Session 06:

JAVA Modifiers and Packages

# Từ khóa static trong Java

## Biến static trong Java

### lấy bộ nhớ chỉ một lần, sử dụng hiệu quả hơn (tiết kiệm bộ nhớ).

### **Ví dụ về biến static trong Java**

//Chuong trinh vi du ve bien static trong Java

class Student8{

int rollno;

String name;

static String college ="BachKhoa";

Student8(int r,String n){

rollno = r;

name = n;

}

void display (){System.out.println(rollno+" "+name+" "+college);}

public static void main(String args[]){

Student8 s1 = new Student8(111,"Hoang");

Student8 s2 = new Student8(222,"Thanh");

s1.display();

s2.display();

}

}

### **Chương trình Counter mà không sử dụng biến static**

class Counter{

int count=0; //se lay bo nho (memory) khi bien instance duoc tao

//Ket qua thuc hien chuong trinh hien ra 3 so 1 o 3 dong

Counter(){

count++;

System.out.println(count);

}

public static void main(String args[]){

Counter c1=new Counter();

Counter c2=new Counter();

Counter c3=new Counter();

}

}

### **Chương trình counter với biến static trong Java**

class Counter2{

static int count=0; //se lay bo nho chi mot lan và giu lai gia tri cua no

//ket qua thuc hien in ra 3 dong cac gia tri : 1,2,3

Counter2(){

count++;

System.out.println(count);

}

public static void main(String args[]){

Counter2 c1=new Counter2();

Counter2 c2=new Counter2();

Counter2 c3=new Counter2();

}

}

## Phương thức static trong Java

### Nếu bạn áp dụng từ khóa static với bất cứ phương thức nào, thì phương thức đó được gọi là phương thức static.

### Một phương thức static thuộc lớp chứ không phải đối tượng của lớp.

### Một phương thức static có thể được triệu hồi mà không cần tạo một instance của một lớp.

### Phương thức static có thể truy cập thành viên dữ liệu static và có thể thay đổi giá trị của nó.

### **Ví dụ về phương thức static trong Java**

//Chuong trinh thay doi thuoc tinh chung cua tat ca doi tuong (truong static).

class Student9{

int rollno;

String name;

static String college = "BachKhoa";

static void change(){

college = "QuocGia";

}

Student9(int r, String n){

rollno = r;

name = n;

}

void display (){System.out.println(rollno+" "+name+" "+college);}

public static void main(String args[]){

Student9.change();

Student9 s1 = new Student9 (111,"Hoang");

Student9 s2 = new Student9 (222,"Thanh");

Student9 s3 = new Student9 (333,"Nam");

s1.display();

s2.display();

s3.display();

}

}

### **Ví dụ khác về phương thức static mà thực hiện phép tính toán thông thường**

//Chuong trinh lay cube (gia tri lap phuong) cua so da cho boi phuong thuc static

class Calculate{

static int cube(int x){

return x\*x\*x;

}

public static void main(String args[]){

int result=Calculate.cube(5);

System.out.println(result);

}

}

### **Một số hạn chế cho phương thức static**

Có hai hạn chế chính cho phương thức static. Đó là:

* Phương thức static không thể sử dụng thành viên dữ liệu non-static hoặc gọi trực tiếp phương thức non-static.
* Từ khóa this và super không thể được sử dụng trong ngữ cảnh static.

class A{

int a=40;//non static

public static void main(String args[]){

System.out.println(a);

}

}

Chạy chương trình trên sẽ cho kết quả là Compile Time Error.

**Câu hỏi**: Tại sao phương thức main trong Java là static?

Bởi vì đối tượng là không cần thiết để gọi phương thức static nếu nó là phương thức non-static, JVM đầu tiên tạo đối tượng và sau đó gọi phương thức main() mà có thể gây ra vấn đề về cấp phát bộ nhớ bộ nhớ phụ.

