LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI JAVA

(Object Oriented Programming with Java)

Bộ môn Công nghệ Phần mềm

Đại học SPKT Hưng Yên

BÀI 1: Các khái niệm cơ bản về lập trình hướng đối tượng

1.1. Lịch sử phát triển của phương pháp lập trình

Ngôn ngữ lập trình là một tập con của ngôn ngữ máy tính, được thiết kế và chuẩn hóa để truyền các chỉ thị cho các máy có bộ xử lý (CPU), nói riêng là máy tính. Ngôn ngữ lập trình được dùng để lập trình máy tính, tạo ra các chương trình máy nhằm mục đích điều khiển máy tính hoặc mô tả các thuật toán để người khác đọc hiểu.

Trước hết dạng chương trình duy nhất mà máy tính có thể thực thi trực tiếp là ngôn ngữ máy hay mã máy. Nó có dạng dãy các số nhị phân, thường được ghép nhóm thành byte 8 bit cho các hệ xử lý 8/16/32/64 bit [note 1]. Nội dung byte thường biểu diễn bằng đôi số hex. Để có được bộ mã này ngày nay người ta dùng ngôn ngữ lập trình để viết ra chương trình ở dạng văn bản và dùng trình dịch để chuyển sang mã máy [1].

Khi kỹ thuật điện toán ra đời chưa có ngôn ngữ lập trình dạng đại diện nào, thì phải lập trình trực tiếp bằng mã máy. Dãy byte viết ra được đục lỗ lên phiếu đục lỗ (punched card) và nhập qua máy đọc phiếu tới máy tính [2]. Sau đó chương trình có thể được ghi vào băng/đĩa từ để sau này nhập nhanh vào máy tính. Ngôn ngữ máy được gọi là *"ngôn ngữ lập trình thế hệ 1"* (1GL, first-generation programming languages) [3].

Sau đó các mã lệnh được thay thế bằng các tên gợi nhớ và trình được lập ở dạng văn bản (text) rồi dịch sang mã máy. Hợp ngữ (assembly languages) ra đời, là *"ngôn ngữ lập trình thế hệ 2"* (2GL, second-generation programming languages). Lập trình thuận lợi hơn, khi dịch có thể liên kết với thư viện chương trình con ở cả dạng macro (đoạn chưa dịch) và lẫn mã đã dịch. Hợp ngữ hiện được dùng là ngôn ngữ bậc thấp (low-level programming languages) để tinh chỉnh ngôn ngữ bậc cao thực hiện truy nhập trực tiếp phần cứng cụ thể trong việc lập trình hệ thống, tạo các hiệu ứng đặc biệt cho chương trình.

Ngôn ngữ bậc cao (high-level programming languages) hay *"ngôn ngữ lập trình thế hệ 3"* (3GL, third-generation programming languages) ra đời vào những năm 1950. Đây là các ngôn ngữ hình thức, dùng trong lập trình máy điện toán và không lệ thuộc vào hệ máy tính cụ thể nào. Nó giải phóng người lập trình ứng dụng làm việc trong hệ điều hành xác định mà không phải quan tâm đến phần cứng cụ thể. Các ngôn ngữ được phát triển liên tục với các dạng và biến thể mới, theo bước phát triển của kỹ thuật điện toán [4].

Đối với ngôn ngữ bậc cao thì định nghĩa *ngôn ngữ lập trình* theo [Loud 94], T.3 là:

> Ngôn ngữ lập trình là một hệ thống được ký hiệu hóa để miêu tả những tính toán (qua máy tính) trong một dạng mà cả con người và máy đều có thể đọc và hiểu được.

Theo định nghĩa ở trên thì một ngôn ngữ lập trình phải thỏa mãn được hai điều kiện cơ bản sau:

1. Dễ hiểu và dễ sử dụng đối với người lập trình, để có thể dùng để giải quyết nhiều bài toán khác nhau.
2. Miêu tả một cách đầy đủ và rõ ràng các tiến trình (tiếng Anh: *process*), để chạy được trên các hệ máy tính khác nhau.

Một tập hợp các chỉ thị được biểu thị qua ngôn ngữ lập trình nhằm mục đích thực hiện các thao tác máy tính nào đó được gọi là một **chương trình**. Khái niệm này còn có những tên khác như **chương trình máy tính** hay **chương trình điện toán**.

Lưu ý: chương trình được viết cho máy vi tính thường được gọi là *phần mềm máy tính*. Ví dụ: *chương trình Microsoft Word* là một cách gọi chung chung; cách gọi *phần mềm Microsoft Word* chỉ rõ hơn nó là một chương trình ứng dụng.

Khái niệm **lập trình** dùng để chỉ quá trình con người tạo ra chương trình máy tính thông qua ngôn ngữ lập trình. Người ta còn gọi đó là **quá trình mã hoá** thông tin tự nhiên thành ngôn ngữ máy. Từ **viết mã** cũng được dùng trong nhiều trường hợp để chỉ cùng một ý.

Như vậy, theo định nghĩa, mỗi ngôn ngữ lập trình cũng chính là một chương trình, nhưng nó có thể được dùng để tạo nên các chương trình khác. Văn bản được viết bằng ngôn ngữ lập trình để tạo nên chương trình được gọi là **mã nguồn**.

Thao tác chuyển đổi từ mã nguồn thành chuỗi các chỉ thị máy tính được thực hiện tương tự như việc chuyển đổi qua lại giữa các ngôn ngữ tự nhiên của con người. Các thao tác này gọi là **biên dịch**, hay ngắn gọn hơn là dịch. Nếu quá trình dịch diễn ra đồng thời với quá trình thực thi, ta gọi đó là **thông dịch**; nếu diễn ra trước, ta gọi đó là **biên dịch**. Phần mềm dịch tương ứng được gọi là phần mềm thông dịch và phần mềm biên dịch.

1. Một **phần mềm thông dịch** là một phần mềm có khả năng đọc, chuyển mã nguồn của một ngôn ngữ và ra lệnh cho máy tính tiến hành các tính toán dựa theo cú pháp của ngôn ngữ.
2. Một **phần mềm biên dịch** hay ngắn gọn hơn **trình biên dịch** là phần mềm có khả năng chuyển mã nguồn của một ngôn ngữ ban đầu sang dạng mã mới thường là một ngôn ngữ cấp thấp hơn.
   Ngôn ngữ cấp thấp nhất là một chuỗi các chỉ thị máy tính mà có thể được thực thi trực tiếp bởi máy tính (thông qua các theo tác trên vùng nhớ). Trước đây, hầu hết các trình biên dịch cũ phải dịch từ mã nguồn sang bộ mã phụ (các tệp có dang *.obj) rồi mới tạo ra tập tin thực thi. Ngày nay, hầu hết các trình biên dịch đều có khả năng dịch mã nguồn trực tiếp thành các tập tin thực thi hay thành các dạng mã khác thấp hơn, tuỳ theo yêu cầu của người lập trình.

Điểm khác nhau giữa thông dịch và biên dịch là: trình thông dịch dịch từng câu lệnh theo yêu cầu thực thi và chương trình đích vừa tạo ra sẽ không được lưu lại; trong khi đó, trình biên dịch sẽ dịch toàn bộ chương trình, cho ra chương trình đích được lưu lại trong máy tính rồi mới thực hiện **chương trình**.

Một chương trình máy tính có thể được thực thi bằng cách biên dịch, thông dịch, hoặc phối hợp cả hai.

Để đạt được yêu cầu về độ chính xác và tính hiệu quả, mã viết ra nhiều khi khó đọc ngay cả với chính người viết ra mã đó, chưa kể tới người khác. Chính vì lý do đó, mọi tài liệu, hướng dẫn lập trình đều khuyên nên thêm các chú giải vào mã nguồn trong quá trình viết. Các chú giải giúp người khác rất nhiều trong việc đọc hiểu mã nguồn; đối với chương trình phức tạp, chú giải là thành phần vô cùng quan trọng trong mã nguồn.

1.2 . Một số khái niệm trong lập trình hướng đối tượng

OOP là chữ viết tắt của *Object Oriented Programming* có nghĩa là **Lập trình hướng đối tượng** được phát minh năm 1965 bởi Ole-Johan Dahl và Kristen Nygaard trong ngôn ngữ Simula. So với phương pháp lập trình cổ điển, thì triết lý chính bên trong loại ngôn ngữ loại này là để tái dụng các khối mã nguồn và cung ứng cho các khối này một khả năng mới: chúng có thể có các hàm (gọi là các phương thức) và các dữ liệu (gọi là thuộc tính) nội tại. Khối mã như vậy được gọi là đối tượng. Các đối tượng thì độc lập với môi trường và có khả năng trả lời với yêu cầu bên ngoài tùy theo thiết kế của người lập trình. Với cách xây dựng này, mỗi đối tượng sẽ tương đương với một chương trình riêng có nhiều đặc tính mới mà quan trọng nhất là tính đa hình, tính đóng, tính trừu tượng và tính thừa kế.

Thừa kế

Đây là đặc tính cho phép tạo các đối tượng mới từ đối tượng ban đầu và lại có thể có thêm những đặc tính riêng mà đối tượng ban đầu không có. Cơ chế này cho phép người lập trình có thể tái sử dụng mã nguồn cũ và phát triển mã nguồn mới bằng cách tạo ra các đối tượng mới thừa kế đối tượng ban đầu.

Đa hình

Tính đa hình được thể hiện trong lập trình hướng đối tượng rất đặc biệt. Người lập trình có thể định nghĩa một thuộc tính (chẳng hạn thông qua tên của các phương thức) cho một loạt các đối tượng gần nhau nhưng khi thi hành thì dùng cùng một tên gọi mà sự thi hành của mỗi đối tượng sẽ tự động xảy ra tương ứng theo từng đối tượng không bị nhầm lẫn.

Ví dụ: khi định nghĩa hai đối tượng "hinh_vuong" và "hinh_tron" thì có một phương thức chung là "chu_vi". Khi gọi phương thức này thì nếu đối tượng là "hinh_vuong" nó sẽ tính theo công thức khác với khi đối tượng là "hinh_tron".

Trừu tượng

Đặc tính này cho phép xác định một đối tượng trừu tượng, nghĩa là đối tượng đó có thể có một số đặc điểm chung cho nhiều đối tượng nhưng bản thân đối tượng này có thể không có các biện pháp thi hành.

Ví dụ: người lập trình có thể định nghĩa đối tượng "hinh" hoàn toàn trừu tượng không có đặc tính mà chỉ có các phương thức được đặt tên chẳng hạn như "chu_vi", "dien_tich". Để thực thi thì người lập trình buộc phải định nghĩa thêm các đối tượng cụ thể chẳng hạn định nghĩa "hinh_tron" và "hinh_vuông" dựa trên đối tượng "hinh" và hai định nghĩa mới này sẽ thừa kế mọi thuộc tính và phương thức của đối tượng "hinh".

Đóng

Tính đóng ở đây dược hiểu là các dữ liệu (thuộc tính) và các hàm (phương thức) bên trong của mỗi đối tượng sẽ không cho phép người gọi dùng hay thay đổi một cách tự do mà chỉ có thể tương tác với đối tượng đó qua các phương thức được người lập trình cho phép. Tính đóng ở đây có thể so sánh với khái niệm "hộp đen", nghĩa là người ta có thể thấy các hành vi của đối tượng tùy theo yêu cầu của môi trường nhưng lại không thể biết được bộ máy bên trong thi hành ra sao.

1.3 Các ưu điểm của lập trình hướng đối tượng bằng Java

1.Đơn giản

Những người thiết kế mong muốn phát triển một ngôn ngữ dễ học và quen thuộc với đa số người lập trình. Java tựa như C++, nhưng đã lược bỏ đi các đặc trưng phức tạp, không cần thiết của C và C++ như: thao tác con trỏ, thao tác định nghĩa chồng toán tử (operator overloading),… Java không sử dụng lệnh "***goto***" cũng như file header (.h). Cấu trúc "***struct***" và "***union***" cũng được loại bỏ khỏi Java. Nên có người bảo Java là "C++--", ngụ ý bảo java là C++ nhưng đã bỏ đi những thứ phức tạp, không cần thiết.

2. Hướng đối tượng

Có thể nói java là ngôn ngữ lập trình hoàn toàn hướng đối tượng, tất cả trong java đều là sự vật, đâu đâu cũng là sự vật.

3. Độc lập với hệ nền

Mục tiêu chính của các nhà thiết kế java là độc lập với hệ nền hay còn gọi là độc lập phần cứng và hệ điều hành. Đây là khả năng một chương trình được viết tại một máy nhưng có thể chạy được bất kỳ đâu

Tính độc lập với phần cứng được hiểu theo nghĩa một chương trình Java nếu chạy đúng trên phần cứng của một họ máy nào đó thì nó cũng chạy đúng trên tất cả các họ máy khác. Một chương trình chỉ chạy đúng trên một số họ máy cụ thể được gọi là phụ thuộc vào phần cứng.

Tính độc lập với hệ điều hành được hiểu theo nghĩa một chương trình Java có thể chạy được trên tất cả các hệ điều hành. Một chương trình chỉ chạy được trên một số hệ điều hành được gọi là phụ thuộc vào hệ điều hành.

Các chương trình viết bằng java có thể chạy trên hầu hết các hệ nền mà không cần phải thay đổi gì, điều này đã được những người lập trình đặt cho nó một khẩu hiệu ***'viết một lần, chạy mọi nơi'***, điều này là không thể có với các ngôn ngữ lập trình khác.

Đối với các chương trình viết bằng C, C++ hoặc một ngôn ngữ nào khác, trình biên dịch sẽ chuyển tập lệnh thành mã máy (machine code), hay lệnh của bộ vi xử lý. Những lệnh này phụ thuộc vào CPU hiện tại trên máy bạn. Nên khi muốn chạy trên loại CPU khác, chúng ta phải biên dịch lại chương trình.

4. Mạnh mẽ Java là ngôn ngữ yêu cầu chặt chẽ về kiểu dữ liệu, việc ép kiểu tự động bừa bãi của C, C++ nay được hạn chế trong Java, điều này làm chương trình rõ ràng, sáng sủa, ít lỗi hơn.Java kiểm tra lúc biên dịch và cả trong thời gian thông dịch vì vậy Java loại bỏ một một số loại lỗi lập trình nhất định.Java không sử dụng con trỏ và các phép toán con trỏ. Java kiểm tra tất cả các truy nhập đến mảng, chuỗi khi thực thi để đảm bảo rằng các truy nhập đó không ra ngoài giới hạn kích thước.

