LẬP TRINH HƯỚNG ĐỐI TƯỢNG TRONG JAVA phần 6

Java là một ngôn ngữ lập trình hướng đối tượng . Nếu bạn chưa bao giờ dùng một ngôn ngữ lập trình hướng đối tượng trước đây , bạn cần phải hiểu các khái niệm sau : lập trình hướ ng đối tượng (Object Oriented Programming) là gì ? đối tượng (Object), lớp (class) là gì , mối quan hệ giữa đối tượng và lớp , gởi thông điệp (Messages) đến các đối tượng là gì ?

I. KHÁI NIỆM LẬP TRINH HƯỚNG ĐỐI TƯỢNG

1. Lập trình hướng đối tượng (Object Oriented Programming)

Mỗi một chương trình máy tính đều gồm có 2 phần : phần mã lệnh và phần dữ liệu. Một số chương trình đặt trọng tâm ở phần mã lệnh , số khác đặt trọng tâm ở phần dữ liệu. Từ đó dẫn đến 2 mô hình quyết định nên cấu trúc của chương trình : một trả lời cho câu hỏi “Điều gì đang xảy ra” , và một cho “Cái gì đang chịu tác động” . Mô hình 1 gọi là mô hình hướng xử lý , nó mô tả như là một chươ ng trình bao gồm một chuỗi các bước thực hiện (mã lệnh). Nhưng khi chương trình càng ngày càng lớn và phức tạp thì khó khăn để sử dụng mô hình thứ nhất.

Vì vậy mô hình thứ 2 được đưa ra , đó là mô hình hướng đối tượng . Chương trình của bạn sẽ xây dựng dựa vào dữ liệu và phần giao diện được định nghĩa cho phần dữ liệu đó. Mô hình này được mô tả như là dữ liệu điều khiển truy xuất đối với mã lệnh.

Ngôn ngữ lập trình hướng đối tượng có các khả năng sau :

- Mô phỏng thế giới thực một cách tự nhiên bởi các đối tượng và mối quan hệ giữa chúng, thuận tiện cho việc thiết kế hệ thống phức tạp
- Thừa kế mã có sẵn một cách dễ dàng, giúp tiết kiệm công sức và nâng cao năng suất của người lập trình, dễ bảo trì, dễ nâng cấp, mở rộng

2. Trừu tượng hoá (Abstraction)

Con người đã đơn giản hoá các vấn đề phức tạp thông qua sự trừu tượng hoá . Ví dụ, người sử dụng máy tính kh ông nhìn máy tính một cách phức tạp . Nhờ sự trừu

tượng hoá mà người ta có thể sử dụng máy tính mà không quan tâm đến cấu trúc chi tiết bên trong máy tính. Họ chỉ sử dụng chúng như là một thực thể

Cách tốt nhất để nắm vững kỹ thuật trừu tượng là dùng hệ thống phân cấp. Điều này cho phép bạn phân lớp các thành phần có ý nghĩa của cả hệ thống phức tạp , chia nhỏ chúng thành những phần đơn giản có thể quản lý được . Nhìn bên ngoài máy tính là một đối tượng, nếu nhìn sâu hơn một cấp , máy tính bao gồm một số bộ phận : hộp điều khiển, màn hình, bàn phím, chuột..., các bộ phận này lại bao gồm các bộ phận nhỏ hơn, ví dụ như hộp điều khiển có bảng mạch chính chứa CPU, các mạch giao tiếp gắn trên bảng mạch chính , đĩa cứng, ổ đĩa mềm… Nhờ sự trừu tượng hoá mà bạn không quan tâm đến chi tiết từng bảng mạch , mà chỉ quan tâm mối quan hệ , giao tiếp giữa các bộ phận. Một mạch giao tiếp dù có chức năng ly kỳ thế nào đi nữa , bạn có thể sử dụng không mấy khó khăn nếu được ấn vừa vặn vào khe cắm trên bảng mạch chính.

Sự phân cấp trừu tượng một hệ thống phức tạp có thể áp dụng cho các chương trình máy tính . Phần dữ liệu từ một chương trình hướng xử lý kinh điển có thể trừu tượng hoá thành các đối tượng thành phần . Dãy các xử lý trở thành các thông điệp giữa các đối tượng. Vì thế các đối tượng cần có hoạt động đặc trưng riêng. Bạn có thể coi các đối tượng này như những thực thể độc lập tiếp nhận các yêu cầu từ bên ngoài . Đây là phần cốt lõi của lập trình hướng đối tượng.

