Chương 8 THIẾT KẾ LỚP

Mục tiêu

Cung cấp các kiến thức về:

- Một số các tiên đề và hệ quả trong thiết kế hệ thống hướng đối tượng nhằm giúp người thiết kế đạt được một kết quả thiết kế tốt.
- Bước đầu tiên tinh chế các lớp trong giai đoạn phân tích thành các lớp có thể cài đặt trong hệ thống phần mềm bao gồm: thiết kế thuộc tính, mối kết hợp, và method

Các tiên đề và hệ luận trong thiết kế hướng đối tượng

Các tiên đề

Theo Suh, trong tiếp cận thiết kế hướng đối tượng có hai tiên đề được áp dụng, các tiên đề là:

Tiên đề 1: tiên đề độc lập.

Duy trì sự độc của các thành phần: Trong quá trình thiết kế, chúng ta đi từ yêu cầu và use case đến một thành phần hệ thống, mỗi thành phần phải thoã mãn yêu cầu mà không ảnh hưởng đến những thành phần khác.

Tiên đề 2: tiên đề thông tin.

Giảm tối đa nội dung thông tin thiết kế. Tiên đề này nói về tính đơn giản hoá trong thiết kế. Mục đích chính là giảm tối đa tính phức tạp, như vậy các đối tượng sẽ dễ bảo trì và nâng cấp hơn. Trong thiết kế hướng đối tượng, cách tốt nhất để giảm độ phức tạp là sử dụng tính thừa kế. (xem hệ luận 6)

Các hệ luận

Từ hai tiên đề trên, có nhiều hệ luận được trích dẫn. Những hệ luận này sẽ mang lại lợi ích hơn trong việc quyết định thiết kế các tình huống cụ thể, vì chúng có thể áp dụng tới các tình huống thực tế dễ dàng hơn so với tiên đề gốc ban đầu.

Hệ luận 1: thiết kế độc lập, giảm tối đa thông tin trao đổi (Uncouple Design with Less Information)

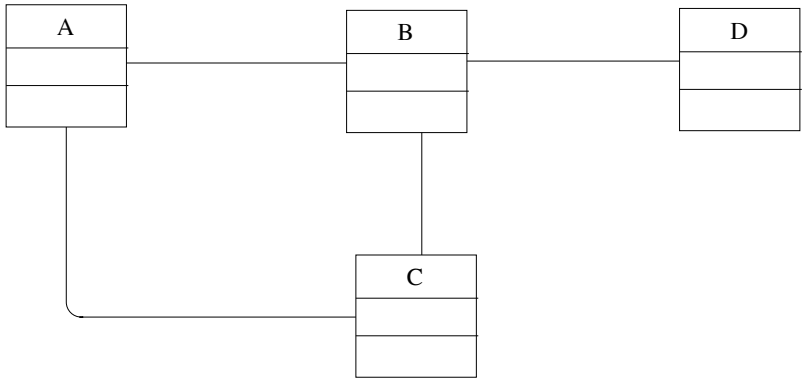
Sự cố kết giữa các đối tượng cao có thể làm giảm tính liên kết giữa chúng. Bởi vì chỉ một lượng nhỏ các thông tin cần thiết trao đổi giữa các đối tượng đó.

Coupling: Ở giai đoạn thiết kế, coupling được dùng để đo mức độ liên kết giữa các đối tượng hoặc giữa thành phần phần mềm. Coupling là một mối kết hợp nhị phân, và là một khái niệm quan trọng khi đánh giá một thiết kế bởi vì nó giúp chúng ta tập trung đúng vào vấn đề quan trọng của thiết kế. Ví dụ, một thành phần trong hệ thống thay đổi sẽ có một tác động tối thiểu đến những thành phần khác. Coupling trong các đối tượng càng mạnh càng mạnh thì hệ thống càng phức tạp. Mức độ của coupling có thể được đánh giá trên:

- Mức độ phức tạp của kết nối
- Kết nối tham chiếu đến chính bản thân đối tượng hoặc bên ngoài đối tượng
- Các thông điệp nhận và gửi đi

Mức độ coupling giữa hai thành phần được xác định bằng mức độ phức tạp của thông tin trao đổi giữa chúng. Coupling càng gia tăng thì càng làm gia tăng độ phức tạp hoặc mơ hồ, tối nghĩa của giao diện. Coupling càng giảm khi sự kết nối được thiết lập ở thành phần giao diện thay vì ở thành phần bên trong. Trong thiết kế hướng đối tượng, có hai loại coupling là coupling tương tác và coupling thừa kế:

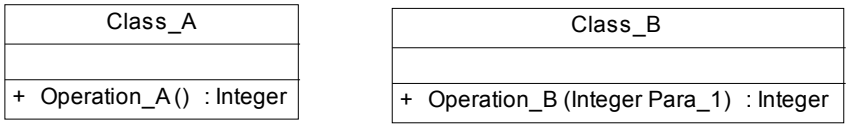
Coupling tương tác: thể hiện qua số lượng và độ phức tạp của các thông điệp giữa những thành phần, sự tương tác càng ít thì càng được ưu tiên. Coupling cũng áp dụng tới mức độ phức tạp của thông điệp. Một chỉ dẫn chung là giữ các thông điệp càng đơn giản và càng ít xảy ra thì càng tốt. Tổng quát, nếu một kết nối thông điệp gồm ba tham số trở lên thì kiểm tra xem có thể làm đơn giản hoá nó nữa được không. Một đối tượng được kết nối với nhiều thông điệp phức tạp thì gọi là liên kết chặt (*tightly coupled)*, có nghĩa là bất kỳ có sự thay đổi nào có thể tác động đến sự thay đổi trong những cái khác.



