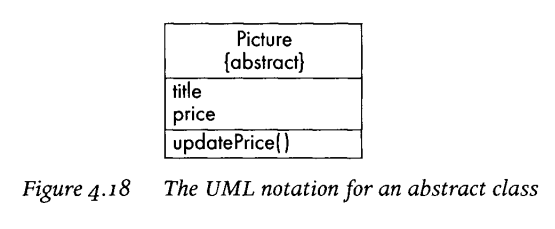


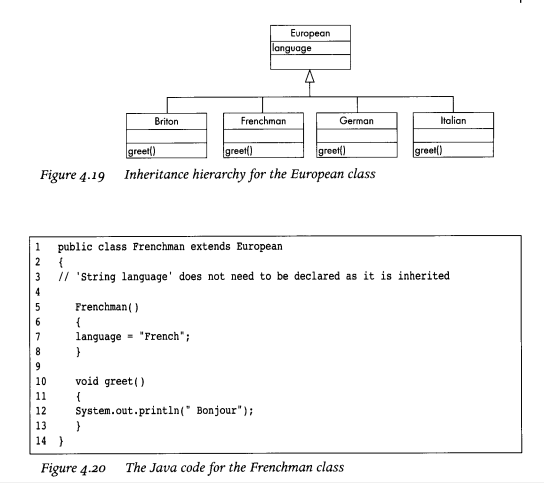
được khởi tạo bằng các từ khác, không có đối tượng nào của lớp tồn tại trong hệ thống. Trong ví dụ về bộ sưu tập nghệ thuật trong Hình 4.14, nếu chúng ta giả sử rằng một bức tranh phải là một bức tranh hoặc một bức ảnh, sẽ không bao giờ có những vật thể chỉ là hình ảnh. Tương tự, trong hệ thống phân cấp trong Hình 4.17, lớp UniversityStaff không bao giờ được khởi tạo, tất cả nhân viên của trường đại học phải là học giả, kỹ thuật viên, quản trị viên hoặc người nội địa.

Các lớp không bao giờ được khởi tạo được gọi là các lớp trừu tượng; đối diện là một lớp cụ thể hoặc ngay lập tức. Ký hiệu UML cho một lớp trừu tượng là {trừu tượng} được đặt bên dưới tên lớp - xem Hình 4.18.

Các lớp trừu tượng thường là những ứng cử viên rất hữu ích cho các thư viện lớp; tính tổng quát của chúng có nghĩa là chúng nắm bắt được bản chất của loại lớp mà không có bất kỳ hệ thống cụ thể nào được thiết kế.

*Polymorphism*. Để thảo luận về đa hình, trước tiên chúng ta phải hiểu sự khác biệt giữa các hoạt động và phương pháp. Hai thuật ngữ thường được sử dụng như thể chúng có thể hoán đổi cho nhau, nhưng điều này không hoàn toàn đúng. Chúng tôi đã mô tả một hoạt động là một quá trình mà một đối tượng có thể thực hiện, một phần của hành vi của nó. Nói một cách chính xác các hoạt động từ đề cập đến giao diện của quá trình; hoạt động phải được gọi (bằng cách truyền tin nhắn) khi mô đun máy khách muốn quá trình được thực thi. Mã thực hiện quy trình được gọi là phương thức.



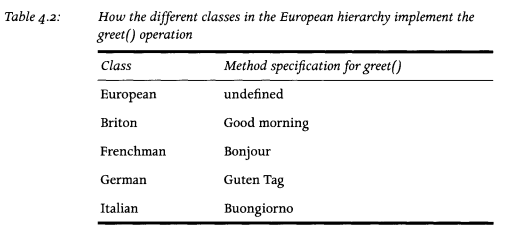


Hình 4.19 cho thấy một hệ thống phân cấp thừa kế đơn giản cho một lớp châu Âu và Hình 4.20 chứa mã Java cho người Pháp, một trong những lớp con trong hệ thống phân cấp.

Trong Hình 4.20, dòng 10 là phần hoạt động của quá trình hello(). Dòng 12 là phương thức.

Lưu ý trong hệ thống phân cấp thừa kế trong Hình 4-19, mặc dù nó được kế thừa, hoạt động hello() xuất hiện trong tất cả các lớp trong hệ thống phân cấp. Điều này không phải vì chúng ta đã quên rằng các tính năng được kế thừa có thể được bỏ qua khỏi sơ đồ, đó là vì hello() được xác định lại (nghĩa là được thực hiện bằng một phương thức khác) trong tất cả các lớp con.

