# Giáo trình

## Ngôn ngữ mô hình hoá thống nhất UML

# LỜI NÓI ĐẦU

Nhiệm vụ của công nghệ thông tin nói chung, công nghệ phần mềm nói riêng là nghiên cứu các mô hình, phương pháp và công cụ để tạo ra những hệ thống phần mềm chất lượng cao nhằm đáp ứng được những nhu cầu thường xuyên thay đổi, ngày một phức tạp của thực tế. Nhiều hệ thống phần mềm đã được xây dựng theo các cách tiếp cận truyền thống tỏ ra lạc hậu, không đáp ứng được các yêu cầu của người sử dụng. *Cách tiếp cận hướng đối tượng* giúp chúng ta có được những công cụ, phương pháp mới, phù hợp để giải quyết những vấn đề nêu trên. Cách tiếp cận này rất phù hợp với cách quan sát và quan niệm của chúng ta về thế giới xung quanh và tạo ra những công cụ mới, hữu hiệu để phát triển các hệ thống có tính mở, dễ thay đổi theo yêu cầu của người sử dụng, đáp ứng được các tiêu chuẩn phần mềm chất lượng cao theo yêu cầu của nền công nghệ thông tin hiện đại.

Giáo trình này trình bày cách sử dụng *ngôn ngữ mô hình hoá thống nhất* UML (*Unified Modeling Language*) để phân tích và thiết kế hệ thống theo cách tiếp cận hướng đối tượng. Cách tiếp cận hướng đối tượng đặt trọng tâm vào việc xây dựng lý thuyết cho các hệ thống tổng quát như là mô hình khái niệm cơ sở. Hệ thống được xem như là tập các thực thể tác động qua lại và trao đổi với nhau bằng các thông điệp để thực hiện những nhiệm vụ đặt ra. Các khái niệm mới của mô hình hệ thống hướng đối tượng và các bước thực hiện phân tích, thiết kế hướng đối tượng được mô tả, hướng dẫn thực hiện thông qua ngôn ngữ chuẩn UML cùng phần mềm công cụ hỗ trợ mô hình hoá *Rational Rose.*

Giáo trình được biên soạn theo yêu cầu giảng dạy, học tập môn học "*Phân tích, thiết kế hệ thống*" của ngành Công nghệ thông tin và dựa vào kinh nghiệm giảng dạy môn học này qua nhiều năm của các tác giả trong các khoá đào tạo cao học, đại học tại các Đại học Khoa học Huế, Đại học Quốc gia Hà Nội, Đại học Bách khoa Hà Nội, Đại học Đà Nẵng, Đại học Thái Nguyên, v.v.

Giáo trình được trình bày trong tám chương. Chương mở đầu giới thiệu những khái niệm cơ sở trong mô hình hoá hệ thống và hai cách tiếp cận chính để phát triển các hệ thống phần mềm hiện nay là hướng thủ tục (chức năng) và hướng đối tượng. Chương II giới thiệu ngôn ngữ mô hình hoá thống nhất UML và vai trò của nó trong quá trình phát triển phần mềm. Vấn đề phân tích các yêu cầu của hệ thống và cách xây dựng biểu đồ ca sử dụng được nêu ở chương III. Chương IV trình bày những khái niệm cơ bản về các lớp đối tượng và các mối quan hệ của chúng trong không gian bài toán. Biểu đồ lớp cho phép biểu diễn tất cả những khái niệm đó một cách trực quan và thông qua mô hình khái niệm là biểu đồ lớp, chúng ta hiểu rõ hơn về hệ thống cần phát triển. Những biểu đồ tương tác thể hiện các hành vi và ứng xử của hệ thống được giới thiệu ở chương V. Dựa vào những kết quả phân tích ở các chương trước, hai chương tiếp theo nêu cách thực hiện để thiết kế các biểu đồ cộng tác cho từng nhiệm vụ, từng ca sử dụng của hệ thống và từ đó có được những thiết kế lớp, biểu đồ lớp chi tiết thực

hiện chính xác các nhiệm vụ được giao. Vấn đề quan trọng là lựa chọn kiến trúc cho hệ thống và khả năng ánh xạ những kết quả thiết kế sang mã chương trình trong một ngôn ngữ lập trình hướng đối tượng như C++ được đề cập ở chương VII. Chương cuối trình bày một số vấn đề chính cần lưu ý khi thiết kế một CSDL HĐT, trong đó chủ yếu giới thiệu về việc ứng dụng ObjectStore trong cài đặt ứng dụng CSDL. Bài toán "*Hệ thống quản lý bán hàng*" được chọn làm ví dụ minh hoạ để phân tích, thiết kế hệ thống phần mềm theo cách tiếp cận hướng đối tượng xuyên suốt cả giáo trình.

Tác giả xin chân thành cám ơn các bạn đồng nghiệp trong Viện CNTT, các bạn trong Khoa CNTT, Đại học Hue, các bạn trong Khoa Công nghệ, Đại học Quốc gia Hà Nội về những đóng góp quí báu, hỗ trợ thiết thực và động viên chân thành để hoàn thành cuốn giáo trình này.

Mặc dù đã rất cố gắng nhưng giáo trình này chắc không tránh khỏi những sai sót. Chúng tôi rất mong nhận được các ý kiến góp ý của các thầy cô, những nhận xét của sinh viên và các bạn đọc để hiệu chỉnh thành cuốn sách hoàn thiện.

*Hà Nội 2004*

***Các tác giả***

# CHƯƠNG I

# PHẦN MỀM VÀ MÔ HÌNH HOÁ HỆ THỐNG

Chương I trình bày các vấn đề cơ sở về:

- ✓ Các khái niệm và đặc trưng cơ bản của hệ thống phần mềm,
- ✓ Vai trò của mô hình hoá hệ thống,
- ✓ Các phương pháp phân tích và thiết kế hệ thống.

## 1.1 Giới thiệu về hệ thống phần mềm

*Hệ thống phần mềm* hay gọi tắt là *hệ thống,* là tổ hợp các phần cứng, phần mềm có quan hệ qua lại với nhau, cùng hoạt động hướng tới mục tiêu chung thông qua việc nhận các dữ liệu đầu vào (*Input*) và sản sinh ra những kết quả đầu ra (*Output*) thường là ở các dạng thông tin khác nhau nhờ một quá trình xử lý, biến đổi có tổ chức. Một cách hình thức hơn chúng ta có thể *định nghĩa phần mềm* [3] bao gồm các thành phần cơ bản như sau:

1. Hệ thống các lệnh (chương trình) khi thực hiện thì tạo ra được các hoạt động và cho các kết quả theo yêu cầu,
2. Các cấu trúc dữ liệu làm cho chương trình thực hiện được các thao tác, xử lý và cho ra các thông tin cần thiết,
3. Các tài liệu mô tả thao tác và cách sử dụng chương trình.

Có nhiều định nghĩa khác nhau về các hệ thống thông tin ([3], [4], [6]). Để hiểu hơn về bản chất của hệ thống thì tốt nhất là phải xem xét các đặc trưng cơ bản của chúng. Hệ thống thông tin cũng giống như các hệ thống khác đều có những đặc trưng cơ bản như sau:

1. *Tính nhất thể hoá được thể hiện thông qua*:
   - Phạm vi và qui mô của hệ thống được xác định như một thể thống nhất và không thay đổi trong những điều kiện nhất định.
   - Tạo ra những đặc tính chung để thực hiện được các nhiệm vụ hay nhằm đạt được các mục tiêu chung mà từng bộ phận riêng lẻ không thể thực hiện được.
2. *Tính tổ chức có thứ bậc:*
   - Mọi hệ thống luôn là hệ thống con của một hệ thống lớn hơn trong môi trường nào đó và chính nó lại bao gồm các hệ thống (các thành phần) nhỏ hơn.

- Giữa các thành phần của một hệ thống có sự sắp xếp theo quan hệ thứ bậc hay một trình tự nhất định.
- *Tính có cấu trúc*: Chính cấu trúc của hệ thống quyết định cơ chế vận hành của hệ thống và mục tiêu mà nó cần đạt được. Cấu trúc của hệ thống được thể hiện bởi:
  - ✓ Các phần tử được sắp xếp theo trật tự và cấu thành hệ thống.
  - ✓ Mối quan hệ giữa các thành phần liên quan chủ yếu đến loại hình, số lượng, chiều, cường độ, v.v.

Những hệ thống có cấu trúc chặt thường được gọi là *hệ thống có cấu trúc*. Cấu trúc của hệ thống là quan trọng, nó có thể quyết định tính chất cơ bản của hệ thống. Ví dụ: *Kim cương* và *than đá* đều được cấu tạo từ các phân tử *các-bon*, nhưng khác nhau về cấu trúc nên: kim cương vô cùng rắn chắc, còn tham đá thì không có tính chất đó.

Sự thay đổi cấu trúc có thể tạo ra những đặc tính mới (sức trồi mới, hay còn gọi là những đột biến) của hệ thống và khi vượt quá một ngưỡng nào đó thì có thể dẫn tới việc phá vỡ hệ thống cũ. *Ví dụ*: công nghệ biến đổi gen chủ yếu là làm thay đổi cấu trúc của các tế bào sinh học.

*3. Tính biến đổi theo thời gian và không gian*

- Các hệ thống phải luôn thay đổi cho phù hợp với điều kiện thực tế theo thời gian và không gian, nghĩa là muốn tồn tại và phát triển thì phải biến đổi cho phù hợp với môi trường xung quanh theo qui luật *tiến hoá của tự nhiên (Darwin).* Sự khác nhau chủ yếu là tốc độ và khả năng nhận biết được về sự thay đổi đó.
- Mọi sự thay đổi luôn có mối liên hệ ngược (*feedback*) trong hệ thống và chịu sự tác động của qui luật "*nhân - quả*".

Hệ thống được đánh giá theo nhiều tiêu chí khác nhau ([3], [6], [12]) và chưa có một hệ thống tiêu chí chuẩn để đánh giá cho các *sản phẩm phần mềm*. Ở đây chúng ta chỉ quan tâm đến một số tính chất quan trọng nhất hiện nay của các sản phẩm phần mềm. Một sản phẩm của công nghệ phần mềm hiện nay, ngoài những tính chất chung của các hệ thống nêu trên thì phải có các tính chất sau:

- *Tính tiện dụng*: sản phẩm phải dễ sử dụng và tiện lợi cho người dùng, hỗ trợ để thực hiện các công việc tốt hơn. Muốn đạt được mục đích này thì phần mềm phải có giao diện thân thiện, phù hợp với người sử dụng và có đầy đủ các tài liệu mô tả, có sự hỗ trợ kịp thời.
- *Khả năng bảo hành và duy trì hoạt động:* Hệ thống phải có khả năng cập nhật, dễ thay đổi, có khả năng mở rộng để thực hiện được những yêu cầu thay đổi của khách hàng.
- *Tính tin cậy:* Tính tin cậy của phần mềm không chỉ thể hiện ở khả năng thực hiện đúng nhiệm đã được thiết kế và cả các khả năng đảm bảo an toàn, an ninh dữ liệu. Hệ thống phải thực hiện bình thường ngay cả khi có sự kiện bất thường xảy ra.

- *Tính hiệu quả:* Phần mềm không gây ra sự lãng phí các tài nguyên như bộ nhớ, bộ xử lý, các thiết bị ngoại vi, v.v.

Hệ thống có thể được *phân loại* theo nhiều quan điểm khác nhau.

- *Theo nguyên nhân xuất hiện*: hệ thống tự nhiên, sẵn có trong tự nhiên và hệ thống nhân tạo, do con người tạo ra.
- *Theo quan hệ với môi trường*: hệ đóng, ít trao đổi với môi trường xung quanh và hệ mở, có trao đổi và có thể thích ứng với các sự kiện xung quanh.
- *Theo qui mô*: lớn, trung bình và nhỏ.
- *Theo sự thay đổi trạng thái trong không gian, thời gian*: hệ động và hệ tĩnh, v.v.

Người ta còn phân loại các *hệ thống phần mềm* theo các đặc tính chung của chúng.

1. *Hệ thống thông tin*: hệ thống lưu trữ, tìm kiếm, biến đổi và biểu diễn mọi thông tin cho người sử dụng. Khi khối lượng dữ liệu lớn, phức tạp thì hệ thống thường được tổ chức thành các hệ CSDL theo mô hình quan hệ hay hướng đối tượng.
2. *Các hệ thống kỹ thuật:* hệ thống xử lý và điều khiển các thiết bị kỹ thuật như các hệ viễn thông, các hệ thống quân sự, các quá trình công nghiệp, v.v. Đó thường là các hệ thống thời gian thực.
3. *Các hệ thống nhúng thời gian thực:* thực hiện trên những thiết bị cứng đơn giản và được nhúng vào các thiết bị khác như: *mobile phone*, hệ thống hướng dẫn lái xe ô tô, hệ thống điều khiển các dụng cụ dân dụng, v.v.
4. *Các hệ thống phân tán:* hệ thống được phân tán trên nhiều máy và dữ liệu được chuyển dễ dàng từ máy này sang máy khác.
5. *Phần mềm hệ thống:* Tạo ra các cơ sở (kiến trúc) cho các phần mềm khác sử dụng như: hệ điều hành, CSDL, giao diện phần mềm ứng dụng API (*Application Programming Interface*), v.v.
6. *Các hệ thống nghiệp vụ:* Mô tả mục đích, tài nguyên, các luật, chiến lược, sách lược hoạt động, kinh doanh và những công việc hiện thời trong các nghiệp vụ.

Khi xây dựng một hệ thống chúng ta cần xác định xem nó thuộc loại hệ thống nào và *mục tiêu chính* của chúng ta là nghiên cứu hệ thống để:

- Hiểu rõ hơn về chúng, nhất là những hệ thống lớn, phức tạp, để mô hình được chúng và từ đó xây dựng được những hệ thống phần mềm tốt.
- Có thể tác động lên hệ thống một cách có hiệu quả.
- Hoàn thiện hay phát triển những hệ thống tốt hơn nhằm đáp ứng mọi yêu cầu của khác hàng.

Để xem xét sự phát triển hệ thống tin học, có hai khía cạnh cần đề cập:

- Các phương pháp để nhận thức và diễn tả hệ thống, còn gọi là các mô hình.
- Các bước nối tiếp trong thời kỳ phát triển hệ thống, còn gọi là chu kỳ phát triển hệ thống.

## 1.2 Mô hình hoá hệ thống

Các bước phát triển hệ thống như tìm hiểu nhu cầu, phân tích và thiết kế hệ thống tuy có khác nhau về nhiệm vụ, mục tiêu, song chúng có chung đặc điểm chung: *phải đối đầu với sự phức tạp và những quá trình nhận thức, diễn tả sự phức tạp thông qua mô hình.* Nói cách khác, để điều khiển được hệ thống hay phát triển được một hệ thống đáp ứng các yêu cầu, mục đích đặt ra thì phải thực hiện được mô hình hoá hệ thống. Thông qua mô hình chúng ta sẽ giới hạn vấn đề nghiên cứu bằng cách chỉ tập trung vào một khía cạnh trong phạm vi không gian và thời gian nhất định. Đó chính là nguyên lý *chia để trị:* tấn công vào những vấn đề khó bằng cách chia nó thành dãy các vấn đề nhỏ hơn mà ta có thể giải quyết được. Như Pascal đã khẳng: "*Không thể hiểu toàn bộ mà không hiểu bộ phận và cũng không thể hiểu bộ phận mà không hiểu tổng thể*".

*Mô hình* là một dạng trừu tượng hoá hệ thống thực của bài toán mà chúng ta đang xét, được diễn đạt một cách hình thức dễ hiểu bằng văn bản, biểu đồ, đồ thị, công thức hay phương trình toán học, v.v.

*Mục đích của mô hình hoá:*

1. *Mô hình giúp ta hiểu và thực hiện được sự trừu tượng, tổng quát hoá các khái niệm cơ sở để giảm thiểu độ phức tạp của hệ thống*. Qua mô hình chúng ta biết được hệ thống gồm những gì? và chúng hoạt động như thế nào?. Jean Piaget [5] từng nói: "*Hiểu tức là mô hình hoá*". Do vậy, quá trình phát triển phần mềm chẳng qua là quá trình nhận thức và mô tả lại tả hệ thống đó. Đó cũng là quá trình thiết lập, sử dụng và biến đổi các mô hình. Vậy, có một mô hình đúng sẽ giúp ta làm sáng tỏ những vấn đề phức tạp và cho ta cái nhìn thấu đáo về vấn đề cần giải quyết.
2. *Mô hình giúp chúng ta quan sát được hệ thống như nó vốn có trong thực tế hoặc nó phải có như ta mong muốn*. Muốn hiểu và phát triển được hệ thống phần mềm theo yêu cầu thực tế thì ta phải quan sát nó theo nhiều góc nhìn khác nhau: theo chức năng sử dụng, theo các thành phần logic, theo phương diện triển khai, v.v.
3. *Mô hình cho phép ta đặc tả được cấu trúc và hành vi của hệ thống:*

   *+ Đảm bảo hệ thống đạt được mục đích đã xác định trước*. Mọi mô hình đều đơn giản hoá thế giới thực, nhưng phải đảm bảo sự đơn giản đó không loại bỏ đi những những yếu tố quan trọng.

   *+ Kiểm tra được các qui định về cú pháp, ngữ nghĩa về tính chặt chẽ và đầy đủ của mô hình, khẳng định được tính đúng đắn của thiết kế, phù hợp với yêu cầu của khách hàng.* Nghĩa là, *mô hình hoá là quá trình hoàn thiện và tiến hoá liên tục.*
4. *Mô hình hoá là nhằm tạo ra khuôn mẫu (template) và hướng dẫn cách xây dựng hệ thống; cho phép thử nghiệm, mô phỏng và thực hiện, hoàn thiện theo mô hình.*
5. *Mô hình là cơ sở để trao đổi, ghi lại những quyết định đã thực hiện trong nhóm tham gia dự án phát triển phần mềm*. Mọi quan sát, mọi sự hiểu biết

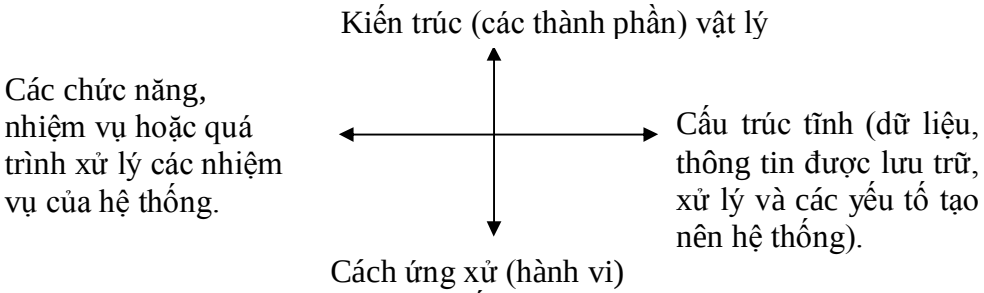
(kết quả phân tích) đều phải được ghi lại chi tiết để phục vụ cho cả quá trình phát triển hệ thống.

Để tìm hiểu một thế giới vô cùng phức tạp, mọi khoa học thực nghiệm đều phải vận dụng một nguyên lý cơ bản, đó là sự trừu tượng hoá (*Absstraction*). *Trừu tượng hoá là một nguyên lý của nhận thức, đòi hỏi phải bỏ qua những sắc thái (của chủ điểm) không liên quan tới chủ định hiện thời, để tập trung hoàn toàn vào các sắc thái chính liên quan tới chủ định đó (từ điểm Oxford).*

Nhìn chung không có mô hình nào là đầy đủ. Mỗi hệ thống thực tế có thể được tiếp cận thông qua một hay một số mô hình khác nhau. Quá trình mô hình hoá hệ thống phần mềm thường thực hiện theo hai cấp:

+ *Mô hình logic*: mô tả các thành phần và mối quan hệ của chúng để tổ chức thực hiện, về biện pháp cài đặt. Mô hình logic trả lời câu hỏi "Là gì?" và bỏ qua câu hỏi "như thế nào?",
+ *Mô hình vật lý*: xác định kiến trúc các thành phần và tổng thể của hệ thống. Trả lời câu hỏi "Như thế nào?", quan tâm tới biện pháp, công cụ, kế hoạch thực hiện.

*Tóm lại, mô hình hoá một hệ thống phải thực hiện theo cả bốn hướng:*




Hình 1-1 Các hướng mô hình hoá

Hướng của điểm xuất phát sẽ kéo theo phương pháp cần lựa chọn để phát triển phần mềm. Nếu ta bắt đầu từ bên trái, nghĩa là tập trung vào chức năng để phân tích thì chúng ta thực hiện phát triển phần mềm theo cách tiếp cận hướng chức năng. Ngược lại, nếu bắt đầu từ bên phải, nghĩa là dựa vào dữ liệu là chính thì chúng ta sử dụng phương pháp hướng đối tượng.

Có bốn yếu tố quan trọng ảnh hưởng tới hiệu quả của dự án phát triển phần mềm:

| **Nhân tố ảnh hưởng** | **Thuộc tính** |
|---|---|
| Sản phẩm phần mềm (bài toán ứng dụng) | Mức độ tin cậy, chính xác của phần mềm yêu cầu<br>Cỡ của CSDL, số lượng dữ liệu<br>Độ phức tạp của sản phẩm phần mềm |
| Máy tính (công nghệ) | Những ràng buộc về thời gian thực hiện<br>Những ràng buộc về bộ nhớ chính<br>Tần xuất thay đổi của hệ điều hành và/hoặc phần cứng<br>Môi trường phát triển chương trình |
| Con người | Khả năng của các nhà phân tích, thiết kế<br>Kinh nghiệm làm việc với những hệ tương tự<br>Khả năng của các lập trình viên<br>Kinh nghiệm làm việc với hệ điều hành và/hoặc phần cứng<br>Mức độ thông thạo ngôn ngữ lập trình được lựa chọn |

| | |
|---|---|
| Qui trình | Sử dụng phương pháp để phát triển phần mềm<br>Sử dụng công cụ phát triển phần mềm<br>Lịch biểu phát triển phần mềm |

Vấn đề rất quan trọng hiện nay trong công nghệ phần mềm là cần phải có những công cụ hỗ trợ để thực hiện mô hình hoá trực quan theo một chuẩn dễ hiểu giúp cho việc trao đổi giữa những người phát triển phần mềm hiệu quả và dễ dàng hơn. Các nhà tin học đã rất cố gắng để phát triển các công cụ thực hiện mô hình hoá trực quan. Từ những khái niệm, ký pháp quen thuộc của Booch, Ericsson, OOSE/Objectory (Jacobson), OMT (Rumbaugh) người ta đã xây dựng được một ngôn ngữ mô hình thống nhất UML được nhiều người chấp nhận và sử dụng như một ngôn ngữ chuẩn trong phân tích và thiết kế hệ thống phần mềm. Hầu hết các hãng sản xuất phần mềm lớn như: Microsoft, IBM, HP, Oracle, v.v… đều sử dụng UML như là chuẩn công nghiệp. Trong tài liệu này chúng ta sử dụng UML để phân tích, thiết kế hệ thống. Chi tiết về UML và cách sử dụng nó để phân tích và thiết kế hệ thống sẽ được trình bày chi tiết ở các phần sau.

## 1.3 Các cách tiếp cận trong phát triển phần mềm

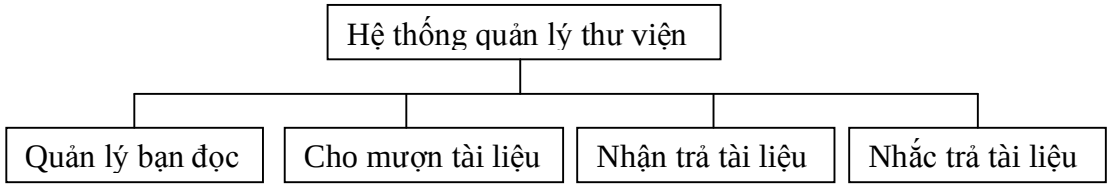
Để thực hiện một dự án phát triển phần mềm thì vấn đề quan trọng đầu tiên chắc sẽ là phải chọn cho được một cách thực hiện thích hợp dựa trên những yếu tố nêu trên. Có hai cách tiếp cận cơ bản để phát triển phần mềm: *cách tiếp hướng chức năng và cách tiếp cận hướng đối tượng*.

### 1.3.1 Cách tiếp cận hướng chức năng

Phần lớn các chương trình được viết bằng ngôn ngữ lập trình như C, hay Pascal từ trước đến nay đều được thực hiện theo cách tiếp cận hướng chức năng hay còn được gọi là cách tiếp cận hướng thủ tục. Cách tiếp cận này có những đặc trưng sau:

1. **Dựa vào chức năng, nhiệm vụ là chính.** Khi khảo sát, phân tích một hệ thống chúng ta thường *tập trung vào các nhiệm vụ* mà nó cần thực hiện. Chúng ta tập trung trước hết nghiên cứu các yêu cầu của bài toán để *xác định các chức năng chính của hệ thống*. Ví dụ khi cần xây dựng "hệ thống quản lý thư viện" thì trước hết chúng ta thường đi nghiên cứu, khảo sát trao đổi và phỏng vấn xem những người thủ thư, bạn đọc cần phải thực hiện những công việc gì để phục vụ được bạn đọc và quản lý tốt được các tài liệu. Qua nghiên cứu "hệ thống quản lý thư viện", chúng ta xác định được các nhiệm vụ chính của hệ thống như: quản lý bạn đọc, cho mượn sách, nhận trả sách, thông báo nhắc trả sách, v.v. Như vậy, khi đã nghiên cứu để hiểu rõ được bài toán và xác định được các yêu cầu của hệ thống thì các chức năng, nhiệm vụ của hệ thống gần như là không thay đổi suốt trong quá trình phát triển tiếp theo ngoại trừ khi cần phải khảo sát lại bài toán. Dựa chính vào chức năng (thuật toán) thì dữ liệu sẽ là phụ và biến đổi theo các chức năng. Do đó, hệ thống phần mềm được xem như là tập các chức năng, nhiệm vụ cần tổ chức thực thi.
2. **Phân rã chức năng và làm mịn dần theo cách từ trên xuống** *(Top/Down)***.** Khả năng của con người là có giới hạn khi khảo sát, nghiên cứu để hiểu và thực thi

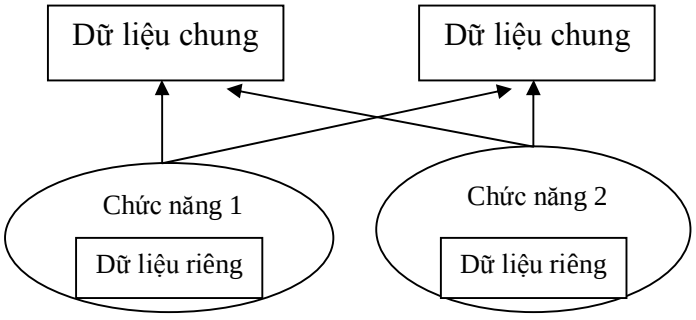
những gì mà hệ thống thực tế đòi hỏi. Để thống trị (quản lý được) độ phức tạp của những vấn đề phức tạp trong thực tế thường chúng ta phải sử dụng *nguyên lý chia để trị,* nghĩa là *phân tách nhỏ các chức năng chính thành các chức năng đơn giản hơn theo cách từ trên xuống*. Quá trình này được lặp lại cho đến khi thu được những đơn thể chức năng tương đối đơn giản, hiểu được và thực hiện cài đặt chúng mà không làm tăng thêm độ phức tạp để liên kết chúng trong hệ thống. Độ phức tạp liên kết các thành phần chức năng của hệ thống thường là tỉ lệ nghịch với độ phức tạp của các đơn thể. Vì thế một vấn đề đặt ra là có cách nào để biết khi nào quá trình phân tách các đơn thể chức năng hay còn gọi là quá trình làm mịn dần này kết thúc. Thông thường thì quá trình thực hiện phân rã các chức năng của hệ thống phụ thuộc nhiều vào độ phức hợp của bài toán ứng dụng và vào trình độ của những người tham gia phát triển phần mềm. Một hệ thống được phân tích dựa trên các chức năng hoặc quá trình sẽ được chia thành các hệ thống con và tạo ra *cấu trúc phân cấp các chức năng.* Ví dụ, hệ thống quản lý thư viện có thể phân chia từ trên xuống như sau:




Hình 1-2 Sơ đồ chức năng của Hệ thống quản lý thư viện

Chúng ta có thể khẳng định là các chức năng của nhiều hệ thống thông tin quản lý đều có thể tổ chức thành sơ đồ chức năng theo cấu trúc phân cấp có thứ bậc.

3. **Các đơn thể chức năng trao đổi với nhau bằng cách truyền tham số hay sử dụng dữ liệu chung.** Một hệ thống phần mềm bao giờ cũng phải được xem như *là một thể thống nhất,* do đó các đơn thể chức năng phải có quan hệ trao đổi thông tin, dữ liệu với nhau. Trong một chương trình gồm nhiều hàm (thực hiện nhiều chức năng khác nhau) muốn trao đổi dữ liệu được với nhau thì nhất thiết phải *sử dụng dữ liệu liệu chung hoặc liên kết với nhau bằng cách truyền tham biến*. Mỗi đơn thể chức năng không những chỉ thao tác, xử lý trên những biến *dữ liệu cục bộ* mà còn phải sử dụng các biến chung, thường đó là *các biến toàn cục.*




Hình 1-3 Mối quan hệ giữa các chức năng trong hệ thống

Với việc sử dụng những biến toàn cục thì những bất lợi trong quá trình thiết kế và lập trình là khó tránh khỏi. Đối với những dự án lớn, phức tạp có nhiều nhóm tham gia, mỗi nhóm chỉ đảm nhận một số chức năng nhất định và như thế khi một nhóm có yêu cầu thay đổi về dữ liệu chung đó thì sẽ kéo theo tất cả các nhóm khác có liên quan cũng phải thay đổi theo. Kết quả là khi có yêu cầu thay đổi của một đơn thể chức năng sẽ ảnh hưởng tới các chức năng khác và do đó sẽ ảnh hưởng tới hiệu xuất lao động của các nhóm cũng như của cả dự án. Mặt khác, các chức năng của hệ thống có nhu cầu phải thay đổi là tất yếu và rất thường xuyên.

4. **Tính mở và thích nghi của hệ thống** được xây dựng theo cách tiếp cận này *là thấp* vì:

   - *Hệ thống được xây dựng dựa vào chức năng là chính mà trong thực tế thì chức năng, nhiệm vụ của hệ thống lại hay thay đổi.* Để đảm bảo cho hệ thống thực hiện được công việc theo yêu cầu, nhất là những yêu cầu về mặt chức năng đó lại bị thay đổi là công việc phức tạp và rất tốn kém. *Ví dụ*: giám đốc thư viện yêu cầu thay đổi cách quản lý bạn đọc hoặc hơn nữa, yêu cầu bổ sung chức năng theo dõi những tài liệu mới mà bạn đọc thường xuyên yêu cầu để đặt mua, v.v. Khi đó vấn đề duy trì hệ thống phần mềm không phải là vấn đề dễ thực hiện. Nhiều khi có những yêu cầu thay đổi cơ bản mà việc sửa đổi không hiệu quả và vì thế đòi hỏi phải thiết kế lại hệ thống thì hiệu quả hơn.

   - *Các bộ phận của hệ thống phải sử dụng biến toàn cục để trao đổi với nhau, do vậy khả năng thay đổi, mở rộng của chúng và của cả hệ thống là bị hạn chế.* Như trên đã phân tích, những thay đổi liên quan đến các dữ liệu chung sẽ ảnh hưởng tới các bộ phận liên quan. Do đó, một thiết kế tốt phải rõ ràng, dễ hiểu và mọi sửa đổi chỉ có hiệu ứng cục bộ.

5. **Khả năng tái sử dụng bị hạn chế và không hỗ cơ chế kế thừa.** Để có độ thích nghi cao thì mỗi thành phần phải là tự chứa. Muốn là tự chứa hoàn toàn thì một thành phần không nên dùng các thành phần ngoại lai. Tuy nhiên, điều này lại mâu thuẫn với kinh nghiệm nói rằng các thành phần hiện có nên là dùng lại được. Vậy là cần có một sự cân bằng giữa tính ưu việt của sự dùng lại các thành phần (ở đây chủ yếu là các hàm) và sự mất mát tính thích ứng được của chúng. Các thành của hệ thống phải có tính cố kết nhưng phải tương đối lỏng để dễ thích nghi. Một trong cơ chế chính hỗ trợ để dễ có được tính thích nghi là *kế thừa* thì cách tiếp cận hướng chức năng lại không hỗ trợ. Đó là cơ chế biểu diễn tính tương tự của các thực thể, đơn giản hoá định nghĩa những khái niệm tương tự từ những sự vật đã được định nghĩa trước trên cơ sở bổ sung hay thay đổi một số các đặc trưng hay tính chất của chúng. Cơ chế này giúp chúng ta thực hiện được nguyên lý tổng quát hoá và chi tiết hoá các thành phần của hệ thống phần mềm.

### 1.3.2 Cách tiếp cận hướng đối tượng

Để khắc phục được những vấn đề tồn tại nêu trên thì chúng ta cần phải nghiên cứu phương pháp, mô hình và công cụ mới, thích hợp cho việc phát triển phần mềm đáp ứng các yêu cầu của khách hàng. Mô hình hướng đối tượng ([1], [4], [9]) có thể giúp chúng ta vượt được khủng hoảng trong công nghệ phần mềm và hy vọng sẽ đưa

ra được những sản phẩm phần mềm thương mại chất lượng cao: tin cậy, dễ mở rộng, dễ thích nghi, cường tráng và phù hợp với yêu cầu của khách hàng. Cách tiếp cận hướng đối tượng có những đặc trưng sau.

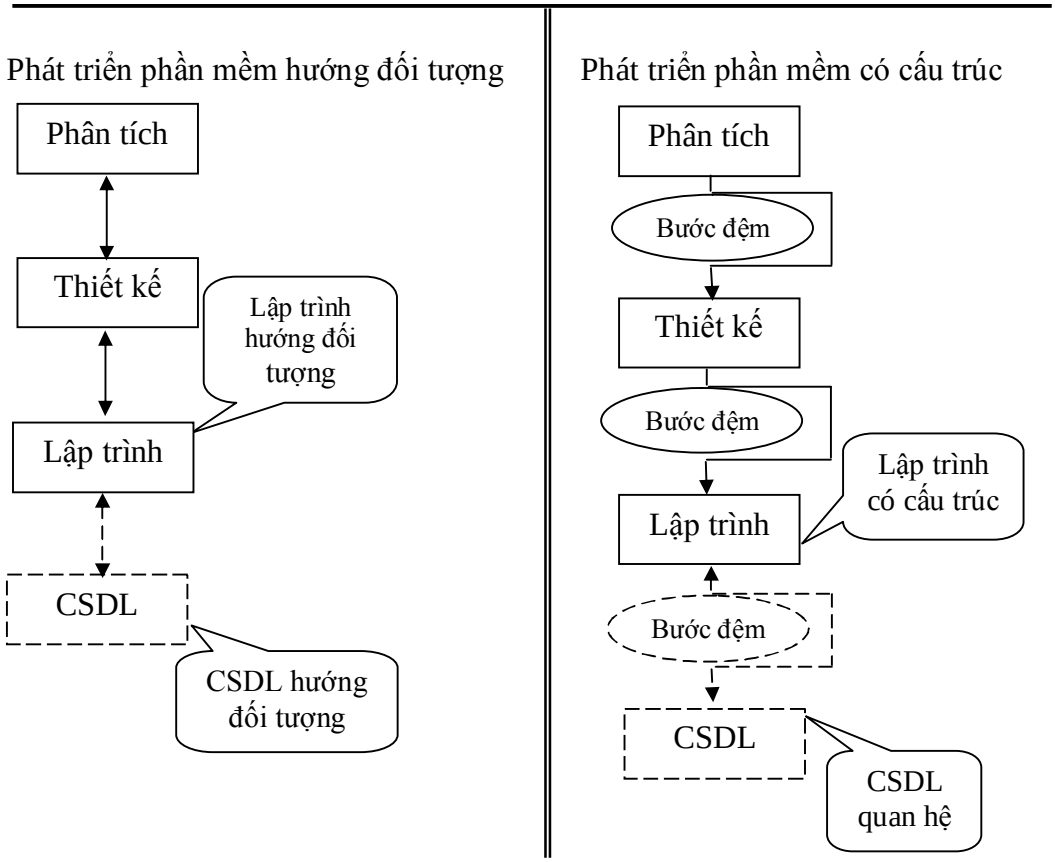
1. **Đặt trọng tâm vào dữ liệu (thực thể).** Khi khảo sát, phân tích một hệ thống chúng ta không tập trung vào các nhiệm vụ như trước đây mà tìm hiểu xem nó gồm những thực thể nào. *Thực thể hay còn gọi là đối tượng*, là những gì như người, vật, sự kiện, v.v. mà chúng ta đang quan tâm, hay cần phải xử lý. Ví dụ, khi xây dựng "*Hệ thống quản lý thư viện*" thì trước hết chúng ta tìm hiểu xem nó gồm những lớp đối tượng hoặc những khái niệm nào.
2. **Xem hệ thống như là tập các thực thể, các đối tượng.** Để hiểu rõ về hệ thống, chúng ta *phân tách hệ thống thành các đơn thể đơn giản hơn*. Quá trình này được lặp lại cho đến khi thu được những đơn thể tương đối đơn giản, dễ hiểu và thực hiện cài đặt chúng mà không làm tăng thêm độ phức tạp khi liên kết chúng trong hệ thống. Xét "*Hệ thống quản lý thư viện*", chúng ta có các lớp đối tượng sau:


Hình 1-4 Tập các lớp đối tượng của hệ thống

3. **Các lớp đối tượng trao đổi với nhau bằng các thông điệp.** Theo nghĩa thông thường thì lớp là nhóm một số người, vật có những đặc tính tương tự nhau hoặc có những hành vi ứng xử giống nhau. Trong mô hình đối tượng, khái niệm lớp là cấu trúc mô tả hợp nhất các thuộc tính, hay dữ liệu thành phần thể hiện các đặc tính của mỗi đối tượng và các phương thức, hay hàm thành phần thao tác trên các dữ liệu riêng và là giao diện trao đổi với các đối tượng khác để xác định hành vi của chúng trong hệ thống. Khi có yêu cầu dữ liệu để thực hiện một nhiệm vụ nào đó, một đối tượng sẽ gửi một thông điệp (gọi một phương thức) cho đối tượng khác. Đối tượng nhận được thông điệp yêu cầu sẽ phải thực hiện một số công việc trên các dữ liệu mà nó sẵn có hoặc lại tiếp tục yêu cầu những đối tượng khác hỗ trợ để có những thông tin trả lời cho đối tượng yêu cầu. Với phương thức xử lý như thế thì một chương trình hướng đối tượng thực sự có thể không cần sử dụng biến toàn cục nữa.
4. **Tính mở và thích nghi của hệ thống cao hơn vì**:

   - Hệ thống được xây dựng dựa vào các lớp đối tượng nên khi có yêu cầu thay đổi thì chỉ thay đổi những lớp đối tượng có liên quan hoặc bổ sung thêm một số lớp đối tượng mới (có thể kế thừa từ những lớp có trước) để thực thi những nhiệm vụ mới mà hệ thống cần thực hiện. *Ví dụ*: Giám đốc thư viện yêu cầu bổ sung chức năng theo dõi những tài liệu mới mà bạn đọc thường xuyên yêu cầu để đặt mua, ta có thể bổ sung thêm lớp mới để theo dõi yêu cầu: lớp Yêu_Cầu.
   - Trong các chương trình hướng đối tượng có thể không cần sử dụng biến toàn cục nên mọi sửa đổi, cập nhật trong mỗi thành phần chỉ có hiệu ứng cục bộ.

5. **Hỗ trợ sử dụng lại và cơ chế kế thừa**. Các lớp đối tượng được tổ chức theo nguyên lý *bao gói* và *che giấu thông tin*, điều này làm tăng thêm hiệu quả của kế thừa và độ tin cậy của hệ thống. Các ngôn ngữ lập trình hướng đối tượng như: C++, Java, C#, Delphi, v.v. đều hỗ trợ quan hệ kế thừa.




Hình 1-5 Hai phương pháp chính trong phát triển phần mềm

### 1.3.3 Ưu điểm chính của phương pháp hướng đối tượng

- Đối tượng là cơ sở để kết hợp các đơn thể có thể sử dụng lại thành hệ thống lớn hơn, tạo ra những sản phẩm có chất lượng cao.
- Qui ước truyền thông điệp giữa các đối tượng đảm bảo cho việc mô tả các giao diện giữa các đối tượng thành phần bên trong hệ thống và những hệ thống bên ngoài trở nên dễ dàng hơn. Điều đó giúp cho việc phân chia những dự án lớn, phức tạp để phân tích, thiết kế theo cách chia nhỏ bài toán thành các lớp đối tượng hoàn toàn tương ứng với quan điểm hướng tới lời giải phù hợp với thế giới thực một các tự nhiên.
- Nguyên lý bao gói, che giấu thông tin hỗ trợ cho việc xây dựng những hệ thống thông tin an toàn.

- Nguyên lý kế thừa dựa chính vào dữ liệu rất phù hợp với ngữ nghĩa của mô hình trong cài đặt.
- Lập trình hướng đối tượng đặc biệt là kỹ thuật kế thừa cho phép dễ dàng xác định các đơn thể và sử dụng ngay khi chúng chưa thực hiện đầy đủ các chức năng (đơn thể mở) và sau đó mở rộng được mà không làm ảnh hưởng tới các đơn thể khác.
- Định hướng đối tượng cung cấp những công cụ, môi trường mới, hiệu quả để phát triển phần mềm theo hướng công nghiệp và hỗ trợ để tận dụng được những khả năng kế thừa, sử dụng lại ở phạm vi diện rộng để xây dựng được những hệ thống phức tạp, nhạy cảm như: hệ thống động, hệ thống thời gian thực, v,v.
- Xoá bỏ được hố ngăn cách giữa các pha phân tích, thiết kế và cài đặt trong quá trình xây dựng phần mềm.

Hình 1-5 mô tả sự giống và khác nhau của hai cách tiếp cận trong quá trình phát triển phần mềm.

## 1.4 Các mô hình chu trình phát triển phần mềm

Có nhiều kiểu mô hình cho quá trình phát triển phần mềm. Ivan Sommerville [3] nói tới năm loại mô hình khác nhau.

1. **Mô hình thác nước** [12]: quá trình phần mềm được chia thành dãy các pha liên tiếp từ phân tích yêu cầu, phân tích, thiết kế hệ thống, lập trình đến thử nghiệm và triển khai hệ thống. Pha sau chỉ được bắt đầu khi pha trước đã hoàn thành. Mô hình này được thiết lập theo cách tiếp cận hướng chức năng và phù hợp cho những dự án lớn, phức tạp.

   *Ưu điểm:* + Thích hợp cho những dự án lớn.

   + Dự án thực hiện lần lượt theo các pha của một tiến trình nên việc quản lý dự án sẽ dễ dàng và thuận tiện.

   *Nhược điểm:* + Các yêu cầu của NSD (người sử dụng) không phản ánh, trao đổi được với nhóm phát triển cho đến khi hoàn tất từng giai đoạn phát triển.

   + Không cho phép thay đổi nhiều theo các đặc tả yêu cầu của hệ thống.

2. **Mô hình thăm dò (hình xoán ốc)**: Phát triển càng nhanh càng tốt một hệ thống rồi cải tiến hệ thống đó cho tới khi nó đáp ứng được các yêu cầu của khách hàng. Các bước thực hiện cũng giống như mô hình thác nước nhưng luôn có xét tới các yếu tố khả thi, các sự cố tác động vào hệ thống, nghĩa là phân tích các yêu tố rủi ro và những yêu cầu mới, thay đổi của NSD nhằm tạo ra những phần mềm gần với những yêu cầu thực tế hơn. Theo Raccoon thì những năm gần đây người ta quan tâm nhiều hơn tới **mô hình xoắn ốc** được Boëhm đưa ra năm 1988. Phát triển phần mềm theo mô hình này dựa trên việc *phân tích các rủi ro*. Quá trình phát

triển được chia thành nhiều thời kỳ, mỗi thời kỳ bắt đầu bằng việc phân tích, rồi tạo nguyên mẫu, các công đoạn để cải tạo, duyệt lại và cứ thế tiếp tục cho tới khi đạt được muc đích.

*Ưu điểm:* + Linh hoạt hơn trong quá trình phát triển hệ thống cho thích hợp với những thay đổi trong đặc tả yêu cầu.

*Nhược điểm:* + Các pha thực hiện bị lặp nhiều trong cả quá trình phát triển hệ thống.

3. **Tạo nguyên mẫu**: (gần như mô hình thăm dò) phát triển một hệ thống cho người dùng thử nghiệm, rồi thiết lập các yêu cầu và tạo ra nguyên mẫu mới cho tới khi sản phẩm đạt yêu cầu. NSD và những người phát triển hệ thống có thể trao đổi với nhau để thống nhất về những yêu cầu trong quá trình phát triển phần mềm.

   Ngày nay với công nghệ thế hệ thứ tư 4GT bao gồm nhiều cái mới như các ngôn ngữ khai báo, các gói chương trình ứng dụng, nhiều phần mềm giao diện rất mạnh, các công cụ trợ giúp CASE, v.v. Lợi dụng khả năng này, người ta nhanh chóng xây dựng một phương án thô để phát triển một nguyên mẫu rồi đem cho NSD dùng thử. Nếu phát hiện được chỗ NSD chưa bằng lòng, thì chỉnh sửa lại và hoàn thiện để có nguyên mẫu tiếp theo. Cứ thế thành lập một dãy các nguyên mẫu, rốt cuộc người ta đạt được hệ thống đáp ứng các yêu cầu NSD.

   *Ưu điểm:* + Cho phép xây dựng những hệ thống thực hiện hiệu quả các chức năng mà NSD yêu cầu.

   + Trong quá trình thực hiện cho phép kiểm tra các yêu cầu của NSD có cần thiết, có đáp ứng thực tế hay không, do vậy cho phép bổ sung kịp thời và đồng thời loại bỏ đi những điểm không cần thiết.

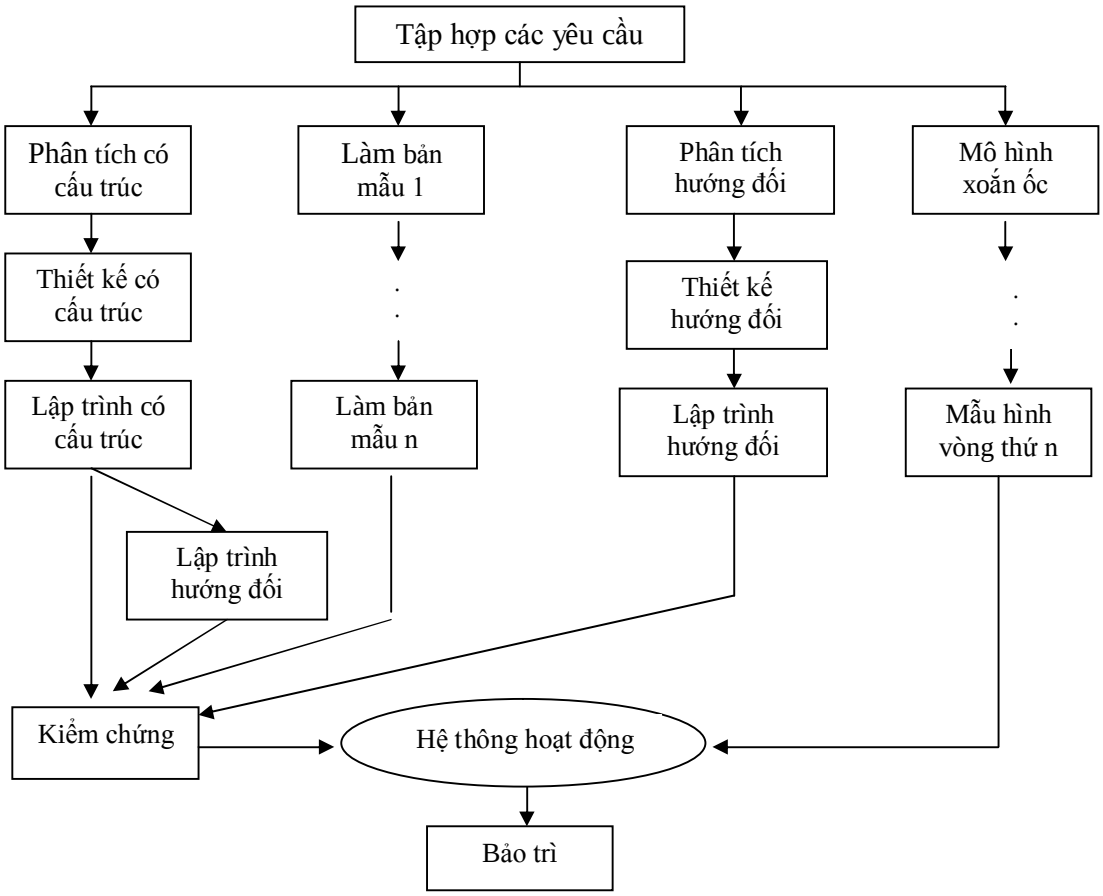
   + Các chức năng, hiệu xuất và khả năng thao tác của hệ thống có thể kiểm nghiệm trong quá trình phát triển hệ thống, do vậy tổng thời gian phát triển có thể sẽ được rút ngắn.

   *Nhược điểm:* + Không thích hợp cho những dự án lớn, chỉ thích hợp cho những dự án vừa và nhỏ.

4. **Biến đổi hình thức**: Phát triển một đặc tả hình thức cho một hệ thống và phân tích để biến đổi các đặc tả đó (đảm bảo tính đúng đắn của các phép biến đổi) cho tới khi có được một chương trình thoả mãn các yêu cầu.

5. **Tập hợp các thành phần dùng lại được** để xây dựng phần mềm thoả các yêu cầu. Việc tạo lập hệ thống được thực hiện bằng cách lắp ráp các thành phần có sẵn. Theo Hooper, Chester và Kang thì quá trình tập hợp các thành phần gồm 6 bước: *nhận thức bài toán, hình thành giải pháp, tìm kiếm các thành phần, điều chỉnh và thích ứng các thành phần, tích hợp chúng và đánh giá hệ thống được tuyển chọn*.

   *Tóm lại*, khuôn cảnh chung của kỹ nghệ phần mềm có thể được mô tả như sau:




Hình 1- 6 Mô hình phát triển phần mềm

## Câu hỏi và bài tập

1.1 Hệ thống phần mềm là gì?, nếu các đặc trưng cơ bản của sản phẩm phần mềm?

1.2 Vai trò và mục đích của mô hình hoá trong quá trình phát triển phần mềm?

1.3 Tại sao lại cần phải có một qui trình phát triển phần mềm thống nhất?

1.4 Phân tích các đặc trưng cơ bản của cách tiếp cận hướng chức năng và hướng đối tượng trong quá trình phát triển phần mềm.

*1.5* Nêu những mô hình cơ bản được ứng dụng để phát triển hệ thống hiện nay?

*1.6* Chọn từ danh sách dưới đây những thuật ngữ thích hợp để điền vào các chỗ [(…)] trong đoạn văn mô tả về hệ thống phần mềm.

*Hệ thống phần mềm* hay gọi tắt là *hệ thống,* là tổ hợp các [(1)], [(2)] có quan hệ qua lại với nhau, [(3)] thông qua việc nhận các dữ liệu đầu vào (*Input*) và sản sinh ra những kết quả đầu ra (*Output*) thường là ở các dạng thông tin khác nhau nhờ một [(4)], biến đổi [(5)].

*Chọn câu trả lời:*

a. cùng hoạt động hướng tới mục tiêu chung

b. quá trình xử lý

c. phần mềm

d. có tổ chức

e. phần cứng

1.7 Hãy chọn những thuật ngữ thích hợp nhất để điền vào các chỗ [(…)] trong đoạn văn dưới đây mô tả về quá trình phân tích hướng chức năng.

Để hiểu được những hệ thống lớn, phức tạp, chúng ta thường phải sử dụng nguyên lý [(1)], nghĩa là [(2)] chính thành các chức năng đơn giản hơn theo cách tiếp cận [(3)]. Qui trình này được lặp lại cho đến khi thu được những đơn thể chức năng tương đối đơn giản, dễ hiểu và thực hiện cài đặt chúng mà không làm tăng thêm độ phức tạp để liên kết chúng trong [(4)].

*Chọn câu trả lời:*

a. *từ trên xuống*

*b. phân tách nhỏ các chức năng*

*c. hệ thống*

*d.* chia để trị (*devide and conquer*)

1.8 Hãy chọn dãy các bước thực hiện trong danh sách dưới đây cho phù hợp với qui trình phát triển phần mềm theo mô hình "*thác nước*".

(1) Xác định các yêu cầu

(2) Thiết kế hệ thống

(3) Cài đặt và kiểm tra hệ thống

(4) Vận hành và bảo trì hệ thống

(5) Phân tích hệ thống

*Chọn câu trả lời:*

*a. (2)* → (1)→*(3)*→*(5)*→*(4)*

*b. (1)* → (2)→*(3)*→*(4)*→*(5)*

*c. (1)* → (5)→*(2)*→*(3)*→*(4)*

*d. (1)* → (3)→*(2)*→*(5)*→*(4)*

# CHƯƠNG II
# UML VÀ QUÁ TRÌNH PHÁT TRIỂN PHẦN MỀM

Nội dung của chương II:

- ✓ Giới thiệu tóm lược về ngôn ngữ mô hình hoá thống nhất UML
- ✓ Các khái niệm cơ bản của phương pháp hướng đối tượng,
- ✓ Mối quan hệ giữa các lớp đối tượng,
- ✓ Quá trình phát triển phần mềm.

Để xây dựng được một sản phẩm phần mềm tốt, đương nhiên là cần một phương pháp phù hợp. Phương pháp phát triển phù hợp là sự kết hợp của ba yếu tố:

(i) Một tập hợp các khái niệm và mô hình, bao gồm các khái niệm cơ bản sử dụng trong phương pháp cùng với cách biểu diễn chúng (thường là dưới dạng đồ thị, biểu đồ).

(ii) Một quá trình triển khai, bao gồm các bước thực hiện lần lượt, các hoạt động cần thiết.

(iii) Một công cụ mạnh trợ giúp cho việc triển khai hệ thống chặt chẽ và nhanh chóng.

UML là ngôn ngữ chuẩn giúp chúng ta thể hiện được các yếu tố nêu trên của phương pháp phân tích, thiết kế hướng đối tượng.

## 2.1 Tổng quát về UML

UML là ngôn ngữ mô hình hoá, trước hết nó mô tả ký pháp thống nhất, ngữ nghĩa các định nghĩa trực quan tất cả các thành phần của mô hình ([1], [2]). UML được sử dụng để hiển thị, đặc tả, tổ chức, xây dựng và làm tài liệu các vật phẩm của quá trình phát triển phần mềm hướng đối tượng, đặc biệt là phân tích, thiết kế dưới dạng các báo cáo, biểu đồ, bản mẫu hay các trang web, v.v. UML là ngôn ngữ mô hình hoá độc lập với các công nghệ phát triển phần mềm.

### 2.1.1 Mục đích của UML

*Mục đích chính của UML:*

1. Mô hình được các hệ thống (không chỉ hệ thống phần mềm) và sử dụng được tất cả các khái niệm hướng đối tượng một cách thống nhất.
2. Cho phép đặc tả, hỗ trợ để đặc tả tường minh (trực quan) mối quan hệ giữa các khái niệm cơ bản trong hệ thống, đồng thời mô tả được mọi trạng thái hoạt động của hệ thống đối tượng. Nghĩa là cho phép mô tả được cả mô hình tĩnh lẫn mô hình động một cách đầy đủ và trực quan.

3. Tận dụng được những khả năng sử dụng lại và kế thừa ở phạm vi diện rộng để xây dựng được những hệ thống phức tạp và nhạy cảm như: các hệ thống động, hệ thống thời gian thực, hệ thống nhúng thời gian thực, v.v.
4. Tạo ra những ngôn ngữ mô hình hoá sử dụng được cho cả người lẫn máy tính.

*Tóm lại, UML là ngôn ngữ mô hình hoá, ngôn ngữ đặc tả và ngôn ngữ xây dựng mô hình trong quá trình phát triển phần mềm, đặc biệt là trong phân tích và thiết kế hệ thống hướng đối tượng*. UML là ngôn ngữ hình thức, thống nhất và chuẩn hoá mô hình hệ thống một cách trực quan. Nghĩa là các thành phần trong mô hình được thể hiện bởi các ký hiệu đồ hoạ, biểu đồ và thể hiện đầy đủ mối quan hệ giữa các chúng một cách thống nhất và có logic chặt chẽ.

*Tuy nhiên cũng cần lưu ý:*

- UML không phải là ngôn ngữ lập trình, nghĩa là ta không thể dùng UML để viết chương trình. Nó cũng không phải là một công cụ CASE. Một số công cụ CASE như Rational Rose [8] sử dụng mô hình UML để phát sinh mã nguồn tự động sang những ngôn ngữ lập trình được lựa chọn như C++, Java, Visual C++, v.v.
- UML cũng không phải là một phương pháp hay một quá trình phát triển phần mềm. Các ký hiệu UML được sử dụng trong các dự án phát triển phần mềm nhằm áp dụng những cách tiếp cận khác nhau cho quá trình phát triển phần mềm nhằm tách chu kỳ phát triển hệ thống thành những hoạt động, các tác vụ, các giai đoạn và các bước khác nhau.

### 2.1.2 Qui trình phát triển phần mềm thống nhất

UML được phát triển để đặc tả quá trình phát triển phần mềm, nhằm mô hình hoá hệ thống. Qui trình phát triển phần mềm này gọi là *qui trình phát triển phần mềm hợp nhất* (USPD) hay qui trình hợp nhất Rational (RUP [8]), gọi tắt là *qui trình hợp nhất* (UP).

UP bao gồm con người, dự án, sản phẩm, qui trình và công cụ. *Con người* là những người tham gia *dự án* để tạo ra *sản phẩm phần mềm* theo một *qui trình* với sự hỗ trợ *của công cụ* được cung cấp.

UP là qui trình phát triển phần mềm được *hướng dẫn bởi các ca sử dụng*. Nghĩa là các yêu cầu của NSD được mô tả trong các ca sử dụng, là chuỗi các hành động được thực hiện bởi hệ thống nhằm cung cấp các dịch vụ, các thông tin cho khách hàng. Các ca sử dụng bao gồm chuỗi các công việc được xem là nền tảng để tạo ra mô hình thiết kế và cài đặt hệ thống.

UP cũng là qui trình *tập trung vào kiến trúc, được lặp và phát triển tăng trưởng* liên tục. Kiến trúc của hệ thống phải được thiết kế nhằm đáp ứng các yêu cầu của các ca sử dụng chính, trong giới hạn của chuẩn phần cứng mà hệ thống sẽ chạy và của cấu trúc của cả hệ thống lẫn các hệ thống con. Tính lặp của quá trình phát triển phần mềm được thể hiện ở chỗ là một dự án được chia thành các dự án nhỏ và được thực hiện lặp lại trong từng bước thực hiện. Mỗi dự án nhỏ đều thực hiện phân tích, thiết kế, cài đặt

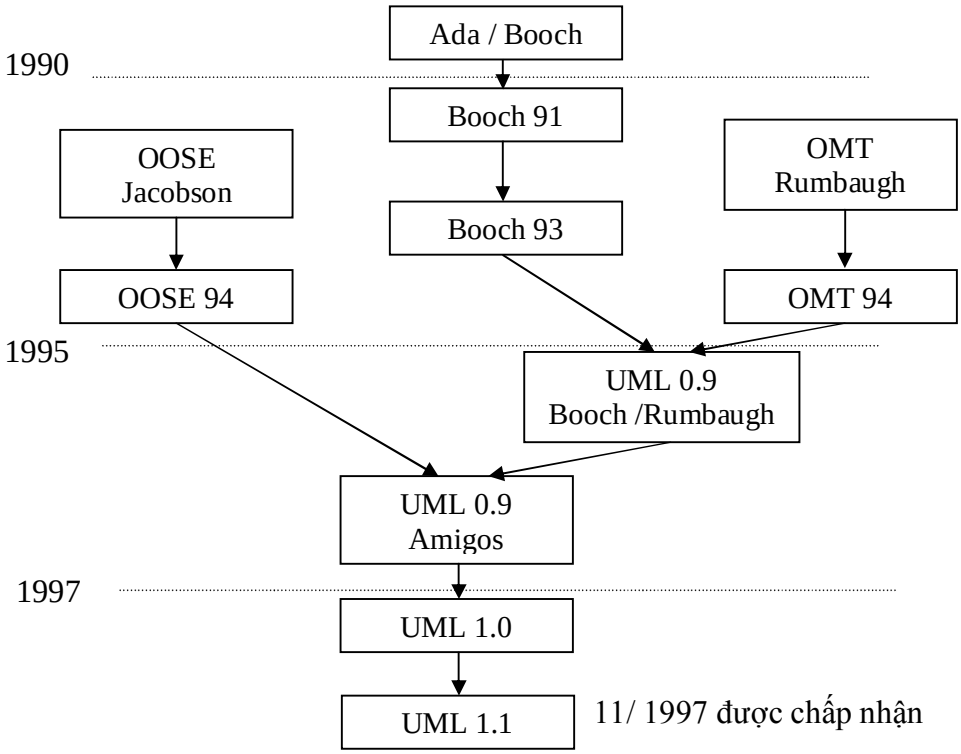
và kiểm thử, v.v. Mỗi phần việc đó được phát triển tăng trường và cả dự án cũng được thực hiện theo sự tăng trưởng này.

UP không chỉ tạo ra một hệ thống phần mềm hoàn chỉnh mà còn tạo ra một số sản phẩm trung gian như các mô hình. Các mô hình chính trong UP là mô hình nghiệp vụ (ca sử dụng), mô hình khái niệm, mô hình thiết kế, mô hình triển khai và mô hình trắc nghiệm. Các mô hình này có sự phụ thuộc theo vết phát triển, nghĩa là có thể lần theo từng mô hình để đến được mô hình trước.

### 2.1.3 Giới thiệu tổng quát về UML

UML được xây dựng dựa chính vào:

- *Cách tiếp cận của Booch (Booch Approach),*
- *Kỹ thuật mô hình đối tượng (OMT – Object Modeling Technique)* của Rumbaugh,
- *Công nghệ phần mềm hướng đối tượng (OOSE – Object-Oriented Software Engineering)* của Jacobson,
- *Đồng thời thống nhất được nhiều ký pháp*, khái niệm của các phương pháp khác. Quá trình hình thành UML bắt đầu từ ngôn ngữ Ada (Booch) trước năm 1990 (hình 2-1).



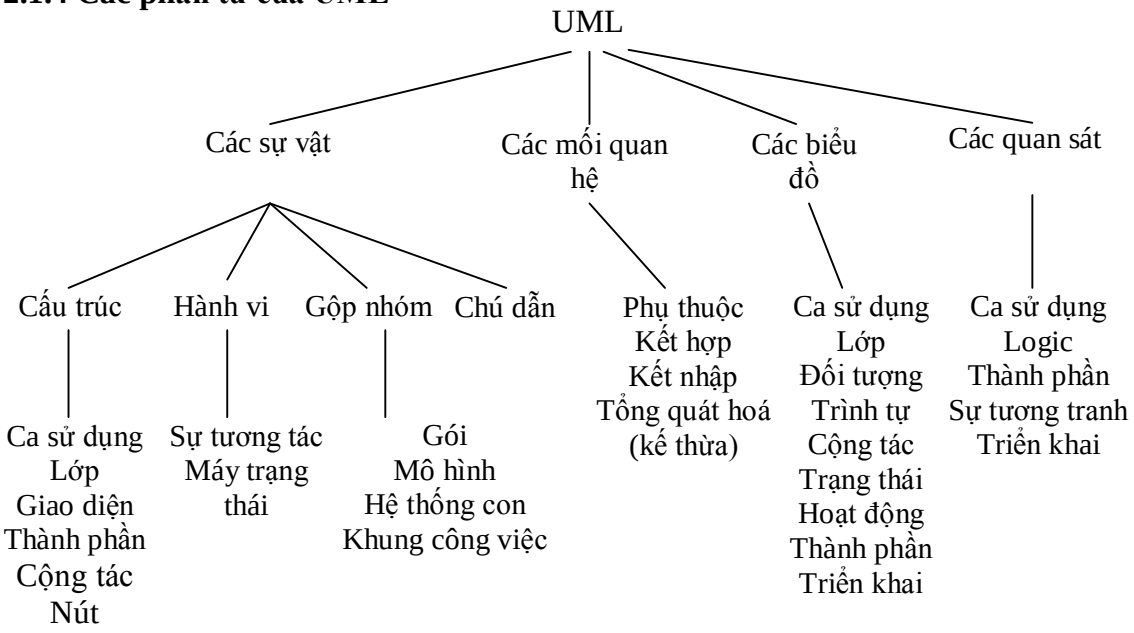

Hình 2-1 Sự phát triển của UML

Để hiểu và sử dụng tốt UML trong phân tích, thiết kế hệ thống, đòi hỏi phải nắm bắt được ba vấn đề chính:

1. Các phần tử cơ bản của UML,

2. Những qui định liên kết giữa các phần tử, các qui tắc cú pháp,
3. Những cơ chế chung áp dụng cho ngôn ngữ mô hình hoá hệ thống.

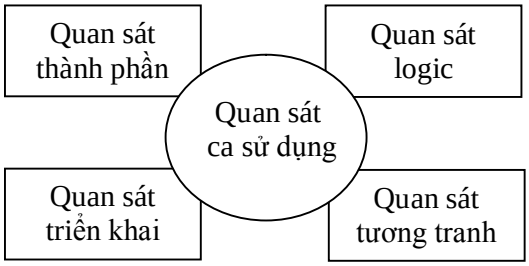
### 2.1.4 Các phần tử của UML




Hình 2-2 Các thành phần cơ sở của UML

**Các quan sát**

Các quan sát (góc nhìn) theo các phương diện khác nhau của hệ thống cần phân tích, thiết kế. Dựa vào các quan sát để thiết lập kiến trúc cho hệ thống cần phát triển. Có năm loại quan sát: *quan sát theo ca sử dụng, quan sát logic, quan sát thành phần, quan sát tương tranh và quan sát triển khai*. Mỗi quan sát tập trung khảo sát và mô tả một khía cạnh của hệ thống (hình 2-3) và thường được thể hiện trong một số biểu đồ nhất định.