Hình 4. Ví dụ về biểu diễn coupling và B là liên kết chặt

Năm mức độ coupling tương tác

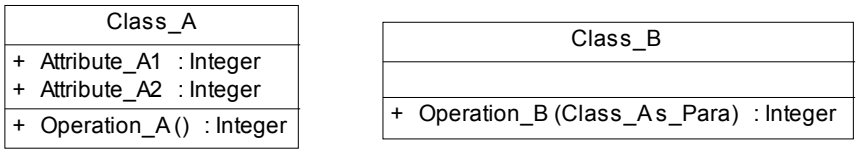
Data coupling: kết nối giữa các thành phần hoặc là dữ liệu dạng nguyên tố hoặc là hoặc cấu trúc tổng hợp tất cả yếu tố được dùng bởi đối tượng nhận. Đây là loại coupling tốt nhất và là mục đích của thiết kế kiến trúc. Các đối tượng trao đổi với nhau thông qua các dữ liệu thành tố hoặc các cờ hiệu thông tin. Thành phần này không quan tâm đến bất cứ gì bên trong thành phần khác mà chỉ quan tâm đến dữ liệu gì nó cần và dữ liệu gì nó gởi trả.




```
integer Operation_A()
{
int x,y;
Class_B cB;
….
y = cB.Operation_B(x);
…
}
```

Operation_A không quan tâm đến nội dung thực hiện của Operation_B mà chỉ quan tâm đến dữ liệu gởi đến và nhận từ Operation_B

Stamp coupling: trong stamp coupling, dữ liệu trao đổi là một phần của cấu trúc hoặc toàn bộ cấu trúc. Loại coupling này được đánh giá là thấp hơn so với data coupling bởi vì sự trao đổi là một cấu trúc thay vì một yếu tố đơn nên làm cho nó phức tạp hơn. Một thay đổi trong cấu trúc sẽ ảnh hưởng đến tất cả đối tượng sử dụng nó. Stamp coupling làm cho các thành phần phụ thuộc nhiều hơn đến thành phần khác, bởi vì, để tránh những lỗi có thể xảy ra, những thành phần sẽ phải có hiểu biết về hoạt động bên trong của thành phần khác sử dụng cùng cấu trúc dữ liệu.




```
integer Operation_A()
{
int x,y;
Class_B cB;
….
y = cB.Operation_B(this);
…
}
```

Operation_B nhận dữ liệu là cấu trúc Class_A để thực hiện. Do đó, nội dung của nó sẽ sử dụng dữ liệu từng thành phần của cấu trúc Class_A. Một sử thay đổi trong cấu trúc này sẽ ảnh hưởng đến Operation_B

Control coupling: khi một thành phần thành phần gởi một thông tin điều khiển tới một thành phần khác, hai thành phần này được gọi là có control coupling. Thông tin điều khiển có thể xuất hiện dưới dạng cờ hiệu thông báo cho thành phần khác hành động sẽ thực hiện. Khi thông tin điều khiển được gởi đi, thành phần gởi phải biết về hoạt động bên trong của thành phần nhận, do đó nó tạo ra một sự phụ thuộc giữa các thành phần.

Common coupling: khi hai thành phần tham khảo đến cùng một vùng dữ liệu chung toàn cục thì có liên kết common coupling. Liên kết này cũng nên tránh bởi vì nó tạo ra một cơ hội lớn về lỗi trên toàn bộ hệ thống. Một lỗi phát sinh trong bất kỳ một thành phần nào sử dụng dữ liệu toàn cục này có thể gây ra lỗi trong bất kỳ thành phần khác sử dụng cùng dữ liệu này.

Content coupling: loại coupling được xếp thấp nhất là content coupling. Bởi vì loại này giới thiệu một thành phần tham khảo trực tiếp hoạt động bên trong của một thành phần khác. Ví dụ, một method có thể thay đổi dữ liệu trong một method khác hoặc thay đổi một đoạn code trong method khác. Hiện nay, các ngôn ngữ lập trình (đặc biệt là ngôn ngữ lập trình hướng đối tượng) đều không cho phép tạo ra content coupling.

Tên coupling	Xếp hạng phụ thuộc
Data coupling	Rất thấp
Stamp coupling	Thấp
Control coupling	Trung bình
Common coupling	Cao
Content coupling	Rất cao

Coupling thừa kế: là coupling giữa lớp tổng quát và lớp chuyên biệt. Một lớp chuyên biệt liên kết với lớp tổng quát của nó thông qua thuộc tính và method. Không như coupling tương tác, coupling thừa kế càng cao thì càng ưu tiên. Tuy nhiên, để đạt được mức độ cao coupling thừa kế trong một hệ thống. Mỗi lớp chuyên biệt không nên thừa kế những method và thuộc tính không liên quan hoặc không cần thiết. Ví dụ, nếu một lớp chuyên biệt chồng lên hầu hết các method hoặc không sử dụng nó từ lớp tổng quát, điều này cho thấy coupling thừa kế là thấp và lúc đó thiết kế viên phải thay đổi lại cách xây dựng tổng quát hoá và chuyên biệt hoá này.

Cohesion

Trong khi coupling đề cập đến mức độ tương tác giữa các đối tượng thành phần thì cohesion lại đề cập đến sự tương tác bên trong một đối tượng thành phần. Cohesion phản ánh tính "đơn mục đích" của một đối tượng. Tất cả các lệnh, chức năng trong một thành phần được định nghĩa gắn liền tới một nhiệm vụ. Trong hệ thống nếu các thành phần đạt được mức độ cohesion càng cao thì mức độ liên kết coupling giữa các thành phần càng yếu, do đó để đạt được đỉnh cao của cohesion thì chúng ta phải làm giảm mức độ coupling. Cohesion cũng trợ giúp trong thiết kế lớp bằng việc giúp xác định mục đích và mục tiêu của lớp một cách rõ ràng.

Method cohesion: một method chỉ nên đảm nhận một chức năng hoặc một nhiệm vụ. Thông thường cách đặt tên của method cũng phải ngụ ý nói lên chức năng liên quan của nó. Ví dụ: Chon_nha_cung_cap(), Tinh_ton(),…

Class cohesion: các method và các thuộc tính trong một lớp phải có mức độ cohesion cao, nghĩa là phải được dùng bởi các method bên trong lớp hoặc chính là method phục vụ cho mục đích của lớp.

Cohesion thừa kế: các lớp chuyên biệt và lớp tổng quát phải cùng mô tả về một loại đối tượng của hệ thống. Các thuộc tính và hành vi của lớp tổng quát phải được thừa hưởng tối đa trong lớp chuyên biệt theo cách hoặc là phục vụ cho hành vi của lớp chuyên biệt hoặc trở thành một hành vi của lớp chuyên biệt.

Hệ luận 2: đơn mục đích (single purpose)

Mỗi lớp phải có một mục đích xác định trong việc đạt được mục tiêu hệ thống. Nếu chúng ta mô tả một lớp phức tạp thì hãy phân chia nó thành những lớp nhỏ hơn, đơn giản và xác định hơn. Mỗi method chỉ cung cấp một dịch vụ, kích thước không quá lớn (không nên vượt quá một trang coding)

Hệ luận 3: nhiều lớp đơn. Tạo các lớp đơn để cho phép tái sử dụng

Kết quả thiết kế hệ thống tốt chính là tạo ra một tập lớn các lớp đơn giản hơn là một tập nhỏ lớp phức tạp. Các lớp càng ít chuyên biệt thì các vấn đề trong tương lai càng khó giải quyết hơn thông qua việc kết nối các lớp đang có và tái sử dụng chúng.

