Java进阶1 第4天

**【学习目标】理解、了解、应用、记忆**

通过今天的学习，参训学员能够：（解释的时候说出二级目标的掌握程度）

1. **【应用】能够使用static关键字与final关键字**
2. 【理解】阐述static修饰符的作用及使用场景
3. 【理解】阐述静态的访问方式及注意事项
4. 【理解】阐述final修饰类、方法、变量有什么特点
5. **【理解】能够定义并使用包，释义四种权限修饰符使用特点**
6. 【理解】阐述包的概念及定义包的格式
7. 【了解】知道包的全名访问格式
8. 【理解】能够使用import导入其他包中的类
9. 【理解】阐述四种权限修饰符在同包的访问情况，跨包的访问情况，不同包子父类的访问情况
10. **【应用】定义内部类并使用内部类**
11. 【理解】阐述阐述内部类的概念及内部类的分类
12. 【理解】阐述成员内部类定义的格式及成员内部类的使用方式
13. 【理解】阐述局部内部类定义的格式及局部内部类的使用方式
14. 【应用】独立编码创建匿名内部类对象并使用匿名内部类

# static与final修饰符

## static简介

### 概述

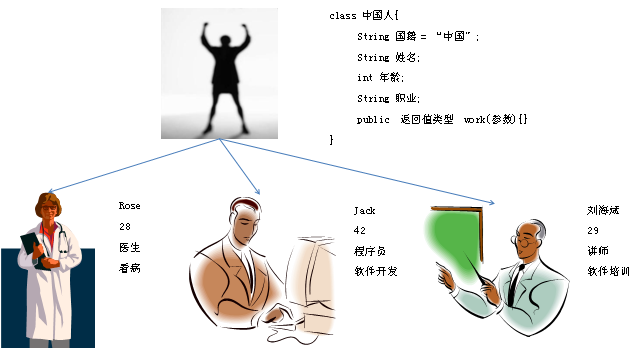
static是静态修饰符，一般修饰成员。被static修饰的成员属于类，不属于单个这个类的某个对象。

static修饰的成员被多个对象共享。

static修饰的成员属于类，但是会影响每一个对象。

被static修饰的成员又叫类成员，不叫对象的成员。

如下例中国籍变量，所有中国人国籍均应该为中国，不应各自定义各自的国籍，所以可以将国籍定义为static，属于类，被多个对象共享。



### 案例代码一

**Chinese类**

|  |
| --- |
| **package** com.igeek\_01;  /\*\*  \* **@ClassName**: Chinese  \* **@Description**: Chinese中国人类  \* **@date** 2017年11月15日 上午9:17:36  \* Company www.igeekhome.com  \*  \* 中国人类  \* 国籍,姓名,年龄,职业  \* 其中,国籍应该被共享  \* 姓名,年龄,职业各个对象有各个对象的值  \*  \*/  **public** **class** Chinese {  //静态成员,被多个对象共享  /\*\*  \* **@Fields** country : 国籍  \*/  **public** **static** String *country* = "中国";  //普通成员,每个对象的普通成员其内容不同  /\*\*  \* **@Fields** name : 姓名  \*/  **private** String name;  /\*\*  \* **@Fields** age : 年龄  \*/  **private** **int** age;  /\*\*  \* **@Fields** work : 职业  \*/  **private** String work;  /\*\*  \* **@Title**: Chinese  \*/  **public** Chinese() {  **super**();  }  /\*\*  \* **@Title**: Chinese  \* **@param** name  \* **@param** age  \* **@param** work  \*/  **public** Chinese(String name, **int** age, String work) {  **super**();  **this**.name = name;  **this**.age = age;  **this**.work = work;  }  /\*\*  \* **@return** the name  \*/  **public** String getName() {  **return** name;  }  /\*\*  \* **@param** name the name to set  \*/  **public** **void** setName(String name) {  **this**.name = name;  }  /\*\*  \* **@return** the age  \*/  **public** **int** getAge() {  **return** age;  }  /\*\*  \* **@param** age the age to set  \*/  **public** **void** setAge(**int** age) {  **this**.age = age;  }  /\*\*  \* **@return** the work  \*/  **public** String getWork() {  **return** work;  }  /\*\*  \* **@param** work the work to set  \*/  **public** **void** setWork(String work) {  **this**.work = work;  }  } |