II. CƠ CHẾ TRIỂN KHAI MÔ HINH HƯỚNG ĐỐI TƯỢNG

Tất cả các ngôn ngữ lập trình hướng đối tượng đều có các cơ chế cho phép bạn triển khai các mô hình hướng đối tượng. Đó là tính đóng gói, kế thừa, và tính đa hình.

1. Tính đóng gói (Encapsulation)

Đây là cơ chế dùng một vỏ bọc kết hợp phần dữ liệu và các thao tác trên dữ liệu đó (phần mã lệnh) thành một thể thống nhất , tạo nên sự an toàn , tránh việc sử dụng không đúng thiết kế, bảo vệ cho mã lệnh và dữ liệu chống việc truy xuất từ những đoạn mã lệnh bên ngoài.

Trong Java tính đóng gói thể hiện qua khái niệm lớp (Class). Lớp là hạt nhân của Java, tạo nền tảng cho lập trình hướng đối tượng trong Java . Nó định nghĩa dữ liệu và các hành vi của nó (dữ liệu và mã lệnh), gọi là các thành viên của lớp, dùng chung cho các đối tượng cùng loại . Từ sự phân tích hệ thống , người ta trừu tượng nên các lớp . Sau đó các đối tượng được tạo ra theo khuôn mẫu của lớp . Mỗi đối tượng thuộc một lớp có dữ liệu và hành vi định nghĩa cho lớp đó , giống như là sinh ra từ một khuôn đúc

của lớp đó. Vì vậy mà lớp là khuôn mẫu của đối tượng, đối tượng là thể hiện của một lớp. Lớp là cấu trúc logic, còn đối tượng là cấu trúc vật lý. Dữ liệu định nghĩa trong lớp gọi là biến, mã lệnh gọi là phương thức. Phương thức định nghĩa cho việc sử dụng dữ liệu như thế nào. Điều này có nghĩa là hoạt động của lớp được định nghĩa thông qua phương thức.

Các đặc trưng của lớp gồm có hai phần chính: thuộc tính (Attribute) và hành vi (Behavior). Giả sử bạn phải tạo ra giao diện với người dùng và cần có những nút nhấn (Button). Thế thì trước hết bạn xây dựng lớp Button với các thuộc tính như nhãn ghi trên nút, chiều rộng, chiều cao, màu của nút, đồng thời quy định hành vi của nút nhấn, nghĩa là nút nhấn cần phản ứng như thế nào khi được chọn, phát yêu cầu gì, có đổi màu hay nhấp nháy chi không. Với lớp Button như vậy, bạn có thể tạo ra nhanh chóng những nút nhấn cụ thể phục vụ cho các mục đích khác nhau

Gói là kỹ thuật của Java, dùng để phân hoạch không gian tên lớp, giao diện thành những vùng dễ quản lý hơn, thể hiện tính đóng gói của Java.

2. Tính kế thừa (Inheritance)

Tính kế thừa là khả năng xây dựng các lớp mới từ các lớp đã có. Tính đóng gói cũng tác động đến tính kế thừa. Khi lớp đóng gói một số dữ liệu và phương thức, lớp mới sẽ kế thừa mọi cấu trúc dữ liệu và các phương thức của lớp mà nó kế thừa. Ngoài ra nó có thể bổ sung các dữ liệu và các phương thức của riêng mình.

Nó rất quan trọng vì nó ứng dụng cho khái niệm cây phân cấp (mô hình TopDown). Không sử dụng cây phân lớp, mỗi lớp phải định nghĩa tất cả các dữ liệu và phương thức của mình một cách rõ ràng. Nếu sử dụng sự kế thừa, mỗi lớp chỉ cần định nghĩa thêm những đặc trưng của mình.

