BÀI THỰC HÀNH 4

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG JAVA (buổi 1)

I/ LÝ THUYẾT

**1. Lớp**

- Lớp là khuôn mẫu mô tả đối tượng, bao gồm các thuộc tính và các phương thức của đối tượng.

- Khai báo lớp

Khi mô tả lớp phải xác định được: *dữ liệu và phương thức của lớp*

*Dữ liệu:* là các thành phần dữ liệu tạo thành lớp, mô tả các thuộc tính của dữ liệu

*Phương thức:* là các thao tác, hành động được thực hiện trên đối tượng

\* Cú pháp khai báo lớp:

<**modifier**> **class** <*class\_name*>

**{**

//*Khai báo dữ liệu của lớp*

//*Khai báo phương thức của lớp*

**}**

Trong đó:

**<modifier**>: xác định thuộc tính của lớp, có thể nhận các giá trị: **public, final, abstract**

**<***class\_name***>**: tên của lớp cần định nghĩa

***Khai báo dữ liệu***: liệt kê các thành phần dữ liệu của lớp, được mô tả như khai báo biến

***Khai báo phương thức:*** liệt kê các phương thức của lớp, được mô tả như khai báo hàm và khai báo phương thức

\* Ví dụ:

class HCN

{

public int a;

public int b;

public int dientich( )

{ return (a\*b); }

public int chuvi( )

{ return (a+b)\*2; }

public static void main( String arg[])

{

…

}

}

**\* Lưu ý:**

Có hai loại Modifier trong Java, đó là: Access Modifier và Non-access Modifier. Access Modifer trong Java xác định phạm vi có thể truy cập của thành viên dữ liệu, phương thức, constructor hoặc lớp. Có 4 loại Access Modifier là: private, default, protected, và public.

* Default: Truy cập trong nội bộ package
* Private: Truy cập trong nội bộ lớp
* Public: Thành phần công khai, truy cập tự do từ bên ngoài
* Protected: Thành phần được bảo vệ, bị hạn chế truy nhập từ bên ngoài

Ngoài ra còn có nhiều Non-access Modifier như static, abstract, synchronized, native, volatile, transient, … Chương này chúng ta sẽ tìm hiểu về Access Modifier.

Theo dõi bảng sau để có cái nhìn sơ lược về các loại Access Modifier trong Java:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **Bên trong lớp** | **Bên trong package** | **Bên ngoài package chỉ bởi lớp con** | **Bên ngoài package** |
| private | C | K | K | K |
| default | C | C | K | K |
| protected | C | C | C | K |
| public | C | C | C | C |

**+ Chế độ truy cập Private**

Các phương thức, biến và constructor mà được khai báo private chỉ có thể được truy cập trong chính lớp được khai báo đó.

Private Access Modifier là chế độ truy cập mang tính hạn chế nhất. Lớp và interface không thể là private.

Các biến được khai báo private có thể được truy cập bên ngoài lớp nếu phương thức public getter có mặt trong lớp đó.

Sử dụng Private Access Modifier trong Java là cách chủ yếu để một đối tượng bao đóng chính nó và ẩn dữ liệu với bên ngoài.

Trong ví dụ dưới đây, chúng ta tạo hai lớp A và Simple. Lớp A chứa thành viên dữ liệu private và phương thức private. Chúng ta đang truy cập các thành viên private này từ bên ngoài lớp, và điều này dẫn đến một Compile time error:

class A{

private int data=40;

private void msg(){System.out.println("Hello java");}

}

public class Simple{

public static void main(String args[]){

A obj=new A();

System.out.println(obj.data);//Compile Time Error

obj.msg();//Compile Time Error

}

}

**+ Chế độ truy cập Default:**

Default Access Modifier nghĩa là chúng ta không khai báo một cách rõ ràng một Access Modifier cho một lớp, trường, phương thức, ... Nói cách khác, nếu bạn không sử dụng bất cứ Modifier nào, thì theo mặc định nó được xem như là default. Default Modifier là chỉ có thể truy cập bên trong package.

Một biến hoặc phương thức được khai báo mà không có bất kỳ Access Modifier nào là có sẵn cho mọi lớp trong cùng package. Các trường này trong một interface là hoàn toàn public static final và các phương thức trong một interface là public theo mặc định.

**Ví dụ**

Trong ví dụ này, chúng ta tạo hai package là pack và mypack. Chúng ta đang truy cập lớp A từ bên ngoài package của nó. Khi lớp A không là public, thì nó không thể được truy cập từ bên ngoài package.

//Luu duoi dang A.java

package pack;

class A{

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();//Compile Time Error

obj.msg();//Compile Time Error

}

}

Trong ví dụ trên, phạm vi của lớp A và phương thức msg() của nó là default, vì thế nó không thể được truy cập từ bên ngoài package.