**StaticDemo类**

|  |
| --- |
| **package** com.igeek\_01;  /\*\*  \* **@ClassName**: StaticDemo  \* **@Description**: static,静态特征测试类  \* **@date** 2017年11月15日 上午9:22:05  \* Company www.igeekhome.com  \*  \* static是静态修饰符，一般修饰成员。  \* 被static修饰的成员属于类，不属于单个这个类的某个对象。  \*  \* static修饰的成员被多个对象共享。  \* static修饰的成员属于类，但是会影响每一个对象。  \* 被static修饰的成员又叫类成员，不叫对象的成员。  \*  \* 所有中国人国籍均应该为中国，不应各自定义各自的国籍，所以可以将国籍定义为static，属于类，被多个对象共享。  \* 国籍,姓名,年龄,职业  \* 其中,国籍应该被共享  \* 姓名,年龄,职业各个对象有各个对象的值  \*  \* 一般static修饰的成员,直接赋值  \*  \* 当多个对象共享使用同一个类中静态成员时,只要该值改变,就会影响所有的对象  \*  \*/  **public** **class** StaticDemo {  **public** **static** **void** main(String[] args) {  //创建人类对象,验证静态static修饰的成员被多个对象共享  Chinese c = **new** Chinese("王宝强", 34, "演员");  System.***out***.println(c.getAge()+"岁的"+c.getName()+"是"+c.*country*+"人");    Chinese c2 = **new** Chinese("周星驰", 48, "演员或导演或制片");  System.***out***.println(c2.getAge()+"岁的"+c2.getName()+"是"+c2.*country*+"人");    c.*country* = "中华人名共和国";  System.***out***.println("================================");    System.***out***.println(c.getAge()+"岁的"+c.getName()+"是"+c.*country*+"人");    System.***out***.println(c2.getAge()+"岁的"+c2.getName()+"是"+c2.*country*+"人");  }  } |

## 静态的使用方式及加载原理

### 原理概述

被static修饰的成员可以并且建议通过类名直接访问。也可以通过某个对象访到属于类的静态成员，原因即多个对象均属于一个类，共享使用同一个静态成员。

格式：

类名.静态成员变量名

类名.静态成员方法名(参数)

对象名.静态成员变量名 ------不建议，出现警告

对象名.静态成员方法名(参数) ------不建议，出现警告

### 案例代码二

|  |
| --- |
| **package** com.igeek\_02;  /\*\*  \* **@ClassName**: StaticDemo  \* **@Description**: static,静态特征测试类  \* **@date** 2017年11月15日 上午9:22:05  \* Company www.igeekhome.com  \*  \* 静态成员直接使用类名访问  \* 类名.静态成员变量名  \* 类名.静态成员方法名(参数)  \*  \*/  **public** **class** StaticDemo {  **public** **static** **void** main(String[] args) {  //创建人类对象,验证静态static修饰的成员被多个对象共享  Chinese c = **new** Chinese("王宝强", 34, "演员");  System.***out***.println(c.getAge()+"岁的"+c.getName()+"是"+Chinese.*country*+"人");    Chinese c2 = **new** Chinese("周星驰", 48, "演员或导演或制片");  System.***out***.println(c2.getAge()+"岁的"+c2.getName()+"是"+Chinese.*country*+"人");    Chinese.*country* = "中华人名共和国";  System.***out***.println("================================");    System.***out***.println(c.getAge()+"岁的"+c.getName()+"是"+Chinese.*country*+"人");    System.***out***.println(c2.getAge()+"岁的"+c2.getName()+"是"+Chinese.*country*+"人");    //类名.方法名()  Chinese.*method*();  }  } |

**StaticDemo2.java类**

|  |
| --- |
| **package** com.igeek\_02;  /\*\*  \* **@ClassName**: StaticDemo2  \* **@Description**: 静态成员的访问  \* **@date** 2017年11月15日 上午9:32:02  \* Company www.igeekhome.com  \*  \* 静态成员只能直接访问静态成员  \*  \* 原因为:静态内容优先于对象存在  \*/  **public** **class** StaticDemo2 {    **public** **static** **void** main(String[] args) {  //由于静态内容随类的加载而加载,有类则有该方法  //但是此时是可以没有对象的  StaticDemo2.*method*();  }  /\*\*  \* **@Fields** name : 姓名的实例成员变量  \*/  **private** String name;    /\*\*  \* **@Title**: normalMethod  \* **@Description**: 普通的实例成员方法  \*/  **public** **void** normalMethod() {  System.***out***.println("普通方法");  }    /\*\*  \* **@Title**: method  \* **@Description**: 静态成员方法  \*/  **public** **static** **void** method() {  //静态方法不能直接访问非静态成员  //System.out.println(name);  //normalMethod();  }  } |