Thiết kế hướng đối tượng luôn đề xuất việc sản sinh thư viện các thành phần tái sử dụng và nhấn mạnh trên tính bao bọc (encapsulation), đơn vị hoá (modularization), và tính đa hình (polymorphism) để tái sử dụng khi xây dụng một đối tượng hơn là làm mới.

Hệ luận 4: ánh xạ kết quả giữa các giai đoạn phải chặt chẽ (strong mapping)

Giai đoạn phân tích và thiết kế dựa trên cùng mô hình, kết quả mô hình. Từ quá trình phân tích đến cài đặt, các chi tiết sẽ được đưa thêm vào nhưng vẫn duy trì về cơ bản giống nhau. Ví dụ, trong giai đoạn phân tích chúng ta xác định lớp Hoá đơn. Trong giai đoạn thiết kế chúng ta thiết kế lớp Hoá đơn: method, dữ liệu,…

Hệ luận 5: chuẩn hoá (standardization).

Đề xuất sự chuẩn hoá bằng việc thiết kế các thành phần có thể thay thế và tái sử dụng các thành phần có sẵn.

Để tái sử dụng, chúng ta phải có một hiểu biết sâu về các lớp trong môi trường lập trình hướng đối tượng chúng ta đang dùng. Hầu hết các hệ thống hướng đối tượng đều có các các lớp thư viện xây dựng sẵn và ngày càng gia tăng khi chúng ta tạo các ứng dụng mới. Tri thức về các lớp tồn tại giúp chúng ta xác định những gì một lớp mới cần có do thừa kế hơn là phải xây dựng mới. Mặc dù vậy, tài liệu mô tả các lớp thư viện thường không được biên tập tốt hoặc ít được cập nhật thường xuyên khi có sự điều chỉnh, nâng cấp. Do đó, trong quá trình xây dựng các hệ thống chúng ta nên tạo ra các lớp thư viện và tích lũy dần nó từ việc xây dựng các hệ thống và luôn luôn xem xét việc thừa kế nó khi xây dựng một hệ thống mới.

Hệ luận 6: thiết kế thừa kế (design inheritance).

Các đặc tính chung phải được chuyển lên lớp tổng quát. Cấu trúc lớp tổng quát – lớp chuyên biệt phải tạo ra ý nghĩa luận lý.

Thiết kế lớp

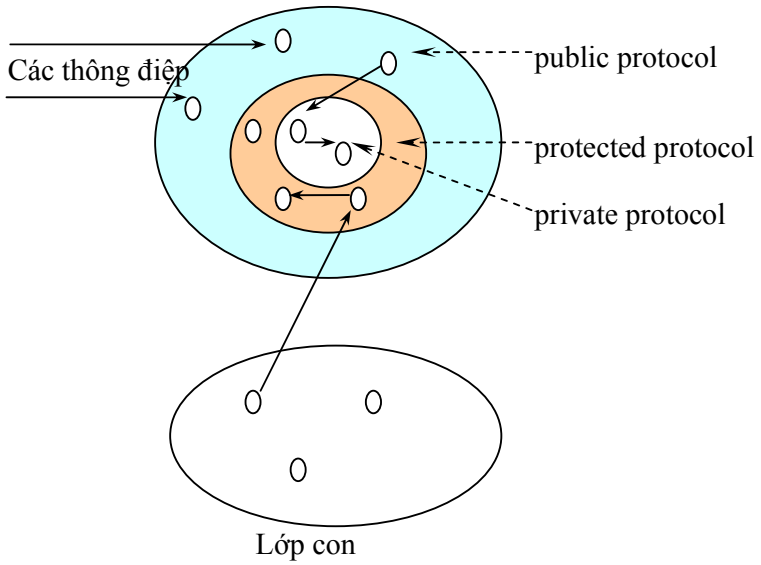
Một tiến trình thiết kế lớp bao gồm:

- Tinh chế thuộc tính
- Thiết kế hành vi (method) và nghi thức (protocol) sử dụng sơ đồ trong UML
- Tinh chế quan hệ giữa các lớp
- Tinh chế sự phân cấp và thiết kế sự kế thừa

Phạm vi ảnh hưởng của lớp

Trong việc thiết kế hành vi và thuộc tính cho các lớp, chúng ta gặp phải hai vấn đề. Thứ nhất là ***nghi thức (protocol)***, hoặc giao diện tới các toán tử và sự có thể thấy (visibility) của nó; thứ hai là cách thức cài đặt nó. Nghi thức được xem là cách thức xác định các đặc trưng của lớp có thể "nhìn thấy" từ bên ngoài hoặc cỉ được xem là "nội bộ" bên trong. Các nghi thức đó là: toàn cục (public protocol), riêng (private protocol), và bảo vệ (protected protocol).

- pupblic protocol: định nghĩ các hành vi trạng thái của lớp
- private protocol: dùng để xác định các đặc trưng của lớp chỉ được truy cập bởi chính lớp đó
- Protected protocol: xác định đặc trưng của một lớp có thể sử dụng bởi chính nó và lớp con của nó.



Tinh chế thuộc tính

Các thuộc tính trong giai đoạn phân tích hướng đối tượng phải được tinh chế theo cách nhìn về việc cài đặt nó trong môi trường tin học. Trong giai doạn phân tích, chúng ta chỉ cần tên của thuộc tính là đủ. Tuy nhiên, trong giai đoạn thiết kế, thông tin chi tiết về thuộc tính đó phải được thêm vào. Mục tiêu chính của hoạt động này là tinh chế các thuộc tính tồn tại (được xác định trong giai đoạn phân tích) và đưa thêm các thuộc tính dùng cho việc cài đặt nhằm đặc tả lớp như một thành phần của môi trường tin học.

Kiểu thuộc tính

Có ba kiểu cơ bản của thuộc tính:

- Thuộc tính đơn trị
- Thuộc tính đa trị
- Thuộc tính dùng để tham chiếu tới các đối tượng khác hoặc tới một thể hiện kết nối

Các thuộc tính thể hiện cho trạng thái của đối tượng. Khi một trạng thái của đối tượng thay đổi, sự thay đổi này được phản ánh qua giá trị của các thuộc tính. Thuộc tính đơn trị là loại phổ biến nhất, nó chỉ có một giá trị hoặc một trạng thái tương ứng với một đối tượng. Ví dụ: *Tên, Ngày sinh, Mức lương*,…

Thuộc tính đa trị có thể có một tập các giá trị tương ứng cho một đối tượng. Ví dụ: chúng ta muốn lưu trữ nhiều số điện thoại của một khách hàng, chúng ta có thể đặt thuộc tính *Số điện thoại* là đa trị.

