第01天 java面向对象

今日内容介绍

* 包和权限修饰符
* 内部类

# 包和权限修饰符

## 包的概述

java的包，其实就是我们电脑系统中的文件夹，包里存放的是类文件。

当类文件很多的时候，通常我们会采用多个包进行存放管理他们，这种方式称为分包管理。

在项目中，我们将相同功能的类放到一个包中，方便管理。并且日常项目的分工也是以包作为边界。

## 包的声明格式

通常使用公司网址反写，可以有多层包，包名采用全部小写字母，多层包之间用”.”连接

类中包的声明格式：

package 包名.包名.包名…;

如：黑马程序员网址itheima.com那么网址反写就为com.itheima

传智播客 itcast.cn 那么网址反写就为 cn.itcast

注意：声明包的语句，必须写在程序有效代码的第一行（注释不算）

代码演示：

**package** com.itheima\_01;

/\*

\* 包的特点：

\* 可以有多层

\* 不同包下的文件名可以重复

\* 包的声明必须是第一行代码

\*

\*/

**public** **class** PackageDemo {

}

## 包之间互相访问

在访问类时，为了能够找到该类，必须使用含有包名的类全名（包名.类名）。

包名.包名….类名

如： java.util.Scanner

java.util.Random

cn.itcast.Demo

带有包的类，创建对象格式：包名.类名 变量名 = new包名.类名();

cn.itcast.Demo d = new cn.itcast.Demo();

前提：包的访问与访问权限密切相关，这里以一般情况来说，即类用public修饰的情况。

类的简化访问

当我们要使用一个类时，这个类与当前程序在同一个包中（即同一个文件夹中），或者这个类是java.lang包中的类时通常可以省略掉包名，直接使用该类。

我们每次使用类时，都需要写很长的包名。很麻烦，我们可以通过import导包的方式来简化。

可以通过导包的方式使用该类，可以避免使用全类名编写（即，包类.类名）。

导包的格式：

import 包名.类名;

### 案例代码一:

**package** com.itheima\_01;

**import** java.util.ArrayList;

/\*

\*

\* 不同包之间的互相访问

\* 使用类的全名

\* 使用关键字import将类导入

\*

\*

\* 注意：\*代表的是通配符，代表导入了这个包下所有的类，并没有导入子包下的类

\*

\* 类的全名：包名.类名

\*

\*

\*/

**public** **class** PackageDemo2 {

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

//相同包下的类可以直接访问，不需要做其他的操作

//PackageDemo pd = new PackageDemo();

java.util.ArrayList list = **new** java.util.ArrayList();

ArrayList list2 = **new** ArrayList();

}

}

## 权限修饰符

在Java中提供了四种访问权限，使用不同的访问权限时，被修饰的内容会有不同的访问权限，以下表来说明不同权限的访问能力：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | public | protected | default | private |
| 同一类中 | √ | √ | √ | √ |
| 同一包中(子类与无关类) | √ | √ | √ |  |
| 不同包的子类 | √ | √ |  |  |
| 不同包中的无关类 | √ |  |  |  |

归纳一下：在日常开发过程中，编写的类、方法、成员变量的访问

A:要想仅能在本类中访问使用private修饰

B:要想本包中的类都可以访问除了private修饰符,其它都可以

C:要想本包中的类与其他包中的子类可以访问使用protected修饰

D:要想所有包中的所有类都可以访问使用public修饰。

注意：如果类用public修饰，则类名必须与文件名相同。一个文件中只能有一个public修饰的类。

### 案例代码二:

**package** com.itheima\_02;

/\*

\* 权限修饰符：

public 当前类，相同包下不同的类,不同包下的类

default 当前类，相同包下不同的类

private 当前类

protected 当前类，相同包下不同的类

default:当前包下使用

protected:让子类对象使用

\*

\*/

**public** **class** PermissionsDemo {

**public** **void** publicMethod() {

System.*out*.println("publicMethod");

}

**void** defaultMethod() {

System.*out*.println("defaultMethod");

}

**private** **void** privateMethod() {

System.*out*.println("privateMethod");

}

**protected** **void** protectedMethod() {

System.*out*.println("protectedMethod");