## final关键字

### final关键字的使用

final是最终修饰符，可以修饰类、成员方法、变量。

final修饰的类无法被继承。

final修饰的方法无法被重写。

final修饰的变量无法被再次赋值。

如：

final int a = 10; //则a无法被2次赋值

我们通常使用public static final来定义静态常量（如接口中的固定修饰符）

引用类型的变量用final修饰，是指其所引用的对象不能改变，即该变量引用的地址值不能改变。

### final修饰类无法被继承

#### 案例代码三

FinalClass类

|  |
| --- |
| **package** com.igeek\_03\_02;  /\*\*  \* **@ClassName**: FinalClass  \* **@Description**: final类  \* **@date** 2017年11月15日 上午10:06:15  \* Company www.igeekhome.com  \*  \* 被final修饰的类，不能有子类，不能被继承  \*/  **public** **final** **class** FinalClass {  } |

FinalSubClass类

|  |
| --- |
| **package** com.igeek\_03\_02;  /\*\*  \* **@ClassName**: FinalSubClass  \* **@Description**: 子类  \* **@date** 2017年11月15日 上午10:07:15  \* Company www.igeekhome.com  \*  \* 被final修饰的类FinalClass不能被继承  \*/  //public class FinalSubClass extends FinalClass{  //  //} |

### final修饰的方法无法被重写

#### 案例代码四

FinalClass2类

|  |
| --- |
| **package** com.igeek\_03\_03;  /\*\*  \* **@ClassName**: FinalClass2  \* **@Description**: 被final修饰方法  \* **@date** 2017年11月15日 上午10:10:33  \* Company www.igeekhome.com  \*  \* 被final修饰方法:无法被重写  \*  \*/  **public** **class** FinalClass2 {  /\*\*  \* **@Title**: method  \* **@Description**: final方法  \*/  **public** **final** **void** method() {  System.***out***.println("被final修饰的方法");  }  } |

FinalSubClass2类

|  |
| --- |
| **package** com.igeek\_03\_03;  /\*\*  \* **@ClassName**: FinalSubClass2  \* **@Description**: 子类  \* **@date** 2017年11月15日 上午10:13:16  \* Company www.igeekhome.com  \*  \* final修饰的方法无法被子类重写  \*/  **public** **class** FinalSubClass2 **extends** FinalClass2{  // final修饰的方法不能被重写  // public void method() {  // System.out.println("被final修饰的方法");  // }  } |

### final修饰变量无法被再次赋值

#### 案例代码五

Person类

|  |
| --- |
| **package** com.igeek\_03\_04;  /\*\*  \* **@ClassName**: Person  \* **@Description**: Person类  \* **@date** 2017年11月15日 上午10:25:23  \* Company www.igeekhome.com  \*  \*/  **public** **class** Person {  /\*\*  \* **@Fields** name : 姓名  \*/  **private** String name;  /\*\*  \* **@return** the name  \*/  **public** String getName() {  **return** name;  }  /\*\*  \* **@param** name the name to set  \*/  **public** **void** setName(String name) {  **this**.name = name;  }  } |

FinalDemo测试类

|  |
| --- |
| **package** com.igeek\_03\_04;  /\*\*  \* **@ClassName**: FinalDemo  \* **@Description**: final修饰变量  \* **@date** 2017年11月15日 上午10:18:42  \* Company www.igeekhome.com  \*  \* final是最终修饰符，可以修饰类、成员方法、变量  \*  \* final修饰的类无法被继承。  \* final修饰的方法无法被重写。  \* final修饰的变量无法被再次赋值,必须被赋值。(包括局部变量与成员变量)  \*  \* 现实开发过程中遇到的最常见成员常量的情况,连续使用public static final在一个类中定义一个常量,采取直接给常量赋值的方式.  \* 注意常量的命名规则:所有字母均大写,用\_连接  \*  \*/  **public** **class** FinalDemo {  // 成员位置定义静态常量  /\*\*  \* **@Fields** PI : PI常量  \*/  **public** **static** **final** **double** ***PI*** = 3.1415926;  **public** **static** **void** main(String[] args) {  // final修饰的变量无法被再次赋值  **int** a = 10;  a = 20;  **final** **int** b = 20;  // 被final修饰的变量不能二次赋值  // b = 30;  // 使用静态常量  **double** r = 2.5;  **double** area = r \* r \* FinalDemo.***PI***;  System.***out***.println(area);  // 为引用变量二次赋值,指p这个变量只能指向一个地址,不能指向第二个地址  **final** Person p = **new** Person();  p.setName("张三");  p.setName("李四");  System.***out***.println(p.getName());  // 地址不能改,为下边这句内容  // p = new Person();  }  } |