Thuộc tính tham chiếu được dùng để cung cấp việc tham khảo cần thiết để một đối tượng đáp ứng các trách nhiệm của nó. Nói cách khác, thuộc tính tham chiếu hiện thực hoá mối kết hợp của các đối tượng.

Hiển thị thuộc tính

Trong UML việc trình bày thuộc tính được đề nghị như sau:

<Phạm vi> <tên> : <kiểu thuộc tính> = <giá trị khởi tạo>

Trong đó, *<phạm vi>* sẽ là:

\+ : toàn cục (public protocol)

\# : bảo vệ (protected protocol)

\- : cục bộ (private protocol)

<kiểu thuộc tính> là một đặc tả cài đặt thuộc tính độc lập ngôn ngữ.

<giá trị khởi tạo> là một biểu thức độc lập ngôn ngữ xác định giá trị khởi tạo khi một đối tượng được tạo mới. tham số này là tuỳ chọn.

Thuộc tính đa trị được xác định bằng việc thêm vào chỉ số mảng theo sau tên thuộc tính. Ví dụ:

Địa_chỉ[3]: string

Tập_hợp_điểm[2..*]: điểm

Ví dụ: tinh chế thuộc tính các lớp của hệ thống ATM

Lớp KháchHàng

#tênKháchHàng: String

#họKháchHàng: String

#mãPIN: String

#sốThẻ: String

#tàiKhoản: TàiKhoản (thuộc tính tham chiếu)

Trong đó, thuộc tính #tàiKhoản dùng để mô tả mối quan hệ giữa lớp KháchHàng và lớp TàiKhoản. Việc thêm thuộc tính này cho phép chúng ta tham khảo đến một đối tượng tài khoản từ một đối tượng khách hàng. Tất cả thuộc tính đều được gán cho một phạm vi truy cập nội bộ dạng nghi thức protected nhằm đảm bảo tính bao bọc và có thể thừa kế nếu sau này các phát triển thêm các lớp con.

Lớp TàiKhoản

#sốTàiKhoản: String

#loạiTàiKhoản: String

#sốDư: float

#giaoTác: GiaoTác (cài đặt mối kết hợp giữa lớp TàiKhoản và lớp GiaoTác)

#khách Hàng: KháchHàng (cài đặt mối kết hợp giữa lớp TàiKhoản và lớp KhácHàng)

Lớp GiaoTác

#giaoDịchID: String

#ngàyGiaoDịch: Date

#thờiGianGiaoDịch: Time

#loạiGiaoDịch: String

#sốTiền: float

#sốDư: float

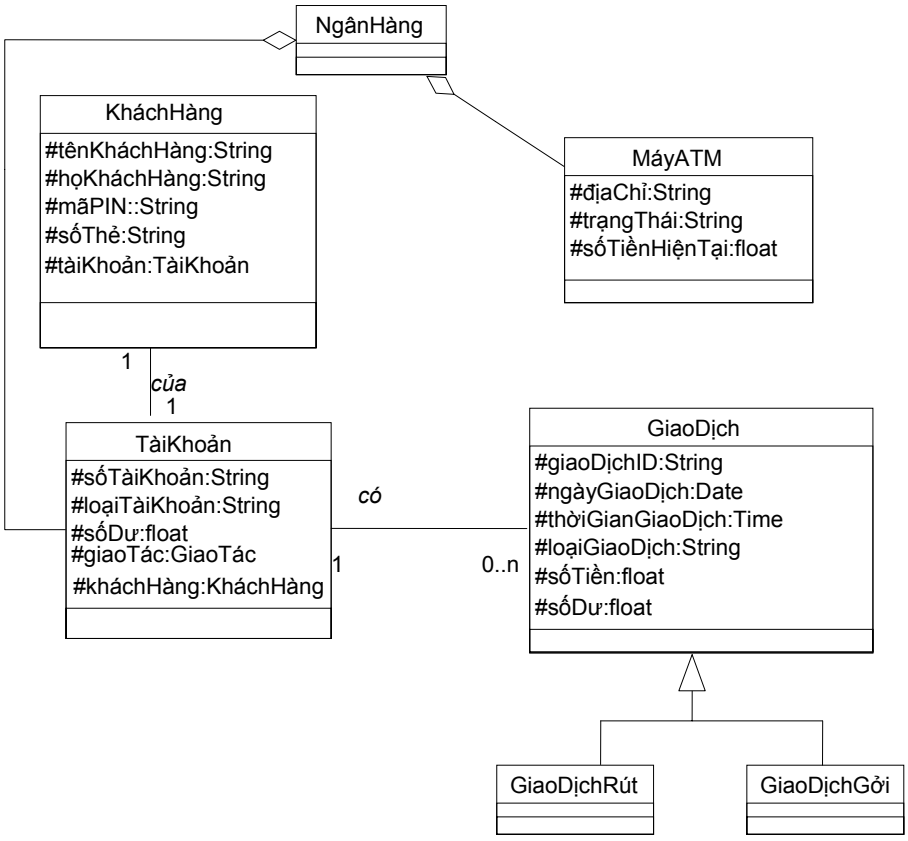
Lớp GiaoTác và lớp TàiKhoản có một mối kết hợp. Khi tính chế thuộc tính cho lớp TàiKhoản, chúng ta đã thêm vào thuộc tính #giaoTác dùng để cài đặt mối kết hợp này. Vậy khi tính chế thuộc tính cho lớp GiaoTác chúng ta cũng có thể thêm vào một thuộc tính cũng để cài đặt cho mối kết hợp này nếu chúng ta có nhu cầu biết thông tin về tài khoản từ một giao tác. Tuy nhiên, với bài toán của chúng ta đây không có nhu cầu đó, vậy nên chúng ta không thêm vào lớp GiaoTác thuộc tính tham chiếu tới lớp TàiKhoản.

Lớp MáyATM

#địaChỉ: String

#trạngThái: String

#sốTiềnHiệnTại:float



Sơ đồ lớp của hệ thống ATM sau khi đã tính chế thuộc tính

Tinh chế hành vi (method)

Mục tiêu chính của hoạt động này là mô tả thuật toán cho các hành vi đã được xác định ở giai đoạn phân tích. Việc mô tả thuật toán cũng có thể được thực hiện trên nhiều cách, hoặc bằng văn bản (mã giả) hoặc bằng sơ đồ. Việc sử dụng sơ đồ (trong UML có thể dùng sơ đồ hoạt động, tuần tự,…) cho phép chúng ta dễ dàng hơn trong việc chuyển đổi chúng sang một ngôn ngữ lập trình một cách thủ công hoặc tự động (thông qua một CASE Tool).