Hình 2-3 Các quan sát của hệ thống

- *Quan sát các ca sử dụng (hay trường hợp sử dụng)*: mô tả các chức năng, nhiệm vụ của hệ thống. Quan sát này thể hiện mọi yêu cầu của hệ thống, do vậy nó phải được xác định ngay từ đầu và nó được sử dụng để điều khiển, thúc đẩy và thẩm định hay kiểm tra các công việc của tất cả các giai đoạn của cả quá trình phát triển phần mềm. Nó cũng là cơ sở để trao đổi giữa các thành viên của dự án phần mềm và với khách hàng. Quan sát ca sử dụng được thể hiện trong các *biểu đồ ca sử* và có thể ở một vài *biểu đồ trình tự, cộng tác*, v.v.
- *Quan sát logic* biểu diễn tổ chức logic của các lớp và các quan hệ của chúng với nhau. Nó mô tả cấu trúc tĩnh của các lớp, đối tượng và sự liên hệ của chúng thể hiện mối liên kết động thông qua sự trao đổi các thông điệp. Quan sát được thể hiện trong các *biểu đồ lớp, biểu đồ đối tượng, biểu đồ tương tác, biểu đồ biến đổi trạng thái*. Quan sát logic tập trung vào cấu trúc của hệ thống. Trong quan sát này ta nhận ra các bộ phận cơ bản cấu thành hệ thống thể hiện mọi quá trình trao đổi, xử lý thông tin cơ bản trong hệ thống.
- *Quan sát thành phần (quan sát cài đặt)* xác định các mô đun vật lý hay tệp mã chương trình và sự liên hệ giữa chúng để tổ chức thành hệ thống phần mềm. Trong quan sát này ta cần bổ sung: chiến lược cấp phát tài nguyên cho từng thành phần, và thông tin quản lý như báo cáo tiến độ thực hiện công việc, v.v. Quan sát thành phần được thể hiện trong các *biểu đồ thành phần* và *các gói.*
- *Quan sát tương tranh (quan sát tiến trình)* biểu diễn sự phân chia các luồng thực hiện công việc, các lớp đối tượng cho các *tiến trình* và sự đồng bộ giữa các *luồng* trong hệ thống. Quan sát này tập trung vào các nhiệm vụ tương tranh, tương tác với nhau trong hệ thống đa nhiệm.
- *Quan sát triển khai* mô tả sự phân bổ tài nguyên và nhiệm vụ trong hệ thống. Nó liên quan đến các tầng kiến trúc của phần mềm, thường là kiến trúc ba tầng, tầng giao diện (tầng trình diễn), tầng logic tác nghiệp và tầng lưu trữ CSDL được tổ chức trên một hay nhiều máy tính. Quan sát triển khai bao gồm các luồng công việc, bộ xử lý và các thiết bị. *Biểu đồ triển khai* mô tả các tiến trình và chỉ ra những tiến trình nào trên máy nào.

**Các biểu đồ**

*Biểu đồ* là đồ thị biểu diễn đồ họa về tập các phần tử trong mô hình. Biểu đồ chứa đựng các nội dung của các quan sát dưới các góc độ khác nhau và một thành phần của hệ thống có thể xuất hiện trong một hay nhiều biểu đồ. UML cung cấp những biểu đồ trực quan để biểu diễn các khía cạnh khác nhau của hệ thống, bao gồm:

- *Biểu đồ ca sử dụng* mô tả sự tương tác giữa các tác nhân ngoài và hệ thống thông qua các ca sử dụng. Các ca sử dụng là những nhiệm vụ chính, các dịch vụ, những trường hợp sử dụng cụ thể mà hệ thống cung cấp cho người sử dụng và ngược lại.
- *Biểu đồ lớp* mô tả cấu trúc tĩnh, mô tả mô hình khái niệm bao gồm các lớp đối tượng và các mối quan hệ của chúng trong hệ thống hướng đối tượng.

- *Biểu đồ trình tự* thể hiện sự tương tác của các đối tượng với nhau, chủ yếu là trình tự gửi và nhận thông điệp để thực thi các yêu cầu, các công việc *theo thời gian*.
- *Biểu đồ cộng tác* tương tự như biểu đồ trình tự nhưng nhấn mạnh vào sự tương tác của các đối tượng trên cơ sở cộng tác với nhau bằng cách trao đổi các thông điệp để thực hiện các yêu cầu *theo ngữ cảnh công việc*.
- *Biểu đồ trạng thái* thể hiện chu kỳ hoạt động của các đối tượng, của các hệ thống con và của cả hệ thống. Nó là một loại ôtômát hữu hạn trạng thái, mô tả các trạng thái, các hành động mà đối tượng có thể có và các sự kiện gắn với các trạng thái theo thời gian.
- *Biểu đồ hành động* chỉ ra dòng hoạt động của hệ thống, bao gồm các trạng thái hoạt động, trong đó từ một trạng thái hoạt động sẽ chuyển sang trạng thái khác sau khi một hoạt động tương ứng được thực hiện. Nó chỉ ra trình tự các bước, tiến trình thực hiện cũng như các điểm quyết định và sự rẽ nhánh theo luồng sự kiện.
- *Biểu đồ thành phần* chỉ ra cấu trúc vật lý của các thành phần trong hệ thống, bao gồm: các thành phần mã nguồn, mã nhị phân, thư viện và các thành phần thực thi.
- *Biểu đồ triển khai* chỉ ra cách bố trí vật lý các thành phần theo kiến trúc được thiết kế của hệ thống.

Các khái niệm cơ bản của biểu đồ và cách xây dựng các biểu đồ trên để phân tích, thiết kế hệ thống sẽ được giới thệu chi tiết ở các chương sau.

## 2.2 Các khái niệm cơ bản của phương pháp hướng đối tượng trong UML

Để phát triển được hệ thống theo mô hình, phương pháp đã lựa chọn thì vấn đề quan trọng nhất là phải hiểu rõ những khái niệm cơ bản của phương pháp đó. Ở đây chúng ta cần thực hiện phân tích, thiết kế hệ thống theo cách tiếp cận hướng đối tượng, do vậy trước hết phải nắm bắt được những khái niệm cơ sở như: đối tượng, lớp, và các mối quan hệ giữa các lớp đối tượng. Những khái niệm này cũng là các phần tử cơ bản của ngôn ngữ mô hình hóa thống nhất UML.

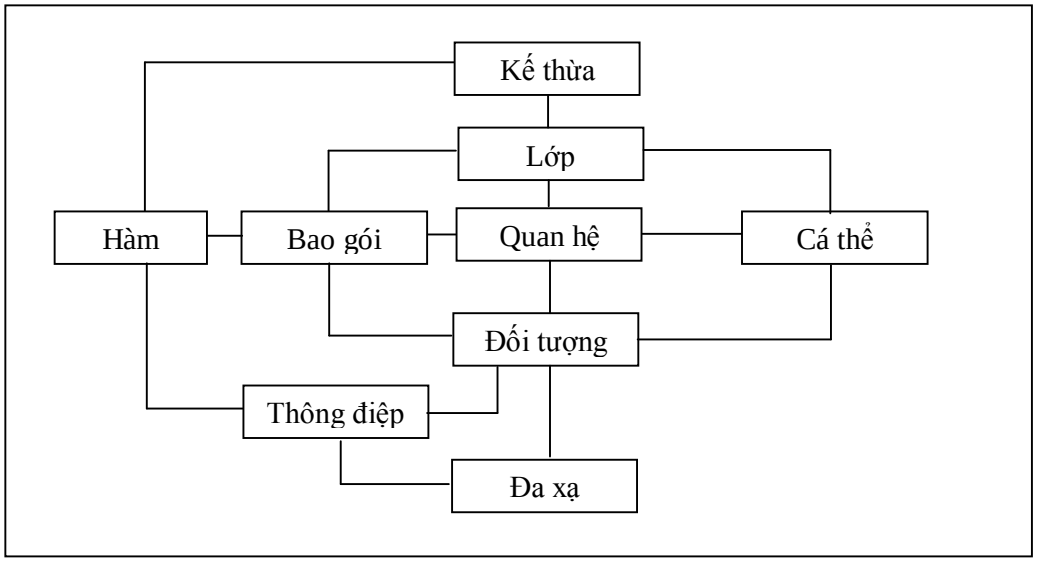
Mô hình hướng đối tượng được sử dụng để phát triển phần mềm dựa trên mô hình dữ liệu trừu tượng và khái niệm lớp để chỉ ra những đặc tính chung các cấu trúc dữ liệu được sử dụng để mô hình hoá hệ thống. Hệ thống các khái niệm cơ bản của phương pháp hướng đối tượng được mô tả như trong hình 2-4.

### 2.2.1 Các đối tượng

*Đối tượng* là khái niệm cơ sở quan trọng nhất của cách tiếp cận hướng đối tượng. *Đối tượng là một khái niệm, một sự trừu tượng hoá hay một sự vật có nghĩa trong bài toán đang khảo sát. Đó chính là các mục mà ta đang nghiên cứu, đang thảo luận về chúng*. Đối tượng là thực thể của hệ thống, của CSDL và được xác định thông qua định danh của chúng. Thông thường các đối tượng được mô tả bởi các danh từ riêng

(tên gọi) hoặc được tham chiếu tới trong các mô tả của bài toán hay trong các thảo luận với người sử dụng. Có những đối tượng là những thực thể có trong thế giới thực như người, sự vật cụ thể, hoặc là những khái niệm như một công thức, hay khái niệm trừu tượng, v.v. Có một số đối tượng được bổ sung vào hệ thống với lý do phục vụ cho việc cài đặt và có thể không có trong thực tế.

Đối tượng là những thực thể được xác định trong thời gian hệ thống hoạt động. Trong giai đoạn phân tích, ta phải đảm bảo rằng các đối tượng đều được xác định bằng các định danh. Đến khâu thiết kế, ta phải lựa chọn cách thể hiện những định danh đó theo cách ghi địa chỉ bộ nhớ, gán các số hiệu, hay dùng tổ hợp một số gái trị của một số thuộc tính để biểu diễn. Theo quan điểm của người lập trình, đối tượng được xem như là một vùng nhớ được phân chia trong máy tính để lưu trữ dữ liệu (thuộc tính) và tập các hàm thao tác trên dữ liệu được gắn với nó. Bởi vì các vùng nhớ được phân hoạch là độc lập với nhau nên các đối tượng có thể tham gia vào nhiều chương trình khác nhau mà không ảnh hưởng lẫn nhau.



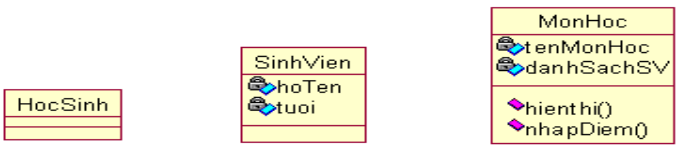

Hình 2-4 Những khái niệm cơ bản của phương pháp hướng đối tượng

### 2.2.2 Lớp đối tượng

Đối tượng là thể hiện, là một đại biểu của một lớp. *Lớp là một mô tả về một nhóm các đối tượng có những tính chất (thuộc tính) giống nhau, có chung các hành vi ứng xử (thao tác gần như nhau), có cùng mối liên quan với các đối tượng của các lớp khác và có chung ngữ nghĩa trong hệ thống*. Lớp chính là cơ chế được sử dụng để phân loại các đối tượng của một hệ thống. Lớp thường xuất hiện dưới dạng những danh từ chung trong các tài liệu mô tả bài toán hay trong các thảo luận với người sử dụng. Cũng như các đối tượng, lớp có thể là những nhóm thực thể có trong thế giới thực, cũng có những lớp là khái niệm trừu tượng và có những lớp được đưa vào trong thiết kế để phục vụ cho cài đặt hệ thống, v.v.

Lớp và mối quan hệ của chúng có thể mô tả trong các biểu đồ lớp biểu đồ đối tượng và một số biểu đồ khác của UML. Trong biểu đồ lớp, lớp được mô tả bằng một hình hộp

chữ nhật, trong đó có tên của lớp, có thể có các thuộc tính và các hàm (phương thức) như hình 2-5.




a/ Tên của lớp b/ Tên và thuộc tính c/ Tên, thuộc tính và phương thức

Hình 2-5 Các ký hiệu mô tả lớp trong UML

Chúng ta nên đặt tên theo một *qui tắc thống nhất* như sau:

+ Tên của lớp thì chữ cái đầu của tất cả các từ đều viết hoa, ví dụ: **SinhVien, HocSinh, KhachHang**, v.v.

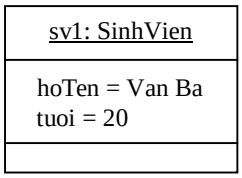
+ Tên của đối tượng, tên của thuộc tính thì viết hoa chữ cái đầu của các từ trừ từ đầu tiên, ví dụ: *hoTen, danhSachSV,* v.v.

+ Tên của hàm (phương thức) viết giống như tên của đối tượng nhưng có thêm cặp ngoặc đơn '(' và ')', ví dụ: *hienThi(), nhapDiem(),* v.v.

Trong biểu đồ ở giai đoạn phân tích, một lớp có thể chỉ cần có tên lớp, tên và thuộc tính, hoặc có cả tên gọi, thuộc tính và các phương thức như hình 2-5.

### 2.2.3 Các giá trị và các thuộc tính của đối tượng

*Giá trị (value)* là một phần của dữ liệu. Các giá trị thường là các số hoặc là các ký tự. *Thuộc tính của đối tượng* là thuộc tính của lớp được mô tả bởi giá trị của mỗi đối tượng trong lớp đó. Ví dụ




Hình 2-6 Ký hiệu đối tượng trong UML

"Van Ba" và 20 là hai giá trị tương ứng với hai thuộc tính *hoTen, tuoi* của đối tượng sv1 trong lớp **SinhVien**.

Không nên nhầm lẫn giá trị với đối tượng. Các đối tượng có định danh chứ không phải là các giá trị. Có thể có ba sinh viên cùng tên "Van Ba", nhưng trong hệ thống các sinh viên này phải được *quản lý theo định danh để xác định duy nhất từng đối tượng.* Giá trị có thể là các giá trị của các kiểu dữ liệu nguyên thuỷ như các kiểu số hoặc các kiểu xâu ký tự, hoặc là tập hợp của các giá trị nguyên thuỷ.

Các dữ liệu thành phần của một lớp có thể được bao gói thông qua các thuộc tính quản lý sự truy nhập để phục vụ việc che giấu thông tin của phương pháp hướng đối tượng. Trong UML ta có thể sử dụng các ký hiệu để đặc tả các thuộc tính đó.

Ký hiệu '+' đứng trước tên thuộc tính, hàm xác định tính công khai (*public*), mọi đối tượng trong hệ thống đều nhìn thấy được. Nghĩa là mọi đối tượng đều có thể truy nhập được vào dữ liệu công khai. Trong Rose [8] ký hiệu là ổ khoá không bị khoá.

'#' đứng trước tên thuộc tính, hàm xác định tính được bảo vệ (*protected*), chỉ những đối tượng có quan hệ kế thừa với nhau nhìn thấy được. Trong Rose ký hiệu là ổ khoá bị khoá, nhưng có chìa để bên cạnh.

'-' đứng trước tên thuộc tính, hàm xác định tính sở hữu riêng (*private*), chỉ các đối tượng trong cùng lớp mới nhìn thấy được. Trong Rose ký hiệu là ổ khoá bị khoá và không có chìa để bên cạnh.

Trong trường hợp không sử dụng một trong ba ký hiệu trên thì đó là trường hợp mặc định. Thuộc tính quản lý truy cập mặc định của những hệ thống khác nhau có thể khác nhau, ví dụ trong C++, các thuộc tính mặc định trong lớp được qui định là *private*, còn trong Java lại qui định khác, đó là những thuộc tính rộng hơn *private*. Những thuộc tính trên thiết lập quyền truy cập cho mọi đối tượng trong các lớp, các *gói*, các *hệ thống con* của hệ thống phần mềm [2].

### 2.2.4 Các thao tác và phương thức

*Thao tác* là một hàm hay thủ tục có thể áp dụng (gọi hàm) cho hoặc bởi các đối tượng trong một lớp. Khi nói tới một thao tác là ngầm định nói tới một đối tượng đích để thực hiện thao tác đó. Ví dụ, thao tác (hàm) *hienThi()* của lớp **MonHoc** khi gọi để hiển thị các về sinh viên học một môn học cụ thể như "Lập trình hướng đối tượng" chẳng hạn.

*Một phương thức* là một cách thức cài đặt của một thao tác trong một lớp [5].

Một số thao tác có thể là *đa xạ*, *được nạp chồng*, nghĩa là nó có thể áp dụng cho nhiều lớp khác nhau với những nội dung thực hiện có thể khác nhau, nhưng cùng tên gọi. Ví dụ lớp **ThietBi** có hàm *tinhGia()*. Hàm này có thể nạp chồng, bởi vì có nhiều phương thức (công thức) tính giá bán khác nhau tuỳ thuộc vào từng loại thiết bị. Tất cả các phương thức này đều thực hiện một nhiệm vụ *tinhGia()*, nhưng được cài đặt với nội dung (các đoạn chương trình) khác nhau. Hệ thống hướng đối tượng tự động chọn phương thức tương ứng với ngữ cảnh của đối tượng đích để thực hiện.

Tương tự như các dữ liệu thành phần, các phương thức cũng được quản lý truy cập và được ký hiệu như trên.

*Lưu ý:* Một số tác giả ([2], [4], [6], [9]) không phân biệt thao tác, hàm với phương thức mà có thể đồng nhất chúng với nhau trong quá trình phân tích, thiết kế và lập trình. Trong các phần sau chúng ta gọi chung là hàm hoặc hàm thành phần.

## 2.3 Các mối quan hệ giữa các lớp

Hệ thống hướng đối tượng là tập các đối tượng tương tác với nhau để thực hiện công việc theo yêu cầu. *Quan hệ là kết nối ngữ nghĩa giữa các lớp đối tượng, trong đó thể hiện mối liên quan về các thuộc tính, các thao tác của chúng với nhau trong hệ*

*thống*. Các quan hệ này được thể hiện chính trong biểu đồ lớp. Giữa các lớp có bốn quan hệ cơ bản:

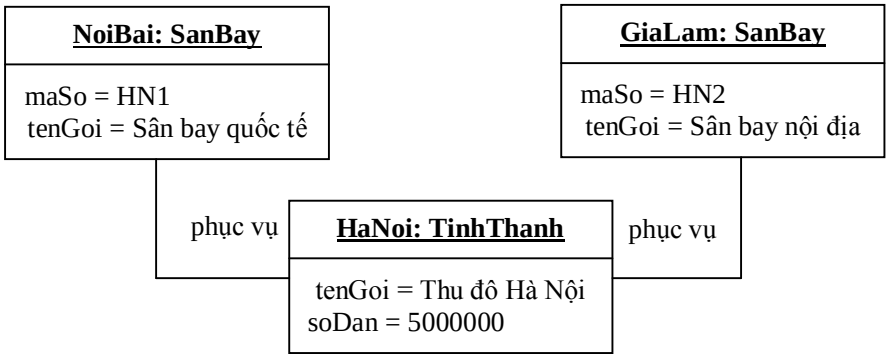
- Quan hệ kết hợp,
- Quan hệ kết tập,
- Quan hệ tổng quát hóa, kế thừa,
- Quan hệ phụ thuộc.

### 2.3.1 Sự liên kết và kết hợp giữa các đối tượng

*Một liên kết là một sự kết nối vật lý hoặc logic giữa các đối tượng với nhau*. Phần lớn các liên kết là sự kết nối giữa hai đối tượng với nhau. Tuy nhiên cũng có những liên kết giữa ba hoặc nhiều hơn ba đối tượng. Nhưng các ngôn ngữ lập trình hiện nay hầu như chỉ cài đặt những liên kết (phép toán) nhiều nhất là ba ngôi.

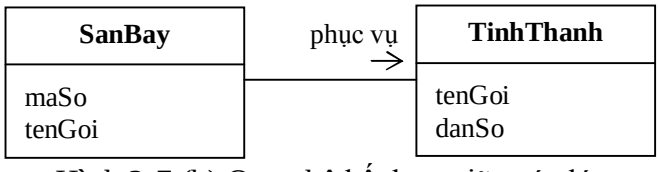
*Một sự kết hợp* là một *mô tả về một nhóm các liên kết có chung cấu trúc và ngữ nghĩa như nhau*. Vậy, *liên kết là một thể hiện của lớp*. Liên kết và kết hợp thường xuất hiện ở dạng các động từ trong các tài liệu mô tả bài toán ứng dụng.

Hình 2-7 mô tả các ký hiệu cho quan hệ liên kết và kết hợp.




Hình 2-7 (a) Liên kết giữa các đối tượng

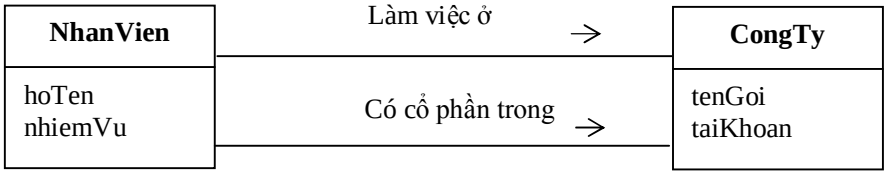
Hai đối tượng thuộc lớp **SanBay**: Nội Bài và Gia Lâm cùng liên kết với đối tượng Hà Nội của lớp **TinhThanh** theo quan hệ *phục vụ*. Quan hệ liên kết giữa hai đối tượng được biểu diễn bằng đoạn thẳng nối chúng với nhau và có tên gọi (nhãn của quan hệ). Nhãn của các quan hệ thường là các động từ. Khi muốn biểu diễn cho quan hệ một chiều thì sử dụng mũi tên để chỉ rõ hướng của quan hệ.




Hình 2-7 (b) Quan hệ kết hợp giữa các lớp

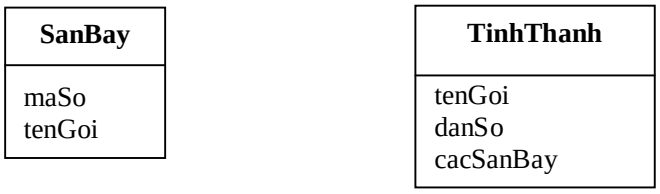
Khi mô hình không có sự nhập nhằng thì tên của kết hợp là tuỳ chọn. Sự nhập nhằng sẽ xuất hiện khi giữa hai lớp có nhiều quan hệ kết hợp, ví dụ: giữa lớp **NhanVien** và lớp **CongTy** có hai quan hệ *làm việc ở* và *có cổ phần trong*. Khi có

nhiều quan hệ như thế thì nên gán tên vào mỗi đường nối hoặc *tên của vai trò* ở mỗi đầu của quan hệ để tránh sự nhập nhằng.




Hình 2-7 (c) Quan hệ kết hợp giữa các lớp

*Lưu ý: không nên nhầm lẫn liên kết với giá trị. Giá trị là dữ liệu nguyên thuỷ như là dữ liệu số hoặc xâu ký tự. Liên kết là mối liên quan giữa hai đối tượng*. Trong giai đoạn phân tích ta phải mô hình (xác định) tất cả các tham chiếu tới các đối tượng thông qua các liên kết và nhận dạng được các nhóm liên kết tương tự thông qua các quan hệ kết hợp. Đến giai đoạn thiết kế ta có thể chọn cấu trúc con trỏ, khóa ngoại, hoặc một số cách khác để cài đặt những quan hệ đó. *Ví dụ*: mô hình phân tích ở hình 2-7 (b) được phát triển sang giai đoạn thiết kế như sau:



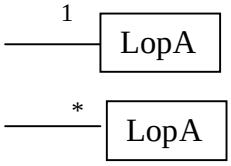

Hình 2-8 Mô hình thiết kế các lớp

Trong đó, lớp **TinhThanh** có thêm thuộc tính *cacSanBay* có thể là danh sách hoặc là cấu trúc mảng, hay con trỏ, v.v. Ta cũng có thể thiết kế theo cách khác, thay vì bổ sung thuộc tính *cacSanBay* vào lớp **TinhThanh** thì bổ sung *oTinhThanh* vào lớp **SanBay**.

### 2.3.2 Bội số

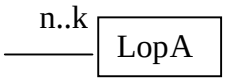
Quan hệ kết hợp thường là quan hệ hai chiều: một đối tượng kết hợp với một số đối tượng của lớp khác và ngược lại. Để xác định số các đối tượng có thể tham gia vào mỗi đầu của mối quan hệ ta có thể sử dụng khái niệm *bội số. Bội số* xác định số lượng các thể hiện (đối tượng) của một lớp trong quan hệ kết hợp với các đối tượng của lớp khác. Cũng cần phân biệt *bội số (hay bản số)* với *lực lượng*. Bội số là ràng buộc về kích cỡ của một tuyển tập, còn lực lượng là đếm số phần tử của tuyển tập đó. Do đó, *bội số là sự ràng buộc về lực lượng của các phần tử trong một lớp tham gia vào quan hệ xác định trước*.

Trong UML các bội số được biểu diễn như sau:

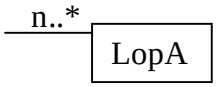



Chính xác 1 (nếu không nhập nhằng có thể không gì cả, được xem mặc định là 1)

Nhiều (0..*)

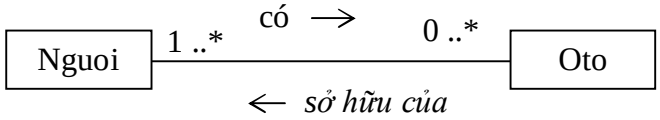


Số lượng được xác định giữa số *n* và *k (≥ 0).*



Số lượng được xác định bởi số *n* cho đến nhiều *(n ≥ 0).*

Để phân biệt chiều của quan hệ kết hợp ta có thể sử dụng mũi tên chỉ chiều kết hợp. Ví dụ:

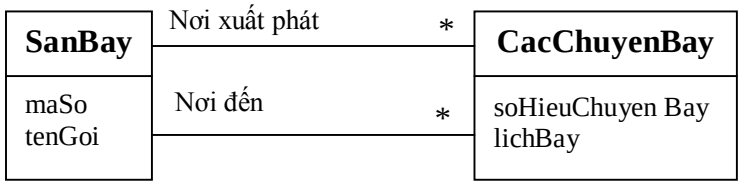



Hình 2-9 Quan hệ kết hợp hai chiều giữa hai lớp

Hình 2-9 mô tả như sau: mỗi người trong lớp **Nguoi** có thể không có hoặc có nhiều ô tô (thuộc lớp **Oto**) và ngược lại một ô tô phải là sở hữu của ít nhất một người nào đó thuộc lớp **Nguoi**.

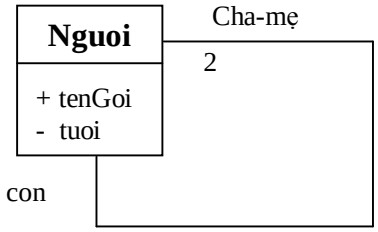
### 2.3.3 Các vai trò trong quan hệ

*Vai trò* là tên gọi một nhiệm vụ thường là một danh từ được gán cho một đầu của quan hệ kết hợp. Hình 2-10 mô tả hai lớp **SanBay** và lớp **CacChuyenBay** có quan hệ kết hợp với nhau. Một sân bay có thể là điểm đến của chuyến bay này và lại có thể là điểm xuất phát của chuyến bay khác. Ngược lại một chuyến bay bao giờ cũng phải bay từ một sân bay này tới một sân bay khác.




Hình 2-10 Vai trò trong các quan hệ kết hợp

Khi mô hình không có sự nhập nhằng thì tên của vai trò là tuỳ chọn. Sự nhập nhằng sẽ xuất hiện khi giữa hai lớp có nhiều quan hệ như hình 2-10, hoặc khi một lớp lại có quan hệ với chính nó (quan hệ đệ qui). Ví dụ: một người có thể là con của hai người (cha-mẹ) và hai cha -mẹ lại có thể có nhiều con. Quan hệ này có thể mô tả như trong hình 2-11.



*

Hình 2-11 Vai trò trong các quan hệ kết hợp

### 2.3.4 Quan hệ kết tập (quan hệ gộp)

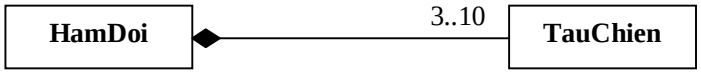
*Kết tập (gộp)* là một loại của quan hệ kết hợp, tập trung *thể hiện quan hệ giữa tổng thể và bộ phận* (*Whole / part*). Kết tập thường biểu diễn cho quan hệ "*có*" (*has-a*), "*là bộ phận của*" (*is-a-part-of*), hoặc "*bao gồm*" (*contains*), v.v. thể hiện mối quan hệ một lớp tổng thể có, gồm, chứa hay liên kết với một hoặc nhiều lớp thành phần. Người ta chia quan hệ kết tập thành ba loại:

- Kết tập thông thường
- Kết tập chia sẻ và
- Kết tập hợp thành hay quan hệ hợp thành.

**Kết tập thông thường**

Quan hệ *kết tập thông thường*, gọi tắt là kết tập thể hiện mối liên kết giữa hai lớp, *trong đó đối tượng của lớp này bao gồm một số đối tượng của lớp kia, song không tồn tại trong nội tại của lớp đó*. Lớp phía bộ phận cũng chỉ là một bộ phận logic của phía tổng thể và chúng không được chia sẻ với các lớp khác. Ví dụ: một hạm đội của lớp **HamDoi** gồm một số (3..10) tàu chiến của lớp **TauChien**, nhưng tàu chiến không chứa trong lớp **HamDoi**. Vậy, lớp **HamDoi** có quan hệ kết tập với **TauChien**. UML sử dụng ký hiệu:

◆— để biểu diễn quan hệ kết tập và luôn được gắn với phía tổng thể. Hình 2-12 thể hiện quan hệ giữa lớp **HamDoi** và lớp **TauChien**.



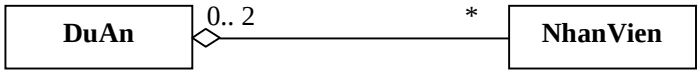

Hình 2-12 Quan hệ kết tập thông thường

Trong quan hệ này, việc quản lý các đối tượng của các lớp liên quan là khác nhau. Ta có thể loại bỏ một số *tàu chiến* của một *hạm đội* sao cho số còn lại ít nhất là 3, tương tự có thể bổ sung vào một số *tàu chiến* sao cho không quá 10. Nhưng khi đã loại bỏ một *hạm đội* thì phải loại bỏ tất cả các *tàu chiến* của *hạm đội* đó vì mỗi *tàu chiến* chỉ thuộc một *hạm đội*. Nói một cách khác, một đối tượng của lớp phía bộ phận sẽ không thể tồn tại độc lập nếu nó không phải là một phần của phái tổng thể.

**Kết tập chia sẻ**

Quan hệ *kết tập chia sẻ* là loại kết tập, trong đó phía bộ phận có thể tham gia vào nhiều phía tổng thể. Ví dụ: một *dự án* của lớp **DuAn** có nhiều *nhân viên* của lớp **NhanVien** tham gia và một nhân viên có thể tham gia vào nhiều (hai) dự án. UML sử dụng ký hiệu:


để biểu diễn quan hệ kết tập chia sẻ và luôn được gắn với phía tổng thể. Hình 2-13 thể hiện quan hệ giữa lớp **DuAn** và lớp **NhanVien**.

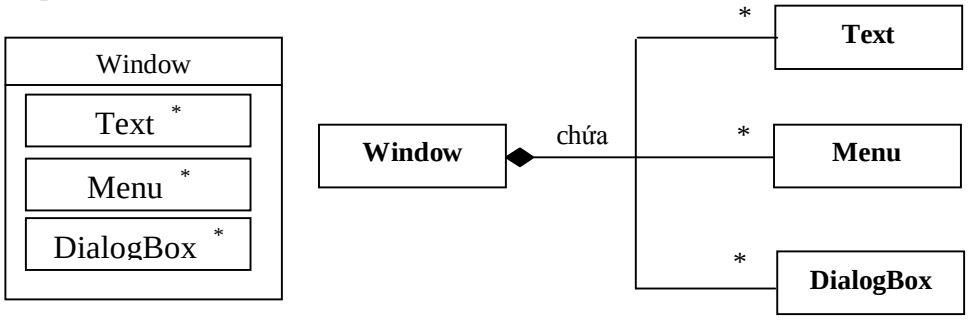



Hình 2-13 Quan hệ kết tập thông thường

Mỗi dự án có thể có nhiều người tham gia và mỗi người lại có thể tham gia nhiều nhất là hai dự án. Trong quan hệ này, ta có thể loại bỏ, hay thành lập một dự án (phía tổng thể) nhưng không nhất thiết phải loại bỏ, hay phải tuyển thêm những người tham gia (phía bộ phận) vào dự án như kiểu kết tập ở trên. Tuy nhiên khi xử lý các mối quan hệ đó thì phải cập nhật lại các mối liên kết của các nhân viên tham gia vào các dự án tương ứng.

**Kết tập hợp thành**

*Quan hệ chỉ ra một vật có chứa một số bộ phận và các bộ phận đó tồn tại vật lý bên trong vật tổng thể.* Do vậy khi thực hiện huỷ bỏ, hay thiết lập mới bên tổng thể thì các bộ phận bên thành phần cũng sẽ bị huỷ bỏ hoặc phải được bổ sung. Ví dụ: lớp **Window** chứa các lớp **Text, Menu** và **DialogBox**. Trong UML có hai cách biểu diễn quan hệ hợp thành như sau:




Hình 2-14 Quan hệ kết tập hợp thành

### 2.3.5 Quan hệ tổng quát hoá

*Tổng quát hoá* và *chuyên biệt hoá* là hai cách nhìn dưới / lên (*buttom–up*) và trên/xuống (*top-down*) về sự phân cấp các lớp, mô tả khả năng quản lý cấp độ phức tạp của hệ thống bằng cách trừu tượng hoá các lớp.

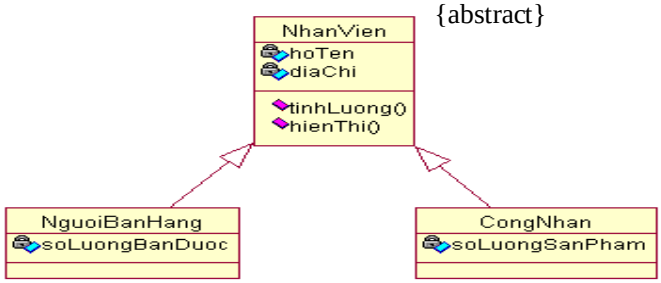
*Tổng quát hoá là đi từ các lớp dưới lên* sau đó hình thành lớp tổng quát (lớp trên, lớp cha), tức là cây cấu trúc các lớp từ lá đến gốc.

*Chuyên biệt hoá là quá trình ngược lại của tổng quát hoá,* nó cho phép tạo ra các lớp dưới (lớp con) khác nhau của lớp cha.

Trong UML, tổng quát hoá chính là quan hệ kế thừa giữa hai lớp. Nó cho phép lớp con (lớp dưới, lớp kế thừa, hay lớp dẫn xuất) kế thừa trực tiếp các thuộc tính và các hàm thuộc loại công khai, hay được bảo vệ (*protected*) của lớp cha (lớp cơ sở, lớp trên). Trong quan hệ tổng quát hoá có hai loại lớp: *lớp cụ thể* và *lớp trừu tượng*.

*Lớp cụ thể (Concrete Class)* là lớp có các đại diện, các thể hiện cụ thể. Ngược lại, *lớp trừu tượng (Abstract Class)* là lớp không có thể hiện (đối tượng) cụ thể trong hệ thống thực. Các lớp con cháu của lớp trừu tượng có thể là lớp trừu tượng, tuy nhiên trong cấu trúc phân cấp theo quan hệ tổng quát hoá thì mọi nhánh phải kết thúc (lớp lá) bằng các lớp cụ thể. Ta có thể định nghĩa các hàm trừu tượng cho các lớp trừu tượng, đó là những hàm chưa được cài đặt nội dung thực hiện trong lớp chúng được khai báo. Những hàm trừu tượng này sẽ được cài đặt trong các lớp con cháu sau đó ở những lớp cụ thể.

*Ví dụ*: Lớp **NhanVien** có ký hiệu {*abstract*} sau hoặc dưới tên lớp là lớp trừu tượng, và do vậy nó không có đối tượng cụ thể. Hai lớp con: lớp **NguoiBanHang** và lớp **CongNhan** là hai lớp cụ thể. Hai lớp này có những thuộc tính, thao tác giống lớp **NhanVien** như có các thuộc tính: *hoTen, diaChi* và có các hàm *tinhLuong(), hienThi()*, ngoài ra mỗi lớp còn có thể bổ sung thêm một số thuộc tính, thao tác để đặc tả cho từng nhóm đối tượng cụ thể. Lớp **NguoiBanHang** được bổ sung thêm thuộc tính *soluongBanDuoc* còn lớp **CongNhan** được bổ sung thuộc tính *soLuongSanPham* sản xuất được. Cấu trúc phân cấp của lớp **NhanVien** được xác định như hình 2-15.

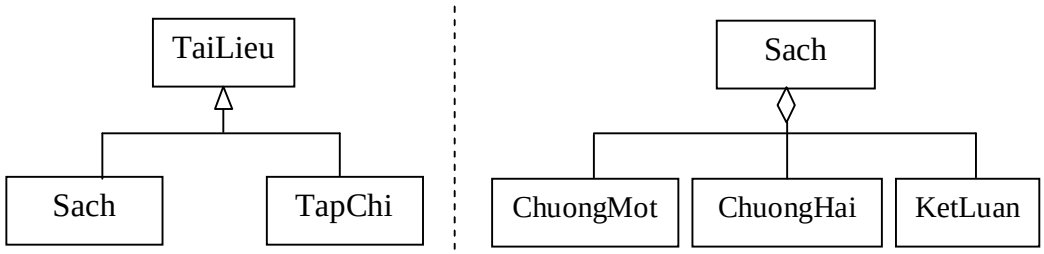



Hình 2-15 Lớp trừu tượng và cụ thể trong quan hệ tổng quát hoá

*Lưu ý:*

- Quan hệ tổng quát và kết hợp là hai quan hệ liên quan đến hai lớp, nhưng chúng có những điểm khác nhau. Quan hệ kết hợp mô tả mối liên kết giữa hai hoặc nhiều hơn đối tượng còn quan hệ khái quát mô tả các phương diện khác nhau của cùng một thể hiện.
- Trong giai đoạn phân tích, các quan hệ kết hợp là quan trọng hơn quan hệ tổng quát hoá. Kết hợp bổ sung thêm các thông tin cho các lớp. Ngược lại, tổng quát hoá là loại bỏ những thông tin bị chia sẻ ở các lớp con cháu vì chúng được kế thừa từ lớp cha.
- Trong giai đoạn thiết kế thì tổng quát hoá lại quan trọng hơn. Người phát triển hệ thống quan tâm để phát hiện ra những cấu trúc dữ liệu ở khâu phân tích và phát hiện ra các hành vi ở khâu thiết kế. Tổng quát hoá cung cấp cơ chế sử dụng lại để thể hiện chính xác các hành vi và mã hoá của các thư viện của các lớp.
- Quan hệ kết tập và tổng quát cũng khác nhau. Cả hai đều làm xuất hiện cấu trúc cây thông qua bao đóng bắc cầu của quan hệ cơ sở, nhưng quan hệ tổng

quát là mối quan hệ "hoặc" (OR) còn quan hệ kết tập là mối quan hệ "và" (AND). Hình 2-16 mô tả sự khác nhau của quan hệ tổng quát hoá và kết tập.



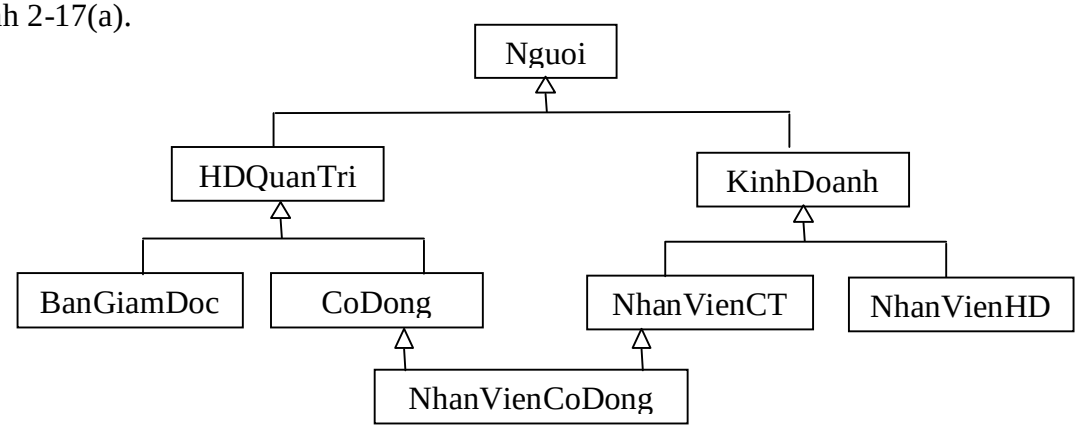

Hình 2-16 Quan hệ tổng quát hoá ngược lại với quan hệ kết tập

### 2.3.6 Kế thừa bội

*Kế thừa bội* cho phép một lớp được kế thừa các thuộc tính, các thao tác và các quan hệ kết hợp từ nhiều lớp cơ sở. Trong quan hệ kế thừa bội có thể dẫn đến sự pha trộn thông tin từ nhiều nguồn dữ liệu khác nhau từ các lớp được kế thừa. *Quan hệ kế thừa đơn*, một lớp được kế thừa từ một lớp trên, thường tạo ra cấu trúc cây, còn *kế thừa bội lại tổ chức các lớp thành đồ thị định hướng phi chu trình.* Kế thừa bội là cơ chế mạnh trong mô hình hệ thống, nhưng đồng thời cũng sẽ tạo ra nhiều phức tạp về tính nhập nhằng, không nhất quán dữ liệu.

**Kế thừa bội từ những lớp phân biệt**

Một lớp có thể kế thừa từ nhiều lớp cơ sở khác nhau. Ví dụ lớp **Nguoi** là cơ sở để tạo ra hai lớp con: **HDQuanTri** (những người trong hội đồng quản trị) và **KinhDoanh** (những người kinh doanh). Từ các lớp trên lại xây dựng các lớp **BanGiamDoc, CoDong** (những người đóng cổ phần) kế thừa từ lớp **HDQuanTri,** lớp **NhanVienCT** (những người làm việc thường xuyên trong công ty) và **NhanVienHD** (những người làm việc theo hợp đồng) kế thừa từ lớp **KinhDoanh.** Trong Công ty lại có những người vừa là cổ đông, vừa là nhân viên. Những người đó chính là các đối tượng của lớp **NhanVienCoDong** kế thừa từ hai lớp con **CoDong** và **NhanVienCongTy** như hình 2-17(a).

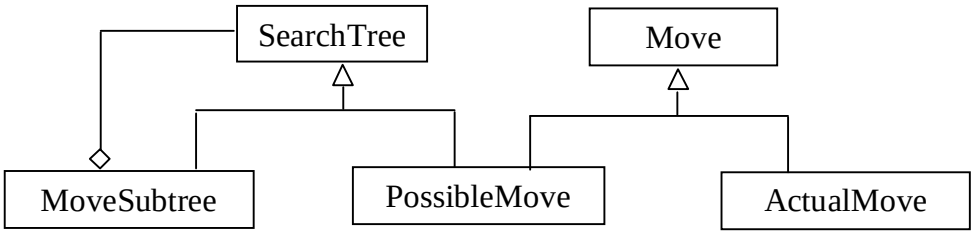


Hình 2-17(a) Kế thừa bội từ hai lớp khác nhau và có lớp cơ sở chung

**Kế thừa bội không có lớp cơ sở chung**

Kế thừa bội như trên là kế thừa có lớp cơ sở chung (lớp **Nguoi**). Chúng ta có thể tạo ra lớp kế thừa bội từ nhiều lớp mà chúng lại không có lớp cơ sở chung. Loại kế thừa này thường xuất hiện khi ta muốn pha trộn một số chức năng của các lớp thư viện khác nhau.

*Ví dụ:* chúng ta hãy xét mô hình của chương trình đánh cờ. Trước khi đi một nước cờ, chương trình phải nghiên cứu vị trí của các quân cờ và các nước đi tiếp theo sao cho nước đi đó là có thể dẫn đến chiến thắng nhanh nhất. Trong hình 2-13 (b), mỗi đối tượng của lớp **SearchTree** (cây tìm kiếm) có thể là đối tượng của lớp **MoveSubtree** (cây con các nước đi) hoặc của lớp **PossibleMove** (lớp các nước có thể chọn)**.** Bản thân lớp **MoveSubtree** lại có thể chứa các **SearchTree** nhỏ hơn. Mỗi nước đi của lớp **Move** lại có thể là nước đi có thể (**PossibleMove)** hoặc lớp các nước đi hiện thời (**ActualMove**). Lớp **PossibleMove** kế thừa hành vi chung của lớp **Move** và **lớp SearchTree.**




Hình 2-17(b) Kế thừa bội không có lớp có sở chung

### 2.3.7 Quan hệ phụ thuộc

*Sự phụ thuộc* là một loại quan hệ liên kết giữa hai phần tử trong mô hình, trong đó thể hiện sự thay đổi trong một phần tử sẽ kéo theo sự thay đổi của phần tử kia. Quan hệ kết hợp thường là quan hệ hai chiều, nhưng quan hệ phụ thuộc lại thường là quan hệ một chiều, thể hiện một lớp phụ thuộc vào lớp khác. Lớp A phụ thuộc vào lớp B khi:

- Lớp A sử dụng một đối tượng của lớp B như là tham số trong các thao tác,
- Trong các thao tác của lớp A có truy nhập tới các đối tượng của lớp B,
- Khi thực hiện một thao tác nào đó trong lớp A lại phải tham chiếu tới miền xác định của lớp B.
- Lớp A sử dụng các *giao diện* của lớp B.

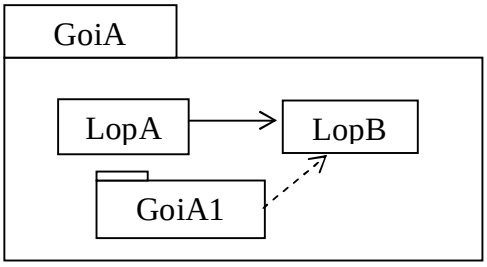
Tương tự, hai *gói (package)* có thể phụ thuộc vào nhau khi có một lớp ở một gói phụ thuộc vào lớp của gói kia.

Trong UML, quan hệ phụ thuộc được thể hiện bằng mũi tên đứt nét. Ví dụ, hình 2-18 mô tả quan hệ phụ thuộc giữa hai lớp và phụ thuộc của hai gói.

Hình 2-18 Quan hệ phụ thuộc giữa các lớp và các gói

## 2.4 Các gói

Để hiểu được những hệ thống lớn, phức tạp có nhiều lớp đối tượng, thì chúng ta phải có cách chia các lớp đó thành một số nhóm được gọi là gói. *Gói* là một nhóm các phần tử của mô hình *gồm các lớp, các mối quan hệ và các gói nhỏ hơn*. Cách tổ chức hệ thống thành các gói (hệ thống con) chính là cách phân hoạch mô hình thành các đơn vị nhỏ hơn để trị dễ hiểu và dễ quản lý hơn. Gói được mô tả trong UML gồm tên của gói, có thể có các lớp, gói nhỏ khác và được ký hiệu như hình 2-19.

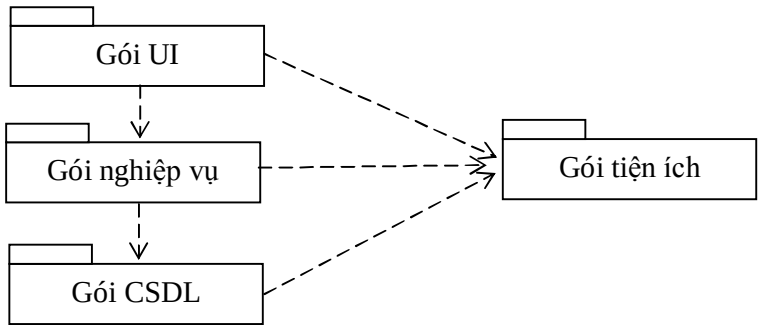



Hình 2-19 Gói các lớp trong UML

Khi phân chia các lớp thành các gói, chúng ta có thể dựa vào: các lớp chính (thống trị), các mối quan hệ chính, các chức năng chính. Theo cách phân chia đó chúng ta có thể chia hệ thống thành các phân hệ phù hợp với cách phân chia trong hệ thống thực. Ví dụ, hệ thống quản lý thư viện có thể tổ chức thành bốn gói: *gói giao diện, gói nghiệp vụ, gói CSDL và gói tiện ích* như hình 2-20. Trong đó,

- *Gói giao diện (UI – User Interface):* bao gồm các lớp giao diện với người dùng, cho các khả năng quan sát và truy nhập vào dữ liệu. Các đối tượng trong gói này có thể thực hiện các thao tác trên các đối tượng tác nghiệp để truy vấn hay nhập dữ liệu.
- *Gói nghiệp vụ (Business Package):* chứa các lớp thực thể thuộc lĩnh vực bài toán ứng dụng.
- *Gói CSDL:* chứa các lớp dịch vị cho các lớp ở gói tác nghiệp trong việc tổ chức, quản lý và lưu trữ dữ liệu.
- *Gói tiện ích:* chứa các lớp dịch vụ cho các gói khác của hệ thống.

Các gói của một hệ thống thường có mối quan hệ với nhau, như quan hệ phụ thuộc.



Hình 2-20 Tổ chức các gói của hệ thống thư viện

## 2.5 Các qui tắc ràng buộc và suy diễn

Trong mô hình hoá hệ thống với UML, ta có thể sử dụng *ngôn ngữ ràng buộc đối tượng OCL (Object Constraints Language)* [4] để đặc tả chính xác các phần tử của hệ thống và các ràng buộc chặt chẽ giữa các mối quan hệ, giới hạn phạm vi của mô hình hệ thống cho phù hợp với điều kiện ràng buộc thực tế.

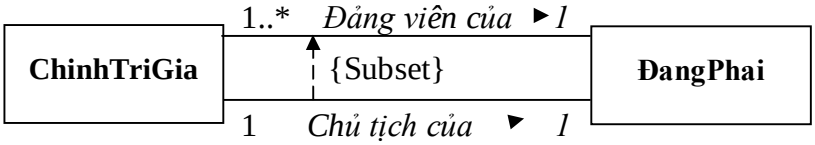
Trong UML có hai qui tắc chính:

1. *Qui tắc ràng buộc* (*Constraint Rule)* được sử dụng để giới hạn phạm vi của mô hình, ví dụ các qui tắc hạn chế, qui định rõ phạm trù của các mối quan hệ như kết hợp, kế thừa hay khả năng nạp chồng trong các lớp.
2. *Qui tắc suy dẫn (Derivation Rule)* chỉ ra cách các sự vật có thể suy dẫn được, ví dụ *tuổi* của một người có thể suy ra được từ *ngày / tháng / năm* hiện thời trừ đi *ngày / tháng / năm sinh*.

*Lưu ý:* Các qui tắc ràng buộc và suy dẫn thường được đặt trong cặp dấu ngoặc '{' và '}' ở bên cạnh những phần tử của mô hình, thường là các thuộc tính, hay các mối quan hệ cần phải tuân theo.

*Ví dụ:*

*1/* Khi mô tả mối quan hệ giữa hai lớp **DangPhai** và **ChinhTriGia**, ta có thể sử dụng qui tắc ràng buộc để khống chế các đối tượng tham gia vào các quan hệ đó. Ví dụ, trong các đảng phái chính trị có qui định rằng lãnh tụ của một đảng phải là đảng viên của chính đảng đó. Khi đó quan hệ *"Chủ tịch của" một đảng* phải là tập con {*Subset*} của quan hệ "*đảng viên của*" *đảng đó* và được mô tả trong UML như hình 2-21 (a).



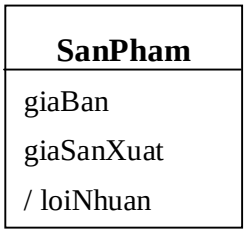

Hình 2-21 (a) Mối ràng buộc giữa hai quan hệ

2/ Các thuộc tính có thể bị khống chế, bị giới hạn trong phạm vi xác định, ví dụ: điều kiện

{0 ≤ mau ≤ 255}

chỉ ra rằng thuộc tính *mau (màu)* có giá trị trong phạm vi các số nguyên từ 0 đến 255.

3/ Một số thuộc tính có thể được suy dẫn từ những thuộc tính khác. Ví dụ khi thiết kế lớp **SanPham** có thuộc tính *giaBan* và *giaSanXuat.* Trong kinh doanh ta có thể xác định được ngay cách tính lợi nhuận *loiNhuan = giaBan – giaSanXuat.* Cách tính và những qui định trên có thể mô tả như hình 2-21 (b).




Hình 2-21 (b) Qui tắc suy dẫn trong OCL

Trong hình trên, ký hiệu "/" được sử dụng để chỉ ra rằng thuộc tính *loiNhuan* là được suy dẫn ra theo qui tắc được gắn bên cạnh của lớp **SanPham.**

*Lưu ý:*

- ✓ Quan hệ tổng quát hoá chỉ áp dụng với các qui tắc hạn chế (bị ràng buộc) chứ không áp dụng được với qui tắc suy dẫn, nghĩa là có thể được nạp chồng, rời nhau, tổng quát hoá toàn bộ hay một phần.
- ✓ Các qui tắc hạn chế có thể viết dưới dạng các biểu thức với toán tử '.' (toán tử xác định thành phần) như trong các ngôn ngữ lập trình hướng đối tượng. Ví dụ:

HopDongBaoHiem.soNguoiMuaBH > 0
Oto.NguoiLai.bangLaiXe = True

## 2.6 Quá trình phát triển phần mềm

Phần mềm *là một sản phẩm* được phát triển hay được kỹ nghệ hoá và *được chế tạo tương tự như các sản phẩm công nghiệp (phần cứng) khác.* Phát triển phần mềm và chế tạo phần cứng cũng có những điểm tương đồng: đều là sản phẩm và chất lượng của chúng phụ thuộc nhiều vào thiết kế, hơn nữa lại phụ thuộc cơ bản vào con người.

Tuy nhiên phần mềm và phần cứng lại có nhiều điểm đặc trưng rất khác nhau.

- Qui trình và phương pháp tổ chức thực hiện để sản xuất ra chúng rất khác nhau,
- Các giai đoạn chế tạo ra phần cứng có thể xác định và có khả năng điều chỉnh được chất lượng của sản phẩm còn đối với phần mềm thì không dễ gì thay đổi được,
- Mối quan hệ giữa người sử dụng và công việc cần thực hiện với từng loại sản phẩm là hoàn toàn khác nhau,
- Cách tiếp cận để xây dựng, chế tạo các sản phẩm cũng khác nhau. Khả năng nhân bản của chúng là hoàn toàn trái ngược nhau. Việc bảo vệ bản quyền sản phẩm phần mềm là cực kỳ khó khăn vì khả năng sao chép thành nhiều bản giống nhau là có thể thực hiện được (tương đối dễ).
- Phần mềm khác hẳn với phần cứng là không bị hư hỏng theo thời gian, không bị tác động của môi trường (thời tiết, nhiệt độ, điều kiện, v.v…). Do vậy, đối với phần cứng việc bảo hành là đảm bảo nó hoạt động được như mới còn đối với phần mềm thì lại khác hẳn. Bảo hành, bảo trì phần mềm là bảo đảm cho hệ thống hoạt động đúng với thiết kế và đáp ứng yêu cầu sử dụng của khách

hàng. Chính vì thế mà công bảo hành phần mềm là rất tốn kém và đòi hỏi phải tập trung nhiều hơn vào khâu phân tích, thiết kế hệ thống.

Mọi hệ thống phần mềm cũng như các hệ thống khác không thể tồn tại độc lập mà nó luôn hoạt động và tồn tại trong một môi trường, tương tác với thế giới xung quanh. Như vậy một hệ thống có thể xem như là hệ thống con của một hệ thống khác và bản thân nó lại có thể chứa một số các hệ thống con khác nhỏ hơn.

Công nghệ phần cứng phát triển nhanh cả về chất lượng và tốc độ xử lý với giá thành ngày một hạ trong khi giá phần mềm lại rất cao. Để phát triển được những hệ thống phần mềm đáp ứng được những yêu cầu trên thì đòi hỏi phải áp dụng *lý thuyết, kỹ nghệ, phương pháp và công cụ mới để tạo ra một qui trình phát triển phần mềm thống nhất*. Như vậy, công nghệ phần mềm (CNPM) là đề cập đến các lý thuyết, phương pháp luận và các công cụ cần thiết để phát triển phần mềm. Mục đích của CNPM là làm ra những phần mềm chất lượng cao, tin cậy với một hạn chế về nguồn lực, theo đúng một lịch trình đặt trước, phù hợp với ngân sách dự kiến và đáp ứng các yêu cầu người dùng. Hơn nữa, CNPM không chỉ là phải làm ra hệ thống phần mềm mà phải làm được các hồ sơ, tài liệu như các tài liệu thiết kế, tài liệu hướng dẫn sử dụng, v.v. làm cơ sở để bảo trì và mở rộng, phát triển hệ thống sau này.

*Tóm lại để xây dựng được những hệ thống phần mềm đáp ứng những yêu cầu trên, chúng ta cần phải:*

- Có một qui trình phát triển phần mềm thống nhất gồm các bước thực hiện rõ ràng, mà sau mỗi bước đều phải có các sản phẩm cụ thể;
- Có các phương pháp và kỹ nghệ phù hợp với từng pha thực hiện phát triển phần mềm;
- Có công cụ để làm ra sản phẩm phần mềm theo yêu cầu.

*Quá trình phát triển một sản phẩm (Product Development Process)* là quá trình định nghĩa ai làm cái gì, làm khi nào và như thế nào. Quá trình xây dựng một sản phẩm phần mềm hoặc nâng cấp một sản phẩm đã có được gọi là *quá trình phát triển phần mềm* (*Software Development Process*).

Hệ thống phần mềm được kiến tạo là sản phẩm của một loạt các hoạt động sáng tạo và có quá trình phát triển. Quá trình phát triển những phần mềm phức tạp phải có nhiều người tham gia thực hiện. Trước hết đó là *khách hàng* và những *nhà đầu tư*, đó là những người đưa ra vấn đề cần giải quyết trên máy tính. Những *người phát triển* hệ thống gồm các *nhà phân tích, thiết kế và lập trình* làm nhiệm vụ phân tích các yêu cầu của khách hàng, thiết kế các thành phần của hệ thống và thực thi cài đặt chúng. Sau đó phần mềm được kiểm tra (*Testing*) và triển khai ứng dụng để thực thi những vấn đề mà người sử dụng yêu cầu.

*Quá trình phát triển phần mềm* được xác định thông qua tập các hoạt động cần thực hiện để chuyển đổi các yêu cầu của khách hàng (người sử dụng) thành hệ thống phần mềm. Có năm bước chính cần thực hiện trong quá trình phát triển phần mềm:

1. Xác định các yêu cầu
2. Phân tích hệ thống

3. Thiết kế hệ thống
4. Lập trình và kiểm tra hệ thống
5. Vận hành và bảo trì hệ thống.

Có thể thực hiện các bước trên theo nhiều phương pháp khác nhau. Theo đó, số các bước đề xuất của các phương pháp cũng có thể khác nhau. Các dự án có thể thực hiện theo những mô hình khác nhau (([3], [12]) như mô hình "thác nước" (waterfall), mô hình "tạo nguyên mẫu" (Prototype), mô hình "xoắn ốc", v.v. tuỳ thuộc vào từng dự án khác nhau.

Như ở chương 1 đã phân tích, có hai cách tiếp cận chính để thực hiện các bước nêu trên: cách tiếp cận hướng chức năng và hướng đối tượng. Ở đây chúng ta tập trung nghiên cứu các *phương pháp phân tích và thiết kế hướng đối tượng*.

### 2.6.1 Xác định các yêu cầu và phân tích hệ thống

**Phân tích các yêu cầu của hệ thống**

Từ các yêu cầu của khách hàng chúng ta xác định được các mục tiêu của phần mềm cần phát triển. Thường đó là các yêu cầu chức năng về *những gì mà hệ thống phải thực hiện*, nhưng chưa cần chỉ ra các chức năng đó thực hiện như thế nào. Việc xác định đúng và đầy đủ các yêu cầu của bài toán là nhiệm vụ rất quan trọng, nó làm cơ sở cho tất cả các bước tiếp trong dự án phần mềm. Muốn vậy, thì phải đặc tả được các yêu cầu của hệ thống.

**UML cung cấp biểu đồ ca sử dụng để đặc tả các yêu cầu của hệ thống**

Tài liệu đặc tả các yêu cầu được sử dụng để:

- Làm cơ sở để *trao đổi với người sử dụng, để thảo luận giữa các nhóm thành viên trong dự án phát triển phần mềm về những gì mà hệ thống sẽ phải thực hiện (và những gì nó không cần thực hiện).*
- *Làm căn cứ c*ơ bản cho các bước tiếp theo trong quá trình phát triển phần mềm.

Muốn đạt được các mục tiêu trên thì quá trình phải thực hiện:

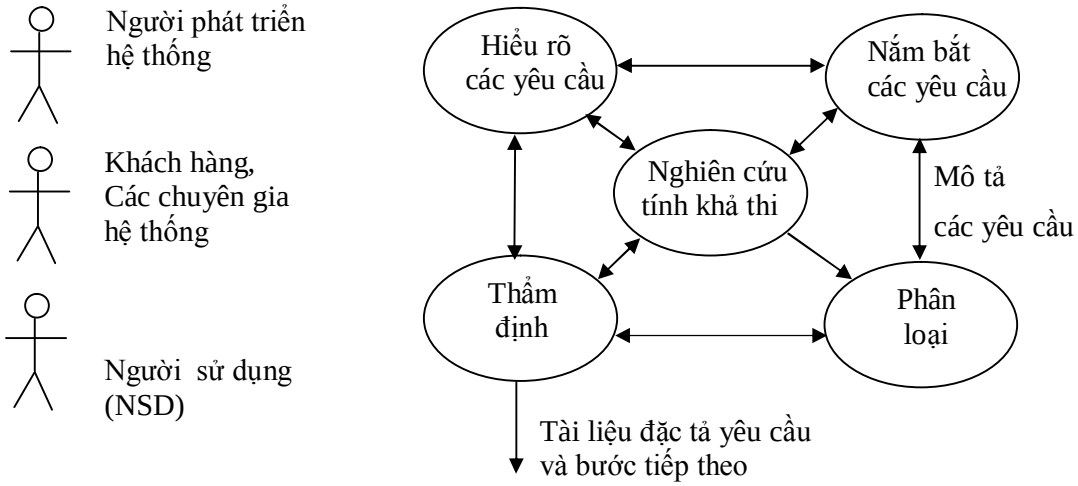
- *Hiể*u rõ miền xác định của bài toán (*Domain Understanding*): Những người phát triển sẽ xây dựng hệ thống theo sự hiểu biết của họ như thế nào về những yêu cầu của khách hàng và những khái niệm cơ sở của bài toán ứng dụng.
- Nắm bắ*t các yêu cầu (Requirement Capture):* Người phân tích phải nắm bắt được tất cả các nhu cầu của khách hàng bằng cách phải trao đổi với mọi người có liên quan đến dự án, tham khảo các tài liệu liên quan. Thông qua việc thảo luận, trao đổi với khách hàng, các chuyên gia của lĩnh vực ứng dụng và những người đã, đang sử dụng những hệ thống có sẵn ta có thể phát hiện và nắm bắt được các yêu cầu của họ. Phương pháp trừu tượng hoá giúp ta dễ dàng nắm bắt được các yêu cầu của hệ thống.
- *Phân loại (*Classification): Vấn đề quan trọng nhất trong giai đoạn này là phải hiểu rõ các yêu cầu đã được xác định. Muốn vậy, ta phải tìm cách phân loại chúng theo tầm quan trọng, hay chức năng chính của những người sử dụng và của khách hàng.

- *Thẩm định* (Validation): Kiểm tra xem các yêu cầu có thống nhất với nhau và đầy đủ không, đồng thời tìm cách giải quyết các mối mâu thuẫn giữa các yêu cầu nếu có.
- Nghiên *cứu tính khả thi (Feasibility Study)*: Tính khả thi của một dự án tin học phải được thực hiện dựa trên các yếu tố bao gồm các khía cạnh tài chính, chiến lược, thị trường, con người, đối tác, kỹ thuật, công nghệ và phương pháp mô hình hoá, v.v.

Nói chung không có các qui tắc hướng dẫn cụ thể để biết khi nào công việc phân tích các yêu cầu sẽ kết thúc và quá trình phát triển có thể chuyển sang bước tiếp theo. Nhưng có thể dựa vào các câu trả lời cho những câu hỏi sau để chuyển sang bước tiếp theo.

- Khách hàng, người sử dụng (NSD) và những người phát triển đã hiểu hoàn toàn hệ thống chưa?
- Đã nêu được đầy đủ các yêu cầu về *chức năng* (dịch vụ), đầu *vào / ra và những dữ liệu cần thiết chưa?*

Bức tranh chung trong pha phân tích các yêu cầu của hệ thống có thể mô tả như trong hình 2-22.




Hình 2-22 Mối quan hệ giữa các công việc trong pha phân tích các yêu cầu

Vấn đề xác định đúng và đầy đủ các yêu cầu của hệ thống là rất quan trọng, nó ảnh hưởng rất lớn tới chất lượng của sản phẩn sau này. Theo *Finkelstein* (1989) khi khảo sát, đánh giá về các pha thực hiện trong quá trình phát triển phần mềm thì trên 55% [10] các lỗi của phần mềm là do các yêu cầu của hệ thống chưa được phát hiện đầy đủ và chi phí cho việc sửa các lỗi đó ( để bảo trì hệ thống) chiếm tới trên 80% chi phí của cả dự án.

### 2.6.2 Phân tích hệ thống hướng đối tượng

*Phân tích hướng đối tượng (Object Oriented Analysis - OOA)*: là một giai đoạn của quá trình phát triển phần mềm, trong đó mô hình khái niệm được mô tả chính xác, súc tích thông qua các đối tượng thực và các khái niệm của bài toán ứng dụng.

Phân tích hướng đối tượng vừa là ngành khoa học vừa là nghệ thuật nên để xây dựng được những *hệ thống tốt, tráng kiện (Robust), có tính mở và dễ bảo trì* thì ta phải biết vận dụng các nguyên lý, phương pháp khoa học kết hợp được cả *heuristic* và các mẫu thử nghiệm (*Patterns*) để nhanh chóng tích luỹ và hoàn thiện kỹ năng phân tích, thiết kế của mình. Thông qua việc áp dụng các nguyên lý và kinh nghiệm thực hiện theo mẫu hướng dẫn [4] chúng ta nhanh chóng học được cách tạo ra các thiết kế hệ thống phần mềm tốt hơn.