Tầm quan trọng của sự khác biệt giữa hoạt động và phương thức là, trong một hệ thống phân cấp thừa kế, một thao tác có thể được thực hiện bằng nhiều phương thức. Khi một lớp con kế thừa một hoạt động, miễn là nó không thay đổi giao diện, nó có thể thay đổi phương thức cho phù hợp với chuyên môn của chính nó. Thay đổi một phương thức được gọi là vượt quá và một hoạt động đó là được thực hiện bởi một số phương pháp khác nhau được gọi là đa hình.

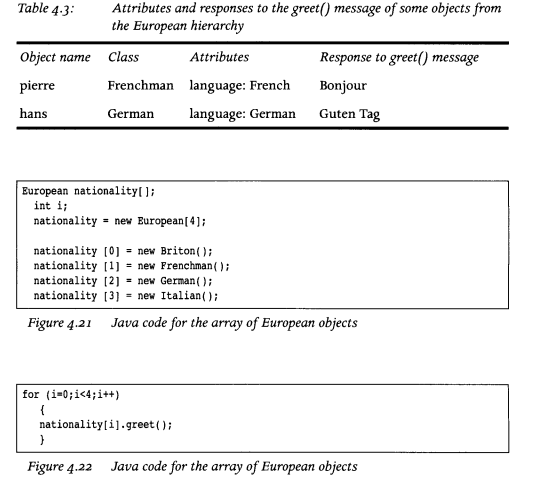


Các hoạt động đa hình được sử dụng trong bối cảnh của hệ thống phân cấp thừa kế trong đó cùng một hoạt động có thể được thực hiện khác nhau trong mỗi lớp con. Một tin nhắn sẽ tạo ra một phản hồi khác nhau tùy thuộc vào lớp của đối tượng mà nó được gửi. Ví dụ, trong hệ thống phân cấp kế thừa được hiển thị trong Hình 4.19, lời chào hoạt động () xuất hiện trong tất cả các lớp trong cấu trúc phân cấp. Phương thức cho hello () không được xác định trong lớp Châu Âu. 6 Mỗi lớp khác trong hệ thống phân cấp có một phương thức khác nhau để chào () (xem Bảng 4.2).

Để minh họa tính đa hình trong hành động, chúng ta hãy tạo ra một đối tượng được gọi là pierre của lớp người Pháp với tiếng Pháp là ngôn ngữ của mình. Nếu chúng tôi gửi cho anh ấy tin nhắn pierre.greet () anh ấy sẽ trả lời bằng cách nói 'Bonjour', xem Bảng 4.3 cũng cho thấy phản hồi của những người thuê, a: tiếng Đức. Chúng tôi đã đặt cho bạn một bài tập để tìm ra câu trả lời của george, a; Briton và antonio, an: Italian (xem Bài tập 4.9)

Để hiểu thêm một chút về cách thức đa hình hoạt động, chúng tôi tạo ra một mảng gồm bốn đối tượng châu Âu, được đặt tên quốc tịch [4]. Chúng ta có thể cư trú điều này với các đối tượng của tất cả các lớp khả thi trong hệ thống phân cấp châu Âu như trong Hình 4.21.

Chúng ta có thể lặp qua mảng gửi thông điệp hello () đến từng đối tượng lần lượt (xem Hình 4.22), mỗi đối tượng sẽ trả lời theo cách thực hiện phương thức của lớp và đầu ra sẽ là: 'Good morning', 'Bonjour', 'Guten Tag', 'Buongiorno'.



Các phần của mã này minh họa ba điểm:

* Một đối tượng lớp con luôn có thể được thay thế cho một đối tượng của lớp bên trên nó trong hệ thống phân cấp hoặc thực sự cho một đối tượng của bất kỳ lớp tổ tiên nào. Mảng được khai báo là kiểu châu Âu, nhưng chúng tôi đã đưa vào đó các đối tượng của các lớp con, Briton, người Pháp, v.v. Điều này được gọi là khả năng thay thế. Nó không hoạt động theo cách khác, chúng tôi không thể khai báo một loạt kiểu người Pháp và đưa nó vào các đối tượng từ siêu lớp của nó, châu Âu (ngay cả khi châu Âu là có thể thực hiện được). Điều này là do các đối tượng của các lớp con thường có thêm các thuộc tính và hoạt động mà các đối tượng siêu lớp không biết đến.
* Thật dễ dàng để mở rộng một hệ thống phân cấp thừa kế để phù hợp với những thay đổi trong yêu cầu của người dùng. Một lớp mới có thể được thêm vào hệ thống phân cấp mà không cần thay đổi nhiều về mã hoặc mô hình lớp. Để thuyết phục bản thân bạn cho rằng đây là trường hợp, hãy thử thực hiện điều này như một bài tập - xem Bài tập 4.14 dựa trên hệ thống phân cấp Robot được mô tả trong phần Điểm kỹ thuật của chương này.
* Chương trình không biết trước các lớp nào có liên quan đến việc xử lý mảng; theo như nó liên quan, nó đang đối phó với một loạt các đối tượng châu Âu. Khi chương trình hoạt động theo cách của nó thông qua mảng, nó chỉ biết phương thức nào sẽ được thực thi khi đến đối tượng tiếp theo và tìm ra nó thuộc lớp nào. This is known as late or dynamic binding. (Điều này được gọi là ràng buộc muộn hoặc năng động.)