Trong giai đoạn này chúng cũng nên giảm tối đa mức độ phức tạp các kết nối thông điệp giữa các lớp số lượng thông điệp được gởi và nhận bởi một đối tượng. Mục tiêu nhằm gia tăng tính đoàn kết (cohension) trong số các đối tượng và các thành phần phần mềm để cải tiến tính liên kết (coupling), bởi vì chúng ta phải giữ một số lượng tối thiểu các thông tin cần thiết chuyển đổi giữa các thành phần. Áp dụng các tiên đề và hệ luận thiết kế chúng ta có các hướng dẫn sau:

- Một tập lớn các lớp đơn giản sẽ tốt hơn một tập nhỏ các lớp phức tạp
- Tạo một lớp tổng quát cho các lớp mà chúng ta tìm thấy có một số nội dung giống nhau. Mục tiêu là tăng tối ta việc tái sử dụng
- Luôn tập trung vào mục tiêu của lớp khi định nghĩa nhằm tránh việc thiết kế lạc đề hoặc mở rộng vượt khỏi phạm vi ý nghĩa của lớp: điều này thường xảy ra khi chúng ta tạo ra các thay đổi trên lớp đang tồn tại khi bài toán có sự thay đổi và càng ngày tạo ra các lớp với nội dung phức tạp không còn đúng với mục tiêu ban đầu của lớp.

Một lớp có thể có những loại hành vi sau:

- *Constructor*: methode tạo thể hiện (đối tượng) của lớp

- *Destructor*: methode huỷ thể hiện của lớp
- *Conversion*: method chuyển đổi một đơn vị đo lường này sang một đơn vị đo lường khác
- *Copy*: method sao chép nội dung của một thể hiện sang một thể hiện khác
- *Attribute set*: method gán giá trị cho một hoặc nhiều thuộc tính
- *Attribute get*: method trả về giá trị của một hoặc nhiều thuộc tính
- *I/O method*: method cung cấp tới hoặc nhần dữ liệu từ một thiết bị
- *Domain specific*: method xác định tới ứng ụng

Hiển thị hành vi

Theo UML một hành vi của lớp được trình bày theo cú pháp:

<phạm vi> <tên> <(danh sách tham số)> : <kiểu trả về>

Trong đó,

<phạm vi> được qui định giống như của thuộc tính

<danh sách tham số>: mỗi tham số cách nhau bởi dấu phẩy và có cú pháp như sau:

<tên tham số>: <kiểu dữ liệu> = <giá trị mặc định>.

<kiểu trả về> là một đặc tả độc lập ngôn ngữ về cài đặt giá trị trả về của một hành vi. Nếu mục này bị bỏ qua thi hành vi không trả về giá trị

Ví du:

+get_Tên(): String

+get_SốTàiKhoản(vtàiKhoản : TàiKhoản): String

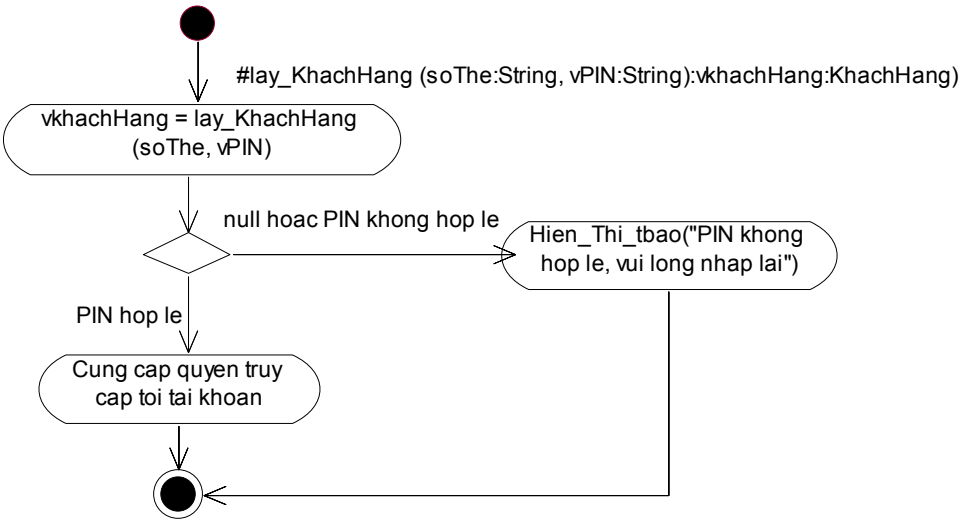
Thiết kế hành vi cho hệ thống ATM

Tại thời điểm này, chúng ta đã xác định các đối tượng hình thành tầng nghiệp vụ của hệ thống, cũng như là các dịch vụ mà các đối tượng này cung cấp. Công việc còn lại là thiết kế hành vi, giao diện người dùng và xử lý truy cập cơ sở dữ liệu. Với hệ thống ATM, chúng ta đã xác định các toán tử ở giai đoạn phân tích bao gồm:

Lớp KháchHàng

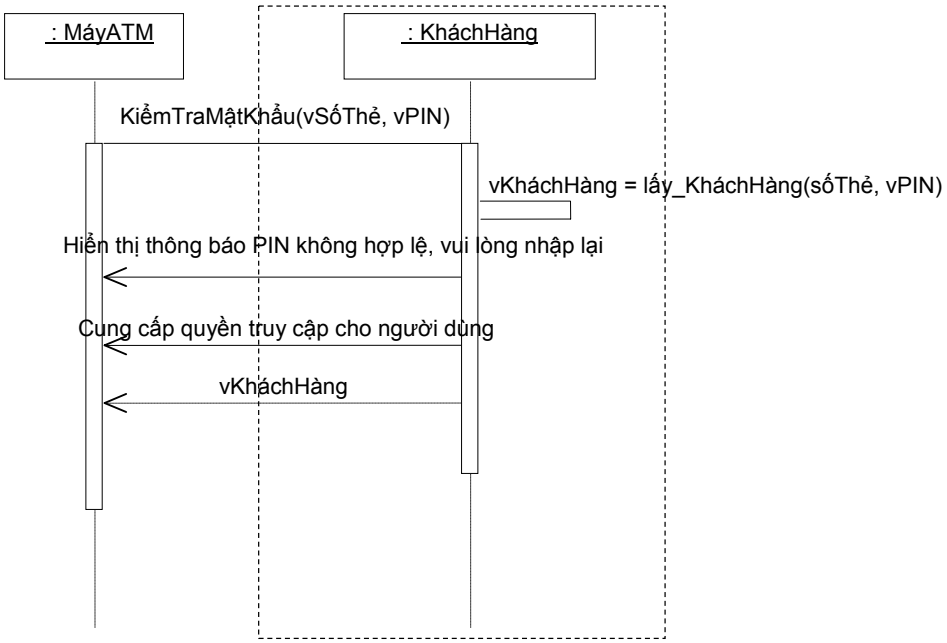
KháchHàng::+kiểmTraMậtKhẩu(sốThẻ:String, vPIN:String): vkháchHàng: KháchHàng