Trong các môi trường lập trình truyền thống, lập trình viên phải tự mình cấp phát bộ nhớ. Trước khi chương trình kết thúc thì phải tự giải phóng bộ nhớ đã cấp. Vấn đề nảy sinh khi lập trình viên quên giải phóng bộ nhớ đã xin cấp trước đó. Trong chương trình Java, lập trình viên không phải bận tâm đến việc cấp phát bộ nhớ. Qúa trình cấp phát, giải phóng được thực hiện tự động, nhờ dịch vụ thu nhặt những đối tượng không còn sử dụng nữa (garbage collection).

Cơ chế bẫy lỗi của Java giúp đơn giản hóa qúa trình xử lý lỗi và hồi phục sau lỗi.

5. Hỗ trợ lập trình đa tuyến

Đây là tính năng cho phép viết một chương trình có nhiều đoạn mã lệnh được chạy song song với nhau. Với java ta có thể viết các chương trình có khả năng chạy song song một cách dễ dàng, hơn thế nữa việc đồng bộ tài nguyên dùng chung trong Java cũng rất đơng giản. Điều này là không thể có đối với một số ngôn ngữ lập trình khác như C/C++, pascal …

6. Phân tán

Java hỗ trợ đầy đủ các mô hình tính toán phân tán: mô hình client/server, gọi thủ tục từ xa…

7. Hỗ trợ internet

Mục tiêu quan trọng của các nhà thiết kế java là tạo điều kiện cho các nhà phát triển ứng dụng có thể viết các chương trình ứng dụng internet và web một cách dễ dàng, với java ta có thể viết các chương trình sử dụng các giao thức TCP, UDP một cách dễ dàng, về lập trình web phía máy khách java có công nghệ java applet, về lập trình web phía máy khách java có công nghệ servlet/JSP, về lập trình phân tán java có công nghệ RMI, CORBA, EJB, Web Service.

8. Thông dịch

Các chương trình java cần được thông dịch trước khi chạy, một chương trình java được biên dịch thành mã byte code mã độc lập với hệ nền, chương trình thông dịch java

sẽ ánh xạ mã byte code này lên mỗi nền cụ thể, điều này khiến java chậm chạp đi phần nào.

Bài 3: Lớp và đối tượng (I)

3.1. Khai báo lớp

1.1. Một lớp được định nghĩa theo mẫu sau:

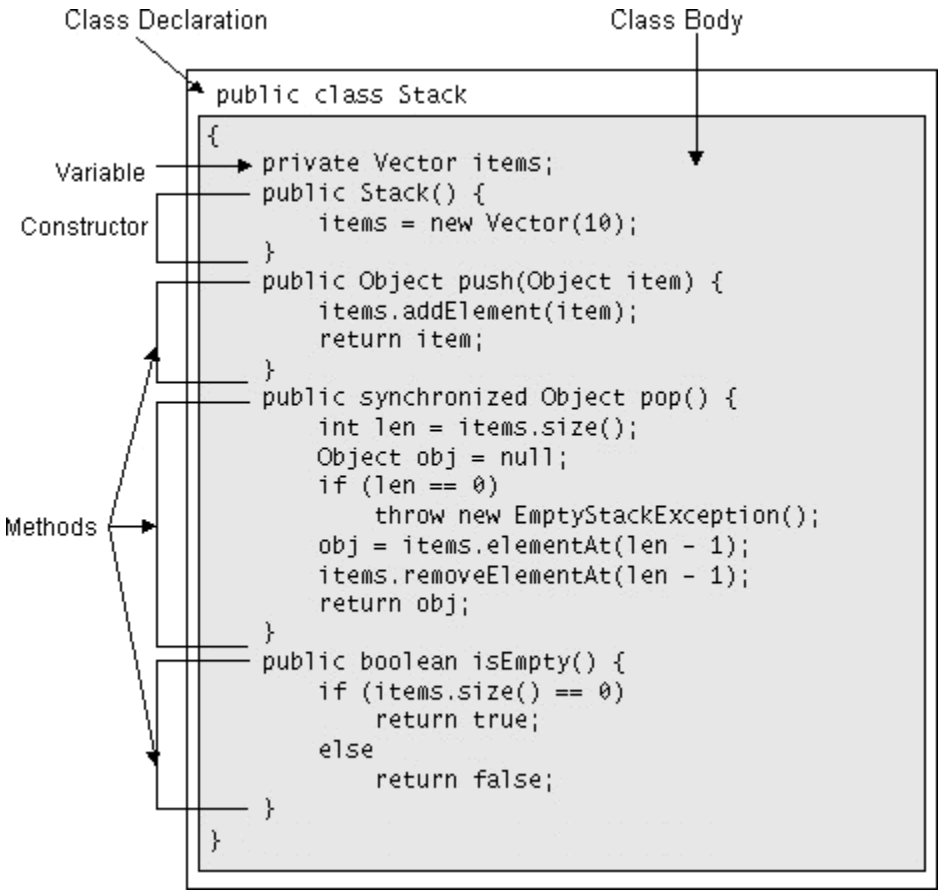
[pbulic][final][abstract] class <tên_lớp>{

// khai báo các thuộc tính

// khai báo các phương thức

}

sau đâu là ví dụ đơn giản định nghĩa lớp ngăn xếp:



Tổng quát: một lớp được khai báo dạng sau:

[public][<abstract><final>][class <Tên lớp>

[***extends*** <Tên lớp cha>] [***implements*** <Tên giao diện>] {

<Các thành phần của lớp, bao gồm: thuộc tính và phương thức>

}

Trong đó:

1) bởi mặc định một lớp chỉ có thể sử dụng bởi một lớp khác trong cùng một gói với lớp đó, nếu muốn gói khác có thể sử dụng lớp này thì lớp này phải được khai báo là lớp ***public***.
2) ***abstract*** là bổ từ cho java biết đây là một lớp trừu tượng, do vậy ta không thể tạo ra một thể hiện của lớp này
3) ***final*** là bổ từ cho java biết đây là một lớp không thể kế thừa
4) ***class*** là từ khoá cho chương trình biết ta đang khai báo một lớp, lớp này có tên là NameOfClass
5) ***extends*** là từ khoá cho java biết lớp này này được kế thừa từ lớp super
6) ***implements*** là từ khoá cho java biết lớp này sẽ triển khai giao diện Interfaces, đây là một dạng tương tự như kế thừa bội của java.

Chú ý:

1) Thuộc tính của lớp là một biến có kiểu dữ liệu bất kỳ, nó có thể lại là một biến có kiểu là chính lớp đó
2) Khi khai báo các thành phần của lớp (thuộc tính và phương thức) có thể dùng một trong các từ khoá ***private, public, protected*** để giứo hạn sự truy cập đến thành phần đó.

   – các thành phần private chỉ có thể sử dụng được ở bên trong lớp, ta không thể truy cập vào các thành phần ***private*** từ bên ngoài lớp

– Các thành phần ***public*** có thể truy cập được cả bên trong lớp lẫn bên ngoài lớp.
– các thành phần ***protected*** tương tự như các thành phần ***private***, nhưng có thể truy cập được từ bất cứ lớp con nào kế thừa từ nó.
– Nếu một thành phần của lớp khi khai báo mà không sử dụng một trong 3 bổ từ ***protected, private, public*** thì sự truy cập là bạn bè, tức là thành phần này có thể truy cập được từ bất cứ lớp nào trong cùng gói với lớp đó.

3) Các thuộc tính nên để mức truy cập ***private*** để đảm bảo tính dấu kín và lúc đó để bên ngoài phạm vi của lớp có thể truy cập được đến thành phần ***private*** này ta phải tạo ra các phương thức phương thức get và set.
4) Các phương thức thường khai báo là public, để chúng có thể truy cập từ bất cứ đâu.
5) Trong một tệp chương trình (hay còn gọi là một đơn vị biên dịch) chỉ có một lớp được khai báo là public, và tên lớp public này phải trùng với tên của tệp kể cả chữ hoa, chữ thường

3.2 Khai báo thuộc tính

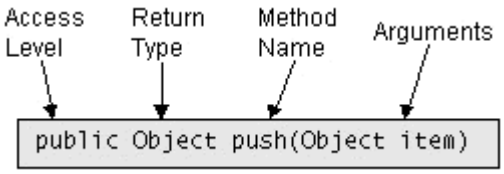
Trở lại lớp Stack

```
public class Stack {
private Vector items;
// a method with same name as a member variable
public Vector items() {
...
}
}
```

Trong lớp Stack trên ta có một thuộc tính được định nghĩa như sau:
private Vector items;

Việc khai báo như trên được gọi là khai báo thuộc tính hay còn gọi là biến thành viên lớp

Tổng quát việc khai báo một thuộc tính được viết theo mẫu sau:



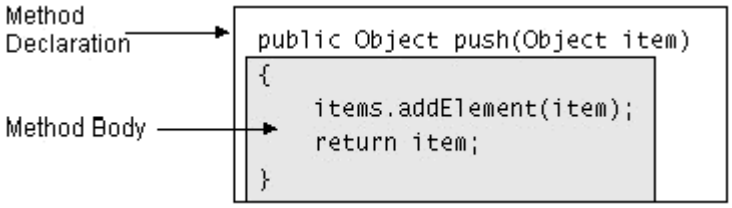
Trong đó:

- *accessLevel có thể là một trong các từ public, private, protected hoặc có thể bỏ* trống, ý nghĩa của các bổ từ này được mô tả ở phần trên
- - *static là từ khoá báo rằng đây là một thuộc tính lớp, nó là một thuộc tính sử* dụng chung cho cả lớp, nó không là của riêng một đối tượng nào.
- - *transient và volatile chưa được dùng*
- - *type là một kiểu dữ liệu nào đó*
- *name là tên của thuộc tính*

Chú ý: Ta phải phân biệt được việc khai báo như thế nào là khai báo thuộc tính, khai báo thế nào là khai báo biến thông thường? Câu trả lời là tất cả các khai báo bên trong thân của một lớp và bên ngoài tất cả các phương thức và hàm tạo thì đó là khai báo thuộc tính, khai báo ở những chỗ khác sẽ cho ta biến.

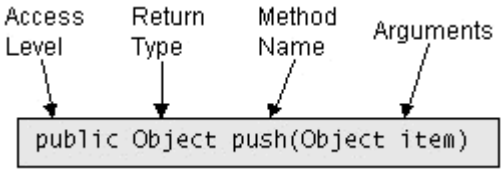
- Khai báo phương thức

Trong lớp Stack trên ta có phương thức push dùng để đẩy một đối tượng vào đỉnh ngăn xếp, nó được định nghĩa như sau:



Cũng giống như một lớp, một phương thức cũng gồm có 2 phần: phần khai báo và *phần thân*

- Phần khai báo gồm có những phần sau(chi tiết của khai báo được mô tả sau):



- Phần thân của phương thức gồm các lệnh để mô tả hành vi của phương thức, các hành vi này được viết bằng các lệnh của java.

Bài 5: Lớp và đối tượng (II)

5.1. Chi tiết về khai báo một phương thức

1. Tổng quát một phương thức được khai báo như sau:

accessLevel	//mô tả mức độ truy cập đến phương thức
static	//đây là phương thức lớp
abstract	//đây là phương thức không có cài đặt
final	//phương thức này không thể ghi đè
native	//phương thức này được viết trong một ngôn ngữ khác
synchronized	//đây là phương thức đồng bộ
returnType	//giá trị trả về của phương thức
MethodName	//tên của phương thức
throws exception	//khai báo các ngoại lệ có thể được nem ra từ phương thức

Trong đó:

- accessLevel có thể là một trong các từ khoá public, private, protected hoặc bỏ trống, ý nghĩa của các bổ từ này được mô tả trong phần khai báo lớp

- static là từ khoá báo cho java biết đây là một phương thức lớp

- abstract từ khoá cho biết đây là một lớp trừu tượng, nó không có cài đặt.

- final đây là từ khoá báo cho java biết đây là phương thức không thể ghi đè từ lớp con

- native đây là từ khoá báo cho java biết phương thức này được viết bằng một ngôn ngữ lập trình nào đó không phải là java (thường được viết bằng C/C++)

- synchronized đây là một phương thức đồng bộ, nó rất hữu ích khi nhiều phương thức cùng truy cập đồng thời vào tài nguyên miền găng

- returnType là một kiểu dữ liệu, đây là kiểu trả về của phương thức, khi phương thức không trả về dữ liệu thì phải dùng từ khoá void

- MethodName là tên của phương thức, tên của phương thức được đặt theo quy tắc đặt tên của java

- throws là từ khoá dùng để khai báo các ngoại lệ có thể được ném ra từ phương thức, theo sau từ khoá này là danh sách các ngoại lệ có thể được phương thức này ném ra

Chú ý:

1) Nếu trong lớp có ít nhất một phương thức trừu tượng thì lớp đó phải là lớp trừu tượng
2) không có thuộc tính trừu tượng
3) ta không thể tạo đối tượng của lớp trừu tượng

4) khác với ngôn ngữ C/C++, java bắt buộc bạn phải khai báo giá trị trả về cho phương thức, nếu phương thức không trả về dữ liệu thi dùng từ khoá void (trong C/C++ khi ta không khai báo giá trị trả về thì mặc định giá trị trả về là int)

2. Nhận giá trị trả về từ phương thức

Ta khai báo kiểu giá trị trả về từ lúc ta khai báo phương thức, bên trong thân của phương thức ta phải sử dụng phát biểu return value; để nhận về kết quả, nếu hàm được khai báo kiểu void thì ta chỉ sử dụng phát biểu return; mệnh đề return đôi khi còn được dùng để kết thúc một phương thức.

3. Truyền tham số cho phương thức

Khi ta viết các phương thức, một số phương thức yêu cầu phải có một số tham số, các tham số của một phương thức được khai báo trong lời khai báo phương thức, chúng phải được khai báo chi tiết có bao nhiêu tham số, mỗi tham số cần phải cung cấp cho chúng một cái tên và kiểu dữ liệu của chúng.

Ví dụ: ta có một phương thức dùng để tính tổng của hai số, phương thức này được khai báo như sau:

```
public double tongHaiSo(double a, double b){

return (a + b);

}
```

1. Kiểu tham số

Trong java ta có thể truyền vào phương thức một tham số có kiểu bất kỳ, từ kiểu dữ liệu nguyên thuỷ cho đến tham chiếu đối tượng.