# 包与访问修饰符

## 包的概述及定义格式

### 包的概念

分包管理是组织软件项目结构的基本方式。我们将同类功能放到一个包中，方便管理。并且日常项目的分工也是以包作为边界。

包在文件系统中是以文件夹的形式存在的。类中定义的包必须与实际class文件所在的文件夹情况相统一，即定义包后，如果类在a包下，则生成的.class文件必须在a文件夹下，否则找不到类。

### 包的定义格式

使用公司域名反写，多级包全部小写，用”.”连接

如：极客营网址为igeekhome.com那么域名反写就为com.igeekhome

甲骨文公司oracle.com 那么域名反写就为 com.oracle

类中的定义格式： package 包名;

编译时：直接编译java文件即可

运行时：由于类的全名是包含包名的，所以必须运行加入了包名的全类名

### 案例代码六

|  |
| --- |
| **package** com.igeek\_01;  /\*\*  \* **@ClassName**: PackageDemo  \* **@Description**: 包的作用  \* **@date** 2018年1月19日 下午2:18:03  \* Company www.igeekhome.com  \*  \* 包：是java程序管理资源的方式  \*  \* 类中声明的包，必须与真正该.class文件所在的文件夹一致  \*  \* 声明的格式:  \* package 包(文件夹).包(文件夹).包(文件夹)...  \*  \*/  **public** **class** PackageDemo {  } |

## 类的全名访问

在访问类时，为了能够找到该类，使用类时，应该使用包含包名的类全名。

如：字符串String在使用时，可以按照如下方式访问：

java.lang.String s = “一个字符串”;

等价于 String s = “一个字符串”;

### 案例代码七

Person类：

|  |
| --- |
| **package** com.igeek\_02;  /\*\*  \* **@ClassName**: Person  \* **@Description**: Person类  \* **@date** 2017年11月15日 上午10:54:11  \* Company www.igeekhome.com  \*  \*/  **public** **class** Person {  /\*\*  \* **@Fields** name : 姓名  \*/  **private** String name;    /\*\*  \* **@Title**: Person  \* **@param** name  \*/  **public** Person(String name) {  **this**.name=name;  }  /\*\*  \* **@return** the name  \*/  **public** String getName() {  **return** name;  }  /\*\*  \* **@param** name the name to set  \*/  **public** **void** setName(String name) {  **this**.name = name;  }  } |

其他包中的Person类：

|  |
| --- |
| **package** com.igeek\_02.other;  /\*\*  \* **@ClassName**: Person  \* **@Description**: Person类  \* **@date** 2017年11月15日 上午10:54:11  \* Company www.igeekhome.com  \*  \*/  **public** **class** Person {  /\*\*  \* **@Fields** name : 姓名  \*/  **private** String name;    /\*\*  \* **@Title**: Person  \* **@param** name  \*/  **public** Person(String name) {  **this**.name=name;  }  /\*\*  \* **@return** the name  \*/  **public** String getName() {  **return** name;  }  /\*\*  \* **@param** name the name to set  \*/  **public** **void** setName(String name) {  **this**.name = name;  }  } |

类全名创建对象测试类：

|  |
| --- |
| **package** com.igeek\_02;  /\*\*  \* **@ClassName**: PackageDemo  \* **@Description**: package包的作用  \* **@date** 2017年11月15日 上午10:37:53  \* Company www.igeekhome.com  \*  \* 包:是java程序管理资源的方式  \*  \* 类中声明的包,必须与真正该.class文件所在的文件夹一致  \*  \* 声明格式:  \* package 文件夹(包).文件夹(包).文件夹(包)...  \*  \* 如何带包使用其他的类:应该加入全名,即包名+类名  \*/  **public** **class** PackageDemo {  **public** **static** **void** main(String[] args) {  //使用类全名访问JRE提供好的类  java.lang.String s = "abc";  System.***out***.println(s);    //使用类全名访问同包下的其他类  com.igeek\_02.Person person=**new** com.igeek\_02.Person("Jack");  person.setName("Rose");  System.***out***.println(person.getName());    //使用类全名访问不同包(跨包)下的其他类  com.igeek\_02.other.Person person2=**new** com.igeek\_02.other.Person("Rose");  System.***out***.println(person2.getName());  }  } |