Bạn sẽ tìm thấy một ví dụ phức tạp hơn về đa hình, hệ thống phân cấp Robot, trong phần Điểm kỹ thuật của chương này.

**Technical points (Điểm kỹ thuật)**

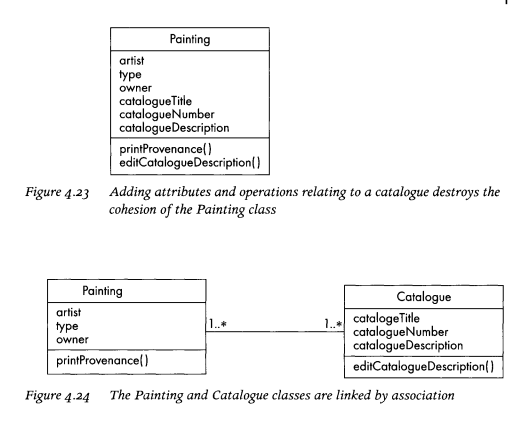
*What makes a good class?(Điều gì tạo nên một class chuẩn)*

Biết những gì làm cho một lớp học tốt đi kèm với kinh nghiệm, nhưng chúng tôi có thể cung cấp một vài gợi ý.

Problem domain (Miền vấn đề) . Trong quá trình phân tích, các lớp nên tương ứng với những thứ trong thế giới thực của vấn đề - Xe đạp, Khách hàng, Thuê và Thanh toán đều tương ứng với những thứ trong miền vấn đề Bánh xe mà người dùng sẽ biết và hiểu.

Functionality (Chức năng). Một lớp (ít nhất là trong quá trình phân tích) thường có cả thuộc tính và hành vi. Hãy nghi ngờ về một lớp dường như chỉ là một hàm lớn và không có thuộc tính; nó có thể chỉ là một hoạt động trên một số lớp khác. Tương tự, một lớp chỉ có các thuộc tính và tập cơ bản và có 7 thao tác nghe có vẻ như là một lớp được thiết kế tồi. Chức năng của hệ thống được chia sẻ giữa các lớp, vì vậy chúng nên được thực hiện nhiều hơn là chỉ duy trì và hiển thị các giá trị thuộc tính của chúng.

Cohesion (Sự gắn kết). Một trong những phẩm chất của một cấu trúc phần mềm tốt, được liệt kê ở đầu chương này, là sự gắn kết. Một lớp được gắn kết nếu nó chỉ liên quan đến một điều, nếu tất cả các thuộc tính và hoạt động của nó liên quan đến cùng một chủ đề. Ví dụ, Hình 4.23 cho thấy một phiên bản của bộ sưu tập nghệ thuật Vẽ tranh, một trong những lớp từ hệ thống phân cấp mà chúng ta đã đề cập trước đó trong chương (chúng tôi đã bỏ qua các lớp còn lại để đơn giản).

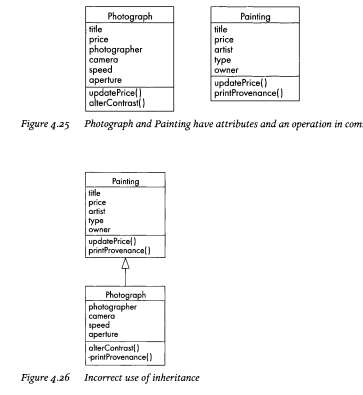


Trong phiên bản của lớp Vẽ trong Hình 4.23, chúng tôi đã thêm các thuộc tính catalogueTitle, catalogueNumber và catalogueDescription, và hoạt động editCatalogueDescrip(). Điều này làm cho lớp Tranh không cân bằng; nó không còn gắn kết bởi vì ba thuộc tính và hoạt động này thuộc về một lớp khác, một thuộc tính liên quan đến danh mục. Hai lớp riêng biệt có thể được liên kết bởi sự liên kết như trong Hình 4.24.