Phân tích hướng đối tượng tập trung vào việc tìm kiếm các đối tượng, khái niệm trong lĩnh vực bài toán và xác định mối quan hệ của chúng trong hệ thống.

Nhiệm vụ của người phân tích là nghiên cứu kỹ các yêu cầu của hệ thống và phân tích các thành phần của hệ thống cùng các mối quan hệ của chúng. Trong khâu phân tích hệ thống chủ yếu trả lời câu hỏi:

- Hệ thống gồm những thành phần, bộ phận nào?
- Hệ thống cần thực hiện những cái gì?

Kết quả chính của pha phân tích hệ thống hướng đối tượng là *biểu đồ lớp, biểu đồ trạng thái, biểu đồ trình tự, biểu đồ cộng tác và biểu đồ thành phần***.**

### 2.6.3 Thiết kế hệ thống hướng đối tượng

Dựa vào các đặc tả yêu cầu và các kết quả phân tích (các biểu đồ nêu trên) để thiết kế hệ thống.

*Thiết kế hướng đối tượng (Object Oriented Design – OOD)* là một giai đoạn trong quá trình phát triển phần mềm, trong đó hệ thống được tổ chức thành tập các đối tượng tương tác với nhau và mô tả được cách để hệ thống thực thi nhiệm vụ của bài toán ứng dụng.

Trong khâu thiết kế hệ thống hướng đối tượng chủ yếu trả lời câu hỏi *làm như thế nào*:

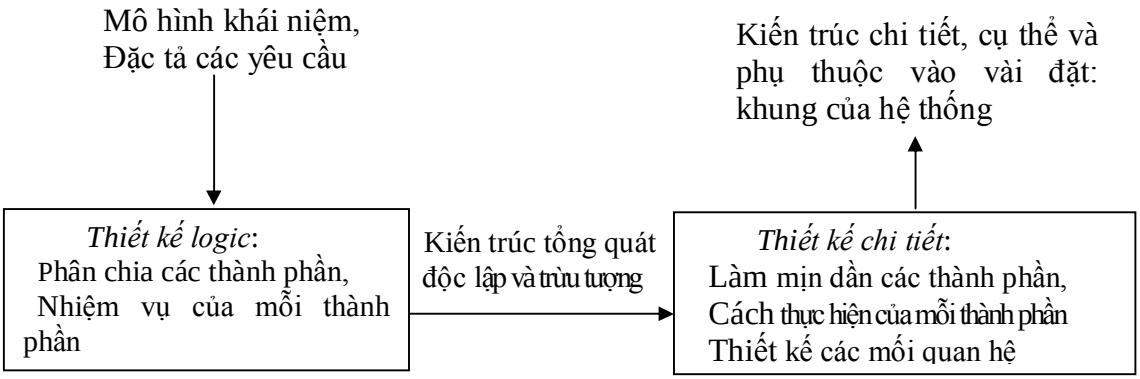
- Trong hệ thống có những lớp đối tượng nào, trách nhiệm của chúng là gì?
- Các đối tượng tương tác với nhau như thế nào?
- Các nhiệm vụ mà mỗi lớp đối tượng phải thực hiện?
- Dữ liệu nghiệp vụ và các giao diện được xây dựng như thế nào?
- Kiến trúc và cấu hình của hệ thống?

Nhiệm vụ chính của thiết kế hệ thống là:

- Xây dựng các thiết kế chi tiết mô tả các thành phần của hệ thống ở mức cao hơn (khâu phân tích) để phục vụ cho việc cài đặt. Nghĩa là, các lớp đối tượng được định nghĩa chi tiết gồm đầy đủ các thuộc tính, các thao tác phục vụ cho việc cài đặt bằng ngôn ngữ lập trình hướng đối tượng được lựa chọn ở các bước sau.

- Đồng thời đưa ra được kiến trúc (là trọng tâm) của hệ thống để đảm bảo cho hệ thống có thể thay đổi, có tính mở, dễ bảo trì, thân thiện với NSD, v.v. Nghĩa là tổ chức các lớp thành các gói hoặc các hệ thống con theo một kiến trúc phù hợp với nhu cầu phát triển của công nghệ (mạng, phân tán, v.v.) đồng thời phù hợp với xu thế phát triển của lĩnh vực ứng dụng.

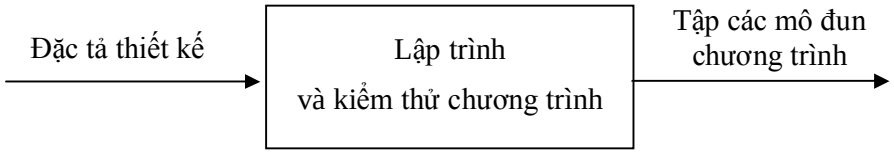
Những kết quả trên được thể hiện trong các biểu đồ: **biểu đồ lớp** (chi tiết), **biểu đồ hành động, biểu đồ thành phần** và **biểu đồ triển khai.** Tất cả các kết quả thiết kế phải được *ghi lại thành các hồ sơ, tài liệu cho hệ thống*. Trong các tài liệu thiết kế phải mô tả cụ thể những *thành phần nào, làm những gì và làm như thế nào*.




Hình 2-23 Thiết kế logic và thiết kế chi tiết

### 2.6.4 Lập trình và kiểm tra chương trình

Trong giai đoạn này, mỗi thành phần đã được thiết kế sẽ được lập trình thành những mô đun chương trình (chương trình con). Mỗi mô đun này sẽ được kiểm chứng hoặc thử nghiệm theo các tài liệu đặc tả của giai đoạn thiết kế. Công việc này được mô tả như sau:




Hình 2-24 Lập trình và kiểm tra chương trình

Sau đó các mô đun chương trình đã được kiểm tra, được tích hợp với nhau thành hệ thống tổng thể và nó sẽ được kiểm tra xem có đáp ứng được các yêu cầu của khách hàng hay không. Kết thúc pha này là phần mềm cần phải xây dựng.

Hiện nay có một số công cụ hỗ trợ cho quá trình phân tích, thiết kế và có thể sinh mã tự động cho những thành phần chính của mô hình như: Rose [8], hay Visual Studio .NET của Microsoft, v.v.

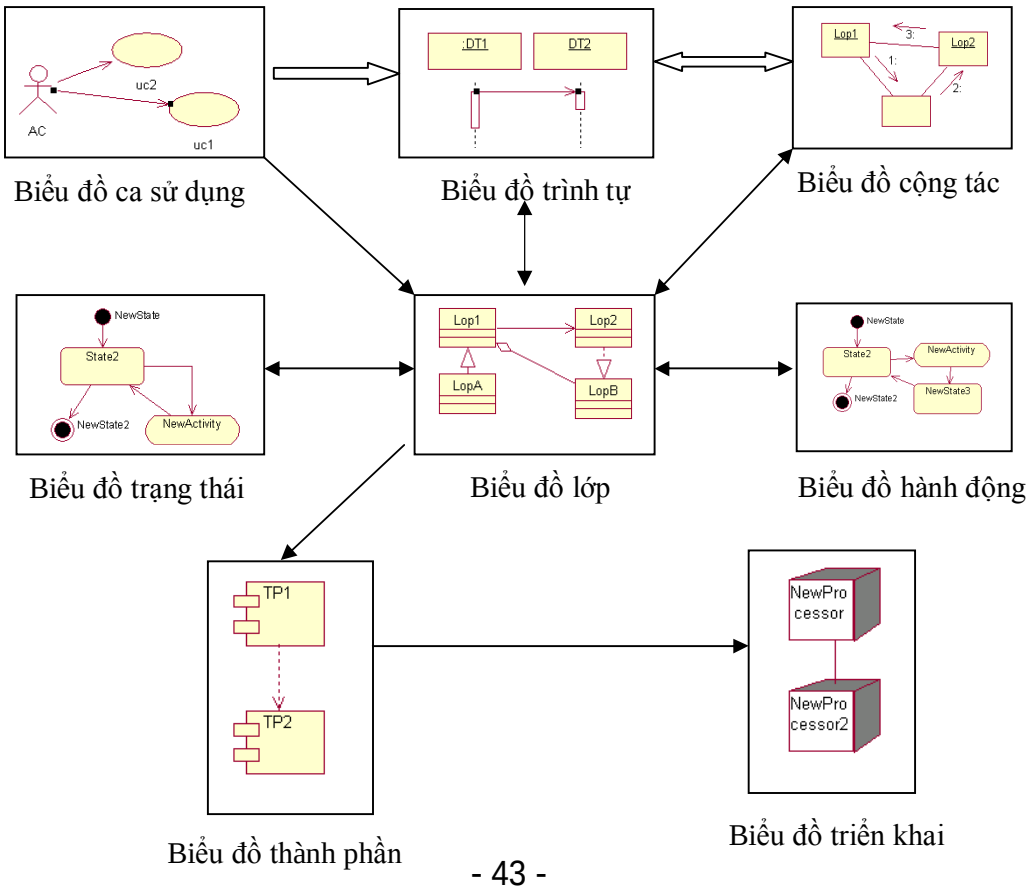
### 2.6.5 Vận hành và bảo trì hệ thống

Giai đoạn này bắt đầu bằng việc cài đặt hệ thống phần mềm trong môi trường sử dụng của khách hàng sau khi sản phẩm đã được giao cho họ. Hệ thống sẽ hoạt động, cung cấp các thông tin, xử lý các yêu cầu và thực hiện những gì đã được thiết kế.

Tuy nhiên vấn đề bảo trì phần mềm hoàn toàn khác với bảo trì của phần cứng. Như đã phân tích ở trên, *bảo trì phần mềm là đảm bảo cho hệ thống hoạt động đáp ứng được các yêu cầu của NSD, của khách hàng*. Mà các yêu cầu này trong thực tế lại hay thay đổi, do vậy công tác bảo trì lại bao gồm cả những sự thay đổi hệ thống sao cho nó phù hợp với yêu cầu hiện tại của họ, thậm chí có những thay đổi chưa phát hiện được trong các pha phân tích, thiết kế. Nghĩa là hệ thống phần mềm phải được nâng cấp, hoàn thiện liên tục và chi phí cho công tác bảo trì là khá tốn kém. Thông thường, có hai loại nâng cấp:

- *Nâng cao hiệu quả của hệ thống*: bao gồm những thay đổi mà khách hàng cho là sẽ cải thiện hiệu quả công việc của hệ thống, như bổ sung thêm các chức năng hay giảm thời gian xử lý, trả lời của hệ thống, v.v.
- *Đảm bảo sự thích nghi đối với sự thay đổi của môi trường* của hệ thống hay sự sửa đổi cho phù hợp với những thay đổi của chính sách, qui chế mới ban hành của Chính phủ.

*Tóm lại, thực hiện phân tích và thiết kế hướng đối tượng bằng UML là xây dựng các biểu đồ mô tả các yêu cầu, khái niệm và kiến trúc của hệ thống*. Quá trình xây dựng các biểu đồ đó có thể thực hiện như trong hình 2-25.




Biểu đồ ca sử dụng | Biểu đồ trình tự | Biểu đồ cộng tác

Biểu đồ trạng thái | Biểu đồ lớp | Biểu đồ hành động

Biểu đồ thành phần | Biểu đồ triển khai

Hình 2-25 Qui trình xây dựng các biểu đồ UML trong phân tích, thiết kế hệ thống

Chi tiết về các biểu đồ và cách xây dựng chúng như thế nào sẽ được đề cập ở các chương sau.

## 2.7 Rational Rose và quá trình phát triển phần mềm thống nhất

*Rational Rose* [8] là phần mềm công cụ mạnh hỗ trợ cho quá trình phân tích, thiết kế hệ thống hướng đối tượng. Nó giúp cho việc mô hình hoá hệ thống trước khi viết chương trình, đồng thời có khả năng kiểm tra đảm bảo tính đúng đắn, hợp lý của kiến trúc hệ thống từ khi khởi đầu dự án.

- Rose hỗ trợ để xây dựng các biểu đồ UML mô hình hoá các lớp, các thành phần và mối quan hệ của chúng trong hệ thống một cách trực quan và thống nhất.
- Nó cho phép mô tả chi tiết hệ thống bao gồm những cái gì, trao đổi tương tác với nhau và hoạt động như thế nào để người phát triển hệ thống có thể sử dụng mô hình như kế hoạch chi tiết cho việc xây dựng hệ thống.
- Rose còn hỗ trợ rất tốt trong giao tiếp với khách hàng và làm hồ sơ, tài liệu cho từng phần tử trong mô hình.
- Rose hỗ trợ cho việc chuyển bản thiết kế chi tiết sang mã chương trình trong một ngôn ngữ lập trình lựa chọn và ngược lại, mã chương trình có thể chuyển trở lại yêu cầu hệ thống. Rose hỗ trợ phát sinh mã khung chương trình trong nhiều ngôn ngữ lập trình khác nhau như: C++, Java, Visual Basic, Oracle 8, v.v.

Ngoài ra Rose hỗ trợ cho các nhà phân tích, thiết kế và phát triển phần mềm:

- Tổ chức mô hình hệ thống thành một hay nhiều tệp, được gọi là đơn vị điều khiển được (Controlled Unit). Cho phép phát triển song song các đơn thể điều khiển được của mô hình,
- Cho phép sao chép hay chuyển dịch các tệp mô hình, các đơn vị điều khiển được giữa các không gian làm việc khác nhau theo cơ chế "ánh xạ đường dẫn ảo" (Virtual Path Map),
- Cho phép quản lý mô hình và tích hợp với những hệ thống điều khiển chuẩn, Rose cung cấp khả năng tích hợp với ClearCase và Microsoft Visual SourceSafe, v.v.
- Sử dụng các bộ tích hợp mô hình (*Model Integator)* để so sánh và kết hợp các mô hình, các đơn vị điều khiển được với nhau.

Bản thân UML không định nghĩa quá trình phát triển phần mềm, nhưng UML và Rose hỗ trợ rất hiệu quả trong cả quá trình xây dựng phần mềm.

## Bài tập và câu hỏi

2.1 Vai trò của UML trong mô hình hoá hệ thống?

2.2 Có bao nhiêu loại biểu đồ, nêu tóm tắt nhiệm vụ của các biểu đồ.

2.3 Nêu những khái niệm cơ sở của phương pháp hướng đối tượng và các ký hiệu của chúng trong UML.

2.4 Quá trình phát triển phần mềm là gì, nêu các pha chính cần thực hiện theo cách tiếp cận hướng đối tượng.

*2.5* Tìm hiểu vai trò của Rational Rose trong quá trình phát triển phần mềm thống nhất.

*2.6* Chọn từ danh sách dưới đây những thuật ngữ thích hợp để điền vào các chỗ [(…)] trong đoạn văn mô tả về ngôn ngữ mô hình hoá UML.

> UML là ngôn ngữ mô hình hoá, trước hết nó mô tả [(1)], ngữ nghĩa các định nghĩa trực quan tất cả các thành phần của [(2)]. UML được sử dụng để hiển thị, đặc tả, tổ chức, xây dựng và [(3)] các vật phẩm (*artifacts*) của [(4)], đặc biệt là phân tích, thiết kế dưới dạng các báo cáo, biểu đồ, bản mẫu hay các trang web, v.v. UML là ngôn ngữ [(2)] hoá độc lập với các công nghệ phát triển [(5)].

*Chọn câu trả lời:*

a. quá trình phát triển phần mềm hướng đối tượng

b. quá trình xử lý

c. mô hình

d. ký pháp thống nhất

e. phần mềm

2.7 Chọn từ danh sách dưới đây những thuật ngữ thích hợp để điền vào các chỗ [(…)] trong đoạn văn mô tả về khái niệm lớp.

> Đối tượng là một thể hiện của một [(1)]. *Lớp là một mô tả về một nhóm các đối tượng có những* [(2)]*, có chung các* [(3)]*, có* [(4)] *với các đối tượng của các lớp khác và có chung ngữ nghĩa trong hệ thống.* [(1)] chính là cơ chế được sử dụng để phân loại các đối tượng của một hệ thống. Lớp thường xuất hiện dưới dạng những [(5)] trong các tài liệu mô tả bài toán hay trong các thảo luận với người sử dụng. Cũng như các đối tượng, lớp có thể là những nhóm các thực thể có trong thế giới thực, cũng có những lớp là khái niệm trừu tượng và có những lớp được đưa vào trong thiết kế để phục vụ cho cài đặt hệ thống, v.v.

*Chọn câu trả lời:*

a. hành vi ứng xử

b. cùng mối quan hệ

c. lớp

d. tính chất (thuộc tính) giống nhau

e. danh từ chung

# CHƯƠNG III
# BIỂU ĐỒ CA SỬ DỤNG
# PHÂN TÍCH CÁC NHU CẦU CỦA HỆ THỐNG

Chương III giới thiệu:

- ✓ Xác định nhu cầu của bài toán ứng dụng,
- ✓ Các thành phần của biểu đồ ca sử dụng: ca sử dụng, tác nhân ngoài,
- ✓ Phương pháp xác định, phân tích các yêu cầu của hệ thống và biểu đồ ca sử dụng,
- ✓ Cách xây dựng biểu đồ ca sử dụng để đặc tả các yêu cầu.

## 3.1 Định nghĩa bài toán

Đây là bước mở đầu của quá trình phát triển hệ thống, nhiệm vụ chủ yếu là xác định các nhu cầu của bài toán. "*Nhu cầu là mẹ của mọi sáng tạo*", cho nên để sáng tạo ra một hệ thống mới, người phát triển trước hết phải làm quen, thâm nhập vào chuyên môn, nghiệp vụ mà hệ thống đó phải đáp ứng và tìm hiểu, tập hợp các nhu cầu của hệ thống.

Sau đây chúng ta định nghĩa **bài toán làm cơ sở để mô tả cách thực hiện phân tích và thiết kế hệ thống phần mềm hướng đối tượng**. Bài toán được mô tả như sau:

**Bài toán***: "Một Công ty muốn xây dựng Hệ thống phần mềm để phục vụ và quản lý các hoạt động kinh doanh, bán hàng. Công ty có nhiều điểm bán hàng đầu cuối (POST: Point Of Sale Terminal) đó là những cửa hàng siêu thị, do vậy hệ thống cần phải ghi nhận các hoạt động bán hàng và xử lý các công việc thanh toán với khách hàng, chủ yếu khách hàng mua lẻ. Ngoài ra hệ thống còn giúp Giám đốc Công ty theo dõi được các hoạt động kinh doanh, tự động kiểm kê các mặt hàng tồn đọng trong kho, các mặt hàng bán chạy, v.v. để hỗ trợ ra quyết định trong các hoạt động kinh doanh của Công ty. Trong mỗi cửa hàng đầu cuối đều có các thiết bị phần cứng như: máy tính, máy đọc mã vạch ( bar code scanner) và phần mềm hệ thống để chạy hệ thống sẽ được xây dựng*".

Hệ thống bán hàng viết tắt là HBH, là chương trình phần mềm được sử dụng để ghi lại các phiên bán hàng, xử lý và thanh toán nhanh với khách hàng, chủ yếu là phục vụ khách hàng mua lẻ. Thông thường thì một hệ thống mới được xây dựng là nhằm để thay thế cho

một hệ thống cũ đã bộc lộ nhiều điều bất cập. Chính vì thế mà việc tìm hiểu nhu cầu với hệ thống mới thường bắt đầu từ việc khảo sát và đánh giá hệ thống cũ.

**Mục đích của HBH**

- Tăng nhanh hoặc tự động hoá việc bán hàng, ghi nhận các mặt hàng: loại sản phẩm, mô tả sản phẩm, số lượng và xác định giá bán, tính tiền, v.v., đáp ứng mọi yêu cầu của khách hàng,
- Thanh toán nhanh với khách hàng bằng các phương thức: tiền mặt, thẻ tín dụng (*Credit Card*), hay séc (*Check*),
- Phân tích, xử lý các kết quả bán hàng nhanh và chính xác để hỗ trợ quyết định trong các hoạt động kinh doanh,
- Thực hiện tự động kiểm kê các mặt hàng trong kho, theo dõi được những mặt hàng bán chạy, những mặt hàng tồn kho để có được những quyết định kịp thời trong kinh doanh.

*Tóm lại, hệ thống xây dựng nhằm tự động hoá các hoạt động kinh doanh, phục vụ khách hàng nhanh hơn, tốt hơn và rẻ hơn.*

**Các chức năng, nhiệm vụ của hệ thống**

*Chức năng của hệ thống* là những gì mà hệ thống được yêu cầu thực hiện.

*Nhiệm vụ "X" sẽ là chức năng của hệ thống nếu trong mô tả bài toán có mệnh đề dạng*: *Hệ thống phải thực hiện X.*

Tất nhiên trong giai đoạn này, các tính chất, các yêu cầu về chất lượng hệ thống như tính hiệu quả, an toàn hệ thống chưa được xem là chức năng của hệ thống, nghĩa là chưa xét tới các đặc tính phi chức năng của hệ thống.

Các chức năng của hệ thống có thể phân loại thành các phạm trù theo các lĩnh vực chức năng hay theo những mức ưu tiên khác nhau để loại trừ sự lẫn lộn giữa chúng. Các chức năng hệ thống có thể chia thành:

- *Những chức năng hiển (Evident)*: Những chức năng cần thực hiện và NSD có thể nhận biết, theo dõi được sự hoạt động của chúng. *Ví dụ*: khi người bán nhập các mặt hàng mà khách đã chọn mua ở trong giỏ hàng vào hệ thống thì mọi thông tin liên quan đến tên gọi sản phẩm, số lượng, giá bán, v.v. đều phải được hiện lên màn hình và khách hàng có thể theo dõi một cách tường minh.
- *Những chức năng ẩn (hiddent):* Những chức năng cần thực hiện và NSD không theo dõi được. Thường đó là những chức năng kỹ thuật như những công việc tổ chức lưu trữ, xử lý dữ liệu để đảm bảo sự bền vững dữ liệu trong các CSDL. *Ví dụ*: sau mỗi phiên bán hàng, nghĩa là sau khi khách đã trả đủ tiền mua hàng, hệ thống bán hàng HBH phải thực hiện cập nhật lại số lượng của những mặt hàng vừa bán được. Những hoạt động này NSD không theo dõi được.

- *Một số chức năng tuỳ chọn (optional):* Những chức năng có thể bổ sung tăng thêm mức độ thân thiện, tiện dụng cho hệ thống nhưng không ảnh hưởng tới giá trị cũng như các chức năng khác của hệ thống.

Các chức năng của hệ thống phải được chia thành các nhóm theo các mối liên hệ cố kết với nhau. Dựa vào cách phân chia các chức năng để sau này chia nhỏ hệ thống thành các gói, các hệ thống con trong quá trình phân tích và thiết kế hệ thống.

*Ví dụ:* Các chức năng của hệ thống HBH có thể chia thành hai nhóm chính: các *chức năng bán hàng* (các chức năng cơ sở) và *các chức năng thanh toán*.

Dựa trên những kết quả khảo sát bài toán bán hàng, nghiên cứu các sổ sách, tài liệu và trên cơ sở trao đổi với những người bán hàng, với khách hàng, v.v. chúng ta xác định được các chức năng chính của hệ thống như sau:

| Qui tắc # | Chức năng | Loại |
|---|---|---|
| R1.1 | Ghi nhận các mặt hàng ở trong giỏ hàng mà khách hàng đã chọn. | Hiển |
| R1.2 | Tính tổng số tiền bán cho khách hàng đang mua. | Hiển |
| R1.3 | Nhập các thông tin về các mặt hàng qua mã vạch bằng máy đọc mã vạch hoặc nhập mã sản phẩm (UPC – Universal Product Code) trực tiếp từ bàn phím. | Hiển |
| R1.4 | Cập nhật, trừ bớt số lượng đã bán sau từng phiên bán hàng. | Ẩn |
| R1.5 | Kết thúc một phiên bán hàng. | Ẩn |
| R1.6 | Người bán hàng (cashier) phải login để khởi động hệ thống (cho biết tên ID và password) để sử dụng hệ thống. | Hiển |
| R1.7 | Cung cấp một cơ chế lưu trữ nhất quán, CSDL. | Ẩn |
| R1.8 | Cung cấp cơ chế trao đổi giữa các tiến trình, trao đổi thông tin giữa các hệ thống với nhau. | Ẩn |
| R1.9 | Hiển thị các thông tin mô tả và giá bán các mặt hàng để khách hàng có thể theo dõi được. | Hiển |

Những chức năng thực hiện thanh toán với khách hàng.

| Qui tắc # | Chức năng | Loại |
|---|---|---|
| R2.1 | Thu tiền mặt, nhập số tiền khách đưa và tính số dư phải trả lại cho khách hàng. | Hiển |
| R2.2 | Thu tiền bằng thẻ tín dụng (*Credit*), nhập thông tin của thẻ tín dụng của khách qua máy đọc thẻ hoặc nhập trực tiếp từ bàn phím. | Hiển |
| R2.3 | Thu tiền bằng séc, nhập số hiệu và số tiền của tờ Check, tính số dư phải trả lại cho khách. | Hiển |

Các chức năng hệ thống thường được đánh số theo các qui tắc tham chiếu (*Reference Rule)* để tiện cho việc sử dụng tham chiếu trong các mục phân tích sau này.

*Chú ý: Trong đánh số các mục, phần, hay qui tắc, v.v. chúng ta sử dụng thống nhất qui tắc đánh số dấu chấm ('.')* như trong các tài liệu vẫn sử dụng.

**Các thuộc tính của hệ thống**

*Thuộc tính của hệ thống* là những yêu cầu phi chức năng (*Non-functional Requirement*), đó là những ràng buộc, những giới hạn và những yêu cầu kỹ thuật mà người phát triển hệ thống phải tuân theo.

Hệ thống HBH phải có các thuộc tính sau:

- *Thời gian xử lý và trả lời nhanh*, ví dụ: khi nhập vào mã từng mặt hàng (máy đọc mã vạch, hay từ bàn phím) thì các thông tin về sản phẩm, giá bán phải được hiển thị ngay tức thì (sau 5 giây chẳng hạn).
- *Dễ sử dụng với những giao diện đồ hoạ thân thiện* phù hợp với người bán hàng: như các window và các hộp thoại, v.v.
- *Hệ thống thực hiện trên những hệ điều hành phổ dụng như Microsoft Window* 95, 98, 2000, NT, Unix, Linux, v.v.

Ngoài ra còn nhiều tính chất khác của hệ thống phần mềm như đã nêu ở chương I mà hệ thống cũng sẽ cần phải đáp ứng.

## 3.2 Phân tích và đặc tả các yêu cầu hệ thống

### 3.2.1 Ca sử dụng

Khái niệm *ca sử dụng, hay trường hợp sử dụng (Use Case)* được Ivan Jacobson đề xuất từ năm 1994 nhằm mô tả các dịch vụ của hệ thống cho khách hàng và xác định mối quan hệ tương tác giữa hệ thống phần mềm với NSD trong nghiệp vụ.

Một cách hình thức hơn, *ca sử dụng mô tả tập các hoạt động của hệ thống theo quan điểm của các tác nhân (Actor). Nó mô tả các yêu cầu của hệ thống* và trả lời cho câu hỏi:

Hệ thống phải làm *cái gì* (What ?).

Ca sử dụng mô tả một quá trình từ bắt đầu cho đến khi kết thúc, gồm dãy các thao tác, các giao dịch cần thiết để sản sinh ra cái gì đó (giá trị, thông tin) theo yêu cầu của một tổ chức, của tác nhân, v.v.

Ca sử dụng được ký hiệu là:


Hình 3-1 Ký hiệu của ca sử dụng

Trong đó, "*Hoạt động"* là các chức năng, nhiệm vụ hay gọi chung là dịch vụ của hệ thống và nó thường được mô tả bằng các *động từ*, hay *mệnh đề động từ đơn,* ví dụ: *bán hàng, thanh toán, khởi động hệ thống, v.v.*

Những ca sử dụng phức tạp sẽ được mô tả chi tiết thông qua các *kịch bản* (*scenario*).

*Mục tiêu* của ca sử dụng trong cả quá trình phát triển phần mềm:

- Mô tả các yêu cầu chức năng của hệ thống, là kết quả của quá trình khảo sát, nghiên cứu các yêu cầu của bài toán và những thoả thuận giữa khách hàng, NSD hệ thống với người phát triển phần mềm.
- Làm cơ sở để người phân tích viên hiểu, người thiết kế xây dựng các kiến trúc, người lập trình cài đặt các chức năng, người kiểm duyệt kiểm tra các kết quả thực hiện của hệ thống.

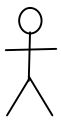
Các ca sử dụng đóng vai trò rất quan trọng trong cả quá trình phát triển phần mềm, tất cả các pha phân tích, thiết kế sau này đều dựa vào các ca sử dụng. Như vậy, quá trình được hướng dẫn bởi ca sử dụng (*use case driven process*) là một cách hữu hiệu để mô hình hoá hệ thống với UML.

### 3.2.2 Tác nhân

*Tác nhân ngoài, hay gọi ngắn gọn là tác nhân (External Actor, Actor)* là những thực thể bên ngoài có tương tác với hệ thống, bao gồm người, vật, thiết bị hay các hệ thống khác có trao đổi thông tin với hệ thống. Nói cách khác, tác nhân đại diện cho người hay một bộ phận của tổ chức mong muốn nhận được các thông tin (dữ liệu) hoặc các câu trả lời từ những ca sử dụng tương ứng.

*Ví dụ: Khách mua hàng, người bán hàng* là hai tác nhân của HBH.

Ký hiệu của tác nhân là *hình nhân cùng với tên gọi* như hình 3-2.

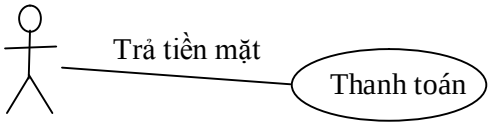


Khách hàng

Hình 3-2 Ký hiệu tác nhân *Khách hàng*

*Tên gọi của tác nhân* được mô tả bằng các danh từ (chung) và thường phải đặc tả được vai trò của nó đối với hệ thống.

Tác nhân trao đổi với hệ thống thông qua việc tương tác, sử dụng các dịch vụ của hệ thống là các ca sử dụng bằng cách trao đổi các thông điệp (*Message).* Như vậy, tác nhân sẽ cung cấp hoặc sử dụng các thông tin của hệ thống thông qua các ca sử dụng.




Khách hàng

Hình 3-3 Tác nhân *Khách hàng* tương tác với ca sử dụng *Thanh toán*

Thông thường trong mỗi hệ thống, khách hàng, NSD, người quản lý, người phụ vụ, v.v. có thể xem như là các tác nhân của hệ thống đó. Chúng ta cũng dễ nhận thấy, một ca sử dụng thì phải được khởi động bởi, hay phục vụ cho một hay nhiều tác nhân.

### 3.2.3 Xác định các ca sử dụng và các tác nhân

Thông thường việc xác định và hiểu rõ các yêu cầu hệ thống là rất khó khăn, phức tạp vì khối lượng thông tin liên quan là rất nhiều, được mô tả lộn xộn và không có cấu trúc. Khái niệm ca sử dụng được đưa ra để biểu thị các yêu cầu từ phía NSD, xuất phát từ quan điểm đơn giản là hệ thống được xây dựng trước hết là cho những NSD chúng, là để phục vụ tốt cho khách hàng.

Như trên đã phân tích, các tác nhân và các ca sử dụng của một hệ thống có mối quan hệ chặt chẽ với nhau. Mỗi tác nhân phải liên quan đến ít nhất một ca sử dụng và ngược lại mỗi ca sử dụng lại phục vụ trực tiếp hoặc gián tiếp cho một số tác nhân. Như vậy, các tác nhân và các ca sử dụng cùng mối quan hệ của chúng mô tả bức tranh khái quát về hệ thống, đặc tả đầy đủ về các yêu cầu của hệ thống. Do đó, vấn đề rất quan trọng đặt ra là làm thế nào để xác định được đầy đủ và chính xác các tác nhân ngoài, các ca sử dụng của hệ thống cần xây dựng.

**Xác định các tác nhân**

Tác nhân là một bộ phận bên ngoài hệ thống nhưng cộng tác chặt chẽ với hệ thống. Nó chính là đối tượng mà hệ thống phục vụ hoặc cần có để cung cấp dữ liệu. Do đó, nhiệm vụ trước tiên của người phân tích là xác định các tác nhân.

Một trong các kỹ thuật hỗ trợ để xác định các tác nhân là dựa trên các câu trả lời những câu hỏi sau:

- Ai sẽ sử dụng các chức năng chính của hệ thống?
- Ai cần sự hỗ trợ của hệ thống để thực hiện các công việc hàng ngày?
- Ai quản trị, bảo dưỡng để đảm bảo cho hệ thống hoạt động thường xuyên?
- Hệ thống quản lý, sử dụng những thiết bị nào?
- Hệ thống cần tương tác với những bộ phận, hệ thống nào khác?
- Ai hay cái gì quan tâm đến kết quả xử lý của hệ thống?

**Xác định các ca sử dụng**

Bước tiếp theo là xác định các ca sử dụng dựa trên những tài liệu đặc tả các yêu cầu, thông qua các tác nhân, v.v. Có hai phương pháp chính hỗ trợ giúp ta xác định các ca sử dụng:

1. Phương pháp thứ nhất là *dựa vào các tác nhân*:

a. Xác định những tác nhân liên quan đến hệ thống hoặc đến tổ chức, nghĩa là tìm và xác định những tác nhân là NSD hay những hệ thống khác tương tác với hệ thống cần xây dựng.

b. Với mỗi tác nhân, tìm những tiến trình (chức năng) được khởi đầu, hay giúp các tác nhân thực hiện, giao tiếp / tương tác với hệ thống.

2. Phương pháp thứ hai để tìm các ca sử dụng là *dựa vào các sự kiện*.

a. Xác định những sự kiện bên ngoài có tác động đến hệ thống hay hệ thống phải trả lời.

b. Tìm mối liên quan giữa các sự kiện và các ca sử dụng.

Tương tự như trên, hãy trả lời những câu hỏi sau đây để tìm ra các ca sử dụng:

1. Nhiệm vụ chính của các tác nhân là gì?
2. Tác nhân cần phải đọc, ghi, sửa đổi, cập nhật, hay lưu trữ thông tin hay không?
3. Những thay đổi bên ngoài hệ thống thì tác nhân có cần phải thông báo cho hệ thống hay không?
4. Những tác nhân nào cần được thông báo về những thay đổi của hệ thống?
5. Hệ thống cần có những đầu vào / ra nào?, từ đâu và đến đâu?

Dựa vào các phương pháp nêu trên, chúng ta hãy xác định các tác nhân và các ca sử dụng của hệ thống HBH.

**1. Danh sách các tác nhân của HBH:**

+ *Khách hàng (Customer)*: là những người được hệ HBH phục vụ, là khách hàng.

+ *Người bán hàng (Cashier):* những người cần sử dụng chức năng bán hàng của hệ thống để thực hiện nhiệm vụ của mình.

+ *Người quản lý (Manager):* những người được phép khởi động (*Start Up*) hay kết thúc cả hệ thống (*Shut Down*) tại các điểm bán hàng đầu cuối.

+ *Người quản trị hệ thống (System Administrator):* có thể bổ sung, thay đổi những NSD.

**2. Danh sách các ca sử dụng của HBH**

1. *Bán hàng, mua hàng (Buy Items)* là nhiệm vụ của hệ thống HBH liên quan trực tiếp tới *khách hàng và người bán hàng.* Trong trường hợp này, hai chức năng *bán hàng* và *mua hàng* là đồng nghĩa, nên có thể chọn một trong hai chức năng đó. Ca sử dụng này liên quan đến cả *người bán hàng* và *khách hàng*.

2. *Thanh toán, trả tiền mua hàng hay thu tiền (Refund Items, Cash Out):* là chức năng mà hệ thống phải thực hiện để thanh toán với khách hàng bằng

phương thức mà họ lựa chọn: trả tiền mặt, thẻ tín dụng, hay trả bằng séc. Ca sử dụng này cũng liên quan đến cả *người bán hàng* và *khách hàng.*

3. *Đăng nhập hệ thống (Log In): Người bán hàng* cần sử dụng để nhập vào hệ thống và sử dụng nó để bán hàng.
4. *Khởi động (Start Up), Đóng hệ thống (Shut Down): Người quản lý* thực hiện để khởi động hay kết thúc hoạt động của hệ thống.
5. *Bổ sung NSD mới (Add New Users), Loại bỏ NSD (Remove User): Người quản trị hệ thống* có thể bổ sung thêm người sử dụng mới hay loại bỏ những NSD không còn cần sử dụng hệ thống.

Sau khi xác định được các tác nhân và các ca sử dụng thì phải đặt lại tên cho chúng. Tên của các tác nhân và ca sử dụng phải đơn giản, rõ nghĩa và phù hợp với lĩnh vực của bài toán ứng dụng.

- Tên của tác nhân phải là *danh từ chung* và biểu hiện được vai trò của nó trong các mối quan hệ với hệ thống.
- Tên của ca sử dụng phải *bắt đầu bằng động từ, là mệnh đề đơn,* ngắn gọn và mô tả đúng nhiệm vụ mà hệ thống cần thực hiện.

*Tóm lại*, danh sách các tác nhân và ca sử dụng trong hệ HBH được xác định như sau:

| Tác nhân | Ca sử dụng |
|---|---|
| Người bán hàng (Cashier) | Đăng nhập hệ thống (Log In)<br>Thu tiền bán hàng (Cash Out) |
| Khách hàng (Customer) | Mua hàng (Buy Items)<br>Thu tiền, thanh toán (Refund Items) |
| Người quản lý (Manager)<br>Hay gian hàng trưởng | Khởi động hệ thống (Start Up) để *người bán hàng* có thể sử dụng để bán hàng.<br>Đóng hệ thống khi hết giờ (Shut Down) |
| Quản trị hệ thống (System Adminitrator) | Bổ sung NSD (Add New Users)<br>Loại bỏ NSD (Delete User) |

*Lưu ý:* Các chức năng *mua hàng* và *bán hàng*là tương ứng với *khách hàng* hay *người bán*, nhưng trong hệ HBH ta có thể sử dụng một tên gọi chung là *bán hàng.* Tương tự, *Thu tiền bán hàng* và *Thu tiền* có thể đồng nhất là *Thu tiền hoặc Thanh toán.*

### 3.2.3 Đặc tả các ca sử dụng

Để hiểu rõ hơn về tiến trình xử lý các yêu cầu của hệ thống, ta nên xây dựng các đặc tả cho các ca sử dụng.

Mẫu (*Format*) đặc tả ca sử dụng có dạng:

*Ca sử dụng*: Tên của ca sử dụng bắt đầu bằng động từ.

*Các tác nhân:* Danh sách các tác nhân liên quan đến ca sử dụng, chỉ rõ ai bắt đầu với ca sử dụng này.

*Mô tả:* Mô tả tóm tắt tiến trình xử lý công việc cần thực hiện.

*Tham chiếu:* Các chức năng, ca sử dụng và những hệ thống liên quan.

*Ví dụ:* Đặc tả một số ca sử dụng trong HBH:

**1. Ca sử dụng** : *Mua hay bán hàng* (Buy Items)

*Tác nhân*: Khách hàng, người bán hàng

*Mô tả*: Khách hàng sau khi đã chọn đủ các mặt hàng cần mua để ở trong giỏ hàng thì đưa hàng đến quầy thu tiền. Người bán (cashier) lần lượt ghi nhận các mặt hàng trong giỏ hàng của khách và thu tiền. Sau khi thanh toán xong khách hàng được mang số hàng đã mua đi ra khỏi cửa hàng.

*Tham chiếu tới*: Các chức năng R1.1, R1.2,2 R1.3, R1.6, R1.7, R1.8, R2.1, R2.2, R2.3.

**2. Ca sử dụng** : *Thu tiền* (thanh toán)

*Tác nhân*: Khách hàng, người bán hàng.

*Mô tả*: Khách hàng có thể trả tiền theo 3 phương thức:

1. Trả tiền mặt (Pay by Cash)
2. Trả bằng thẻ tín dụng (Pay by Credit)
3. Trả bằng séc (Pay by Check)

Người bán nhận tiền mặt, thẻ tín dụng, tiền séc rồi thanh toán tiền thừa cho khách hàng sau khi thẻ tín dụng, séc đã được kiểm duyệt.

*Tham chiếu tới*: Các chức năng R1.1, R1.2,2 R1.3, R1.9, R2.1, R2.2, R2.3.

Tương tự mô tả tiếp các ca sử dụng còn lại.

Như trong đặc tả ca sử dụng "*Thu tiền"* ta thấy nó lại được phân làm ba trường hợp: *Thu tiền mặt, thu bằng thẻ tín dụng và thu bằng séc*. Do đó, để hiểu rõ hơn các hoạt động của hệ thống chúng ta có thể bổ sung thêm ba ca sử dụng mới: *Thu tiền mặt, thu bằng thẻ tín dụng và thu bằng séc.* Để thành toán được *bằng thẻ tín dụng và bằng séc* thì thẻ tín dụng, séc phải được kiểm duyệt bởi các tác nhân:

+ *Bộ phận kiểm duyệt thẻ tín dụng* (Credit Authorization Service): Giúp hệ thống kiểm tra thẻ tín dụng.

+ *Bộ phận kiểm duyệt séc* (Check Authorization Service): Giúp hệ thống kiểm tra séc.

Ngoài những đặc tả nêu trên, ta còn có thể xây dựng các *kịch bản (scenario)* hành động để mô tả các sự kiện xảy ra trong hệ thống. Mỗi kịch bản có thể mô tả theo hai luồng: luồng thực hiện của các tác nhân và luồng tương ứng với hệ thống. Ví dụ: đối với ca sử dụng "*Bán hàng"* có kịch bản thực hiện như sau:

| **Hành động của các tác nhân** | **Hành động của Hệ thống** |
|---|---|
| 1. Khách hàng sau khi chọn đủ số hàng cần thiết thì đưa hàng đã chọn đến cho quầy thu tiền | |
| 2. Người bán ghi nhận từng mặt hàng. Nếu một mặt hàng mua với số lượng nhiều hơn thì người bán nhập vào số lượng đó vào từ bàn phím. | 3. Xác định giá và các thông tin về sản phẩm được hiển thị. |
| 4. Khi đã nhập xong các mặt hàng của khách đã chọn mua thì người bán phải chỉ cho hệ HBH biết là đã kết thúc phiên bán hàng bằng cách nhấn phím Enter hoặc nhấn nút "Kết thúc" phiên bán hàng ( *EndSale*). | |
| | 5. Tính và hiển thị tổng số tiền bán hàng. |
| 6. Người bán thông báo cho *khách hàng* biết tổng số tiền phải trả. | |
| 7. Khách hàng chọn phương thức thanh toán:<br>a) Nếu chọn trả tiền mặt: xem tiếp kịch bản con (Sub_scenario) *Thu tiền mặt.*<br>b) Nếu trả bằng thẻ tín dụng: xem kịch bản con *Thu bằng thẻ tín dụng*.<br>c) Nếu trả tiền séc: xem kịch bản con *Thu bằng Check.* | 8. Hiển thị số tiền dư phải trả cho khách hàng |
| | 9. Kết thúc một phiên giao dịch bán hàng. |
| | 10. Cập nhật lại các hàng |

| | |
|---|---|
| | trong cửa hàng. |
| | 11. Phát sinh phiếu bán hàng (hoá đơn). |
| 12. Người bán trả tiền thừa và đưa phiếu bán hàng cho khách hàng. | |
| 13. Khách hàng ra khỏi cửa hàng với hàng đã thanh toán. | |

*Lưu ý:* Nếu khách hàng không trả đủ tiền, hoặc khi thẻ tín dụng, séc không hợp lệ thì huỷ bỏ phiên giao dịch đó.

Sau đó xây dựng những kịch bản khác hoặc những kịch bản con để hiểu và năm bắt được mọi yêu cầu của hệ thống.

Sau đây chúng ta xét tiếp kịch bản con *Thu tiền mặt* của ca sử dụng *Thu tiền.*

| **Hành động của các tác nhân** | **Hành động của Hệ thống** |
|---|---|
| 1. Khách hàng chọn phương thức trả bằng tiền mặt và trả cho người bán tiền mặt. | |
| 2. Người bán nhập vào số tiền khách hàng trả (số này có thể lớn hơn số tiền phải trả). | |
| | 3. Hệ thống hiển thị số dư phải trả lại cho khách. |
| 4. Người bán trả lại tiền dư. | |

Kịch bản con "*Thu bằng thẻ tín dụng*

| **Hành động của các tác nhân** | **Hành động của hệ thống** |
|---|---|
| 1. Khách hàng trả bằng thẻ tín dụng. | |
| | 2. Phát sinh yêu cầu trả thẻ tín dụng và gửi nó tới bộ phận kiểm tra thẻ tín dụng. |
| 3. Bộ phận kiểm tra thẻ cho phép trả tiền tín dụng sau khi đã kiểm tra. | |
| | 4. Trừ số tiền phải trả vào tài |

khoản tín dụng.

## 3.3 Biểu đồ ca sử dụng

*Biểu đồ ca sử dụng* chỉ ra mối quan hệ giữa các tác nhân và các ca sử dụng trong hệ thống.

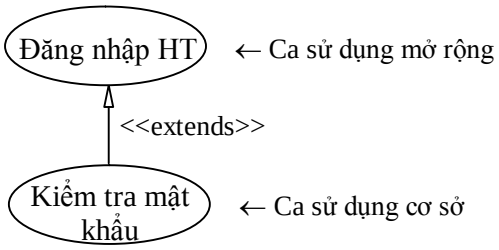
Mỗi ca sử dụng cần phải biểu diễn trọn vẹn một giao dịch giữa NSD và hệ thống.

**Mối quan hệ giữa các ca sử dụng**

Giữa các ca sử dụng có ba quan hệ chính: *quan hệ mở rộng (extends), quan hệ sử dụng (uses)* và *quan hệ theo nhóm hay theo gói (package).*

*1. Quan hệ mở rộng*

Trong khi xây dựng những ca sử dụng, ta nhận thấy có những ca sử dụng lại được sử dụng như là một phần của chức năng khác. Trong những trường hợp như thế ta nên xây dựng một ca sử dụng mới mở rộng một hay nhiều ca sử dụng đã xác định trước. Ca sử dụng mới được gọi là *ca sử dụng mở rộng* của những ca sử dụng cũ. Mối quan hệ mở rộng giữa các ca sử dụng được mô tả và ký hiệu giống như quan hệ tổng quát hoá với nhãn của quan hệ là <<extends>>. Ví dụ,




Hình 3-4 Mối quan hệ *mở rộng* giữa các ca sử dụng

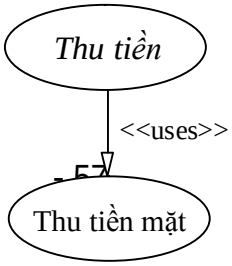
Để đăng nhập được vào một hệ thống phần mềm, ngoài việc *kiểm tra mật khẩu (password)* còn phải khai báo một số thuộc tính khác ví dụ như tên NSD chẳng hạn. Ngoài ra ca sử dụng *kiểm tra mật khẩu* có thể được sử dụng ở một số ca sử dụng khác ngoài ca sử dụng *đăng nhập hệ thống*.

Ca sử dụng *B mở rộng <<extends>> A* nếu: B là biến thể của A, nó chứa thêm một số sự kiện cho những điều kiện nào đó.

***2.*** *Quan hệ sử dụng*

Khi một số ca sử dụng có một hành vi chung thì hành vi này có thể xây dựng thành một ca sử dụng để có thể được sử dụng bởi những ca sử dụng khác. Mối quan hệ như thế được gọi là *mối quan hệ sử dụng*. Nói cách khác, trong mối quan hệ sử dụng, có một ca sử dụng sử dụng ca sử dụng khác để chỉ ra dạng chuyên biệt hoá ca sử dụng cơ sở.

Giống như mối quan hệ mở rộng, mối quan hệ sử dụng cũng sử dụng ký hiệu tổng quát hoá để thể hiện với nhãn <<uses>>. Ví dụ:



Hình 3-5 Quan hệ sử dụng giữa các ca sử dụng

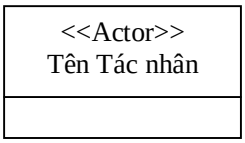
Để thực hiện được ca sử dụng *Thu tiền* thì phải gọi tới ca sử dụng *Thu tiền mặt* khi khách hàng chọn phương thức trả bằng tiền mặt.

Trong UML 1.3, quan hệ sử dụng <<uses>> được gọi là quan hệ *gộp vào* <<includes>>.

***3.*** *Mối quan hệ gộp nhóm*

Khi có một số ca sử dụng cùng xử lý những chức năng giống nhau hoặc có quan hệ với ca sử dụng khác theo cùng một cách thì tốt nhất là nhóm chúng lại thành từng *gói*. Ví dụ: hệ thống HBH có thể chia các ca sử dụng thành các gói *Bán hàng, gói Thanh toán* và *gói Quản lý hệ thống, v.v.*

*Lưu ý,* trong UML (thể hiện trong Rose) khái niệm *mẫu rập khuôn (stereotype)* được mô tả bằng một xâu và được đặt trong cặp << … >>, được sử dụng để phân nhóm các thành phần của mô hình hệ thống. Ví dụ, đối với hai ca sử dụng *UC A, UC B,* ta có thể tạo ra hai *mẫu rập khuôn* tương ứng <<UC A>>, <<UC B>>. *Stereotype* không được sử dụng thường xuyên cho các ca sử dụng, nó thường sử dụng cho lớp hay cho các mối quan hệ. Ví dụ, <<Actor>> là một mẫu rập khuôn trong UML và khi một lớp được chỉ ra trong biểu tượng (*icon)* với *stereotype: <<Actor>>* với dạng:




Thì nó được xem là lớp có kiểu *Actor*.

Sau khi đã xác định xong các ca sử dụng cho hệ thống thì phải kiểm tra lại xem chúng đã được phát hiện hết chưa. Công việc này được thực hiện bằng cách trả lời cho các câu hỏi sau:

- Mỗi yêu cầu chức năng của hệ thống đã có trong ít nhất một ca sử dụng chưa? Những chức năng chưa được mô tả trong các ca sử dụng sẽ không được cài đặt sau này.
- Các mối tương tác giữa các tác nhân và hệ thống đã xác định hết chưa?
- Tác nhân cung cấp những gì cho hệ thống?
- Tác nhân nhận được những gì từ hệ thống?
- Đã nhận biết được tất cả các hệ thống bên ngoài có tương tác với hệ thống chưa?

- Những thông tin nào mà hệ thống ngoài gửi tới hoặc nhận được từ hệ thống?

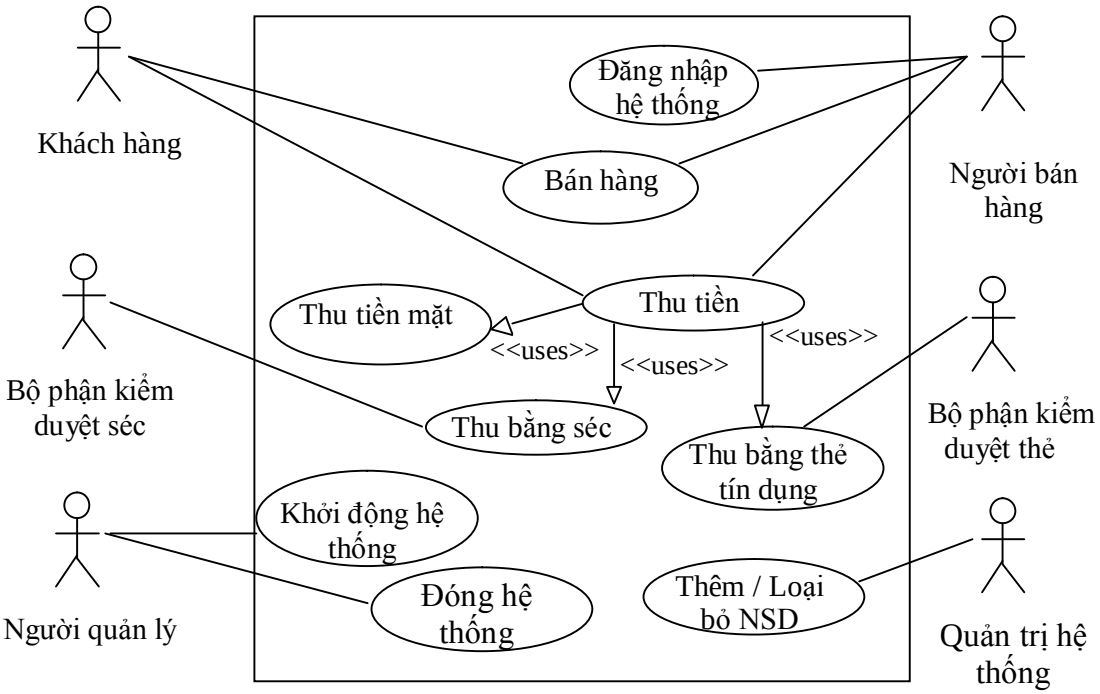
Một lần nữa cần duyệt lại xem đã xác định đầy đủ, chính xác các tác nhân, ca sử dụng và mối quan hệ của chúng chưa.

**Xây dựng biểu đồ ca sử dụng cho hệ HBH**

Trước khi xây dựng biểu đồ ca sử dụng chúng ta cần lưu ý:

- Mỗi ca sử dụng luôn được ít nhất một tác nhân kích hoạt một cách trực tiếp hay gián tiếp, nghĩa là giữa tác nhân và ca sử dụng thường có mối quan hệ giao tiếp (kết hợp).
- Ca sử dụng cung cấp các thông tin cần thiết cho tác nhân và ngược lại, tác nhân cung cấp dữ liệu đầu vào cho ca sử dụng thực hiện.

Từ những kết quả phân tích ở trên, đến đây chúng ta có thể xây dựng biểu đồ ca sử dụng cho HBH như sau:




Hình 3-6 Biểu đồ ca sử dụng của hệ thống HBH

Một số điểm cần chú ý khi xây dựng biểu đồ ca sử dụng:

- Giữa các tác nhân với nhau không có quan hệ giao tiếp với nhau.
- Giữa các ca sử dụng không có quan hệ trực tiếp, trừ mối quan hệ mở rộng <<*extends*>> hoặc <<*uses*>> như đã nói ở trên.

## 3.4 Tạo lập biểu đồ ca sử dụng trong Rational Rose

Trong Rose các biểu đồ ca sử dụng được tạo lập trong quan sát *Use Case.* Rose cung cấp biểu đồ ca sử dụng mặc định là *Main.* Số lượng các biểu đồ ca sử dụng là tuỳ ý tạo lập.

Trong Rose có thể thực hiện được những chức năng sau (chi tiết xem [11]):

- Mở biểu đồ mặc định *Main Use Case,*
- Tạo lập biểu đồ ca sử dụng mới,
- Loại bỏ một biểu đồ ca sử dụng,
- Tạo lập, bổ sung các ca sử dụng vào một biểu đồ ca sử dụng,
- Xoá bỏ ca sử dụng trong một biểu đồ ca sử dụng,
- Đặc tả các ca sử dụng: đặt tên, gán mức ưu tiên, gắn tệp cho ca sử dụng, v.v.
- Tạo lập, bổ sung các tác nhân,
- Xoá bỏ các tác nhân,
- Đặc tả các tác nhân: đặt tên, đặt bội số trong mục *Detail tab* sau khi chọn *Open Specification.*
- Thiết lập các mối quan hệ giữa các tác nhân với ca sử dụng và giữa một số ca sử dụng với nhau nếu có.

## Bài tập và câu hỏi

**3.1** Mối quan hệ giữa các chức năng của hệ thống với các ca sử dụng, chúng khác nhau như thế nào?, nêu cách để xác định các ca sử dụng.

**3.2** Tác nhân là gì và vai trò củ nó trong hệ thống? Nêu cách để xác định các tác nhân.

**3.3** Xây dựng kịch bản cho ca sử dụng "Đăng nhập hệ thống", "Thu bằng séc", "Thêm NSD".

**3.4** Phát biểu bài toán "Quản lý thư viện", xác định các yêu và xây dựng biểu đồ ca sử dụng cho hệ thống này.

**3.5** Phát biểu bài toán "Xây dựng hệ thống rút tiền tự động ATM (Automatic Teller Machine)", xác định các yêu và xây dựng biểu đồ ca sử dụng cho hệ thống này.

**3.6** Phát biểu bài toán xây dựng hệ thống "Mô phỏng hệ thống thang máy cho các nhà cao tầng", xác định các yêu và xây dựng biểu đồ ca sử dụng cho hệ thống này.

**3.7** Sử dụng Rational Rose để tạo lập biểu đồ ca sử dụng cho hệ thống HBH với đầy đủ các chức năng như trong mục 3.4.

**3.8** A/ Chọn những từ thích hợp để điền vào chỗ trống của những câu sau:

- Mẫu rập khuôn (stereotype) trong UML được mô tả . . . . .
- Có . . . . loại quan hệ giữa các ca sử dụng với nhau.

- Biểu đồ ca sử dụng mô tả . . . . . của hệ thống.

B/ Những mệnh đề sau đúng hay sai?

- Một tác nhân trong ca sử dụng luôn là một người có liên quan đến hệ thống.
- Ca sử dụng mô tả những chi tiết để cài đặt.
- Một hệ thống khác cũng có thể là một tác nhân trong biểu đồ ca sử dụng của hệ thống.
- Trong mỗi hệ thống chỉ có một biểu đồ ca sử dụng.

# CHƯƠNG IV

# PHÂN TÍCH HỆ THỐNG – MÔ HÌNH KHÁI NIỆM VÀ BIỂU ĐỒ LỚP

Chủ đề chính của chương V:

- ✓ Tìm hiểu những khái niệm cơ sở trong miền bài toán ứng dụng để xây dựng mô hình khái niệm,
- ✓ Các phương pháp xác định các lớp đối tượng và mối quan hệ giữa các lớp,
- ✓ Các thuộc tính của các lớp,
- ✓ Biểu đồ lớp – mô hình khái niệm của hệ thống.

## 4.1 Mô hình khái niệm – mô hình đối tượng

Mục tiêu chính của phương pháp hướng đối tượng là phân tách hệ thống thành các đối tượng hoặc xác định các *khái niệm* (*concepts*) là những sự vật (*things*) mà chúng ta biết rõ về chúng. Mô hình khái niệm là cách biểu diễn các khái niệm, các thực thể của phạm vi bài toán.

**Các khái niệm cơ bản**

Một cách không hình thức, một *khái niệm là một ý tưởng (idea), sự vật, hoặc một đối tượng.* Hình thức hơn, *một khái niệm có thể được suy nghĩ thấu đáo về các phương diện:*

- *Ký hiệu*: những từ hay hình ảnh biểu diễn cho một khái niệm
- *Định nghĩa* về một khái niệm và
- *Sự mở rộng* bao gồm tập các ví dụ hoặc các thể hiện của một khái niệm.

Sự khác biệt chính giữa phân tích hướng đối tượng và phân tích có cấu trúc là sự phân rã hệ thống thành các khái niệm (đối tượng ) thay vì phân rã thành các chức năng. Điểm mấu chốt trong cách phân rã này là *phải tìm cái gì hoạt động trong hệ thống* để sau đó quyết định thiết kế và cài đặt chúng nhằm thực hiện các chức năng (yêu cầu) của bài toán. Do vậy, nhiệm vụ trọng tâm của giai đoạn phân tích hướng đối

tượng là tìm cách xác định các lớp đối tượng và các mối quan hệ của chúng trong hệ thống.

Nhắc lại định nghĩa lớp trong UML, *lớp là một mô tả về tập các đối tượng có cùng chung các thuộc tính, các phương thức hành động, các mối quan hệ và giống nhau về ngữ nghĩa.* Chúng ta nhận thấy những khái niệm về *thuộc tính, phương thức, thao tác và các mối quan hệ* thì dần từng bước sẽ được làm sáng tỏ trong các bước thực hiện của quá trình phát triển hệ thống.

Các đối tượng trong hệ thống cần phải được phân biệt với nhau. Hai đối tượng có cùng tập các tính chất, ví dụ hai sinh viên của lớp **SinhVien** có cùng tên, tuổi và thậm chí cùng điểm thi, học cùng năm, v.v. nhưng vẫn phải có tính chất nào đó để phân biệt được họ. Thuộc tính để phân biệt các đối tượng trong hệ thống chính là *định danh* (*Identity*). Như vậy, những đặc trưng quan trọng của đối tượng là:

- *Định danh đối tượng* (*Object's Identity*): dùng để phân biệt với những đối tượng khác, ngay cả khi chúng có các tính chất giống nhau.
- *Tính bền vững của đối tượng* (*Object's Persistence*): mỗi đối tượng đều có một thời gian sống (tồn tại trong hệ thống) và điều này dẫn tới bản chất tĩnh của hệ thống.
- *Mỗi đối tượng phải hoặc có thể tương tác với các đối tượng khác*, điều này dẫn đến bản chất động của hệ thống.

Khi xây dựng mô hình cho một hệ thống phần mềm thì những khái niệm được sử dụng để tạo ra mô hình phải đơn giản, dễ hiểu, và làm cơ sở để trao đổi với nhau trong quá trình phát triển hệ thống. Những khái niệm về nghiệp vụ phải được thiết kế lại sao cho phù hợp với qui trình xử lý công việc hiện thời đồng có thể thích ứng được với xu thế phát triển trong tương lai.

Trong UML, mô hình khái niệm của một hệ thống được mô tả bởi *biểu đồ lớp* (*Class Diagram*). Vậy, vấn đề quan trọng của giai đoạn này là xác định đầy đủ và chính xác các lớp đối tượng và mối quan hệ của chúng trong hệ thống cần phát triển.

Như đã khẳng định ở chương III, quá trình phát triển phần mềm hướng đối tượng với UML là được hướng dẫn bởi các ca sử dụng, nên các lớp đối tượng chủ yếu cũng sẽ được xác định dựa trên cơ sở phân tích các ca sử dụng của hệ thống.

## 4.2 Xác định các lớp đối tượng

Có sáu cách chính để tìm các lớp đối tượng:

(i) Dựa vào văn bản, kịch bản mô tả bài toán: các danh từ có thể là các đại biểu của lớp.

(ii) Dựa vào danh sách phân loại các phạm trù khái niệm.

(iii) Dựa vào mục đích của các ca sử dụng [10],

(iv) Dựa vào kinh nghiệm và kiến thức của người phân tích,

(v) Dựa vào hồ sơ tài liệu những hệ thống có liên quan,

(vi) Dựa vào ý kiến tham khảo với các chuyên gia hệ thống.

Trong quá trình phân tích, thường phải kết hợp các cách trên để tìm ra các lớp của một hệ thống. Ở đây chúng ta tập trung tìm hiểu ba cách chính (đầu tiên) để xác định các lớp đối tượng trong quá trình phân tích một hệ thống.

**1. Dựa vào văn bản, kịch bản mô tả bài toán để tìm các lớp** ([6], [10])

Ở pha trước, giai đoạn phân tích các yêu cầu của hệ thống đã có các mô tả bài toán, đã xây dựng đặc tả các ca sử dụng và các kịch bản mô tả chi tiết những ca sử dụng phức tạp để hiểu rõ bài toán. Các *danh từ* trong các mô tả văn bản đó có thể là đại biểu của lớp hoặc thuộc tính của lớp. Do vậy, có thể *gạch chân tất cả các danh từ chung* trong văn bản mô tả bài toán. *Ví dụ,* gạch chân những danh từ chung trong đoạn văn mô tả hệ HBH:

"*Một Công ty muốn xây dựng hệ thống phần mềm để phục vụ và quản lý các hoạt động kinh doanh. Công ty có nhiều điểm bán hàng đầu cuối (point- of sale terminal) đó là những cửa hàng siêu thị, do vậy hệ thống cần phải ghi nhận các hoạt động bán hàng và xử lý các công việc thanh toán với khách hàng, chủ yếu khách hàng mua lẻ. Ngoài ra hệ thống còn giúp giám đốc Công ty theo dõi được các hoạt động kinh doanh, tự động kiểm kê các mặt hàng tồn đọng trong kho, các mặt hàng bán chạy, v.v. để hỗ trợ ra quyết định trong các hoạt động kinh doanh của Công ty. Trong mỗi điểm bán hàng đầu cuối đều có các thiết bị phần cứng như: máy tính,máy đọc mã vạch ( bar code scanner) và phần mềm hệ thống để chạy hệ thống sẽ được xây dựng*".

*Lưu ý:*

- Không phải tất cả các danh từ đã gạch chân trong văn bản mô tả bài toán đều là đại biểu lớp. Ví dụ: cụm danh từ phần mềm hệ thống, máy tính, thiết bị phần cứng, v.v. không phải là những thực thể mà chúng ta quan tâm trong hệ thống HBH.
- Một số danh từ, cụm danh từ đồng nghĩa với nhau hay mô tả cho những thực thể có vai trò như nhau trong hệ thống thì có thể chọn một trong số chúng hoặc đặt tên khác cho những danh từ đó. Ví dụ: hệ thống và hệ thống phần mềm, hay hoạt động bán hàng và hoạt động kinh doanh ở đây là giống nhau, vậy nên có thể thay hệ thống và hệ thống phần mềm bằng HBH, còn hoạt động bán hàng và hoạt động kinh doanh được gọi là giao dịch bán hàng (hay phiên bán hàng), v.v.