**+ Chế độ Protected**

Protected Access Modifier là có thể truy cập bên trong package và bên ngoài package nhưng chỉ thông qua tính kế thừa. Protected Access Modifier có thể được áp dụng trên thành viên dữ liệu. Nó không thể được áp dụng trên lớp. Các biến, phương thức và constructor, mà được khai báo protected trong một lớp cha (superclass), chỉ được truy cập bởi các lớp cha trong package khác hoặc bất kỳ lớp nào bên trong package đó của lớp được protected.

Protected Access Modifier không thể được áp dụng cho lớp và interface. Các phương thức và trường có thể được khai báo protected, tuy nhiên, các phương thức và trường trong một interface không thể được khai báo là protected.

Chế độ protected cung cấp cho lớp phụ cơ hội để sử dụng phương thức hoặc biến helper, trong khi ngăn cản một lớp không liên quan từ việc cố gắng sử dụng nó.

**Ví dụ**

Trong ví dụ này, chúng ta tạo hai package là pack và mypack. Một lớp A của pack package là public, vì thế có thể được truy cập từ bên ngoài package. Nhưng phương thức msg của package này được khai báo là protected, vì thế nó có thể được truy cập từ bên ngoài lớp nhưng chỉ thông qua tính kế thừa.

//Luu duoi dang A.java

package pack;

public class A{

protected void msg(){System.out.println("Hello");}

}

//Luu duoi dang B.java

package mypack;

import pack.\*;

class B extends A{

public static void main(String args[]){

B obj = new B();

obj.msg();

}

}

## + Chế độ Public

Public Access Modifier là có thể truy cập ở bất cứ đâu. Nó có phạm vi rộng nhất trong tất cả Modifier. Một lớp, phương thức, constructor, interface, ... được khai báo public có thể được truy cập từ bất cứ lớp nào khác. Do đó, các trường, phương thức và khối được khai báo bên trong một lớp public có thể được truy cập từ bất kỳ lớp nào trong thế giới Java.

Tuy nhiên, nếu lớp public chúng ta đang cố gắng truy cập là trong một package khác, thì lớp public này vẫn cần để được import.

Bởi vì tính kế thừa lớp, tất cả phương thức và biến của một lớp được kế thừa bởi các lớp phụ của nó.

**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();

}

}

**2. Phương thức**

*Khái niệm:*

Hàm trong Java là các khối lệnh thực hiện các chức năng riêng biệt => *Hàm luôn gắn liền vào các lớp đối tượng nên hàm trong java được gọi là phương thức*

Cú pháp:

<access\_specifier> <modifier> <datatype> **<name\_function>** (parameter)

{

…..//nội dung định nghĩa phương thức

}

Trong đó:

***<access\_specifier>*:** chỉ định cách thức truy nhập vào phương thức ⬄ dùng để giới hạn khả năng truy cập từ bên ngoài vào phương thức được định nghĩa. Có thể nhận các giá trị: *private, protected, public*

* *private*: riêng tư - chỉ có thể được truy nhập được nhờ các phương thức public trong cùng lớp
* *protected:* cho phép các lớp mở rộng(kế thừa) có thể truy nhập tới các phương thức này
* *public:* phương thức có thể truy xuất từ mọi nơi và mọi lúc

***<modifier>***: xác định thuộc tính cho các phương thức, có các giá trị sau:

* *static*: cho phép phương thức được gọi đến mà không cần đối tượng, phương thức này chỉ được sử dụng từ các dữ liệu và phương thức tĩnh khác
* *abstract*: ngụ ý phương thức không có mã trong đó, sẽ được bổ sung ở các lớp sau
* *final:* phương thức được bảo vệ không cho các lớp dẫn xuất khai báo và định nghĩa lại
* *native:* chỉ ra rằng phần thân hàm có thể được viết bằng một ngôn ngữ khác java (C, C++,…), phương thức này chỉ có tên và khối lệnh rỗng
* *synchronized:* đảm bảo dữ liệu không bị sai lạc khi dùng cùng lúc hai phương thức truy cập một DL

***<datatype>:*** xác định kiểu mà kết quả phương thức trả lại*( int, float,…),* không có giá trị trả về thì dùng kiểu **void**

***<name\_function>*:** tên của phương thức được định nghĩa

***<parameter>***: xác định và chứa các đối số của phương thức, đối số được mô tả như khai báo biến. Các đối số phân cách nhau bằng dấu phẩy

\* Ví dụ:

class callmethod

{

public static int S\_Hinhvuong( int x)

{

return x\*x;

}

public static void main(String arg[])

{

int canh=5, s;

s = S\_Hinhvuong( canh );

System.out.println( “Dien tich:” + s );

}

}

**3. Phương thức khởi tạo lớp**

Constructor trong Java là một kiểu phương thức đặc biệt mà được sử dụng để khởi tạo đối tượng. Constructor được triệu hồi tại thời gian tạo đối tượng. Nó xây dựng giá trị, cung cấp dữ liệu cho đối tượng, đó là lý do nó được gọi là Constructor.