---------------------------------------------------------------------------------------------

Substitutability (Khả năng thay thế). Trong một hệ thống phân cấp thừa kế, các đối tượng của các lớp con cháu phải luôn luôn có thể thay thế cho các đối tượng phía trên chúng trong hệ thống phân cấp.(giông phần giải thích đa hình phía trên). Đây là những gì chúng ta đã thấy xảy ra trong ví dụ ở châu Âu mà chúng ta đã sử dụng để minh họa cho đa hình. Chúng tôi đã khai báo một mảng các đối tượng châu Âu và đưa vào đó các đối tượng của các lớp con châu Âu (xem Hình 4.21). Tất cả các đối tượng lớp con đều có thể được đối xử như thể chúng là các đối tượng châu Âu; bất kỳ thông điệp nào có thể được hiểu bởi một đối tượng châu Âu có thể được hiểu bởi con cháu của nó. Để điều này xảy ra, chúng ta cần xây dựng hệ thống phân cấp thừa kế một cách cẩn thận.

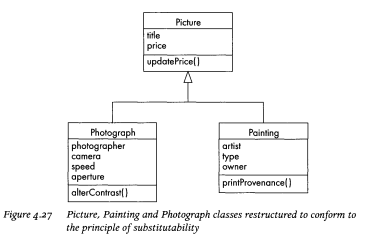
Như một ví dụ về cách không xây dựng cây thừa kế, chúng ta sẽ xem lại ví dụ về bộ sưu tập nghệ thuật (xem Hình 4.14). Chúng tôi đã bắt đầu với hai lớp trong Hình 4-13, được sao chép ở đây trong Hình 4.25.



Chúng ta có thể đã cố gắng tạo ra Ảnh kế thừa trực tiếp từ Tranh, như trong Hình 4.26, nhưng trong trường hợp này, chúng ta sẽ tạo ra một lớp con mà các đối tượng không thể thay thế cho các đối tượng của siêu lớp của nó.

Trong Ảnh, thao tác printProvenance() đã bị vô hiệu hóa một cách hiệu quả bằng cách được đặt ở chế độ riêng tư; một đối tượng khách hàng có thể gửi thông điệp printProvenance() đến các đối tượng Tranh nhưng không gửi đến các đối tượng Chụp ảnh. Hình ảnh cũng thừa hưởng các thuộc tính nghệ sĩ, loại và chủ sở hữu mà nó không sử dụng.

Một cách tốt hơn để giới thiệu sự kế thừa trong tình huống này được thể hiện trong Hình 4.27 (lặp lại từ Hình 4.14). Chúng tôi đã giới thiệu một lớp mới, Hình ảnh, trong đó chúng tôi có thể đặt các tính năng phổ biến. Ảnh và Tranh giữ lại những nét đặc sắc của chúng và chia sẻ những hình ảnh.



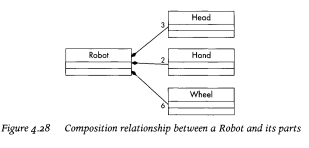
Composition (Thành phần)

UML có một hình thức tổng hợp mạnh hơn được gọi là thành phần (đôi khi nó còn được gọi là tổng hợp). Thành phần giống như tập hợp ở chỗ nó mô hình mối quan hệ toàn bộ giữa các đối tượng, nhưng không giống như tập hợp ở chỗ nó có ý nghĩa rất chính xác. Trong mối quan hệ thành phần:

* Toàn bộ đối tượng có quyền sở hữu độc quyền các bộ phận của nó, tức là đối tượng bộ phận chỉ có thể tham gia vào một tập hợp
* Do đó, các phần đối tượng không có sự tồn tại riêng biệt với toàn bộ
* Các bộ phận sống và chết với toàn bộ, tức là:
  + toàn bộ tạo ra các bộ phận của nó và
  + khi toàn bộ bị xóa, các phần của nó sẽ bị xóa (một tầng xóa bỏ).