*Tóm lại*, dựa vào những danh từ, cụm danh từ đã được gạch chân, sau đó dựa vào kinh nghiệm, trình độ, kiến thức của mình mà loại bỏ đi những mục không phải là đại biểu của lớp. Danh sách các đại biểu có thể là lớp của hệ thống HBH do đó sẽ là:

**HBH**: đại diện cho *hệ thống phần mềm*, *hệ thống*,

**CuaHang** (cửa hàng): *điểm bán hàng đầu cuối*, *cửa hàng siêu thị*, *v.v*

**PhienBanHang** (phiên bán hàng): đại diện cho hoạt động bán hàng, hoạt động kinh doanh,

**ThanhToan** (thanh toán): đại diện cho công việc thanh toán, trả tiền

**KhachHang** (khách hàng): cho khách hàng, người mua hàng,

**NguoiQL** (Người quản lý): đại diện cho giám đốc Công ty, cửa hàng trưởng,

**MatHang** (Mặt hàng): đại diện cho mặt hàng, sản phẩm, v.v.

Tương tự, đối với các kịch bản. Ví dụ: xét kịch bản mô tả ca sử dụng "Bán hàng"

| Hành động của các tác nhân | Hành động của Hệ thống |
|---|---|
| 1. Khách hàng sau khi chọn đủ số hàng cần thiết thì đưa hàng đã chọn đến cho quầy thu tiền | |
| 2. Người bán ghi nhận từng mặt hàng. Nếu một mặt hàng mua với số lượng nhiều hơn thì người bán nhập vào số lượng đó vào từ bàn phím. | 3. Xác định giá và các thông tin về sản phẩm được hiển thị. |
| 4. Khi đã nhập xong các mặt hàng của khách đã chọn mua thì người bán phải chỉ cho hệ HBH biết là đã kết thúc phiên bán hàng bằng cách nhấn phím Enter hoặc nhấn nút "Kết thúc" phiên bán hàng ( EndSale). | |
| | 5. Tính và hiển thị tổng số tiền bán hàng. |
| 6. Người bán thông báo cho *khách hàng* biết tổng số tiền phải trả. | |
| 7. Khách hàng chọn phương thức thanh toán:<br>d) Nếu chọn trả tiền mặt: xem tiếp kịch bản con (Sub_scenario) *Thu tiền mặt.*<br>e) Nếu trả thẻ tín dụng: xem kịch bản con *Thu bằng thẻ tín dụng*.<br>f) Nếu trả tiền séc: xem kịch bản con *Thu bằng Check.* | 8. Hiển thị số tiền dư phải trả cho khách hàng |
| | 9. Kết thúc một phiên giao dịch bán hàng. |
| | 10. Cập nhật lại các hàng trong cửa hàng. |

11. Phát sinh phiếu bán hàng (hoá đơn).

12. Người bán trả tiền thừa và đưa phiếu bán hàng cho khách hàng.

13. Khách hàng ra khỏi cửa hàng với hàng đã thanh toán.

Ngoài những đại biểu lớp đã nêu trên, có thể liệt kê thêm những danh từ là đại biểu lớp trong HBH:

**NguoiBan** (người bán): đại diện cho người bán, người bán hàng, người thu tiền,
**MoTaMatHang** (thông tin về mặt hàng): thông tin về sản phẩm, thông tin về mặt hàng, đặc tả sản phẩm, v.v.
**PhieuBanHang** (phiếu bán hàng): phiếu bán hàng, hoá đơn, v.v.

**2. Dựa vào danh sách phân loại các phạm trù khái niệm** ([6], [10])

Bên cạnh những phương pháp nêu trên, chúng ta còn có thể dựa vào danh sách phân loại các phạm trù khái niệm để xác định thêm những đại biểu mới cho hệ thống hoặc loại đi những danh từ không thực sự là đại biểu của lớp.

*Ví dụ:* Xét các phạm trù khái niệm (*concept category*) của hệ thống "*bán hàng*" và hệ thống "Đặt vé máy bay".

| Các phạm trù khái niệm | Ví dụ |
|---|---|
| Các đối tượng vật lý, hay hữu hình | Hệ HBH, NguoiBan, KhachHang<br>MayBay (máy bay) |
| Mô tả, đặc tả, tài liệu của sự vật | MoTaMatHang<br>MoTaChuyenBay (mô tả chuyến bay) |
| Địa điểm, nơi, chốn | CuaHang<br>SanBay (sân bay) |
| Các mục trên dòng giao dịch (Transaction line items) | DongHangBan (Sales Line Item) |
| Các giao dịch (Transaction) | PhienBanHang (Sale), ThanhToan (Payment),<br>DatCho (Reservation) |
| Vai trò của con người | NguoiBan, KhachHang<br>PhiCong |
| Các vật chứa trong container (vật chứa) | MatHang<br>KhachDiMB (Passenger) |
| Vật chứa các vật khác (Container of other thing) | CuaHang<br>SanBay |
| Các tổ chức (Organizations) | PhòngBánHàng (SaleDepartment) |

| | |
|---|---|
| | PhòngBánVeMáyBay |
| Các sự kiện (Events) | BanHang, ThanhToan<br>CấtCánh, HạCánh |
| Các danh mục (Catalogs) | DanhMucMatHang (ProceductCatalog)<br>DanhMụcThiếtBị (PartsCatalog) |
| Các bản ghi về tài chính, dòng công việc, hợp đồng, | PhieuBan (Receipt), HoaDon<br>PhiếuĐặtChỗ |

Trong các phạm trù nêu trên, ta thấy hệ **HBH** có thêm:

**MoTaMatHang** (Thông tin về các mặt hàng), và

**DanhMucMatHang** (Danh mục về các mặt hàng).

*Lưu ý:* Trong hệ thống bán hàng đang phân tích, những khái niệm như *Phiếu bán hàng* (*Receipt*) không cần thiết phải đưa vào làm đại biểu của lớp bởi vì: nói chung, một bản ghi hay một bản báo cáo về các sự vật như *phiếu bán hàng,* không thật cần thiết trong mô hình khái niệm này vì tất cả các thông tin trong đó đề có thể suy ra được từ những khái niệm khác. Ví dụ: *phiếu bán hàng* được tạo lập từ thông tin ở **MoTaMatHang** và **ThanhToan.**

Tuy nhiên, những khái niệm như trên, nhiều khi cũng đóng những vai trò đặc biệt trong hệ thống thông tin quản lý, nó cần thiết để duy trì hệ thống. *Ví dụ*: nếu khách hàng được phép trả lại những mặt hàng kém chất lượng cho cửa hàng chẳng hạn, khi đó phải lưu lại các *phiếu bán hàng* để làm căn cứ nhận lại những mặt hàng mà khách hàng trả lại. Trong những trường hợp như thế phải đưa *phiếu bán hàng* vào danh sách các lớp cần xây dựng.

Ngoài ra, ta còn có thể dựa vào các câu hỏi sau để tìm ra các lớp đối tượng:

- Những thông tin nào cần phân tích, cần lưu trữ?
- Những hệ thống ngoài nào cần thiết cho hệ thống và ngược lại?
- Những mẫu (*pattern*), thư viện lớp, thành phần nào được sử dụng trong hệ thống?
- Hệ thống quản lý những thiết bị ngoại vi nào?
- Vai trò của các tác nhân đối với hệ thống là gì?

Những câu trả lời cho các câu hỏi trên có thể là đại biểu của lớp.

**3. Dựa vào mục đích của các ca sử dụng để xác định các lớp đối tượng** [10]

*Công việc chính trong quá trình phát triển phần mềm được hướng dẫn bởi ca sử dụng là ánh xạ từ biểu đồ ca sử dụng sang biểu đồ lớp*. Hai cách tiếp cận nêu trên tập trung phân tích các văn bản mô tả các ca sử dụng và các phạm trù cơ sở, nghĩa là theo các tiếp cận *mô hình đối tượng hướng danh từ* (*noun-oriented object modeling*) để xác định các lớp.

Một vấn đề trở ngại chung của cả hai cách làm trên là một lúc có thể phát hiện quá nhiều các đại biểu lớp, ví dụ có thể xem mỗi *danh từ* là một đại biểu lớp. Ngoài ra

theo kinh nghiệm của các nhà phân tích viên đã sử dụng UML thì hai cách tiếp cận trên còn có những vấn đề sau trong mô hình hoá đối tượng:

- Một mô tả của ca sử dụng có thể chỉ là một kịch bản. Nghĩa là nó chỉ mô tả một cách thực hiện để đạt được mục đích của ca sử dụng đó, chứ không phải tất cả. Do vậy, các lớp được xác định theo cách này bị giới hạn bởi chính mô tả này. Bởi vì, một ca sử dụng có thể được mô tả với những câu khác nhau bởi nhiều người khác nhau, dẫn tới là *một thực thể có thể được xác định tương ứng bởi nhiều tên lớp khác nhau*. Ví dụ, người mua hàng có thể được gọi là khách hàng, nhân viên có thể được gọi là cán bộ, công chức, v.v. Như thế có thể dẫn đến trường hợp có quá nhiều danh từ để lựa chọn ra những tên lớp có nghĩa trong một hệ thống.
- Nhiều phân tích viên có thể sử dụng những kiểu phân loại phạm trù các sự vật khác nhau, đôi khi dẫn đến xung đột và do vậy, rất khó thoả thuận được với nhau. Trong một số hệ thống, có thể có quá nhiều phạm trù đóng những vai trò khác nhau trong những bộ phận khác nhau. Điều này cũng gây trở ngại cho các nhà phân tích khi phải lựa chọn trong số những phạm trù đó những phạm trù nào là lớp chính xác cần xác định.

Ngoài ra, từ yêu cầu của hệ thống chúng ta có thể phát hiện được cả các chức năng và các thực thể trong hệ thống. Nhưng, mô tả ca sử dụng chỉ nhấn mạnh vào mô tả chức năng, cách thực hiện công việc. Do đó, người phân tích khó phát hiện được các thực thể một cách hiệu quả. Nghĩa là, theo các cách trên thì có thể một số lớp được phát hiện không được chấp nhận bởi các tác nhân, không thật phù hợp với bài toán ứng dụng.

Để khắc phục những nhược điểm trên, chúng ta có thể thực hiện phân tích các mục đích của ca sử dụng để xác định lớp.

Các lớp đối tượng chính là các nhóm thực thể tham gia thực hiện để đạt được mục đích của hệ thống trong thế giới thực. Các thực thể và các mục đích phải được mô tả bởi các khách hàng (NSD), chứ không phải bởi các nhà phân tích. Theo cách tiếp cận này, chúng ta có thể tạo ra biểu đồ lớp từ biểu đồ ca sử dụng theo một thuật toán gồm năm bước.

**Bước 1:** *Xác định mục đích (goal) của mỗi ca sử dụng*

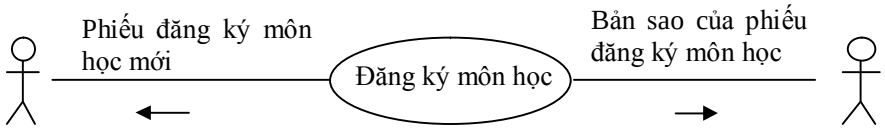
Ca sử dụng thể hiện những chức năng, nhiệm vụ của hệ thống mà NSD yêu cầu. Mục đích của NSD cũng chính là mục đích của ca sử dụng. Tác nhân của ca sử dụng có mục tiêu là sử dụng ca sử dụng để đạt được những điều mà họ muốn. *Mục đích* của ca sử dụng được xác định như sau:

> *Mục đích* của ca sử dụng là những mục tiêu mà hệ thống cần thực hiện. Mục đích thường được thể hiện dưới dạng các giá trị mà ca sử dụng cung cấp cho tác nhân hoặc những đáp ứng của ca sử dụng đó. Như vậy, mục đích là duy nhất ứng với mỗi ca sử dụng.

Để xác định các mục đích, chúng ta phải tìm cách trả lời những câu hỏi sau:

- Mục tiêu của ca sử dụng là gì?
- Ca sử dụng này cung cấp những dịch vụ nào?
- Những giá trị hay những đáp ứng nào mà ca sử dụng có thể cung cấp.

*Ví dụ*: biểu đồ ở hình 4-1 mô tả ca sử dụng "*Đăng ký môn học*" và hai tác nhân: *Bộ phận tiếp nhận* và *Phòng đào tạo.* Mục đích của *Bộ phận tiếp nhận* là nhận được *phiếu đăng ký môn học mới* của sinh viên còn mục đích của *Phòng đào tạo* là có bản sao của phiếu đăng ký đó.



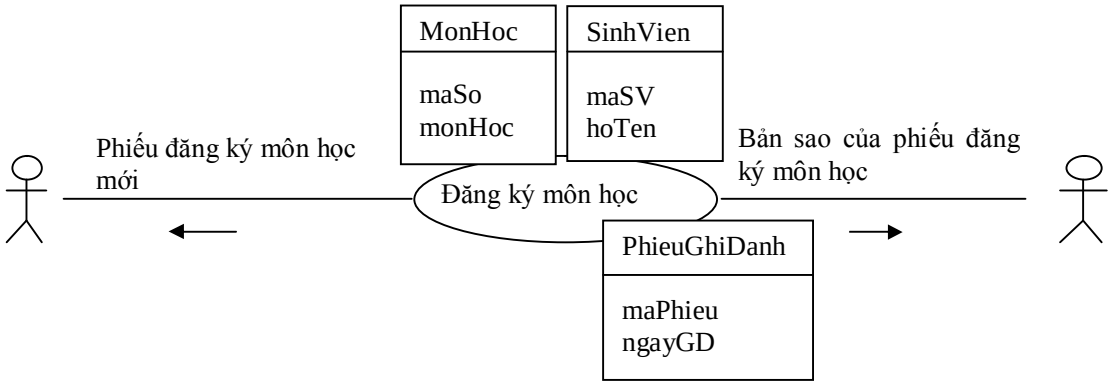

Hình 4-1 Biểu đồ ca sử dụng "Đăng ký môn học"

**Bước 2:** *Dựa vào các mục đích để xác định các thực thể*

Để thực hiện được những mục đích của ca sử dụng thì bao giờ cũng phải có các thực thể tham gia. *Thực thể* là người nào đó, cái gì đó (*something*) đóng một vai trò nhất định trong ca sử dụng để thực hiện được những mục đích của ca sử dụng. Một thực thể có thể tham gia vào nhiều ca sử dụng với những vai trò khác nhau. Nói chung, mỗi thực thể đều có những đặc tính (thuộc tính) và những hành vi ứng xử (phương thức xử lý) để phân biệt với những thực thể khác. Mặt khác, những thực thể này lại cộng tác với nhau để cùng nhau đạt được mục đích của ca sử dụng. Dựa vào các câu hỏi sau để xác định các thực thể (lớp) cho mỗi ca sử dụng:

- Những thực thể nào là cần thiết và quan trọng để thực hiện được mục đích của ca sử dụng?
- Những thuộc tính nào của thực thể là cần thiết và quan trọng để thực hiện được mục đích của ca sử dụng?

*Ví dụ:* để có *phiếu đăng ký môn học* thì phải có sự tham gia của các thực thể: *môn học, sinh viên* và *phiếu ghi danh* (trả lời câu hỏi thứ nhất). *Môn học* phải có *thuộc tính: mã số của khoá học, tên môn học, sinh viên* phải có thuộc tính: *mã SV, họ tên,* và *phiếu ghi danh* có: *mã ghi danh, ngày ghi danh, v.v.* (trả lời câu hỏi thứ hai). Từ các mục đích, chúng ta xác định được các lớp và các thuộc tính tương ứng như hình 4-2.



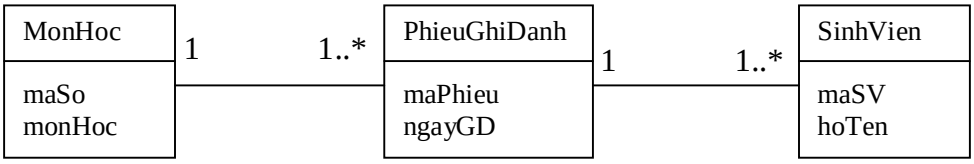
Hình 4-2 Ca sử dụng "Đăng ký môn học" và các thực thể liên quan

**Bước 3:** *Xác định các mối quan hệ (chủ yếu kết hợp) giữa các lớp*

Một ca sử dụng có thể liên quan tới nhiều thực thể như hình 4-2, những thực thể đó kết hợp với nhau để thực hiện được mục đích của ca sử dụng. Thông qua các câu hỏi tác nhân để xác định được mối liên quan giữa các lớp:

- Với mỗi thực thể, nó được sinh ra dựa vào hay bị phụ thuộc vào những thực thể khác? Nếu có, nó phải tham chiếu tới những thực thể đó.
- Với mỗi thực thể, nó có thể tác động vào hay bị tác động bởi những thực thể khác? Nếu có, nó phải tham chiếu tới những thực thể đó.

*Ví dụ:* hãy xét *phiếu ghi danh* ở hình 4-2, nó được sinh ra dựa vào sự lựa chọn *môn học* của *sinh viên*. Một *sinh viên* lại có thể *ghi danh nhiều môn học,* một *môn học* được nhiều *sinh viên ghi danh*. Sự tham chiếu giữa các lớp được biểu diễn bằng quan hệ kết hợp trong biểu đồ lớp như hình 4-3.




Hình 4-3 Mối quan hệ giữa các lớp

**Bước 4:** *Xác định các hàm thành phần thể hiện sự cộng tác của các lớp trong ca sử dụng*

Những thực thể liên quan đến một ca sử dụng thì chúng cộng tác với nhau để thực hiện một số công việc nhằm đạt được mục đích của ca sử dụng. Những công việc đó chính là các hàm thành phần của lớp. Có thể xác định được các hàm thành phần của lớp thông qua các câu hỏi các tác nhân như sau:

- Ca sử dụng này cần làm gì với mỗi thực thể liên quan với nó?
- Ca sử dụng này cần biết gì về mỗi thực thể liên quan với nó?
- Mỗi thực thể liên quan có thể đóng góp được gì trong ca sử dụng này?

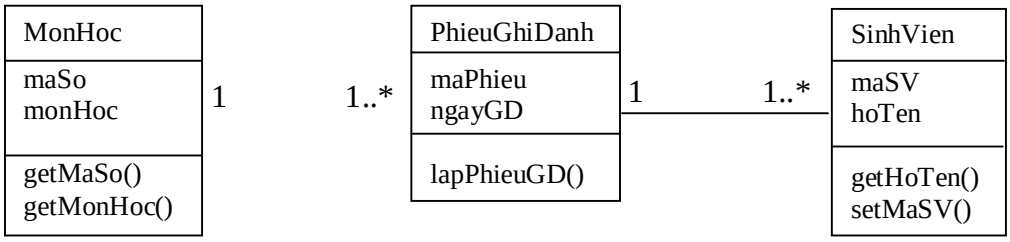
*Ví dụ:* hãy xét các tác nhân ở hình 4-2.

(i) Ca sử dụng gán *maSV* cho *sinh viên,* nó cần biết về họ tên của sinh viên. *Sinh viên* phải cho ca sử dụng biết họ và tên.

(ii) Ca sử dụng cần phải biết mã số và tên môn học.

(iii) Ca sử dụng tạo lập *phiếu ghi danh mới* với *ngày ghi danh* của sinh viên và với số ghi danh duy nhất.

Từ những phân tích như trên chúng ta có biểu đồ lớp với các hàm thành phần được bổ sung như hình 4-4.



Hình 4-4 Biểu đồ lớp

**Bước 5:** *Kiểm tra các biểu đồ ca sử dụng.*

Chúng ta có thể kiểm tra các biểu đồ được xây dựng theo các qui tắc sau:

- Kiểm tra các yêu cầu chức năng xem:
  + Tất cả các ca sử dụng có thực hiện được hết các yêu cầu chưa?
  + Mục đích của mỗi ca sử dụng có đúng như các tác nhân yêu cầu không?
- Kiểm tra các thực thể của các ca sử dụng:
  + Các thực thể trong biểu đồ lớp có cần và đủ để thực hiện các mục đích của mối ca sử dụng hay không?
  + Các thuộc tính của mỗi thực thể có phải là những cái mà ca sử dụng cần biết hay không?
  + Các hàm thành phần của lớp có cần và đủ để thực hiện các mục đích của mối ca sử dụng hay không?

Sau này, việc kiểm tra biểu đồ lớp được xây dựng tương ứng với một ca sử dụng sẽ đỡ phức tạp hơn là kiểm tra cả hệ thống.

**4. Danh sách các lớp của hệ HBH**

Bằng các phương pháp nêu trên chúng ta có được danh sách các đại biểu lớp, nhưng không phải tất cả đều có thể là đại biểu lớp. Hãy loại bỏ những đại biểu dư thừa.

- *Lớp dư thừa (Redundant Class):* khi có 2 lớp trở lên định nghĩa cho cùng một thực thể thì chỉ cần giữ lại một.
- *Lớp độc lập (Irrelevant Class):* những lớp không có quan hệ với các lớp khác trong hệ thống thì cần phải loại bỏ.
- *Lớp mơ hồ (Vague Class):* những lớp không có chức năng rõ ràng thì phải định nghĩa lại chính xác hoặc loại bỏ đi.

Bằng những cách nêu trên, hệ thống HBH có những lớp sau:

| | |
|---|---|
| HBH (POST - Point Of Sale Terminal) | DongBanHang (SaleLineItem) |
| MatHang (Items) | ThanhToan (Thanh toán – Payment) |
| CuaHang (Store) | MoTaMatHang (Product Specification) |
| PhienBanHang (Sale) | KhachHang (Customer) |
| DanhMucMatHang (ProductCatalog) | NguoiBan (Cashier) |
| NguoiQL (Manager) | |

## 4.3 Mối quan hệ giữa các lớp đối tượng

Như chúng ta đã biết, hệ thống bao giờ cũng là một thể thống nhất, nghĩa là các phần tử của hệ thống phải có quan hệ tương tác với nhau. Mô hình khái niệm do vậy,

phải bao gồm những khái niệm (các lớp đối tượng) khác nhau nhưng phải có sự tương tác, hợp tác với nhau. Nhiệm vụ tiếp theo của chúng ta là đi tìm các mối quan hệ đó.

Trong chương II chúng ta đã biết, có thể có bốn mối quan hệ giữa các lớp. Trong pha phân tích, chúng ta chủ yêu tập trung phát hiện các mối quan hệ *kết hợp, kết tập* của các lớp trong danh sách nêu trên. Trong đó quan hệ kết hợp là quan trọng nhất, nó thể hiện các mối liên hệ giữa các lớp trong hệ thống.

Trong UML, sự kết hợp (*Association*) là quan hệ giữa hai lớp, nó xác định cách các đối tượng của các lớp có thể liên kết với nhau để thực hiện công việc như thế nào. Tương tự như đối với lớp, thể hiện của lớp là các đối tượng, đối với mối quan hệ kết hợp, thể hiện của nó là sự *liên kết (link)* giữa các đối tượng của hai lớp. Nghĩa là các đối tượng của một lớp cùng chia sẻ với nhau các mối quan hệ.

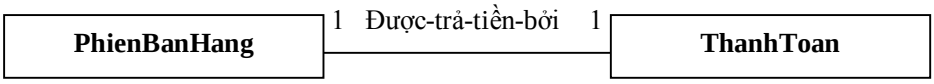
Các đối tượng có thể có nhiều loại quan hệ và vì thế, các lớp (khái niệm) cũng phải có tất cả các loại quan hệ đó trong lĩnh vực ứng dụng.

- Quan hệ kết hợp giữa hai lớp là *sự kết nối vật lý hay khái niệm* giữa các đối tượng của hai lớp đó.
- Chỉ những đối tượng có mối quan hệ kết hợp với nhau mới có thể cộng tác với nhau theo các đường liên kết được thiết lập giữa các lớp.

*Booch* đã mô tả vai trò của mối liên kết giữa các đối tượng như sau:

"*Một liên kết chỉ rõ sự kết hợp mà qua đó, một đối tượng được một đối tượng khác phục vụ hoặc một đối tượng có thể điều khiển đối tượng kia*".

Ví dụ: **PhienBanHang** và **ThanhToan** là hai lớp đã phân tích ở trên trong HBH là có quan hệ kết với nhau, mỗi lần thanh toán là để trả tiền cho một lần mua hàng. Quan hệ này được mô tả như hình 4-5.




Hình 4-5 Mối quan hệ kết hợp giữa hai lớp

### 4.3.1 Đặt tên cho các quan hệ kết hợp

- Tên của quan hệ kết hợp thường là mệnh đề động từ đơn dễ đọc và có nghĩa trong ngữ cảnh của mô hình, thể hiện được mối liên hệ giữa các lớp.
- Tên của quan hệ kết hợp thường bắt đầu bằng động từ, hay tính động từ với chữ đầu được viết hoa.
- Giữa các từ trong tên của quan hệ được nối với nhau bằng '-'.

*Ví dụ*: Tên quan hệ giữa hai lớp **PhienBanHang** và **ThanhToan** là Được-t*rả-tiền-bởi* như ở hình 4-5.

Vấn đề quan trọng đặt ra là làm thế nào để xác định chính xác các mối quan hệ giữa các lớp trong hệ thống.

### 4.3.2 Các phương pháp xác định các mối quan hệ kết hợp

Có hai phương pháp chính để xác định các mối quan hệ giữa các lớp trong hệ thống:

1. Mối quan hệ kết hợp giữa các lớp đối tượng là *cần để biết* về những thông tin liên quan đến các lớp đó. Nghĩa là dựa vào nguyên lý *"Cần để biết"*.
2. Dựa vào sự phân loại các phạm trù các quan hệ trong hệ thống.

**Xác định quan hệ kết hợp theo nguyên lý *"Cần để biết"***

Khi phân tích các yêu cầu, ta cần phải tuân theo các nguyên lý sau:

- Quan hệ kết hợp hữu ích thường cho ta sự hiểu biết về một mối quan hệ cần được duy trì trong một thời khoảng nào đó, được gọi là sự kết hợp "*cần để biết" (Need-to-know)* [10].
- Sự liên kết quan trọng giữa hai đối tượng phải thể hiện được vai trò của sự cộng tác hay sự tương tác giữa các đối tượng đó.

Dựa vào hai nguyên lý trên, trước hết là nguyên lý "*Cần để biết"* áp dụng vào ca sử dụng *"Mua hàng bằng tiền mặt",* chúng ta thấy có những quan hệ sau:

- **HBH** *Xử-lý* **PhienBanHang** để biết về lần bán hàng hiện thời, biết tổng số tiền khách hàng phải trả và để in phiếu bán hàng giao cho khách.
- **PhienBanHang** *Được-trả-tiền-bởi* **ThanhToan** để biết xem hàng vừa bán đã được trả tiền hay chưa, nó cũng liên quan đến số tiền mà khách đưa, hệ thống phải trả lại tiền dư và vấn đề in phiếu bán hàng.
- **DanhMucMatHang** *Ghi-lại* **MoTaMatHang** để tìm các thông tin mô tả về các mặt hàng như: chủng loại, giá cả, chất lượng, v.v. khi biết mã sản phẩm.

**Xác định mối quan hệ kết hợp dựa vào việc phân loại các phạm trù quan hệ**

Việc tìm các quan hệ kết hợp cũng giống như việc tìm kiếm đối với các lớp, chúng ta có thể dựa vào danh sách các phạm trù kết hợp để xác định.

**Ví dụ**: Xét hệ thống bán hàng **HBH** và "*Hệ thống đặt vé máy bay*", những phạm trù quan hệ sau cần xem xét để tìm kiếm các mối quan hệ kết hợp.

| Các phạm trù kết hợp | Các ví dụ |
|---|---|
| A là một bộ phận logic của B | DongBanHang - PhienBanHang<br>ChuyenBay - TuyenBay |
| A là một loại / lớp con / kiểu con của B | ThanhToanMat – ThanhToan<br>ChuyenBayNonStop - ChuyenBay |
| A được chứa (vật lý) trong / trên B | HBH - CuaHang<br>KhachBay – MayBay |
| A được chứa (logic) trong B | MoTaMatHang - DanhMucMatHang<br>ChuyenBay - LichBay |
| A là một mô tả của B | MoTaMatHang - MatHang<br>MoTaChuyenBay - ChuyenBay |
| A là một mục trong một giao dịch | DongBanHang – PhienBanHang |
| A là thành viên của B | NguoiBan - CuaHang<br>PhiCong – DoiBay |

| | |
|---|---|
| A sử dụng hoặc quản lý B | NguoiBan - HBH<br>PhiCong – MayBay |
| A trao đổi với B | KhachHang - NguoiBan<br>KhachBay – HangDatCho |
| Giao dịch A có quan hệ với giao dịch B | ThanhToan - PhienBanHang<br>DatCho – HuyCho |
| A là sở hữu của B | HBH - CuaHang<br>MayBay – SanBay |

*Tóm lại,* dựa vào việc phân loại các phạm trù kết hợp như trên, dựa vào kinh nghiệm, kiến thức về hệ thống và dựa vào các kết quả khảo sát hệ thống trong thực tế để liệt kê tất cả các mối quan hệ kết hợp thực giữa các lớp trong hệ thống. Các lớp trong HBH có các quan hệ như sau:

| | |
|---|---|
| DongBanHang - PhienBanHang | HBH - CuaHang |
| MoTaMatHang – DanhMucMatHang | MoTaMatHang - MatHang |
| NguoiBan – HBH | ThanhToan - PhienBanHang |
| KhachHang – HBH | NguoiBan – PhienBanHang |
| NguoiQL – HBH | PhienBanHang – CuaHang |
| MatHang – CuaHang | CuaHang – DanhMucMatHang |
| DongBanHang – MatHang | DongBanHang – MoTaMatHang |

*Lưu ý:* Trong giai đoạn phân tích, quan hệ kết hợp không cần phải mô tả về các dòng dữ liệu, các biến thể hiện, hoặc các mối kết nối của các đối tượng như trong lời giải cụ thể mà chỉ cần thể hiện được những mối liên hệ có nghĩa trong thế giới thực. Trong các pha sau, pha thiết kế và cài đặt, nhiều mối liên hệ này sẽ được cài đặt như là các đường dẫn thông tin liên kết giữa các lớp thể hiện được khả năng quan sát giữa các đối tượng trong hệ thống khi nó thực hiện.

Các mối quan hệ khác chúng ta sẽ đề cập đến ở các chương sau.

## 4.4 Biểu đồ lớp

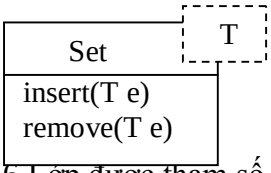
*Biểu đồ lớp mô tả quan sát tĩnh của hệ thống thông qua các lớp và các mối quan hệ của chúng*. Nó được sử dụng để hiển thị các lớp và gói các lớp cùng các mối quan hệ của chúng. Biểu đồ lớp giúp người phát triển phần mềm quan sát và lập kế hoạch cấu trúc hệ thống trước khi lập trình. Nó đảm bảo rằng hệ thống được thiết kế tốt ngay từ đầu.

### 4.4.1 Các loại lớp trong biểu đồ

Biểu đồ lớp có thể chứa nhiều loại lớp khác nhau, chúng có thể là những *lớp thông thường, lớp tham số hoá, lớp hiện thực, lớp tiện ích, và lớp metaclass*.

**Lớp tham số hoá** *(Parameterized Class)*

*Lớp tham số hoá* là lớp được sử dụng để tạo ra một họ các lớp khác. Trong những ngôn ngữ lập trình có kiểu mạnh như C++, lớp tham số hoá chính là lớp mẫu (*template).* Trong UML, có thể khai báo lớp tham số hoá (lớp mẫu) Set cho họ các lớp có các phần tử là kiểu T bất kỳ, được xem như là tham số như sau:

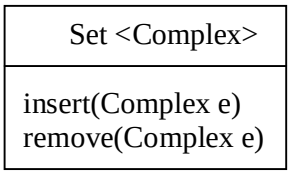



Hình 4-6 Lớp được tham số hoá

Lớp tham số hoá có thể sử dụng để thể hiện quyết định thiết kế về các giao thức trao đổi giữa các lớp. Lớp tham số hoá ít được sử dụng trong mô hình khái niệm mà chủ yếu được sử dụng trong các mô hình cài đặt, nhưng cũng chỉ khi ngôn ngữ lập trình được chọn để lập trình có hỗ trợ cơ chế lớp mẫu (*template class*) như C++ chẳng hạn. Cũng cần lưu ý là không phải tất cả các ngôn ngữ lập trình hướng đối tượng đều hỗ trợ kiểu lớp mẫu, ví dụ Java không hỗ trợ.

**Lớp hiện thực** *(Instantiated Class)*

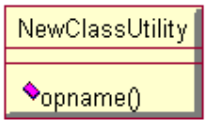
*Lớp hiện thực* là loại lớp tham số hoá mà đối số của nó là kiểu giá trị cụ thể. Như vậy, lớp tham số hoá là khuôn để tạo ra các lớp hiện thực. Ví dụ lớp **Set<Complex>** tập các số phức (*Complex)* là lớp hiện thực được biểu diễn trong UML như hình 4-7.

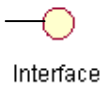



Hình 4-7 Lớp hiện thực hoá

**Lớp tiện ích** *(Class Utility)*

*Lớp tiện ích* là tập hợp các thao tác được sử dụng nhiều nơi trong hệ thống, chúng được tổ chức thành lớp tiện ích để các lớp khác có thể cùng sử dụng. Trong biểu đồ, lớp tiện ích được thể hiện bằng lớp có đường viền bóng như hình 4-8 (a).




Hình 4-8 (a) Lớp tiện ích (b) Giao diện

**Giao diện** (*Interface)*

Giao diện là tập những thao tác quan sát được từ bên ngoài của một lớp và / hoặc một thành phần, và không có nội dung cài đặt của riêng lớp đó. *Giao diện* thuộc quan sát logic và có thể xuất hiện trong cả biểu đồ lớp và biểu đồ thành phần với ký hiệu đồ hoạ như hình 4-8 (b).

### 4.4.2 Mẫu rập khuôn (stereotype) của các lớp

*Mẫu rập khuôn (Stereotype)là cơ chế mở rộng các phần tử của mô hình để tạo ra những phần tử mới.* Nó cho phép dễ dàng bổ sung thêm các thông tin cho các phần tử

của mô hình và những phần tử này được đặc tả trong các dự án hay trong quá trình phát triển phần mềm.

Rational Rose đã xây dựng một số *stereotype* như <<boundary>>, <<entity>>, <<control>>, <<interface>>, v.v., ngoài ra chúng ta có thể định nghĩa những loại kiểu mới cho mô hình hệ thống.

**Lớp biên** (*Boundary Class)*

*Lớp biên* là lớp nằm trên đường biên của hệ thống với phần thế giới bên ngoài. Nó có thể là biểu mẫu (*form),* báo cáo (*report),* giao diện với các thiết bị phần cứng như máy in, máy đọc ảnh (*Scanner),* v.v. hoặc là giao diện với các hệ thống khác. Trong UML, lớp biên được ký hiệu như trong hình 4-9 (a).

Để tìm lớp biên hãy khảo sát biểu đồ ca sử dụng, một tác nhân có thể xác định tương ứng một lớp biên. Nếu có hai tác nhân cùng kích hoạt một ca sử dụng thì chỉ cần tạo ra một lớp biên cho cả hai.

**Lớp thực thể** (*Entity Class)*

*Lớp thực thể* là lớp lưu giữ các thông tin mà nó được ghi vào bộ nhớ ngoài. Ví dụ lớp **SinhVien** là lớp thực thể. Trong UML, lớp thực thể được ký hiệu như trong hình 4-9 (b).

Lớp thực thể có thể tìm thấy trong các luồng sự kiện và biểu đồ tương tác. Thông thường phải tạo ra các bảng dữ liệu trong CSDL cho mỗi lớp thực thể. Mỗi thuộc tính của lớp thực thể trở thành trường dữ liệu trong bảng dữ liệu.

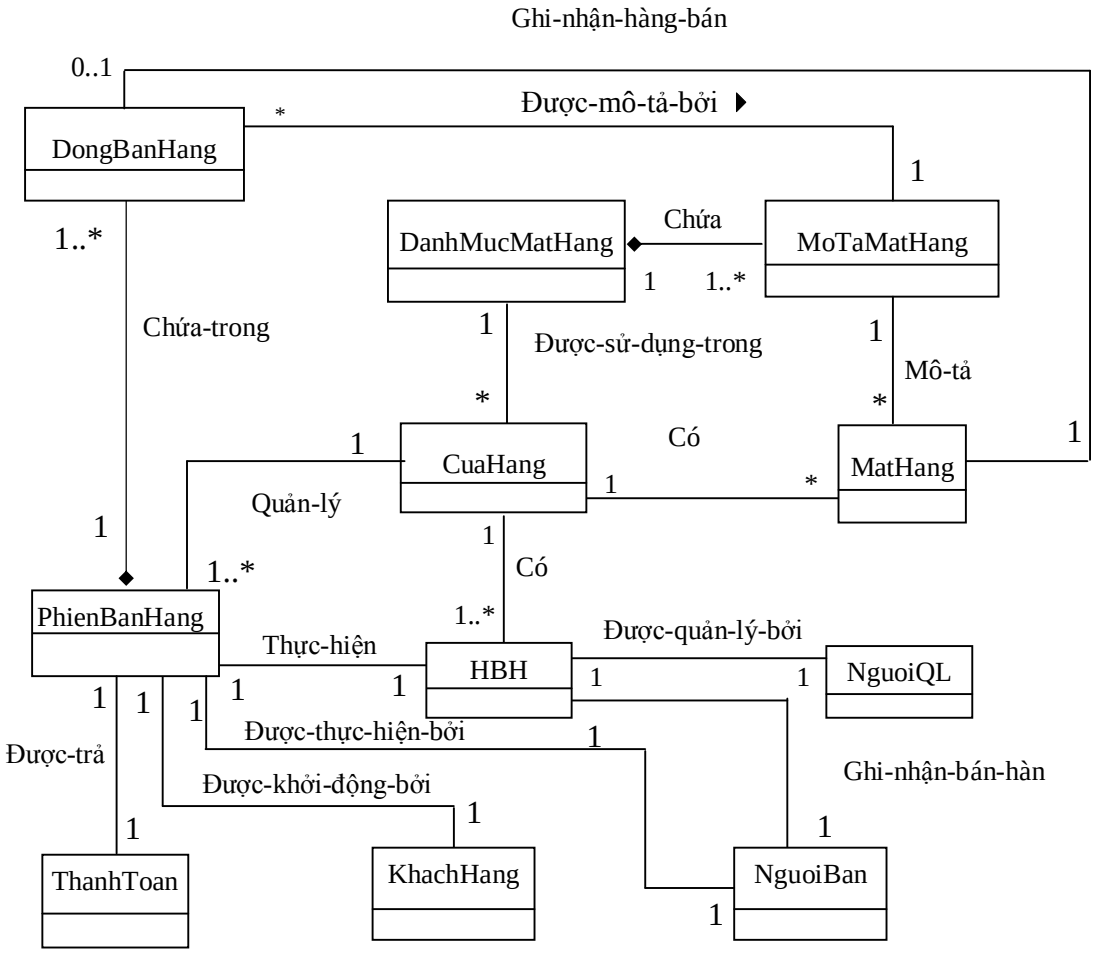
**Lớp điều khiển** (Control *Class)*

*Lớp điều khiển* là lớp làm nhiệm vụ điều phối hoạt động của các lớp khác. Thông thường mỗi ca sử dụng có một lớp điều khiển để điều khiển trình tự các sự kiện xảy ra trong nó. Chú ý, lớp điều khiển không tự thực hiện các chức năng, nhưng chúng lại gửi nhiều thông điệp cho những lớp có liên quan, do vậy còn được gọi là *lớp quản lý.* Trong UML, lớp điều khiển được ký hiệu như trong hình 4-9 (c).

Hình 4-9 (a) Lớp biên (b) Lớp thực thể (c) Lớp điều khiển

### 4.4.3 Biểu đồ lớp trong Hệ HBH

Trong số những quan hệ kết hợp đã phát hiện ở mục 4.3, dễ nhận thấy lớp **DanhMucMatHang** *chứa các* **MoTaMatHang**, do vậy giữa hai lớp này có quan hệ kết tập. Tương tự, trong mỗi **PhienBanHang** *có một số* **DongBanHang**, nghĩa là giữa hai lớp này cũng sẽ có quan hệ kết tập. Kết hợp tất cả các mối quan hệ giữa các lớp như đã phân tích ở trên, chúng ta xây dựng được biểu đồ lớp như sau:




Hình 4-10 Biểu đồ lớp của HBH

## 4.5 Thuộc tính của lớp

*Thuộc tính của lớp (attribute)* mô tả các đặc tính của các đối tượng trong lớp đó. *Ví dụ*: *mỗi khách hàng của lớp* **KhachHang** có *họ và tên, địa chỉ, số tài khoản, v.v.* Ở mỗi thời điểm thuộc tính của một đối tượng của một lớp là *giá trị dữ liệu logic biểu diễn cho tính chất tương ứng* của đối tượng, được gọi là *giá trị thuộc tính của* đối tượng tại thời điểm đó.

Mỗi thuộc tính có:

*+ Tên của thuộc tính,*

*+ Kiểu xác định các loại giá trị mà thuộc tính mô tả,*

*+ Giá trị mặc định (khởi đầu) cho mỗi thuộc tính.*

*Kiểu của thuộc tính* nói cho ta biết về *loại giá trị kiểu số nguyên (Integer, Int), số thực (Real, Float), giá trị logic (Boolean), ký tự (Character), thời gian (Time), v.v. được gọi là các kiểu nguyên thuỷ (primitive type).* Ngoài các kiểu nguyên thuỷ, người phát triển hệ thống có thể tạo ra những kiểu mới tuỳ ý.

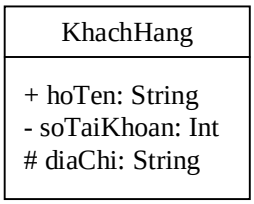
Thuộc tính của lớp còn có thêm đặc tính để thể hiện khả năng nhìn thấy được hay đặc tính quản lý khả năng truy nhập của thuộc tính đối với các đối tượng khác, gọi chung là phạm vi quan sát của thuộc tính. Đó là các đặc tính được khai báo trong lớp bằng các ký hiệu:

'+' đứng trước thuộc tính trong UML, hoặc biểu tượng ổ khoá nhưng không bị khoá trong Rose để thể hiện thuộc tính này là công khai (*public),* mọi đối tượng đều nhìn thấy được,

'#' đứng trước thuộc tính trong UML, hoặc biểu tượng ổ khoá và có chìa để bên cạnh trong Rose để thể hiện thuộc tính này là được bảo vệ (*protected),* những đối tượng có quan hệ kế thừa có thể nhìn thấy được,

'-' đứng trước thuộc tính trong UML, hoặc biểu tượng khoá bị khoá và chìa bị cất đi trong Rose để thể hiện thuộc tính này là sở hữu riêng (private), chỉ bản thân đối tượng của một lớp nhìn thấy được.

Trường hợp những thuộc tính không có ký hiệu đặc tính phạm vi nào đứng trước thì được xem là mặc định, nghĩa là những thuộc tính có thể quan sát được đối với các lớp trong cùng gói.




Hình 4-11 Các thuộc tính của lớp

Thuộc tính *hoTen* có kiểu *String* và là công khai, *soTaiKhoan* có kiểu Int (các số nguyên) *là riêng* còn *diaChi* là được bảo vệ, có kiểu *String.*

*Lưu ý:* Khi cài đặt chương trình thì phạm vi quan sát của thuộc tính còn tuỳ thuộc vào những qui định khác nhau của ngôn ngữ lập trình được lựa chọn. Ví dụ đặc tính mặc định, hay được bảo vệ của thuộc tính trong C++ và Java là khác nhau.

### 4.5.1 Tìm kiếm các thuộc tính

Với mỗi lớp được tạo ra trong biểu đồ lớp như hình 4-6, chúng ta mong muốn tìm được những thuộc tính sao cho:

- *Đầy đủ (Complete):* chứa đựng tất cả các thông tin về đối tượng của lớp,

- *Tách biệt hoàn toàn (full factored):* mỗi thuộc tính thể hiện được một đặc tính khác nhau của đối tượng,
- *Độc lập với nhau (mutually independent):* đối với mỗi đối tượng, các giá trị của các thuộc tính là độc lập với đối tượng khác, tốt nhất là loại bỏ những thuộc tính có thể được suy dẫn từ những thuộc tính khác.
- *Liên quan đến các yêu cầu và các ca sử dụng:* các thuộc tính được đưa vào lớp phải được xác định trên cơ sở các yêu cầu thực hiện công việc hoặc cần để tổ chức, lưu trữ các thông tin về đối tượng.

Trong mô hình hoá các khái niệm *có thể có những nhầm lẫn giống nhau là nhiều khi ta sử dụng thuộc tính để biểu diễn cho những cái mà đáng lý ra phải sử dụng khái niệm lớp hay các mối quan hệ liên kết để thể hiện nó.*

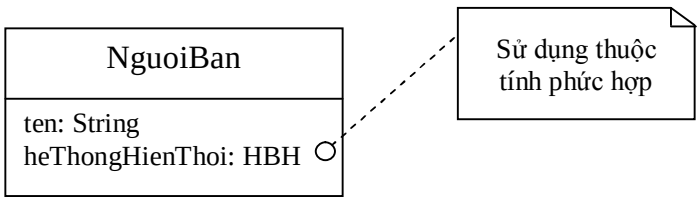
Sau đây chúng ta xét một số lưu ý nhằm xác định chính xác các thuộc tính cho lớp nhằm đáp ứng các nguyên tắc cơ bản và tránh được những sai sót trên.

**Đảm bảo các thuộc tính đơn giản**

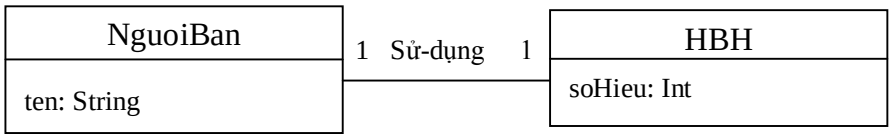
Các thuộc tính trong mô hình lớp phải là những thuộc tính đơn giản hoặc là những *kiểu dữ liệu thuần tuý (pure data type)*:

- Một cách trực quan, thuộc tính đơn giản nhất là những thuộc tính có giá trị kiểu dữ liệu nguyên thuỷ (*primitive data type): Boolean, Number, String (Text), Time, v.v.*
- Một số các kiểu phổ dụng bao gồm: *DiaChi (Address), MauSac (Color), HinhHoc (Geometrics), SoDienThoai (PhoneNumber), SoBaoHiem (SocialSecurityNumber), MaSanPham (Universa ProductCode), các kiểu liệt kê (EnumeratedTypes), v.v.*

Thông thường, nên cố tránh những thuộc tính phức tạp và nên thay vào đó là những quan hệ liên kết giữa các lớp. *Ví dụ*: hãy xét lớp **NguoiBan**, có thể đưa thuộc tính *heThongHienThoi* vào để thể hiện là người bán hàng đang sử dụng hệ thống bán hàng nào đó. Bởi vì hệ thống **HBH** là lớp, nên nó là kiểu phức tạp. Do vậy, cách làm như thế không phải là tốt (hình 4-12 (a)). Để thể hiện được mối quan hệ này, ta có thể sử dụng mối quan hệ kết hợp như hình 4-12 (b).




Hình 4-12 (a) Trường hợp thiết kế lớp không tốt




Hình 4-12 (b) Tốt hơn là chuyển thuộc tính phức thành quan hệ kết hợp

Qui tắc hướng dẫn đầu tiên là:

> *1. Liên kết các khái niệm với nhau bằng quan hệ kết hợp, không bằng các thuộc tính phức hợp.*

**Sử dụng giá trị dữ liệu thuần tuý**

Nói một cách tổng quát, *các thuộc tính phải có giá trị dữ liệu thuần tuý hoặc kiểu Data Type trong UML*, trong đó việc xác định duy nhất không có nhiều ý nghĩa trong ngữ cảnh của mô hình hệ thống. *Ví dụ*: đối với những kiểu dữ liệu nguyên thuỷ thì:

+ Không cần tách biệt các giá trị số giống nhau,
+ Không cần tách biệt các thể hiện của *SoDienThoai* mà chúng có thể có cùng số,
+ Không cần tách biệt hai địa chỉ vì chúng có thể giống nhau, v.v.

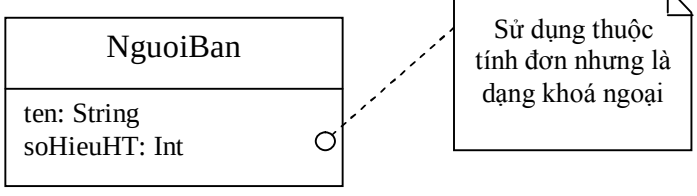
Song, đối với các đối tượng thì việc phân biệt chúng (thông qua định danh) lại *có ý nghĩa và bắt buộc.* Ví dụ: hai khách hàng có thể cùng tên *"Nguyễn Lam"*, nhưng trong hệ thống phải được xác định riêng biệt bằng định danh (*Identity*).

Qui tắc hướng dẫn thứ hai là:

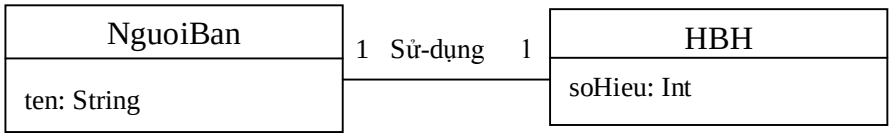
> *2. Phần tử của kiểu dữ liệu thuần tuý có thể được biểu diễn trong một thuộc tính của một lớp, mặc dù nó cũng có thể được sử dụng như một khái niệm (lớp) tách biệt trong mô hình.*

**Không sử dụng thuộc tính như khoá ngoại**

*Các thuộc tính không nên sử dụng để liên kết các khái niệm lại với nhau trong mô hình khái niệm, mà chỉ được sử dụng để lưu giữ các thông tin chính các đối tượng của lớp có các thuộc tính đó*. Trong thiết kế mô hình CSDL quan hệ thì nguyên lý này bị vi phạm vì, người ta thường sử dụng thuộc tính là khoá ngoại (*Foreign Key*) để kết nối hai kiểu thực thể với nhau. Ví dụ: trong hình 4-13 (a), lớp **NguoiBan** có thuộc tính *soHieuHT* được xem như là khoá ngoại để thể hiện một người bán hàng sử dụng hệ thống bán hàng có *số hiệu xác định*. Mặc dù thuộc tính này là thuộc tính đơn (không vi phạm hướng dẫn 1), nhưng nó là khoá ngoại vì thế vẫn không tốt và do đó, nên thay bằng quan hệ kết hợp như hình 4-13 (b).



Hình 4-13 (a) Trường hợp sử dụng khoá ngoại




Hình 4-13 (b) Tốt hơn là chuyển thuộc tính phức thành quan hệ kết hợp

Từ đó ta có qui tắc hướng dẫn thứ ba.

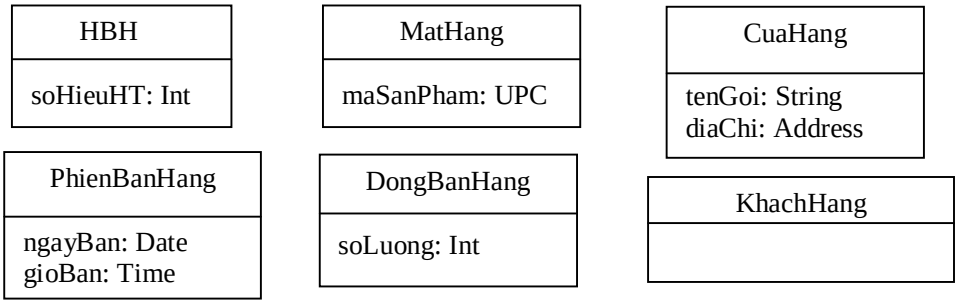
> *3. Nên kết nối các khái niệm với nhau bằng các quan hệ kết hợp, không sử dụng thuộc tính.*

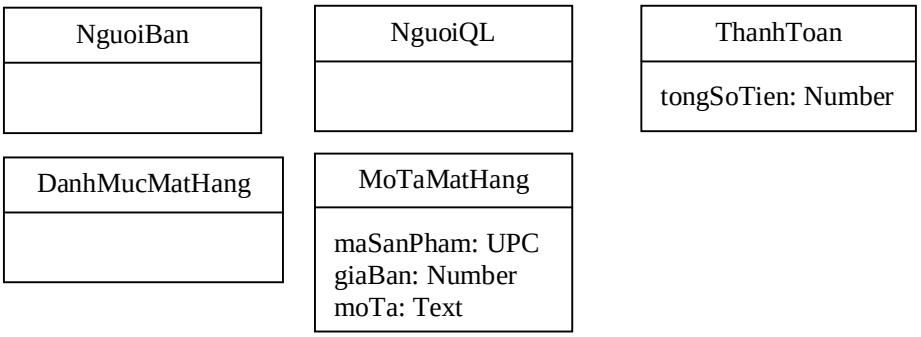
**Tìm thuộc tính ở đâu và như thế nào?**

- Đọc kỹ các mô tả bài toán, nghiên cứu các hồ sơ các chức năng hệ thống, các đặt tả ca sử dụng, các kịch bản để *tìm tất cả những thông tin, dữ liệu cần phải lưu trữ, xử lý và cập nhật.* Các mục này thường là các *danh từ, hoặc mệnh đề danh từ đơn, được xem như là đại biểu của các thuộc tính. Ví dụ:* khi xem xét luồng các sự kiện: "Đối với mỗi mặt hàng, người bán nhập vào mã sản phẩm thông qua máy đọc thẻ và số lượng hàng mà khách hàng chọn mua". Như vậy, trong lớp **MatHang** tất nhiên phải có thuộc tính *maSanPham,* và qua nó có thể xác định được *tên gọi, chủng loại, giá bán*, v.v là những thuộc tính của **MoTaMatHang**.
- *Sử dụng các qui tắc hướng dẫn* nêu trên để xác định chính xác các thuộc tính: đặc tính xác định phạm vi quan sát, tên gọi, kiểu và giá trị khởi đầu (nếu có) của mỗi thuộc tính.
- *Đọc những giả thiết, sự phân loại hay những qui ước cần áp dụng cho hệ thống hiện thời* để khẳng định lại những thuộc tính của từng lớp.
- *Gán các thuộc tính cho các lớp đối tượng* trong biểu đồ lớp.

### 4.5.2 Các thuộc tính của các lớp trong HBH

Theo cách hướng dẫn nêu trên, chúng ta có thể xác định được các thuộc tính cho các lớp để thực hiện ca sử dụng "*Mua* hành bằng tiền mặt" trong biểu đồ ở hình 4-10 như sau:






Hình 4-14 Các thuộc tính của các lớp

Đó là những thuộc tính được xác định để thực hiện ca sử dụng *"Mua hàng bằng tiền mặt"*. Tương tự, xem xét các chức năng của hệ thống, nghiên cứu các ca sử dụng còn lại và dựa vào những tài liệu khác để xác định đầy đủ và chính xác các thuộc tính cho các lớp đối tượng.

Biểu đồ các lớp với các thuộc tính mô tả *cấu trúc tĩnh của hệ thống*. Nó mô tả mối liên kết có cấu trúc giữa các mục dữ liệu được thao tác, xử lý trong hệ thống. Nó cũng mô tả cách các thông tin được phân chia thành từng phần cho các đối tượng, chỉ ra cách các đối tượng được chia thành các lớp và thể hiện mối quan hệ giữa các đối tượng là gì.

*Tóm lại,* trong mô hình khái niệm chúng ta tập trung mô tả:

- Những khái niệm là các lớp đối tượng trong hệ thống,
- Các mối liên kết giữa các lớp,
- Các thuộc tính của các lớp.

## 4.6 Ghi nhận trong từ điển thuật ngữ

*Từ điển thuật ngữ (Glossary)* là loại tài liệu đơn giản để định nghĩa các hạng mục (*term*), các thuật ngữ được sử dụng trong quá trình phát triển phần mềm. Từ điển thuật ngữ hay *từ điển mô hình,* liệt kê và định nghĩa các hạng mục một cách rõ ràng, dễ hiểu nhằm phục vụ tốt hơn trong giao tiếp, trao đổi giữa các thành viên trong dự án và giảm thiểu được những may rủi đáng tiếc do sự hiểu lầm có thể gây ra.

Thông thường từ điển thuật ngữ được xây dựng từ pha đầu tiên, pha định nghĩa bài toán và khảo sát các ca sử dụng, các hạng mục đã được tạo ra, và sau đó tiếp tục bổ sung, làm mịn hơn ở các pha tiếp theo. Nghĩa là việc xây dựng từ điển thuật ngữ được thực

hiện song hành với các công việc đặc tả yêu cầu, xây dựng ca sử dụng, thiết lập mô hình khái niệm, biểu đồ tuần tự, cộng tác, v.v.

Không có mẫu chính thức nào có thể áp dụng cho việc mô tả từ điển thuật ngữ. Sau đây là một ví dụ về từ điển thuật ngữ trong hệ HBH.

| **Hạng mục** | **Phạm trù** | **Chú thich** |
|---|---|---|
| *Bán hàng* | *Ca sử dụng* | *Mô tả quá trình bán hàng cho khách trong một cửa hàng* |
| MoTaMatHang.moTa: Text | Thuộc tính | Mô tả ngắn gọn về mặt hàng ở trong lớp MoTaMatHang |
| MatHang | Lớp (kiểu) | Hàng để bán trong một cửa hàng |
| ThanhToan | Lớp (kiểu) | Lớp làm nhiệm vụ thu tiền mà khách hàng trả khi mua hàng |
| MoTaMatHang.giaBan: Number | Thuộc tính | Giá bán của mặt hàng ở trong MoTaMatHang |
| DongBanHang.soLuong: Int | Thuộc tính | Số lượng một mặt hàng mà khách mua |
| PhienBanHang | Lớp (kiểu) | Một giao dịch bán hàng |
| DongBanHang | Lớp (kiểu) | Một dòng trong phiên giao dịch bán hàng |
| CuaHang | Lớp (kiểu) | Nơi có các giao dịch bán hàng |
| PhienBanHang.tongSoTien: Number | Thuộc tính | Tổng số tiền khách phải trả trong một lần mua hàng |
| ThanhToan.tongSo: Number | Thuộc tính | Tổng số tiền cần phải thu của khác |
| MoTaMatHang.maSanPham: UPC | Thuộc tính | Mã sản phẩm ở trong MoTaMatHang |

## 4.7 Thực hành trong Rational Rose

Sử dụng Rational Rose ([8], [11]) để thực hiện những công việc sau:

1. Tạo lập và huỷ bỏ biểu đồ lớp. Lưu ý: trong Rose, biểu đồ lớp được thiết lập trong quan sát logic (*Logical View),* khi tạo lập mô hình mới biểu đồ lớp *Main* được tạo ra ngay trong *Logical View.* Ta có thể tạo ra một số biểu đồ khác trong quan sát này để mô tả mô hình cấu trúc tĩnh của hệ thống.
2. Bổ sung thêm các loại lớp: lớp thông thường, lớp hiện thực, lớp tiện ích, v.v.
3. Đặc tả các lớp: tên lớp, chọn *stereotype*, đặc tính xác định phạm lớp *(Public, Protected, Private, Package* hay *Implementation),* đặc tính lưu trữ của l*ớp (Persistent, Transient), v.v.*
4. Tạo lập và huỷ bỏ gói (*Package*)

5. Đưa các thuộc tính vào lớp: tên, kiểu và gán trị khởi đầu, gán các thuộc tính lưu trữ (*By Value, By Reference, Unspecified),* gán thuộc tính tĩnh (*Static),* gán thuộc tính suy dẫn (*Derived),*
6. Thiết lập các mối quan hệ giữa các lớp trong biểu đồ: tên gọi và hướng của quan hệ, gán *stereotype, các vai trò Role* cho quan hệ, phạm vi của quan hệ *(Public, Protected, Private, Package* hay *Implementation)* và các thuộc tính khác như *Static, Friend, By Value, By Reference, Unspecified, Link Element, Key / Qualifier, v.v.*

Thực hành các chức năng trên để xây dựng biểu đồ lớp ở hình 4-10 và sau đó bổ sung thêm các thuộc tính từ hình 4-14.

*Lưu ý:* Rational Rose 4.0 có thể nạp phiên bản demo với ngôn ngữ lập trình C++ và trong môi trường Window 95 từ http://www.ratinal.com/demos. Rose là công cụ tốt giúp ta thể hiện được những kết quả phân tích, thiết kế hệ thống hướng đối tượng. Tuy nhiên còn một số ký pháp của UML chưa được thể hiện trong Rose.

- Rose không cho phép vẽ hình (hộp) giới hạn đường biên của hệ thống.
- *Biểu đồ lớp* trong Rose thiếu ký hiệu giống như hình tròn cho lớp giao diện (interface) và thiếu các ký hiệu biểu diễn cho các loại quan hệ kết tập khác nhau.
- *Trong biểu đồ trình tự* không có ký hiệu cho sự kiện tạo lập và huỷ bỏ đối tượng.

## Câu hỏi và bài tập

4.1 Điền vào chỗ chống của những câu sau:

+ Khái niệm là ý tưởng, . . .

+ Sự khác biệt chính giữa phân tích hướng đối tượng và phân tích có cấu trúc là sự phân rã hệ thống thành . . . .

+ Trong UML mô hình khái niệm của một hệ thống được mô tả bởi . . .

. . . . . trong các mô tả văn bản đó có thể là đại biểu của lớp hoặc thuộc tính của lớp.

+ Có thể dựa vào sự phân loại các phạm trù khái niệm để . . . . . cho hệ thống.Quan hệ kết hợp giữa hai lớp là . . . . . các đối tượng của hai lớp đó.

+ Các đối tượng có mối quan hệ . . . . với nhau mới có thể cộng tác với nhau theo các đường . . . . . giữa các lớp.

+ Lớp trừu tượng là . . . . còn lớp cụ thể là . . . .

+ Trong hệ thống hướng đối tượng, các đối tượng được xác định duy nhất . . . .

4.2 Nêu một số phương pháp chính để phát hiện các lớp đối tượng trong giai đoạn phân tích hệ thống theo cách tiếp cận hướng đối tượng.

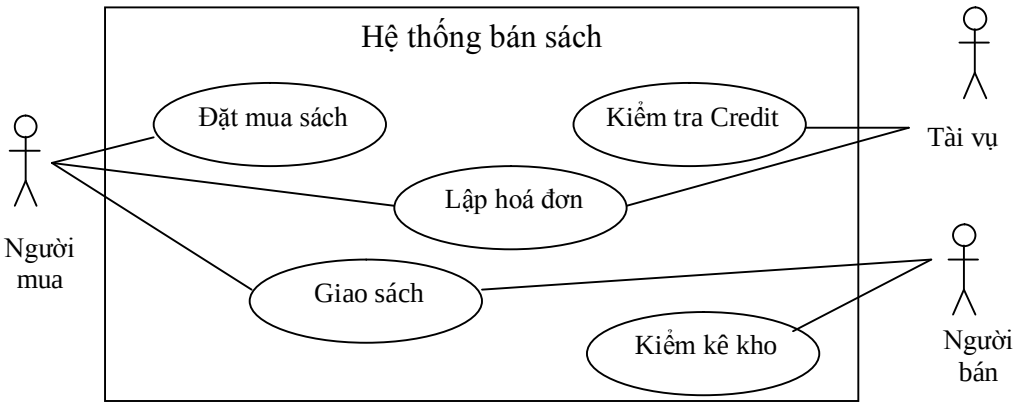
4.3 Mô tả trong UML để thể hiện: "Mỗi sinh viên có thể theo học nhiều nhất là 6, ít nhất là 4 môn học và mỗi môn học có nhiều nhất là 30 sinh viên có thể ghi danh.

4.4 Xác định các lớp và thiết lập biểu đồ lớp cho hệ thống "Quản lý thư viện" (Tiếp theo của bài tập 3.4).

4.5 Xây dựng mô hình khái niệm cho "Hệ thống rút tiền tự động ATM (Automatic Teller Machine)" (Tiếp theo của bài toán 3.5).

4.6 Xây dựng mô hình khái niệm cho hệ thống "Mô phỏng hệ thống thang máy cho các nhà cao tầng" (Tiếp theo của bài toán 3.6).

4.7 Áp dụng phương pháp phân tích các mục đích của các ca sử dụng để chuyển biểu đồ ca sử dụng ở hình sau sang biểu đồ lớp.




*4.8* Chọn từ danh sách dưới đây những thuật ngữ thích hợp để điền vào các chỗ [(…)] trong đoạn văn mô tả về mục tiêu của phương pháp hướng đối tượng.

Mục tiêu chính của [(1)] là phân tách hệ thống thành các đối tượng hoặc xác định các [(2)], đó là những [(3)] mà chúng ta biết rõ về chúng. Mô hình [(2)] là cách biểu diễn các [(4)] trong phạm vi của bài toán.

*Chọn câu trả lời:*

a. khái niệm

b. thực thể

c. phương pháp hướng đối tượng

d. sự vật

# CHƯƠNG V

# MÔ HÌNH ĐỘNG THÁI: CÁC BIỂU ĐỒ TƯƠNG TÁC VÀ HÀNH ĐỘNG TRONG HỆ THỐNG

Chương này trình bày về mô hình mô tả hành vi của hệ thống:

- ✓ Mô tả hành vi của các đối tượng: Biểu đồ trạng thái, trình tự, cộng tác và biểu đồ hành động.
- ✓ Các sự kiện, trạng thái và thao tác của các đối tượng trong hệ thống,
- ✓ Sự trao đổi, tương tác giữa các đối tượng,
- ✓ Xây dựng các biểu đồ trạng thái và biểu đồ trình tự mô tả các hoạt động của hệ thống phần mềm.

## 5.1 Mô hình hoá hành vi hệ thống

Tất cả các hệ thống đều có cấu trúc tĩnh và hành vi động cần được mô hình hoá. UML cung cấp các biểu đồ để thể hiện được cả hai phương diện đó:

- *Cấu trúc tĩnh được mô tả bởi*: biểu đồ lớp, các đối tượng và các mối quan hệ của chúng.
- *Hành vi động được mô tả bởi*: biểu đồ trạng thái, trình tự, cộng tác và biểu đồ hành động.

Các đối tượng trao đổi với nhau bằng cách gửi các thông điệp (*messages)* để thực hiện các nhiệm vụ trong hệ thống. Sự trao đổi hay còn gọi là *sự tương tác (Interaction)* trong hệ thống được thể hiện trong các biểu đồ:

(i) *Biểu đồ trạng thái (StateDiagram):* mô tả các trạng thái, hành vi của các đối tượng. Biểu đồ trạng thái bao gồm những thông tin về những trạng thái khác nhau của các đối tượng, thể hiện các đối tượng chuyển từ trạng thái

này sang trạng thái khác như thế nào, hành vi ứng xử của mỗi đối tượng khi có các sự kiện (*events)* xảy ra để làm thay đổi trạng thái.

(ii) *Biểu đồ trình tự (Sequence Diagram):* mô tả sự trao đổi, tương tác của các đối tượng với nhau theo *trình tự thời gian.* Biểu đồ trình tự bao gồm các phần tử biểu diễn cho các đối tượng, các thông điệp được gửi và nhận trình tự theo thời gian để thực hiện các ca sử dụng của hệ thống.

(iii) *Biểu đồ cộng tác (Collaboration Diagram):* mô tả sự tương tác của các đối tượng với nhau theo ngữ cảnh và không gian công việc.

(iv) *Biểu đồ hành động (Activity Diagram):* mô tả cách các đối tượng tương tác với nhau nhưng nhấn mạnh về công việc, xác định các hành động và thứ tự thực hiện những hành động đó.

Xây dựng biểu đồ tương tác là thực hiện việc gán trách nhiệm cho các đối tượng. Từ biểu đồ tương tác, người thiết kế có thể phát hiện thêm các lớp, các thao tác cần thực hiện của mỗi lớp, v.v. Do vậy, biểu đồ tương tác trở thành nền tảng cho các bước còn lại của quá trình phát triển phần mềm.