### Khối static trong Java

Được sử dụng để khởi tạo thành viên dữ liệu static. Nó được thực thi trước phương thức main tại thời gian tải lớp. Dưới đây là ví dụ về khối static trong Java:

class A2{

static{System.out.println("Khoi static duoc trieu hoi");}

public static void main(String args[]){

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

}

}

**Câu hỏi**: Chúng ta có thể thực thi một chương trình mà không có phương thức main()?

Có, một trong các cách đó là khối static trong phiên bản trước của JDK, không trong JDK 1.7.

class A3{

static{

System.out.println("Khoi static duoc trieu hoi");

System.exit(0);

}

}

# Từ khóa final trong Java

## Biến final trong Java

Nếu bạn tạo bất cứ biến nào là final, bạn không thể thay đổi giá trị của biến final (nó sẽ là hằng số).

### **Ví dụ của biến final trong Java**

Giả sử có một biến final có tên là speedlimit, chúng ta đang thay đổi giá trị của biến này, nhưng nó không thể bị thay đổi bởi vì một khi biến final đã được gán giá trị thì không bao giờ bị thay đổi.

class Bike9{

final int speedlimit=90;//bien final

void run(){

speedlimit=400;

}

public static void main(String args[]){

Bike9 obj=new Bike9();

obj.run();

}

}//phan cuoi cua lop

## Phương thức final trong Java

Nếu bạn tạo bất cứ phương thức nào là final, thì bạn không thể ghi đè nó.

Ví dụ của phương thức final

class Bike{

final void run(){System.out.println("running");}

}

class Honda extends Bike{

void run(){System.out.println("Chay an toan voi 100kmph");}

public static void main(String args[]){

Honda honda= new Honda();

honda.run();

}

}

## Lớp final trong Java

Nếu bạn tạo bất cứ lớp nào là final thì bạn không thể kế thừa nó.

final class Bike{}

class Honda1 extends Bike{

void run(){System.out.println("Chay an toan voi 100kmph");}

public static void main(String args[]){

Honda1 honda= new Honda();

honda.run();

}

}

**Câu hỏi**: Phương thức final có được kế thừa không?

Có, phương thức final được kế thừa nhưng bạn không thể ghi đè nó. Ví dụ:

class Bike{

final void run(){System.out.println("dang chay...");}

}

class Honda2 extends Bike{

public static void main(String args[]){

new Honda2().run();

}

}

**Câu hỏi**: Biến final trống hoặc không được khởi tạo là gì?

Một biến final mà không được khởi tạo tại thời điểm khai báo được gọi là biến final trống. Nếu bạn muốn tạo một biến mà được khởi tạo tại thời điểm tạo đối tượng và một khi nó đã được khởi tạo thì không thể bị thay đổi, thì biến final trống là hữu ích trong trường hợp này. Ví dụ như số thẻ PAN CARD của một nhân viên.

Nó chỉ có thể được khởi tạo trong Constructor. Sau đây là ví dụ về biến final trống:

class Student{

int id;

String name;

final String PAN\_CARD\_NUMBER;

...

}

**Câu hỏi**: Chúng ta có thể khởi tạo biến final trống không?

Có, nhưng chỉ trong Constructor. Ví dụ:

class Bike10{

final int speedlimit;//bien final trong

Bike10(){

speedlimit=70;

System.out.println(speedlimit);

}

public static void main(String args[]){

new Bike10();

}

}

### **Biến static final trống trong Java**

### Một biến static final mà không được khởi tạo tại thời điểm khai báo thì đó là biến static final trống. Nó chỉ có thể được khởi tạo trong khối static.

Dưới đây là ví dụ về biến static final trống trong Java:

class A{

static final int data;//bien static final trong

static{ data=50;}

public static void main(String args[]){

System.out.println(A.data);

}

}

**Câu hỏi**: Tham số final là gì?

Nếu bạn khai báo bất cứ tham số nào là final, thì bạn không thể thay đổi giá trị của nó.

class Bike11{

int cube(final int n){

n=n+2;//khong the duoc thay doi khi n la final

n\*n\*n;

}

public static void main(String args[]){

Bike11 b=new Bike11();

b.cube(5);

}

}

Chạy chương trình trên sẽ cho Compile Time Error.