2. Tên tham số

Khi bạn khai báo một tham số để truyền vào phương thức thì bạn phải cung cấp cho nó một cái tên, tên nay được sử dụng bên trong thân của phương thức để tham chiếu đến tham số được truyền vào.

Chú ý: tên của tham số có thể trùng với tên của thuộc tính, khi đó tên của tham số sẽ "che" đi tên của phương thức, bởi vậy bên trong thân của phương thức mà có tham số có tên trùng với tên của thuộc tính, thì khi nhắc đến cái tên đó có nghĩa là nhắc đến tham số.

3. Truyền tham số theo trị

Khi gọi một phương thức mà tham số của phương thức có kiểu nguyên thuỷ, thì bản sao giá trị của tham số thực sự sẽ được chuyển đến phương thức, đây là đặc tính truyền theo trị (pass- by – value), nghĩa là phương thức không thể thay đổi giá trị của các tham số truyền vào.

Ta kiểm tra điều này qua ví dụ sau:

```
public class TestPassByValue {

public static void test(int t) {

        t++;

        System.out.println("Gia tri của t bi?n trong ham sau khi tang len 1 la " + t);

}

public static void main(String[] args) {

        int t = 10;

        System.out.println("Gia tri của t tru?c khi gọi ham = " + t);

        test(t);

        System.out.println("Gia tri của t truoc khi gọi ham = " + t);

}


}
```

ta se nhận được kết quả ra như sau:

Gia tri của t truoc khi gọi ham = 10

Gia tri của t bên trong ham sau khi tang len 1 la 11

Gia tri của t truoc khi gọi ham = 10

4. Thân của phương thức

Trong ví dụ sau thân của phương thức isEmpty và phương thức pop được in đậm và có mầu đỏ

```
class Stack {

static final int STACK_EMPTY = -1;

Object[] stackelements;

int topelement = STACK_EMPTY;

...

boolean isEmpty() {

        if (topelement == STACK_EMPTY)

        return true;

        else

        return false;

}

Object pop() {

        if (topelement == STACK_EMPTY)

        return null;
```

```
        else {
        return stackelements[topelement--];
        }
}
```

5.2. Từ khoá this

Thông thường bên trong thân của một phương thức ta có thể tham chiếu đến các thuộc tính của đối tượng đó, tuy nhiên trong một số tình huống đặc biệt như tên của tham số trùng với tên của thuộc tính, lúc đó để chỉ các thành viên của đối tượng đó ta dùng từ khoá this, từ khoá this dùng để chỉ đối tượng này.

Ví dụ sau chỉ ra cho ta thấy trong tình huống này bắt buộc phải dùng từ khoá this vì tên tham số của phương thức tạo dựng lại trùng với tên của thuộc tính

```
class HSBColor {
        int hue, saturation, brightness;
        HSBColor (int hue, int saturation, int brightness) {
        this.hue = hue;
        this.saturation = saturation;
        this.brightness = brightness;
}
```

5.3. Từ khoá super

Khi một lớp được kế thừa từ lớp cha trong cả lớp cha và lớp con đều có một phương thức trùng tên nhau, thế thì làm thế nào có thể gọi phương thức trùng tên đó của lớp cha, java cung cấp cho ta từ khoá *super dùng để chỉ đối tượng của lớp cha*

Ta xét ví dụ sau

```
class ASillyClass {
   boolean aVariable;
   void aMethod() {
            aVariable = true;
   }
}

class ASillierClass extends ASillyClass {
   boolean aVariable;
   void aMethod() {
        aVariable = false;
        super.aMethod();
        System.out.println(aVariable);
        System.out.println(super.aVariable);
   }
}
```

trong ví dụ trên ta thấy trong lớp cha có phương thức tên là aMethod trong lớp con cũng có một phương thức cùng tên, ta còn thấy cả hai lớp này cùng có một thuộc tính tên aVariable để có thể truy cập vào các thành viên của lớp cha ta phải dùng từ khoá super.

Chú ý: ta không thể dùng nhiều từ khoá này để chỉ lớp ông, lớp cụ… chẳng hạn viết như sau là sai: super.super.add(1,4);

5.4. Sử dụng lớp

Sau khi khao một một lớp ta có thể xem lớp như là một kiểu dữ liệu, nên ta có thể tạo ra các biến, mảng các đối tượng, việc khai báo một biến, mảng các đối tượng cũng tương tự như khai báo một biến, mảng của kiểu dữ liệu nguyên thuỷ

Việc khai báo một biến, mảng được khai báo theo mẫu sau:

Tên_Lớp tên_biến;

Tên_Lớp tên_mang[kích thước mảng];

Tên_Lớp[kích thước mảng] tên_mang;

Về bản chất mỗi đối tượng trong java là một con trỏ tới một vùng nhớ, vùng nhớ này chính là vùng nhớ dùng để lưu trữ các thuộc tính, vùng nhớ dành cho con trỏ này thì được cấp phát trên stack, còn vùng nhớ dành cho các thuộc tính của đối tượng này thì được cấp phát trên heap.

5.5. Điều khiển việc truy cập đến các thành viên của một lớp

Khi xây dựng một lớp ta có thể hạn chế sự truy cập đến các thành viên của lớp, từ một đối tượng khác.

Ta tóm tắt qua bảng sau:

Từ khoá	Truy cập trong chính lớp	Truy cập trong lớp con cùng	Truy cập trong lớp con khác	Truy cập trong lớp khác cùng gói	Truy cập trong lớp khác khác gói

	đó	gói	gói		
private	X	-	-	-	-
protected	X	X	X	X	-
public	X	X	X	X	X
default	X	X	-	X	-

Trong bảng trên thì X thể hiện cho sự truy cập hợp lệ còn – thể hiện không thể truy cập vào thành phần này.

1. Các thành phần private

Các thành viên private chỉ có thể sử dụng bên trong lớp, ta không thể truy cập các thành viên private từ bên ngoài lớp này.

Ví dụ

```
class Alpha
{
private int iamprivate;
private void privateMethod()
{
System.out.println("privateMethod");
}
}

class Beta {
void accessMethod()
{
Alpha a = new Alpha();
```

```
a.iamprivate = 10;// không hợp lệ
a.privateMethod();// không hợp lệ
}
}
```

2. Các thành phần protected

Các thành viên protected sẽ được thảo luận trong chương sau

3. Các thành phần public

Các thành viên public có thể truy cập từ bất cứ đâu, ta se xem ví dụ sau:

```
package Greek;
public class Alpha {
    public int iampublic;
    public void publicMethod() {
    System.out.println("publicMethod");
}
}
```

```
package Roman;
import Greek.*;
class Beta {
    void accessMethod() {
    Alpha a = new Alpha();
```

```
        a.iampublic = 10;// hợp lệ
        a.publicMethod();// hợp lệ
    }
    }
```

4. Các thành phần có mức truy xuất gói

khi ta khai báo các thành viên mà không sử dụng một trong các từ public, private, protected thì java mặc định thành viên đó có mức truy cập gói.

Ví dụ

```
package Greek;
class Alpha {
   int iampackage;
   void packageMethod() {
   System.out.println("packageMethod");
}
}
```

Bài 7: Bài tập và thảo luận về Lớp và Đối tượng

7.1. Lớp và đối tượng có ưu điểm gì?

OOP có 4 tính chất đặc thù chính, các ngôn ngữ OOP nói chung đều có cách để diễn tả:

- Tính đóng gói: Có thể gói dữ liệu (data, ~ biến, trạng thái) và mã chương trình (code, ~ phương thức) thành một cục gọi là lớp (class) để dễ quản lí. Trong cục này thường data rất rối rắm, không tiện cho người không có trách nhiệm truy cập trực tiếp, nên thường ta sẽ che dấu data đi, chỉ để lòi phương thức ra ngoài. Ví dụ hàng xóm sang mượn búa, thay vì bảo hàng xóm cứ tự nhiên vào lục lọi, ta sẽ bảo: "Ấy bác ngồi chơi để tôi bảo cháu lấy cho". Ngôn ngữ Ruby "phát xít" đến nỗi dấu tiệt data, cấm không cho truy cập từ bên ngoài. Ngoài ra, các lớp liên quan đến nhau có thể được gom chung lại thành package (tùy ngôn ngữ mà còn gọi là module, namespace v.v.).
- Tính trừu tượng: Có câu "program to interfaces, not to concrete implementations". Nghĩa là khi viết chương trình theo phong cách hướng đối tượng, khi thiết kế các đối tượng, ta cần rút tỉa ra những đặc trưng của chúng, rồi trừu tượng hóa thành các interface, và thiết kế xem chúng sẽ tương tác với nhau như thế nào. Nói cách khác, chúng ta định ra các interface và các contract mà chúng cần thỏa mãn.
- Tính thừa kế: Lớp cha có thể chia sẻ dữ liệu và phương thức cho các lớp con, các lớp con khỏi phải định nghĩa lại những logic chung, giúp chương trình ngắn gọn. Nếu lớp cha là interface, thì lớp con sẽ di truyền những contract trừu tượng từ lớp cha.
- Tính đa hình: Đối tượng có thể thay đổi kiểu (biến hình). (1) Với các ngôn ngữ OOP có kiểu, có thể mượn phát biểu của C++ "con trỏ kiểu lớp cha có thể dùng để trỏ đến đối tượng kiểu lớp con". Như vậy khi khai báo chỉ cần khai báo p có kiểu lớp cha, còn sau đó nó trỏ đến đâu thì kệ cha con nó: nếu cha và con cùng có phương thức m, thì từ p cứ lôi m ra gọi thì chắc chắn gọi được, không cần biết hiện tại p đang trỏ đến cha hay con. Khi lớp B thừa kế từ lớp A, thì đối tượng của lớp B có thể coi là đối tượng của lớp A, vì B chứa nhiều thứ thừa kế từ A. (2) Với ngôn ngữ OOP không có kiểu như Ruby, có thể mượn phát biểu của phương pháp xác định kiểu kiểu con vịt: "nếu p đi như vịt nói như vịt, thì cứ coi nó là vịt". Như vậy nếu lớp C có phương thức m, mà có thể gọi phương thức m từ đối tượng p bất kì nào đó, thì cứ coi p có kiểu là C.

Để dễ nhớ, có thể chia 4 đặc thù làm 2 nhóm:

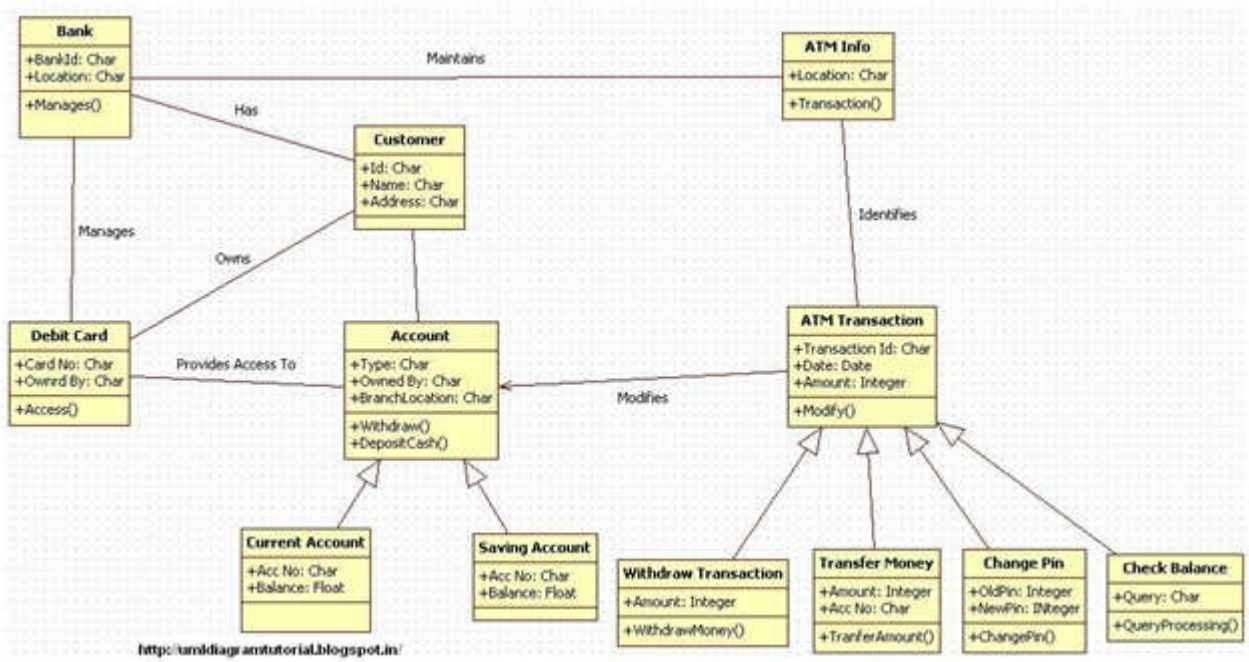
1. Nhóm 1: tính chất 1. Tính đóng gói là tính dễ nhận thấy nhất nếu bạn bắt đầu học OOP sau khi đã học qua những ngôn ngữ thủ tục như C và Pascal (thường trường phổ thông ở Việt Nam đều dạy).

2. Nhóm 2: tính chất 2, 3, và 4 đi một dây với nhau.

7.2. Cách xây dựng lớp và đối tượng

1. Các thành phần trong bản vẽ Class

Trước tiên, chúng ta xem một bản vẽ Class.

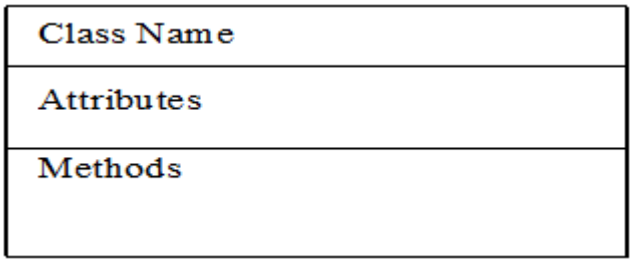


Hình 1. Ví dụ về Class Diagram của ATM

Ví dụ trên là Class Diagram của ứng dụng ATM. Tiếp theo chúng ta sẽ bàn kỹ về các thành phần của bản vẽ này và lấy ứng dụng về ATM ở trên để minh họa.

Classes (Các lớp)

Class là thành phần chính của bản vẽ Class Diagram. Class mô tả về một nhóm đối tượng có cùng tính chất, hành động trong hệ thống. Ví dụ mô tả về khách hàng chúng ta dùng lớp "Customer". Class được mô tả gồm tên Class, thuộc tính và phương thức.