Thiết kế thuật giải cho hành vi dùng sơ đồ hoạt động



Hành vi *kiểmTraMậtKhẩu()* trước hết sẽ thi hành tạo một đối tượng khách hàng và thực hiện lấy thông tin về khách hàng dựa trên dựa trên một số thẻ và mã PIN. Tại đây, chúng ta lại nhận thấy rằng cần phải có một hành vi khác để thực hiện điều này đó là *lấy_KháchHàng()* và có phạm vi nội bộ dạng protected (#). Hành vi này sẽ lấy tham số đầu vào là số thẻ và mã PIN, kết quả trả về là một đối tượng khách hàng tìm thấy hoặc là "null" nếu ngược lại. Nếu giá trị trả về là "null", sẽ gởi một thông điệp tới hệ thống để thực hiện thông báo "PIN không hợp lệ, vui lòng nhập lại", ngược lại, sẽ gán quyền truy cập cho người dùng.

Thiết kế thuật giải dùng sơ đồ tuần tự

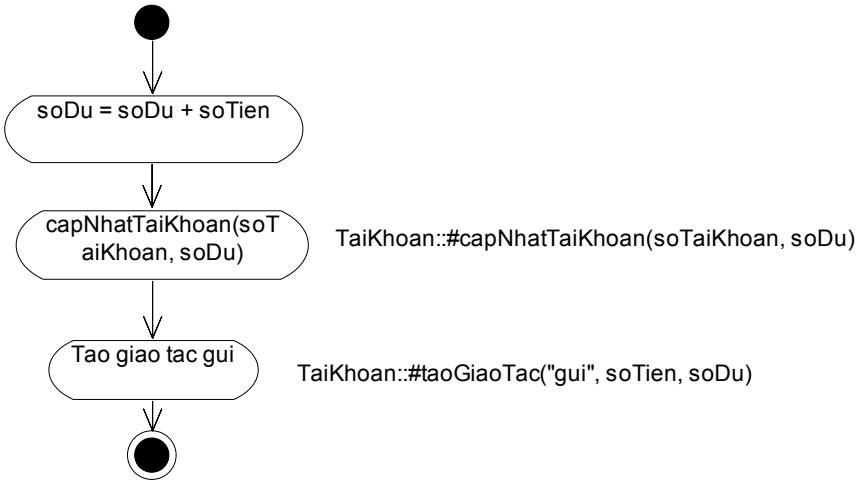


Với đồ trên chúng ta chỉ quan tâm đến các lớp (nghiệp vụ) sẽ cung cấp các dịch vụ gì để thực hiện *kiểmTraMậtKhẩu()*. Chúng ta chưa quan tâm đến các đối tượng ở các tầng khác như là: truy cập cơ sở dữ liệu hoặc giao diện hiển thị. Ví dụ như hành vi *lấy_KháchHàng(sốThẻ, vPIN)* sẽ phải truy cập cơ sở dữ liệu để thực hiện, cũng như đối tượng: MáyATM chỉ là giả lập giao diện giữa khách hàng và hệ thống, chúng ta chưa xác định đối tượng giao diện cụ thể

nào (form, trang web,…) sẽ thực hiện. Phần này sẽ được đề cập chi tiết trong những chương sau.

Lớp TàiKhoản

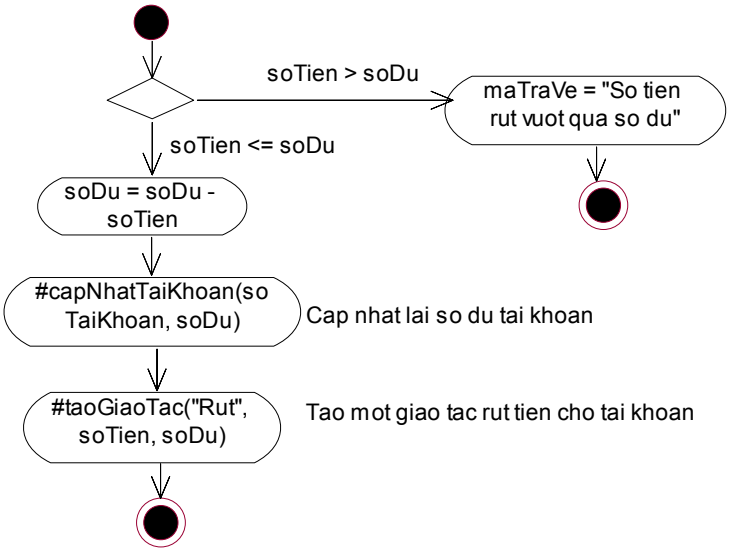
TàiKhoản::+gửiTiền(sốTiền: foat)



Khi thực hiện một việc gửi tiền, số tiền được gửi được chuyển đến một đối tượng tài khoản và được dùng như là một đối số cho hành vi *gửiTiền()*. Tài khoản này điều chỉnh số dư hiện hành của nó bằng cách cộng thêm với số tiền gửi. Sau đó, tài khoản này sẽ lưu thông tin lần gửi bằng cách tạo ra một đối tượng giao tác.

Một lần nữa chúng ta lại khảm phá ra những hành vi khác: *cậpNhậtTàiKhoản()*, hành vì này là cục bộ (#) cho phép cập nhật dữ liệu tài khoản. *tạoGiaoTác()*, cũng là hành vi cục bộ cho phép tạo một giao tác tương ứng với tài khoản này.