*Nhận xét:* Không phải tất cả các hệ thống đều cần cả bốn biểu đồ trên để mô tả hành vi ứng xử của các đối tượng trong các ca sử dụng. Số các biểu đồ tương tác cần xây dựng hoàn toàn phụ thuộc vào mức độ khó, phức tạp của bài toán ứng dụng. Một số người sử dụng biểu đồ trình tự, biểu đồ trạng thái trong pha phân tích để mô tả hoạt động của hệ thống, sau đó xây dựng biểu đồ cộng tác, biểu đồ hành động để phục vụ cho việc thiết kế chi tiết các thành phần của hệ thống ([4], [6], [7]). Đối với những hệ thống tương đối đơn giản thì chỉ cần biểu đồ trình tự và biểu đồ trạng thái là đủ.

### 5.1.1 Các sự kiện và hành động của hệ thống

Trong quá trình tương tác với hệ thống, các tác nhân gây ra các *sự kiện* (*Event*) cho làm hệ thống hoạt động và yêu cầu hệ thống phải thực hiện một số thao tác để đáp ứng các yêu cầu của những tác nhân đó. *Các sự kiện phát sinh bởi các tác nhân có liên quan chặt chẽ với những hoạt động mà hệ thống cần thực hiện.* Điều này suy ra là chúng ta phải xác định được các hoạt động của hệ thống thông qua các sự kiện mà các tác nhân gây ra.

Vậy, *sự kiện là một hành động (Action) kích hoạt* hệ thống để nó hoạt động, hoặc tác động lên hệ thống để nó hoạt động tiếp theo một cách nào đó. Nói cách khác, *sự kiện là cái gì đó xảy ra và kết quả là nó có thể gây ra một số hoạt động sau đó của hệ thống.* Ví dụ: sau khi nhập vào hết các mặt hàng mà khách đã chọn mua, *người bán hàng* nhấn phím *"Kết thúc",* thì hệ thống chuyển sang thực hiện chức năng *thanh toán* với khách mua hàng. Việc người bán hàng nhấn phím *"Kết thúc"* chính là sự kiện làm cho hệ thống chuyển sang trạng thái khác.

Các sự kiện có thể là *độc lập hoặc có liên hệ với nhau*. Ví dụ: *Nhập thông tin về các mặt hàng* và *Thanh toán* là hai sự kiện phụ thuộc, sự kiện sau phải xảy ra sau sự kiện thứ nhất, còn sự kiện *Trả tiền mặt* và *trả bằng séc* là độc lập với nhau.

Những sự kiện độc lập có thể là những sự kiện đồng thời. Bởi vì những sự kiện này không phụ thuộc vào nhau nên có thể xảy ra trong cùng một thời điểm. Ví dụ *Hiển thị số tiền dư trả lại cho khách* và *Cập nhật các mặt hàng* trong hệ thống HBH là hai sự kiện độc lập với nhau và có thể xảy ra đồng thời.

Các sự kiện cũng có thể chia thành hai loại: *các sự kiện bên trong và các sự kiện bên ngoài.*

- *Sự kiện bên trong (Internal Events)* là sự kiện được tạo ra ngay bên trong hệ thống, ở trong một đối tượng và được kích hoạt bởi đối tượng khác.
- *Sự kiện ngoài* (*External Events*) là sự kiện được tạo ra ở bên ngoài phạm vi của hệ thống. *Sự kiện vào của hệ thống (System Input Event)* là những sự kiện ngoài tác động vào hệ thống và do các tác nhân tạo ra.

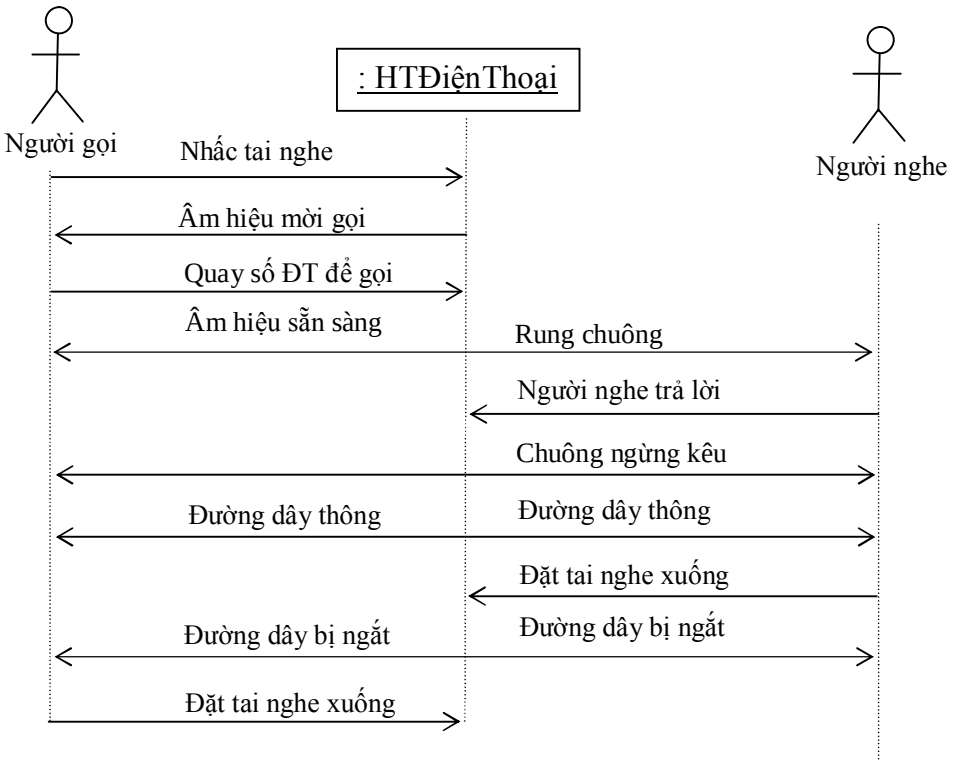
*Hành động của hệ thống (System Operation)* là hoạt động mà hệ thống phải thực hiện để trả lời cho những sự kiện vào. Một số hoạt động của hệ thống có thể tạo ra những *sự kiện ra* cho các tác nhân để thông báo những sự kiện tiếp theo của hệ thống có thể xảy ra, hoặc nhắc các tác nhân phải hành động như thế nào để có những thông tin mong muốn.

Điều hiển nhiên là: *Các sự kiện vào (input) sẽ kích hoạt hệ thống hoạt động và hệ thống hoạt động là để trả lời cho các sự kiện vào mà các tác nhân tạo ra.*

Các sự kiện và hoạt động của hệ thống thường được sử dụng để mô tả hình thức các kịch bản cho ca sử dụng. *Ví dụ*, khảo sát kịch bản của ca sử dụng "*Gọi điện thoại*", trong đó có hai tác nhân là *người gọi* và *người nghe.* Dãy các sự kiện của ca sử dụng này được mô tả như sau:

| Các sự kiện vào | Các sự kiện ra |
|---|---|
| 1. Người gọi nhấc tai nghe | 2. Tiếng bíp bíp báo hiệu máy điện thoại sẵn sàng để bắt đầu trao đổi đàm thoại |
| 3. Người gọi quay số (ví dụ 5652 288) | 4. Tín hiệu điện thoại được nối với người nghe. |
| | 5. Điện thoại của người được gọi rung chuông (nếu không bận đàm thoại) |
| 6. Người nghe nhấc ống tai nghe và trả lời | 7. Chuông ngừng kêu. |
| | 8. Đường dây điện thoại được kết nối để hai người đàm thoại với nhau. |
| 9. Người nghe đặt tai nghe xuống | 10. Đường dây bị ngắt. |
| 11. Người gọi đặt tai nghe xuống | |

Các sự kiện vào là do *người gọi* tạo ra, các sự kiện ra lại tác động đến *người nghe.* Hệ thống điện thoại sẽ hoạt động để trả lời cho các sự kiện vào đồng thời phát sinh ra các sự kiện ra. Dãy các sự kiện và hoạt động của *Hệ thống điện thoại* được mô tả một cách trực quan hơn như sau:




Hình 5-1 Biểu đồ vết các sự kiện khi thực hiện ca sử dụng "*Gọi điện thoại"*

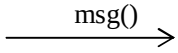
### 5.1.2 Sự trao đổi thông điệp giữa các đối tượng

Trong các biểu đồ tương tác, các đối tượng trao đổi với nhau bằng các thông điệp. Các đối tượng thường được gửi, nhận theo:

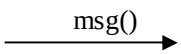
- Các giao thức trao đổi tin (*Communication Protocol*),
- Hay các lời gọi hàm, một đối tượng gọi một hàm của đối tượng khác để xử lý các yêu cầu.

Có thể thực hiện việc trao đổi thông tin trên mạng, hoặc trong một máy tính và phần lớn thực hiện trong thời gian thực (*Real-Time*).

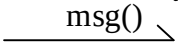
Các thông điệp có ba kiểu trao đổi chính:

1. *Kiểu đơn giản* (*Simple*): được ký hiệu là 

Biểu diễn cho dòng điều khiển để chuyển thông điệp *msg()* từ đối tượng này sang đối tượng khác mà không cần biết chi tiết về sự trao đổi thông tin.

2. *Kiểu đồng bộ (Synchronous):* được ký hiệu là 


   Biểu diễn cho dòng điều khiển được đồng bộ hoá, nghĩa là khi có nhiều thông điệp gửi đến (nhận được) thì thông điệp trước phải được xử lý xong và sau khi đã kết thúc công việc thì thông điệp tiếp theo mới được xử lý.

3. *Kiểu dị bộ (Asynchronous):* được ký hiệu là 


Biểu diễn cho dòng điều khiển thông điệp không cần đồng bộ, nghĩa là khi có nhiều thông điệp gửi đến (hay nhận được) thì các thông điệp đó được xử lý mà không cần biết những thông điệp khác đã kết thúc hay chưa.

## 5.2 Biểu đồ trình tự

Theo yêu cầu của giai đoạn phân tích, chúng ta chỉ cần định nghĩa hệ thống như là một *hộp đen* (*Black Box*), trong đó hành vi của hệ thống thể hiện được *những gì* (*What*?) nó cần thực hiện và không cần thể hiện những cái đó thực hiện như thế nào (*How?*). Vì vậy, nhiệm vụ chính của chúng ta trong giai đoạn này là xác định và mô tả được các hoạt động của hệ thống theo yêu cầu của các tác nhân. Nghĩa là phải tìm được các sự kiện, các thao tác (sự tương tác) của các đối tượng trong từng ca sử dụng. Biểu đồ trình tự giúp chúng ta thể hiện được những kết quả đó.

### 5.2.1 Các thành phần của biểu đồ trình tự

Biểu đồ trình tự mô tả sự trao đổi thông điệp giữa các đối tượng trình tự theo thời gian, thông điệp được gửi và nhận bởi các đối tượng đang hoạt động trong hệ thống. Biểu đồ trình tự được thể hiện theo hai trục:

(i) *Trục dọc trên xuống chỉ thời gian* xảy ra các sự kiện, hay sự truyền thông điệp, được biểu diễn bằng các đường gạch - gạch thẳng đứng bắt đầu từ đỉnh đến đáy của biểu đồ.

(ii) *Trục ngang từ trái qua phải* là dãy các đối tượng tham gia vào tham gia vào việc trao đổi các thông điệp với nhau theo chiều ngang, có thể có cả các tác nhân.

Đối tượng được biểu diễn bằng hình chữ nhật trong đó có tên đối tượng cụ thể và/hoặc tên lớp cùng được gạch dưới (hoặc tên lớp được gạch dưới biểu diễn cho một đối tượng bất kỳ của lớp đó).

*Biểu đồ trình tự được đọc từ trên xuống dưới, từ trái sang phải*. Thứ tự các đối tượng trong biểu đồ phải được sắp xếp sao cho đơn giản nhất có thể để dễ quan sát. Thời gian thực hiện một thông điệp của một đối tượng được biểu diễn bằng *hình chữ nhật hẹp* dọc theo trục thẳng đứng của đối tượng đó.

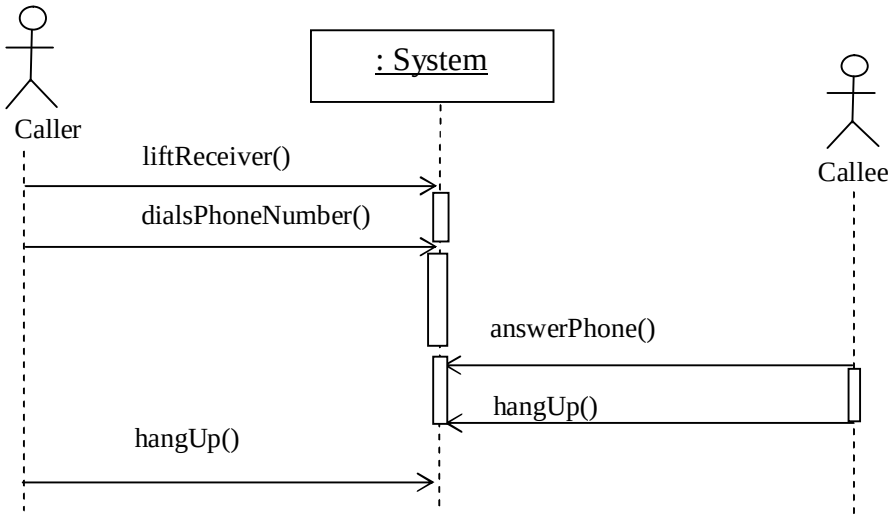
Mỗi thông điệp đều có tên gọi thể hiện được ý nghĩa của thông tin cần gửi và các tham số về dữ liệu liên quan. Thông thường đó là các *lời gọi hàm*. Ví dụ:

*print(aFile)*

được ghi kèm với mũi tên ký hiệu tương ứng cho kiểu thông điệp (loại đơn, được đồng bộ hoá hay dị bộ).

### 5.2.2 Xây dựng biểu đồ trình tự

Chúng ta xét tiếp ca sử dụng "*Gọi điện thoại",* trong đó có hai tác nhân là *Caller* (người gọi) và *Callee* (người nghe) . Hệ thống nhận được các sự kiện vào được ký hiệu là các lời gọi hàm: *liftReceiver()* (nhấc tai nghe), *dialsPhoneNumber()* (quay số), *answerPhone()* (trả lời điện thoại) và *hangUp()* (đặt tai nghe xuống). Biểu đồ trình tự mô tả cho các hoạt động trên được xác định như sau:




Hình 5-2 Biểu đồ trình tự mô tả hoạt động "*Gọi điện thoại"*

**Các bước xây dựng biểu đồ trình tự**

1. Xác định các tác nhân, các đối tượng tham gia vào ca sử dụng và vẽ chúng theo hàng ngang trên cùng theo đúng các ký hiệu,
2. Xác định những thông điệp (lời gọi hàm) mà tác nhân cần trao đổi với một đối tượng nào đó, hoặc giữa các đối tượng tương tác với nhau theo trình tự thời gian và vẽ lần lượt các hoạt động đó từ trên xuống theo thứ tự thực hiện trong thực tế.

### 5.2.3 Các biểu đồ trình tự mô hình hành động của hệ

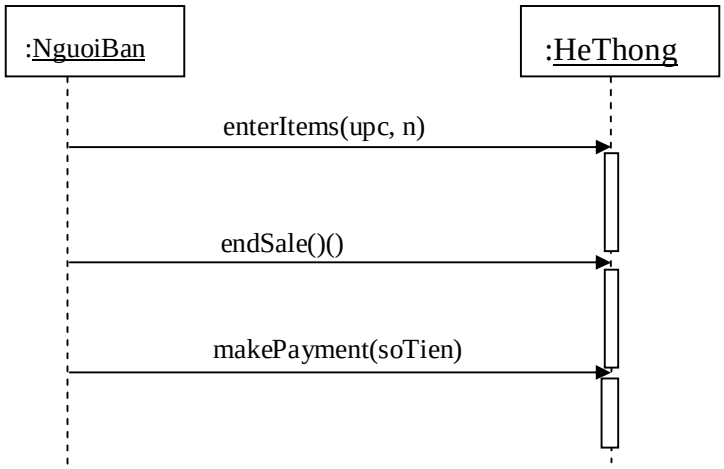
Chúng ta tiếp tục xây dựng một số biểu đồ trình tự mô tả các hành vi của hệ thống bán hàng **HBH**. Hành vi của hệ thống được thể hiện trong các biểu đồ trình tự thông qua sự tương tác của các đối tượng với nhau. Sự tương tác đó được thể hiện bằng các lời gọi hàm và sẽ được khai báo công khai (*public*) trong các lớp.

**Biểu đồ mô tả hoạt động của người bán hàng và hệ thống**

Như trong các kịch bản đã phân tích ở trên, sau khi chọn đủ hàng, người mua sẽ đưa giỏ (hoặc xe) hàng đến quầy để trả tiền. Người bán hàng phải nhập vào các thông tin và số lượng của mỗi mặt hàng. Những thông tin về mặt thường được xác định

thông qua mã sản phẩm *upc (Universe Product Code).* Vậy đối tượng :NguoiBan phải gửi đến cho :HeThong (một hệ thống bán hàng) thông điệp *enterItems()(upc, n),* để nhập vào mã sản phẩm *upc* và *n* đơn vị. Khi đã nhập xong tất cả các mặt hàng của khách muốn mua thì phải báo cho hệ thống biết là đã nhập xong bằng cách nhấn nút *Kết thúc*, nghĩa là gửi cho hệ thống thông điệp *endSale().* Hệ thống sẽ thông báo cho khách mua hàng biết tổng số tiền phải trả. Nếu khách hàng trả bằng tiền mặt thì :NguoiBan gửi tiếp đến cho :HeThong thông điệp *makePayment(soTien).*

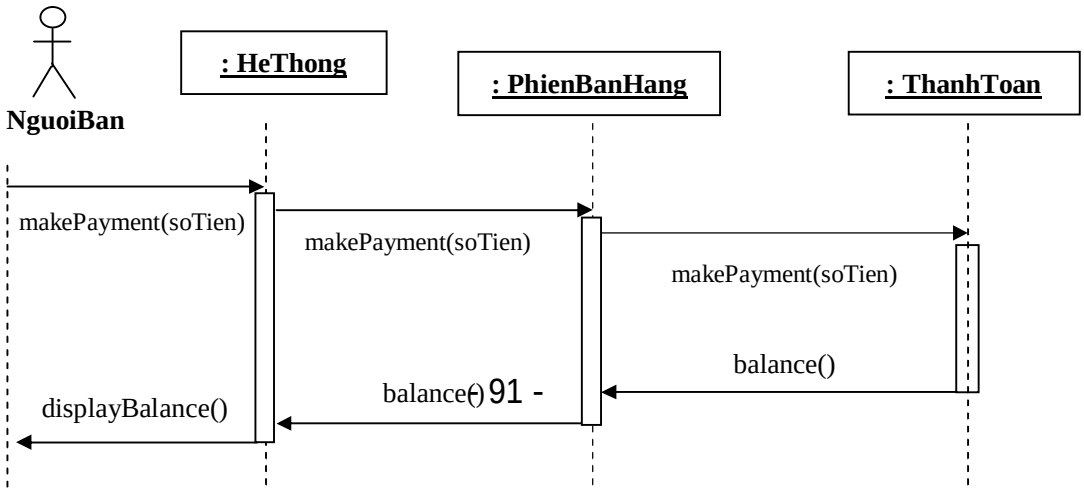
Các hoạt động trên được mô tả trong biểu đồ trình tự như hình 5-3.




Hình 5-3 Biểu đồ trình tự mô tả sự tương tác giữa *người bán* và *hệ thống*

**Biểu đồ trình tự mô tả ca sử dụng "Thu tiền mặt"**

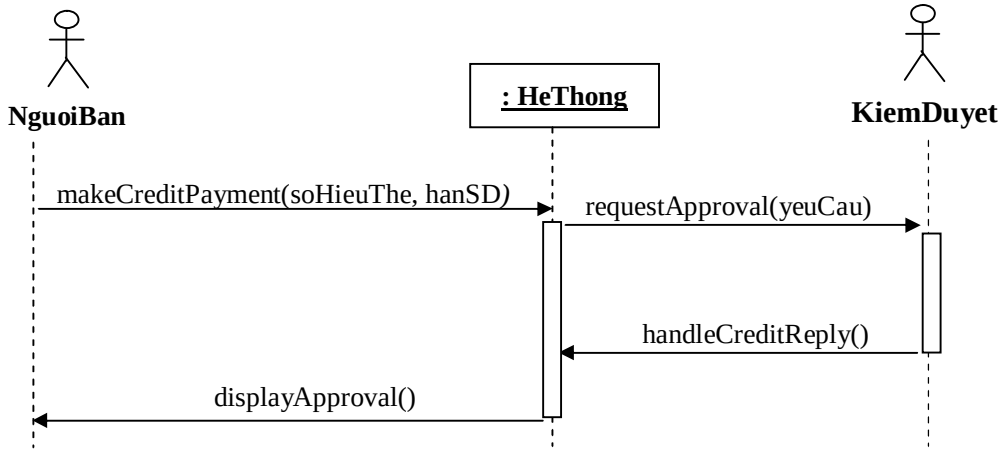
Khi đối tượng :HeThong nhận được yêu cầu thanh toán của khách hàng thông qua *người bán*, nghĩa là nó nhận được thông điệp *makePayment(soTien)* thì nó phải gửi cho một đối tượng *một phiên bán hàng*: PhienBanHang một yêu cầu tương tự. Đối tượng *một phiên bán hàng* lại yêu cầu đối tượng tạo ra một đối tượng *một lần thanh toán*: ThanhToan để thanh toán theo *soTien* mà khách đưa. Đối tượng *một lần thanh toán* gửi lại thông điệp *balance()* cho hệ thống *:HeThong biết số tiền dư (soTien* khách đưa trừ đi số tiền mua hàng) phải trả lại cho khách và để hiển thị (*displayBalance())* lên màn hình, để *người bán* thông báo lại cho khách hàng khi khách trả tiền mặt. Biểu đồ trình tự mô tả dãy các hành động trao đổi giữa các đối tượng nêu trên được thiết lập như sau:



Hình 5-4 Biểu đồ trình tự mô tả ca sử dụng "*Thanh toán (tiền mặt)"*

**Biểu đồ trình tự mô tả ca sử dụng "Thu bằng thẻ tín dụng"**

Khi thanh toán tiền hàng, khách hàng có thể trả bằng thẻ tín dụng. Thẻ tín dụng được xác định thông qua *soHieuThe* (*số hiệu thẻ*) và *hanSD* (*hạn sử dụng*). Điều đó được thể hiện bằng việc gửi một thông điệp *makeCreditPayment(soHieuThe, hanSD) cho hệ thống*. Thông thường hệ thống phải gửi một yêu cầu kiểm duyệt thẻ *requestApproval(yeuCau)*) tới bộ phận kiểm duyệt thẻ tín dụng là tác nhân ngoài của hệ thống. Hệ thống sẽ dựa vào kết quả trả lời của bộ phận kiểm duyệt để thanh toán với khách hàng bằng cách trừ đi số tiền mua hàng trong thẻ tín dụng nếu nó hợp pháp, nghĩa là thực hiện *handleCreditReply() (xử lý thẻ đã kiểm duyệt)*, dựa vào kết quả của *bộ phận kiểm duyệt*. Biểu đồ trình tự mô tả sự tương tác giữa *người bán hàng, hệ thống và bộ phận kiểm duyệt thẻ* **KiemDuyet** được thiết lập như sau:




Hình 5-5 Biểu đồ trình tự mô tả sự trao đổi giữa người bán, hệ thống và bộ phận kiểm duyệt

Tương tự như trên, chúng ta có thể tạo ra những biểu đồ trình tự cho các ca sử dụng khác của hệ thống.

### 5.2.4 Ghi nhận các hoạt động của các lớp đối tượng

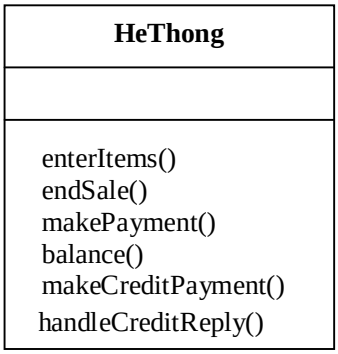
Ở trên chúng ta đã xây dựng các lớp và mới chỉ chú ý đến các thuộc tính của chúng. Khi đã xây dựng được các biểu đồ trạng thái và biểu đồ trình tự thì chúng ta đã hiểu hơn về sự tương tác giữa các đối tượng trong hệ thống. Chúng ta có thể dần dần

xác định được hành vi ứng xử của các đối tượng, nghĩa là xác định các hàm thành phần của các lớp.

Nghiên cứu các biểu đồ trình tự chúng ta nhận thấy:

> *Khi có một thông điệp mgs() gửi đến cho một đối tượng thì lớp của đối tượng đó phải có hàm thành phần là msg().*

Dựa vào nguyên lý nêu trên và căn cứ vào các biểu đồ trình tự ở hình 5-3, 5-4, 5-5, lớp **HeThong** sẽ có những hàm thành phần như sau:




Hình 5-6 Các thao tác của hệ thống được ghi nhận vào lớp có tên là **HeThong**

Khi xây dựng thêm biểu đồ trình tự cho ca sử dụng "*Thu bằng séc*" thì những hàm như: *makeCheckPayment(), handleCheckReply(),* v.v sẽ được bổ sung thêm vào lớp HBH.

*Lưu ý:*

1. *Để đơn gian,* trong giai đoạn này chúng ta có thể bỏ qua các đối số của các hàm thành phần của các lớp.
2. Các hàm thành phần cũng những đặc trưng quản lý quyền truy cập giống như các thuộc tính trong các lớp đối tượng. Do vậy, người thiết kế có thể khai báo chúng là công khai (*public*), được bảo vệ (*protected*), sở hữu riêng (*private*) hay mặc định như đã nêu ở phần định nghĩa các thuộc tính (chương IV).

Tương tự, chúng ta bổ sung các hàm thành phần vào các lớp còn khác của HBH.

### 5.2.5 Các hợp đồng về hoạt động của hệ thống

Như đã thấy ở trên, biểu đồ trình tự mới chỉ cho chúng ta biết tên của những nhiệm vụ mà mỗi đối tượng cần phải thực hiện khi nhận được một thông điệp, nhưng chưa mô tả cách thực hiện những công việc đó như thế nào. Để hiểu rõ hơn về những hành vi của các đối tượng và để hỗ trợ cho thiết kế và cài đặt các lớp sau này, chúng ta cần xây dựng các *hợp đồng (Contract)* hay *các đặc tả* cho những hoạt động (thao tác, hàm) đã xác định được. *Hợp đồng cho các hoạt động của hệ thống mô tả về sự thay đổi trạng thái mà khi bắt đầu thì*

*thoả tiền điều kiện (pre-conditions) và sau đó, khi kết thúc sẽ thoả mãn hậu điều kiện (post-conditions).*

- *Pre-conditions:* là những điều kiện mà trạng thái của hệ thống được giả thiết là thoả mãn trước khi thực hiện một thao tác, một hàm.
- *Post-conditions:* là những điều kiện mà trạng thái của hệ thống phải thoả mãn sau khi thực hiện xong một thao tác, một hàm.

Một cách hình thức, hợp đồng cho hoạt động *Op()* có thể viết theo bộ ba của *Hoare:*

*{Pre-conditions} Op() {Post-conditions}*

Thể hiện rằng: Nếu hoạt động *Op()* bắt đầu từ trạng thái thoả mãn *Pre-conditions* thì sau khi kết thúc thực hiện hệ thống sẽ ở trạng thái thoả mãn *Post-conditions.*

*Pre-condition và Post-condition* là những mệnh đề *boolean (mệnh đề điều kiện logic).*

*Ví dụ:* hãy xét thao tác *enterItems()(upc: UPC, soHang: Int)* của lớp **HBH**.

- *Pre-conditions*: Hệ mã chuẩn các sản phẩm toàn cầu **UPC** được biết trước đối với hệ thống.
- *Post-conditions*: bao gồm những điều kiện sau:
  + Nếu mặt hàng nhập vào là đầu tiên thì phải tạo ra một đối tượng mới là *phienBanHang* của lớp **PhienBanHang** và kết hợp nó vào hệ thống **HBH**,
  + Tạo ra một đối tượng *dongBanHang* của lớp **DongBanHang** cho mặt hàng vừa nhập vào và gán *dongBanHang.soLuong = soHang* (đối số của hàm),
  + *dongBanHang* được liên kết với *MoTaMatHang* dựa vào mã *upc*.

Ngoài *Pre-conditions* và *Post-coditions*, chúng có thể bổ sung thêm *mô tả tóm tắt về trách nhiệm, kiểu, lớp nào, tham chiếu tới những chức năng nào, chú thích, những ngoại lệ và kết quả, v.v.*

Chúng ta có thể xây dựng các hợp đồng như sau:

1. Từ các biểu đồ trình tự, xác định các thao tác, hàm, hoạt động của các lớp trong hệ thống,
2. Với mỗi thao tác trên hãy xây dựng một hợp đồng,
3. Mô tả tóm tắt những trách nhiệm chính mà hệ thống phải thực hiện khi thực thi thao tác này.
4. Trong *Post-condition* phải nêu được các trạng thái của các đối tượng sau khi kết thúc thao tác.
5. Để mô tả *Post-conditions*, hãy căn cứ vào:

   + Việc tạo lập, huỷ bỏ các đối tượng
   + Những thay đổi của các thuộc tính
   + Thiết lập hay huỷ bỏ các mối liên kết.

**Đặc tả các thao tác (hàm thành phần) của lớp HBH**

*1. Hợp đồng để nhập các thông tin về các mặt hàng*

*Hợp đồng*

| | |
|---|---|
| *Tên gọi:* | enterItems()(upc: UPC, soLuong: Int) |
| *Trách nhiệm:* | Nhập lần lượt *các thông tin* về những mặt hàng mà khách đã chọn mua và đưa chúng vào *phiên bán hàng*. Hiển thị các thông tin mô tả và *giá bán* của từng mặt hàng. |
| *Kiểu / Lớp:* | System (Hệ thống) |
| *Tham chiếu tới:* | R1.1, R1.3, R1.9 và ca sử dụng *Bán hàng* |
| *Chú ý:* | Sử dụng phương pháp truy nhập nhanh vào CSDL |
| *Ngoại lệ:* | Nếu *upc* không hợp lệ thì có lỗi |
| *Kết quả (Output):* | |
| *Pre-conditions:* | UPC được biết trước |
| *Post-conditions:* | + Nếu mặt hàng nhập vào là đầu tiên thì phải tạo ra một đối tượng *phienBanHang* của lớp **PhienBanHang** và kết hợp nó vào hệ thống HBH,<br>+ Tạo ra một đối tượng *dongBanHang* của lớp **DongBanHang** cho mặt hàng vừa nhập vào và gán *dongBanHang.soLuong = soHang* (đối số của hàm),<br>+ *dongBanHang* được liên kết với *MoTaMatHang* dựa vào mã *upc*. |

*2. Hợp đồng về việc kết thúc nhập hàng*

*Hợp đồng*

| | |
|---|---|
| *Tên gọi:* | endSale()() |
| *Trách nhiệm:* | Ghi nhận những mặt hàng đã được nhập. Hiển thị tổng số tiền bán hàng. |
| *Kiểu / Lớp:* | System (Hệ thống) |
| *Tham chiếu tới:* | R1.2, và ca sử dụng *Bán hàng* |
| *Chú ý:* | |
| *Ngoại lệ:* | Nếu *phiên bán hàng* không thực hiện được thì có lỗi |
| *Kết quả (Output):* | |
| *Pre-conditions:* | UPC được biết trước |
| *Post-conditions:* | *PhienBanHang.ketThuc* nhận giá trị *true.* |

*Lưu ý: Lớp* **PhienBanHang** từ hình 4-14 sẽ được bổ sung thêm thuộc tính *ketThuc* khi phân tích ca sử dụng "Thu tiền". Thuộc tính này có kiểu logic và sẽ nhận giá trị *true* khi kết thúc việc nhập dữ liệu và bắt đầu thực hiện thu tiền của khách hàng.

3. *Hợp đồng thực hiện thu tiền mặt*

*Hợp đồng*

| | |
|---|---|
| *Tên gọi:* | makePayment(soTien: DVT), trong đó DVT là lớp các loại tiền |
| *Trách nhiệm:* | Ghi nhận các thông tin liên quan đến việc *thanh toán*, tính *số tiền dư* cần trả lại cho khách. |
| *Kiểu / Lớp:* | System (Hệ thống) |
| *Tham chiếu tới:* | R2.1, và ca sử dụng *Bán hàng* |
| *Chú ý:* | |
| *Ngoại lệ:* | Nếu phiên bán hàng không kết thúc thì có lỗi<br>Nếu *soTien nhỏ hơn tongSoTien* thì cũng có lỗi |
| *Kết quả (Output):* | |
| *Pre-conditions:* | |
| *Post-conditions:* | + Một đối tượng *thanhToan* được tạo lập,<br>+ số tiền dư là *soTien - ThanhToan.tongSoTien*<br>+ *thanhToan* được liên kết với *phienBanHang* đã được tạo lập ở hợp đồng 1. để thực hiện việc cập nhật những mặt hàng đã bán, tổng số tiền, v.v. |

Tương tự, hãy xây dựng các hợp đồng cho những thao tác còn lại của các lớp.

*Lưu ý:* Đối với những ca sử dụng có nhiều đối tượng tham gia thì biểu đồ trình tự là khá phức tạp, do vậy nó không thích hợp. Muốn hiểu rõ hoạt động của các đối tượng thì tốt nhất là nên phân tách những ca sử dụng phức hợp thành các ca sử dụng tương đối đơn giản, dễ hiểu và mô tả được bằng biểu đồ trình tự một cách đơn giản.

## 5.3 Biểu đồ trạng thái

Bước nghiên cứu tiếp theo sau biểu đồ trình tự là biểu đồ trạng thái (*State Diagram, State Machine Diagram, State Chart Diagram* ).

Biểu đồ trạng thái thể hiện chu kỳ hoạt động của đối tượng, các hệ thống con và của cả hệ thống. Biểu đồ trạng thái mô tả:

- *Các trạng thái* mà các đối tượng có thể có,

- *Các sự kiện:* các thông điệp nhận được, các lỗi có thể xuất hiện, điều kiện nào đó có thể trở thành đúng (*true*), khoảng thời gian đã qua, v.v. tác động lên trang thái để làm biến đổi.

Biểu đồ này là giải pháp tốt để mô hình hoá hành vi động của các lớp đối tượng. Trong một dự án, không nhất thiết phải tạo ra các biểu đồ trạng thái cho tất cả các lớp. Tuy nhiên, đối với những lớp có nhiều hành vi động, có nhiều trạng thái hoạt động khác nhau thì biểu đồ trạng thái là hữu ích, giúp chúng ta hiểu rõ hệ thống hơn.

### 5.3.1 Trạng thái và sự biến đổi trạng thái

Mọi đối tượng trong hệ thống đều có chu kỳ sống và mỗi thời điểm đều có một trạng thái nào đó. Ví dụ, *người bán hàng* trong hệ thống HBH *đang bán hàng, phiên bán hàng* đã được *thanh toán, v.v.*

*Trạng thái* là một trong các điều kiện có thể để đối tượng tồn tại, là kết quả của một hoạt động trước đó của đối tượng.

Trạng thái của đối tượng thường được mô tả trong hình chữ nhật góc tròn và được xác định bởi:

- *Tên gọi trạng thái,* thường bắt đầu bằng động từ,
- *Biến trạng thái (State Variable)* mô tả các giá trị hiện thời của trạng thái,
- *Hoạt động (Activity)* là hành vi mà đối tượng sẽ thực hiện khi nó ở vào trạng thái đó.

*Hoạt động* của trạng thái được mô tả hình thức như sau:

*event_name argument_list '/' action_exp*

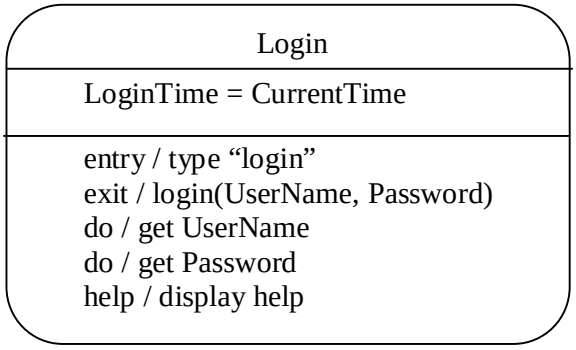
Trong đó,

*event_name:* Tên của sự kiện, có thể là một trong các sự kiện chuẩn: *exit (thoát ra), entry (vào), do (thực hiện).*

*argument_list:* danh sách các sự kiện,

*action_exp:* những hoạt động cần thực hiện bao gồm các lời gọi hàm, thao tác trên các biến trạng thái, v.v.

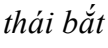
*Ví dụ*: trạng thái *Login được mô tả trong UML:*



Hình 5-7 Trạng thái Login

Khi hệ thống ở trạng thái *Login* thì biến *LoginTime (thời gian khi khởi nhập)* được gán là *CurrentTime (thời gian hiện thời)* của máy tính. Sự kiện vào của trạng thái này là gõ từ "*login*" và để thoát ra khỏi trạng thái này thì phải thực hiện lời gọi hàm *login(UserName, Password).* Các hoạt động của đối tượng ở trạng thái này là: Nhận vào *UserName (tên người sử dụng), Password (mật khẩu),* và hiển thị sự trợ giúp *display help*.

*Lưu ý:*

- Khi không cần mô tả chi tiết thì có thể chỉ cần tên gọi để xác định trạng thái trong các biểu đồ.
- Có hai trạng thái đặc biệt là *trạng thái bắt đầu* được ký hiệu là:  và *trạng thái kết thúc*, được ký hiệu là 

Biểu đồ trạng thái thường có trạng thái bắt đầu còn trạng thái kết thúc thì có thể có hoặc không tuỳ vào chu kỳ hoạt động của các đối tượng.

Trong biểu đồ *đường mũi tên chỉ ra sự biến đổi từ một trạng thái sang trạng thái khác* khi có các sự kiện xảy ra làm thay đổi các trạng thái. Trạng thái của đối tượng sẽ *bị thay đổi* khi có cái gì đó xảy ra, nghĩa là khi có một hay nhiều sự kiện xuất hiện. *Sự biến đổi trạng thái hay sự chuyển trạng* (*transition*) thể hiện mối quan hệ giữa các trạng thái với nhau.

*Sự chuyển trạng* được thể hiện trong biểu đồ bằng *mũi tên có nhãn là* sự kiện, thao tác (hàm có đối số), hoặc *điều kiện cầm canh* (*guard*). Sự chuyển trạng có thể là *đệ qui*, nghĩa là trong một điều kiện nhất định, *một đối tượng có thể quay lại trạng thái cũ của nó.*

### 5.3.2 Xác định các trạng thái và các sự kiện

Để xác định được các trạng thái và các sự kiện chúng ta cần trả lời cho các câu hỏi sau:

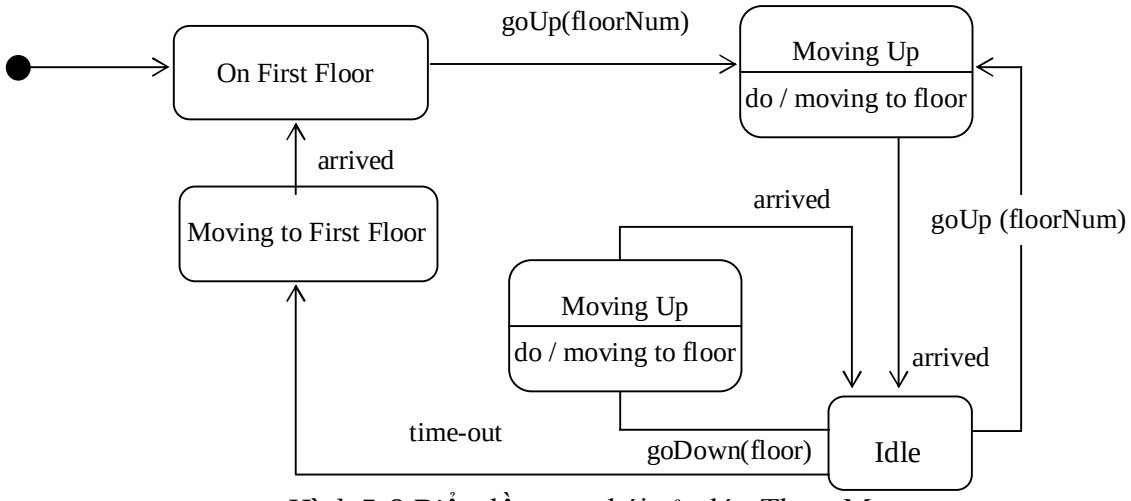
- Một đối tượng có thể ở những trạng thái nào? Liệt kê tất cả các trạng thái có thể có trong hệ thống của mỗi đối tượng.
- Những sự kiện nào có thể xuất hiện? Bởi vì sự kiện có thể làm biến đổi trạng thái, do vậy, từ các sự kiện có thể xác định được các trạng thái của đối tượng.
- Những trạng thái mới nào sẽ xuất hiện? Từ một trạng thái, đối tượng có thể chuyển sang trạng thái mới khi một số sự kiện xác định xuất hiện.
- Ở mỗi trạng thái, hoạt động của đối tượng là gì?
- Sự tương tác giữa các đối tượng là gì? Sự tương tác giữa các đối tượng thường gắn chặt với các trạng thái của đối tượng.
- Những sự kiện, hay chuyển đổi trạng thái nào là không thể xảy ra? Một số sự kiện, hay trạng thái không thể chuyển đổi sang trạng thái khác được, ví dụ khi khách mua hàng *trả bằng thẻ tín dụng không hợp pháp* thì *phiên bán đó không thực hiện được*.
- Cái gì làm cho đối tượng được tạo ra? Đối tượng thường được tạo ra bởi một, hay một số sự kiện.

- Cái gì làm cho đối tượng bị huỷ bỏ? Đối tượng thường được loại bỏ khi không còn cần thiết nó nữa.

### 5.3.3 Xây dựng biểu đồ trạng thái

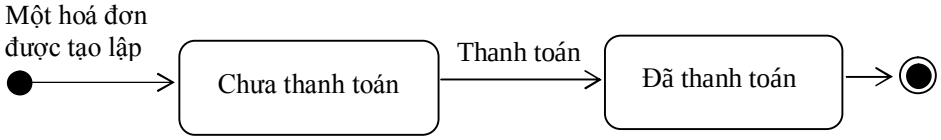
Biểu đồ trạng thái được sử dụng để chỉ ra cách các đối tượng phản ứng lại đối với các sự kiện và cách *biến đổi các trạng thái theo các sự kiện đó.*

*Ví dụ:* Hãy mô tả hoạt động của hệ thống thang máy. Thường thang máy bắt đầu hoạt động từ tầng một (*OnFirstFloor*). Khi đang ở *OnFirstFloor* và có người ở tầng trên (*floorNum*) nhấn nút yêu cầu thang máy (*goUp(floorNum)*) thì nó chuyển sang trạng thái *chuyển lên (MovingUp).* Khi chuyển đến tầng yêu cầu (*arrived*) thì nó chuyển sang trạng thái dừng, nghỉ (*Idle*) để mở cửa cho người vào /ra khỏi thang máy. Đang ở trạng thái nghỉ *Idle*, nếu có ai ở tầng trên yêu cầu thì nó lại chuyển về *MovingUp,* nếu có người ở tầng dưới yêu cầu thì thang máy *chuyển xuống* (*MovingDown*), còn khi hết giờ (*time-out*) nó sang trạng thái *chuyển về tầng một* (*MovingtoFirstFloor*) rồi về tầng một. Biểu đồ trạng thái mô tả hoạt động của thang máy được vẽ như hình 5-8.




Hình 5-8 Biểu đồ trạng thái của lớp ThangMay

Chúng ta hãy xây dựng biểu đồ trạng thái cho lớp **HoaDon.**

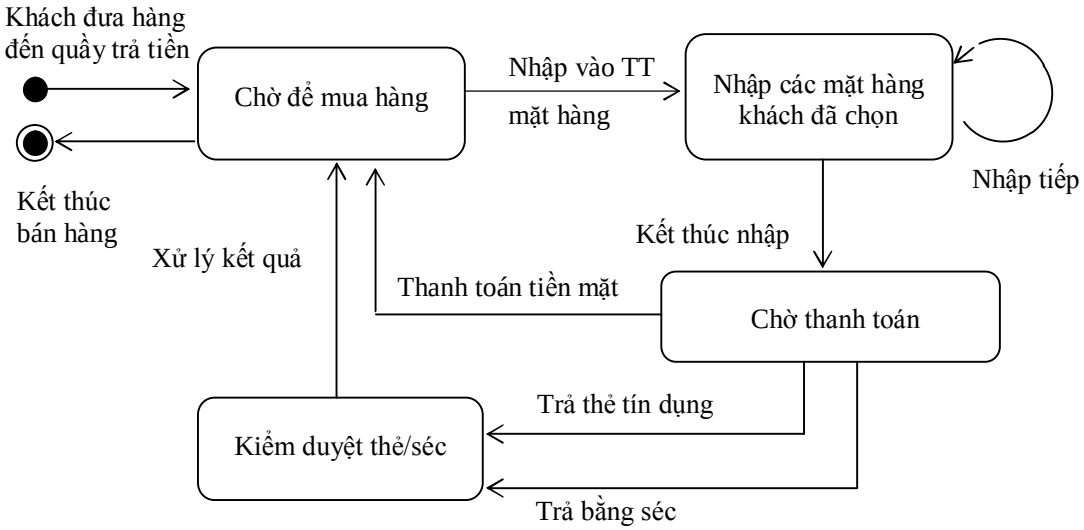



Hình 5-9 Biểu đồ các trạng thái của lớp **HoaDon**

Khi *một hoá đơn (đối tượng của lớp **HoaDon**)* được tạo lập thì nó ở trạng thái *chưa thanh toán,* sau đó khi có sự kiện *khách hàng thanh toán,* nghĩa là khách trả tiền cho các mặt hành đã chọn mua thì nó chuyển sang trạng thái *đã thanh toán.*

Như đã đề cập ở trên, các ca sử dụng là rất quan trọng, nó thể hiện những nhiệm vụ mà hệ thống phải thực hiện. Vì vậy, thường chúng phải xây dựng các biểu đồ trạng thái để mô tả cho các lớp trong những ca sử dụng quan trọng nhất của hệ thống.

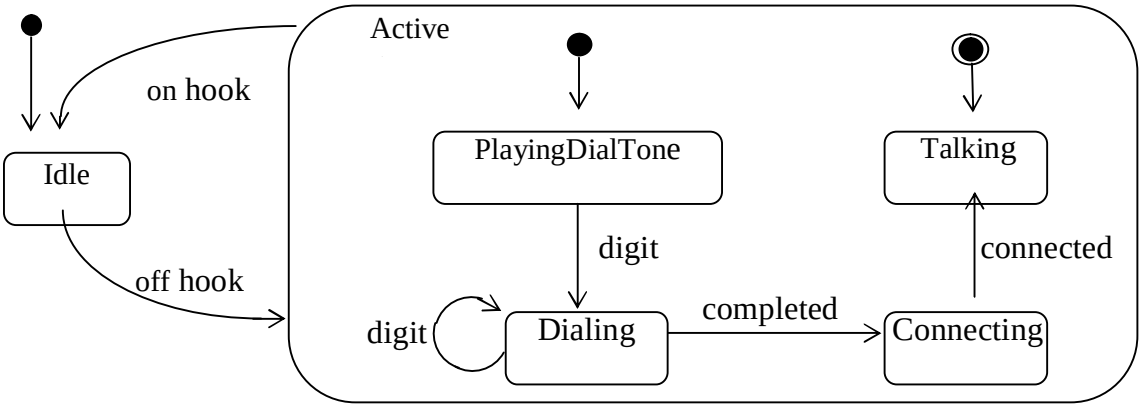
Biểu đồ trạng thái của hệ *HBH* được xây dựng như sau:




Hình 5-10 Biểu đồ trạng thái của lớp HBH

Trạng thái của một đối tượng cũng có khi là trạng thái phức hợp, nghĩa là nó có thể chứa các trạng thái con được lồng bên trong. Một số trạng thái, ví dụ trạng thái *Kiểm duyệt thẻ* trong biểu đồ trên có thể tiếp tục làm mịn hơn ở pha sau.

Chúng ta xét tiếp **Telephone** trong ca sử dụng "*Gọi điện thoại*" đã được mô tả bằng biểu đồ vết các sự kiện ở trên. **Telephone** có hai trạng thái chính: *Idle* (*rỗi)* và *Active* (*hoạt động*). Trạng thái *Active* lại có thể phân tách tiếp thành *PlayingDialTone* (*âm hiệu ĐT mời gọi*), *Dialing* (*quay số*), *Connecting* (*kết nối hai đầu dây*) và *Talking* (*đàm thoại*). Biểu đồ trạng thái cho các hoạt động trên được mô tả như sau.



Hình 5-11 Biểu đồ trạng thái của Telephone

Máy điện thoại ở trạng thái *Idle*, khi người gọi nhấc tai nghe lên (*off hook*) thì nó chuyển sang trạng thái *hoạt động (Active)* sẵn sàng phục vụ đàm thoại giữa hai điểm trong mạng điện thoại. *Trạng thái Active* lại được mô tả dưới dạng một biểu đồ trạng thái con. Bắt đầu là trạng thái *Có âm hiệu điện thoại mời gọi* (*PlayingDialTone*), khi người gọi quay số (*digit*) nó chuyển sang trạng thái *Quay số (Dialing)*. Khi quay xong (*completed*), nó chuyển tiếp sang trạng thái *Kết nối (Connecting)* và khi đường dây được *kết nối (connected)* thì hai người có thể nói chuyện được với nhau (*Talking*). Trạng thái *Talking* lại có thể mô tả chi tiết hơn bằng một biểu đồ trạng thái con nếu cần thiết.

*Lưu ý:*

- Biểu đồ trạng thái chỉ cần xây dựng cho những đối tượng có nhiều hoạt động quan trọng trong hệ thống,
- Dựa vào các ca sử dụng để xây dựng biểu đồ trạng thái,
- Dựa vào các thuộc tính liên quan để định nghĩa các trạng thái.

*Tóm lại,* biểu đồ trạng thái là cần thiết vì nó giúp người phân tích, thiết kế và người lập trình hiểu, nắm bắt được các hành vi ứng xử của các đối tượng tham gia vào các ca sử dụng. Họ không chỉ cài đặt đối tượng mà còn cần phải làm cho chúng thực hiện những công việc mà hệ thống yêu cầu. Tuy nhiên biểu đồ trạng thái không được sử dụng để sinh mã tự động trong khâu lập trình sau này.

Biểu thức trạng thái trong phân tích hướng đối tượng cũng tương tự như sơ đồ khối trong phân tích có cấu trúc, nó mô tả các bước cần thực hiện (thuật toán) của hệ thống.

## 5.4 Biểu đồ hoạt động

Biểu đồ hoạt động (*Activity Diagram*) trong UML gần giống với lưu đồ (*Flow Chart*) mà chúng ta đã quen sử dụng trong phân tích thiết kế có cấu trúc. Nó chỉ ra các bước thực hiện, các hành động, các nút quyết định và điều kiện rẽ nhánh để điều khiển luồng thực hiện của hệ thống. Biểu đồ hành động mô tả các hành động và các kết quả của những hành động đó và:

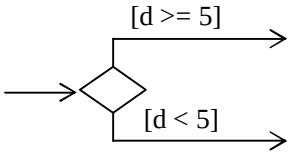
- Nhấn mạnh hơn về công việc thực hiện khi cài đặt một thao tác của từng đối tượng,
- Tương tự như biểu đồ trạng thái, nhưng khác chủ yếu ở chỗ nó tập trung mô tả về các hoạt động (công việc và những thao tác cần thực thi) cùng những kết quả thu được từ việc thay đổi trạng thái của các đối tượng.
- Trạng thái trong biểu đồ hành động là các trạng thái hoạt động, nó sẽ được chuyển sang trạng thái sau, nếu hành động ở trạng thái trước được hoàn thành.

**Trạng thái và sự chuyển trạng**

Trạng thái và sự chuyển đổi trạng thái được ký hiệu và cách sử dụng hoàn toàn giống như trong biểu đồ trạng thái đã nêu ở trên.

**Nút quyết định và rẽ nhánh**

Một đối tượng khi hoạt động thì từ một trạng thái có thể rẽ nhánh sang những trạng thái khác nhau tuỳ thuộc vào những điều kiện, những sự kiện xảy ra để quyết định. Điều kiện rẽ nhánh thường là các biểu thức *Boolean.* Trong UML, nút quyết định rẽ nhánh được biểu diễn bằng hình thoi có các đường rẽ nhánh với những điều kiện đi kèm để lựa chọn như hình 5-12.

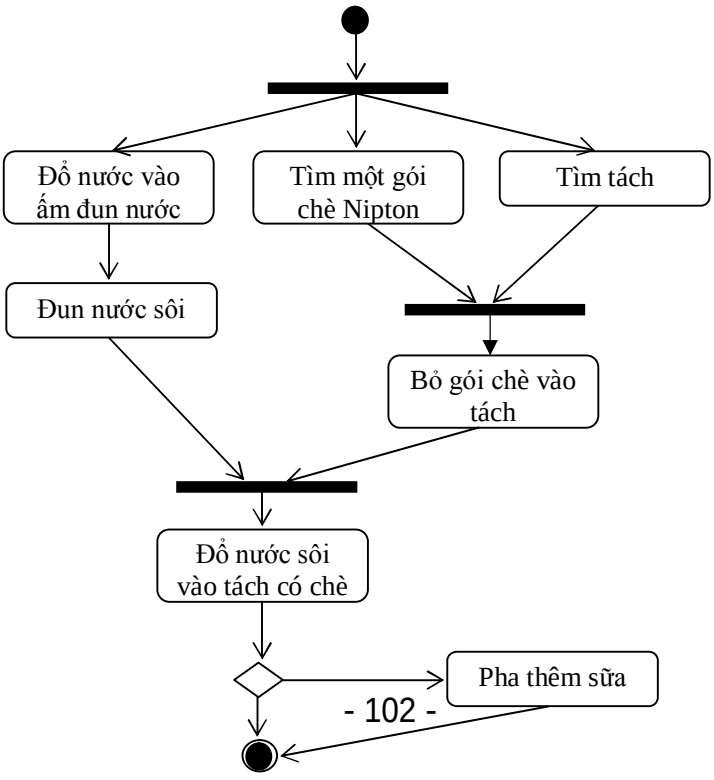



Hình 5-12 Nút rẽ nhánh trong biểu đồ hành động

**Thanh tương tranh hay thanh đồng bộ** (Conccurency Bar)

Trong hoạt động của hệ thống, có thể có nhiều luồng hành động được bắt đầu thực hiện hay kết thúc đồng thời. Trong UML, *thanh đồng bộ* được vẽ bằng đoạn thẳng đậm được sử dụng để kết hợp nhiều luồng hành động đồng thời và để chia nhánh cho những luồng có khả năng thực hiện song song.

*Ví dụ:* hãy vẽ biểu đồ hành động mô tả các hoạt động "*Đun nước và pha một tách chè Nipton"*. Chúng ta thấy một số hoạt động có thể thực hiện song hành như "*Đun nước*", "*Tìm một gói chè Nipton*", "*Tìm tách"*, v.v. Biểu đồ hành động cho các hoạt động trên có thể mô tả như hình 5-13.

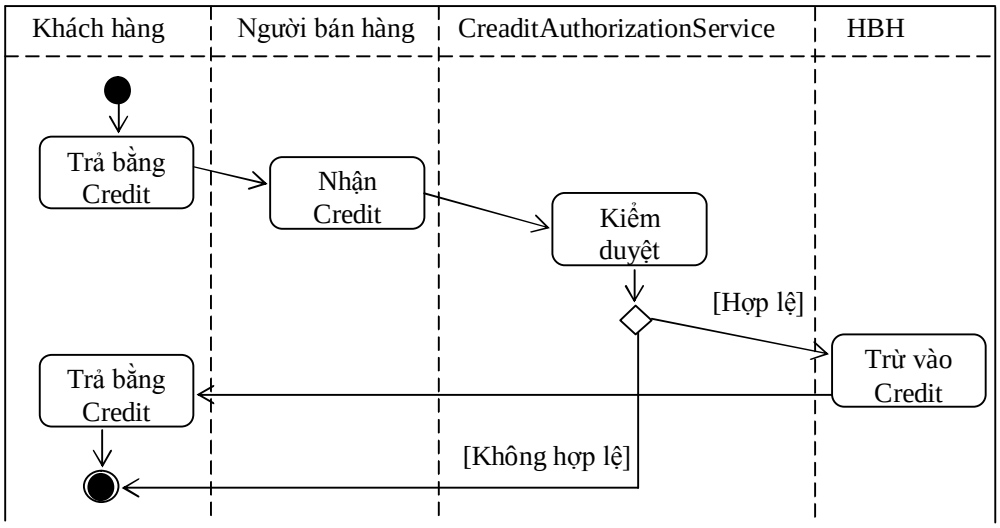


Hình 5-13 Biểu đồ hành động "Đun nước và pha chè"

**Tuyến công việc** (*swimlane*)

*Tuyến công việc* được sử dụng để phân hoạch các hoạt động (trạng thái) theo các nhóm đối tượng hay theo tuyến hoạt động của từng đối tượng. Giống như trong một cuộc *thi bơi*, trong bể bơi mỗi vận động viên bơi lội chỉ được bơi theo tuyến bơi đã được qui định. Trong hệ thống phần mềm cũng vậy, mỗi đối tượng hoạt động theo tuyến đã được xác định, nhưng có khác là giữa các tuyến này có sự chuyển đổi thông tin với nhau.

*Ví dụ:* hãy xét các hoạt động xảy ra khi khách mua hàng chọn phương thức thanh toán bằng Credit. Người bán hàng nhận thẻ từ khách hàng, chuyển thẻ cho bộ phận kiểm duyệt. Nếu là thẻ hợp lệ thì trừ vào thẻ số tiền mua hàng của khách phải trả và giao lại thẻ cho khách. Các hoạt động trên được mô tả trong biểu đồ hành động như ở hình 5-14.




Hình 5-14 Các tuyến công việc trong biểu đồ hành động

Biểu đồ hành động sử dụng để thể hiện những hành động sẽ thực hiện của mỗi đối tượng và chỉ ra cách thực hiện các công việc nghiệp vụ thông qua các dòng công việc (w*orkflow*), theo tổ chức của các đối tượng.

## 5.5 Sử dụng Rational Rose để tạo lập biểu đồ trình tự

- Tạo lập ba biểu đồ trình tự như hình 5-3, 5.4, 5.5

- Thực hiện một số khai báo đặc tả chi tiết:

+ Gắn tệp vào biểu đồ trình tự

+ Bổ sung thông điệp vào biểu đồ trình tự

+ Sắp xếp lại các thông điệp

+ Đánh số lại các thông điệp

+ Ánh xạ đối tượng vào lớp

+ Gán trách nhiệm cho các đối tượng.

## 5.6 Sử dụng Rational Rose để tạo lập biểu đồ trạng thái

Rational Rose hỗ trợ để tạo lập nhanh các biểu đồ trạng thái. Tương tự như đối với các biểu đồ khác, trong Rose 2000 biểu đồ trạng thái có thể được tạo lập mới bằng hai cách:

1. Nhấn chuột trái ở mục *Browser* trong thanh thực đơn chính và chọn *State Machine Diagram*
2. Nhấn chuột trái ở biểu tượng của *Logical View* hoặc *Use Case View ở* danh sách trình duyệt, rồi nhấn chuột phải để chọn *New > StateChart Diagram.*

Tất cả các biểu đồ đã được tạo lập có thể mở (*Open*), in (*print*), xoá (*delete*), đổi tên (*Rename*), hay bổ sung thêm các thành phần của biểu đồ (*New>*) bằng cách chọn biểu đồ tương ứng trong *Logical View*, hoặc *Use Case View* (nhấn chuột trái), rồi nhấn chuột phải để chọn một trong những chức năng trên.

Để mở một biểu đồ đã được tạo lập trước thì đơn giản nhất là nhấn đúp chuột trái vào tên của biểu đồ đó trong danh sách trình duyệt (*Browser*) ở bên trái màn hình.

Hãy thực hiện:

- Tạo lập biểu đồ trạng thái như hình 5-8, 5-10, 5-11.
- Xây dựng biểu đồ trạng thái cho lớp **DigitalWatch** (đồng hồ điện tử). Lớp này có hai hàm thành phần: *modeButton()* làm nhiệm vụ thay đổi *mode* hiệu chỉnh thời gian giữa giờ, phút và *inc()* để tăng lên một đơn vị thời gian ứng với *mode tương ứng*. Tất nhiên khi nhấn *inc()* mà số đơn vị thời gian, ví dụ là giờ mà đã là 24 thì sau đó sẽ trở về số 1, còn đối với phút thì sau 60 sẽ là 1. Nó có ba trạng thái và cách chuyển trạng được mô tả như sau:

+ Trạng thái *Display*: trong đó hiển thị thời gian hiện thời: *do /display currentTime.*

+ Khi NSD nhấn vào *modeButton* thì chuyển sang trạng thái *Set Hours* (Đặt lại giờ), trong đó thực hiện: *do / display hours.*

+ Khi NSD nhấn tiếp vào *modeButton* thì chuyển sang trạng thái *Set Minute* (Đặt lại phút), trong đó thực hiện: *do / display minutes.*

+ Tất nhiên nếu lại nhấn *modeButton* lần thứ ba thì nó quay lại trạng thái ban đầu. Sau đó lại từ trạng thái *Display* có thể chuyển sang trạng thái tiếp theo như trên khi NSD nhấn *modeButton.*

Trong hai trạng thái *Set Hours, Set Minute* nếu nhấn *inc* thì thuộc tính *hours, minute* của đồng hồ sẽ được tăng lên một.

(Chi tiết về cách xây dựng biểu đồ tương tác tham khảo trong ([8], [11]))

*Lưu ý:* Đối với hầu hết các biểu đồ và các phần tử của chúng, hãy mô tả tóm tắt các chức năng, hay các tính chất đặc trưng cơ bản của chúng ở cửa sổ *Documentation*. Những thông tin này sẽ rất hữu ích cho việc theo dõi, hiểu biết về chúng trong quá trình phát triển phần mềm. Mục *Documentation* là một ô cửa sổ được mở ra dưới cửa sổ hiển thị danh sách các biểu đồ ở bên trái (*Browser* từ *View*). Nếu nó chưa được mở thì từ thực đơn *View* chọn *Documentation* để mở nó và viết các tài liệu đặc tả, chú thích cho những hoạt động, khái niệm trong các kết quả phân tích, thiết kế. *Đây chính là một trong các yêu cầu bắt buộc của công nghệ phần mềm.*

## Bài tập và câu hỏi

5.1 Hãy cho biết những mệnh đề sau đúng hay sai (*true /false*), giải thích tại sao?

+ Các sự kiện độc lập cũng có thể xảy ra đồng thời.

+ Một lớp có thể có trạng thái khởi đầu và kết thúc.

+ Không nhất thiết phải có trạng thái cho một đối tượng.

+ Cần phải xây dựng biểu đồ trạng thái cho tất cả các lớp trong hệ thống.

+ Hành vi của hệ thống được thể hiện trong các biểu đồ trình tự thông qua sự tương tác của các đối tượng với nhau.

+ Các sự kiện vào kích hoạt hệ thống hoạt động và hệ thống hoạt động là để trả lời cho các sự kiện vào mà các tác nhân tạo ra.

+ Biểu đồ trạng thái được sử dụng để sinh mã tự động cho chương trình.

5.2 Xây dựng biểu đồ trạng thái cho ca sử dụng "Đặt lại giờ, phút, giây cho đồng hồ điện tử".

5.3 Xây dựng biểu đồ trạng thái cho ca sử dụng "Đăng ký môn học".

5.4 Xây dựng biểu đồ trình tự mô tả ca sử dụng "Thu tiền bằng séc" của hệ thống bán hàng.

5.5 Thiết lập biểu đồ trình tự và biểu đồ trạng thái cho lớp chính trong ca sử dụng "*Cho mượn sách"* của hệ thống "Quản lý thư viện" (Tiếp theo của bài tập 4.3).

5.6 Xây dựng biểu đồ trình tự mô tả sự tương tác giữa các lớp đối tượng trong ca sử dụng "*Rút tiền tự động"* trong "Hệ thống rút tiền tự động ATM (Automatic Teller Machine)" (Tiếp theo của bài toán 4.4).

*5.7* Chọn từ danh sách dưới đây những thuật ngữ thích hợp để điền vào các chỗ [(…)] trong đoạn văn mô tả về biểu đồ tương tác.

Xây dựng [(1)] là thực hiện việc gán trách nhiệm cho [(2)]. Từ [(1)], người thiết kế có thể phát hiện thêm các [(3)], các thao tác cần thực hiện của mỗi [(3)], v.v. Do vậy, [(1)] trở thành nền tảng cho các bước còn lại của quá trình phát triển phần mềm.

*Chọn câu trả lời:*

a. lớp

b. các đối tượng

c. biểu đồ tương tác

# CHƯƠNG VI

# THIẾT KẾ CÁC BIỂU ĐỒ CỘNG TÁC VÀ BIỂU ĐỒ THÀNH PHẦN CỦA HỆ THỐNG

Chương VI đề cập đến:

- ✓ Thiết kế các biểu đồ cộng tác của các đối tượng để thực hiện các nhiệm vụ của hệ thống.
- ✓ Xây dựng các mẫu gán trách nhiệm cho các lớp đối tượng,
- ✓ Thiết kế chi tiết các lớp và biểu đồ lớp, tập trung vào quan hệ kế thừa và sử dụng lại.

Trong pha phân tích ở các chương trước, chúng ta đã tập trung khảo sát hệ thống để trả lời cho câu hỏi hệ thống gồm *những cài gì* và cũng đã bắt đầu nghiên cứu các *mối quan hệ, tương tác giữa* các lớp đối tượng thông qua biểu đồ trình tự và biểu đồ trạng thái. Nhiệm vụ chính của giai đoạn thiết kế là chuyển từ câu hỏi *"Cái gì?"* (*What*) sang trả lời cho câu hỏi "*như thế nào?"* (*How?*). Như vậy nhiệm vụ chính trong thiết kế hướng đối tượng là xây dựng các biểu đồ cộng tác của các đối tượng để hệ thống thực hiện được các yêu cầu đã được xác định trong pha phân tích.