## 带包的类访问

### 同包下省略包名

当被使用的类与使用的类在同一个文件夹下，或者被使用的类是java.lang包下时，我们通常可以省略掉类的包，直接使用类型，避免使用类全名。

### 跨包时导包访问

当被使用的类与使用的类不在同一个文件夹下时，我们可以通过导包的方式使用该类，避免使用类全名。

导包格式：

package后，class前

使用import 包名.包名.类名;

当多个文件夹下有相同的类名时，只能有一个导包使用，其他必须仍然书写全名。

当想一下导入包中多个类时，可以使用\*代表该包下的所有类。

### 案例代码八

同包下的Person类：

|  |
| --- |
| **package** com.igeek\_03;  /\*\*  \* **@ClassName**: Person  \* **@Description**: Person类  \* **@date** 2017年11月15日 上午10:54:11  \* Company www.igeekhome.com  \*  \*/  **public** **class** Person {  /\*\*  \* **@Fields** name : 姓名  \*/  **private** String name;    /\*\*  \* **@Title**: Person  \* **@param** name  \*/  **public** Person(String name) {  **this**.name=name;  }  /\*\*  \* **@return** the name  \*/  **public** String getName() {  **return** name;  }  /\*\*  \* **@param** name the name to set  \*/  **public** **void** setName(String name) {  **this**.name = name;  }  } |

com.igeek\_03.other1包下的PersonOther类：

|  |
| --- |
| **package** com.igeek\_03.other1;  /\*\*  \* **@ClassName**: PersonOther  \* **@Description**: PersonOther类  \* **@date** 2017年11月15日 上午10:54:11  \* Company www.igeekhome.com  \*  \*/  **public** **class** PersonOther {  /\*\*  \* **@Fields** name : 姓名  \*/  **private** String name;    /\*\*  \* **@Title**: Person  \* **@param** name  \*/  **public** PersonOther(String name) {  **this**.name=name;  }  /\*\*  \* **@return** the name  \*/  **public** String getName() {  **return** name;  }  /\*\*  \* **@param** name the name to set  \*/  **public** **void** setName(String name) {  **this**.name = name;  }  } |

com.igeek\_03.other2包下的PersonOther类：

|  |
| --- |
| **package** com.igeek\_03.other2;  /\*\*  \* **@ClassName**: PersonOther  \* **@Description**: PersonOther类  \* **@date** 2017年11月15日 上午10:54:11  \* Company www.igeekhome.com  \*  \*/  **public** **class** PersonOther {  /\*\*  \* **@Fields** name : 姓名  \*/  **private** String name;    /\*\*  \* **@Title**: Person  \* **@param** name  \*/  **public** PersonOther(String name) {  **this**.name=name;  }  /\*\*  \* **@return** the name  \*/  **public** String getName() {  **return** name;  }  /\*\*  \* **@param** name the name to set  \*/  **public** **void** setName(String name) {  **this**.name = name;  }  } |

不同包中各个类的导入测试类：

|  |
| --- |
| **package** com.igeek\_03;  //导包声明  **import** com.igeek\_03.other1.PersonOther;  //不能导入相同名称的类,所以,当多个文件夹下有相同的类名时，只能有一个导包使用，其他必须仍然书写全名  //import com.igeek\_03.other2.PersonOther;  /\*\*  \* **@ClassName**: PackageAccessDemo  \* **@Description**: 不同包下面各个类的访问方式  \* **@date** 2017年11月15日 上午11:17:49  \* Company www.igeekhome.com  \*  \* JRE提供的java.lang包下的类,是可以直接使用类名,无需加包名的,比如String  \*  \* 同包访问时,可以省略全名,使用类名  \*  \* 跨包访问:  \* 可以不使用全名,使用导包的方式声明出要使用的类时哪个  \*  \* 导包格式：  \* package后，class前  \* 使用import 包名.包名.类名;  \*/  **public** **class** PackageAccessDemo {  **public** **static** **void** main(String[] args) {    //JRE提供的lang包下的类,是可以直接使用类名,无需加包名的,比如String  String s = "hit the road";    //同包访问时,可以省略全名,使用类名  Person person = **new** Person("Jack");  System.***out***.println(person.getName());    //跨包访问,需要先导包,再使用  PersonOther Person2 = **new** PersonOther("Rose");  System.***out***.println(Person2.getName());    //不能导入相同名称的类,所以,当多个文件夹下有相同的类名时，只能有一个导包使用，其他必须仍然书写全名  com.igeek\_03.other2.PersonOther Person3 = **new** com.igeek\_03.other2.PersonOther("Rose");  System.***out***.println(Person3.getName());    }  } |