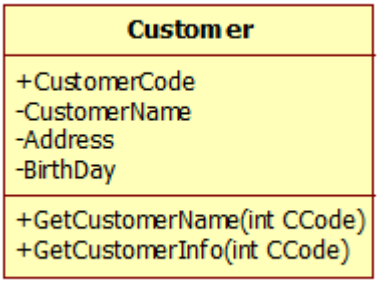
Hình 2. Ký hiệu về Class

Trong đó,

– Class Name: là tên của lớp.

– Attributes (thuộc tính): mô tả tính chất của các đối tượng. Ví dụ như khách hàng có Mã khách hàng, Tên khách hàng, Địa chỉ, Ngày sinh v.v…

– Method (Phương thức): chỉ các hành động mà đối tượng này có thể thực hiện trong hệ thống. Nó thể hiện hành vi của các đối tượng do lớp này tạo ra.



Hình 3. Ví dụ về một Class

Một số loại Class đặc biệt như Abstract Class (lớp không tạo ra đối tượng), Interface (lớp khai báo mà không cài đặt) v.v.. chúng ta xem thêm các tài liệu về lập trình hướng đối tượng để hiểu rõ hơn các vấn đề này.

2. Relationship (Quan hệ)

Relationship thể hiện mối quan hệ giữa các Class với nhau. Trong UML 2.0 có các quan hệ thường sử dụng như sau:

– Association

– Aggregation

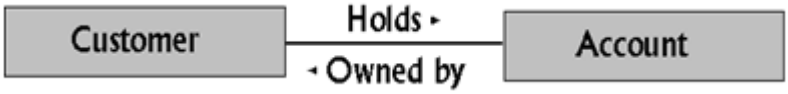
– Composition

– Generalization

Chúng ta sẽ lần lượt tìm hiểu về chúng.

+ ***Association***

Association là quan hệ giữa hai lớp với nhau, thể hiện chúng có liên quan với nhau. Association thể hiện qua các quan hệ như "has: có", "Own: sở hữu" v.v…

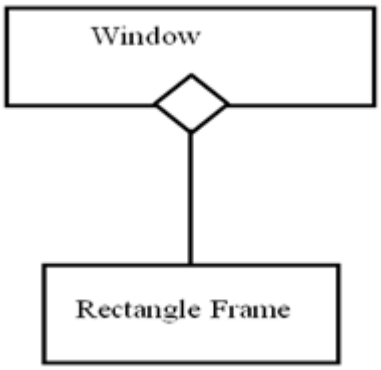


Hình 4. Ví dụ về Association

Ví dụ quan hệ trên thể hiện Khách hàng nắm giữ Tài khoản và Tài khoản được sở hữu bởi Khách hàng.

+ ***Aggregation***

Aggregation là một loại của quan hệ Association nhưng mạnh hơn. Nó có thể cùng thời gian sống (cùng sinh ra hoặc cùng chết đi)

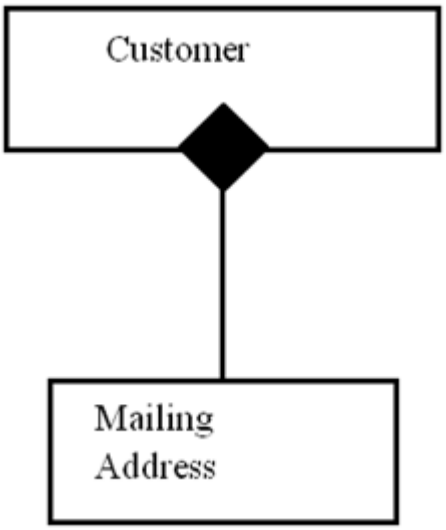


Hình 5. Ví dụ về Aggregation

Ví dụ quan hệ trên thể hiện lớp Window(cửa sổ) được lắp trên Khung cửa hình chữ nhật. Nó có thể cùng sinh ra cùng lúc.

+ ***Composition***

Composition là một loại mạnh hơn của Aggregation thể hiện quan hệ class này là một phần của class kia nên dẫn đến cùng tạo ra hoặc cùng chết đi.

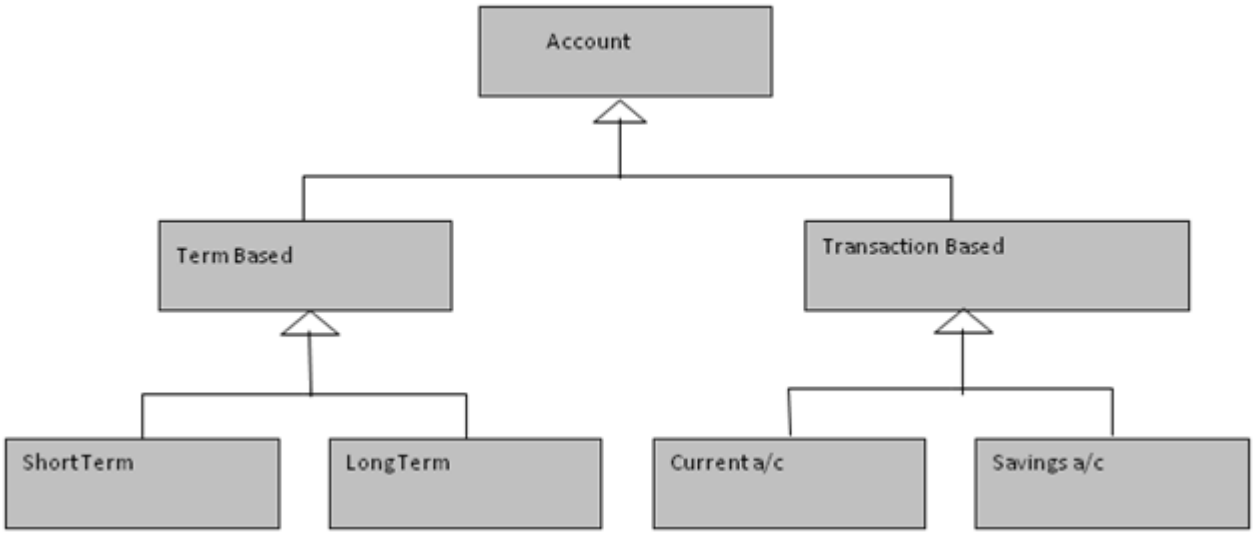


Hình 5. Ví dụ về Composition

Ví dụ trên class Mailing Address là một phần của class Customer nên chỉ khi nào có đối tượng Customer thì mới phát sinh đối tượng Mailing Address.

+Generalization

Generalization là quan hệ thừa kế được sử dụng rộng rãi trong lập trình hướng đối tượng.



Hình 6. Ví dụ về Genelization

Các lớp ở cuối cùng như Short Term, Long Term, Curent a/c, Savings a/c gọi là các lớp cụ thể (concrete Class). Chúng có thể tạo ra đối tượng và các đối tượng này thừa kế toàn bộ các thuộc tính, phương thức của các lớp trên.

Các lớp trên như Account, Term Based, Transaction Based là những lớp trừu tượng (Abstract Class), những lớp này không tạo ra đối tượng.

Ngoài ra, còn một số quan hệ như khác như dependence, realization nhưng ít được sử dụng nên chúng ta không bàn ở đây.

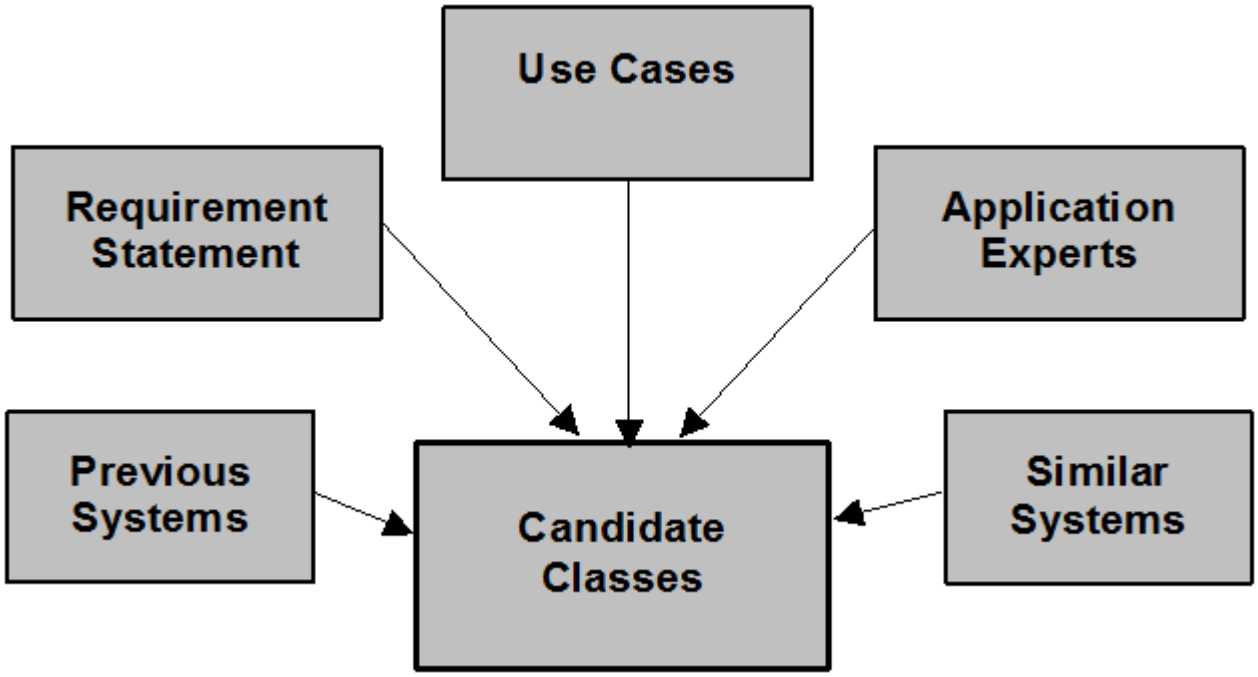
3. Cách xây dựng bản vẽ Class

Class Diagram là bản vẽ khó xây dựng nhất so với các bản vẽ khác trong OOAD và UML. Bạn phải hiểu được hệ thống một cách rõ ràng và có kinh nghiệm về lập trình hướng đối tượng mới có thể xây dựng thành công bản vẽ này.

Thực hiện theo các bước sau đây để xây dựng Class Diagram.

Bước 1: Tìm các Classes dự kiến

Entity Classes(các lớp thực thể) là các thực thể có thật và hoạt động trong hệ thống, bạn dựa vào các nguồn sau để xác định chúng.



Hình 7. Các nguồn thông tin có thể tìm Class dự kiến

– **Requirement statement**: Các yêu cầu. Chúng ta phân tích các danh từ trong các yêu cầu để tìm ra các thực thể.

– **Use Cases**: Phân tích các Use Case sẽ cung cấp thêm các Classes dự kiến.

– **Previous và Similar System:** có thể sẽ cung cấp thêm cho bạn các lớp dự kiến.

– **Application Experts:** các chuyên gia ứng dụng cũng có thể giúp bạn.

Xem xét, ví dụ ATM ở trên chúng ta có thể thấy các đối tượng là Entity Class như sau:

– **Customers**: khách hàng giao dịch là một thực thể có thật và quản lý trong hệ thống.

– **Accounts**: Tài khoản của khách hàng cũng là một đối tượng thực tế.

– **ATM Cards**: Thẻ dùng để truy cập ATM cũng được quản lý trong hệ thống.

– **ATM Transactions**: Các giao dịch được lưu giữ lại, nó cũng là một đối tượng có thật.

– **Banks**: Thông tin ngân hàng bạn đang giao dịch, nếu có nhiều nhà Bank tham gia vào hệ thống bạn phải quản lý nó. Lúc đó Bank trở thành đối tượng bạn phải quản lý.

– **ATM**: Thông tin ATM bạn sẽ giao dịch. Nó cũng được quản lý tương tự như Banks.

Lưu ý: Chỉ các thực thể bên trong hệ thống được xem xét, các thực thế bên ngoài hệ thống không được xem xét. Ví dụ Customers là những người khách hàng được quản lý trong hệ thống chứ không phải người dùng máy ATM bên ngoài. Bạn phải lưu ý điều này để phân biệt Class và Actor.

Bước 2: Tìm các thuộc tính và phương thức cho lớp

– **Tìm thuộc tính**: phân tích thông tin từ các form mẫu có sẵn, bạn sẽ tìm ra thuộc tính cho các đối tượng của lớp. Ví dụ các thuộc tính của lớp Customer sẽ thể hiện trên Form đăng ký thông tin khách hàng.

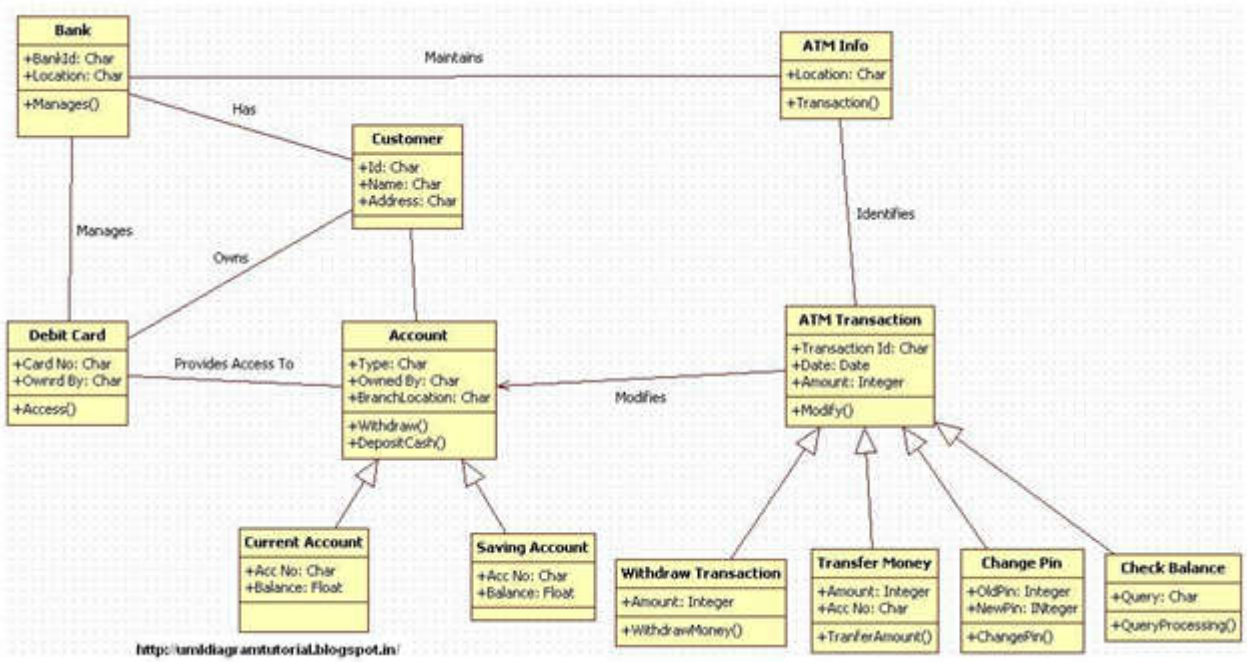
– **Tìm phương thức**: phương thức là các hoạt động mà các đối tượng của lớp này có thể thực hiện. Chúng ta sẽ bổ sung phương thức đầy đủ cho các lớp khi phân tích Sequence Diagram sau này.