TàiKhoản::+rútTiền(sốTiền:float): mãTrảVề:String



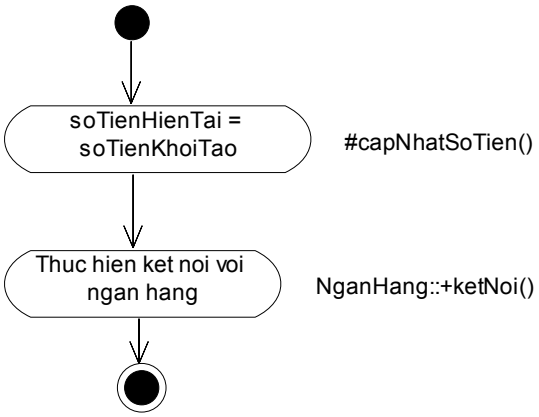
Một số tiền mà khách hàng muốn rút sẽ được chuyển đến một đối tượng tài khoản như là một tham số đầu vào. Tài khoản này sẽ kiểm tra số dư hiện hành của nó so với số tiền này. Nếu

vẫn lớn hơn hoặc bằng số tiền rút thì tài khoản sẽ cập nhật lại số dư và tạo một giao tác rút tiền, ngược lại thông báo lỗi với mãTrảVề "Số tiền rút vượt quá số dư".

Một lần nữa chúng ta thấy hành vi *rútTiền* lại sử dụng *cậpNhậtTàiKhoản* và *tạoGiaoTác* đã đề cập đến trong khi thiết kế hành vi *gửiTiền*.

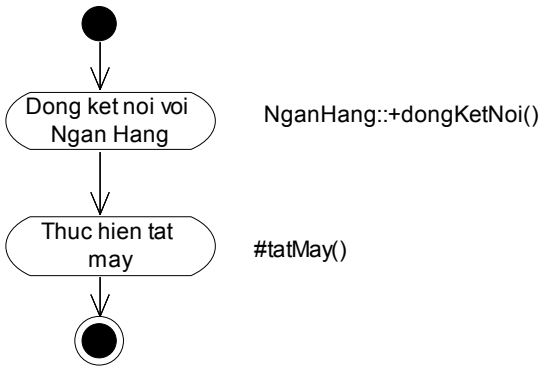
Lớp MáyATM

MáyATM::+khởiĐộngMáy(sốTiềnKhởiTạo:float)



Sau khi bật máy ATM, một số tiền khởi tạo được nhập từ nhân viên vận hành sẽ được chuyển đến đối tượng máyATM như là một tham số đầu vào. Đối tượng máyATM sẽ cập nhật lại số tiền ban đầu cho máy và sau đó thực hiện việc kết nối tới ngân hàng nhằm thực hiện việc liên kết truy cập cơ sở dữ liệu. Quá trình này chúng ta lại có nhu cầu phát sinh thêm hành vi cục bộ *#cậpNhậtSốTiền* và hành vi toàn cục *NgânHàng::+kếtNối()*.

MáyATM::+đóngMáy()

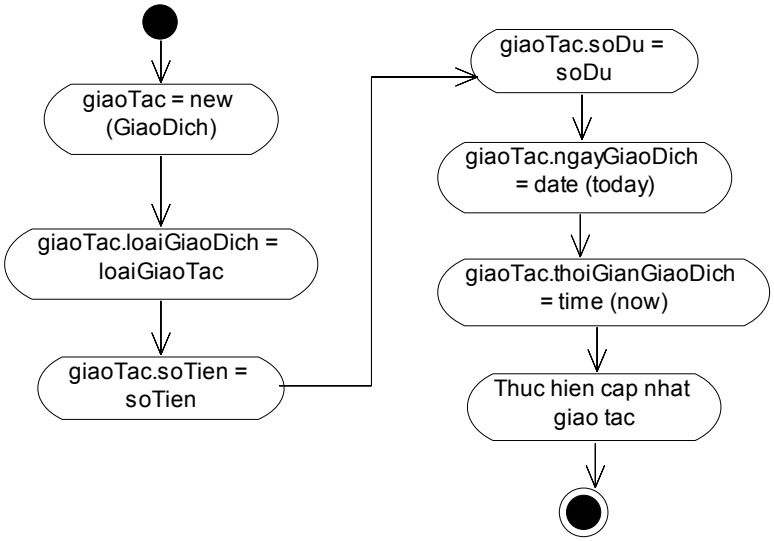


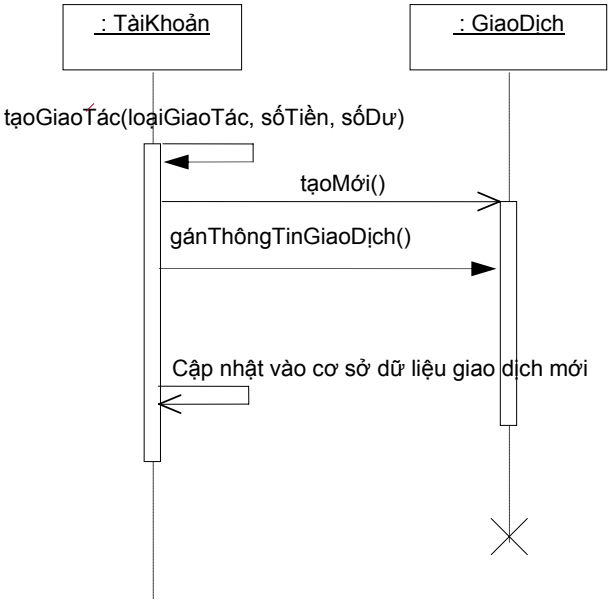
Đối tượng máyATM thực hiện đóng máy bằng cách gọi thực hiện việc đóng kết nối với ngân hàng và gọi thực hiện tắt máy. Quá trình này phát sinh thêm hai hành vi: *+đóngKếtNối()*do đối tượng NgânHàng đảm nhận và *#tắtMáy()* là hành vi cục bộ của đối tượng MáyATM.

Như vậy, chúng ta đã thiết kế xong các hành vi đã được xác định ở giai đoạn phân tích quá trình thiết kế này lại phát sinh các hành vi: *#lấy_KháchHàng(), #cậpNhậtTàiKhoản(), #tạoGiaoTác(), +kếtNối(),+đóngKếtNối(), #cậpNhậtSốTiền(), #tắtMáy()*. Chúng ta lặp lại quá trình thiết kế cho những hành vi này. Chú ý rằng các hành vi liên quan đến việc truy cập dữ liệu hoặc xử lý giao diện sẽ được thiết kế ở những giai đoạn tiếp theo trong những chương sau. Các hành vi đó là: *#lấy_KháchHàng(), #cậpNhậtTàiKhoản(),....* Sau đây chúng ta tiếp

tục thiết kế thuật giải cho các hành vi *tạoGiaoTác(), +kếtNối(), +đóngKếtNối(), #cậpNhậtSốTiền()*:

TàiKhoản::#tạoGiaoTác(loạiGiaoTác:String, sốTiền:float, soDu:float)





Một đối tượng tài khoản tạo một giao tác liên quan đến nó bằng cách truyền các tham số: loại giao tác (gửi, rút), số tiền, và số dư sau khi thực hiện giao tác. Đối tượng tài khoản sẽ tạo mới một đối tượng giao tác ứng với thuộc tính giaoTác của nó và gán các thông tin về giao tác đó, sau đó, lưu lại giao tác này vào cơ sở dữ liệu. Quá trình này lại phát sinh mới hai hành vi: hành vi +gánThôngTinGiaoDịch() của đối tượng giao dịch có phạm vi toàn cục; hành vi cập nhật vào cơ sở dữ liệu mà chúng ta sẽ thiết kế chi tiết trong các chương sau. Tạm thời ở đây là vấn đề nội bộ của đối tượng tài khoản.

