收藏 313 23

java内部类 编辑

java内部类分为： 成员内部类、静态嵌套类、方法内部类、匿名内部类 。

中文名	java内部类	分　为	成员方法匿名内部类静态嵌套类
外文名	Java inner classes	内部类是	独立的类
		不　能	用普通的方式访问

目录

java内部类图册

内部类的共性 编辑

(1)、内部类仍然是一个独立的类，在编译之后内部类会被编译成独立的.class文件，但是前面冠以外部类的类名和$符号 。

(2)、内部类不能用普通的方式访问。内部类是外部类的一个成员，因此内部类可以自由地访问外部类的成员变量，无论是否是private的 。

(3)、内部类声明成静态的，就不能随便的访问外部类的成员变量了，此时内部类只能访问外部类的静态成员变量 。

成员内部类 编辑

```
class Outer {

class Inner{}

}
```

编译上述代码会产生两个文件：Outer.class和Outer$Inner.class。

方法内部类 编辑

```
把类放在方法内

class Outer {

public void doSomething(){

class Inner{

public void seeOuter(){

}

}

}

}
```

(1)、方法内部类只能在定义该内部类的方法内实例化，不可以在此方法外对其实例化。

(2)、方法内部类对象不能使用该内部类所在方法的非final局部变量。

因为方法的局部变量位于栈上，只存在于该方法的生命期内。当一个方法结束，其栈结构被删除，局部变量成为历史。但是该方法结束之后，在方法内创建的内部类对象可能仍然存在于堆中！例如，如果对它的引用被传递到其他某些代码，并存储在一个成员变量内。正因为不能保证局部变量的存活期和方法内部类对象的一样长，所以内部类对象不能使用它们。

下面是完整的例子：

```
class Outer {

public void doSomething(){

final int a =10;

class Inner{

public void seeOuter(){

System.out.println(a);

}

}

Inner in = new Inner();

in.seeOuter();

}

public static void main(String[] args) {

Outer out = new Outer();

out.doSomething();

}

}
```

匿名内部类

编辑

顾名思义，没有名字的内部类。表面上看起来它们似乎有名字，实际那不是它们的名字。

当程序中使用匿名内部类时，在定义匿名内部类的地方往往直接创建该类的一个对象。匿名内部类的声明格式如下：

```
new ParentName（）{

...// 内部类的定义

}
```

[1]

匿名内部类就是没有名字的内部类。什么情况下需要使用匿名内部类？如果满足下面的一些条件，使用匿名内部类是比较合适的：

·只用到类的一个实例 。

·类在定义后马上用到。

·类非常小（SUN推荐是在4行代码以下）

·给类命名并不会导致你的代码更容易被理解。

在使用匿名内部类时，要记住以下几个原则：

·匿名内部类不能有构造方法。

·匿名内部类不能定义任何静态成员、静态方法。

·匿名内部类不能是public,protected,private,static。

·只能创建匿名内部类的一个实例。

·一个匿名内部类一定是在new的后面，用其隐含实现一个接口或实现一个类。

·因匿名内部类为局部内部类，所以局部内部类的所有限制都对其生效。

A、继承式的匿名内部类

```
public class Car {
public void drive(){
System.out.println("Driving a car!");
}
public static void main(String[] args) {
Car car = new Car(){
public void drive() {
System.out.println("Driving another car!");
}
};
car.drive();
}
}
```

结果输出了：Driving another car! Car引用变量不是引用Car对象，而是Car匿名子类的对象。

B、接口式的匿名内部类。

```
interface Vehicle {
public void drive();
}
class Test{
public static void main(String[] args) {
Vehicle v = new Vehicle(){
public void drive(){
System.out.println("Driving a car!");
}
};
v.drive();
}
}
```

上面的代码很怪，好像是在实例化一个接口。事实并非如此，接口式的匿名内部类是实现了一个接口的匿名类。而且只能实现一个接口。

C、参数式的匿名内部类。

```
class Bar{
void doStuff(Foo f){
f.foo();
}
}
interface Foo{
void foo();
}
class Test{
```

```
static void go(){
Bar b = new Bar();
b.doStuff(new Foo(){
public void foo(){
System.out.println("foofy");
}
});
}
}
```

静态嵌套类

编辑

静态内部类中可以定义静态或者非静态的成员。

从技术上讲，静态嵌套类不属于内部类。因为内部类与外部类共享一种特殊关系，更确切地说是对实例的共享关系。而静态嵌套类则没有上述关系。它只是位置在另一个类的内部，因此也被称为顶级嵌套类。

静态的含义是该内部类可以像其他静态成员一样，没有外部类对象时，也能够访问它。静态嵌套类仅能访问外部类的静态成员和方法。

```
class Outer{
static class Inner{}
}
class Test {
public static void main(String[] args){
Outer.Inner n = new Outer.Inner();
}
}
```

在静态方法中定义的内部类也是StaticNested Class，这时候不能在类前面加static关键字，静态方法中的StaticNested Class与普通方法中的内部类的应用方式很相似，它除了可以直接访问外部类中的static的成员变量，还可以访问静态方法中的局部变量，但是，该局部变量前必须加final修饰符。

为什么需要内部类

典型的情况是，内部类继承自某个类或实现某个接口，内部类的代码操作创建其他外围类的对象。所以你可以认为内部类提供了某种进入其外围类的窗口。使用内部类最吸引人的原因是：

每个内部类都能独立地继承自一个（接口的）实现，所以无论外围类是否已经继承了某个（接口的）实现，对于内部类都没有影响。如果没有内部类提供的可以继承多个具体的或抽象的类的能力，一些设计与编程问题就很难解决。从这个角度看，内部类使得多重继承的解决方案变得完整。接口解决了部分问题，而内部类有效地实现了“多重继承”。

参考资料

1. 桂珠、张平、陈爱国．Java面向对象程序设计（jdk1.6）第三版：北京邮电大学出版社，2005

新手上路
成长任务 编辑入门
编辑规则 百科术语

我有疑问
我要质疑 我要提问
参加讨论 意见反馈

投诉建议
举报不良信息 未通过词条申诉
投诉侵权信息 封禁查询与解封