}

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

PermissionsDemo pd = **new** PermissionsDemo();

pd.publicMethod();

pd.defaultMethod();

pd.privateMethod();

pd.protectedMethod();

}

}

**package** com.itheima\_02;

**public** **class** PermissionsDemo2 {

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

PermissionsDemo pd = **new** PermissionsDemo();

pd.publicMethod();

pd.defaultMethod();

//pd.privateMethod();

pd.protectedMethod();

}

}

**package** com.itheima\_03;

**import** com.itheima\_02.PermissionsDemo;

**public** **class** PermissionsDemo3 {

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

PermissionsDemo pd = **new** PermissionsDemo();

pd.publicMethod();

//pd.defaultMethod();

//pd.privateMethod();

//pd.protectedMethod();

}

}

**package** com.itheima\_03;

**import** com.itheima\_02.PermissionsDemo;

**public** **class** PermissionsDemo4 **extends** PermissionsDemo {

**public** **void** function() {

**super**.publicMethod();

**super**.protectedMethod();

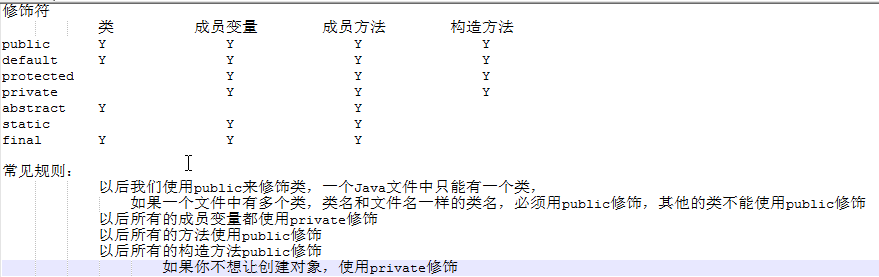
}

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

}

}

## 修饰符总结



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 修饰符总结 | | | | |
| 修饰符 | 类 | 成员变量 | 成员方法 | 构造方法 |
| public | Y | Y | Y | Y |
| default | Y | Y | Y | Y |
| protected |  | Y | Y | Y |
| private |  | Y | Y | Y |
| abstract | Y |  | Y |  |
| static |  | Y | Y |  |
| final | Y | Y | Y |  |

# 内部类

## 内部类概述

A:什么是内部类

将类写在其他类的内部，可以写在其他类的成员位置和局部位置，这时写在其他类内部的类就称为内部类。其他类也称为外部类。

B:什么时候使用内部类

在描述事物时，若一个事物内部还包含其他可能包含的事物，比如在描述汽车时，汽车中还包含这发动机，这时发动机就可以使用内部类来描述。

class 汽车 { //外部类

class 发动机 { //内部类

}

}

## 成员内部类

成员内部类，定义在外部类中的成员位置。与类中的成员变量相似，可通过外部类对象进行访问

A:定义格式

class 外部类 {

修饰符 class 内部类 {

//其他代码

}

}

B:访问方式

外部类名.内部类名 变量名 = new 外部类名().new 内部类名();

### 案例代码三

**package** com.itheima\_01;

/\*

\* 成员内部类：

\* 在类的成员位置，和成员变量以及成员方法所在的位置是一样的

\* 在内部类当中，可以直接访问外部类的成员，包括私有成员

\*/

**public** **class** InnerDemo {

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

//Outer o = new Outer();

//o.method();

Outer.Inner i = **new** Outer().**new** Inner();

i.function();

}

}

**class** Outer {

**private** **int** num = 10;

**public** **void** method() {

Inner i = **new** Inner();

i.function();

}

**class** Inner {

**public** **void** function() {

System.*out*.println(num);

}

}

}

成员内部类

成员内部类可以使用的修饰符:private,public,procted,final,static,abstract

### 案例代码四

**package** com.itheima\_01;

/\*

\* 成员内部类的修饰符：

\* 我们可以使用权限修饰符修饰成员内部类，但是如果使用私有来修饰，则无法在其他类中访问

\* 我们可以使用static修饰成员内部类，不用再创建外部类的对象了

\*

\* 我们可以使用abstract,final修饰成员内部类

\*/

**public** **class** InnerDemo2 {

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

//Outer2.Inner2 i;