Các đối tượng một phần được ẩn (chứa) trong toàn bộ trong đó chúng không thể nhìn thấy đối với phần còn lại của chương trình. Trong cùng một cách mà các hoạt động hình thành giao diện của một đối tượng được gói gọn và ẩn đối tượng ~ dữ liệu, giao diện của toàn bộ đối tượng sẽ đóng gói và ẩn các đối tượng bộ phận của nó. Phần còn lại của chương trình chỉ có thể giao tiếp với toàn bộ; bất kỳ giao tiếp với các bộ phận được thực hiện bởi toàn bộ đối tượng. Ký hiệu cho một mối quan hệ thành phần cũng giống như đối với tập hợp, nhưng với một viên kim cương đen thay vì màu trắng. Điều này được thể hiện trong Hình 4.28, minh họa mối quan hệ thành phần giữa Robot và các bộ phận của nó.

Nếu chúng ta muốn bắt Robot lấy một chiếc cốc, chúng ta sẽ gửi tin nhắn đến đối tượng Robot chứ không phải đến tay của nó.

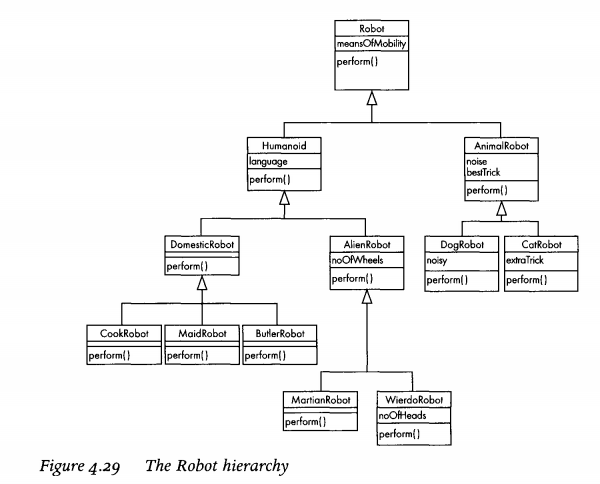


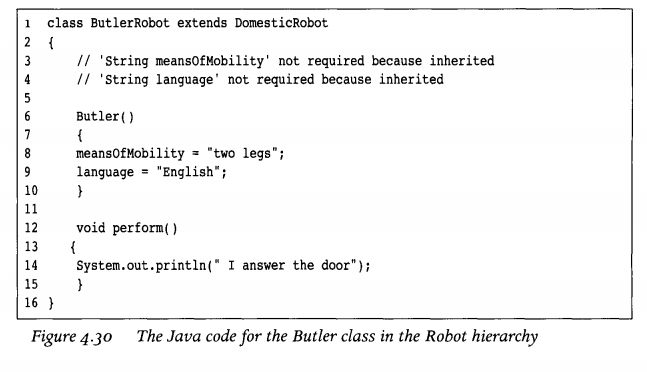
Một ví dụ phức tạp hơn về đa hình

Hình 4.29 cho thấy một hệ thống phân cấp kế thừa của lớp Robot và Hình 4.30 cho thấy mã Java cho ButlerRobot, một trong những lớp con trong hệ thống phân cấp.

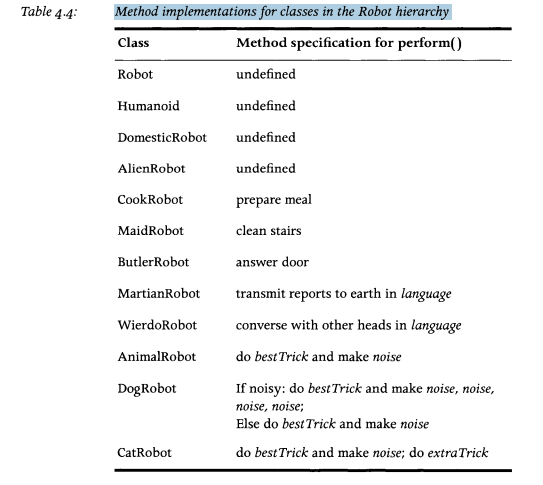
Trong mã này, dòng 12 là phần hoạt động của quá trình perform (). Dòng 14 là phương thức.

Trong hệ thống phân cấp kế thừa được hiển thị trong Hình 4.29, hoạt động perform () xuất hiện trong tất cả các lớp trong cấu trúc phân cấp. Phương thức perform () không được xác định trong các lớp Robot, Humanoid, InternalRobot và AlienRobot. Mỗi lớp khác trong hệ thống phân cấp có một phương thức perform () (xem Bảng 4-4).