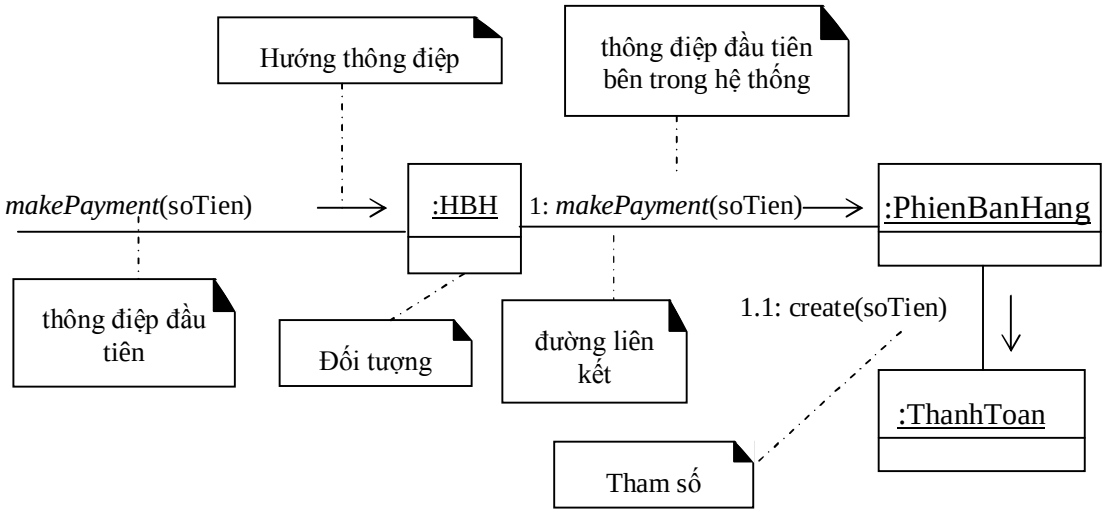
## 6.1 Các biểu đồ cộng tác

Trong chương V chúng ta đã khẳng định, các hoạt động của hệ thống có thể mô tả trực quan bởi các biểu đồ tương tác. UML định nghĩa hai loại biểu đồ tương tác: *biểu đồ trình tự* và *biểu đồ cộng tác*.

*Biểu đồ cộng tác* giống như biểu đồ trình tự (như đã trình bày ở trên), mô tả sự tương tác của các đối tượng với nhau, nhưng khác với biểu đồ trình tự là ở đây tập trung vào ngữ cảnh và không gian thực hiện công việc.

*Biểu đồ cộng tác* chính là một đồ thị chỉ ra một số các đối tượng và những sự liên kết giữa chúng, trong đó các đỉnh là các đối tượng còn cạnh thể hiện sự trao đổi thông điệp giữa các đối tượng.

*Ví dụ:* hãy xét vấn đề thanh toán trong hệ thống bán hàng. Giả sử khách hàng trả tiền mua hàng bằng tiền mặt. Người bán phải ghi lại số tiền mà khách đưa và qua đó hệ thống HBH nhận được thông điệp *makePayment(soTien). soTien* là số tiền khách đưa, thường là lớn hơn số tiền hàng và hệ thống phải tính số dư để trả lại cho khách. Để thực hiện được yêu cầu thanh toán thì HBH lại gửi một thông điệp tương tự cho *phiên bán hàng* và kết quả là tạo ra một đối tượng của lớp **ThanhToan** theo thông điệp *create(soTien)* để thực hiện thanh toán với khách hàng. Hình 5-4 mô tả biểu đồ trình tự trao đổi các thông điệp trên lần lượt theo thời gian. Sau đây chúng ta sử dụng biểu đồ cộng tác để thể hiện cùng mối tương tác đó nhưng theo ngữ cảnh thực hiện công việc.




Hình 6-1 Biểu đồ cộng tác để thực hiện *thanh toán tiền mặt*

Biểu đồ trên được đọc như sau:

1. Thông điệp đầu *makePayment(soTien)* (chính là lời gọi hàm) được gửi tới một đối tượng của HBH: HBH.
2. Đối tượng :HBH gửi tiếp thông điệp tới một đối tượng của **PhienBanHang** là :PhienBanHang. Hàm này được đánh số là 1.
3. :PhienBanHang gọi toán tử tạo lập của lớp **ThanhToan** để tạo ra một đối tượng :ThanhToan theo thông điệp *created(soTien)* để làm nhiệm vụ thu tiền. Thông điệp này được đánh số là 1.1.

Qua đó có thể khẳng định: *biểu đồ cộng tác của một hoạt động thể hiện thuật toán để thực thi hành động đó.*

Trước khi thiết kế các biểu đồ cộng tác cho các hoạt động của hệ thống, hãy lưu ý một số ký hiệu, cách biểu diễn các thông điệp và một số qui ước như sau:

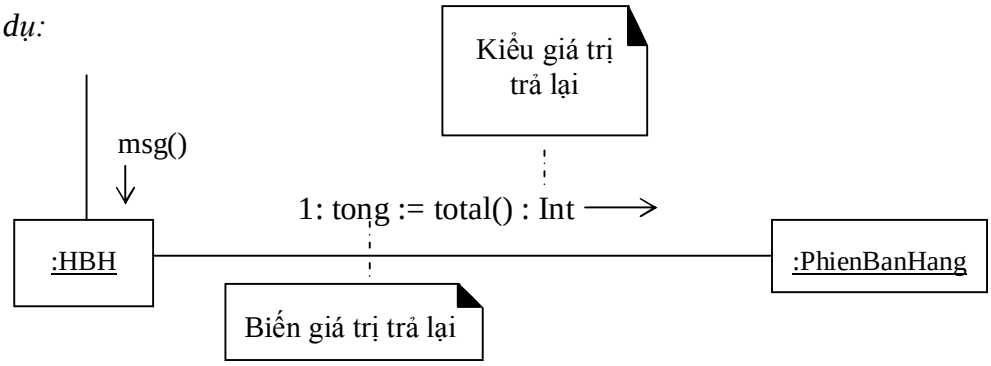
**(1) Thể hiện giá trị trả lại** (*return value*)

Một số thông điệp được gửi đến cho một đối tượng và yêu cầu có giá trị trả lại cho đối tượng gửi. Giá trị này có thể chỉ ra thông qua phép gán cho một tên biến và tên của thông điệp đó. Trong lập trình, thực chất đây là lời gọi hàm có kiểu giá trị trả lại (trong C, đó là những hàm có kiểu trả lại khác *void*).

Cú pháp chung của thông điệp này có dạng:

*ReturnVariableName := message(parameter:ParameterType): ReturnType*

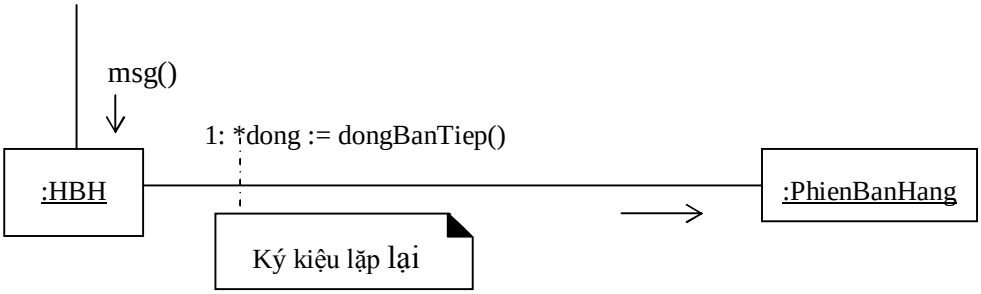
*Ví dụ:*



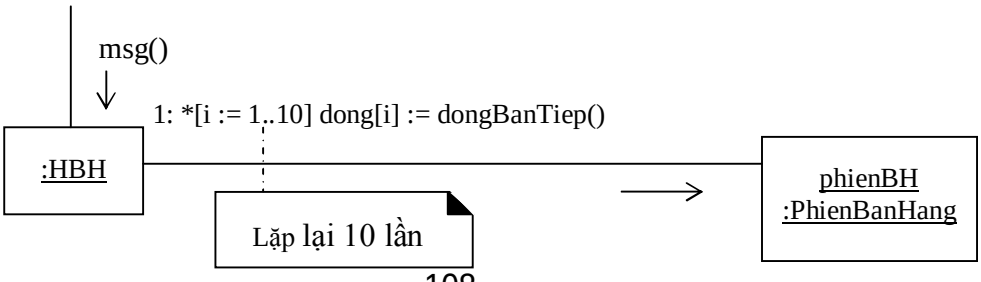

Hình 6-2 Thông điệp có giá trị trả lại

**(2) Thể hiện những thông điệp lặp**

Một đối tượng có thể gửi một thông điệp lặp lại một số lần cho đối tượng khác. Thông điệp được gửi lặp lại nhiều lần có thể biểu diễn bằng dấu '*' trước thông điệp. *Ví dụ*:




Hình 6-3 Thông điệp lặp lại với số lần không xác định



Hình 6-4 Thông điệp lặp lại với số lần xác định

Có một số cách khác nhau để biểu diễn cho chu trình lặp, ví dụ thay vì viết

*[i := 1..10] ta có thể viết *[ i < 11].

Như đã khẳng định từ trước, khi một đối tượng nhận được một thông điệp, ví dụ :HBH nhận được *msg()* như hình 6-4, thì lớp **HBH** phải có hàm thành phần *msg()*. Mặt khác, biểu đồ cộng tác thể hiện thuật toán mô tả hoạt động của hệ thống. Vậy dựa vào biểu đồ cộng tác ở hình 6-4, người lập trình có thể viết hàm *msg()* (trong C) cho lớp **HBH** như sau:

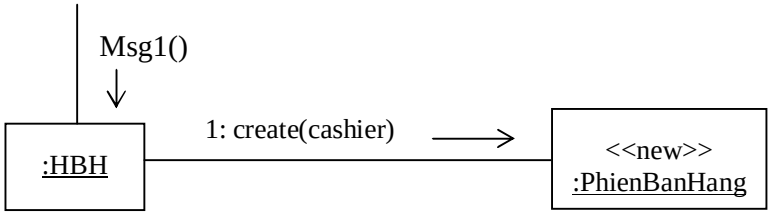
```
void msg(){
    for(int i = 1; i < 11; i++)
     dong[i] = phienBH.dongBanTiep();
}
```

Hàm này gọi tới hàm *dongBanTiep()* của lớp **PhienBanHang** thông qua đối tượng *phienBH* để đọc liên tiếp 10 *dòng bán hàng.*

*Lưu ý*: Một đối tượng có thể gửi thông điệp cho chính nó, nghĩa là thông điệp có thể quay vòng tròn.

**(3) Tạo lập đối tượng mới**

UML sử dụng thông điệp *create()* để tạo lập một đối tượng mới (một thể hiện nào đó của một lớp). Trong biểu đồ có thể sử dụng thêm ký tự <<new>> ở đối tượng nhận thông điệp, ví dụ:




Hình 6-5 Tạo lập đối tượng mới

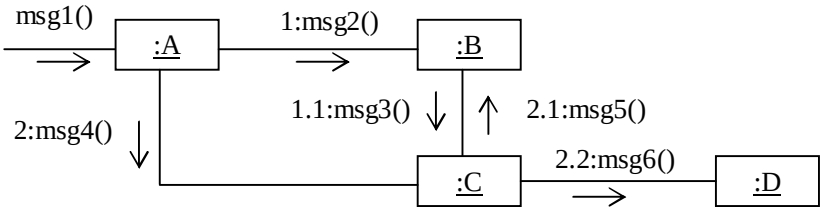
**(4) Quy tắc đánh số các thông điệp**

(i) Thông điệp đầu không được đánh số.

(ii) Những thông điệp được gửi tới cho các đối tượng trực tiếp được đánh số tăng dần 1:--, 2:--, v.v.

(iii) Các thông điệp gửi tiếp cho các đối tượng tiếp theo được đánh số theo quy tắc "dấu chấm" ('.').

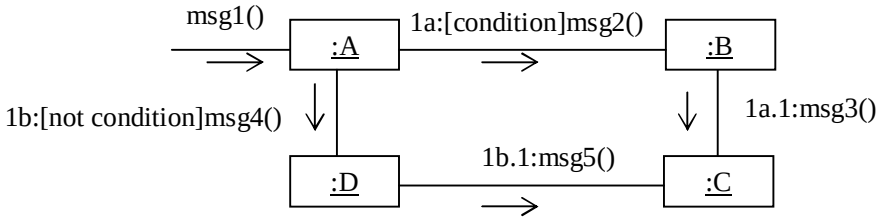
Ví dụ:




Hình 6-6 Đánh số các thông điệp

**(5) Các điều kiện gửi thông điệp**

Đôi khi, một thông điệp có thể *bị cầm canh (guarded)* và nó được gửi đến cho một đối tượng khác chỉ khi thoả mãn một điều kiện nào đó. Điều kiện *condition* được để trong cặp ngoặc '[' và ']'. Ví dụ:



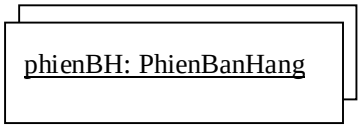

Hình 6-7 Các thông điệp được gửi đi có điều kiện

Biểu đồ trên thể hiện:

+ Thông điệp số *1a* được gửi cho :B nếu điều kiện *condition* thoả mãn,
+ Ngược lại, nếu *condition sai* (*non condition*) thì thông điệp *1b* sẽ được gửi đến cho :D.

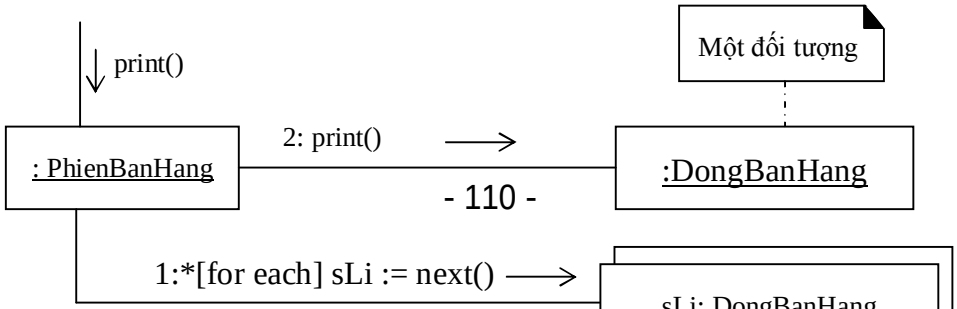
**(6) Các thông điệp gửi cho một tập (nhiều) các đối tượng**

UML ký hiệu tập các thể hiện / đối tượng như là một kho các đối tượng dạng:




Ký hiệu tập các đối tượng *phienBH* của lớp **PhienBanHang**.

Ví dụ: Đối tượng :PhienBanHang gửi thông điệp *next()* tất cả các *phienBH* và thông điệp *print()* cho một :DongBanHang bất kỳ.



Hình 6-8 Gửi thông điệp cho nhiều đối tượng

## 6.2 Thiết kế các biểu đồ cộng tác và các lớp đối tượng

Như trên đã khẳng định, nhiệm vụ chính của pha thiết kế là xây dựng các biểu đồ cộng tác mô tả chính xác các hoạt động của hệ thống và từ đó thiết kế chi tiết các lớp. Một trong những khó khăn chính của công việc trên là *gán trách nhiệm cho các đối tượng* như thế nào. Trong chương này chúng ta sẽ thảo luận nguyên lý tổng quát để gán giá trị cho các đối tượng còn được gọi là mẫu (*pattern*) gán trách nhiệm.

Việc tạo lập các biểu đồ cộng tác phụ thuộc vào các kết quả trong pha phân tích:

- *Mô hình khái niệm*: từ những mô hình này, người thiết kế có thể chọn để định nghĩa các lớp ứng với từng khái niệm trong hệ thống. Các đối tượng của những lớp này tương tác với nhau và được thể hiện trong các *biểu đồ tương tác* như biểu đồ trình tự, biểu đồ trạng thái.
- *Các hợp đồng hoạt động của hệ thống:* từ những đặc tả này, người thiết kế xác định được các trách nhiệm và các điều kiện (*pre-condition, post-condition*) trong các biểu đồ tương tác.
- *Các ca sử dụng cốt yếu và thực tế:* từ những trường hợp này, người thiết kế có thể lượm lặt được những thông tin về những nhiệm vụ mà các biểu đồ tương tác phải thoả mãn.

### 6.2.1 Ca sử dụng thực tế

Những ca sử dụng được xác định trong pha phân tích các yêu cầu thường là ít liên quan đến kỹ thuật cài đặt và cũng chưa có liện hệ nhiều với giao diện sử dụng (*User Interface*). Ở đây chúng ta thảo luận thêm về một loại ca sử dụng được gọi là *cốt yếu và thực tế*.

Một *ca sử dụng được gọi là cốt yếu* nếu nó mô tả quá trình hoạt động chủ yếu và các động cơ thúc đẩy những hoạt động đó.

Ngược lại, *ca sử dụng được gọi là thực tế* nếu nó mô tả một quá trình hoạt động thông qua những thiết kế theo thực tế và được uỷ thác cho công nghệ vào / ra đã được xác định trước.

Như vậy, ca sử dụng thực tế là một thiết kế cụ thể về một ca sử dụng, trong đó đã xác định kỹ thuật vào / ra và hỗ trợ cho cài đặt.

*Ví dụ*: hãy mô tả ca sử dụng thực tế cho ca sử dụng *Mua hàng bằng tiền mặt.* Như trước đã qui ước, ca sử dụng *Mua hàng bằng tiền mặt* được mô tả như sau:

*Ca sử dụng:* *Mua hàng bằng tiền mặt* (*Buy Items with Cash*)

*Tác nhân:* Người mua (khách hàng), người bán hàng

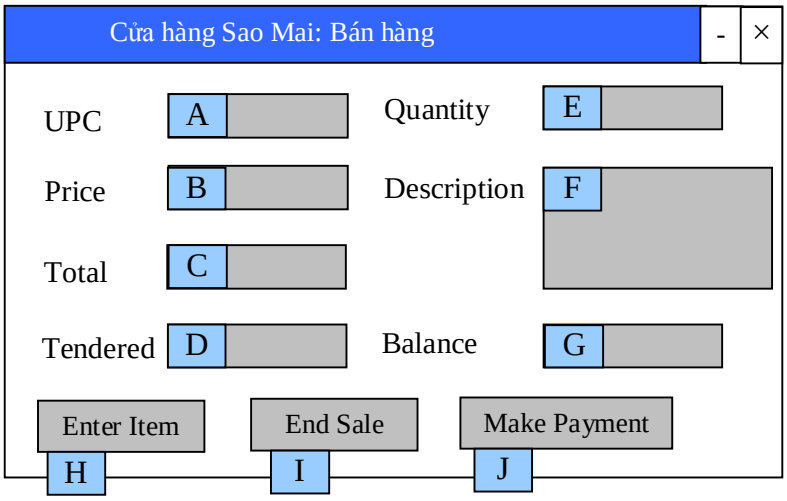
*Mục đích:* Ghi nhận các thông tin về phiên bán hàng và thu tiền hàng

*Mô tả tóm tắt*: Sau khi chọn đủ hàng, người mua đưa giỏ hàng (xe hàng) đến quầy trả tiền. Người bán ghi nhận thông tin về các mặt hàng và thu tiền bán hàng bằng tiền mặt. Thanh toán xong, khách hàng có thể đưa hàng ra khỏi cửa hàng.

*Kiểu*: Thực tế

*Tham chiếu:* R1.1, R1.2, R1.3, R1.7, R1.9, R2.1.

Trên cơ sở khảo sát bài toán thực tế, người thiết kế có thể xây dựng màn hình thực hiện nhiệm vụ trên như sau:




Hình 6-9 Màn hình giao diện của ca sử dụng thực tế "Bán hàng"

Kịch bản mô tả ca sử dụng thực tế trên được viết cụ thể như sau:

| Hoạt động của tác nhân | Hoạt động của hệ thống |
|---|---|
| 1. Ca sử dụng bắt đầu khi khách đưa hàng đến quầy trả tiền. | |
| 2. Với mỗi mặt hàng, người bán nhập vào cửa sổ **A** mã *upc*. Nếu số lượng mua nhiều hơn 1 thì nhập số đó vào cửa sổ **E**. Ấn **H** (nút *Enter Item)* sau mỗi lần nhập xong một mặt hàng. | 3. Bổ sung các thông tin của từng mặt hàng vào phiên bán hàng.<br><br>Mô tả của mặt hàng vừa nhập vào được hiển thị ở ô **F** và giá bán ở ô **B.** |

| | |
|---|---|
| 4. Khi nhập xong các mặt hàng thì ấn **I** (nút *End Sale*)**.** | 5. Tính toán và hiển thị tổng số tiền phải trả ở ô **C.** |
| 6. Khách hàng đưa một số tiền để trả tiền mua hàng, người nhập số đó vào ô **D** và ấn **J (**nút *Make Payment***), s**ố đó thường lớn hơn *total.* | 7. Hệ thống xác định số tiền trả lại cho khách và hiển thị ở ô **G.** |
| . . . | |

### 6.2.2 Mẫu gán trách nhiệm

Nhiệm vụ chính của người thiết kế là định nghĩa được các lớp để các đối tượng của chúng thực hiện được những nhiệm vụ mà hệ thống yêu cầu. Muốn vậy, chúng ta phải thiết kế được chi tiết các biểu đồ cộng tác mô tả chính xác từng hoạt động của hệ thống và gán nhiệm vụ cho các lớp đối tượng.

**Trách nhiệm**

*Trách nhiệm* (*Responsibility*) được mô tả trong hợp đồng, là nghĩa vụ của từng đối tượng. Hoạt động (hành vi) của các đối tượng liên quan chặt chẽ tới các trách nhiệm của các đối tượng đó.

Nói chung có hai loại trách nhiệm:

1. *Các trách nhiệm thực hiện:* đó là những hoạt động mà đối tượng thực hiện bao gồm:
   - Tự động thực hiện cái gì đó,
   - Khởi động một hoạt động hoặc kích hoạt một thao tác của những đối tượng khác,
   - Điều khiển hoặc phối hợp các hành động giữa các đối tượng khác nhau,
2. *Những trách nhiệm để biết:* những hiểu biết về các đối tượng, bao gồm:
   - Hiểu biết về dữ liệu riêng, được bao gói và bị che giấu,
   - Hiểu biết về những đối tượng liên quan,
   - Hiểu biết về những sự vật có thể xuất phát hoặc những tính toán nào đó.

Tất cả các thông tin trong hệ thống hướng đối tượng đều được lưu trữ theo các đối tượng và nó chỉ được xử lý khi các đối tượng đó được ra lệnh thực hiện những nhiệm vụ tương ứng. Để sử dụng được các đối tượng, ngoài các thuộc tính, chúng ta phải biết cách giao tiếp với chúng, nghĩa là phải biết chúng có những hàm thành phần nào. Điều này có thể tìm thấy trong các biểu đồ cộng tác.

**Các bước thiết kế biểu đồ cộng tác**

Việc tạo ra biểu đồ cộng tác có thể thực hiện như sau:

1. Xác định các trách nhiệm từ các ca sử dụng, mô hình khái niệm (biểu đồ lớp) và các hợp đồng hành động của hệ thống,

2. Gán trách nhiệm cho các đối tượng, quyết định xem đối tượng nào phải thực thi những trách nhiệm trên,
3. Lặp lại hai bước trên cho đến khi gán hết các trách nhiệm trong biểu đồ cộng tác.

*Lưu ý: Các trách nhiệm của đối tượng sẽ được cài đặt trong lớp của những đối tượng đó bằng các hàm thành phần (phương thức).*

Trong UML, các trách nhiệm sẽ được gán cho đối tượng khi tạo ra biểu đồ cộng tác và biểu đồ cộng tác thể hiện cả hai khía cạnh, gán trách nhiệm và sự cộng tác giữa các đối tượng trong biểu đồ đó.

### 6.2.3 Mẫu gán trách nhiệm

Những người thiết kế hướng đối tượng có kinh nghiệm thường sử dụng những nguyên lý chung và những phương pháp giải phù hợp với một ngôn ngữ lập trình, được gọi là *mẫu (pattern)* hướng dẫn để tạo ra phần mềm. Một cách đơn giản hơn, *mẫu* là cách gọi một cặp (*bài toán / lời giải*) có thể áp dụng trong những ngữ cảnh mới, với những lời khuyên làm thế nào để áp dụng được vào tình hình mới.

Sau đây chúng ta sẽ xét năm mẫu gán trách nhiệm cơ bản (**GRASP**: General Responsibility Assignment Software Patterrn): *Expert (Chuyên gia), Creator (Tạo lập mới), Low Coupling (Móc nối yếu), Hight Cohension (Cố kết chặt), và Controller (Điều khiển).*

**1. Gán trách nhiệm theo chuyên gia** (*Expert*)

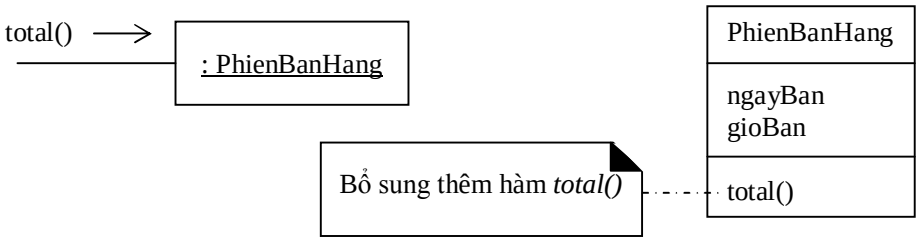
Ví dụ trong hệ thống bán hàng HBH, một số lớp cần phải biết số tiền của mỗi phiên bán hàng và để gán được các trách nhiệm thì phải trả lời:

*Ai có trách nhiệm để biết về số tiền của mỗi phiên bán hàng?*

Theo ý kiến *chuyên gia*, để trả lời câu hỏi trên thì phải tìm những lớp chứa những thông tin trên. Đó là: Tất cả các *dòng bán hàng* của mỗi *phiên bán hàng*, trong đó cần tính tổng (*total*) của các mục *thành tiền* (*subtotal*) của từng *dòng bán hàng*.

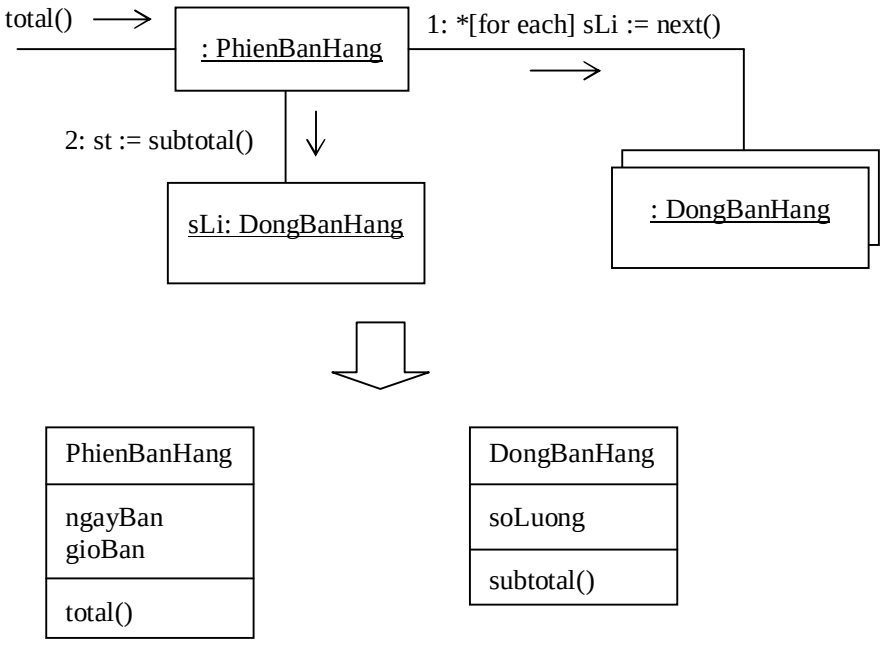
Chỉ có *phienBanHang* (đối tượng phiên bán hàng hiện thời) biết về thông tin này, vì vậy, theo chuyên gia thì gán trách nhiệm cho lớp **PhienBanHang**.

Chúng ta bắt đầu vẽ biểu đồ cộng tác liên quan đến việc gán trách nhiệm tính tiền bán hàng (*total()*) cho lớp **PhienBanHang**.



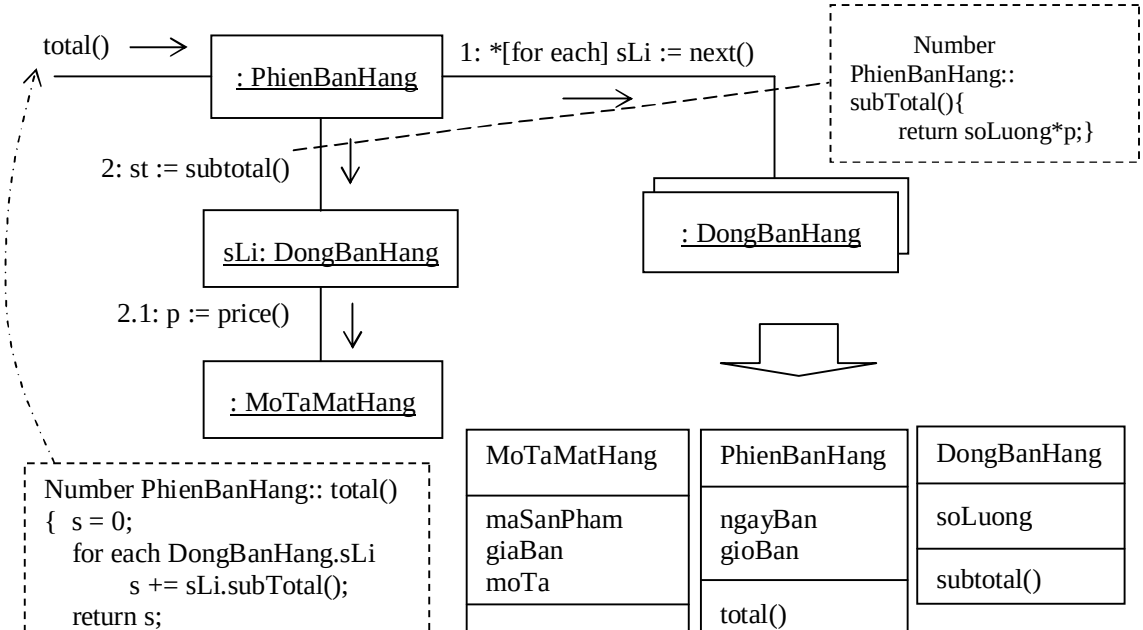

Hình 6-10 Phần đầu của biểu đồ cộng tác

Sau khi gán *total()* cho lớp **PhienBanHang** thì lại phải biết thêm là ai cung cấp những mục con (số tiền bán từng mặt hàng). Thông tin này được xác định thông qua hàm *subtotal()*. Vì vậy, :PhienBanHang phải gửi tiếp thông điệp *subtotal()* cho từng : DongBanHang như hình 6-11.




Hình 6-11 Trao đổi giữa các đối tượng để tính được *total()*

Để *biết và tính được subtotal()* thì :DongBanHang phải biết thông tin về giá bán được lấy từ đâu. Như hình 6-11 thì *subtotal()* được xác định từ :DongBanHang và là tích của *DongBanHang.soluong * MoTaMatHang.giaBan,* do vậy biểu đồ cộng tác của hoạt động tính *total()* sẽ được xây dựng như hình 6-12.



Hình 6-12 Biểu đồ cộng tác để thực hiện việc tính *total()*

*Lưu ý:* mẫu gán trách nhiệm theo ý kiến chuyên gia được áp dụng nhiều hơn những mẫu khác, nó là *nguyên lý hướng dẫn chung trong thiết kế hướng đối tượng*. Mẫu này thể hiện được những cảm giác trực quan chung về các đối tượng, chúng phải thực hiện những gì để có được những thông tin cần có. Sử dụng loại mẫu này cũng đảm bảo *duy trì được tính chất bao gói*, che giấu thông tin vì các đối tượng chỉ sử dụng những thông tin riêng mà chúng có để thực hiện những nhiệm vụ được giao.

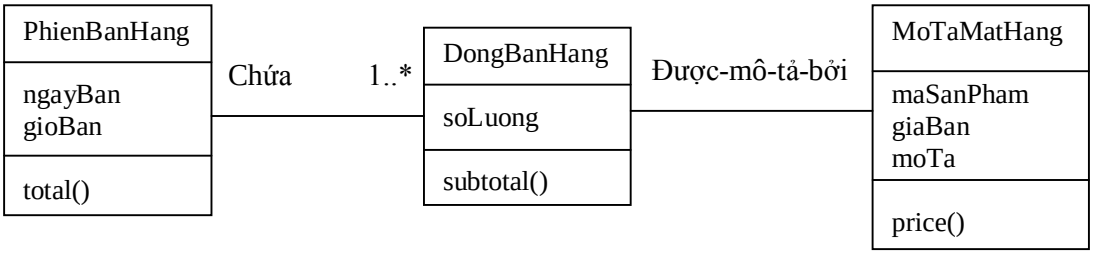
**2. Mẫu gán trách nhiệm tạo lập mới** (*Creator*)

Tạo lập các đối tượng khi cần thiết là một trong những hoạt động quan trọng của hệ thống hướng đối tượng. Do đó cần phải có nguyên lý chung để gán trách nhiệm tạo lập đối tượng trong hệ thống.

*Phương pháp giải:* **Gán trách nhiệm cho lớp B để tạo ra đối tượng của lớp A (B** là phần tử tạo lập các đối tượng của **A)** nếu có một trong các điều kiện sau:

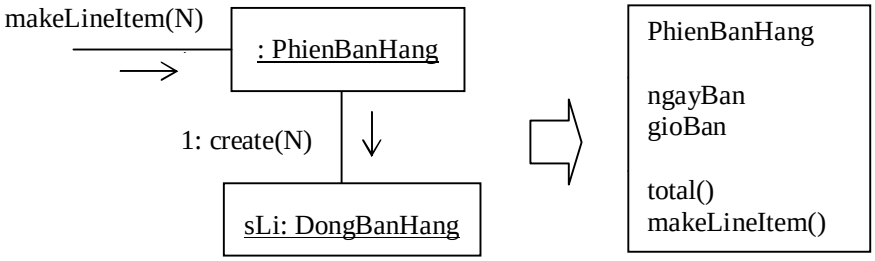
- ***B** gồm các đối tượng của **A**,*
- ***B** chứa các đối tượng của **A**,*
- ***B** ghi chép các thể hiện của **A**,*
- ***B** sử dụng trực tiếp các đối tượng của **A**,*
- ***B** có những dữ liệu ban đầu sẽ được truyền cho các đối tượng của **A** khi chúng được tạo ra.*

*Ví dụ*: hãy xét một phần của mô hình khái niệm như hình 6-13 (trích từ hình 4-10 sau khi được bổ sung thêm các hàm được xác định ở bước trước).



Hình 6-13 Một phần của biểu đồ lớp

Một :PhienBanHang chứa (thực chất là bao gồm) nhiều đối tượng :DongBanHang, do vậy theo mẫu này thì có thể gán trách nhiệm **PhienBanHang** để tạo lập các đối tượng của **DongBanHang**. Trách nhiệm này được gán khi có yêu cầu cần tạo ra một dòng bán hàng với *N* đơn vị, nghĩa là khi :PhienBanHang nhận được thông điệp *makeLineItem(N)* thì nó sẽ gửi cho :DongBanHang thông điệp *create(N)* như hình 6-14.



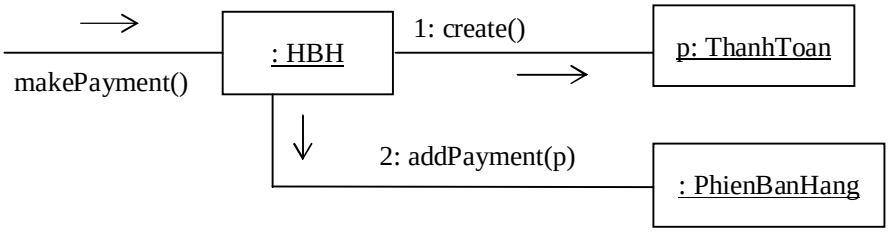

Hình 6-14 Tạo ra DongBanHang

**3. Mẫu móc nối yếu** (*Low Coupling*)

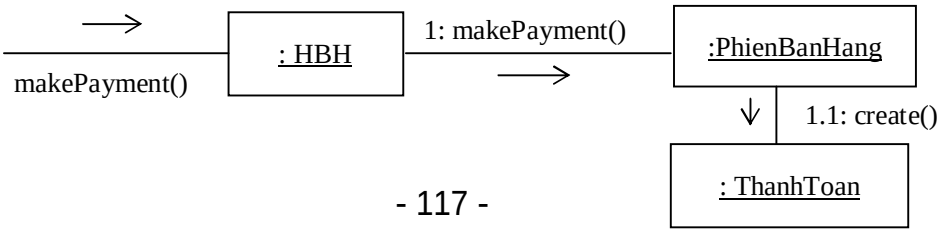
*Độ móc nối là một độ đo* về sự kết nối của một lớp để biết về, hoặc phụ thuộc vào các lớp khác như thế nào. Một lớp có độ *móc nối yếu thì không phụ thuộc nhiều vào nhiều lớp khác*. Ngược lại, một lớp có độ móc nối mạnh thì phụ thuộc vào nhiều lớp khác.

Do đó, khi gán trách nhiệm cho một lớp, hãy *cố gắng sao cho độ móc nối giữa các lớp ở mức yếu*.

*Ví dụ*: hãy xét mối quan hệ giữa các lớp **ThanhToan, HBH** và **PhienBanHang** trong hệ thống bán hàng. Giả sử cần phải tạo ra đối tượng :ThanhToan tương ứng với :PhienBanHang và :HBH nhận được yêu cầu thanh toán *makePayment()*. Bởi vì **HBH** phải " ghi nhận" :ThanhToan nên theo mẫu tạo lập mới ở trên thì lớp **HBH** là đại biểu để tạo lập. Điều này dẫn đến thiết kế biểu đồ cộng tác như hình 6-15.




Hình 6-15 HBH tạo ra :ThanhToan



Hình 6-16 ThanhToan tạo ra :ThanhToan

Với cách gán trách nhiệm như hình 6-15 thì **HBH** phải biết về lớp **ThanhToan**. Nhưng thực tế điều này không đòi hỏi, vì chỉ cần **PhienBanHang** cần biết về **ThanhToan** là đủ. Từ đó chúng ta có thiết kế tốt hơn, thể hiện được mức độ móc nối lỏng hơn như hình 6-16. Sau đó lại áp dụng các qui tắc khác để gán trách nhiệm cho các đối tượng.

Trong các ngôn ngữ lập trình hướng đối tượng như C++, Java, **ClassA** với **ClassB** được móc nối với nhau khi:

- **ClassA** có những thuộc tính mà đối tượng của **ClassB** cần tham khảo
- **ClassA** có những hàm mà đối tượng của **ClassB** cần sử dụng, hay tham chiếu
- **ClassA** là lớp con của **ClassB**.

**4. Cố kết cao** (Hight Cohension)

*Cố kết là độ đo* về mức độ liên kết trong lớp, tập trung vào các trách nhiệm được gán cho mỗi lớp. Một lớp có *tính cố kết cao* nếu các nhiệm vụ có liên quan chức năng với nhau. *Lớp cố kết là lớp có tối thiểu số các hàm đơn giản và chúng phải có liên hệ chức năng với nhau*.

Vấn đề quan trọng trong thiết kế hướng đối tượng là phải gán được trách nhiệm cho các lớp sao cho một mặt phù hợp với thực tế của bài toán, mặt khác đảm bảo có mối liên hệ chặt chẽ về chức năng nhưng không để có những lớp phải làm việc quá tải.

*Ví dụ:* khi xét mối quan hệ giữa các lớp **ThanhToan, HBH** và **PhienBanHang** trong hệ thống bán hàng ta có thể đưa ra một thiết kế biểu đồ cộng tác như hình 5-15. Vì HBH là lớp chính, nên việc để cho lớp **HBH** đảm nhận vai trò tạo lập *create()* :ThanhToan và thực hiện thêm nhiều công việc khác có thể dẫn đến quá tải. Mặt khác, nhiệm vụ *create()* không thực sự liên hệ với các nhiệm vụ còn lại, nên có thể gán trách nhiệm này cho lớp **PhienBanHang** như hình 6-16 thì hợp lý hơn.

Những lớp không cố kết sẽ khó hiểu, khó sử dụng lại và rất khó duy trì hoạt động của hệ thống cho có hiệu quả.

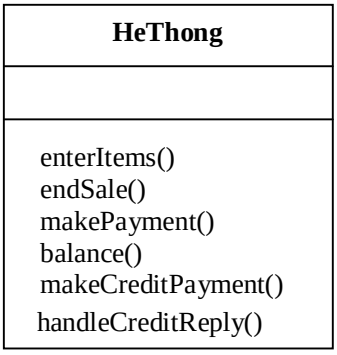
**5. Mẫu điều khiển** (*Controller*)

Như trên đã phân tích, những sự kiện vào (input) sẽ tác động và làm cho hệ thống được kích hoạt. Việc gán trách nhiệm để xử lý các sự kiện vào của hệ thống cho các lớp được thực hiện như sau:

- Gán trách nhiệm cho những lớp đại diện cho cả hệ thống (điều khiển bề mặt),
- Gán trách nhiệm cho những lớp đại diện cho toàn bộ nghiệp vụ hoặc cho cả tổ chức (điều khiển bề mặt),
- Gán trách nhiệm cho những lớp đại diện cho cái gì đó trong thế giới thực mà nó là tích cực và có thể bị lôi kéo theo trong công việc (điều khiển vai trò),
- Gán trách nhiệm cho những lớp đại diện cho bộ phận nhân tạo để xử lý các sự kiện vào ở mỗi ca sử dụng thường được đặt tên "<*UseCaseName*> *Handler*" (điều khiển ca sử dụng).

*Ví dụ:* trong hệ thống bán hàng, chúng ta đã xác định một số thao tác chính như: e*nterItems()* (nhập dữ liệu của mặt hàng), *endSale()* (kết thúc việc nhập dữ liệu của một phiên bán hàng) và *makePayment()* (thu tiền, thanh toán), v.v.

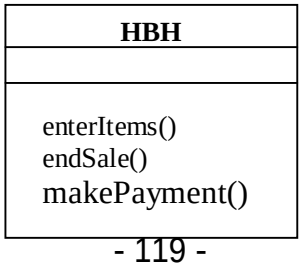
Trong pha phân tích, những thao tác này của hệ thống đã được ghi nhận vào lớp **HeThong.**




Hình 6-17 Các thao tác của hệ thống được ghi nhận vào lớp có tên là **HeThong**

Tuy nhiên, điều này không có nghĩa là lớp có tên **HeThong** phải thực hiện tất cả những nhiệm vụ đó. Trong giai đoạn thiết kế, những thao tác của hệ thống có thể được gán cho lớp điều khiển.

Do vậy, ở pha thiết kế có thể gán trách nhiệm thực hiện các thao tác của hệ thống cho một hay một số lớp điều khiển như gán cho lớp **HBH**: đại diện cho cả hệ thống.



Hình 6-18 Gán các thao tác của hệ thống cho lớp điều khiển

## 6.3 Thiết kế hệ thống HBH

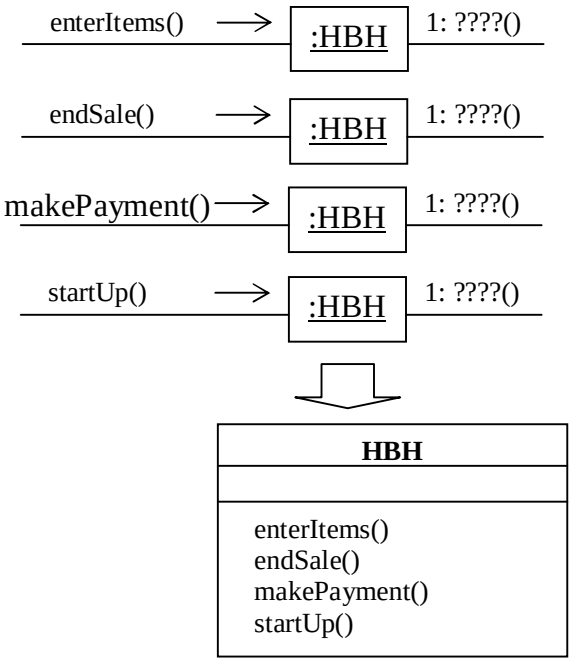
Trong phần này chúng ta hãy áp dụng GRASP để gán trách nhiệm cho các lớp và tạo ra các biểu đồ cộng tác. Chúng ta thực hiện mẫu đối với ca sử dụng "*Mua Hàng bằng tiền mặt*" (*Buy Items with Cash*) và "Khởi động" (*Start Up*), sau đó thực hiện tương tự đối với những ca sử dụng khác.

Khi thiết kế biểu đồ cộng tác, chúng ta hãy thực hiện theo những hướng dẫn sau:

1. Với mỗi thao tác (hoạt động chính) đã được mô tả trong các *hợp đồng* của hệ thống nên tạo ra một biểu đồ riêng.
2. Nếu biểu đồ này phức tạp thì có thể chia nó thành những biểu đồ đơn giản hơn.
3. Sử dụng các hợp đồng trách nhiệm, trong đó dựa vào những điều kiện cần thoả mãn sau khi hoàn thành (*post-condition*) và các mô tả ca sử dụng để xác định trách nhiệm của các đối tượng trong hệ thống theo mẫu gán trách nhiệm như đã nêu trên.

Để thực hiện ca sử dụng "*Mua Hàng bằng tiền mặt*" (*Buy Items with Cash*) và "Khởi động" (*Start Up*), hệ thống phải thực hiện: *enterItems()* (nhập các mặt hàng), *endSale()* (kết thúc một phiên bán hàng), *makePayment()* (thanhToan) và *startUp()* (khoiDong). Theo hướng dẫn trên, chúng ta thiết kế các biểu đồ cộng tác cho những thao tác đó.

Chúng ta bắt đầu từ bốn biểu đồ như hình 6-19.



Hình 6-19 Các thao tác của hệ thống

**Biểu đồ cộng tác cho** *enterItems()*

Trước hết chúng ta xem lại *hợp đồng* thực hiện e*nterItems() để biết* hệ thống phải làm những gì.

*1. Hiển thị thông tin mô tả và giá bán của mặt hàng được nhập vào.*

Nói chung, các đối tượng của hệ thống không có trách nhiệm hiển thị các thông tin. Những công việc này có thể được các đối tượng giao diện (*Interface Object*) đảm nhiệm bằng cách truy nhập vào CSDL về các mặt hàng và gửi các thông điệp chứa những thông tin tương ứng cho các đối tượng đang hoạt động.

2. *Tạo lập phiên bán hàng mới.*

Như trong hợp đồng đã nêu rõ: khi nhập vào mặt hàng đầu tiên của mỗi phiên bán hàng thì phải tạo lập *phienBanHang* mới. Trách nhiệm này được gán cho **HBH** và nó có nhiệm vụ là phải ghi lại các phiên bán hàng như thế.

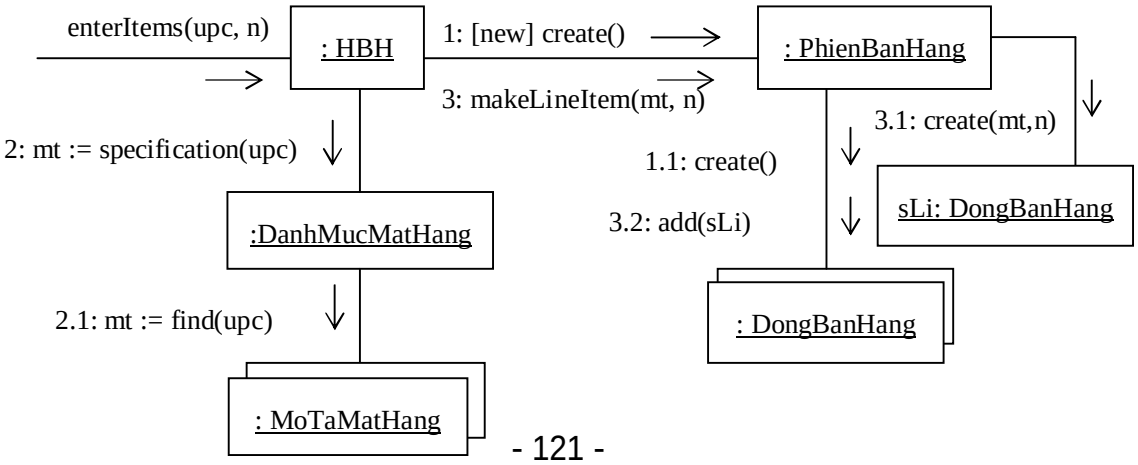
Hơn nữa, mỗi khi *phienBanHang* được tạo mới thì nó là tập rỗng và sau đó được bổ sung thêm các *dongBanHang* mỗi khi nhập xong một mặt hàng.

Áp dụng các mẫu gán trách nhiệm nêu trên, gán việc tạo lập *dongBanHang* mới cho **PhienBanHang l**à hợp lý.

*3. Xác định các thông tin mô tả và giá bán.*

Sau khi các *dongBanHang* được tạo lập thì phải được đưa bổ sung vào *phienBanHang* đang hoạt động, do đó nó cần gửi thông điệp *add()* (bổ sung vào) cho những *dongBanHang* đang được nhập vào. Những *dongBanHang* được đưa vào *phienBanHang* phải được xác định từ **DanhMucMatHang** và sau đó là **MoTaMatHang**. Vậy để có được những thông tin trên thì HBH phải gửi thông điệp *specification(upc),* xác định mô tả mặt hàng có mã là *upc* cho :DanhMucMatHang, đối tượng này lại gửi tiếp đi thông điệp *find(upc),* để tìm trong tập các mô tả mặt hàng có mã *upc.*

Dựa vào những thảo luận như trên, biểu đồ cộng tác sẽ được xây dựng như sau:



Hình 6-20 Biểu đồ cộng tác cho *enterItems()*

**Tầm nhìn của các lớp**

Như đã khẳng định nhiều lần, các đối tượng trong hệ thống tương tác với nhau bằng cách trao đổi các thông điệp, cụ thể hơn như trong các biểu đồ tương tác là trao đổi thông qua các lời gọi hàm. Trong biểu đồ ở hình 6-20, khi hệ thống cần xác định những thông tin về mặt hàng có mã *upc* cho trước, như giá bán chẳng hạn thì nó gửi tới cho :DanhMucMatHang lời gọi hàm *specification(upc)*, đối tượng này lại gửi cho :MoTaMatHang lời gọi hàm *find(upc)*.

Một đối tượng muốn có được những thông tin từ những đối tượng khác thì đối tượng đó phải có khả năng nhìn thấy được những gì mà nó cần thiết. Một cách hình thức hơn,

> *Để đối tượng :A có thể gửi một thông điệp cho đối tượng :B thì lớp A phải được liên kết với lớp B.*

Trong thiết kế hướng đối tượng, sự liên kết có liên quan chặt chẽ với khái niệm *khả năng nhìn thấy được của các đối tượng*.

- Nếu :A có thể nhìn thấy :B thì phải có một liên kết giữa hai đối tượng đó, nghĩa là giữa hai lớp tương ứng có quan hệ kết hợp.
- Nếu giữa hai đối tượng :A và :B hiện thời có liên kết với nhau thì một trong hai đối tượng đó có một đối tượng nhìn thấy đối tượng kia.

Về sự trao đổi thông điệp giữa các đối tượng có thể phát biểu chính xác như sau:

> *Để đối tượng :A gửi được thông điệp cho đối tượng :B thì hiện thời đối tượng :A phải nhìn thấy được đối tượng :B.*

Có bốn cách để đối tượng A nhìn thấy được đối tượng :B.

1. *Tầm nhìn thuộc tính:* :B là thuộc tính của :A.
2. *Tầm nhìn tham số:* :B là tham số của một hàm nào đó của :A.
3. *Tầm nhìn khai báo cục bộ:* :B được khai báo là đối tượng cục bộ trong định nghĩa hàm của :B.
4. *Tầm nhìn toàn cục:* :B là toàn cục.

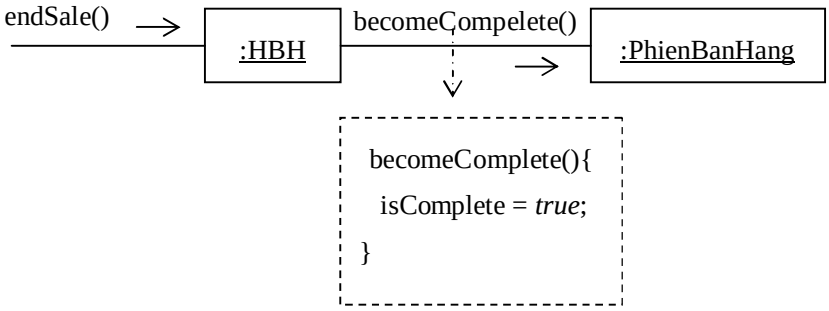
Dựa vào phạm vi quan sát của các đối tượng trong các biểu đồ để khai báo các đặc tính *private, protected, public cho các thuộc tính và hàm thành phần trong các lớp đối tượng.*

**Biểu đồ cộng tác cho** *endSale*

Có thể chọn HBH để điều khiển thao tác này của hệ thống. Hợp đồng của thao tác này yêu cầu:

- *PhienBanHang.isComplete* được gán trị *true* khi kết thúc nhập dữ liệu.

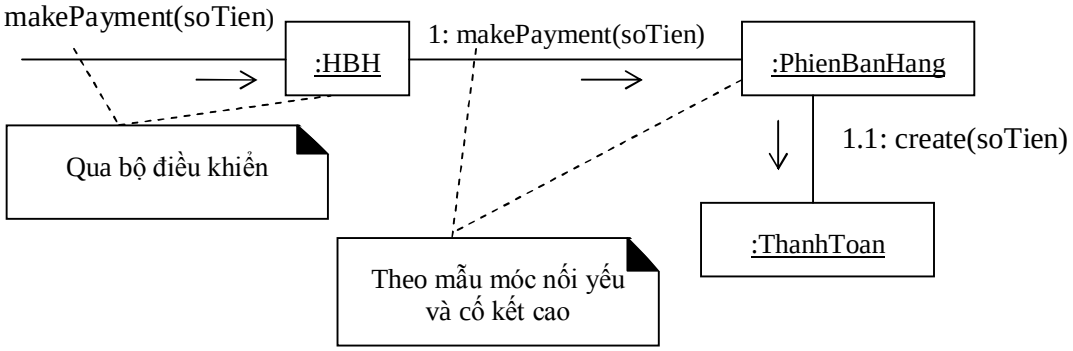
Như vậy, hệ HBH có thể gửi thông điệp *becomeComplete() cho PhienBanHang* để đặt thuộc tính *isComplelet nhận giá trị true.*




Hình 6-21 Biểu đồ cộng tác thể hiện *Kết thúc nhập dữ liệu endSale()*

**Biểu đồ cộng tác cho makePayment**

Một lần nữa nhắc lại, chúng ta sẽ xem HBH như là bộ điều khiển. Trong phần thảo luận về mẫu gán trách nhiệm để đảm bảo mức độ móc nối giữa các lớp yêu, nhưng độ cố kết lại cao, chúng ta đã gán trách nhiệm tạo lập *đối tượng ThanhToan* cho lớp **PhienBanHang**, chứ không gán cho lớp **HBH** (hình 6-16). Biểu đồ cộng tác mô tả thao tác *makePayment()* được vẽ lại như sau:



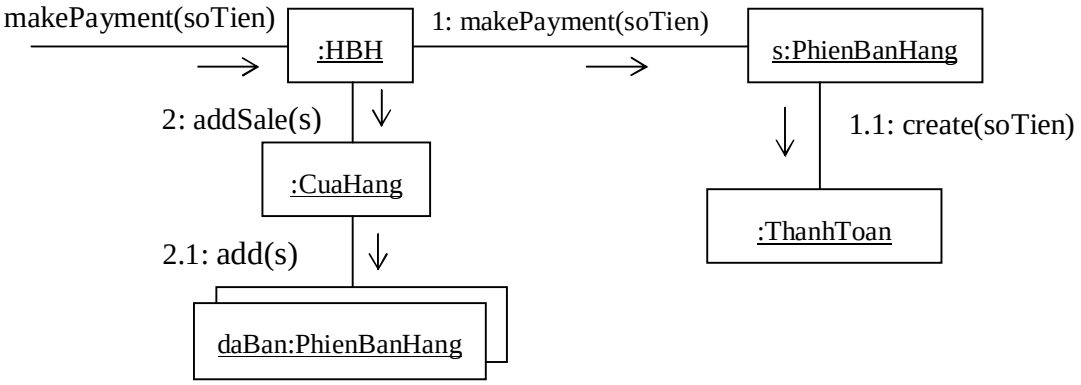

Hình 6-22 Biểu đồ cộng tác cho *makePayment()*

Biểu đồ này đáp ứng những điều kiện sau:

- Một đối tượng của lớp **ThanhToan** để thanh toán,
- **ThanhToan** được kết hợp với **PhienBanHang**
- *ThanhToan.soluong = soTien.*

**Ghi nhận những phiên bán hàng đã kết thúc**

Mỗi khi kết thúc một phiên bán hàng, theo yêu cầu trong hoạt động kinh doanh của Công ty, hệ thống phải ghi lại ký sự của những phiên bán hàng đó. Theo kinh nghiệm của chuyên gia, ta có thể gán trách nhiệm này, trách nhiệm *addSale()* cho **CuaHang** sau khi đã gán trách nhiệm *makePayment*() cho **PhienBanHang**. Biểu đồ ở hình 6-22 được bổ sung thành:




Hình 6-23 Biểu đồ cộng tác cho *makePayment()* và ghi nhận thông tin đã bán

**Biểu đồ cộng tác cho ca sử dụng StartUp**

Phần lớn các hệ thống (nhưng không phải tất cả) đều có ca sử dụng *Khởi động (StartUp)* và một số thao tác liên quan đến việc khởi tạo các giá trị cho một ứng dụng khi bắt đầu thực thi. Mặc dù thao tác này thường là thao tác đầu tiên hệ thống phải thực hiện, nhưng chúng ta để lại thiết kế sau để đảm bảo mọi thông tin liên quan đến các hoạt động sau này của hệ thống đều đã được phát hiện.

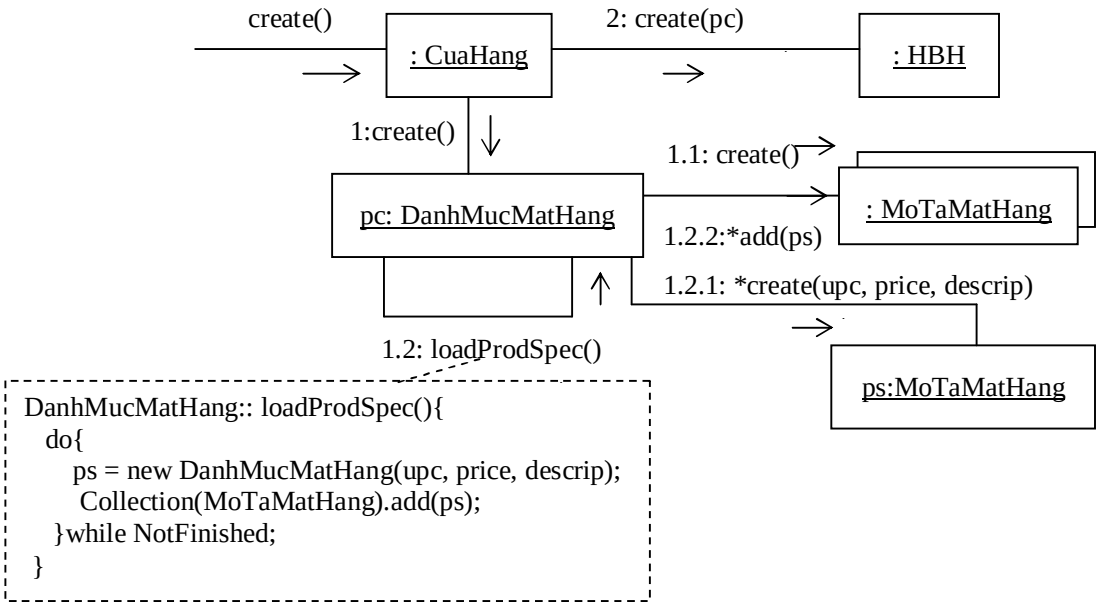
Biểu đồ cộng tác của *StartUp* chỉ ra những gì có thể xảy ra khi đối tượng của miền ứng dụng được tạo ra (created). Nghĩa là trong hệ thống bán hàng phải gán trách nhiệm *create*() cho những lớp chính, đó là **HBH** hoặc **CuaHang**.

Chúng ta chọn **CuaHang** bởi vì **HBH** đã được chọn làm bộ điều khiển cho các hoạt động của cả hệ thống.

Căn cứ vào hợp đồng của *StartUp* và những thảo luận ở trên, ta có thể thiết kế biểu đồ cộng tác cho *StartUp* như sau:

1. Bắt đầu gửi thông điệp *create()* cho **CuaHang,**
2. Để tạo ra đối tượng thuộc lớp **HBH** và cho phép những đối tượng đó gửi được các thông điệp cho **DanhMucMatHang** để yêu cầu các thông tin về các mặt hàng (xem biểu đồ cộng tác của *enterItems*) thì trước tiên nó cần phải tạo ra các đối tượng **DanhMucMatHang**. Nghĩa là nó gửi thông điệp *create()* trước cho **DanhMucMatHang**, sau đó mới gửi cho **HBH** thông điệp tương tự.
3. Khi một đối tượng **DanhMucMatHang** đã được tạo lập thì nó sẽ yêu cầu tạo lập **MoTaMatHang** và sau đó bổ sung vào danh sách các mô tả mặt hàng, đồng thời bản thân nó tự nạp những thông tin mô tả những mặt hàng tiếp theo.

Biểu đồ này được vẽ như hình 6-24.




Hình 6-24 Biểu đồ cộng tác cho StartUp

**Sự chênh lệch giữa phân tích và thiết kế**

Trong biểu đồ cộng tác cho StartUp mới chỉ thể hiện việc tạo ra *một đối tượng* đại diện cho một điểm bán hàng đầu cuối. Trong khi ở mô hình khái niệm ở pha phân tích, nó được xây dựng để mô hình cho những cửa hàng thực tế có thể có nhiều điểm bán hàng đầu cuối, nghĩa là khi hệ thống hoạt động thì sẽ có nhiều đối tượng của **HBH** được tạo ra.

Từ đó có thể thấy có một số điển chênh lệch giữa kết quả phân tích và thiết kế:

1. *Đối tượng* :CuaHang trong các biểu đồ không phải là cửa hàng thực, nó là đối tượng mềm,
2. *Đối tượng* :HBH trong các biểu đồ không phải là điểm bán hàng đầu cuối cụ thể, nó là đối tượng mềm,
3. Biểu đồ cộng tác thể hiện mối quan hệ tương tác giữa một đối tượng :CuaHang và :HBH.
4. Tổng quát hoá từ một đối tượng của **HBH** sang nhiều đối tượng đòi hỏi **CuaHang** phải tạo ra nhiều thể hiện. Và những sự tương tác giữa các đối tượng **HBH** với nhiều đối tượng khác cần phải được đồng bộ hoá để đảm bảo sự toàn vẹn trong việc chia sẻ các thông tin trong hệ thống. Điều này dẫn đến tính toán đa luồng (*multi-thread computation*) hoặc tính toán tương tranh

(*concurrent computation*), vấn đề này vượt ra ngoài phạm vi của đề tài đang thảo luận ở đây.

## 6.4 Thiết kế chi tiết các biểu đồ lớp

Khi tạo ra các biểu đồ cộng tác, chúng ta đã ghi nhận những phương thức (hàm) tương ứng và được gán vào cho các lớp (như hình 6-22, 6-23, 6-24, v.v.).

Các lớp cùng với các hàm đã xác định là những lớp phần mềm (*software class*) biểu diễn cho những lớp khái niệm trong mô hình khái niệm. Dựa vào những lớp phần mềm và dựa vào các biểu đồ khái niệm, biểu đồ cộng tác, chúng ta xây dựng các thiết kế chi tiết biểu đồ lớp để thể hiện được những thông tin sau:

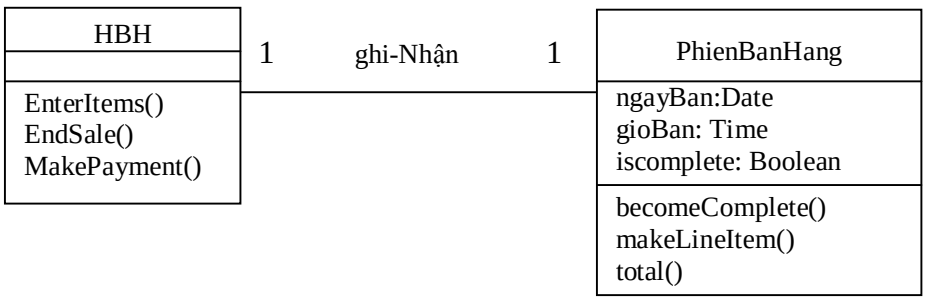
- Các lớp, các thuộc tính và các mối quan hệ kết hợp,
- Các hàm thành phần,
- Các kiểu của các thuộc tính,
- Tình trạng có thể điều khiển được,
- Sự phụ thuộc giữa các lớp.

**Các bước thực hiện để thiết kế biểu đồ lớp**

1. Xác định tất cả các lớp có các đối tượng tương tác với nhau. Điều này thực hiện được thông qua phân tích các biểu đồ tương tác.
2. Vẽ chúng trong một biểu đồ lớp.
3. Xác định thuộc tính của chúng (sao chép từ các khái niệm trong biểu đồ khái niệm) và bổ sung cho đầy đủ.
4. Phân tích các biểu đồ cộng tác để xác định các hàm và bổ sung cho các lớp.
5. Xác định các kiểu của các thuộc tính và các giá trị trả lại của phép toán.
6. Bổ sung những mối liên kết cần thiết để quản lý các quyền truy nhập (khả năng nhìn thấy) của các thuộc tính.
7. Bổ sung các quan hệ phụ thuộc dữ liệu.
8. Xác định mối quan hệ tổng quát hoá / chi tiết hoá và bổ sung quan hệ kế thừa vào biểu đồ lớp.

Trước khi bắt tay thiết kế biểu đồ lớp, chúng ta cần phân biệt mô hình khái niệm (biểu lớp phân tích) với biểu đồ lớp thiết kế.