**Câu hỏi**: Chúng ta có thể khai báo một constructor final không?

Không, bởi vì constructor không bao giờ được kế thừa.

# Access Modifier trong Java

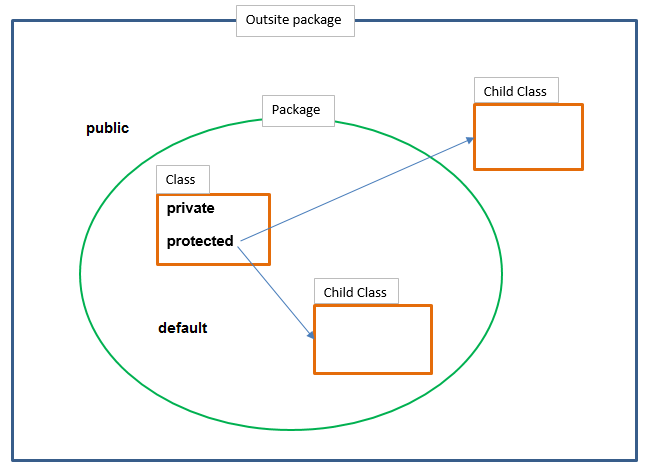
### Có hai loại Modifier trong Java, đó là: Access Modifier và Non-access Modifier. Access Modifer trong Java xác định phạm vi có thể truy cập của thành viên dữ liệu, phương thức, constructor hoặc lớp. Có 4 loại Access Modifier là: private, default, protected, và public.

#### 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



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

## Private Access Modifier trong Java

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

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

}

}

### **Qui tắc cho Private Access Modifier trong Java**

Nếu bạn tạo bất cứ constructor nào là private, bạn không thể tạo instance (sự thể hiện) của lớp đó từ bên ngoài lớp. Ví dụ:

class A{

private A(){}//private constructor

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

}

public class Simple{

public static void main(String args[]){

A obj=new A();//Compile Time Error

}

}

## Default Access Modifier trong Java

### Default Modifier là chỉ có thể truy cập bên trong package.

### **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.

## Protected Access Modifier trong Java

Protected Access Modifier là có thể truy cập bên trong package và bên ngoài package nhưng chỉ thông qua tính kế thừa

### **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();

}

}

## Public Access Modifier trong Java

Public Access Modifier là có thể truy cập ở bất cứ đâu.

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();

}

}

## Access Modifier trong Java với Ghi đè phương thức

### Nếu bạn đang ghi đè bất cứ phương thức nào, phương thức được ghi đè (ví dụ được khai báo trong lớp con) phải không nhiều giới hạn.

class A{

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

}

public class Simple extends A{

void msg(){System.out.println("Hello java");}//Compile Time Error

public static void main(String args[]){

Simple obj=new Simple();

obj.msg();

}

}

### Default Modifier là nhiều giới hạn hơn protected. Đó là lý do tại sao đây là compile time error.

## Access Modifier và tính kế thừa trong Java

Các qui tắc sau là bắt buộc cho các phương thức được kế thừa trong Java:

* Các phương thức được khai báo public trong một lớp cha cũng phải là public trong tất cả lớp phụ.
* Các phương thức được khai báo protected trong một lớp cha phải hoặc là protected hoặc public trong các lớp phụ; chúng không thể là private.
* Các phương thức được khai báo mà không có điều khiển truy cập (không sử dụng modifier nào) có thể được khai báo private trong các lớp phụ.
* Các phương thức được khai báo private không được kế thừa, do đó không có qui tắc nào cho chúng.

# Package 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.

## Ví dụ:

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

## Từ khóa import trong Java

package payroll;

public class Boss

{

public void payEmployee(Employee e)

{

e.mailCheck();

}

}

* 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?

### **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.packageA.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.packageA.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{}

### **Cách đặt hai lớp public trong một package?**

Nếu bạn muốn đặt hai lớp public trong một package, bạn có hai java source file chứa một lớp public nhưng giữ tên package là giống nhau. Ví dụ:

//Luu duoi dang A.java

package packageA;

public class A{}

//Luu duoi dang B.java

package packageA;

public class B{}