Ví dụ: Xe có thể xem như một lớp và các xe Pergout, BWM, Dream là các đối tượng của lớp xe. Các xe đều có thể lái đi, dừng lại... Từ lớp xe ở trên, ta có thể xây dựng các lớp xe đạp, xe ôtô. Xe ôtô có thêm máy và có thể tự khởi động…

3. Tính đa hình (Polymorphism)

Khi một lớp được kế thừa từ các lớp tổ tiên thì nó có thể thay đổi cách thức làm việc của lớp tổ tiên trong một số phương thức nào đó (nhưng tên, kiểu trả về, danh sách tham đối của phương thức thì vẫn giữ nguyên). Điều này gọi là viết chồng. Như vậy với một tên phương thức, chương trình có thể có các hành động khác nhau tùy thuộc vào lớp của đối tượng gọi phương thức. Đó là tính đa hình

Ví dụ : với một phương thức chạy , xe ôtô, xe máy có thể tăng ga , còn xe đạp thì phải đạp…

Tính đa hình còn thể hiện ở việc một giao diện có thể sử dụng cho các hoạt động của một lớp tổng quát , hay còn gọi là "một giao diện , nhiều phương thức" . Có nghĩa là có thể thiết kế một giao diện tổng quát cho một nhóm các hành vi liên quan . Điều này giảm thiểu sự phức tạp bằng cách cho phép một giao diện có thể sử dụng cho các hoạt động của một lớp tổng quát . Trình biên dịch sẽ xác định hoạt động cụ thể nào sẽ được thi hành tùy theo điều kiện . Bạn chỉ cần nhớ các giao diện của lớp tổng quát và sử dụng nó.

Sự kết hợp đúng đắn giữa : đa hình , đóng gói và kế thừa tạo nên một môi trường lập trình có khả năng phát triển tốt hơn rất nhiều so với môi trường không hỗ trợ hướng đối tượng. Một cây phân cấp lớp thiết kế tốt là điều căn bản cho việc sử dụng lại những đoạn mã lệnh mà bạn đã tốn công sức nhiều cho việc phát triển và kiểm tra . Tính đóng gói cho phép bạn sử dụng các đối tượng và ra lệnh thi hành tới chúng mà không phá vỡ cấu trú c các đoạn mã lệnh đã bảo vệ bởi giao diện của các lớp . Sự đa hình cho phép bạn tạo ra những đoạn mã lệnh gọn gàng , dễ đọc, dễ hiểu và có tính ổn định.

Java là ngôn ngữ lập trình hướng đối tượng nên có đầy đủ cá c tính năng trên , thư viện lớp Java được cung cấp khá đầy đủ cho người lập trình để bắt đầu một dự án mới

Chương : **ĐỐI TƯỢNG VÀ LỚP, MẢNG**

I. XÂY DỰNG LỚP

Khi định nghĩa một lớp , bạn chỉ ra thuộc tính mà nó chứa được thể hiện bằng biến (Member Variable) và hành vi được thể hiện bởi hàm (Method)

Các biến định nghĩa bên trong một lớp gọi là các biến thành viên (Member Variables). Mã lệnh chứa trong các phương thức (Method). Các phương thức và biến định nghĩa trong lớp gọi chung là thành phần của lớp . Trong hầu hết các lớp , các biến thể hiện được truy cập bởi các phương thức định nghĩa trong lớp đó . Vì vậy, chính các phương thức quyết định dữ liệu của lớp có thể dùng như thế nào . Lớp định nghĩa một kiểu dữ liệu mới, dùng để tạo các đối tượng thuộc kiểu đó.

Dạng đầy đủ của một định nghĩa lớp như sau :

[public]	Lớp được truy xuất chung cho các Package khác , mặc định chỉ có các đoạn mã trong cùng một gói mới có quyền truy xuất nó
[abstract]	Lớp trừu tượng, không thể khởi tạo
[final]	Lớp hằng không có lớp con , không kế thừa
class *ClassName*	Tên lớp
[extends *SuperClass*]	Kế thừa lớp cha SuperClass
[implements *Interfaces*]	Giao diện được cài đặt bởi Class
{ //Member Variables Declarations	Khai báo các biến
// Methods Declarations	Khai báo các phương thức
}	

Ví dụ : Tạo một lớp Box đơn giản với ba biến : width, height, depth

```
/* Định nghĩa lớp
*/
class Box {
        double width;
        double height;
```

```
        double depth;
    }
```

II. TẠO ĐỐI TƯỢNG

1. Khai báo đối tượng

Để có được các đối tượng của một lớp phải qua hai giai đoạn :

- ***ClassName ObjectName;***

Ví dụ : Box myBox

Khai báo biến myBox có kiểu lớp Box . Khai báo này thực ra không cấp phát ký ức đủ chứa đối tượng thuộc lớp Box , mà chỉ tạo ra quy chiếu trỏ đến đối tượng Box . Sau câu lệnh này, quy chiếu myBox xuất hiện trên ký ức chứa giá trị null chỉ ra rằng nó chưa trỏ đến một đối tượng thực tế nào

Khác với câu lệnh khai báo biến kiểu sơ cấp là dành chỗ trên ký ức đủ chứa một trị thuộc kiểu đó :

Ví dụ : int i;

Sau câu lệnh này, biến nguyên i hình thành.