Trong mô hình khái niệm, ví dụ:



Hình 6-25 Hai lớp trong mô hình khái niệm

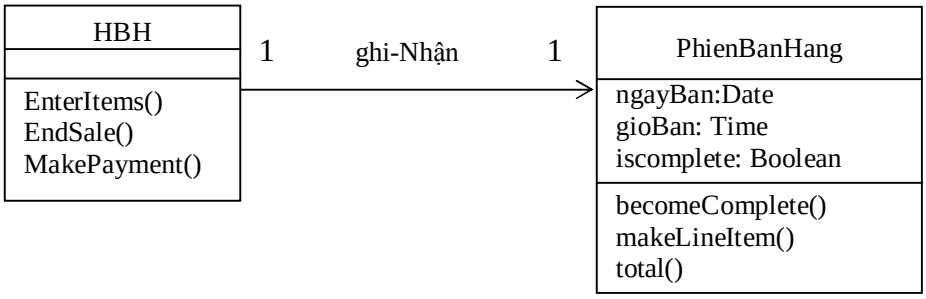
Các lớp: **PhienBanHang** và **HBH** ở đây là các khái niệm trừu tượng.

Trong pha thiết kế, các lớp trên là các thành phần của hệ thống.

Các bước thiết kế lớp từ 1. đến 5. hầu như đã được thực hiện dần từ giai đoạn phân tích và khá đơn giản.. Ở đây chủ yếu tập trung giới thiệu cách thực hiện ba bước cuối cùng để hoàn thiện thiết kế biểu đồ lớp.

**6. Bổ sung các mối quan hệ kết hợp và khả năng điều khiển được**

Như chúng ta đã phân tích, giữa hai lớp trong hệ thống thường có những mối quan hệ xác định, trong đó phổ biến là quan hệ kết hợp (*association*). Mỗi đầu của quan hệ kết hợp có vai trò (*role*) nhất định. Trong thiết kế biểu đồ lớp, vai trò có thể được gán với mũi tên chỉ hướng điều khiển. *Khả năng điều khiển* (*Navigability*) là đặc tính của vai trò và chỉ ra rằng nó có thể điều khiển một chiều thông qua quan hệ kết hợp từ đối tượng của lớp nguồn tới đối tượng của lớp đích. *Ví dụ*: biểu đồ lớp ở hình 6-25 được bổ sung thêm chiều điều khiển như hình 6-26.




Hình 6-26 Chỉ rõ hướng điều khiển của vai trò trong biểu đồ lớp

Khả năng điều khiển được trong biểu đồ lớp thường được thể hiện như là khả năng nhìn thấy được của các thuộc tính của lớp đích từ lớp nguồn. Trong cài đặt bằng ngôn ngữ lập trình, nó được thực hiện bằng cách xây dựng lớp nguồn có những thể hiện là các đối tượng của lớp đích. Ví dụ: khi cài đặt các lớp ở hình 6-26, lớp **HBH** sẽ có thuộc tính tham chiếu tới đối tượng của lớp **PhienBanHang**.

Khả năng điều khiển và quan hệ kết hợp giữa các lớp đã được chỉ ra trong các biểu đồ cộng tác. Chúng ta căn cứ vào các tình huống gợi ý để xác định mối kết hợp và khả năng điều khiển từ **A** tới **B**:

- :A gửi một thông điệp tới cho :B thì **A** kết hợp với **B** theo chiều từ **A** tới **B**.
- **A** tạo ra một đối tượng của **B** thì **A** kết hợp với **B** theo chiều từ **A** tới **B**.
- **A** có quan hệ kết nối với **B** thông qua thuộc tính có giá trị thuộc kiểu **B,** thì **A** kết hợp với **B** theo chiều từ **A** tới **B**.
- :A nhận được một thông điệp trong đó có đối tượng của **B** đóng vai trò là tham số thì **A** kết hợp với **B** theo chiều từ **A** tới **B**.

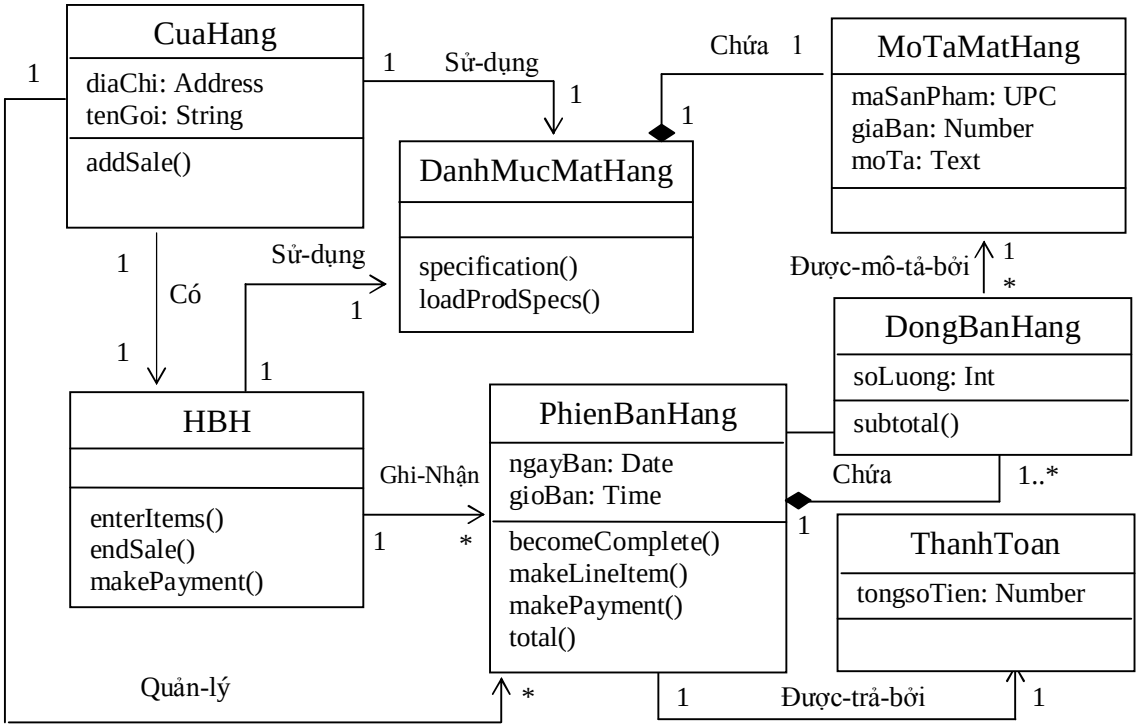
Dựa vào những qui ước nêu trên chúng ta thiết kế một phần biểu đồ lớp có đủ các quan hệ kết hợp và chiều điều khiển như hình 6-27.

## 7. Bổ sung các quan hệ phụ thuộc dữ liệu

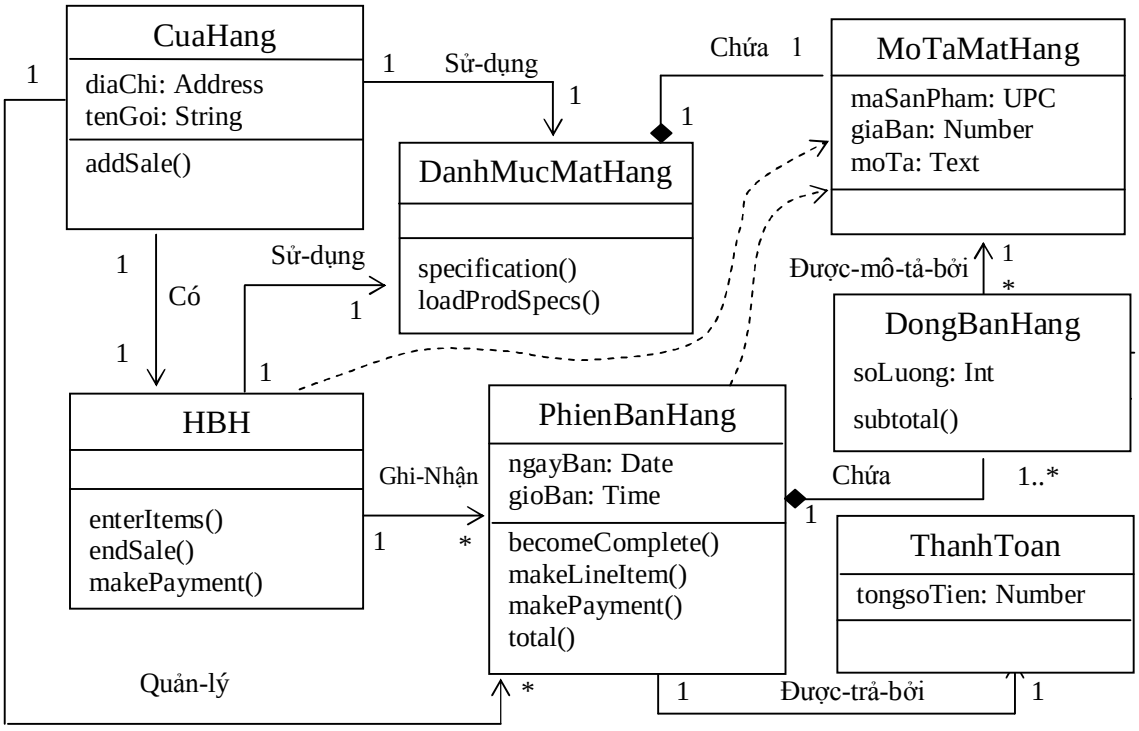
Ngoài khả năng nhìn thấy được của các thuộc tính còn có ba khả năng khác, gồm: tầm nhìn tham số, tầm nhìn khai báo cục bộ và tầm nhìn toàn cục. Trong thiết kế, khả năng nhìn thấy giữa các lớp được thể hiện bằng quan hệ phụ thuộc, được mô tả bằng đường đứt nét có mũi tên chỉ rõ lớp nguồn phụ thuộc vào lớp đích. Ví dụ:

1. Đối tượng của **HBH** nhận được thông tin mô tả mặt hàng (**MoTaMatHang**) sau khi nó gửi thông điệp đến cho **DanhMucMatHang**. Do vậy, **MoTaMatHang** phải được khai báo cục bộ trong **HBH**, nghĩa là **HBH** phụ thuộc vào **MoTaMatHang**.
2. Tương tự, **PhienBanHang** nhận được **MoTaMatHang** như là tham số trong hàm *makeLineItem()*, nghĩa là **MoTaMatHang** nằm trong tầm nhìn tham số của **PhienBanHang**. Vậy, **PhienBanHang** cũng phụ thuộc vào **MoTaMatHang**.

Từ đó chúng ta có biểu đồ lớp cùng quan hệ phụ thuộc như hình 6-28.



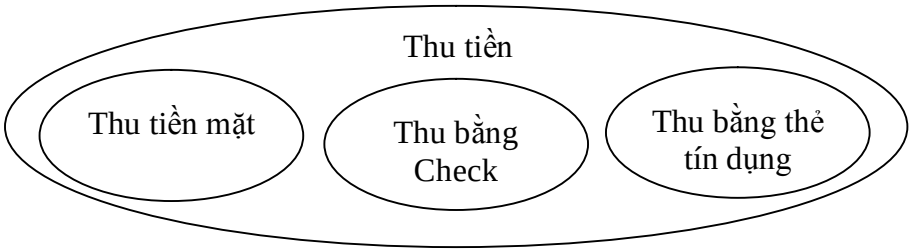

Hình 6-27 Thiết kế biểu đồ lớp được bổ sung quan hệ kết hợp




Hình 6-28 Thiết kế biểu đồ lớp được bổ sung quan hệ phụ thuộc

## 8. Xác định lớp tổng quát và bổ sung các quan hệ kế thừa

Như ở pha phân tích đã nêu, ca sử dụng *"Thu tiền" (Thanh toán)* có thể chia nhỏ thành các ca sử dụng con: "*Thu bằng thẻ tín dụng*" (*makeCreditPayment*), "*Thu tiền mặt"* (*makeCashPayment*), và "*Thu bằng Check*" (*makeCheckPayment*) tuỳ thuộc vào phương thức thanh toán của khách.




Hình 6-29 Phân tích tiếp ca sử dụng *Thu tiền* từ hình 3-6

Trong các phần trước chúng ta đã phân tích ca sử dụng "*Thu tiền mặt"* và đã xác định được các lớp đối tượng để thực hiện ca sử dụng này. Bằng cách làm tương tự như trên, chúng ta tiếp tục phân tích hai ca sử dụng còn lại để xác định những lớp tương ứng. Có thể căn cứ vào các kịch bản mô tả ca sử dụng để tìm các lớp đối tượng.

Mô tả chi tiết các ca sử dụng trên:

*Thu bằng thẻ tín dụng*

| Các tác nhân | Hệ thống |
|---|---|
| 1. Ca sử dụng này được bắt đầu thực hiện khi khách được thông báo tổng số tiền phải trả và họ chọn phương thức thanh toán bằng Credit.<br>2. Khách hàng đưa thẻ Credit và thẻ đưa đọc qua đầu đọc. | |
| | 3. Hệ thống phát sinh yêu cầu kiểm tra thẻ Credit và gửi tới bộ phận dịch vụ kiểm tra thẻ *CreditAuthorization Service* (CAS) qua modem được gắn với HBH. Hệ thống chờ trả lời.<br>4. Khi nhận được kết quả trả lời về tính hợp pháp của thẻ từ CAS, hệ thống ghi lại kết quả trả lời.<br>5. Nếu thẻ Credit hợp lệ thì nó được gửi tới bộ phận tài vụ, *Account* số tiền và thẻ bị trừ đi số tiền phải trả.<br>6. Hiển thị kết quả. |

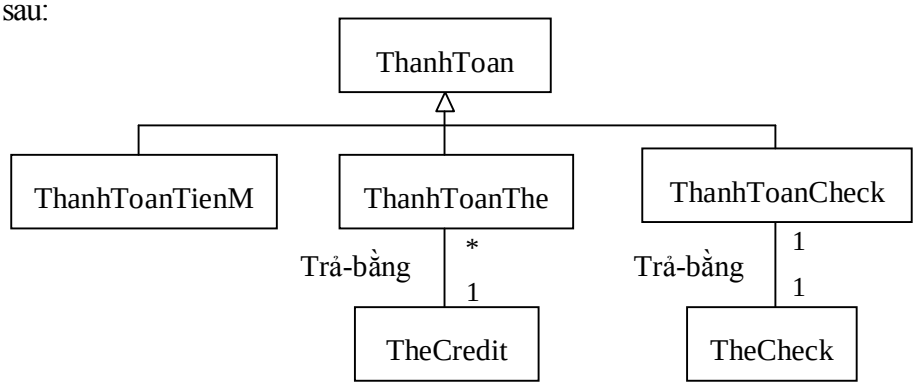
*Thu bằng Check*

| Các tác nhân | Hệ thống |
|---|---|
| 1. Ca sử dụng này được bắt đầu thực hiện khi khách hàng được thông báo số tiền phải trả và chọn phương pháp thanh toán trả séc (Check).<br>2. Khách hàng viết Check, đưa Check và xuất trình *drivers license* (giấy phép sử dụng check) cho người bán hàng. | |
| 3. Người bán kiểm tra *drivers license* và *Check* và nhấn nút: *CheckAuthorization* để yêu cầu kiểm tra. | |
| | 4. Hệ thống phát sinh yêu cầu kiểm tra Check và gửi nó cho bộ phận kiểm tra CheckAuthorization Service qua modem gắn với HBH.<br>5. Nhận được kết quả kiểm tra hệ thống ghi lại các thông tin trên Check.<br>6. Hiển thị kết quả xử lý và thông báo cho khách hàng. |

Do vậy, ngoài những lớp đã phân tích thiết kế ở trên, chúng ta cần bổ sung thêm các lớp có kiên quan:

**ThanhToanTienM** (CashPayment) **ThanhToanThe** (*CreditPayment*), **ThanhToanCheck** (*CheckPayment*). Liên quan đến những lớp này là **TheCredit** (*CreditCard*) và **Check.** Ngoài ra còn cần những lớp phục vụ cho việc kiểm duyệt thẻ Credit và Check là **CreditAutorizationService** và **CheckAutorizationService.**

Chúng ta dễ nhận thấy các lớp **ThanhToanTienM, ThanhToanThe và ThanhToanCheck** là khá giống nhau về tính chất và hành vi. Do vậy, có thể xem chúng như là kiểu con (*subtype*) kế thừa từ lớp **ThanhToan**. Mặt khác, để thực hiện được thanh toán bằng thẻ phải sử dụng **TheCredit** và để trả bằng Check thì phải có **TheCheck**. Mỗi **TheCredit** có thể mua hàng được nhiều lần, còn mỗi tờ **TheCheck** chỉ mua được một lần. Các mối quan hệ trên của chúng được thể hiện trong biểu đồ như sau:




Hình 6-30 Các lớp kế thừa của ThanhToan

**Một số lưu ý về lớp tổng quát (lớp cơ sở) và các lớp con**

Trong mối quan hệ kế thừa giữa các lớp luôn yêu cầu phải thoả mãn hai qui tắc sau:

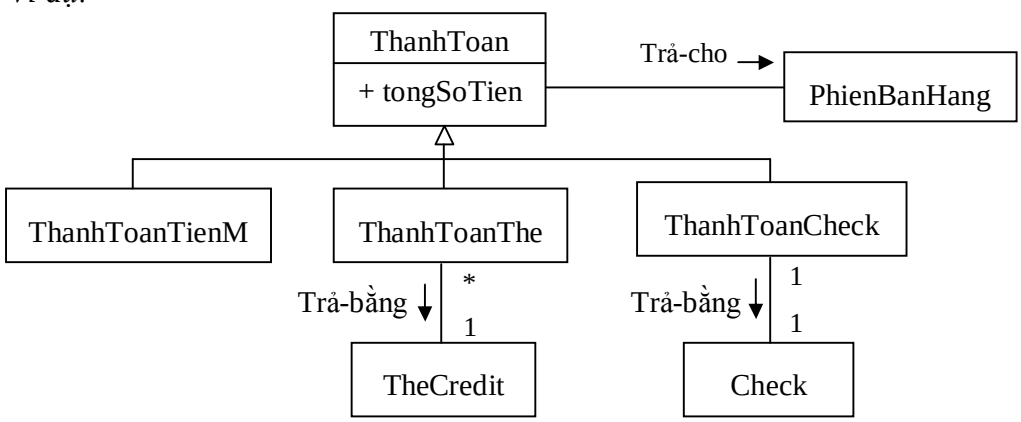
*1. Luật thành viên* (*Is-a Rule*)*:*

> *Tất cả các thành viên của lớp con (lớp kế thừa) cũng là thành viên của lớp cha (lớp tổng quát hơn).*

*2. Luật phù hợp 100%:*

> *Tất cả các đối tượng của lớp (kiểu) con đều phải phù hợp 100% về:*
>
> *+ Các thuộc tính,*
>
> *+ Các mối quan hệ*
>
> *với các đối tượng của lớp cha.*

*Ví dụ:*




Hình 6-31 Các lớp kế thừa và luật thành viên, luật 100%

Theo hai luật trên thì mọi đối tượng của các lớp con **ThanhToanTienM, ThanhToanThe** và **ThanhToanCheck** đều có thuộc tính *tongSoTien* như ở lớp cha và chúng đều có quan hệ kết hợp *Trả-cho* với lớp **PhienBanHang.**

Trong thiết kế hướng đối tượng, một vấn đề rất quan trọng là phải thiết lập được các lớp tổng quát và kế thừa để tạo ra một cấu trúc phân cấp giữa các lớp nhiều nhất có thể. Quan hệ kế thừa sẽ hỗ trợ để sử dụng lại những thuộc tính, hàm thành phần của lớp cha trong thiết kế và cài đặt các lớp con.

Thường chúng ta có hai cách thực hiện công việc trên.

1. Phân tách một lớp thành nhiều lớp con.
2. Gộp một số lớp con thành lớp tổng quát hơn.

Câu hỏi đặt ra là với những điều kiện nào thì thực hiện phân tách và những điều kiện nào có thể gộp các lớp lại được?

**Nguyên nhân cần phân chia một lớp thành một số lớp con**

Khi xét một lớp, ngoài những thuộc tính và hành vi chung, các đối tượng còn có:

1. Một số thuộc tính khác nhau cần được bổ sung vào để tạo thành những lớp con cho đối tượng để có thêm những đặc tính khác ngoài những đặc tính chung của lớp cha.
2. Một số quan hệ được bổ sung và khác với lớp đang xét.
3. Cần xử lý, thao tác khác nhau ở những lớp con.

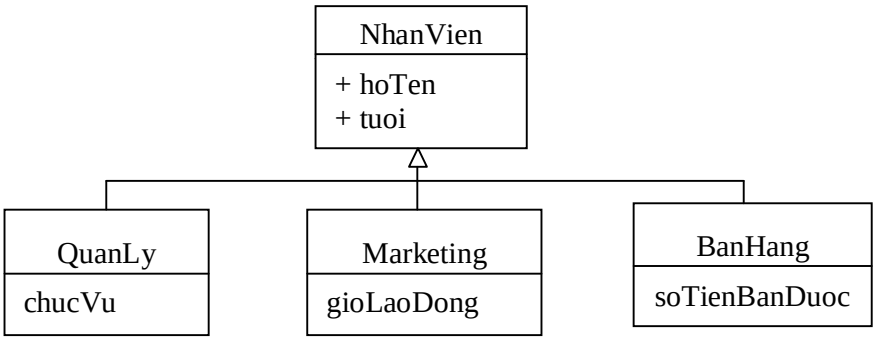
*Ví dụ:* hãy phân tích và thiết kế các lớp phục vụ cho hệ thống tính tiền lương trả cho nhân viên của một cơ quan. Trước tiên hãy xét lớp **NhanVien.** Lớp này có những thuộc tính chung như *họ và tên (hoTen), tuổi (tuoi).* Các nhân viên của cơ quan được tổ chức thành các bộ phận như: cán bộ quản lý, bộ phận *marketing,* bộ phận bán hàng. Cách tính lương được thực hiện như sau:

+ Cán bộ quản lý tính lương theo chức vụ hay nhiệm vụ được giao,

+ Nhân viên *marketing* được tính lương theo giờ làm việc trong tháng,

+ Các nhân viên bán hàng được trả lương theo % hoa hồng số tiền bán được hàng.

Như vậy, các bộ phận trên, ngoài những thuộc tính chung của **NhanVien** còn có thêm những thuộc tính khác nhau, lớp **QuanLy** có thêm thuộc tính *chucVu*, lớp **Marketing** có thêm *gioLaoDong* và lớp **BanHang** có thêm *soTienBanDuoc* để mô tả được đúng các nhân viên trong từng bộ phận. Nghĩa là ta có thể chia **NhanVien** thành **QuanLy**, **Marketing** và **BanHang** như sau:




Hình 6-32 Chia lớp NhanVien thành các lớp con

Trong quá trình thiết kế, chúng ta thường thực hiện hoặc gộp một số lớp có một số điểm chung thành lớp tổng quát, hoặc ngược lại chia nhỏ và bổ sung thêm một số tính chất để tạo ra những lớp con kế thừa lớp đã được xây dựng trước. Cách làm này làm tăng khả năng sử dụng lại và đảm bảo tính mở cao cũng như tính tin cậy của hệ thống.

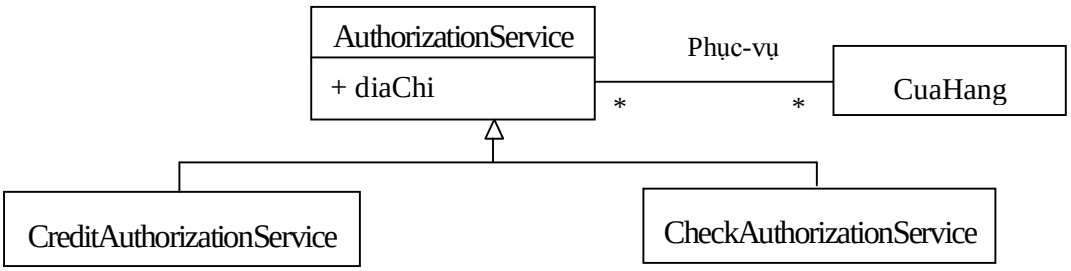
**Nguyên nhân cần gộp lại thành lớp tổng quát**

Khi một số lớp (gọi là lớp con) có những tính chất sau thì có thể gộp chúng lại và tạo ra một lớp tổng quát:

1. Các lớp con cùng thể hiện các phiên bản của một khái niệm tương tự.
2. Các lớp con đều thoả hai luật: luật 100% và luật thành viên.
3. Các lớp con có một số thuộc tính giống nhau để có thể nhóm lại thành lớp tổng quát hơn.
4. Các lớp con đó có một số liên kết chung để có thể đưa chúng vào lớp cha.

*Ví dụ*: các lớp **ThanhToanTienM, ThanhToanThe** và **ThanhToanCheck** có nhiệm vụ giống nhau là cùng thanh toán cho một phiên bán hàng, nên có thể gộp chúng thành lớp **ThanhToan** như hình 6-30, 6-31.

Tương tự, hai lớp **CreditAutorizationService** và **CheckAutorizationService** có hành vi giống nhau là cùng làm nhiệm vụ kiểm duyệt và có cùng quan hệ với **CuaHang**, do vậy có thể gộp chúng thành lớp tổng quát là **AutorizationService**. Biểu đồ mô tả những quan hệ đó như trong hình 6-33.




Hình 6-33 Lớp tổng quát hoá

Đối với những lớp mới, chúng ta lại tìm cách xác định các thuộc tính, các hàm thành phần và bổ sung những quan hệ cần thiết để hoàn thiện thiết kế biểu đồ lớp như hình 6-28, 6-29 cho tất cả các lớp trong hệ thống.

## 6.5 Thiết kế biểu đồ cộng tác và hoàn thiện thiết kế biểu đồ lớp trong Rose

Tương tự như các chương trước, chi tiết về cách sử dụng các chức năng của Rose có thể tham khảo ở ([8], [11]).

### 6.5.1 Xây dựng biểu đồ cộng tác

+ Tạo lập biểu đồ cộng tác,

+ Tạo lập, bổ sung các tác nhân, đối tượng vào biểu đồ,

+ Bổ sung các thông điệp (hàm), đặt tên và các hướng điều khiển trong các quan hệ kết hợp,

+ Gán trách nhiệm cho các đối tượng,

+ Ánh xạ đối tượng vào lớp và thao tác vào thông điệp.

Thực hiện đối với các biểu đồ cộng tác hình 6-12, 6-15, 6-20, 6-23, 6-24.

### 6.5.2 Hoàn thiện thiết kế biểu đồ lớp

Tiếp theo phần thực hành xây dựng biểu đồ lớp ở chương 4, ở đây tập trung thực hiện bổ sung một số tính chất cho lớp:

+ Bổ sung tham số và đặt đối số cho tham số,

+ Bổ sung lớp tiện ích (*Class Utility*), lớp tham số tiện ích (*Parameterized Class Utility*), lớp tiện ích hiện thực (*Instantiated Parameterized Class Utility*) và *Metaclass.*

+ Đặc tả chi tiết các lớp: gán các kiểu, *stereotype,* các thuộc tính phạm vi (*Visibility*), đặt bội số và các thuộc tính lưu trữ (*persistent, tránient*),

+ Bổ sung đầy đủ các thuộc tính, các hàm cho các lớp,

+ Bổ sung những quan hệ: kết hợp, kết tập, phụ thuộc và kế thừa giữa các lớp.

Vẽ các biểu đồ lớp như các hình 6-27, 6-28, 6-30, 6-31.

## Bài tập và câu hỏi

6.1 Hãy cho biết những mệnh đề sau đúng hay sai (*true / false*), giải thích tại sao?

+ *Biểu đồ cộng tác* chính là một đồ thị chỉ ra một số các đối tượng và những sự liên kết giữa chúng.

+ Biểu đồ cộng tác của một hoạt động thể hiện thuật toán để thực thi hành động đó.

+ Ca sử dụng được xác định trong pha phân tích các yêu cầu hỗ trợ để cài đặt và có liện hệ nhiều với giao diện sử dụng.

+ Một lớp được thiết kế tốt là lớp có độ móc nối cao và mức độ cố kết thấp.

+ Từ một lớp bất kỳ luôn tạo ra được một lớp con kế thừa từ lớp đó.

6.2 Xây dựng biểu đồ cộng tác cho hoạt động "Đăng ký môn học" (tiếp theo bài 5.3).

6.3 Xây dựng biểu đồ lớp đầy đủ cho hệ thống "Quản lý thư viện" (tiếp theo bài 5.5).

6.4 Thiết lập biểu đồ lớp cho "Hệ thống rút tiền tự động ATM (Automatic Teller Machine)" (Tiếp theo của bài toán 5.6).

6.5 Chọn từ danh sách dưới đây những thuật ngữ thích hợp để điền vào các chỗ [(…)] trong đoạn văn mô tả về công việc trong phân tích và thiết kế hệ thống.

> Trong pha phân tích chúng ta tập trung khảo sát hệ thống để trả lời cho câu hỏi hệ thống gồm [(1)] và cũng đã bắt đầu nghiên cứu các [(2)], *tương tác giữa* các lớp đối tượng thông qua biểu đồ trình tự và biểu đồ trạng thái. Nhiệm vụ chính của giai đoạn thiết kế là chuyển từ câu hỏi [(1)] sang trả lời cho câu hỏi [(3)]. Như vậy nhiệm vụ chính trong thiết kế hướng đối tượng là xây dựng các [(4)] của các đối tượng để hệ thống thực hiện được các yêu cầu đã được xác định trong pha phân tích.

*Chọn câu trả lời:*

a. "như thế nào?"

b. biểu đồ cộng tác

c. những cài gì

d. mối quan hệ

# CHƯƠNG VII

# KIẾN TRÚC HỆ THỐNG VÀ PHÁT SINH MÃ TRÌNH

Chương VII trình bày:

- ✓ Kiến trúc của hệ thống phần mềm,
- ✓ Biểu đồ thành phần và biểu đồ triển khai,
- ✓ Thiết kế giao diện hệ thống,
- ✓ Ánh xạ một thiết kế sang mã cộng tác trong Rose.

## 7.1 Kiến trúc của Hệ thống

Thiết kế chính là đưa ra cách giải logic bài toán ứng dụng và mô tả cách hệ thống thực thi những nhiệm vụ đã xác định. Nghĩa là:

- Xây dựng các thiết kế chi tiết để mô tả các thành phần của hệ thống ở mức cao hơn, phục vụ cho việc cài đặt ở pha sau.
- Đồng thời đưa ra được kiến trúc (là trọng tâm) của hệ thống để đảm bảo có thể thay đổi được, có tính mở, dễ bảo trì, thân thiện với NSD, v.v.

*Kiến trúc hệ thống* là cấu trúc tổ chức của hệ thống. Kiến trúc gồm nhiều bộ phận có thể ở nhiều mức khác nhau, tương tác với nhau thông qua giao diện, các mối quan hệ kết nối và các ràng buộc để kết hợp chúng thành một thể thống nhất.

*Kiến trúc phần mềm* là một mô tả về các hệ thống con, các thành phần và mối quan hệ giữa chúng. Các hệ thống con và các thành phần được xác định theo nhiều góc nhìn khác nhau để chỉ ra các thuộc tính chức năng và phi chức năng của hệ thống phần mềm.

Kiến trúc hệ thống được chia thành hai loại: *logic và vật lý.*

- *Kiến trúc logic* chỉ ra các lớp và đối tượng, các quan hệ và sự cộng tác để hình thành chức năng của hệ thống. Kiến trúc logic được mô tả bởi các biểu đồ ca sử dụng, biểu đồ lớp và các biểu đồ tương tác. Kiến trúc phổ biến chung hiện nay là kiến trúc ba tầng: tầng giao diện, tầng tác nghiệp và tầng lưu trữ.
- *Kiến trúc vật lý* đề cập đến mô tả chi tiết hệ thống về phương diện phần cứng và phần mềm của hệ thống. Đồng thời nó cũng mô tả cấu trúc vật lý và sự phụ thuộc của các mô đun cộng tác trong cài đặt những khái niệm đã được định nghĩa trong kiến trúc logic. Kiến trúc vật lý của hệ thống liên quan nhiều đến cài đặt, do vậy, nó được mô hình hoá trong các *biểu đồ thành phần* (*Component Diagram*) và *biểu đồ triển khai* (*Deployment Diagram*).

**Kiến trúc ba tầng**

Như ở phần phân tích đã nêu: Kiến trúc chung cho các hệ thống phần mềm hiện là kiến trúc ba tầng.

1. *Tầng trình diễn* (*Presentation*): Biểu diễn và giới thiệu các thành phần của hệ thống thông qua các giao diện đồ hoạ, các Window, các hộp thoại, v.v. Các thực thể trong hệ thống được thể hiện sao cho vừa thân thiện với người sử dụng và phù hợp với bài toán ứng dụng. Ví dụ, cửa sổ *"Bán hàng"* của hệ thống bán hàng được thiết kế ở tầng trình diễn như ở hình 6-9 thể hiện giao diện giữa người bán và hệ thống.

2. *Tầng logic ứng dụng* (*Application Logic*): Mô tả các đối tượng thực thi các nhiệm vụ và các qui luật điều hành các tiến trình của hệ thống. Hệ thống phần mềm có hai loại đối tượng cơ bản:

   *Những đối tượng nghiệp vụ* (*Domain Object*): là những đối tượng đại biểu cho các khái niệm của miền ứng dụng. Ví dụ, trong hệ thống bán hàng, các đối tượng :PhienBanHang, :ThanhToan, v.v. là những đối tượng thực hiện các nhiệm vụ của bài toán ứng dụng.

   *Những đối tượng dịch vụ* (*Service Object*): những đối tượng không nằm trong phạm vi bài toán nhưng cung cấp các dịch vụ cho hệ thống như: làm môi giới, tương tác với CSDL, trao đổi thông tin, lập báo cáo, đảm bảo án toán, an ninh dữ liệu, v.v.

   Tầng hai được thể hiện ở các biểu đồ: lớp, trạng thái, cộng tác, hành động và biểu đồ triển khai.

3. *Tầng lưu trữ* (*Storage*): thể hiện cơ chế lưu trữ đảm bảo nhất quán và bền vững dữ liệu. Hiện nay có hai mô hình CSDL chính đang được sử dụng phổ biến là: *mô hình dữ liệu quan hệ và mô hình dữ liệu hướng đối tượng*. Để lưu trữ dữ liệu cho hệ thống, ta phải lựa chọn hệ quản trị CSDL và những phương pháp biến đổi dữ liệu, biến đổi các câu truy vấn cho phù hợp với công nghệ hiện đại và với phạm vi ứng dụng. Kiến trúc sư đối tượng phải lựa chọn công nghệ CSDL thích hợp để phát triển hệ thống. Khi phân tích, thiết kế hướng đối tượng và nếu lựa chọn mô hình dữ liệu quan hệ thì người phát triển hệ thống phải quan tâm đến những vấn đề về tích hợp dữ liệu để đảm cho phép người sử dụng truy nhập được tới tất cả các dữ liệu từ nhiều mô hình khác nhau một cách trong suốt.

Quá trình phát triển phần mềm cũng giống như quá trình học và nhận thức của con người, đó là quá trình lặp, tích luỹ để phát triển liên tục. Vì vậy kiến trúc của hệ thống cũng phải được xây dựng sao cho phù hợp với sự phát triển và khả năng mở rộng của hệ thống.

Trong triển khai cài đặt, kiến trúc ba tầng thường được tổ chức theo nhiều phương án khác nhau. Có thể thực hiện:

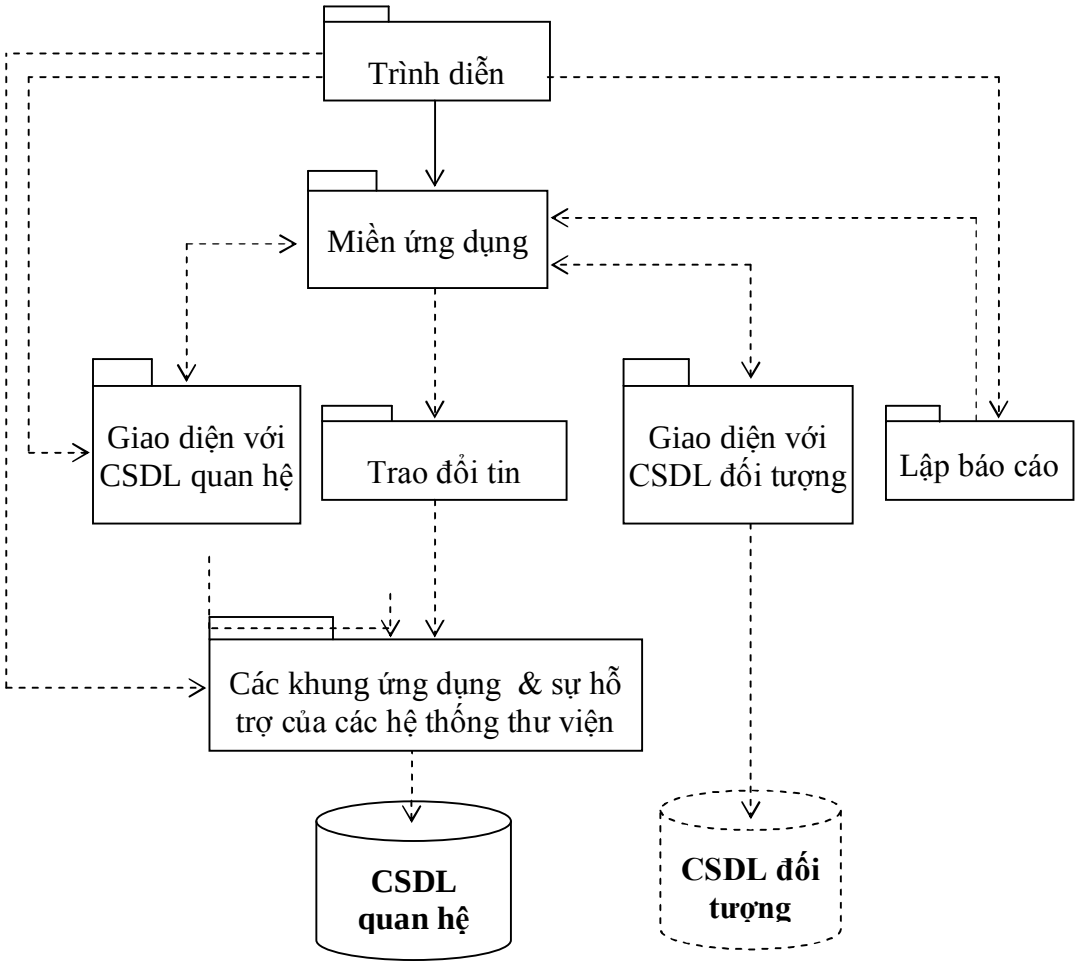
1. Cả 3 tầng trên cùng máy (hệ thống nhỏ).

2. Tầng trình diễn và tầng logic ứng dụng trên máy khách (*Client Computer*), tầng lưu trữ trên Sever (hệ thống loại vừa).

3. Tầng trình diễn trên máy người sử dụng, tầng logic ứng dụng trên máy trạm phục vụ ứng dụng (*Application Server*), còn tầng lưu trữ tổ chức ở trạm phục vụ dữ liệu (Data Server) (những hệ thống lớn).

Hiện nay có những ngôn ngữ lập trình hướng đối tượng như Java và các công nghệ CSDL hướng đối tượng đã sẵn sàng hỗ trợ cài đặt phân tán theo các phương án 2. và 3.

Để thiết kế được hệ thống thì ta phải chia nó thành các hệ thống con gọi là các gói (*packages*). Các *package* trong hệ thống có mức độ phụ thuộc vào nhau. Do vậy, một kiến trúc chi tiết chung cho hệ thống trong đó biểu diễn được cả quan hệ phụ thuộc sẽ được xây dựng tổng quát như sau:




Hình 7-1 Kiến trúc chung của hệ thống

*Một số điểm cần lưu ý trong kiến trúc trên*

1. Các gói giao diện CSDL quan hệ và CSDL đối tượng cung cấp các cơ chế nhất quán để trao đổi với các CSDL khác.

2. Các gói dịch vụ hướng đối tượng ở mức cao như: lập báo cáo (*reporting*), giao diện với CSDL (*Database Interfaces*), đảm bảo an ninh (*security*), trao đổi (*Communication*), v.v. được phát triển theo kỹ nghệ hướng đối tượng và thường được các nhà phát triển phần mềm hệ thống cung cấp .

   Ngược lại, ở dịch vụ mức thấp như các chức năng xử lý tệp, Windows, hội thoại, v.v. thường được cung cấp bởi thư viện chuẩn của ngôn ngữ lập trình hoặc được mua trực tiếp của các hãng sản xuất phần mềm.

3. Các khung ứng dụng (*Application Frameworks and Support Libraries*) hỗ trợ để tạo ra Windows, định nghĩa các phần tử hợp tác ứng dụng, truy cập tới tệp và CSDL, trao đổi giữa các tiến trình, v.v.

4. Quan hệ phụ thuộc của các gói A và gói B chỉ ra rằng gói A biết rõ (do vậy sử dụng được) về gói B. Khi A không có quan hệ phụ thuộc vào B thì từ A không thể quy chiếu (tham khảo) đến các lớp, thành phần, giao diện, hàm, hoặc các dịch vụ của B được.

*Tổ chức các Packages và mối quan hệ giữa chúng*

1. Hướng dẫn để nhóm các phần tử thành một *package*

- Nhóm những phần tử có các dịch vụ chung, có quan hệ hợp tác, cố kết tương hỗ cao thành một gói.
- Ở cùng mức trừu tượng thì mỗi gói được xem như một bộ phận, gồm một số phần tử có cấu kết và liên hệ chặt chẽ với nhau.
- Ngược lại, các phần tử ở hai gói khác nhau sẽ liên kết với nhau ở mức thấp.

2. Khả năng quan sát giữa các lớp của các gói

- Gói trình diễn (*Presentation Package*) có thể nhìn thấy được nhiều lớp trong miền ứng dụng.
- Gói miền ứng dụng (*Domain Package*) có thể nhìn được một số lớp đối tượng trong mỗi gói dịch vụ cụ thể.

## 7.2 Biểu đồ thành phần

*Biểu đồ thành phần* (*Component Diagram*) là biểu đồ mô tả các thành phần và sự phụ thuộc của chúng trong hệ thống. Các thành phần của hệ thống có thể là:

- *Thành phần mã nguồn* (*Source Code*), có ý nghĩa vào thời điểm dịch chương trình. Thông thường nó là tập các chương trình cài đặt các lớp. Ví dụ, trong C++, mỗi tệp *.cpp* và *.h* là một thành phần. Trước khi phát sinh mã chương trình, phải thực hiện ánh xạ từng tệp vào thành phần tương ứng, thông thường mỗi lớp được ánh xạ vào hai tệp (*.cpp*, và *.h*).
- *Thành phần mã nhị phân* là mã trình nhị phân được dịch từ mã chương trình nguồn. Nó có thể là tệp mã đích (*.obj*), tệp thư viện tĩnh (*.lib*) hay tệp thư

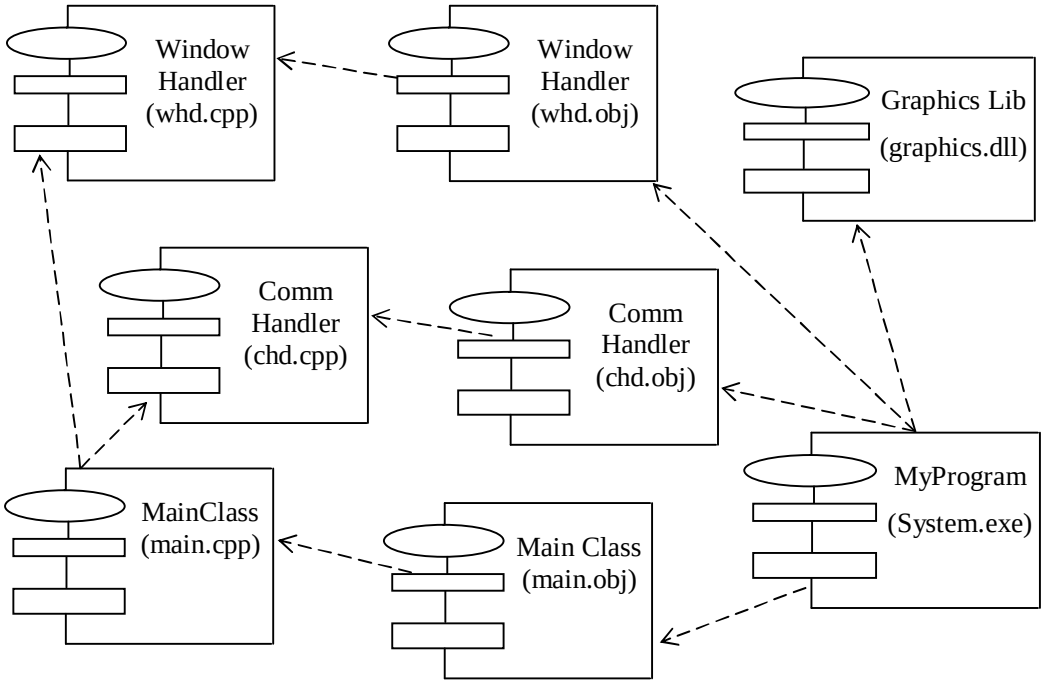
viện động (*.dll*). Thành phần nhị phân được sử dụng để liên kết, hoặc để thực thi chương trình (đối với thư viện động).

- *Thành phần thực thi* là tệp chương trình có thể thực thi được (các tệp *.exe*). Nó là kết quả của chương trình liên kết các thành phần nhị phân.

Với biểu đồ thành phần, người phát triển thực hiện dịch, triển khai hệ thống sẽ biết thư viện mã trình nào tồn tại và những tệp có thể thực thi (*.exe*) khi dịch và liên kết thành công.

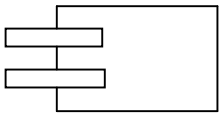
Giữa các thành phần chỉ có một loại quan hệ phụ thuộc được biểu diễn bằng đường mũi tên đứt nét. *Lưu ý,* nên tránh phụ thuộc vòng trong biểu đồ thành phần.

Ví dụ: biểu đồ thành phần mô tả sự phụ thuộc giữa các thành phần của hệ thống.




Hình 7-2 Sự phụ thuộc của các thành phần trong biểu đồ thành phần

*Lưu ý*: Trong Rose và một số ngôn ngữ khác, ngoài biểu tượng trên, thành phần còn được ký hiệu là:



Trong C++, hay sử dụng các biểu tượng đặc tả gói (*Package Specification*) cho tệp *.h,* biểu tượng nội dung gói (*Package Body*) cho tệp *.cpp.*

Tương tự như các phần tử khác trong UML, các thành phần có thể bổ sung một số đặc tả chi tiết:

+ *Stereotype:* điều khiển biểu tượng nào sẽ được sử dụng để biểu diễn thành phần. Nó có thể là một trong các lựa chọn: *<none>, đặc tả chương trình con, chương trình chính, đặc tả gói, nội dung của gói, đặc tả nhiệm vụ, nội dung công việc, ActiveX, Applet, ứng dụng, v.v.*

+ *Ngôn ngữ:* Rose cho phép lựa chọn ngôn ngữ lập trình cho từng thành phần, như C++, Java, Visual Basic, v.v.

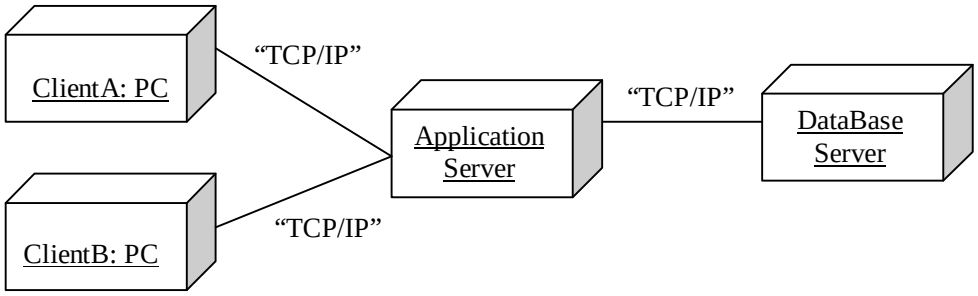
+ *Khai báo:* phụ thuộc được gộp vào mã chương trình cho mỗi thành phần. Lệnh #include của C++ được xem như là lệnh khai báo.

+ *Lớp:* trước khi phát sinh mã chương trình thì lớp phải được ánh xạ vào thành phần. Điều này báo cho Rose biết mã chương trình của lớp sẽ được ghi vào tệp nào. Có thể ánh xạ một hay nhiều lớp vào một thành phần.

Biểu đồ thành phần được xem như là tập các biểu tượng thành phần biểu diễn cho các thành phần vật lý trong một hệ thống. Ý tưởng cơ bản của biểu đồ thành phần là tạo ra cho những người thiết kế và phát triển hệ thống một bức tranh chung về các thành phần của hệ thống.

## 7.3 Biểu đồ triển khai

Biểu đồ triển khai (*Deployment Diagram*) chỉ ra cấu hình các phần tử xử lý lúc chương trình chạy, các nút trên mạng và các tiến trình phần mềm thực hiện trên những phần tử đó. Nó chỉ ra mối quan hệ giữa các phần cứng và phần mềm của hệ thống. Ví dụ, biểu đồ triển khai của hệ thống có thể tổ chức như hình 7-3.




Hình 7-3 Biểu đồ triển khai của hệ thống

Mỗi nút là một đối tượng vật lý (các thiết bị) có tài nguyên tính toán. Chúng có thể là máy tính, máy in, máy đọc ảnh, thiết bị truyền tin, v.v. Các nút được kết nối với nhau thông qua các giao thức (*protocol*) như các giao thức "TCP/IP" ở hình 7-3.

**Các phần tử (nút) của biểu đồ triển khai**

- *Bộ xử lý (processor):* bộ xử lý của máy tính, máy chủ, trạm làm việc, v.v. Các bộ xử lý được đặc tả chi tiết bằng cách bổ sung thêm các thông tin:

+ *stereotype:* để phân nhóm các bộ xử lý.

+ *Đặc tính:* mô tả các tính chất vật lý của mỗi bộ xử lý như: tốc độ tính toán, dung lượng bộ nhớ, v.v.

+ *Lịch biểu (Schelduling):* mô tả loại lịch biểu thời gian xử lý, bao gồm:

  - *Preemptive* cho phép những tiến trình có mức ưu tiên cao hơn có thể chiếm quyền xử lý đối với những tiến trình có mức ưu tiên thấp hơn
  - *Non Preemptive* không có ưu tiên, một tiến trình chỉ dừng khi nó tự kết thúc
  - *Cyclic* chỉ ra chu kỳ điều khiển giữa các tiến trình
  - *Executive:* các lịch biểu được điều khiển bằng thuật toán, bằng chương trình
  - *Manual:* tiến trình được điều khiển bằng người sử dụng.

- *Thiết bị* là máy móc hay bộ phận phần cứng nhưng không phải là bộ xử lý trung tâm, như: màn hình, máy in, máy vẽ, v.v. Thiết bị cũng có thể đặc tả một số thông tin chi tiết như: *stereotype và một số tính chất vật lý.*
- *Tiến trình (Process)* là một luồng thực hiện của một chương trình trong một bộ xử lý. Một chương trình thực thi được xem như là một tiến trình.

## 7.4 Ánh xạ các thiết kế sang mã chương trình

Ở đây chúng ta không đề cập nhiều đến pha lập trình hướng đối tượng mà chỉ giới thiệu một số cách ánh xạ những kết quả thiết kế sang mã chương trình.

Hiện nay nhiều công cụ phát triển hiện đại cung cấp những môi trường tiện lợi để nhanh chóng chuyển đổi giữa thiết kế và mã hoá chương trình.

Cài đặt trong một ngôn ngữ lập trình hướng đối tượng đòi hỏi phải viết mã nguồn cho:

- Các phần định nghĩa lớp,
- Các định nghĩa hàm thành phần.

Trong các phần sau chúng ta thảo luận về phần sinh mã trong C++ cho những thiết kế nêu trên.

### 7.4.1 Tạo lập các định nghĩa lớp từ những thiết kế biểu đồ lớp

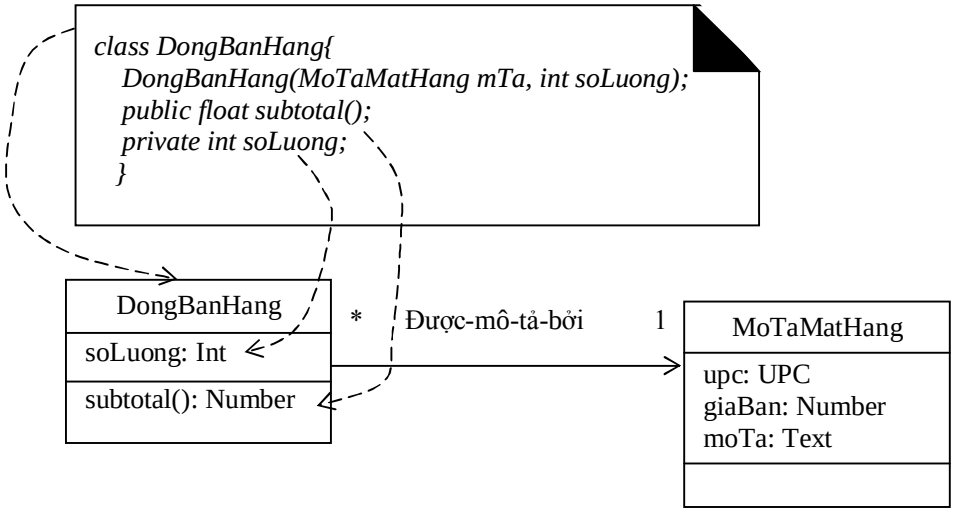
Trong pha thiết kế, biểu đồ lớp được xây dựng chi tiết để mô tả đầy đủ tên gọi của lớp, tên các thuộc tính, các hàm thành phần và mối quan hệ của các đối tượng trong hệ thống. Những kết quả đó đủ để tạo ra các định nghĩa lớp trong ngôn ngữ lập trình hướng đối tượng như C++.

**Định nghĩa lớp với các hàm và thuộc tính đơn**

Từ các lớp đã được thiết kế chi tiết, thực hiện ánh xạ như sau:

- Các thuộc tính đơn được chuyển tương ứng sang các biến có kiểu dữ liệu phù hợp,
- Các hàm được chuyển sang các hàm *prototype.*

Ví dụ, định nghĩa lớp **DongBanHang** trong C++ như sau:




Hình 7-4 Định nghĩa lớp dựa vào biểu đồ lớp

*Lưu ý:* khi định nghĩa, chúng ta thường phải bổ sung thêm hàm tạo lập (*constructor*) để tạo ra những đối tượng khi cần sử dụng trong chương trình. Điều này cũng được suy ra từ biểu đồ cộng tác hình 6.20, trong đó :DongBanHang nhận được thông điệp *create(mt, n )* để tạo ra dòng bán hàng với mô tả là *mt* và số lượng là *n.* Điều này được toán tử tạo lập trong C++ là hàm có cùng tên với tên của lớp hỗ trợ.

Chúng ta cũng nhận thấy, dữ liệu kết quả của hàm *subtotal()* là đã bị thay đổi, từ kiểu *Number* trong thiết kế được chuyển thành *float.* Nghĩa là người lập trình có thể chọn những kiểu dữ liệu thích hợp cho các biến thuộc tính và các hàm, chứ không nhất thiết phải theo đúng bản thiết kế.

**Bổ sung thêm các thuộc tính tham chiều**

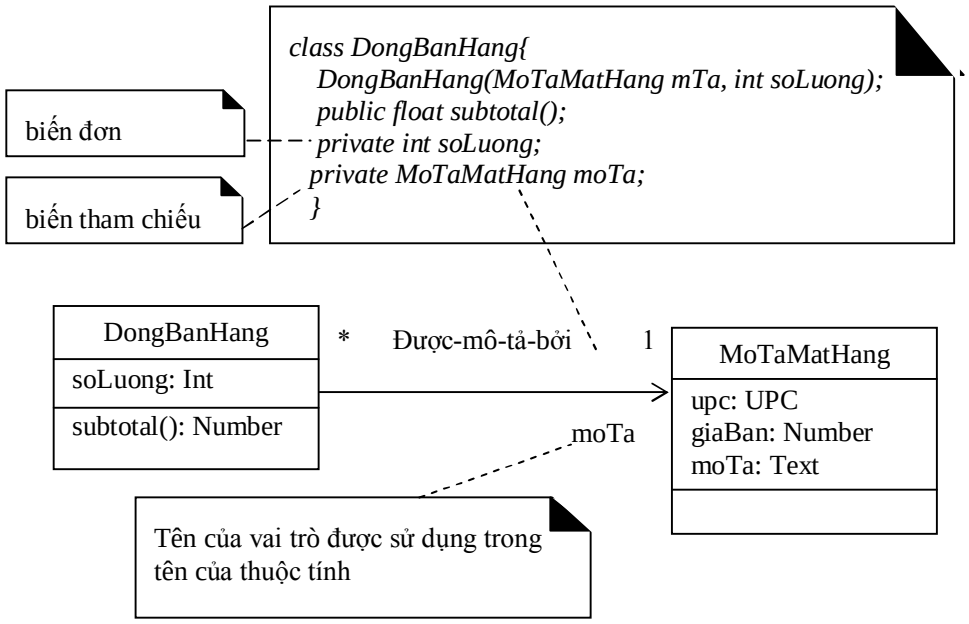
Thuộc tính tham chiếu là thuộc tính được sử dụng để tham chiếu đến đối tượng phức hợp khác, không phải là những kiểu dữ liệu nguyên thuỷ.

*Thuộc tính tham chiếu của một lớp được đề xuất bởi những quan hệ kết hợp và sự điều khiển trong thiết kế biểu đồ lớp*.

*Ví dụ*: trong biểu đồ lớp của hệ thống bán hàng, lớp **DongBanHang** có quan hệ kết hợp với **MoTaMatHang** và mũi tên điều khiển chỉ hướng gửi thông điệp khi trao đổi thông tin như hình 7-4. Trong C++, điều này có nghĩa là trong lớp **DongBanHang** phải khai báo biến tham chiếu tới **MoTaMatHang**.

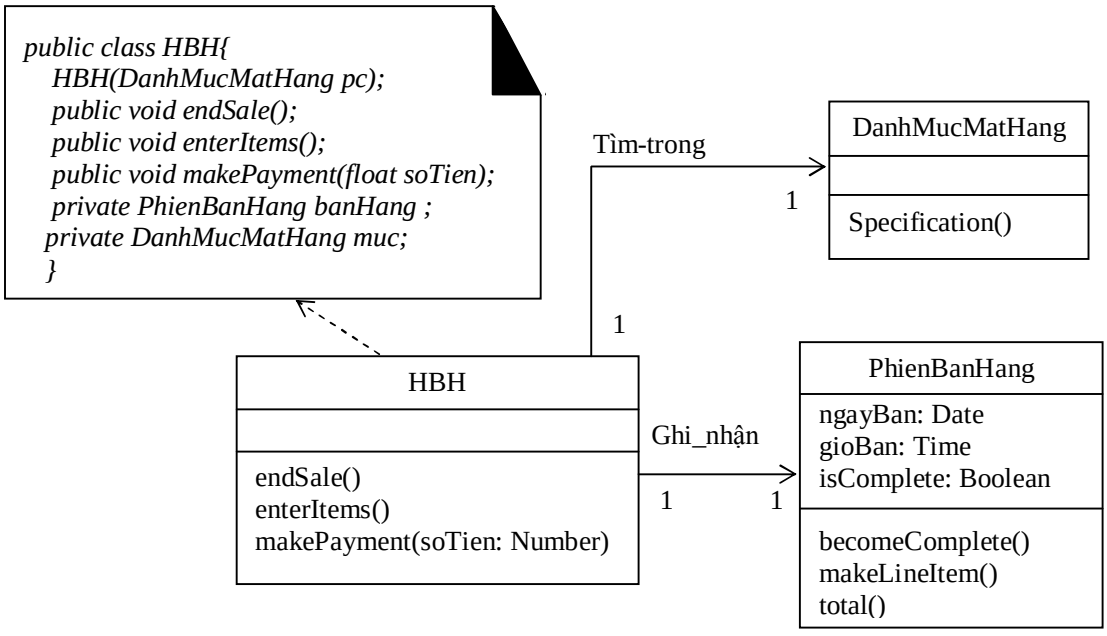
Thuộc tính tham chiếu thường là không tường minh, chúng được suy ra từ các mối quan hệ trong biểu đồ lớp. Đôi khi, nếu *vai trò* của đầu quan hệ có mặt trong biểu

đồ lớp thì chúng ta cũng có thể sử dụng như là tên của thuộc tính tham chiếu. Định nghĩa của lớp **DongBanHang**, do vậy được định nghĩa hoàn chỉnh như hình 7-5.




Hình 7-5 Bổ sung thêm thuộc tính tham chiếu vào lớp

Tương tự như trên, chúng ta có thể định nghĩa các lớp khác, ví dụ lớp **HBH** được định nghĩa như hình 7-6.

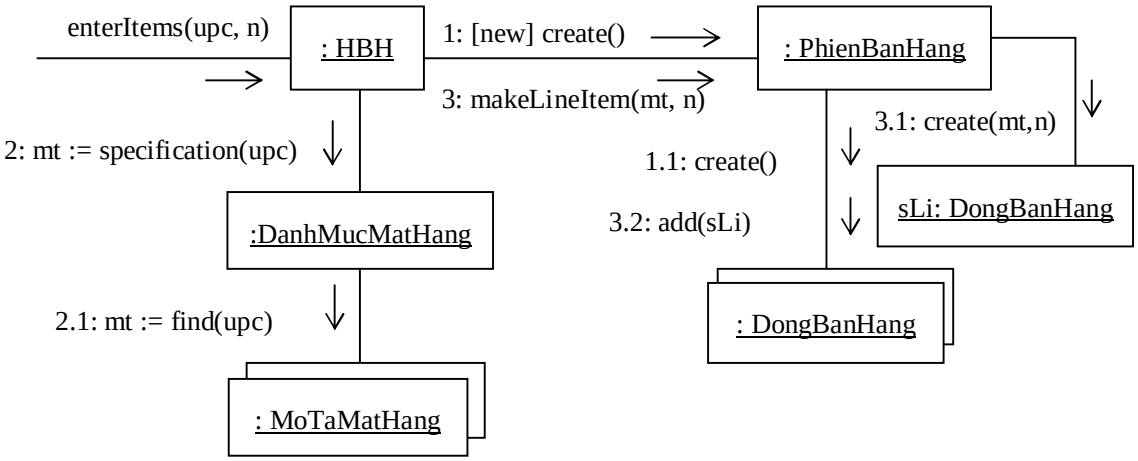



Hình 7-6 Định nghĩa lớp HBH

### 7.4.2 Định nghĩa hàm từ biểu đồ cộng tác

Biểu đồ cộng tác chỉ ra cách các thông điệp được gửi đến cho các đối tượng bằng các lời gọi hàm. Dãy các thông điệp nhận được sẽ được dịch tường ứng sang dãy các lệnh trong định nghĩa hàm thành phần của một lớp.

Chúng ta hãy xét biểu đồ cộng tác mô tả nhiệm vụ *enterItems* (nhập dữ liệu vào) và dựa vào đó để định nghĩa hàm *enterItems()*.




Hình 7-7 Biểu đồ cộng tác cho *enterItems*

Thông điệp *enterItems* được gửi cho một đối tượng của HBH, do vậy, trong lớp HBH sẽ có hàm *enterItems()* được định nghĩa như sau:

```
public void enterItems(UPC upc, int soLuong);
```

**Thông điệp 1:** Theo biểu đồ trên, để thực hiện được nhiệm vụ *enterItems* thì trước tiên nó phải tạo ra :PhienBanHang mới, nếu đó là mặt hàng đầu tiên được nhập vào. Nghĩa là

```
if (isNewSale()){banHang = new PhienBanHang();}
```

Hàm đầu tiên chúng ta cần khảo sát là: *isNewSale()*. Hàm này có thể khai báo *private*, làm nhiệm vụ kiểm tra xem biến đối tượng của **PhienBanHang** có rỗng hay không, nghĩa là đối tượng PhienBanHang chưa được thiết lập hoặc là đã kết thúc phiên bán hàng trước đó. Vậy

```
private int isNewSale(){return (banHang == null)
                              || banHang.isComplete();}
```

*Lưu ý:* C++ không có kiểu *Boolean*, nhưng nó xem giá trị của biểu thức khác 0 là đúng (*true*) và bằng 0 là sai (*false*). *Hàm này được bổ sung vào lớp* **HBH** ngoài những hàm được xác định như ở hình 7.6.

**Thông điệp 2:** Tiếp theo, một thông điệp được gửi cho :DanhMucMatHang để xác định các thông tin về mặt hàng có mã là *upc*.

```
DanhMucMatHang moTa = muc.specification(upc);
```

*muc* là một đối tượng của **DanhMucMatHang**.