Khi bàn luận về các lớp, một trong những chủ đề quan trọng là các constructor. Mỗi lớp có ít nhất 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. Đặc điểm phương thức khởi tạo:

* *Phương thức khởi tạo trùng tên với tên của lớp*
* *Phương thức không có kiểu trả về*
* *Một lớp có thể có nhiều phương thức khởi tạo*
* *Phương thức sẽ tự động được gọi đến khi tạo ra đối tượng*

**+ Cú pháp:**

*<tên\_lớp>(){}*

hoặc

*<tên lớp> (d/s tham số){}*

**+ Trong đó:**

- Tên Contructor phải trùng tên lớp

- Contructor không có kiểu trả về, phạm vi khai báo thường là public.

- Một lớp có thể có nhiều Contructor, nếu lớp không có Contructor thì Java tự xây dựng một Contructor cho lớp, nếu lớp có Contructor thì Java sử dụng Contructor đã xây dựng

+ Ví dụ 1:

**public** **class** Exam2 {

**private** **int** x,y;

Exam2(){}

Exam2(**int** a, **int** b)

{

x=a;

y=b;

}

**public** **void** Display()

{

System.*out*.println("x="+x+", y="+y);

}

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

// **TODO** Auto-generated method stub

Exam2 E=**new** Exam2();

Exam2 F= **new** Exam2(2,3);

E.Display();//In gia tri mac dinh cua x,y

F.Display();//In gia tri duoc truyen vao khi khoi tao

}

}

*+ Ví dụ 2: Xây dựng lớp HCN. AD tạo một HCN và in ra diện tích HCN*

class HCN

{

private int a;

private int b;

**HCN ( int aa, int bb)**

**{**

**a = aa; b = bb;**

**}**

int dientich()

{ return a\*b; }

static public void main()

{

HCN x = new HCN(4,5);

int dt;

dt = x.dientich();

system.out.println( “Dien tich HCN:” + dt );

}

}

**4/ Nạp chồng**

Nếu một lớp có nhiều phương thức cùng tên nhưng có số tham số khác nhau, thì đó là nạp chồng phương thức (Method Overloading). Nếu bạn phải thực hiện chỉ một hoạt động, có cùng tên phương thức, thì kỹ thuật này làm tăng tính có thể đọc cho chương trình. Giả sử bạn viết phương thức như a(int, int) cho hai tham số, và b(int, int, int) cho ba tham số, thì khi đó điều này có thể gây khó khăn cho bạn cũng như cho các lập trình viên khác để hiểu hành vi của phương thức, bởi vì tên của nó là khác nhau. Vì thế, chúng ta thực hiện nạp chồng phương thức để giúp việc phân tích chương trình nhanh hơn.

Có hai cách để nạp chồng phương thức trong Java, đó là:

* Bằng việc thay đổi số tham số
* Bằng việc thay đổi kiểu dữ liệu

**public** **class** Exam1 {

**private** **int** x;

**public** **int** Tong(**int** a, **int** b)

{

**return** (a+b);

}

//Nap chong khac kieu

**public** **float** Tong(**float** a, **float** b)

{

**return** (a+b);

}

//Nap chong them tham so

**public** **int** Tong(**int** a, **int** b, **int** c)

{

**return** (a+b+c);

}

//Nap chong ham main

**public** **static** **void** main(**int** a)

{

System.*out*.println("ham main duoc nap chong:"+a);

}

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

**int** a=10, b=11, c=12;

**float** d=11, e=12;

Exam1 E=**new** Exam1();

System.*out*.println("Tong 1="+E.Tong(a,b));

System.*out*.println("Tong 2="+E.Tong(d,e));

System.*out*.println("Tong 3="+E.Tong(a,b,c));

E.*main*(10);

}

}

**Chú ý:**

+ Trong Java, nạp chồng phương thức là không thể bằng việc thay đổi kiểu trả về của phương thức

+ Cho phép nạp chồng cả hàm main()

II/ BÀI TẬP THỰC HÀNH TẠI LỚP:

**Bài 1:** Xây dựng lớp phân số, yêu cầu định nghĩa:

* Nhập
* Xuất
* Tính tổng hai phân số
* Tính hiệu hai phân số
* Tính tích hai phân số
* Tính thương của hai phân số

**Bài 2:** Thông tin về sinh viên gồm: Mã sinh viên, họ tên, điểm. Xây dựng lớp sinh viên, viết chương trình quản lý danh sách sinh viên với các yêu cầu sau:

* Nhập danh sách sinh viên
* Hiển thị danh sách sinh viên
* Tìm kiếm sinh viên theo tên
* Sắp xếp danh sách sinh viên theo chiều tăng dần của điểm.
* Thêm một sinh viên vào danh sách đã sắp xếp mà không làm thay đổi tính sắp xếp của danh sách
* Loại bỏ sinh viên có mã nhập vào từ bàn phím

Chú ý: viết chương trình theo hai cách: sử dụng mảng đối tượng và mảng có sẵn.