## 访问权限

### 四种权限访问修饰符

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 | 空的(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

* 要想仅能在本类中访问使用private修饰；
* 要想本包中的类都可以访问不加修饰符即可；
* 要想本类的子类可以访问使用protected修饰
* 要想任意包中的任意类都可以访问使用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。

### 同包中权限访问测试

#### 案例代码九

**AccessClass类**

|  |
| --- |
| **package** com.igeek\_04\_02;  /\*\*  \* **@ClassName**: AccessClass  \* **@Description**: 测试访问权限的类  \* **@date** 2017年11月15日 下午2:24:35  \* Company www.igeekhome.com  \*  \*/  **public** **class** AccessClass {    /\*\*  \* **@Title**: method  \* **@Description**: 私有化的方法  \*/  **private** **void** method() {  System.***out***.println("私有化的方法");  }    /\*\*  \* **@Title**: method2  \* **@Description**: 默认权限的方法  \*/  **void** method2() {  System.***out***.println("默认权限的方法");  }    /\*\*  \* **@Title**: method3  \* **@Description**: 受保护的方法  \*/  **protected** **void** method3() {  System.***out***.println("受保护的方法");  }    /\*\*  \* **@Title**: method4  \* **@Description**: 公共的方法  \*/  **public** **void** method4() {  System.***out***.println("公共的方法");  }    /\*\*  \* **@Title**: testAccess  \* **@Description**: 本类当中的权限测试方法  \*/  **public** **void** testAccess() {  method();  method2();  method3();  method4();  }  **public** **static** **void** main(String[] args) {    AccessClass ac = **new** AccessClass();    //本类当中,可以访问所有的方法  ac.method();  ac.method2();  ac.method3();  ac.method4();  }  } |

**AccessDemo类**

|  |
| --- |
| **package** com.igeek\_04\_02;  /\*\*  \* **@ClassName**: AccessDemo  \* **@Description**: 访问权限的测试类  \* **@date** 2017年11月15日 下午2:26:38  \* Company www.igeekhome.com  \*  \*/  **public** **class** AccessDemo {  **public** **static** **void** main(String[] args) {    AccessClass ac = **new** AccessClass();    //同包下,可以访问其他类的非私有方法  //ac.method();  ac.method2();  ac.method3();  ac.method4();  }  } |

### 跨包访问

#### 案例代码十

**AccessDemo类**

|  |
| --- |
| **package** com.igeek\_04\_03;  **import** com.igeek\_04\_02.AccessClass;  /\*\*  \* **@ClassName**: AccessDemo  \* **@Description**: 访问权限的测试类  \* **@date** 2017年11月15日 下午2:26:38  \* Company www.igeekhome.com  \*  \*/  **public** **class** AccessDemo {  **public** **static** **void** main(String[] args) {    AccessClass ac = **new** AccessClass();    //测试跨包访问,只能访问公共方法  //ac.method();  //ac.method2();  //ac.method3();  ac.method4();  }  } |

### 不同包下子父类之间访问

#### 案例代码十一

**AccessDemo类**

|  |
| --- |
| **package** com.igeek\_04\_04;  **import** com.igeek\_04\_02.AccessClass;  /\*\*  \* **@ClassName**: AccessDemo  \* **@Description**: 访问权限的测试类  \* **@date** 2017年11月15日 下午2:26:38  \* Company www.igeekhome.com  \*  \* AccessDemo类继承父类AccessClass  \*/  **public** **class** AccessDemo **extends** AccessClass{    **public** **static** **void** main(String[] args) {    AccessClass ac = **new** AccessClass();    //测试跨包访问,只能访问公共方法  //ac.method();  //ac.method2();  //ac.method3();  ac.method4();    AccessDemo ad=**new** AccessDemo();  ad.method0();  }    /\*\*  \* **@Title**: method0  \* **@Description**: 子类可以访问父类的公共与受保护方法,无论是否同包  \*/  **public** **void** method0() {  //super.method();  //super.method2();  **super**.method3();  **super**.method4();  }  } |

