匿名内部类详解

概 念

匿名内部类也就是没有名字的内部类

正因为没有名字，所以匿名内部类只能使用一次，它通常用来简化代码编写

但使用匿名内部类还有个前提条件：必须继承一个父类或实现一个接口

注意事项：

- 编译后的文件命名：外部类$数字 .class
- 无法使用 public、private、abstract、static 修饰，匿名内部类不能出现抽象方法
- 无法编写构造方法，但可以添加构造代码块
- 不能出现静态成员
- 匿名内部类可实现接口也可以继承类，但是不可兼得
- 匿名内部类不能是抽象的，它必须要实现继承的类或者实现接口的所有抽象方法

匿 名 内 部 类 初 始 化

我们一般都是利用构造器来完成某个实例的初始化工作的，但是匿名内部类是没有构造器的，那怎么来初始化匿名内部类呢？使用构造代码块！利用构造代码块能够达到为匿名内部类创建一个构造器的效果。

实例：

```
public class OutClass {
    public InnerClass getInnerClass(final int age,final String name){
        return new InnerClass() {
            int age_ ;
            String name_;
            //构造代码块完成初始化工作
            {
                if(0 < age && age < 200){
                    age_ = age;
                    name_ = name;
                }
            }
            public String getName() {
                return name_;
            }

            public int getAge() {
                return age_;
            }
        };
    }

    public static void main(String[] args) {
        OutClass out = new OutClass();

        InnerClass inner_1 = out.getInnerClass(201, "chenssy");
        System.out.println(inner_1.getName());

        InnerClass inner_2 = out.getInnerClass(23, "chenssy");
        System.out.println(inner_2.getName());
    }
}
```

实 例

匿名内部类可以有不同的表现形式，下面用实例向大家展示一下：

继承式的匿名内部类：

```
abstract class Car {
    public abstract void drive();
}

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

输出结果：Driving another car!

引用变量不是引用 Car 对象，而是 Car 匿名子类的对象。

建立匿名内部类的关键点是重写父类的一个或多个方法。再强调一下，是重写父类的方法，而不是创建新的方法。因为用父类的引用不可能调用父类本身没有的方法，创建新的方法是多余的。

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();
    }
}
```

输出结果：Driving a car!

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。

参数式的匿名内部类：

```
abstract class Bar{
    void doStuff(Foo f){}
}

class BarOne extends Bar{
    void doStuff(Foo f){}
}

interface Foo{
    void foo();
}

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

由上面的三个例子可以看出，只要一个类是抽象的或是一个接口，那么其子类中的方法都可以使用匿名内部类来实现。最常用的情况就是在多线程的实现上，因为要实现多线程必须继承 Thread 类或是实现 Runnable 接口

Thread 类的匿名内部类实现：

```
public class Demo {
    public static void main(String[] args) {
        Thread t = new Thread() {
            public void run() {
                for (int i = 1; i <= 5; i++) {
                    System.out.print(i + " ");
                }
            }
        };
        t.start();
    }
}
```

Runnable 接口的匿名内部类实现：

```
public class Demo {
    public static void main(String[] args) {
        Runnable r = new Runnable() {
            public void run() {
                for (int i = 1; i <= 5; i++) {
                    System.out.print(i + " ");
                }
            }
        };
        Thread t = new Thread(r);
        t.start();
    }
}
```