Phương thức triển khai cho các lớp trong hệ thống phân cấp Robot



Để minh họa tính đa hình trong hành động, chúng ta sẽ tạo ra một đối tượng được gọi là jeeves của lớp ButlerRobot với hai chân là phương tiện của mìnhOfMobility và tiếng Anh là ngôn ngữ của mình. Nếu chúng tôi gửi cho anh ấy tin nhắn jeeves.perform () anh ấy sẽ trả lời bằng cách trả lời cửa, xem Bảng 4.5. Các câu trả lời của mart, a: MartianRobot và pat, a: DogRobot, cũng được hiển thị trong Bảng 4.5, chúng tôi đã thiết lập cho bạn một bài tập để tìm ra phản ứng của các đối tượng của các lớp còn lại, xem Bài tập 4.11.

Để minh họa một lần nữa cách đa hình hoạt động trong mã, chúng ta hãy tạo một mảng gồm bảy đối tượng Robot, được đặt tên là automaton. Chúng ta có thể cư trú điều này với các đối tượng của tất cả các lớp khả thi trong hệ thống phân cấp Robot như trong Hình 4.31.

Chúng ta có thể lặp qua mảng gửi thông báo biểu diễn () đến từng đối tượng lần lượt (xem Hình 4.32); mỗi người sẽ trả lời theo phương thức thực hiện của lớp họ.

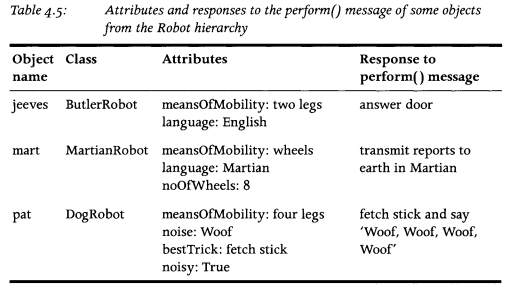
*Reuse (tái sử dụng)*

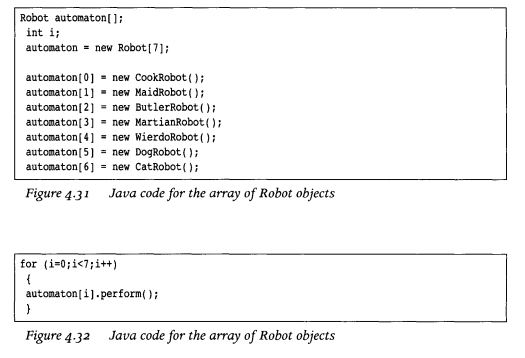
Các nhà phát triển phần mềm, không giống như các đồng nghiệp của họ trong hầu hết các hình thức phát triển phần cứng và thực sự là hầu hết các ngành công nghiệp khác, chưa bao giờ thực sự sử dụng để tái sử dụng thành phần. Những lý do thường xuyên nhất được ghi nhận cho điều này dường như là như sau.

* Các lập trình viên dường như không thích sử dụng mã được viết bởi người khác - hội chứng 'Không được phát minh ở đây' (the 'Not Invented Here' syndrome)
* Các vấn đề được gây ra nếu các thành phần có thể phù hợp để tái sử dụng được viết bằng ngôn ngữ lập trình khác hoặc cho một nền tảng phần cứng khác với nền tảng được sử dụng bởi hệ thống mới.
* Có nhiều thư viện các thành phần phần mềm cả thương mại và nội bộ. Tuy nhiên, các thành phần thư viện thường quá cụ thể hoặc quá chung chung. Trong cả hai trường hợp, họ không hoàn toàn làm những gì được yêu cầu. Các thư viện phần mềm không phải lúc nào cũng được ghi chép tốt và thật khó để tìm ra chính xác những gì một mô-đun làm.

Cách tiếp cận hướng đối tượng cung cấp một số trợ giúp.

* Thư viện của các lớp hiện tồn tại và được sử dụng rộng rãi.
* Cơ chế kế thừa cho phép các lập trình viên điều chỉnh các lớp thư viện để đáp ứng các yêu cầu của một hệ thống mới.
* Thường thì một nhóm các lớp liên quan hữu ích hơn một lớp duy nhất như là một thành phần để tái sử dụng. Các lớp liên quan theo thành phần tạo thành một đơn vị phần mềm mạch lạc với giao diện chung được xác định rõ ràng.
* Các lớp được thiết kế tốt và các thành phần của các lớp được gắn kết và dễ hiểu; họ có một nhận dạng rõ ràng và mục đích dễ dàng. Các lập trình viên tìm kiếm một thư viện có thể dễ dàng xác định các thành phần phần mềm để đáp ứng nhu cầu của họ.