Bước 3: Xây dựng các quan hệ giữa các lớp và phát hiện các lớp phát sinh

– Phân tích các quan hệ giữa các lớp và định nghĩa các lớp phát sinh do các quan hệ sinh ra. Chúng ta phân tích các thực thể ở trên và nhận thấy.

- Lớp ***Accounts*** có thể chia thành nhiều loại tài khoản như ***Current Accounts*** và ***Saving Accounts*** và có quan hệ thừa kế với nhau.
- Lớp ***ATM Transactions*** cũng có thể chia thành nhiều loại giao dịch như ***Deposit***, ***Withdraw, Transfer*** v.v.. và chúng cũng có quan hệ thừa kế với nhau.

– Tách chúng ta và vẽ chúng lên bản vẽ chúng ta sẽ có Class Diagram cho hệ thống ATM như sau:



Hình 8. Ví dụ về Class Diagram cho hệ thống ATM

4. Đặc tả Class

Nhìn vào Class Diagram chúng ta có thể thấy cấu trúc của hệ thống gồm những lớp nào nhưng để cài đặt chúng, chúng ta phải đặc tả chi tiết hơn nữa. Trong đó, cần mô tả:

– Các thuộc tính: Tên, kiểu dữ liệu, kích thước

– Các phương thức:

- + Tên
- + Mô tả
- + Tham số đầu vào: Tên, kiểu dữ liệu, kích thươcs
- + Kết quả đầu ra: Tên, kiểu dữ liệu, kích thước
- + Luồng xử lý
- + Điều kiện bắt đầu
- + Điều kiện kết thúc

Tuy nhiên, việc này cũng mất khá nhiều thời gian. Nếu phát triển theo mô hình Agile thì bạn không phải làm việc này mà các thành viên phát triển phải nắm điều này để cài đặt.

5. Sử dụng bản vẽ Class

Có thể tóm tắt một số ứng dụng của bản vẽ Class Diagram như sau:

– Hiểu cấu trúc của hệ thống

– Thiết kế hệ thống

– Sử dụng để phân tích chi tiết các chức năng (Sequence Diagram, State Diagram v.v…)

– Sử dụng để cài đặt (coding)

6. Kết luận

Như vậy, chúng ta đã tìm hiểu xong về Class Diagram, các bạn cần thực hành nhiều để hiểu về bản vẽ quan trọng này.

Để giúp các bạn nắm rõ hơn về Class Diagram, trong bài tiếp theo chúng ta sẽ thực hành xây dựng Class Diagram cho hệ thống eCommerce đã mô tả trong Case Study ở bài 3.

Bài 8: Gói trong java

8.1. Vai trò của gói (package) trong lập trình

Một package trong Java là một nhóm các kiểu lớp, Interface và package con tương tự nhau. Package trong Java có thể được phân loại thành: Package đã xây dựng sẵn và package do người dùng định nghĩa. Có nhiều package đã xây dựng sẵn như java, lang, awt, javax, swing, net, io, util, sql, … Chương này chúng ta sẽ tìm hiểu cách tạo và sử dụng các package do người dùng tự định nghĩa.

Gói (package) được sử dụng trong Java để ngăn cản việc xung đột đặt tên, điều khiên truy cập, giúp việc tìm kiếm/lưu trữ và sử dụng lớp, interface, enumeration, annotation dễ dàng hơn.

Một package có thể được định nghĩa như một nhóm các kiểu có liên quan đến nhau (lớp, interface, enumeration và annotation) cung cấp việc bảo vệ truy cập và quản lý tên.

Một vài package có sẵn trong Java như:

java.lang - Các lớp cơ bản

java.io - Các lớp input và output cơ bản

Lập trình viên có thể định nghĩa gói riêng để bao bọc một nhóm các class/interface. Trong thực tế, việc nhóm các class liên quan đến nhau giúp cho lập trình viên dễ dàng xác định class, interface, enumeration, annotation liên quan đến nhau.

Từ việc một gói tạo một không gian tên mới trong các package khác nhau có thể tránh việc xung đột đặt chung tên tại các gói khác nhau. Với việc sử dụng package, có thể dễ dàng cung cấp khả năng truy cập và nó dễ dàng để chứa các class liên quan đến nhau.

8.2. Cách tạo gói trong Java

Tạo một package trong Java

Khi tạo một package trong Java, bạn nên chọn tên cho package và đặt câu lệnh khai báo package ở trên cùng của source file.

Lệnh package nên đặt tại dòng code đầu tiên. Bạn chỉ có thể khai báo lệnh package này một lần trong một source file, và nó áp dụng tới tất cả các kiểu trong file.

Nếu một lệnh khai báo package không được sử dụng, kiểu class, interface, enumerations hoặc annotation sẽ được đặt vào package mặc định không có tên.

Ví dụ: Cùng xem ví dụ về việc tạo một package tên là animals. Trong thực tế lập trình, việc sử dụng các package thường lấy tên viết thường để tránh xung đột giữa với tên class và tên interface.

Đặt một interface trong package animals:

```
/* Ten File : Animal.java */

package animals;

interface Animal {

  public void eat();

  public void travel();

}
```

Lợi thế của package trong Java

Java package được sử dụng để phân loại các lớp và các interface để mà chúng có thể được duy trì dễ dàng hơn.

Java package cung cấp bảo vệ truy cập.

Java package xóa bỏ các xung đột về đặt tên.

Ví dụ khác về package trong Java

Từ khóa package được sử dụng để tạo một package trong Java.

```
//Luu duoi dang Simple.java

package mypack;

public class Simple{

 public static void main(String args[]){

   System.out.println("Chao mung ban den voi package trong Java");

  }
```

}

Cách biên dịch Java package

Nếu bạn không sử dụng bất cứ IDE nào, bạn cần theo cú pháp sau:

javac -d thu_muc ten_javafile

Ví dụ:

javac -d . Simple.java

Tùy chọn –d xác định đích, là nơi để đặt class file đã tạo. Bạn có thể sử dụng bất cứ tên thư mục nào như /home (với Linux), d:/abc (với Windows), … Nếu bạn muốn giữ package bên trong cùng thư mục, bạn có thể sử dụng dấu chấm (.).

Cách chạy chương trình Java package

Bạn cần sử dụng tên đầy đủ (ví dụ mypack.Simple) để chạy lớp đó.

Để biên dịch: javac -d . Simple.java

Để chạy: java mypack.Simple

–d là một switch mà nói cho trình biên dịch Compiler nơi để đặt class file (nó biểu diễn đích đến). Dấu chấm (.) biểu diễn folder hiện tại.

8.3. Truy suất gói trong Java

Từ khóa import trong Java

Nếu một class sử dụng một class khác cùng package, tên package không cần được sử dụn. Lớp trong cùng package tìm thấy nhau mà không cần cú pháp đặc biệt nào.

Ví dụ:

Tại đây, một lớp Boss được thêm vào một package payroll đã chứa Employee. Lớp Boss có thể ám chỉ đến lớp Employee mà không cần sử dụng tiền tố payroll, như được minh họa như sau bởi lớp Boss.

package payroll;

public class Boss

```
{
  public void payEmployee(Employee e)
  {
    e.mailCheck();
  }
}
```

Nếu xảy ra trường hợp Boss không nằm trong payroll package, lớp Boss phải sử dụng một trong những kỹ thuật sau đây để tham chiếu đến class thuộc package khác.

Sử dụng tên đầy đủ của class có thể được sử dụng. Ví dụ:

payroll.Employee

Package có thể được nhập bởi sử dụng từ khóa import và wild card (*). Ví dụ:

import payroll.*;

Một class có thể import chính nó với từ khóa import. Ví dụ:

import payroll.Employee;

Ghi chú: Một class file có thể chứa bất kỳ số lệnh import nào. Lệnh import phải xuất hiện sau mỗi lệnh khai báo package và trước từ khóa khai báo lớp.

Cách truy cập package từ package khác?

Có nhiều cách để truy cập package từ package bên ngoài, đó là:

Sử dụng tenpackage.*

Nếu bạn sử dụng package.*, thì tất cả các lớp và interface của package này sẽ là có thể truy cập, nhưng không với các package con. Từ khóa import được sử dụng để làm cho các lớp và interface của package khác có thể truy cập tới package hiện tại. Ví dụ:

```
//Luu duoi dang A.java
  package pack;
public class A{
```

```
  public void msg(){System.out.println("Hello");}
}
//Luu duoi dang B.java
  package mypack;
import pack.*;
  class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Sử dụng tenpackage.tenlop

Nếu bạn import tenpackage.tenlop, thì chỉ có lớp được khai báo của package này sẽ là có thể truy cập. Ví dụ:

```
//Luu duoi dang A.java
  package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
//Luu duoi dang B.java
  package mypack;
import pack.A;

class B{
```

```
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Sử dụng tên đầy đủ

Nếu bạn sử dụng tên đầy đủ, thì chỉ có lớp được khai báo của package này sẽ là có thể truy cập. Bây giờ bạn không cần import. Nhưng bạn cần sử dụng tên đầy đủ mỗi khi bạn đang truy cập lớp hoặc interface. Nói chung, nó được sử dụng khi hai package có cùng tên lớp, ví dụ: hai package là java.util và java.sql chứa lớp Date. Ví dụ:

```
//Luu duoi dang A.java
  package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
//Luu duoi dang B.java
  package mypack;
class B{
  public static void main(String args[]){
   pack.A obj = new pack.A();//Su dung ten day du
   obj.msg();
  }
}
```

Ghi chú: Nếu bạn import một package, thì các package con sẽ không được import.

Nếu bạn import một package, thì tất cả các lớp và interface của package đó sẽ được import ngoại trừ lớp và interface của package con. Vì thế, bạn cũng cần import cả các package con.

Package con trong Java

Package mà bên trong package khác thì được gọi là package con (subpackage). Ví dụ: Sun Microsystem đã định nghĩa một package có tên là java chứa nhiều lớp như System, String, Reader, Writer, Socket, … Những lớp này biểu diễn một nhóm cụ thể, ví dụ như các lớp Reader và Writer là cho hoạt động I/O, các lớp Socket và ServerSocket là cho lập trình mạng, …. Vì thế, Sun đã lại phân loại java package thành các subpackage như lang, net, io, … và đặt các lớp liên quan tới IO vào io package, …

Ví dụ về subpackage

```
package com.vietjack.core;

class Simple{

  public static void main(String args[]){

   System.out.println("Hello subpackage");

  }

}
```

Để biên dịch: javac -d . Simple.java

Để chạy: java com.vietjack.core.Simple

Cách gửi class file tới thư mục hoặc drive khác?

Giả sử một tình huống, bạn muốn đặt class file của A.java source file trong thư mục classes của c: drive. Ví dụ:

```
//Luu duoi dang Simple.java

  package mypack;

public class Simple{

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

```
    System.out.println("Chao mung den voi package");
  }
}
```

Để biên dịch: e:\sources> javac -d c:\classes Simple.java

Để chạy: Để chạy chương trình này từ thư mục e:\source, bạn cần thiết lập classpath của thư mục, nơi mà class file ở đó.

e:\sources> set classpath=c:\classes;.;

e:\sources> java mypack.Simple

Cách khác với -classpath switch

Bạn có thể sử dụng –class switch với javac và java tool. Để chạy chương trình từ thư mục e:\source, bạn có thể sử dụng –class switch của java mà nói cho nó biết nơi để tìm class file. Ví dụ:

e:\sources> java -classpath c:\classes mypack.Simple

Cách để tải class file hoặc jar file

Cách để tải class file hoặc jar file:

Tạm thời: bởi thiết lập classpath trong command prompt hoặc bởi –classpath switch.

Vĩnh viễn: bởi thiết lập classpath trong biến môi trường hoặc bởi tạo jar file, chứa tất cả class file, và sao chép jar file trong thư mục jre/lib/ext.

Qui tắc: Chỉ có một lớp public trong một java source file và nó phải được lưu trữ bởi tên lớp public.