# 内部类

## 内部类基本概念

内部类即在类中定义另外一个类，是一个相对概念。

内部类分为成员内部类与局部内部类。

定义时是一个正常定义类的过程，同样包含各种修饰符、继承与实现关系等。

内部类可以直接访问外部类的所有成员。

外部类编译后会出现两个class文件。

(在日常的企业级开发中，我们很少会使用到内部类来实现业务逻辑)

## 成员内部类

定义在成员位置的内部类

定义格式

public class Outer{

class Inner{

//其他代码

}

}

访问格式：

Outer.Inner x = new Outer().new Inner();

### 代码案例十二

**NestedClass成员内部类**

|  |
| --- |
| **package** com.igeek\_02;  /\*\*  \* **@ClassName**: NestedClass  \* **@Description**: 成员内部类  \* **@date** 2017年11月15日 下午4:02:57  \* Company www.igeekhome.com  \*  \* 外部类  \*/  **public** **class** NestedClass {  /\*\*  \* **@Fields** name : 姓名  \*/  **private** String name = "Jack";    /\*\*  \* **@ClassName**: InnerClass  \* **@Description**: 成员内部类  \* **@date** 2017年11月15日 下午4:03:49  \* Company www.igeekhome.com  \*  \*/  **public** **class** InnerClass {    **public** **void** innerMethod() {  System.***out***.println("内部类的方法被调用了");  System.***out***.println("外部类的成员变量name的值为:"+name);  }  }  /\*\*  \* **@return** the name  \*/  **public** String getName() {  **return** name;  }  /\*\*  \* **@param** name the name to set  \*/  **public** **void** setName(String name) {  **this**.name = name;  }  } |

**NestedClassDemo成员内部类测试类**

|  |
| --- |
| **package** com.igeek\_02;  /\*\*  \* **@ClassName**: NestedClassDemo  \* **@Description**: 成员内部类测试类  \* **@date** 2017年11月15日 下午4:05:03  \* Company www.igeekhome.com  \*  \* 内部类是一个相对概念，即类中定义类  \* 内部类可以直接访问外部类的所有成员  \*  \* 内部类：  \* 成员内部类：定义在成员位置  \* 局部内部类：定义在局部位置，即一个方法内  \*  \* 成员内部类创建对象，必须有外部类对象，才能有内部类对象  \* Outer.Inner x = new Outer().new Inner();  \*/  **public** **class** NestedClassDemo {  **public** **static** **void** main(String[] args) {  //创建成员内部类对象  NestedClass.InnerClass inner = **new** NestedClass().**new** InnerClass();  inner.innerMethod();  }  } |

## 局部内部类

定义在局部位置的内部类

定义格式

public class Outer{

public void method{

class Inner{

//其他代码

}

}

}

访问格式：

只能在外部类的方法中创建对象并访问。

### 案例代码十三

**NestedClass局部内部类**

|  |
| --- |
| **package** com.igeek\_03;  /\*\*  \* **@ClassName**: NestedClass  \* **@Description**: 局部内部类  \* **@date** 2017年11月15日 下午4:02:57  \* Company www.igeekhome.com  \*  \* 外部类  \*/  **public** **class** NestedClass {  /\*\*  \* **@Fields** name : 姓名  \*/  **private** String name = "Jack";    /\*\*  \* **@Title**: outerMethod  \* **@Description**: 外部类的方法  \*/  **public** **void** outerMethod() {  /\*\*  \* **@ClassName**: InnerClass  \* **@Description**: 局部内部类  \* **@date** 2017年11月15日 下午4:10:17  \* Company www.igeekhome.com  \*/  **class** InnerClass {    **public** **void** innerMethod() {  System.***out***.println("内部类的方法被调用了");  System.***out***.println("外部类的成员变量name的值为:"+name);  }  }    //定义好类之后可以使用类  InnerClass inner = **new** InnerClass();  inner.innerMethod();  }  /\*\*  \* **@return** the name  \*/  **public** String getName() {  **return** name;  }  /\*\*  \* **@param** name the name to set  \*/  **public** **void** setName(String name) {  **this**.name = name;  }  } |