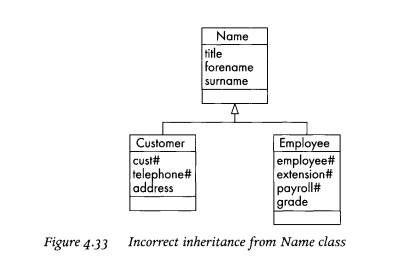


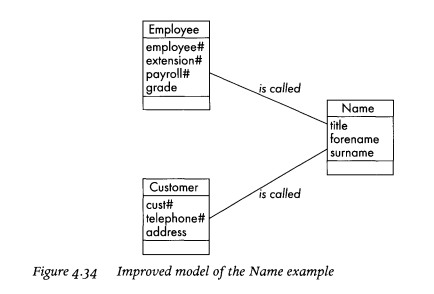
* Các lớp và các thành phần của các lớp gói gọn các chi tiết bên trong của chúng sao cho tất cả những gì một thành phần khách hàng cần biết là giao diện. Điều này giúp giải quyết vấn đề sử dụng các thành phần ban đầu được viết cho các hệ thống khác nhau bằng các ngôn ngữ khác nhau.

Common problems (vấn đề chung)

**1)** Khi tôi nhìn vào các đối tượng có vẻ giống nhau nhưng có các giá trị khá khác nhau, làm thế nào để tôi biết liệu chúng nên được mô hình hóa thành các lớp khác nhau hoặc như các đối tượng khác nhau của cùng một lớp? Ví dụ: nếu tôi có cha, mẹ và các con thứ bảy, tôi nên tạo một lớp mới cho mỗi người hay tất cả chúng đều là đối tượng của một lớp chung như Người? Nó phụ thuộc vào những gì hệ thống của bạn cần ghi lại về các đối tượng này và những gì nó muốn chúng làm. Trong một hệ thống thư viện, tất cả chúng chỉ là đối tượng của một lớp như FamilyMember. Trong một hệ thống trường học, tất cả trẻ em sẽ là đối tượng của lớp Học sinh và phụ huynh có thể chỉ là những thuộc tính. Tuy nhiên, nếu bạn đang viết một hệ thống mô phỏng hành vi của các gia đình, bạn có thể mô hình hóa chúng thành các lớp riêng biệt. Để các thực thể bảo đảm được mô hình hóa thành các lớp khác nhau, chúng phải có các tính năng phân biệt, tức là hành vi hoặc thuộc tính riêng biệt, không chỉ các giá trị khác nhau cho các thuộc tính của chúng.

**2)** Tôi có hai lớp, Khách hàng và Nhân viên. Cả Khách hàng và Nhân viên cần ghi lại tiêu đề, tên và họ; Tôi có thể biến chúng thành các lớp con của **Tên** lớp để chúng kế thừa chi tiết tên, như trong Hình 4.33- Đây sẽ là cách sử dụng kế thừa hoàn toàn không chính xác. Khách hàng và nhân viên không nên được mô hình hóa thành các chuyên môn của **Tên** vì không có mối quan hệ giữa họ; không phải khách hàng hay nhân viên là những loại tên. Một cách tốt hơn để mô hình hóa điều này sẽ là sử dụng mối quan hệ liên kết như trong Hình 4.34





**3)** Khi tôi đang xử lý một hệ thống phân cấp thừa kế, đôi khi tôi không thể biết liệu một lớp nên là một lớp con của một lớp khác hay nếu tôi đang xem xét hai thành viên của cùng một lớp. Ví dụ, ngựa Shetland có phải là một lớp con của Ngựa không?

Để một cái gì đó đủ điều kiện là một lớp con, nó phải có ít nhất một thuộc tính hoặc hoạt động bổ sung hoặc vượt quá một hoạt động được kế thừa.

**4)** Tổ chức các lớp thành một hệ thống phân cấp thừa kế có vẻ như rất nhiều công việc phụ; nó luôn luôn đáng làm?

Tổ chức các lớp thành một hệ thống phân cấp thừa kế liên quan đến nỗ lực thêm. Thật đáng làm nếu nó đơn giản hóa mô hình của bạn và làm rõ suy nghĩ của bạn. Cũng đáng giá nếu có vẻ như bạn sẽ cần phải giới thiệu các lớp mới trong tương lai và một hệ thống phân cấp thừa kế sẽ đơn giản hóa quy trình. Một yếu tố khác là liệu bạn có nghĩ đến việc sử dụng lại bất kỳ lớp học nào không. Chúng càng chung chung, chúng càng dễ sử dụng lại.