```
//Luu duoi dang C.java neu khong se gay ra Compilte Time Error
  class A{}
class B{}
public class C{}
```

Bài 10: Kế thừa (I)

10.1. Lớp cơ sở và lớp dẫn xuất

- Một lớp được xây dựng thông qua kế thừa từ một lớp khác gọi là lớp dẫn xuất (hay còn gọi là lớp con, lớp hậu duệ), lớp dùng để xây dựng lớp dẫn xuất được gọi là lớp cơ sở (hay còn gọi là lớp cha, hoặc lớp tổ tiên)

- Một lớp dẫn xuất ngoài các thành phần của riêng nó, nó còn được kế thừa tất cả các thành phần của lớp cha

10.2. Cách xây dựng lớp dẫn xuất

Để nói lớp b là dẫn xuất của lớp a ta dùng từ khoá extends, cú pháp như sau:

```
class b extends a{
// phần thân của lớp b
}
```

10.3. Thừa kế các thuộc tính

Thộc tính của lớp cơ sở được thừa kế trong lớp dẫn xuất, như vậy tập thuộc tính của lớp dẫn xuất sẽ gồm: các thuộc tính khai báo trong lớp dẫn xuất và các thuộc tính của lớp cơ sở, tuy nhiên trong lớp dẫn xuất ta không thể truy cập vào các thành phần private, package của lớp cơ sở

10.4 Thừa kế phương thức

Lớp dẫn xuất kế thừa tất cả các phương thức của lớp cơ sở trừ:

- Phương thức tạo dựng
- Phương thức finalize

10.5 Khởi đầu lớp cơ sở

Lớp dẫn xuất kế thừa mọi thành phần của lớp cơ, điều này dẫn ta đến một hình dung, là lớp dẫn xuất có cùng giao diện với lớp cơ sở và có thể có các thành phần mới bổ sung thêm. nhưng thực tế không phải vậy, kế thừa không chỉ là sao chép giao diện của lớp của lớp cơ sở. Khi ta tạo ra một đối tượng của lớp suy dẫn, thì nó chứa bên trong nó một sự vật con của lớp cơ sở, sự vật con này như thể ta đã tạo ra một sự vật tường minh của lớp cơ sở, thế thì lớp cơ sở phải được bảo đảm khởi đầu đúng, để thực hiện điều đó trọng java ta làm như sau:

Thực hiện khởi đầu cho lớp cơ sở bằng cách gọi cấu tử của lớp cơ sở bên trong cấu tử của lớp dẫn xuất, nếu bạn không làm điều này thì java sẽ làm giúp ban, nghĩa là java luôn tự động thêm lời gọi cấu tử của lớp cơ sở vào cấu tử của lớp dẫn xuất nếu như ta quên làm điều đó, để có thể gọi cấu tử của lớp cơ sở ta sử dụng từ khoá super

Ví dụ 1: ví dụ này không gọi cấu tử của lớp cơ sở một cách tường minh

```
class B
{
public B ()
{
        System.out.println ( "Ham tao của lop co so" );
}
}

public class A
extends B
{
public A ()
{// không gọi hàm tạo của lớp cơ sở tường minh
```

```
        System.out.println ( "Ham tao của lop dan xuat" );
    }

    public static void main ( String arg[] )
    {
        A thu = new A ();
    }
    }
```

Kết quả chạy chương trình như sau:

```
    Ham tao của lop co so
    Ham tao của lop dan xuat
```

Ví dụ 2: ví dụ này sử dụng từ khoá super để gọi cấu tử của lớp cơ sở một cách tường minh

```
    class B
    {
    public B ()
    {
        System.out.println ( "Ham tao của lop co so" );
    }
    }

    public class A
    extends B
    {
    public A ()
    {
        super();// gọi tạo của lớp cơ sở một cách tường minh
```

```
        System.out.println ( "Ham tao của lop dan xuat" );
    }

    public static void main ( String arg[] )
    {
        A thu = new A ();
    }
    }
```

khi chạy chưng trình ta thấy kết quả giống hệt như ví dụ trên

Chú ý 1: nếu gọi tường minh cấu tử của lớp cơ sở, thì lời gọi này phải là lệnh đầu tiên, nếu ví dụ trên đổi thành

```
    class B
    {
    public B ()
    {
        System.out.println ( "Ham tao của lop co so" );
    }
    }

    public class A
    extends B
    {
    public A ()
    {// Lời gọi cấu tử của lớp cơ sở không phải là lệnh đầu tiên
        System.out.println ("Ham tao của lop dan xuat");
        super ();

    }
```

```
public static void main ( String arg[] )
{
        A thu = new A ();
}
}
```

nếu biên dịch đoạn mã này ta sẽ nhân được một thông báo lỗi như sau: "A.java": call to super must be first statement in constructor at line 15, column 15

Chú ý 2: ta chỉ có thể gọi đến một hàm tạo của lớp cơ sở bên trong hàm tạo của lớp dẫn xuất, ví dụ chỉ ra sau đã bị báo lỗi

```
class B
{
public B ()
{
        System.out.println ( "Ham tao của lop co so" );
}

public B ( int i )
{
        System.out.println ( "Ham tao của lop co so" );
}

}

public class A
extends B
{
public A ()
```

```
{
        super ();
        super ( 10 );/ / không thể gọi nhiều hơn 1 hàm tạo của lớp cơ sở
        System.out.println ( "Ham tao của lop dan xuat" );

}

public static void main ( String arg[] )
{
        A thu = new A ();
}
}
```

1. Trật tự khởi đầu

Trật tự khởi đầu trong java được thực hiện theo nguyên tắc sau: java sẽ gọi cấu tử của lớp cơ sở trước sau đó mới đến cấu tử của lớp suy dẫn, điều này có nghĩa là trong cây phả hệ thì các cấu tử sẽ được gọi theo trật tự từ gốc xuống dần đến lá

2. Trật tự dọn dẹp

Mặc dù java không có khái niệm huỷ tử như của C++, tuy nhiên bộ thu rác của java vẫn hoạt động theo nguyên tắc làm việc của cấu tử C++, tức là trật tự thu rác thì ngược lại so với trật tự khởi đầu.

VI. Ghi đè phương thức (Override)

Hiện tượng trong lớp cơ sở và lớp dẫn xuất có hai phương thức giống hệt nhau (cả tên lẫn bộ tham số) gọi là ghi đè phương thức (Override), chú ý Override khác Overload.

Gọi phương thức bị ghi đè của lớp cơ sở

Bên trong lớp dẫn xuất, nếu có hiện tượng ghi đè thì phương thức bị ghi đè của lớp cơ sở sẽ bị ẩn đi, để có thể gọi phương thức bị ghi đè của lớp cơ sở ta dùng từ khoá super để truy cập đến lớp cha, cú pháp sau:

super.*overriddenMethodName*();

Chú ý: Nếu một phương thức của lớp cơ sở bị bội tải (Overload), thì nó không thể bị ghi đè (Override) ở lớp dẫn xuất.

Bài 12: Kế thừa (II)

12.1. Thành phần protected

Trong một vài bài trước ta đã làm quen với các thành phần private, public, sau khi đã học về kế thừa thì từ khoá protected cuối cùng đã có ý nghĩa.

Từ khoá protected báo cho java biết đây là thành phần riêng tư đối với bên ngoài nhưng lại sẵn sàng với các con cháu

12.2. Từ khoá final

Từ khoá final trong java có nhiều nghĩa khác nhau, nghĩa của nó tuỳ thuộc vào ngữ cảnh cụ thể, nhưng nói chung nó muốn nói "cái này không thể thay đổi được".

1. Thuộc tính final

Trong java cách duy nhất để tạo ra một hằng là khai báo thuộc tính là final

Ví dụ:

```
public class A
{
// định nghĩa hằng tên MAX_VALUE giá trị 100
static final int MAX_VALUE = 100;
public static void main ( String arg[] )
{
        A thu = new A ();
        System.out.println("MAX_VALUE= " +thu.MAX_VALUE);
}
}
```

Chú ý:

1) khi đã khai báo một thuộc tính là final thì thuộc tính này la hăng, do vậy ta không thể thay đổi giá trị của nó
2) khi khai báo một thuộc tính là final thì ta phải cung cấp giá trị ban đầu cho nó
3) nếu một thuộc tính vừa là final vừa là static thì nó chỉ có một vùng nhớ chung duy nhất cho cả lớp

2. Đối số final

Java cho phép ta tạo ra các đối final bằng việc khai báo chúng như vậy bên trong danh sách đối, nghĩa là bên trong thân của phương pháp này, bất cứ cố gắng nào để thay đổi giá trị của đối đều gây ra lỗi lúc dịch

Ví dụ sau bị báo lỗi lúc dịch vì nó cố gắng thay đổi giá trị của đối final

```
public class A
{
static public void thu ( final int i )
{
        i=i+1;//không cho phép thay đổi giá trị của tham số final
        System.out.println ( i );;
}

public static void main ( String arg[] )
{
        int i = 100;
        thu ( i );

}
}
```

chương trình này sẽ bị báo lỗi:

"A.java": variable i might already have been assigned to at line 5, column 9

3. Phương thức final

Một phương thức bình thường có thể bị ghi đè ở lớp dẫn xuất, đôi khi ta không muốn phương thức của ta bị ghi đè ở lớp dẫn xuất vì lý do gì đó, mục đích chủ yếu của các phương thức final là tránh ghi đè, tuy nhiên ta thấy rằng các phương thức

Bài 14: Đa hình (I)

14.1. Giới thiệu chung về đa hình

Đa hình thái trong lập trình hướng đối tượng đề cập đến khả năng quyết định trong lúc thi hành (runtime) mã nào sẽ được chạy, khi có nhiều phương thức trùng tên nhau nhưng ở các lớp có cấp bậc khác nhau.

Chú ý: khả năng đa hình thái trong lập trình hướng đối tượng còn được gọi với nhiều cái tên khác nhau như: tương ứng bội, kết ghép động,..

Đa hình thái cho phép các vấn đề khác nhau, các đối tượng khác nhau, các phương thức khác nhau, các cách giải quyết khác nhau theo cùng một lược đồ chung.

Các bước để tạo đa hình thái:

1. Xây dựng lớp cơ sở (thường là lớp cơ sở trừu tượng, hoặc là một giao diện), lớp này sẽ được các lớp con mở rộng(đối với lớp thường, hoặc lớp trừu tượng), hoặc triển khai chi tiết (đối với giao diện).
2. 2. Xây dựng các lớp dẫn xuất từ lớp cơ sở vừa tạo. trong lớp dẫn xuất này ta sẽ ghi đè các phương thức của lớp cơ sở(đối với lớp cơ sở thường), hoặc triển khai chi tiết nó (đối với lớp cơ sở trừu tượng hoặc giao diện).
3. Thực hiện việc tạo khuôn xuống, thông qua lớp cơ sở, để thực hiện hành vi đa hình thái

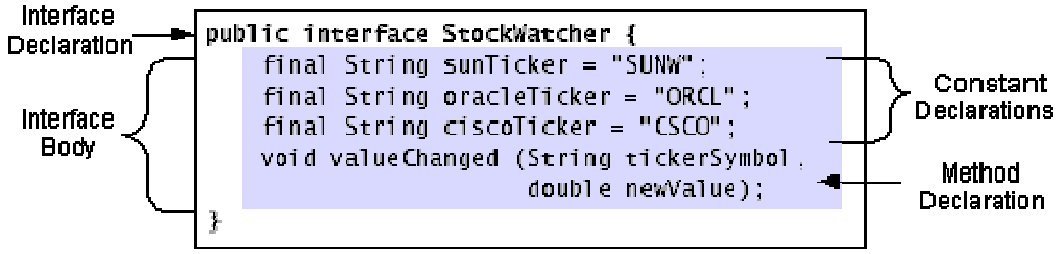
Khái niệm về tạo khuôn lên, tạo khuôn xuống

- Hiện tượng một đối tượng của lớp cha tham trỏ đến một đối tượng của lớp con thì được gọi là tạo khuôn xuống, việc tạo khuôn xuống luôn được java chấp thuận, do vậy khi tạo khuôn xuống ta không cần phải ép kiểu tường minh.
- Hiện tượng một đối tượng của lớp con tham trỏ tới một đối tượng của lớp cha thì được gọi là tạo khuôn lên, việc tạo khuôn lên là an toàn, vì một đối tượng của lớp con cũng có đầy đủ các thành phần của lớp cha, tuy nhiên việc tạo khuôn lên sẽ bị báo lỗi nếu như ta không ép kiểu một cách tường minh.

14.2 *Giao diện*

Từ khoá interface đã đưa khái niệm abstract đi xa thêm một bước nữa. Ta có thể nghĩ nó như là một lớp abstract "thuần tuý", nó cho phép ta tạo ra một lớp thuần ảo, lớp này chỉ gồm tập các giao diện cho các lớp muốn dẫn xuất từ nó, một interface cũng có thể có các trường, tuy nhiên java tự động làm các trường này thành *static và final*

Để tạo ra một interface, ta dùng từ khoá interface thay vì từ khoá class. Một interface gồm có 2 phần: phần khai báo và phần thân, phần khai báo cho biết một số thông tin như: tên của interface, nó có kế thừa từ một giao diện khác hay không. Phần thân chứa các khai báo hằng, khai báo phương thức (nhưng không có cài đặt). Giống như một lớp ta cũng có thể thêm bổ từ public vào trước định nghĩa của interface. Sau đây là hình ảnh của một interface.



Nhưng do java tự động làm các trường thành final nên ta không cần thêm bổ từ này, do vậy ta có thể định nghĩa lại giao diện như sau:

Nhưng do java tự động làm các trường thành final nên ta không cần thêm bổ từ này

```
public interface StockWatcher
{
final String
sunTicker = "SUNW";
final String oracleTicker = "ORCL";
final String ciscoTicker = "CSCO";
```

void valueChanged(String tickerSymbol, double newValue);
}

1. Phần khai báo của giao diện

Tổng quát phần khai báo của một giao diện có cấu trúc tổng quát như sau:

Public	//giao diện này là công cộng
interface InterfaceName	//tên của giao diện
Extends SuperInterface	//giao diện này là mở rộng của 1 giao diện khác
{	
InterfaceBody	//thân của giao diện
}	

Trong cấu trúc trên có 2 phần bắt buộc phải có đó là phần interface và *InterfaceName, các phần khác là tuỳ chọn.*

2. Phần thân

Phần thân khai báo các các hằng, các phương thức rỗng (không có cài đặt), các phương thức này phải kết thúc với dấu chấm phẩy ';', bởi vì chúng không có phần cài đặt

Chú ý:

1) Tất cả các thành phần của một giao diện tự động là public do vậy ta không cần phải cho bổ từ này vào.
2) Java yêu cầu tất cả các thành phần của giao diện phải là public, nếu ta thêm các bổ từ khác như private, protected trước các khai báo thì ta sẽ nhận được một lỗi lúc dịch
3) Tất cả các trường tự động là final và static, nên ta không cần phải cho bổ từ này vào.

3. Triển khai giao diện

Bởi một giao diện chỉ gồm các mô tả chúng không có phần cài đặt, các giao diện được định nghĩa để cho các lớp dẫn xuất triển khai, do vậy các lớp dẫn xuất từ lớp này phải triển khai đầy đủ tất cả các khai báo bên trong giao diện, để triển khai một giao diện bạn bao gồm từ khoá implements vào phần khai báo lớp, lớp của bạn có thể triển khai một hoặc nhiều giao diện (hình thức này tương tự như kế thừa bội của C++)

Ví dụ

```
public class StockApplet extends Applet implements StockWatcher {
.. .