**Thông điệp 3:** Thông điệp thứ ba *makeLineItem* được gửi cho :PhienBanHang để xác lập một dòng bán hàng thông qua đối tượng *banHang*.

```
banHang.makeLineItem(moTa, soLuong);
```

*Tóm lại, mỗi thông điệp được gửi đi để thực hiện yêu cầu của thông điệp đầu tiên trong biểu đồ cộng tác sẽ được ánh xạ thành dãy các câu lệnh tương ứng trong định nghĩa hàm thành phần ứng với thông điệp đó.*

Theo nguyên tắc đó, hàm *enterItems()* được định nghĩa đầy đủ như sau:

```
public void enterItems(UPC upc, int soLuong){
 if (isNewSale()){banHang = new PhienBanHang();}
 DanhMucMatHang moTa = muc.specification(upc);
 banHang.makeLineItem(moTa, soLuong);
}
```

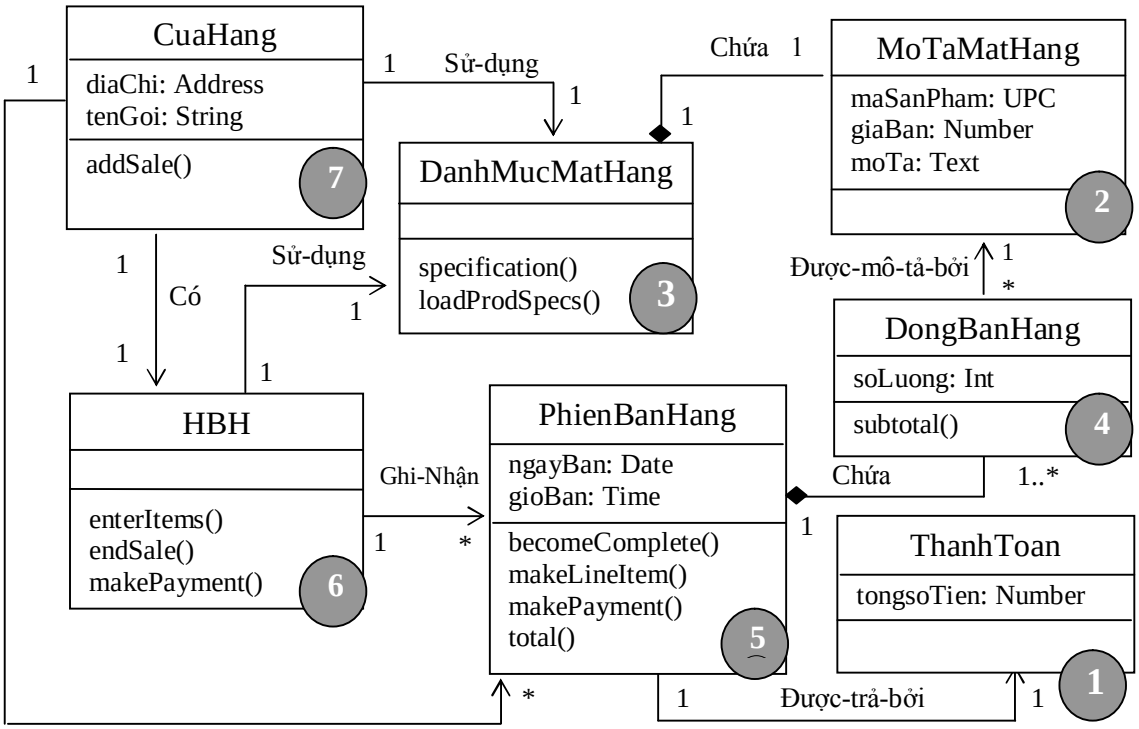
*Định nghĩa hàm makeLineItem()*

Xét tiếp biểu đồ cộng tác cho *enterItems(upc, soLuong)* ở hình 6-20, lớp **PhienBanHang** nhận được thông điệp *makeLineItem(moTa, soLuong).* Để thực hiện được yêu cầu này, nó lại phải gửi đi hai thông điệp: *3.1: create(moTa, soLuong)* cho sl: DongBanHang và *3.2: adds(sl)* cho tập các đối tượng :DongBanHang. Do đó, hàm *makeLineItem()* trong lớp **PhienBanHang** sẽ được định nghĩa như sau:

```
public void makeLineItem(MoTaMatHang moTa, int soLuong){
 lineItem.addElement(new DongBanHang(moTa, soLuong);
}
```

**Thứ tự cài đặt các lớp**

Các lớp cần được cài đặt từ những lớp cặp bộ kém dần đến những lớp cặp bộ cao hơn. Ví dụ, những lớp đầu tiên có thể chọn để cài đặt trong hệ HBH là: hoặc **ThanhToan**, hoặc **MoTaMatHang**, sau đó là **DanhMucMatHang, PhienBanHang**, rồi đến **HBH** và **CuaHang,** v.v. như hình 7-8.




Hình 7-8 Thứ tự cài đặt các lớp

## 7.5 Danh sách một số lớp được định nghĩa trong C++

**1. Lớp ThanhToan**

```
class ThanhToan{
 private:
   float tongSoTien;
 public:
   ThanhToan(float soTien){
        tongSoTien = soTien;
        }
   float getAmount(){return tongSoTien;}
 };
```

**2. Lớp MoTaMathang**

```
#include <string.h>
public class MoTaMatHang{
 private:
   int upc = 0;
   float giaBan = 0.0;
   String moTa = "";
 public:
   MoTaMatHang(int ma, float p, String mt){
        upc = ma; giaBan = p; strcpy(moTa, mt);
```

```
        }
    MoTaMatHang(int ma, float p, String mt){
        upc = 0; giaBan = 0; strcpy(moTa, "");
        }
    int getUPC(){return upc;}
    float getPrice(){return giaBan;}
    String getDescrip(){return moTa;}

  };
```

**3. Lớp DanhMucMatHang**

```
#define N 100
public class DanhMucMatHang{
 private:
   MoTaMatHang danhSachMH[N];
 public:
   DanhMucMatHang(){
        for(inh k = 0; k < N; k++)
          danhSachMH[k] = new MoTaMatHang();
        }
   MoTaMatHang find(int upc){
        for(k = 0; k < N; k++)
           if (danhSachMH[k].getUPC() == upc) return danhSachMH[k];
        }
 };
```

**4. Lớp DongBanHang**

```
class DongBanHang{
 private:
   int soLuong;
   MoTaMatHang moTa;
 public:
   DongBanHang(MoTaMatHang mt, int n){
        moTa = mt;
        soLuong = n;
        }
   float subtotal(){
        return soLuong * moTa.getPrice();
        }
 };
```

**5. Lớp PhienBanHang**

```
#define M 20
class PhienBanHang{
 private:
   DongBanHang danhSachBan[M];
   Date ngayBan;  // Kiểu ngày/tháng/năm
```

```
  Time gioBan;          // Kiểu giờ/phút/giây
  int soDong = 0;
  int isComplete = 0;   // false
  ThanhToan traTien;
public:
  getBalance(){
       return traTien.getAmount() - total();
       }
  void becomeComplete(){isComplete = 1;}
  int isComplete() {return isComplete;}
  void makeLineItem(MoTaMatHang mt, int sl){
       lineItem[soDong++]= new DongBanHang(mt, sl);
       }
  float total(){
       float t = 0.;
       for (int k = 0; k < soDong; k++)
         t += lineItem[k].subtotal();
       return t;
       }
  void makePayment(float soTien){
       traTien = new ThanhToan(soTien);
       }
};
```

**6. Lớp HBH**

```
class HBH{
 private:
  DanhMucMatHang catalog;
  PhienBanHang banHang;
 public:
  HBH(DanhMucMatHang danhMuc){ catalog = danhMuc;}
  void endSale(){ banHang.becomeComplete();}
  void enterItems(int upc, int soLuong){
       if (isNewSale(){
            banHang = new PhienBanHang();
       }
       MoTaMatHang mt = catalog.find(upc);
      banHang.makeLineItem(mt, soLuong);
  }
  void makePayment(soTien){
       banHang.makePayment(soTien);
  }
  int isNewSale(){
       return (banHang == null) || (banHang.isComplete());
  }
};
```

**7. Lớp CuaHang**

```
class CuaHang{
 private:
   DanhMucMatHang dm = new DanhMucMatHang();
   HBH post = new HBH(dm);
 public:
    HBH getPOST() {return post;}
 };
```

Tương tự, dễ dàng định nghĩa cho tất cả các lớp khác của hệ thống HBH.

## 7.6 Thực hành trên Rose

### 7.6.1 Xây dựng biểu đồ thành phần

+ Tạo lập mới hoặc mở một biểu đồ thành phần đã được tạo lập trước,

+ Bổ sung, loại bỏ các thành phần,

+ Đặc tả chi tiết các thành phần: gán *stereotype,* chọn ngôn ngữ *Language,* chọn lớp để gán *Assign*,

+ Thiết lập các mối quan hệ phụ thuộc giữa các thành phần.

### 7.6.2 Xây dựng biểu đồ triển khai

+ Tạo lập mới hoặc mở một biểu đồ triển khai đã được tạo lập trước,

+ Bổ sung, loại bỏ các nút, các phần tử xử lý *Processor,*

+ Đặc tả chi tiết các phần tử xử lý: gán *stereotype,* bổ sung một số đặc tính như *Preemtive, Non preemtive, Cyclic, v.v.* và gán *Schelduling,*

+ Bổ sung các thiết bị *Device* và đặc tả chúng,

+ Thiết lập các mối quan hệ kết nối giữa các nút,

+ Bổ sung và huỷ bỏ các tiến trình *Process.*

### 7.6.3 Phát sinh mã trình bằng Rose

Có sáu bước cơ bản để phát sinh mã chương trình:

1. Kiểm tra mô hình để phát hiện những sai phạm, những điểm không thống nhất trong mô hình. Sau khi chọn *Tools* > *Check Model*, lỗi của mô hình sẽ được hiển thị ở cửa sổ *Log.* Mục *Access Violation* sẽ tìm ra những vi phạm khi có những quan hệ giữa hai lớp ở hai gói khác nhau, nhưng hai gói đó lại không có quan hệ với nhau. Chọn *Report* > *Show Access Violation* để biết được những vi phạm đó.
2. Tạo lập hay mở biểu đồ thành phần.
3. Gán lớp vào thành phần:
   + Kích hoạt thành phần trong biểu đồ thành phần bằng cách nhấn chuột phải,
   + Chọn *Open Specification* > *Realizes*
   + Nhấn chuột phải trên lớp cần gán và chọn tiếp *Assign.*

4. Đặt thuộc tính cho quá trình sinh mã: chọn *Tools* > *Options* để hiển thị bảng đặc tính và có thể chọn: ngôn ngữ, *Class, Attribute, Operation,* hay chọn các đặc tính tạm thời *Tools* > *Model Properties* > *Edit* > *C++ tab* >*Edit Set,* v.v.
5. Chọn lớp, thành phần hay gói để phát sinh mã chương trình.
6. Phát sinh mã chương trình: chọn *Tools* > *Add-Ins* > *Add-In Manager* để hiển thị hộp thoại để lựa chọn ngôn ngữ lập trình.

**Cái gì được phát sinh?**

Khi phát sinh mã chương trình, *Rose* tập hợp các thông tin từ *Logical View* và *Component View. Rose* chỉ phát sinh khung chương trình.

*Class:* mọi lớp trong mô hình đều có thể được phát sinh mã chương trình,

*Attribute:* thuộc tính được xác định miền tác động, kiểu dữ liệu và giá trị mặc định,

*Operation:* các thao tác được khai báo kèm theo danh sách các tham số, kểu của tham số, kiểu dữ liệu trả lại,

*Relationships:* một số mối quan hệ sẽ được phát sinh mã tương ứng,

*Component:* mỗi thành phần sẽ được cài đặt trong một tệp mã nguồn tương ứng.

*Lưu ý:* Rose không hướng vào thiết kế giao diện đồ hoạ. Do vậy, sau khi phát sinh khung chương trình như trên, nhiệm vụ tiếp theo của người phát triển là tập hợp những tệp nguồn đã được tạo lập, lập trình cho những thao tác của lớp chưa được sinh mã và thiết kế giao diện đồ hoạ cho hệ thống ứng dụng.

## Bài tập và câu hỏi

7.1 Hãy cho biết những mệnh đề sau đúng hay sai (*true / false*), giải thích tại sao?

+ *Kiến trúc vật lý* thể hiện các lớp đối tượng, các quan hệ và sự cộng tác để hình thành chức năng của hệ thống.
+ Kiến trúc phổ biến chung hiện nay cho các hệ thống phần mềm là kiến trúc ba tầng: tầng giao diện, tầng tác nghiệp và tầng lưu trữ.
+ Biểu đồ thành phần mô tả các thành phần và sự phụ thuộc của chúng trong hệ thống.
+ Có thể chọn mô hình dữ liệu quan hệ để lưu trữ dữ liệu cho hệ thống được phân tích, thiết kế hướng đối tượng.
+ Tất cả các tên gọi, biến và kiểu dữ liệu của các lớp trong biểu đồ lớp được thiết kế phải được chuyển tương ứng sang mã chương trình trong các định nghĩa lớp ở ngôn ngữ lập trình đã được lựa chọn.

7.2 Xây dựng biểu đồ thành phần cho hệ thống "Đăng ký môn học".

7.3 Xây dựng biểu đồ thành phần cho hệ thống "Quản lý thư viện" (tiếp theo bài 6.3).

7.4 Thực hiện sinh mã tự động trong Rose cho các lớp ở hình 7.8.

7.5 Chọn từ danh sách dưới đây những thuật ngữ thích hợp để điền vào các chỗ [(…)] trong đoạn văn mô tả về kiến trúc của hệ thống phần mềm.

*Kiến trúc phần mềm* là [(1)] về các hệ thống con, [(2)] và [(3)] giữa chúng. Các hệ thống con và [(2)] được xác định theo nhiều góc nhìn khác nhau để chỉ ra các [(4)] và của hệ thống phần mềm. Kiến trúc hệ thống được chia thành hai loại: *logic và vật lý.*

*Chọn câu trả lời:*

a. thuộc tính chức năng

b. mối quan hệ

c. các thành phần

d. một mô tả

e. phi chức năng

7.6 Chọn từ danh sách dưới đây những thuật ngữ thích hợp để điền vào các chỗ [(…)] trong đoạn văn mô tả về biểu đồ thành phần.

Biểu đồ thành phần được xem như là tập các biểu tượng thành phần biểu diễn cho các [(1)] vật lý trong một [(2)]. Ý tưởng cơ bản của biểu đồ [(1)] là tạo ra cho những [(3)] và phát triển một bức tranh chung về các thành phần của [(2)].

*Chọn câu trả lời:*

a. hệ thống

b. người thiết kế

c. thành phần vật lý

d. một mô tả

# CHƯƠNG VIII
# CƠ SỞ DỮ LIỆU HƯỚNG ĐỐI TƯỢNG

Chương cuối thảo luận về:

- ✓ Cơ sở dữ liệu quan hệ và hướng đối tượng,
- ✓ Giới thiệu hệ quản trị CSDL HĐT, hệ ObjectStore,
- ✓ Thiết kế CSDL HĐT,
- ✓ Cài đặt mô hình đối tượng trong ObjectStore với C++.

Cơ sở dữ liệu là nơi lưu giữ lâu dài các dữ liệu của hệ thống ở bộ nhớ ngoài. Các dữ liệu này phải được tổ chức tốt theo hai tiêu chí: *hợp lý, nghĩa là đầy đủ, không dư thừa* và *truy cập thuận lợi, nghĩa là tạo lập, loại bỏ, tìm kiếm, cập nhật nhanh chóng và tiện dùng.*

## 8.1 Cơ sở dữ liệu quan hệ và hướng đối tượng

*Một cơ sở dữ liệu (CSDL) là một kho chứa dữ liệu được lưu giữ trong một hoặc nhiều tệp.* CSDL chứa cả cấu trúc dữ liệu (*lược đồ* mô tả về dữ liệu) và cả chính dữ liệu. Phần lớn các hệ CSDL hoạt động như là một *thế giới đóng*, nghĩa là một sự kiện không tìm thấy trong CSDL được xem như là sai.

Một *hệ quản trị CSDL* (*DataBase Management System - DBMS*), viết tắt là **QTCSDL**, là phần mềm để có thể tạo lập, quản lý truy cập và thao tác trên CSDL. Hệ QTCSDL hứa hẹn cung cấp cho ta những chức năng chung cho một phạm vi rộng những ứng dụng khác nhau. Một trong các mục đích chính của công nghệ HĐT là khuyến khích tái sử dụng phần mềm, nhất là đối với những ứng dụng, trong đó dữ liệu sử dụng với cường độ cao thì những hệ QTCSDL có thể thay thế cho nhiều mã chương trình ứng dụng. Hệ QTCSDL có một hay nhiều *ngôn ngữ truy vấn* (*query language*). Ngôn ngữ truy vấn cung cấp hệ lệnh để đọc và ghi dữ liệu trực tiếp hoặc thông qua một chương trình ứng dụng.

Hiện nay có hai mô hình CSDL chính thường được sử dụng để phát triển các hệ thống thông tin: mô hình quan hệ và mô hình hướng đối tượng.

### 8.1.1 Cơ sở dữ liệu quan hệ

*Một CSDL quan hệ,* viết tắt là CSDLQH, là một CSDL trong đó dữ liệu được quan sát dưới dạng các bảng (*table*). *Một hệ QTCSDL quan hệ* quản lý các bảng dữ liệu và các cấu trúc kết hợp nhằm làm tăng khả năng thực thi của các bảng. Các d*ạng chuẩn* (*Normal Forms*) là những nguyên tắc chỉ đạo cho những người thiết kế CSDLQH nhằm làm tăng tính nhất quán của dữ liệu. Khi các bảng thoả mãn những dạng chuẩn bậc cao thì dữ liệu được lưu trữ trong chúng sẽ đảm bảo chính xác hơn.

Mô hình quan hệ có những đặc điểm chính:

1. *Mô hình đơn giản (Simple model):* CSDL là tập các đối tượng trong thế giới thực được lưu trữ dưới dạng bảng.
2. *Độc lập với dữ liệu (Data independency):*

   + Mô tả mô hình độc lập với phương pháp cài đặt vật lý của dữ liệu

   + Dữ liệu không phụ thuộc vào cấu trúc logic của hệ thống.
3. *Dữ liệu được định nghĩa và truy vấn là phi thủ tục* (*Non-procedural inquiry language*).

Như chúng ta đã biết, các CSDL QH đã và đang được sử dụng rất hiệu quả trong nhiều bài toán ứng dụng với những chi phí khá lớn và nhiều người đã có nhiều kinh nghiệm trong thiết kế và cài đặt các ứng dụng theo mô hình này. Tuy nhiên nó cũng có nhiều hạn chế.

1. Không mô hình được những hệ thống lớn phức tạp, nhất là những hệ thống đa phượng tiện (multimedia).

   + Các kiểu và cấu trúc dữ liệu yếu và bị hạn chế,

   + Rất khó thể hiện được các mối quan hệ giữa các thành phần dữ liệu.
2. Môi trường phát triển và quản lý chương trình là yếu:

   + Không tính toán, điều phối được mối quan hệ giữa ngôn ngữ lập trình và ngôn ngữ CSDL.

   + Không có những khả năng mạnh để tạo lập và không có các chức năng che giấu để đảm bảo an toàn, an ninh dữ liệu trong các hệ thống thông tin.

### 8.1.2 Cơ sở dữ liệu hướng đối tượng ([5], [14])

*Một CSDL hướng đối tượng,* viết tắt là CSDL HĐT, được xem như là một kho bền vững các đối tượng được tạo ra bởi một ngôn ngữ lập trình, hay một hệ quản trị CSDL HĐT. Với một ngôn ngữ lập trình bất kỳ, các đối tượng sẽ ngừng tồn tại (kết thúc) khi chương trình ứng dụng kết thúc, nhưng trong CSDL HĐT, các đối tượng duy trì ở bên ngoài phạm vi thực hiện của chương trình. Hệ *QTCSDL HĐT* quản lý dữ liệu, quản lý mã chương trình và các cấu trúc kết hợp nhằm thiết lập một CSDL HĐT.

Khác với các hệ QT CSDL quan hệ, các hệ QTCSDL HĐT khác nhau rất nhiều về cú pháp và các khả năng ứng dụng. Tổ chức chuẩn hoá việc quản lý dữ liệu đối tượng ODMG (*Object Data Management Group*) cố gắng tìm cách giải quyết sự khác biệt đó bằng cách thống nhất đưa ra những kỹ thuật mô hình hoá đối tượng OMT [7] hay ngôn ngữ mô hình hoá thống nhất UML ([1],[2]).

Câu hỏi đặt ra là tại sao lại cần CSDL ĐT? Để trả lời cho câu hỏi trên, ta hãy tìm hiểu thêm một số điểm đặc trưng của các CSDL ĐT khác so với CSDLQH:

- *Hỗ trợ những kiểu dữ liệu được định nghĩa bởi NSD.* CSDL ĐT có khả năng lưu trữ các kiểu phức hợp, kiểu được định nghĩa bởi NSD như các lớp và thao tác trên chúng một cách dễ dàng. Trong khi CSDL quan hệ lưu trữ các giá trị số và ký tự như dữ liệu, còn trong CSDL HĐT là đối tượng bao gồm

thêm cả dữ liệu ảnh (images), tín hiệu (signal), hay hình ảnh động. Trong một đối tượng, dữ liệu và hàm (thao tác) được bao gói thành một cấu trúc được gọi là lớp.

- *Cung cấp một mẫu hình phát triển CSDL cho cả phân tích, thiết kế và cài đặt ứng dụng*. Bạn không phải chuyển từ mẫu hình này sang mẫu hình khác như trong quá trình phát triển phần mềm.
- *Cải tiến đáng kể về chất lượng dữ liệu*. Ta có thể đưa ra nhiều ràng buộc vào cấu trúc dữ liệu. Mô hình còn cho phép thể hiện cả những ràng buộc phi cấu trúc để chương trình phải thoả mãn khi nó thực hiện trong CSDL. Vấn đề khá quan trọng đặt ra là có thể chuyển một CSDLQH có thể chuyển về một CSDL HĐT [(13)].
- *Tốc độ phát triển phần mềm nhanh hơn*. Cấu trúc CSDL nhất quán và rõ ràng, giúp cho lập trình ứng dụng trở nên đơn giản và nhanh hơn. Những người phát triển ứng dụng có kinh nghiệm thường sử dụng những câu lệnh rất mạnh của các hệ QT CSDL thay cho những đoạn chương trình của NSD.
- *Tích hợp dễ dàng*. Việc tích hợp nhiều hệ thống độc lập với nhau vào một hệ thống ứng dụng có thể giảm bớt sự sao chép dữ liệu của con người và mở rộng phạm vi những câu truy vấn có thể trả lời được. Mô hình hướng đối tượng cung cấp cách biểu diễn thống nhất làm thuận tiện hơn cho việc tìm hiểu và tích hợp thông tin.

## 8.2 Các hệ quản trị CSDL hướng đối tượng

Hệ *QTCSDL HĐT* quản lý dữ liệu, mã chương trình và các cấu trúc kết hợp nhằm thiết lập một CSDL HĐT. Nhiều hệ QTCSDL HĐT được xây dựng có cú pháp, và những khả năng rất khác nhau. Một số hệ QTCSDL HĐT phổ biến trên thị trường như: ObjectStore (www.odi.com), GemStore (www.gemstore.com), Objectivity (www.Objectivity.com), $O_2$ (www.ardensoftawre.com), Jasmine (www.cai.com), Versant (www.versant.com) và POET (www.poet.com).

### 8.2.1 Các đặc trưng của hệ quản trị CSDL HĐT

Ta có thể dễ dàng cài đặt các mô hình đối tượng OMT hay UML bằng những hệ QTCSDL HĐT. Những hệ QTCSDL HĐT cho phép biểu diễn trực tiếp cho các lớp, thuộc tính, phương thức và quan hệ kế thừa từ các mô hình đối tượng, từ các biểu đồ lớp, một số hệ cũng đã hỗ trợ cài đặt những quan hệ nhị nguyên. Trong các chương sau chúng ta sẽ thảo luận những vấn đề về cài đặt các mô hình UML bằng hệ QTCSDL HĐT.

Hệ QTCSDL HĐT được phát triển là do sự kém cỏi của các hệ QTCSDL QH trong việc trả lời những câu truy vấn liên quan đến đối tượng vào đối tượng. Ngoài ra, kiểu dữ liệu của các CSDL QH còn quá đơn giản đối với nhiều ứng dụng hiện thời và SQL còn khá lúng túng trong việc kết hợp với các ngôn ngữ lập trình. Ngược lại, hệ QTCSDL HĐT hỗ trợ rất nhiều kiểu dữ liệu phong phú và thông thường là kết hợp chặt chẽ với ít nhất một ngôn ngữ lập trình. Các hệ QTCSDL HĐT thực hiện nhanh

hơn những ứng dụng điều khiển từ các đối tượng vào các đối tượng, hỗ trợ kế thừa, quản lý định danh đối tượng và nhiều tính chất khác.

Tuy nhiên, các hệ QTCSDL HĐT cũng có những yếu điểm [5] sau:

- *Thiếu cơ sở lý thuyết và chuẩn hoá.* Các hệ CSDL QH được thiết kế và xây dựng dựa trên mô hình đại số quan hệ rất chuẩn mực, trong khi các hệ QT CSDL HĐT được cài đặt nhưng thiếu cơ sở lý thuyết hình thức. Hậu quả là các sản phẩm QTCSDL HĐT rất khác nhau và gây ra nhiều trở ngại cho quá trình phát triển ứng dụng khi không muốn phụ thuộc vào các hãng sản xuất phần mềm hệ thống.
- *Có thể sửa đổi làm sai lệch CSDL.* Một số hệ QTCSDL HĐT thực hiện trong không gian của tiến trình ứng dụng, kết quả là CSDL có thể là chủ điểm dẫn đến vi phạm tính an ninh hoặc dữ liệu bị sửa đổi bởi những con trỏ, tham chiếu lạ.
- *Không có khả năng mở rộng logic.* Các sản phẩm hiện nay đều không có sự độc lập dữ liệu cần thiết và chưa có các quan sát (*view*) CSDL tương tự như các hệ QTCSDL QH.
- *Chưa hỗ trợ các siêu (meta) ứng dụng*. Một số hệ QTCSDL HĐT dựa vào C++ không thực hiện được liên kết động, liên kết lúc thực hiện, mà chỉ cung cấp liên kết tĩnh, liên kết lúc dịch chương trình ứng dụng. Hạn chế này là do hạn chế của ngôn ngữ. C++ sử dụng những khai báo về kiểu để sinh mã chương trình tối ưu trong quá trình biên dịch và sau đó huỷ bỏ các khai báo đó. Ngược lại, hầu hết các CSDL QH và những hệ QTCSDL HĐT dựa vào Smallalk hỗ trợ cho cả liên kết tĩnh và liên kết động.

Nói chung, các hệ QTCSDL HĐT có những ưu điểm như của CSDL HĐT đã nêu trên, các CSDL HĐT là giải pháp thích hợp để khắc phục những yếu điểm của CSDLQH. Các hệ QTCSDL HĐT là thích hợp hơn đối với những ứng dụng mới, như:

- *Những ứng dụng thiết kế công nghệ.* Các hệ QTCSDL HĐT rất phù hợp cho những chương trình thiết kế ứng dụng, như thiết kế với sự trợ giúp máy tính (*CAD*: *Computer-Aided Design*), chế tạo với sự trợ giúp của máy tính (*CAM: Computer-Aided Manufacturing*), chế tạo tích hợp với máy tính (*CIM: Computer-Integrated Manufacturing*), và kỹ nghệ phần mềm với sự trợ giúp của máy tính (CASE: *Computer-Aided Software Engineering* ).
- *Các ứng dụng đa phương tiện (Multimedia).* Các hệ QTCSDL HĐT rất thích hợp cho những ứng dụng đa phương tiện với những đồ hoạ, audio, video phức hợp.
- *Các cơ sở tri thức.* Các luật của các hệ chuyên gia rất khó lưu trữ trong các hệ CSDL QH. Mỗi khi có một luật mới cần bổ sung thì phải kiểm tra toàn bộ cơ sở luật xem tính phi mâu thuẫn và dư thừa của hệ thống có bị vi phạm hay không. Hệ QTCSDL HĐT có thể hỗ trợ để thực hiện công việc trên khá hiệu quả.

- *Những ứng dụng đòi hỏi xử lý phân tán và tương tranh*. Hệ QTCSDL cho phép thực hiện những truy nhập cần thiết vào các dịch vụ mức thấp.
- *Các phần mềm nhúng*. Hệ QTCSDL HĐT thích hợp để tạo ra những phần mềm nhúng vào các thiết bị điện, các thiết bị điều khiển, v.v.

### 8.2.2 Giới thiệu về hệ ObjectStore

Khác với các CSDLQH, trọng tậm của các chương trình ứng dụng với ObjectStore là thao tác trên các đối tượng. ObjectStore có những đặc trưng chính như sau:

*1. Hỗ trợ mô hình đối tượng*

ObjectStore hỗ trợ tất cả các khái niệm của mô hình đối tượng:

*Lớp:* Đối tượng và lớp là những khái niệm trung tâm của ObjectStore. Đối tượng hiện hữu trong hệ thống dựa vào định danh.

*Các thuộc tính:* ObjectStore cho phép ánh xạ trực tiếp các thuộc tính vào những cấu trúc xác định của ngôn ngữ lập trình. Đối với những thuộc tính dẫn xuất, ObjectStore không hỗ trợ trực tiếp mà bạn phải viết chương trình để xác định những thuộc tính đó.

*Các phương thức:* ObjectStore cung cấp những khả năng dịch khá hiệu quả các phương thức (hàm) sang mã chương trình thực hiện được.

*Các quan hệ kết hợp:* ObjectStore hỗ trợ những quan hệ kết hợp nhị nguyên, không hỗ trợ những khái niệm nâng cao như các thuộc tính liên kết (*link attributes*), lớp kết hợp (*association classes*).

*Kế thừa:* ObjectStore hỗ trợ cả kế thừa đơn và kế thừa bội.

*2. Đối tượng bền vững và tạm thời*

Đối tượng ở trong các ứng dụng của ObjectStore có thể là bền vững (*persistent*) hoặc tạm thời (*transient*). *Đối tượng bền vững* là đối tượng được lưu trong CSDL và có thể trải qua nhiều ứng dụng khác nhau. *Đối tượng tạm thời* là đối tượng chỉ tồn tại trong bộ nhớ và bị biến mất khi ứng dụng kết thúc. Như thế, đối tượng tạm thời là đối tượng lập trình thông thường. Một lớp có thể có cả đối tượng bền vững lẫn những đối tượng tạm thời.

CSDL của ObjectStore liên kết các đối tượng bằng các con trỏ bền vững và các tham chiếu (*reference*). Tham chiếu của ObjectStore không giống như tham chiếu trong C++. Nó là lớp được tham số hoá (*parameterized class*) của ObjectStore, có thể sử dụng như là con trỏ, nhưng cho phép viết đè để hạn chế sự tham chiếu trong các CSDL và trong các giao dịch.

Bạn có thể truy cập các đối tượng tạm thời khi muốn sử dụng những đối tượng lập trình khác hay sử dụng các truy vấn của ObjectStore. Có thể truy cập đối tượng bền vững thông qua gốc của CSDL (*database roots*) hoặc các câu truy vấn.

Bạn có thể tạo ra những đối tượng bền vững bằng các cơ chế của C++ (trong bộ nhớ tĩnh, ở Stack hay trên Heap).

Nói chung, tất cả các tính chất của các kiểu đều có thể áp dụng cho cả đối tượng bền vững và tạm thời, kể cả quan hệ kế thừa. Tuy nhiên, ta cũng phải tuân theo một số qui tắc khi sử dụng các con trỏ và tham chiếu tới các đối tượng trong một hay nhiều CSDL.

| Loại con trỏ | Con trỏ hợp lý khi |
|---|---|
| Con trỏ tạm thời trỏ tới đối tượng tạm thời | Chỉ khi chương trình thực hiện |
| Con trỏ tạm thời trỏ tới đối tượng bền vững | Hợp lệ cho một giao tác hoặc trong khoảng thời gian chạy của một ứng dụng. |
| Con trỏ bền vững trỏ tới đối tượng bền vững trong cùng một CSDL | Luôn luôn hợp lý |
| Con trỏ bền vững trỏ tới đối tượng bền vững trong các CSDL khác nhau | Hợp lý cho một giao tác hoặc nhiều giao tác phụ thuộc vào sự lựa chọn trong ứng dụng |
| Con trỏ bền vững trỏ tới đối tượng tạm thời | Hợp lý cho một giao tác. Phải gán giá trị null cho con trỏ hoặc chuyển nó về đối tượng bền vững trước khi giao tác được uỷ thác. |

*3. Các lớp tuyển tập (Collection)*

Thư viện lớp là thành phần quan trọng của môi trường phát triển phần mềm hướng đối tượng, trong đó các lớp *Collection là* quan trọng nhất.

Một *Collection* là một đối tượng kết nhập từ những đối tượng khác. Những đối tượng này có thể là đối tượng nguyên thuỷ hoặc lại là tuyển tập nhỏ hơn. ObjectStore hỗ trợ cả những tuyển tập thuần nhất (các phần tử có kiểu cùng loại) và những tuyển tập không thuần nhất (các phần tử có nhiều kiểu khác nhau).

Các *Collection* được sử dụng cho nhiều mục đích khác nhau. Ta có thể sử dụng *Collection* để thể hiện, cài đặt các quan hệ kết hợp. ObjectStore cho phép truy vấn trên các *Collection*. Có thể sử dụng *Collection* để cài đặt những thành phần điều khiển các đối tượng. ObjectStore hỗ trợ những lớp *Collection* sau:

**Set**: tuyển tập không được sắp thứ tự và không lặp lại các đối tượng. Chỉ chèn được một đối tượng vào **Set** khi trong đó không có đối tượng đó. Có thể truy cập các đối tượng lặp lại theo thứ tự bất kỳ.

**Bag**: tuyển tập không được sắp thứ tự nhưng được phép lặp lại các đối tượng. Có thể chèn thêm một đối tượng vào **Bag.** Lấy ra từ **Bag** một đối tượng là giảm

đi một lần xuất hiện của đối tượng đó. Có thể truy cập đến các đối tượng lặp lại theo thứ tự xác định.

**List**: tuyển tập được sắp thứ tự và được phép lặp lại các đối tượng. Chèn thêm, lấy ra một đối tượng từ **List** được thực hiện ở những vị trí xác định của **List.** Có thể truy cập đến các đối tượng lặp lại theo thứ tự xác định.

**Array**: tuyển tập được chỉ số hoá, được sắp thứ tự và được phép lặp lại các đối tượng. Khi một đối tượng mảng được tạo lập, tất cả các phần tử của nó là Null. Những phép toán cơ bản như đọc / ghi đều được thực hiện ở những vị trí được chỉ số hoá của mảng. Kích thước của mảng là không thay đổi. Các đối tượng của mảng được truy cập lặp lại theo chỉ số.

**Dictionary**: tuyển tập không được sắp thứ tự và được phép lặp lại các đối tượng. Từ điển (*Dictionary*) khác với những lớp trước đó (**Bag**) ở chỗ nó cho phép liên kết khoá với mỗi phần tử. Khoá có thể là một giá trị của con trỏ và kiểu nào đó trong C++. Chèn thêm, lấy ra một đối tượng từ **Dictionary** được thực hiện thông qua khoá (*key*) được gán với các đối tượng**.** Có thể truy cập đến các đối tượng lặp lại theo thứ tự bất kỳ.

ObjectStore hỗ trợ các phép toán trên tập hợp đối với các lớp tuyển tập: hợp, giao, hiệu và tuyển tập con.

*4. Một số đặc tính khác của ObjectStore*

ObjectStore có cả những tính năng như các CSDL truyền thống và CSDL phát triển.

+ *Khả năng thực hiện đồng thời* (*concurrency*). ObjectStore có thể đọc/ghi từng trang trên đĩa (*disc page*). Cơ chế khoá trang (*page locking*) là rất hiệu quả của ObjectStore. Tuy nhiên, đối với một số ứng dụng, khoá trang cũng hạn chế khả năng thực hiện đồng thời hơn so với khoá đối tượng.

+ *Khôi phục những phần bị phá vỡ* (*Crash Decovery*). ObjectStore có khả năng phục hồi dữ liệu và khôi phục những phần việc đã bị đổ vỡ của phần mềm, máy tính và của đĩa.

+ *Khả năng đảm bảo an ninh* (*security*). Cũng giống như nhiều hệ QTCSDL HĐT khác, ObjectStore có thể đảm bảo an ninh cho hệ thống tệp trong CSDL.

+ *Cung cấp từ điển dữ liệu* (*Data Dictionary*). ObjectStore có sẵn từ điển dữ liệu, song việc truy cập vào nó là khó đối với người lập trình.

+ *Xử lý các giao dịch* (*Transaction*). Mỗi giao tác thực hiện có thể liên quan đến nhiều đối tượng bền vững và tạm thời của nhiều CSDL. Đối với những đối tượng bền vững, hoặc là toàn bộ chương trình ứng dụng phải được viết vào những CSDL đó hoặc không viết gì cả. ObjectStore yêu cầu tất cả các giao dịch trên các đối tượng bền vững phải xuất hiện trong cùng một giao dịch. Tất nhiên nó hỗ trợ cả những giao dịch lồng nhau. *Giao dịch lồng nhau* là những giao dịch được đặt trong một giao dịch khác.

+ *Chỉ số hoá* (*Index*). Chỉ số hoá của ObjectStore có thể làm tăng tốc độ trả lời cho các câu truy cập. Ta có thể tạo ra một hoặc nhiều chỉ số cho một tuyển tập. ObjectStore duy trì cơ chế chỉ số hoá một cách trong suốt và sử dụng nó để cải tiến hiệu suất trong truy vấn đối tượng. Ta cũng có thể tạo lập hoặc huỷ bỏ chỉ số bất kỳ lúc nào trong thời gian thực hiện của chương trình ứng dụng.

  Khi định nghĩa một chỉ số, ta có thể sử dụng *cơ chế đường đi của ObjectStore. Đường đi của ObjectStore* là một dãy các đỉnh, trong đó mỗi đỉnh là giá trị của thuộc tính của đối tượng tương ứng được chỉ số hoá. Ví dụ: đối tượng *:KhachHang* có thể có con trỏ chỉ tới đối tượng :DiaChi với các thuộc tính *phố, quận/huyện, tỉnh/thành.* Khi đó ta có thể đánh chỉ số trên tuyển tập các đối tượng **KhachHang** sử dụng đường đi theo *tỉnh/thành.*

  *Lưu ý:* Đường đi trong ObjectStore khác với đường đi trong OMT ở chỗ mỗi đỉnh trên đường đi chỉ có thể nhiều nhất là một đối tượng.

+ *Truy vấn đối tượng.* ObjectStore cung cấp các câu truy vấn (*queries*) để tìm kiếm các đối tượng. *Truy vấn* tìm trên tuyển tập các đối tượng thoả mãn tân từ được xác định dựa trên các thuộc tính của những đối tượng mong đợi.

+ *Hướng dẫn tìm kiếm* (*Navigation*). Ta có thể tìm kiếm các đối tượng của ObjectStore theo hướng dẫn của dãy các con trỏ như (obj1 -> obj2 -> obj3).

+ *Sự tiến hoá của sơ đồ* (*Schema evolution*). *Sự tiến hoá sơ đồ* muốn đề cập đến việc di trú dữ liệu ngang qua những thay đổi của cấu trúc dữ liệu. Người phát triển phải đồng thời giữ lại những dữ liệu quá khứ trong khi phải mở rộng cấu trúc theo những yêu cầu mới. Việc di trú dữ liệu sang sơ đồ mới được thực hiện qua hai pha.

  *1.* Thay đổi các con trỏ và sự tham chiếu tới dữ liệu cần phải di chuyển và đánh dấu các chỉ số.

  *2.* Thực hiện chương trình mà bạn viết để đảm bảo duy trì hệ thống.

  ObjectStore hỗ trợ cho sự tiến hoá sơ đồ bằng cách duy trì riêng biệt siêu dữ liệu (*metadata*) cho cả sơ đồ CSDL và sơ đồ ứng dụng.

+ *Hỗ trợ phân tán*. Các đối tượng bền vững được lưu trữ trong các CSDL, mỗi đối tượng ở một CSDL. ObjectStore không cho phép lặp lại. Nếu bạn muốn có sự lưu trữ lặp thì phải viết chương trình ứng dụng riêng.

  ObjectStore được xây dựng tối ưu trên kiến trúc *Client-Server.* Mỗi CSDL được quản lý bởi một *Server* và *mỗi Server* có thể quản lý nhiều CSDL. Một ứng dụng có thể có nhiều *Servers, Clients (máy khách),* và nhiều CSDL.

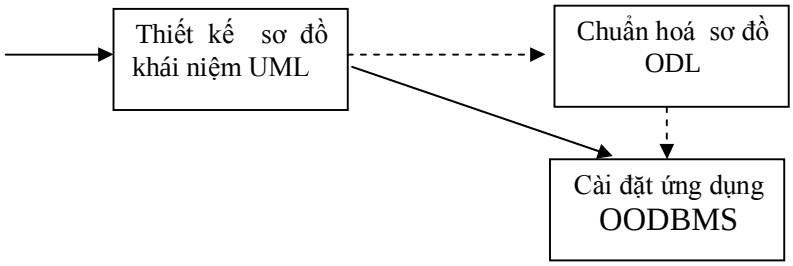
+ *Khách hàng là trung tâm* (*Client-centric Computing*). ObjectStore chấp nhận kiến trúc lấy khách hàng là trung tâm và hướng tới kiến trúc *Client-Server.* Trong các hệ QTCSDL QH thì chủ yếu dựa vào kiến trúc lấy *Server* làm trọng tâm. Những hệ QTCSDL sử dụng kiến trúc lấy *Client* là trung tâm sẽ hỗ trợ khách hàng nhiều hơn.

## 8.3 Thiết kế CSDL HĐT

Trong phần này chúng ta nghiên cứu phương pháp thiết kế sơ đồ CSDL HĐT. Đây có thể được xem như là một hướng dẫn để thiết kế sơ đồ CSDL ứng dụng.

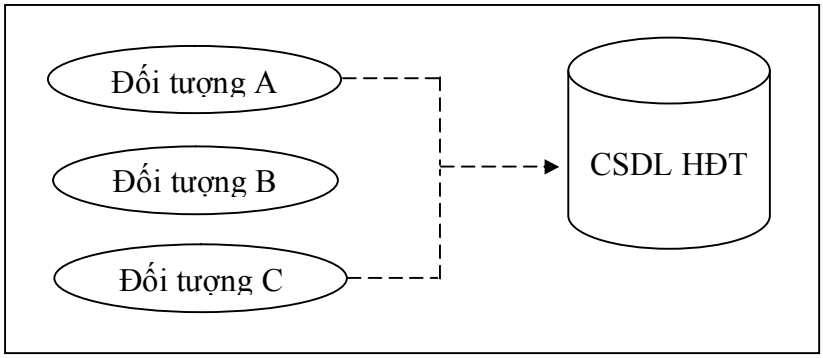
Quá trình thiết kế CSDL có thể thực hiện theo ba bước:

1. Thiết kế mô hình (sơ đồ) khái niệm,
2. Thiết kế sơ đồ chuẩn,
3. Cài đặt CSDL ứng dụng.




Hình 8-1 Quá trình thiết kế sơ đồ CSDL HĐT

Một đặc tính quan trọng của CSDL HĐT là các đối tượng được lưu lại sau khi chương trình kết thúc. Những đối tượng được tự động lưu vào CSDL được gọi là *đối tượng bền vững*, những đối tượng không được lưu lại gọi là *đối tượng tạm thời.*




Hình 8-2 Lưu trữ các đối tượng bền vững vào CSDL

Ở hình trên, đối tượng A, C là bền vững còn B là tạm thời.

Việc đọc dữ liệu trong CSDL HĐT được thực hiện như sau. Các đối tượng trong CSDL HĐT được lưu trữ và có mối quan hệ với nhau thông qua định danh **ID**. Một đối tượng có thể tham chiếu tới nhiều **ID** đối tượng, nghĩa là mối quan hệ giữa các đối tượng có dạng tổng quát là n:m. Mối quan hệ tham chiếu giữa các đối tượng được hệ QT CSDL HĐT thiết lập.

Sau đây chúng ta sẽ xét quá trình thiết kế CSDL cho hệ thống bán hàng.

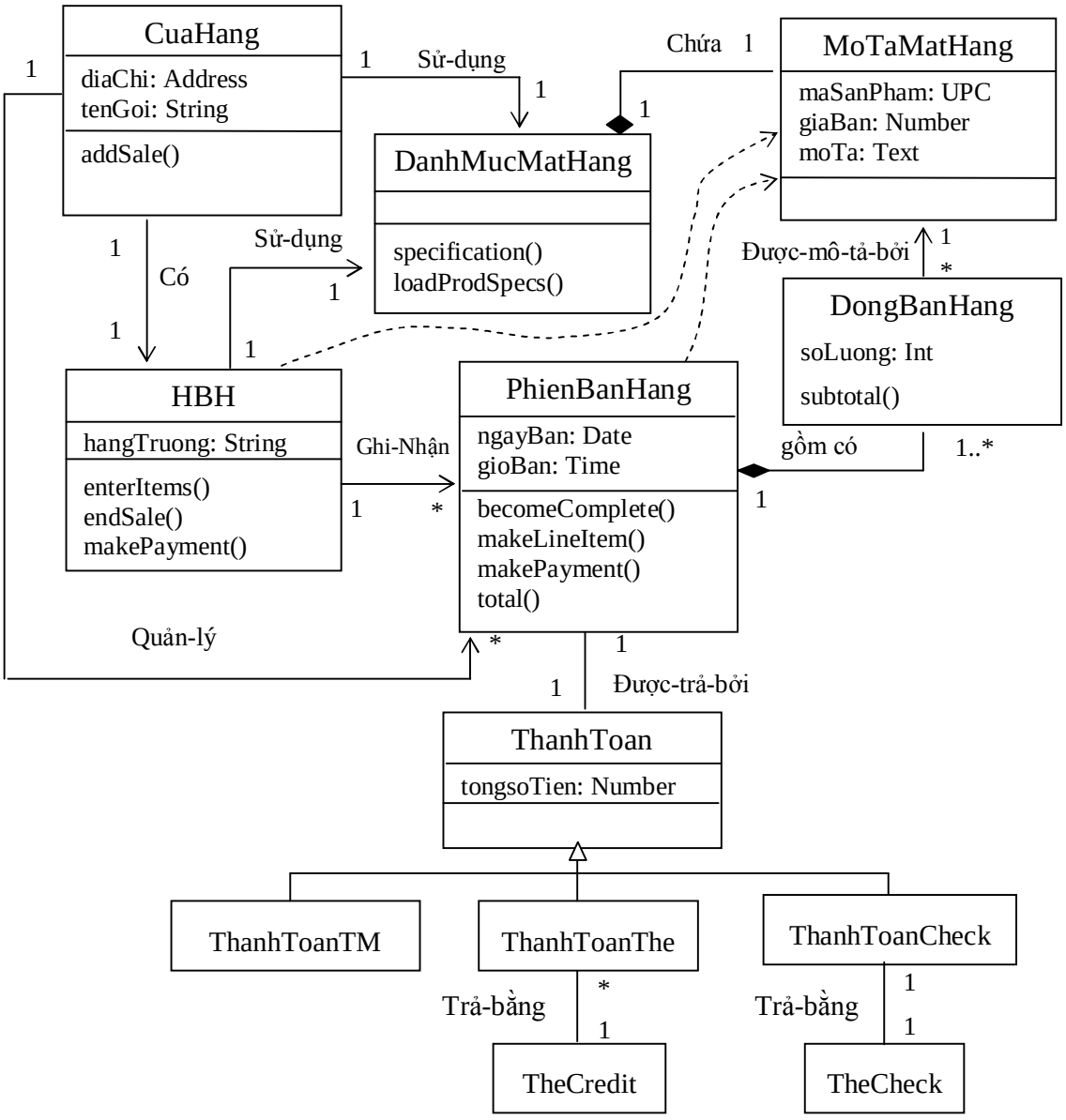
### 8.3.1 Thiết kế sơ đồ khái niệm

Mô hình khái niệm chính là biểu đồ lớp cho hệ thống HBH được thiết kế như ở chương VII (hình 8-3).

### 8.3.2 Thiết kế sơ đồ chuẩn – sơ đồ CSDL

Bước này thực hiện việc chuyển đổi từ mô hình khái niệm (biểu đồ lớp trong UML) sang sơ đồ quản lý dữ liệu đối tượng.

Như chúng ta đã khẳng định, sơ đồ CSDL chỉ lưu trữ những đối tượng bền vững.

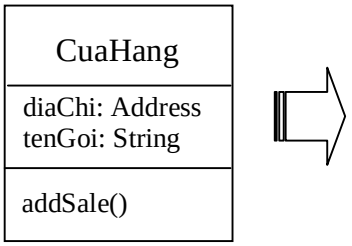



Hình 8-3 Mô hình khái niệm - biểu đồ lớp của hệ thống bán hàng

**Chuyển đổi các kiểu đối tượng**

Mỗi lớp bền vững trong UML phải được chuyển đổi sang lớp trong ODL (*Object Definition Language*) thể hiện được cả hành vi trừu tượng lẫn các trạng thái trừu tượng, như hình 8-3.

Chúng ta xét một lớp điển hình, lớp **CuaHang** để chuyển các thuộc tính của lớp được thiết kế tương ứng sang thuộc tính của lớp trong ODL.




```
Class CuaHang
{ attribute String(30) tenGoi;
  attribute struct diaChi{
     char(3) soPho, char(20) tenPho,
       char(15) tinhThanh};
  void Nguoi();    // Constructor
  void ~Nguoi();   // Destructor
  void addSale(){…}
};
```

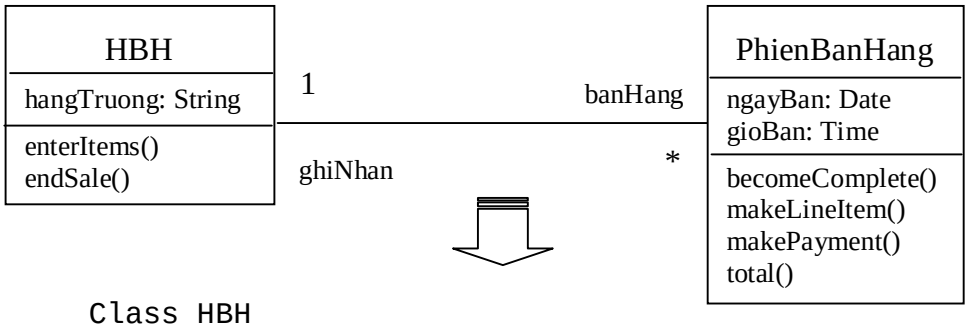
Hình 8-3 Định nghĩa lớp trong ODL

Những thuộc tính phức hợp như *diaChi* được chuyển thành kiểu *struct* như trên.

*Lưu ý:* trong CSDL cũng có các ràng buộc tương với các ràng buộc trong UML, như: *UNIQUE, NOT NULL, CHECK, ASSERTION, TRIGGER* trong SQL.

**Chuyển đổi các quan hệ kết hợp**

Quan hệ kết hợp nhị nguyên được chuyển sang mối quan hệ liên kết *relationship* trong ODL. Chúng ta hãy xét mối quan hệ kết hợp giữa các lớp như hình 8-4.




```
Class HBH
{
  attribute String(25) hangTruong;
  attribute String(15) tenTruong;
  relationship PhienBanHang ghiNhan
      inverse banHang::PhienBanHang;
  void HBH();
```

```
  void ~HBH();
  void enterItems(){…}
  void endSale(){…}
};
Class PhienBanHang
{ attribute Date ngayBan;
  attribute Time gioBan;
  relationship Set<PhienBanHang> banHang
      inverse ghiNhan::HBH;
  void PhienBanHang();
  void ~PhienBanHang();
  Boolean becomeComplete(){…}
  void makeLineItem()
  makePayment(){…}
  Number total(){…}
};
```

Hình 8-4 Chuyển đổi quan hệ kết hợp

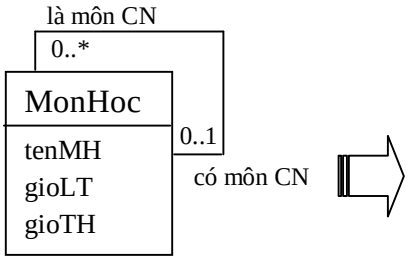
*Lưu ý:* khi một đối tượng kết hợp với nhiều đối tượng của lớp khác thì mối quan hệ đó được có thể được cài đặt bằng kiểu tuyển tập: Set hoặc List, Bag, Array, v.v. Trong biểu đồ lớp ở hình 8-4, **HBH** có quan hệ kết hợp * (nhiều) với **PhienBanHang** , do vậy nó được cài đặt thành **Set<PhienBanHang>** trong ODL. Ngoài ra, tên vai trò ở mỗi đầu quan hệ sẽ là tên xác định trong các nút của đường duyệt các mối quan hệ giữa các lớp.

ODMG hỗ trợ để định nghĩa các mối quan hệ kết hợp hai chiều, song đối với những quan hệ một chiều thì hiệu quả hơn. Trong các hệ QTCSDL, các mối quan hệ sẽ được cài đặt bằng các con trỏ.

Nếu quan hệ kết hợp có các thuộc tính thì có hai khả năng:

1. Nếu bội số trên quan hệ là *nhiều-nhiều,* thì chuyển quan hệ kết hợp thành một lớp mới, trong đó định nghĩa các thuộc tính như hai quan hệ giữa hai lớp.
2. Nếu bội số của quan hệ là *một- nhiều*, thì có thể định nghĩa thuộc tính bên trong lớp kết hợp ở đầu *một.*

Ví dụ, lớp **MonHoc** tự kết hợp với chính nó được định nghĩa trong ODL như sau:



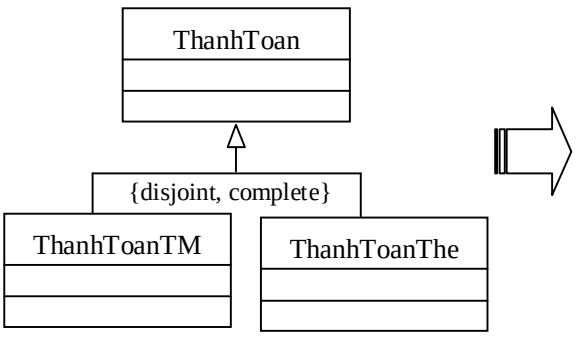

```
Class MonHoc
{ attribute String(20) tenMH;
  attribute short gioLT;
  attribute short gioTH;
  relationship Set<MonHoc> coMonCN
     inverse laMonCN::MonHoc;
  relationship MonHoc laMonCN
     inverse coMonCN::MonHoc;
  void MonHoc();
  void ~MonHoc();};
```

Hình 8-5 Định nghĩa quan hệ tự kết hợp trong ODL

**Quan hệ tổng quát hoá và cụ thể hoá**

UML hỗ trợ hầu như tất cả các quan hệ tổng quát hoá: rời nhau / giao nhau (*disjoint/overlapping*), đầy đủ / không đầy đủ (*complete/incomplete*).

Mô hình đối tượng ODMG không hỗ trợ tổng quát hoá mà có giao nhau. Nói chung nó hỗ trợ quan hệ kế thừa rời nhau và không đầy đủ như hình 2-6.




```
Class ThanhToanTM extends ThanhToan
(extent ThanhToanTM Key number)
{
…
};

Class ThanhToanThe extends ThanhToan
(extent ThanhToanThe)
{
…
};
```

Hình 8-6 Định nghĩa quan hệ kế thừa trong ODL

**Quan hệ kết tập**

UML hỗ trợ hai loại quan hệ kết tập:

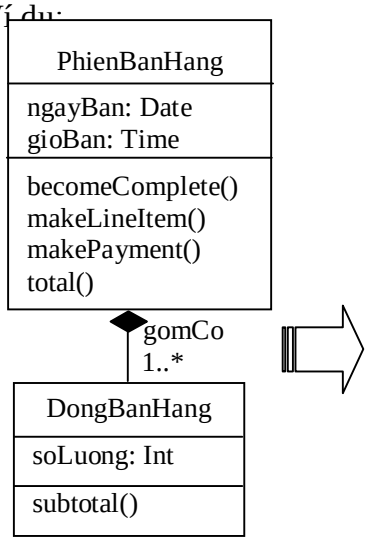
*Kết tập tuyển tập các thành viên* (*member-collection aggregation*) là một loại đặc biệt của quan hệ kết hợp (*association*). Quan hệ này biểu diễn tuyển tập của các đối tượng.

*Cây kết tập (aggregation tree)* là mối quan hệ kết tập giữa bộ phận và tổng thể (*part-whole aggregation*). Quan hệ này thể hiện như là lớp cấu trúc gồm hầu như là hai lớp khác nhau.

ODL không cung cấp một toán tử tạo lập nào để hỗ trợ trực tiếp việc định nghĩa kết tập, và do vậy cả hai loại quan hệ trên có thể được định nghĩa thông qua các thuộc tính.

1. Kết tập tuyển tập được chuyển kiểu tuyển tập, nếu các phần tử được sắp thì chuyển thành **List**, **Array**, **Sequence**, ngược lại có thể sử dụng **Set, Bag**.
2. Kết tập bộ phận / tổng thể định nghĩa thành lớp thành phần như là thuộc tính trong lớp cấu thành.

Ví dụ:




```
Class PhienBanHang
{ attribute Date ngayBan;
  attribute Time gioBan;
  attribute List<DongBanHang> gomCo;
  Boolean becomeComplete(){…}
  void makeLineItem()
  void makePayment(){…}
  Number total(){…}
  PhienBanHang();
  ~PhienBanHang();
};
Class DongBanHang
{ attribute Integer soLuong;
  Number subtotal(){…}
};
```

Hình 8-7 Định nghĩa kết tập tuyển tập trong ODL

**Các kiểu cấu trúc và các tuyển tập**

Như chúng ta đã thảo luận, kiểu dữ liệu có cấu trúc và các tuyển tập là những khái niệm chính cần xây dựng trong thiết kế CSDL HĐT. Chúng cho phép định nghĩa các thuộc tính đa trị, các lực lượng của quan hệ kết hợp, kết tập, v.v.

ODMG hỗ trợ các loại tuyển tập khác nhau:

+ Kiểu **Bag**, cho phép lặp, không được sắp thứ tự và số phần tử không giới hạn.

+ Kiểu **Set**, không lặp, không sắp thứ tự, không bị giới hạn về số lượng.

+ Kiểu **List,** được sắp thứ tự, không bị giới hạn về số lượng, có thể lặp hoặc không lặp.

+ Kiểu **Dictionary**, các cặp phần tử (Key-value) không bị chặn về số lượng với điều kiện là Key không bị lặp.

+ Kiểu **Array** hoặc **Sequence,** giới hạn về số lượng các phần tử.

+ Kiểu cấu trúc (**Struct**), được sử dụng khi bạn muốn biểu diễn cho những đối tượng phức hợp, trong đó mỗi phần tử có thể có nhiều kiểu khác nhau.

Mặc dù việc cài đặt chúng là có thể giống nhau, song các giá trị và các kiểu là cơ bản khác nhau. Một đối tượng trong CSDL là một thể hiện kết quả tính toán của đối tượng trong thế giới thực. Các giá trị tự chúng không tồn tại độc lập, chúng cho phép định nghĩa các đối tượng của hệ thống.

Tương tự như trên, chúng ta thực hiện chuyển đổi tất cả các lớp và các mối quan hệ của chúng về ODL.

## Bài tập và câu hỏi

**8.1** Những mệnh đề sau mô tả Java như là ngôn ngữ lập trình hướng đối tượng, hãy chọn những cụm từ dưới đây để điền vào chỗ [(…)] cho thích hợp.

> "CSDL HĐT là một [(1)] [(2)]. Trong CSDL QH lưu trữ các dữ liệu như [(3)] và [(4)] thì CSDL HĐT chỉ lưu giữ [(5)]. Một phần mềm cho phép tạo lập, quản lý, xử lý, huỷ bỏ các đối tượng được gọi là [(6)]. Tập các mục dữ liệu được tập hợp lại bởi hệ QT CSDL HĐT được gọi là [(2)]".

*Chọn các câu trả lời:*

a. CSDL HĐT (cơ sở dữ liệu hướng đối tượng)

b. đối tượng

c. thế hệ mới

d. các ký tự

e. các giá trị số

f. hệ QT CSDL HĐT (quản trị cơ sở dữ liệu hướng đối tượng)

**8.2** Những mệnh đề sau mô tả về cơ chế đọc/ghi trong CSDL hướng đối tượng, hãy chọn những cụm từ dưới đây để điền vào chỗ [(…)] cho thích hợp.

> "Trong [(1)], đối tượng sau khi xử lý được lưu trữ vào CSDL bởi hệ QT CSDL HĐT sau [(2)] kết thúc. Những đối tượng được lưu trữ vào CSDL gọi là [(3)], còn đối tượng không được lưu vào CSDL gọi là [(4)]".

*Chọn câu trả lời:*

a. chương trình

b. đối tượng tạm thời

c. một CSDL HĐT

d. đối tượng bền vững

**8.3** Hãy chọn những từ thích hợp để điền vào các chỗ trong [(…)] cho đúng với những đặc tính của CSDL HĐT.

> + CSDL HĐT cho phép xử lý [(1)] có nhiều [(2)] có thể được xáo trộn vào một CSDL.
>
> + CSDL HĐT có thể lưu trữ [(3)] mà CSDL truyền thống (CSDL QH) không lưu trữ và quản lý được.
>
> + Trong CSDL HĐT, các dữ liệu và [(4)] là [(5)], và vì vậy dữ liệu có thể nhận được từ [(4)].
>
> + Trong CSDL HĐT, NSD không cần [(6)] về việc đọc/ghi dữ liệu như thế nào.

**8.4** Sử dụng ObjecStore và C++ để cài đặt tất cả các lớp ở hình 8-3 (chi tiết tham khảo http: www.odi.com.

# TÀI LIỆU THAM KHẢO


[1] Booch G., Rumbaugh J. and Jacobson I., *The Unified Software Development Process,* Addision – Wesley, 1998

[2] Booch G., Rumbaugh J. and Jacobson I., *The Unified Modeling Language User Guide,* Addision – Wesley, 1999

[3] Sommerville I., *Software Engineering*, 4th Edition, Addition Wesley, 1994

[4] Larman C., *Applying UML and Patterrns: An Instruction to Object-Oriented Analysis and Design,* Prentice Hall, 1997.

[ 5] Michael B., William P., *Object – Oriented Modeling and Design for Database Applications,* Prentice Hall, New Jersey 1998

[6] Oestereich B., *Developing Software with UML, Object-Oriented Analysis and Design in Prctice,* Addision – Wesley, 2000.

[7] OMG, "*The OMG Unified Modeling Language Specification", http:// www.omg.org/uml* , 1999.

[8] Quatrani T., *Visual Modeling With Rational Rose and UML*, Addison-Wesley, *http:// www.rational.com*, 2000.

[9] Liang Y., From use cases to classes: a way of building object model with UML, *Information and Software Technology, 45 (2003) 83-93, www.elservier.com/locate/infsof.*

[10] Zhiming L., *Object-Oriented Software Development Using UML,* UNU /IIST, Macau 2001.

[11] Đặng Văn Đức, *Phân tích thiết kế hướng đối tượng bằng UML (Thực hành với Rational Rose),* NXB Khoa học và Kỹ thuật, Hà Nội 2002.

[12] Đoàn Văn Ban, Phân tích, thiết kế và lập trình hướng đối tượng, NXB Thống Kê 1997.

[13] Đoàn Văn Ban, Hoàng Quang, *Chuyển đổi các biểu thức đại số quan hệ thành câu truy vấn trong mô hình dữ liệu hướng đối tượng,* Tạp chí Khoa học và Công nghệ, Tập 40-Số ĐB, 2002 (120-129)

[14] Đoàn Văn Ban, Cơ sở dữ liệu hướng đối tượng, *giáo trình Khoa CNTT, HN 2003*

## Danh sách thuật ngữ và các từ viết tắt

*1. Từ viết tắt tiếng Anh*

| | |
|---|---|
| UML | *Unified Modeling Language* |
| OOA | Object Oriented Analysis |
| OOD | Object Oriented Design |
| API | Application Programming Interface |
| CASE | Computer Aided Software Engineering |
| OMT | Object Modeling Technique |
| OOSE | Object-Oriented Software Engineering |
| OCL | Object Constraints Language |
| GRASP | General Responsibility Assignment Software Patterrn |
| ODMG | Object Data Management Group |
| USPD | Unified Software Development Process |
| RUP | Rational Unified Process |
| UP | Unified Process |

*2. Từ viết tắt tiếng Việt*

| | |
|---|---|
| CNTT | Công nghệ thông tin |
| CSDL | Cơ sở dữ liệu |
| NSD | Người sử dụng |
| CNPM | Công nghệ phần mềm |
| HBH | Hệ thống bán hàng |
| CSDLQH | Cơ sở dữ liệu quan hệ |
| CSDLHĐT | Cơ sở dữ liệu hướng đối tượng |
| QTCSDL | Hệ quản trị cơ sở dữ liệu |

*3. Các thuật ngữ Việt - Anh*

| | |
|---|---|
| Hệ thống phần mềm | Software System |
| Tính tiện dụng | usability |
| Khả năng duy trì hoạt động | Maintainability |
| Tính tin cậy | Dependability |
| Tính hiệu quả | Efficiency |
| Hệ thống thông tin | Information System |
| Các hệ thống kỹ thuật | Technical Systems |
| Các hệ thống nhúng thời gian thực | Embedded Real_time Systems |
| Các hệ thống nghiệp vụ | Business Systems |
| Cách tiếp hướng chức năng | Functional-Oriented Approach |
| Cách tiếp cận hướng đối tượng | Object-Oriented Approach |
| Hướng thủ tục | Procedure-Oriented |
| Chia để trị | Devide and conquer |
| Cố kết | Cohension |
| Thông điệp | Message |
| Lớp | Class |

| | |
|---|---|
| Thuộc tính | Attribute |
| Dữ liệu thành phần | Data member |
| Phương thức | Method |
| Thao tác | Operation |
| Hàm thành phần | Member function |
| Bao gói | Encapsulation |
| Che giấu thông tin | Information hidding |
| Mô hình thác nước | Waterfall model |
| Mô hình xoán ốc | Spiral model |
| Công nghệ thế hệ thứ tư 4GT | Fourth Generation Technology |
| Nguyên mẫu | Prototype |
| Vật phẩm | Artifact |
| Được hướng dẫn bởi ca sử dụng | Use-case-driven |
| Tập trung vào kiến trúc | Architecture-centric |
| Quan sát theo ca sử dụng | Use case view |
| Quan sát logic | Logic view |
| Quan sát thành phần | Component view |
| Quan sát tương tranh | Concurrency view |
| Quan sát triển khai | Deployement view |
| Biểu đồ ca sử dụng | Use case diagram |
| Biểu đồ lớp | Class diagram |
| Biểu đồ | Object diagram |
| Biểu đồ trình tự | Sequence diagram |
| Biểu đồ cộng tác | Collaboration diagram |
| Biểu đồ thành phần | Component diagram |
| Biểu đồ triển khai | Deployment diagram |
| Biểu đồ hành động | Activity diagram |
| Biểu đồ trạng thái | State diagram, State chart diagram |
| Gói | Package |
| Luồng | Thread |
| Tác nhân | Actor |
| Định danh | Identifier (ID) |
| Đa xạ, đa hình | Polymorphic |
| Nạp chồng | Overloading |
| Liên kết | Link |
| Sự kết hợp | Association |
| Sự kết tập, gộp lại | Aggregation |
| Quan hệ hợp thành | Composition Aggregation |
| Quan hệ kết hợp | Association relationship |
| Quan hệ kết tập | Aggregation relationship |
| Quan hệ phụ thuộc | Depedency relationship |
| Tổng quát hóa | Generalization |
| Kế thừa | Inheritance |
| Kế thừa đơn | Single inheritance |

| | |
|---|---|
| Kế thừa bội | Multiple inheritance |
| Bội số, bản số | Multiplicity |
| Tên của vai trò | Role name |
| Qui tắc ràng buộc | Constraint rule |
| Qui tắc suy dẫn | Derivation rule |
| Lớp mẫu | Template class |
| Lớp hiện thực | Instantiated class |
| Lớp tiện ích | Class utility |
| Mẫu rập khuôn | Stereotype |
| Lớp biên | Boundary class |
| Lớp thực thể | Entity class |
| Lớp điều khiển | Control class |
| Từ điển thuật ngữ | Glossary |
| Tuyển tập | Collection |
| Chỉ số | Index |
| Siêu dữ liệu | Meta data |

# MỤC LỤC