- Sau đó, để thực sự tạo ra một đối tượng và gán địa chỉ của đối tượng cho biến này, dùng toán tử new

ObjectName* = new *ClassName();

Ví dụ : myBox = new Box();

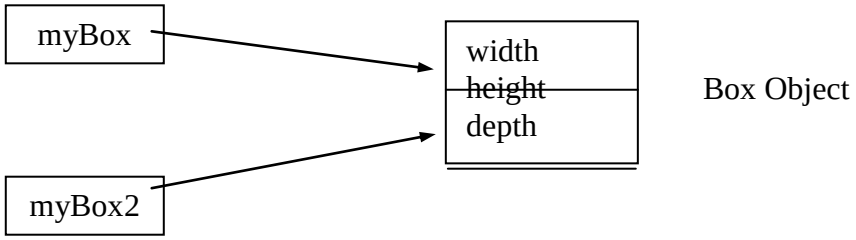
- Có thể kết hợp cả hai bước trên vào một câu lệnh :

ClassName ObjectName* = new *ClassName();

Ví dụ : Box myBox = new Box();

Box myBox2 = myBox;

myBox2 tham chiếu đến cùng đối tượng mà myBox tham chiếu



2. Cách truy xuất thành phần của lớp

◆ Biến khai báo trong định nghĩa lớp gồm có hai loại :

- Biến đối tượng (Instance Variable hay Object Variable) : chỉ thuộc tính đối tượng , khi truy xuất phải khởi tạo đối tượng

+ Cách khai báo biến đối tượng :

Type InstanceVar;

+ Cách truy cập biến đối tượng :

ObjectName.InstanceVar

- Biến lớp (Class Variable) : về bản chất là biến toàn cục , là biến tĩnh được tạo lập một lần cùng với lớp, dùng chung cho mọi đối tượng thuộc lớp, khi truy xuất không cần khởi tạo đối tượng, để trao đổi thông tin của các đối tượng cùng lớp

+ Cách khai báo biến lớp :

static *Type ClassVar;*

+ Cách truy cập biến lớp :

ClassName.ClassVar

◆ Hàm khai báo trong định nghĩa lớp gồm có hai loại :

- Hàm đối tượng (Object Method) : cách truy xuất hàm đối tượng như biến đối tượng

***ObjectName.ObjectMethod*(*Parameter-List*)**

- Hàm lớp (Class Method) : thông thường một thành phần của lớp chỉ truy xuất trong sự liên kết với một đối tượng thuộc lớp của nó . Tuy nhiên, có thể tạo ra một thành phần mà có thể dùng một độc lập một mình, không cần tham chiếu đến một đối tượng cụ thể, có thể được truy xuất trước khi bất kỳ đối tượng nào của lớp đó được tạo ra, bằng cách đặt trước khai báo của nó từ khoá static. Cách truy xuất hàm lớp :

***ClassName.ClassMethod*(*Parameter-List*)**

Các hàm toán học của lớp Math trong Package Java.Lang là hàm lớp nên khi gọi không cần phải khởi tạo đối tượng

Ví dụ : double a = Math.sqrt(453.28);

Ví dụ 1: class BaiTho {

```
static int i;              // Biến lớp
String s;                  // Biến đối tượng
BaiTho(String ss) {  // Hàm khởi tạo
        s = ss;
        i++;
```

```
        }
        void content( ) {
            System.out.println(s);
        }
    }
    class UngDung {
        public static void main(String args[]){
            BaiTho p1 = new BaiTho("Chi co thuyen moi hieu");
            BaiTho p2 = new BaiTho("Bien menh mong nhuong nao");
            p1.content();
            p2.content();
            System.out.println("So cau tho la : "+BaiTho.i);
        }
    }
```

Khi tạo đối tượng p1, p2 bởi toán tử new, hàm dựng BaiTho() được gọi, và i tăng lên 1