public void valueChanged(String tickerSymbol, double newValue) {
if (tickerSymbol.equals(sunTicker)) {
.. .
} else if (tickerSymbol.equals(oracleTicker)) {
.. .
} else if (tickerSymbol.equals(ciscoTicker)) {
.. .
}
}
}
```

Chú ý:

1) Nếu một lớp triển khai nhiều giao diện thì các giao diện này được liệt kê cách nhau bởi dấu phẩy ‘,’

2) Lớp triển khai giao diện phải thực thi tất cả các phương thức được khai báo trong giao diện, nếu như lớp đó không triển khai, hoặc triển khai không hết thì nó phải được khai báo là abstract
3) Do giao diện cũng là một lớp trừu tượng do vậy ta không thể tạo thể hiện của giao diện
4) Một lớp có thể triển khai nhiều giao diện, do vậy ta có lợi dụng điều này để thực hiện hành vi kế thừa bội, vốn không được java hỗ trợ
5) Một giao diện có thể mở rộng một giao diện khác, bằng hình thức kế thừa

Bài 15: Bài tập và thảo luận về Kế thừa

15.1. Tại sao lại cần Kế thừa?

Tính kế thừa là một hình thức của việc sử dụng lại phần mềm trong đó các lớp mới được tạo từ các lớp đã có bằng cách "hút" các thuộc tính và hành vi của chúng và tô điểm thêm với các khả năng mà các lớp mới đòi hỏi. Việc sử dụng lại phần mềm tiết kiệm thời gian trong việc phát triển chương trình. Nó khuyến khích sử dụng lại phần mềm chất lượng cao đã thử thách và gỡ lỗi, vì thế giảm thiểu các vấn đề sau khi một hệ trở thành chính thức. Tính đa hình cho phép chúng ta viết các chương trình trong một kiểu cách chung để xử lý các lớp có liên hệ nhau. Tính kế thừa và tính đa hình các kỹ thuật có hiệu lực đối với sự chia với sự phức tạp của phần mềm.

Khi tạo một lớp mới, thay vì viết các thành viên dữ liệu và các hàm thành viên, lập trình viên có thể thiết kế mà lớp mới được kế thừa các thành viên dữ liệu và các hàm thành viên của lớp trước định nghĩa là lớp cơ sở (base class). Lớp mới được tham chiếu là lớp dẫn xuất (derived class). Mỗi lớp dẫn xuất tự nó trở thành một ứng cử là một lớp cơ sở cho lớp dẫn xuất tương lai nào đó.

Bình thường một lớp dẫn xuất thêm các thành viên dữ liệu và các hàm thành viên, vì thế một lớp dẫn xuất thông thường rộng hơn lớp cơ sở của nó. Một lớp dẫn xuất được chỉ định hơn một lớp cơ sở và biểu diễn một nhóm của các đối tượng nhỏ hơn. Với đối tượng đơn, lớp dẫn xuất, lớp dẫn xuất bắt đầu bên ngoài thực chất giống như lớp cơ sở. Sức mạnh thực sự của sự kế thừa là khả năng định nghĩa trong lớp dẫn xuất các phần thêm, thay thế hoặc tinh lọc các đặc tính kế thừa từ lớp cơ sở.

Mỗi đối tượng của một lớp dẫn xuất cũng là một đối tượng của lớp cơ sở của lớp dẫn xuất đó. Tuy nhiên điều ngược lại không đúng, các đối tượng lớp cơ sở không là các đối tượng của các lớp dẫn xuất của lớp cơ sở đó. Chúng ta sẽ lấy mối quan hệ "đối tượng lớp dẫn xuất là một đối tượng lớp cơ sở" để thực hiện các thao tác quan trọng nào đó. Chẳng hạn, chúng ta có thể luồn một sự đa dạng của các đối tượng khác nhau có liên quan thông qua sư kế thừa thành danh sách liên kết của các đối tượng lớp cơ sở. Điều này cho phép sự đa dạng của các đối tượng để xử lý một cách tổng quát.

Chúng ta phân biệt giữa "là một" (is a) quan hệ và "có một" (has a) quan hệ. "là một" là sự kế thừa. Trong một "là một" quan hệ, một đối tượng của kiểu lớp dẫn xuất cũng có thể được xử lý như một đối tượng của kiểu lớp cơ sở. "có một" là sự phức hợp (composition). Trong một "có một" quan hệ, một đối tượng lớp có một hay nhiều đối tượng của các lớp khác như là các thành viên, do đó lớp bao các đối tượng này gọi là lớp phức hợp (composed class).

Tóm lại:

- Để ghi đè phương thức (Method Overriding), do đó có thể thu được tính đa hình tại runtime.

- Để làm tăng tính tái sử dụng của code.

15.2. Các loại kế thừa trong Java

Trên cơ sở các lớp thì có 3 loại kế thừa trong Java, đó là *single (đơn), multilevel (nhiều tầng) và hierarchical (có cấu trúc)*. Trong lập trình Java, *đa kế thừa (multiple) và kế thừa lai (hybrid)* chỉ được hỗ trợ thông qua Interface. Chúng ta sẽ tìm hiểu về Interface trong chương sau đó.

Ghi chú: Đa kế thừa không được hỗ trợ trong Java thông qua lớp. Khi một lớp kế thừa từ nhiều lớp, thì đây là đa kế thừa.

Câu hỏi: Tại sao đa kế thừa không được hỗ trợ trong Java thông qua lớp?

Trả lời: Để giảm tính phức tạp và làm đơn giản hóa ngôn ngữ, đa kế thừa không được hỗ trợ trong Java. Giả sử có tình huống có ba lớp là A, B và C. Lớp C kế thừa lớp A và B. Nếu các lớp A và B có cùng phương thức và bạn gọi nó từ đối tượng lớp con, thì điều này gây là tính lưỡng nghĩa là để gọi phương thức của lớp A hoặc lớp B.

Bởi vì, compile time error thì tốt hơn là runtime error, Java sẽ thông báo một compile time error nếu bạn kế thừa 2 lớp. Do đó, dù bạn có hay không có cùng phương thức hay khác phương thức, thì đó cũng là một lỗi tại compile time.

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B
{ //gia su neu no da co

 Public Static void main(String args[]){
   C obj=new C();
   obj.msg();//Bay gio phuong thuc msg() nao se duoc goi?
```

```
}
}
```

Chương trình trên sẽ cho một Compile Time Error.

Khi bạn đã hiểu rõ về từ khóa **extends**, chúng ta cùng tìm hiểu về từ khóa **implements** trong quan hệ IS-A.

Từ khóa **implements** được sử dụng bởi các lớp mà kế thừa từ Interface. Interface có thể không bao giờ được kết thừa bởi các lớp.

Ví dụ:

```
public interface A {}

public class B implements A{
}

public class C extends B{
}
```

15.3 Quan hệ HAS-A trong Java

Có những quan hệ chủ yếu dựa vào cách sử dụng. Nó xác định có hay không một lớp cụ thể HAS-A. Quan hệ này giúp chúng ta giảm được dư thừa trong code cũng như tránh các bug.

Cùng xem ví dụ dưới đây:

```
public class Vehicle{}
public class Speed{}
public class Van extends Vehicle{
        private Speed sp;
```

```
}
```

Điều này chỉ ra rằng lớp Van có quan hệ HAS-A với lớp Speed. Việc sử dụng lớp riêng rẽ cho lớp Speed, chúng ta không cần thiết phải đặt toàn bộ code của lớp Speed bên trong lớp Van, điều này tăng tính tái sử dụng của lớp Speed cho nhiều ứng dụng.

Một đặc điểm quan trọng nữa phải ghi nhớ là Java chỉ hỗ trợ kế thừa đơn. Điều này nghĩa là một lớp không thể kế thừa từ nhiều hơn một lớp. Do đó, đoạn code dưới đây là không hợp lệ:

```
public class C extends A, B{}
```

Mặc dù vậy một lớp vẫn có thể implement một hoặc nhiều interface. Điều này loại bỏ khả năng không thể đa kế thừa trong Java.

Bài 17 Đa hình (II)

17.1 Giới thiệu

Lớp trừu tượng là một trong những khái niệm quan trọng trên nền tảng .NET. Thường, bạn muốn tạo ra các lớp mà chỉ làm lớp gốc (base class – lớp đầu tiên trong cây thừa kế hay còn gọi là lớp tổ tiên), và dĩ nhiên, bạn sẽ không muốn người khác tạo ra đối tượng cho các lớp này. Chúng ta có thể sử dụng khái niệm lớp trừu tượng để cài đặt chức năng này trong C# sử dụng từ khóa 'abstract'.

Một lớp trừu tượng có nghĩa là không khởi tạo được đối tượng của lớp này, nhưng cho phép thừa kế để tạo ra lớp con.

Khai báo lớp trừu tượng trong C#:

abstract class tên_lớp_trừu_tượng

{

}

Ví dụ:

2

3

abstract class Window

{

}

17.2 Phương thức trừu tượng (abstract method)

Trong thiết kế hướng đối tượng, khi các bạn thiết kế ra một base class, và các bạn mong muốn rằng người khác khi thừa kế lớp này thì phải ghi đè (override) lên các phương thức xác định trước. Trong trường hợp người khác thừa kế lớp của các bạn mà không ghi đè

lên những phương thức này thì trình biên dịch sẽ báo lỗi. Khái niệm phương thức trừu tượng sẽ giúp các bạn trong tình huống này.

Một lớp trừu tượng có thể chứa cả phương thức trừu tượng (phương thức không có phần thân) và phương thức bình thường (phương thức có phần thân hay phương thức thàdow

```
public class ListBox : Window
{
// Khởi dựng có tham số
public ListBox(int top, int left, string theContents)
: base(top, left) // gọi khởi dựng của lớp cơ sở
{
mListBoxContents = theContents;
}

public override void DrawWindow()
{
Console.WriteLine("ListBox write: {0}", mListBoxContents);
}
// biến thành viên private
private string mListBoxContents;
public override string Content
{
set { mListBoxContents = value; }
get { return mListBoxContents; }
}
}
```

[/code]

Trong ví dụ trên, các bạn thấy thuộc tính (property) Content được khai báo trong lớp Window, nhưng không có biến chứa dữ liệu cho nó không được khai báo trong lớp này. Do đó nó được cài đặt kiểu abstract.

Thuộc tính Content được override trong lớp con ListBox, và biến chứa dữ liệu cho nó là mListBoxContents.

17.3 Một số quy tắc áp dụng cho lớp trừu tượng

– Một lớp trừu tượng không thể là một sealed class. Khai báo như ví dụ dưới đây là sai:

4

```
// Khai báo sai
abstract sealed class Window
{
}
```

– Phương thức trừu tượng chỉ khai báo trong lớp trừu tượng.

– Một phương thức trừu tượng không sử dụng chỉ định từ truy xuất là private.

```
// Khai báo sai
abstract class Window
{
    private abstract void DrawWindow();
}
```

Chỉ định từ truy xuất của phương thức trừu tượng phải giống nhau trong phần khai báo ở lớp cha lẫn lớp con. Nếu bạn đã khai báo chỉ định từ truy xuất protected cho phương thức trừu tượng ở lớp cha thì trong lớp con bạn cũng phải sử dụng chỉ định từ truy xuất protected. Nếu chỉ định từ truy xuất không giống nhau thì trình biên dịch sẽ báo lỗi.

– Một phương thức trừu tượng không sử dụng chỉ định từ truy xuất virtual. Bởi vì bản thân phương thức trừu tượng đã bao hàm khái niệm virtual.

```
// Khai báo sai
abstract class Window
{
   private abstract virtual void DrawWindow();
}
```

– Một phương thức trừu tượng không thể là phương thức static.

```
// Khai báo sai
abstract class Window
{
   private abstract static void DrawWindow();
}
```

17.4 Lớp trừu tượng (abstract class) và giao diện (interface)

Trong lớp trừu tượng chứa cả phương thức trừu tượng lẫn phương thức thành viên. Nhưng trong interface thì chỉ chứa phương thức trừu tượng và lớp con khi thừa kế từ interface cần phải ghi đè (override) lên các phương thức này.

```
interface IFile
{
   void Save();
   void Load();
}
```

Các phương thức trừu tượng khai báo trong interface không sử dụng chỉ định từ truy xuất, mặc định sẽ là public.

Một lớp chỉ có thể thừa kế từ một lớp cha, nhưng có thể thừa kế từ nhiều interface.

Bài 18: Bài tập và thảo luận về Đa hình

18.1. Tại sao lại cần Đa hình?

Tính đa hình trong Java là một khái niệm mà từ đó chúng ta có thể thực hiện một hành động đơn theo nhiều cách khác nhau. Tính đa hình được suy ra từ hai từ Hy Lạp là Poly và Morphs. Poly nghĩa là nhiều và morphs nghĩa là hình, dạng. Có hai kiểu đa hình trong Java: Đa hình tại compile time và đa hình runtime. Chúng ta có thể thực hiện tính đa hình trong Java bởi nạp chồng phương thức và ghi đè phương thức.

Nếu bạn nạp chồng phương thức static trong Java, thì đó là ví dụ về đa hình tại compile time. Ở chương này chúng sẽ tập trung vào đa hình tại runtime trong Java.

Điều quan trọng để biết là có cách nào truy cập một đối tượng qua các biến tham chiếu. Một biến tham chiếu có thể chỉ là một kiểu. Khi được khai báo, kiểu của biến tham chiếu này không thể thay đổi.

Biến tham chiếu có thể được gán cho những đối tượng khác được cung cấp mà không được khai báo final. Kiểu của biến tham chiếu sẽ xác định phương thức mà có thể được triệu hồi trên đối tượng.

Một biến tham chiếu có thể được hướng đến bất kì đối tượng với kiểu khai báo hoặc bất kì kiểu con nào của kiểu khai báo. Một biến tham chiếu có thể được khai báo như là một class hoặc một interface.

18.2. Cách sử dụng Đa hình trong lập trình hướng đối tượng

Đa hình tại runtime trong Java

Đa hình tại runtime là một tiến trình mà trong đó một lời gọi tới một phương thức được ghi đè được xử lý tại runtime thay vì tại compile time. Trong tiến trình này, một phương thức được ghi đè được gọi thông qua biến tham chiếu của một lớp cha. Việc quyết định

phương thức được gọi là dựa trên đối tượng nào đang được tham chiếu bởi biến tham chiếu.

Trước khi tìm hiểu về đa hình tại runtime, chúng ta cùng tìm hiểu về Upcasting.

Upcasting là gì?

Khi biến tham chiếu của lớp cha tham chiếu tới đối tượng của lớp con, thì đó là Upcasting. Ví dụ:

```
class A{}
class B extends A{}
A a=new B();//day la upcasting
```

Ví dụ về đa hình tại runtime trong Java

Trong ví dụ, chúng ta tạo hai lớp Bike và Splendar. Lớp Splendar kế thừa lớp Bike và ghi đè phương thức run() của nó. Chúng ta gọi phương thức run bởi biến tham chiếu của lớp cha. Khi nó tham chiếu tới đối tượng của lớp con và phương thức lớp con ghi đè phương thức của lớp cha, phương thức lớp con được triệu hồi tại runtime.