//Outer2.Inner2 i = new Outer2.Inner2();

//i.function();

Outer2.Inner2.*function*();

}

}

**class** Outer2 {

**public** **void** method() {

Inner2 i = **new** Inner2();

}

**static** **class** Inner2 {

**public** **static** **void** function() {

System.*out*.println("function");

}

}

}

## 局部内部类

局部内部类，定义在外部类方法中的局部位置。与访问方法中的局部变量相似，可通过调用方法进行访问

A:定义格式

class 外部类 {

修饰符 返回值类型 方法名(参数) {

class 内部类 {

//其他代码

}

}

}

B:访问方式

在外部类方法中，创建内部类对象，进行访问

### 案例代码五:

**package** com.itheima\_02;

/\*

\* 局部内部类

\* 在方法内，出了方法之后就无法使用

\*

\*

\*/

**public** **class** InnerDemo3 {

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

Outer o = **new** Outer();

o.method();

}

}

**class** Outer {

**public** **void** method() {

**int** num = 10;

**class** Inner {

**public** **void** function() {

System.*out*.println("function");

}

}

Inner i = **new** Inner();

i.function();

}

**public** **void** test() {

//Inner i = new Inner();

//System.out.println(num);

}

}

## 匿名内部类

A:作用：匿名内部类是创建某个类型子类对象的快捷方式。

B:格式：

new 父类或接口(){

//进行方法重写

};

代码演示

//已经存在的父类：

public abstract class Person{

public abstract void eat();

}

//定义并创建该父类的子类对象，并用多态的方式赋值给父类引用变量

Person p = new Person(){

public void eat() {

System.out.println(“我吃了”);

}

};

//调用eat方法

p.eat();

使用匿名对象的方式，将定义子类与创建子类对象两个步骤由一个格式一次完成，。虽然是两个步骤，但是两个步骤是连在一起完成的。

匿名内部类如果不定义变量引用，则也是匿名对象。代码如下：

new Person(){

public void eat() {

System.out.println(“我吃了”);

}

}.eat();

### 案例代码六:

**package** com.itheima\_03;

/\*

\* 匿名内部类:

\* 可以把匿名内部类看成是一个没有名字的局部内部类

\* 定义在方法当中

\* 必须在定义匿名内部类的时候创建他的对象

\* 格式：

\* new 类/接口(){

\* 如果是创建了继承这个类的子类对象，我们可以重写父类的方法

\* 如果是创建了实现这个接口的子类对象，我们必须要实现该接口的所有方法

\* };

\* 原理：而是创建了继承这个类的子类对象或者是创建了实现这个接口的子类对象

\*

\*/

**public** **class** InnerDemo4 {

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

Outer o = **new** Outer();

o.method();

}

}

**interface** Inner {

**public** **void** function();

}

**class** Outer {

**public** **void** method() {

/\*new Inner() {

@Override

public void function() {

System.out.println("function");

}

}.function();;\*/

Inner i = **new** Inner() {

@Override

**public** **void** function() {

System.*out*.println("function");

}

};

i.function();

i.function();

}

}

### 案例代码七:

匿名内部类作为参数传递

**package** com.itheima\_04;

**public** **interface** Animal {

**public** **abstract** **void** eat();

}

**package** com.itheima\_04;

**public** **class** Cat **implements** Animal {

@Override

**public** **void** eat() {

System.*out*.println("猫吃鱼");

}

}

**package** com.itheima\_04;

**public** **class** Dog **implements** Animal {

@Override

**public** **void** eat() {

System.*out*.println("狗啃骨头");

}

}

**package** com.itheima\_04;

/\*

\* 匿名内部类的应用场景:

\* 作为参数进行传递

\*

\*

\*/

**public** **class** InnerDemo5 {

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

//method(new Dog());

//method(new Cat());

*method*(

**new** Animal() {

@Override

**public** **void** eat() {

System.*out*.println("猫吃鱼");

}

}

);

}

**public** **static** **void** method(Animal a) {

a.eat();

}

}