Ví dụ 2:

```
    class BaiTho2 {
        static int i;
        String s;
        BaiTho2(String ss) {// Hàm khởi tạo
            s = ss;i++;
        }
        static int number() { // Hàm lớp
            return i;
        }
        String content() {     // Hàm đối tượng
            return s;
        }
    }
    class UngDung2 {
        public static void main (String args[]) {
            System.out.println("Bai tho co "+BaiTho2.number()+" cau");
            BaiTho2.p1 = new BaiTho2("Chi co thuyen moi hieu");
```

```
                    BaiTho2.p2 = new BaiTho2("Bien menh mong nhuong nao");
                    System.out.println("Bai tho co "+BaiTho2.number()+" cau");
                    System.out.println("Cau    tho\n"+p1.content().toUpperCase()+"\nco"
+
                            p1.content().length() +" ky tu");
                    System.out.println("Tu \"tinh yeu\" bat dau sau ky tu thu"+
                            p2.content().indexOf("tinh yeu")+" trong cau\n"+
                            p2.content().toUpperCase());
            }
        }
```

Gọi hàm lớp BaiTho2.number() lúc chưa gọi hàm dựng BaiTho 2 để khởi tạo đối tượng sẽ cho trị 0

p1.content() trả về một đối tượng String

III. GIỚI THIỆU VỀ PHƯƠNG THỨC

1. Khai báo phương thức (hàm)

Dạng tổng quát của một phương thức như sau :

[*acess*]	**điều khiển truy xuất**
[static]	**hàm lớp**
[abstract]	**hàm trừu tượng**
[final]	**hàm hằng**
[*Type*] *MethodName*(*Parameter-List*) throws *exceptions* { // Body of method }	

- Type : Kiểu dữ liệu do hàm trả về, có thể là kiểu bất kỳ, kể cả các kiểu lớp do bạn tạo ra. Nếu hàm không trả về giá trị nào, kiểu trả về của nó phải là void.
- Các hàm có kiểu trả về không phải là void sẽ trả về một giá trị cho chương trình gọi nó dùng dạng câu lệnh return như sau :

return biểu thức;

Giá trị của biểu thức được tính và trả về cho hàm

- Tất cả thông tin bạn muốn truyền được gởi thông qua tham số nằm trong hai dấu ( ) ngay sau tên hàm. Nếu không có tham số vẫn phải có ( )

Parameter-List : Danh sách tham đối phân cách bởi các dấu phẩy , mỗi tham đối phải được khai báo kiểu, có thể là kiểu bất kỳ, có dạng : *Type Parameter1*, *Type Parameter2* ...

2. Phạm vi truy xuất thành phần của lớp

Các điều khiển truy xuất của Java là public, private và protected. protected chỉ áp dụng khi có liên quan đến kế thừa sẽ xét đến sau

Khi bổ sung tiền tố cho một thành phần của lớp (biến và hàm) là :

- Từ khoá public : chỉ ra rằng thành phần này có thể được truy xuất bởi bất kỳ dòng lệnh nào dù ở trong hay ngoài lớp mà nó khai báo
- private : chỉ có thể được truy xuất trong lớp của nó , mọi đoạn mã nằm ngoài lớp, kể cả những lớp con đều không có quyền truy xuất
- Khi không có điều khiển truy xuất nào được dùng, mặc nhiên là public nhưng chỉ trong gói của nó, không thể truy xuất từ bên ngoài gói của nó

3. Phương thức main()

Khi chạy ứng dụng độc lập , bạn chỉ tên Class muốn chạy , Java tìm gọi hàm main() trước tiên trong Class đó , phương thức main sẽ điều khiển chạy các phương thức khác.

Dạng tổng quát của phương thức main()

```
public static void main(String args[]) {
        // Body of  Method
}
```

- Một chương trình chỉ cần một lớp có phương thức main () gọi là lớp ứng dụng độc lập Primary Class.
- Từ khoá static cho phép hàm main () được gọi khi không cần khởi tạo đối tượng . Vì main() được trình thông dịch của Java gọi trước khi bất kỳ lớp nào được khởi tạo
- Từ khoá void cho biết hàm main() không trả về giá trị
- Từ khoá public chỉ ra rằng hàm này được gọi bởi dòng lệnh bên ngoài lớp khi chương trình khởi động.
- Tham đối String args[ ] khai báo tham số tên args thuộc lớp String , chứa chuỗi ký tự .