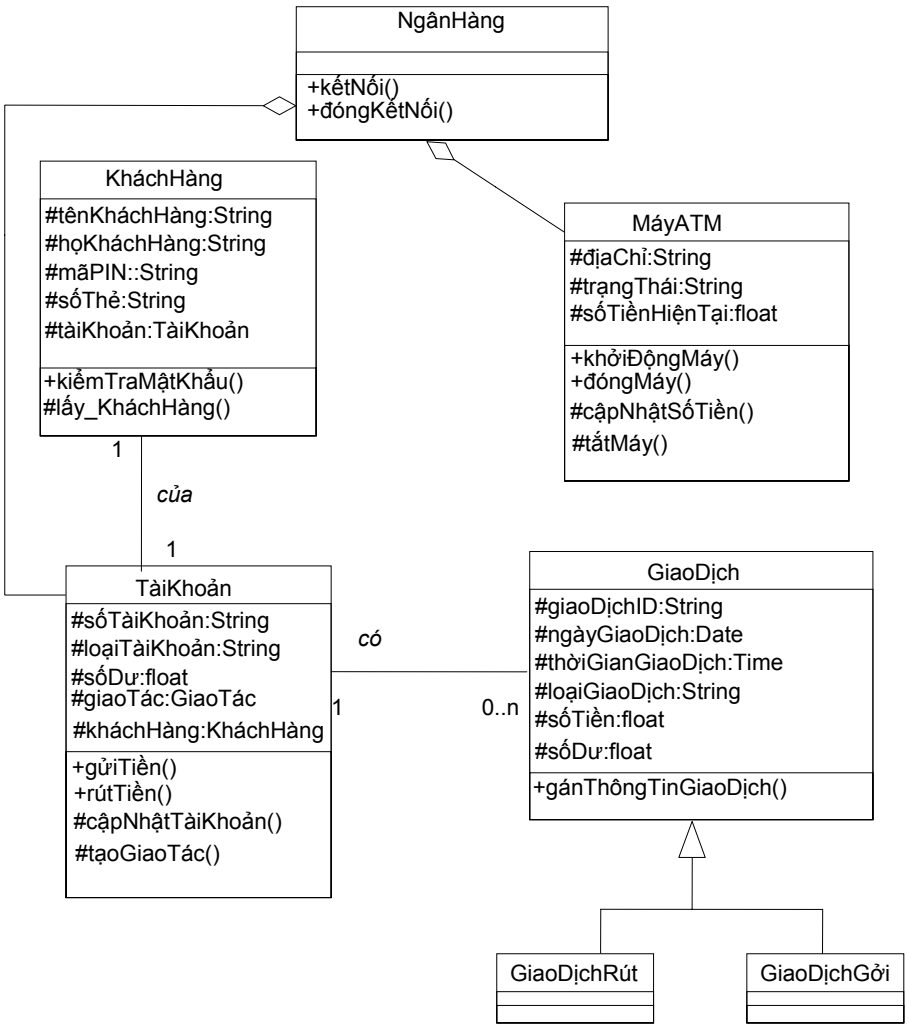
GiaoDịch:: +gánThôngTinGiaoDịch(loạiGD:String, sốTiền:float, sốDư:float, ngàyGD:Date, giờGD:Time)

Các hành vi *+kếtNối(), +đóngKếtNối(), +cậpNhậtSốTiền()* thì đơn giản do đó chúng ta chỉ mô tả khai báo nó:

NgânHàng::+kếtNối()

NgânHàng::+đóngKếtNối()

MáyATM::#cậpNhậtSốTiền(sốTiền:float)



Sơ đồ lớp của hệ thống máy ATM với kết quả tinh chế đầu tiên về thuộc tính và hành vi

Tổng kết

Bước đầu tiên trong tiến trình thiết kế hướng đối tượng là việc áp dụng các tiên đề và hệ luận để thiết kế các lớp, thuộc tính, hành vi, mối kết hợp, cấu trúc, phạm vi; quá trình này là một quá trình lặp và tinh chế. Trong giai đoạn phân tích, chỉ cần tên thuộc tính là đủ. Tuy nhiên, trong giai đoạn thiết kế, các thông tin chi tiết sẽ phải được thêm vào mô hình (đặc biệt là các định nghĩa của thuộc tính và hành vi).

Thiếu đi một nghi thức thiết kế tốt có thể phát sinh các kẽ hở về tính bao bọc (encapsulation). Vấn đề này xảy ra khi khi chi tiết về cài đặt bên trong của một lớp được phơi bày ra ở giao diện. Càng nhiều thông tin chi tiết bên trong của lớp bị phơi bày, sự uyển chuyển trong các thay đổi hệ thống sau này càng giảm. Bởi vì nó làm giảm đi tính độc lập của đối tượng trong hệ thống. Do đó, chúng ta sử dụng khai báo phạm vi cục bộ (private) hoặc bảo vệ (protected)

để xác định việc cài đặt đối tượng; sử dụng khai báo phạm vi toàn cục (public) để xác định chức năng của đối tượng.

Thiết kế hướng đối tượng là một quá trình lặp. Do đó, chúng ta đừng ngại việc thay đổi tới một lớp, mỗi sự thay đổi hãy tin rằng sẽ là một sự cải tiến mô hình hiện tại. Một điều ghi nhớ rằng các vấn đề cần được giải quyết càng sớm càng tốt để khỏi phải trả giá lớn trong các giai đoạn sau.

Câu hỏi và bài tập

1. Các nghi thức public và private là gì? Ý nghĩa của việc sử dụng những nghi thức này trong thiết kế?
2. Khi thiết kế mối kết hợp, chúng ta cần thêm một thuộc tính tham chiếu tới một lớp khi chúng ta cần xác định điều gì?