**NestedClassDemo局部内部类测试类**

|  |
| --- |
| **package** com.igeek\_03;  /\*\*  \* **@ClassName**: NestedClassDemo  \* **@Description**: 局部内部类测试类  \* **@date** 2017年11月15日 下午4:05:03  \* Company www.igeekhome.com  \*  \* 内部类是一个相对概念，即类中定义类  \* 内部类可以直接访问外部类的所有成员  \*  \* 内部类：  \* 成员内部类：定义在成员位置  \* 局部内部类：定义在局部位置，即一个方法内，符合先定义再使用！  \*  \* 成员内部类创建对象，必须有外部类对象，才能有内部类对象  \* Outer.Inner x = new Outer().new Inner();  \*/  **public** **class** NestedClassDemo {  **public** **static** **void** main(String[] args) {  //创建外部类对象  NestedClass nc = **new** NestedClass();  //通过外部类对象调用方法访问局部内部类对象的操作  nc.outerMethod();  }  } |

## 匿名内部类

### 概述

匿名内部类是局部内部类的一种。

过程：

临时定义一个类型的子类

定义后即刻创建刚刚定义的这个类的对象

目的：

匿名内部类是创建某个类型子类对象的快捷方式。

我们为了临时定义一个类的子类，并创建这个子类的对象而使用匿名内部类。

### 匿名内部类格式

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的、即时的。

匿名内部类如果不定义变量引用，则也是匿名对象。格式如下：

new 父类(){

//重写需要重写的方法

};

例如：

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”);

}

};

### 案例代码十四

**Animal抽象父类**

|  |
| --- |
| **package** com.igeek\_04;  /\*\*  \* **@ClassName**: Animal  \* **@Description**: 抽象父类Animal  \* **@date** 2017年11月15日 下午4:18:17  \* Company www.igeekhome.com  \*  \* 吃饭,睡觉的抽象方法  \*/  **public** **abstract** **class** Animal {  //抽象方法  /\*\*  \* **@Title**: eat  \* **@Description**: 吃饭的方法  \*/  **public** **abstract** **void** eat();  /\*\*  \* **@Title**: sleep  \* **@Description**: 睡觉的方法  \*/  **public** **abstract** **void** sleep();  } |

**Dog具体子类**

|  |
| --- |
| **package** com.igeek\_04;  /\*\*  \* **@ClassName**: Dog  \* **@Description**: 子类Dog类  \* **@date** 2017年11月15日 下午4:19:33  \* Company www.igeekhome.com  \*  \* 定义子类,重写父类方法  \*/  **public** **class** Dog **extends** Animal {  //重写方法  /\*\*  \* **@Title**: eat  \* **@Description**: 狗吃饭的方法  \* **@see** com.igeek\_04.Animal#eat()  \*/  @Override  **public** **void** eat() {  System.***out***.println("吃骨头");  }  /\*\*  \* **@Title**: sleep  \* **@Description**: 狗睡觉的方法  \* **@see** com.igeek\_04.Animal#sleep()  \*/  @Override  **public** **void** sleep() {  System.***out***.println("趴着睡");  }  } |

**AnonymousInnerClass匿名内部类的测试类**

|  |
| --- |
| **package** com.igeek\_04;  /\*\*  \* **@ClassName**: AnonymousInnerClass  \* **@Description**: 匿名内部类的测试类  \* **@date** 2017年11月15日 下午4:20:54  \* Company www.igeekhome.com  \*  \* 匿名内部类  \*  \* 目的:创建某个类(接口)的子类对象  \* 步骤1:定义带名字的子类  \* 步骤2:创建对象  \*  \* 匿名内部类是创建某个类型子类对象的快捷方式  \* 我们为了临时定义一个类的子类，并创建这个子类的对象而使用匿名内部类。  \*  \* 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  \*  \* 匿名内部类格式:  \* new 父类(){  \* //重写需要重写的方法  \* };  \*/  **public** **class** AnonymousInnerClass {    **public** **static** **void** main(String[] args) {    //使用有名字的子类(多态使用)  Animal ad = **new** Dog();  ad.eat();  ad.sleep();    //使用匿名内部类  Animal a = **new** Animal(){  /\*\*  \* **@Title**: eat  \* **@Description**: 吃草的方法  \* **@see** com.igeek\_04.Animal#eat()  \*/  @Override  **public** **void** eat() {  System.***out***.println("吃草");  }  /\*\*  \* **@Title**: sleep  \* **@Description**: 睡觉的方法  \* **@see** com.igeek\_04.Animal#sleep()  \*/  @Override  **public** **void** sleep() {  System.***out***.println("仰着睡");  }  };    a.eat();  a.sleep();    }  } |

重点和总结

1、static与final修饰符

2、package打包，import导入包，及四种访问修饰符

3、内部类的各种形式：成员内部类、局部内部类、匿名内部类