Khi việc gọi phương thức được quyết định bởi JVM chứ không phải Compiler, vì thế đó là đa hình tại runtime.

```
class Bike{
  void run(){System.out.println("dang chay");}
}
class Splender extends Bike{
  void run(){System.out.println("chay an toan voi 60km");}

  public static void main(String args[]){
    Bike b = new Splender();//day la upcasting
    b.run();
```

```
  }
}
```

18.3. Case study: Đa hình tại runtime trong Java với thành viên dữ liệu

Phương thức bị ghi đè không là thành viên dữ liệu, vì thế đa hình tại runtime không thể có được bởi thành viên dữ liệu. Trong ví dụ sau đây, cả hai lớp có một thành viên dữ liệu là speedlimit, chúng ta truy cập thành viên dữ liệu bởi biến tham chiếu của lớp cha mà tham chiếu tới đối tượng lớp con. Khi chúng ta truy cập thành viên dữ liệu mà không bị ghi đè, thì nó sẽ luôn luôn truy cập thành viên dữ liệu của lớp cha.

Qui tắc: Đa hình tại runtime không thể có được bởi thành viên dữ liệu.

```
class Bike{
 int speedlimit=90;
}
class Honda3 extends Bike{
 int speedlimit=150;

 public static void main(String args[]){
  Bike obj=new Honda3();
  System.out.println(obj.speedlimit);//90
}
```

Đa hình tại runtime trong Java với kế thừa nhiều tầng (Multilevel)

Bạn theo dõi ví dụ sau:

```
class Animal{
void eat(){System.out.println("an");}
```

```
}

class Dog extends Animal{
void eat(){System.out.println("an hoa qua");}
}

class BabyDog extends Dog{
void eat(){System.out.println("uong sua");}

public static void main(String args[]){
Animal a1,a2,a3;
a1=new Animal();
a2=new Dog();
a3=new BabyDog();

a1.eat();
a2.eat();
a3.eat();
}
}
```

Và:

```
class Animal{
void eat(){System.out.println("animao dang an...");}
}
```

```
class Dog extends Animal{
void eat(){System.out.println("dog dang an...");}
}

class BabyDog1 extends Dog{
public static void main(String args[]){
Animal a=new BabyDog1();
a.eat();
}}
```

Vì, BabyDog không ghi đè phương thức eat(), do đó phương thức **eat()** của lớp Dog() được triệu hồi.

Bài 20: Bài tập và thảo luận tổng kết môn học

20.1. Những ưu điểm của lập trình hướng đối tượng

OOP giúp việc thiết kế, phát triển và bảo trì dễ dàng hơn trong khi với lập trình hướng thủ tục thì việc quản lý code là khá khó khăn nếu lượng code tăng lên. Điều này làm tăng hiệu quả có quá trình phát triển phần mềm.

OOP cung cấp Data Hiding (ẩn dữ liệu) trong khi đó trong hướng thủ tục một dữ liệu toàn cục có thể được truy cập từ bất cứ đâu.

OOP cung cấp cho bạn khả năng để mô phỏng các sự kiện trong thế giới thực một cách hiệu quả hơn. Chúng ta có thể cung cấp giải pháp cho các vấn đề trong thế giới thực nếu chúng ta sử dụng Lập trình hướng đối tượng.

20.2. Tóm tắt lập trình HĐT

Trong kỹ thuật lập trình hướng đối tượng, chúng ta thiết kế một chương trình bởi sử dụng các lớp và các đối tượng.

Object - Đối tượng là thực thể mang tính vật lý cũng như mang tính logic, trong khi lớp chỉ là thực thể logic. Đối tượng có các trạng thái và các hành vi. Ví dụ: Một dog có trạng thái là color, name, breed (dòng dõi) và cũng có các hành vi: Wag (vẫy đuôi), bark (sủa), eat (ăn). Một đối tượng là một instance (ví dụ,trường hợp) của một lớp.

Class - Một lớp là một nhóm các đối tượng mà có các thuộc tính chung. Lớp là một Template hoặc bản thiết kế từ đó đối tượng được tạo.

Đối tượng trong Java

Đó là một thực thể có trạng thái và hành vi, ví dụ như bàn, ghế, xe con, mèo, … Nó có thể mang tính vật lý hoặc logic. Ví dụ về logic đó là Banking system.

Một đối tượng có ba đặc trưng sau:

Trạng thái: biểu diễn dữ liệu (giá trị) của một đối tượng.

Hành vi: biểu diễn hành vi (tính năng) của một đối tượng như gửi tiền vào, rút tiền ra, …

Nhận diện: việc nhận diện đối tượng được triển khai thông qua một ID duy nhất. Giá trị của ID là không thể nhìn thấy với người dùng bên ngoài. Nhưng nó được sử dụng nội tại bởi JVM để nhận diện mỗi đối tượng một cách duy nhất.

Ví dụ: Bút là một đối tượng. Nó có tên là Thiên Long, có màu trắng, … được xem như là trạng thái của nó. Nó được sử dụng để viết, do đó viết là hành vi của nó.

Đối tượng là sự thể hiện (Instance) của một lớp. Lớp là một Template hoặc bản thiết kế từ đó đối tượng được tạo. Vì thế đối tượng là Instance (kết quả) của một lớp.

Lớp trong Java

Một lớp là một nhóm các đối tượng mà có các thuộc tính chung. Lớp là một Template hoặc bản thiết kế từ đó đối tượng được tạo. Một lớp trong Java có thể bao gồm:

Thành viên dữ liệu

Phương thức

Constructor

Block

Lớp và Interface

Cú pháp để khai báo một lớp

```
class ten_lop{
    thanh_vien_du_lieu;
    phuong_thuc;
}
```

Ví dụ đơn giản về Lớp và Đối tượng trong Java

Trong ví dụ này, chúng ta tạo một lớp Student có hai thành viên dữ liệu là id và name. Chúng ta đang tạo đối tượng của lớp Student bởi từ khóa new và in giá trị đối tượng.

```
class Student1{
 int id; //thanh vien du lieu (cung la bien instance)
```

```
String name; //thanh vien du lieu (cung la bien instance)


public static void main(String args[]){
 Student1 s1=new Student1(); //tao mot doi tuong Student
 System.out.println(s1.id);
 System.out.println(s1.name);
 }
}
```

Một lớp có thể chứa bất kỳ loại biến sau:

Biến Local: Các biến được định nghĩa bên trong các phương thức, constructor hoặc block code được gọi là biến Local. Biến này sẽ được khai báo và khởi tạo bên trong phương thức và biến này sẽ bị hủy khi phương thức đã hoàn thành.

Biến Instance: Các biến instance là các biến trong một lớp nhưng ở bên ngoài bất kỳ phương thức nào. Những biến này được khởi tạo khi lớp này được tải. Các biến instance có thể được truy cập từ bên trong bất kỳ phương thức, constructor hoặc khối nào của lớp cụ thể đó.

Biến Class: Các biến class là các biến được khai báo với một lớp, bên ngoài bất kỳ phương thức nào, với từ khóa static.

Phương thức trong Java

Trong Java, một phương thức là khá giống hàm, được sử dụng để trưng bày hành vi của một đối tượng. Phương thức giúp code tăng tính tái sử dụng và tối ưu hóa code.

Từ khóa new được sử dụng để cấp phát bộ nhớ tại runtime.

Constructor trong Java:

Khi bàn luận về các lớp, một trong những chủ đề quan trọng là các constructor. Mỗi lớp có một constructor. Nếu chúng ta không viết một constructor một cách rõ ràng cho một lớp thì bộ biên dịch Java xây dựng một constructor mặc định cho lớp đó.

Mỗi khi một đối tượng mới được tạo ra, ít nhất một constructor sẽ được gọi. Quy tắc chính của các constructor là chúng có cùng tên như lớp đó. Một lớp có thể có nhiều hơn một constructor.

Sau đây là ví dụ về một constructor:

```
public class Xecon{

  public Xecon(){

  }


  public Xecon(String ten){

    // Contructor nay co mot tham so la ten.

  }

}
```

Java cũng hỗ trợ **Lớp Singleton trong Java**, ở đây bạn sẽ có thể tạo chỉ một instance của một lớp.

Tạo một đối tượng trong Java:

Như đã đề cập trước đó, một lớp cung cấp bản thiết kế cho các đối tượng. Vì thế, về cơ bản, một đối tượng được tạo từ một lớp. Trong Java, từ khóa new được sử dụng để tạo một đối tượng mới.

Có ba bước khi tạo một đối tượng từ một lớp:

Khai báo: Một khai báo biến với một tên biến với một loại đối tượng.

Cài đặt: Từ khóa new được sử dụng để tạo đối tượng

Khởi tạo: Từ khóa new được theo sau bởi một lời gọi một constructor. Gọi hàm này khởi tạo đối tượng mới.

Dưới đây là ví dụ về tạo một đối tượng:

```
public class Xecon{


  public Xecon(String ten){

    // Contructor nay co mot tham so la ten.

    System.out.println("Ten xe la :" + ten );
```

```
  }
  public static void main(String []args){
    // Lenh sau se tao mot doi tuong la Xecuatoi
    Xecon Xecuatoi = new Xecon( "Toyota" );
  }
}
```

Nếu chúng ta biên dịch và chạy chương trình, nó sẽ cho kết quả sau:

```
Ten xe la :Toyota
```

Truy cập các biến instance và các phương thức trong Java

Các biến instance và các phương thức được truy cập thông qua các đối tượng được tạo. Để truy cập một biến instance, path sẽ là như sau:

```
/* Dau tien, ban tao mot doi tuong */
Doituongthamchieu = new Constructor();

/* Sau do ban goi mot bien nhu sau */
Doituongthamchieu.TenBien;

/* Bay gio ban co the goi mot phuong thuc lop nhu sau */
Doituongthamchieu.TenPhuongThuc();
```

Ví dụ:

Ví dụ này giải thích cách để truy cập các biến instance và các phương thức của một lớp:

```
public class Xecon{

  int Giaxe;
```

```
  public Xecon(String ten){
    // Contructor nay co mot tham so la ten.
    System.out.println("Ten xe la :" + ten );
  }
  public void setGia( int gia ){
     Giaxe = gia;
  }

  public int getGia( ){
     System.out.println("Gia mua xe la :" + Giaxe );
     return Giaxe;
  }
  public static void main(String []args){
    /* Tao doi tuong */
    Xecon Xecuatoi = new Xecon( "Toyota" );

    /* Goi mot phuong thuc lop de thiet lap gia xe */
    Xecuatoi.setGia( 1000000000 );

    /* Goi mot phuong thuc lop khac de lay gia xe */
    Xecuatoi.getGia( );

    /* Ban cung co the truy cap bien instance nhu sau */
    System.out.println("Gia tri bien :" + Xecuatoi.Giaxe );
  }
}
```

Biên dịch và thực thi chương trình sẽ cho kết quả sau:

```
Ten xe la :Toyota

Gia mua xe la :1000000000

Gia tri bien :1000000000
```

Ví dụ đối tượng và lớp mà duy trì bản ghi các sinh viên

Trong ví dụ này, chúng ta tạo hai đối tượng của lớp Student và khởi tạo giá trị của các đối tượng này bằng việc triệu hồi phương thức insertRecord trên nó. Ở đây, chúng ta đang hiển thị trạng thái (dữ liệu) của các đối tượng bằng việc triệu hồi phương thức displayInformation.

```
class Student2{

 int rollno;

 String name;


 void insertRecord(int r, String n){  //phuong thuc

 rollno=r;

  name=n;

 }


 void displayInformation(){System.out.println(rollno+" "+name);} //phuong thuc


 public static void main(String args[]){

  Student2 s1=new Student2();

  Student2 s2=new Student2();


  s1.insertRecord(111,"HoangThanh");

  s2.insertRecord(222,"ThanhHuong");
```

```
 s1.displayInformation();
 s2.displayInformation();

 }
}
```

Ví dụ khác về lớp và đối tượng trong Java

Ví dụ khác duy trì các bản ghi của lớp Rectangle. Phần giải thích tương tự như trên:

```
class Rectangle{
 int length;
 int width;

 void insert(int l,int w){
  length=l;
  width=w;
 }

 void calculateArea(){System.out.println(length*width);}

 public static void main(String args[]){
  Rectangle r1=new Rectangle();
  Rectangle r2=new Rectangle();

  r1.insert(11,5);
  r2.insert(3,15);

  r1.calculateArea();
```

```
 r2.calculateArea();
}
}
```

Các cách khác nhau để tạo đối tượng trong Java?

Có nhiều cách để tạo một đối tượng trong Java. Đó là:

Bằng từ khóa new

Bằng phương thức newInstance()

Bằng phương thức clone(), ….

Bằng phương thức factory, ...

Đối tượng vô danh (annonymous) trong Java

Vô danh hiểu đơn giản là không có tên. Một đối tượng mà không có tham chiếu thì được xem như là đối tượng vô danh. Nếu bạn phải sử dụng một đối tượng chỉ một lần, thì đối tượng vô danh là một hướng tiếp cận tốt.

```
class Calculation{


 void fact(int  n){
  int fact=1;
  for(int i=1;i<=n;i++){
   fact=fact*i;
  }
 System.out.println("factorial is "+fact);
}


public static void main(String args[]){
 new Calculation().fact(5); //Goi phuong thuc voi doi tuong vo danh (annonymous)
}
```

```
}
```

Tạo nhiều đối tượng bởi chỉ một kiểu

Chúng ta có thể tạo nhiều đối tượng bởi chỉ một kiểu như khi chúng ta thực hiện trong các kiểu gốc. Ví dụ:

```
Rectangle r1=new Rectangle(),r2=new Rectangle(); //Tao hai doi tuong
```

Ví dụ:

```
class Rectangle{
 int length;
 int width;

 void insert(int l,int w){
  length=l;
  width=w;
 }

 void calculateArea(){System.out.println(length*width);}

 public static void main(String args[]){
  Rectangle r1=new Rectangle(),r2=new Rectangle(); //Tao hai doi tuong

  r1.insert(11,5);
  r2.insert(3,15);

  r1.calculateArea();
  r2.calculateArea();
}
```

```
}
```

Kết quả là:

```
Output:55
    45
```

20.3. Trao đổi:

- So sánh Java với một vài ngôn ngữ lập trình hướng đối tượng khác?
- Sau ngôn ngữ lập trình hướng đối tượng sẽ là thế hệ ngôn ngữ lập trình nào?