**5**) Các lớp con có luôn kế thừa tất cả các tính năng của lớp cha không?

Vâng, cho tất cả các mục đích thực tế. Một số ngôn ngữ lập trình hướng đối tượng cho phép các tính năng được kế thừa bị loại bỏ, nhưng điều này thường được coi là thực tiễn xấu.

**6)** Làm thế nào tôi có thể nhận ra một lớp trừu tượng?

Về mặt kỹ thuật, một lớp trừu tượng là một lớp không có phương thức được định nghĩa cho một hoặc nhiều hoạt động của nó; điều này có nghĩa là không thể có trường hợp nào của lớp trong một hệ thống.

Chapter summary (Tóm tắt chương)

Phần mềm được sản xuất bằng cách tiếp cận phát triển có cấu trúc vẫn còn nhiều vấn đề đặc biệt liên quan đến bảo trì, tái sử dụng và thử nghiệm. Những thất bại này được coi là do thiếu một cấu trúc phần mềm phù hợp. Cách tiếp cận hướng đối tượng dựa trên đối tượng, một cấu trúc phần mềm sẽ khắc phục các vấn đề mà phần mềm có cấu trúc phải chịu.

Các đối tượng dựa trên dữ liệu trong một hệ thống, nhưng cũng có thể cung cấp các chức năng cần thiết. Mỗi đối tượng thuộc về một lớp, xác định các thuộc tính, hành vi và các mối quan hệ của nó. Một lớp học tốt cần thể hiện những phẩm chất được liệt kê trong chương này. Nó nên tạo ra các đối tượng tự chủ, gắn kết và dễ hiểu. Chúng tôi đã thảo luận về cách các đối tượng đóng gói dữ liệu, cùng với các thao tác để thao tác với nó, trong một cấu trúc duy nhất và ẩn dữ liệu đằng sau một giao diện hoạt động công khai. Chúng tôi cũng đã thấy rằng các lớp trừu tượng có thể được sử dụng để xây dựng một số bằng chứng trong tương lai vào hệ thống.

Có ba cách chính trong đó các lớp có thể được liên kết: liên kết, tổng hợp và kế thừa. Kế thừa là một kỹ thuật mạnh mẽ cho phép chúng ta tạo ra các lớp mới bằng cách chuyên biệt hóa các lớp hiện có; do đó nó là một công cụ quan trọng trong việc tái sử dụng phần mềm. Kế thừa cũng cho phép đa hình, trong đó một hoạt động có thể được thực hiện theo những cách khác nhau bởi các lớp khác nhau.

Bibliography(Thư mục)

Câu hỏi:

**d) Sự khác biệt giữa một lớp và một đối tượng là gì?**

Một lớp là bao gồm nhiều đối tượng.

Đối tượng là chỉ 1 đối tượng cụ thể.

VD: Sinh viên là k có nói sinh viên nào cả. đối tượng ở đây cụ thể là 1 sinh viên có tên gì gì đó

**e) Sự khác biệt trong ký hiệu sơ đồ UML cho một lớp và một đối tượng là gì?**

Đối tượng được biểu diễn dạng một hình chữ nhật có hai phần Phần:

+ trên cùng là tên của đối tượng và phần thứ hai được sử dụng cho thuộc tính

+ Tên đối tượng luôn được gạch chân

**Class có 3 phần**: phần trên là : tên class, phần giữa là : thuộc tính, cuối cùng là : phương thức.

**m) Tại sao bạn có thể quyết định mô hình hóa một lớp là một lớp con của một lớp khác?**

- Nếu 1 lớp con có ít nhất 1 thuộc tính khác với lớp cha thì mới kế thừa còn nếu lớp con có các thuộc tính giống với lớp cha thì không cần kế thừa mà sử dụng luôn lớp cha

- Phải có mỗi quan hệ giữa các lớp với nhau.

**Tại sao chúng ta sử dụng lớp trừu tượng?**

* Tính trừu tượng cho phép các lập trình viên loại bỏ tính chất phức tạp của đối tượng bằng cách chỉ đưa ra các thuộc tính và phương thức cần thiết của đối tượng trong lập trình, cải thiện khả năng bảo trì của hệ thống.
* Tính trừu tượng giúp chúng ta tập trung vào những cốt lõi cần thiết của đối tượng thay vì quan tâm đến cách nó thực hiện.
* Tính trừu tượng cung cấp nhiều tính năng mở rộng khi sử dụng kết hợp với tính đa hình và kế thừa trong lập trình hướng đối tượng.

**Sự khác nhau giữa thành phần và tổng hợp**

