# 第5章 内部类

1. 定义内部类
2. 内部类的种类
3. 内部类的使用
4. 为什么要内部类？

# 内部类与异常类

## 导读

**重点：**

成员内部类

匿名类

**难点：**

内部类的作用

# 1　内部类

类可以有两种重要的成员：成员变量和方法，实际上Java还允许类可以有一种成员：内部类。

```
Class A{
      float price=0.5;
      public A(){
       }

}

Class B{
}
```

```
Class A{
      float price=0.5;
       public A(){
        }

      Class B{
      }

}
```

# 内部类的文件

**非内部类：**

几个class就会有对应的几个独立的class字节文件

Outter.class　Inner.class

**含内部类：**

除了外部类还有几个以外部类为前缀的class字节文件

Outter.class　Outter$Inner.class

# 1　内部类

Java支持在一个类中声明另一个类，这样的类称作**内部类**，而包含内部类的类成为内部类的**外嵌类**。

◆内部类可以作为外部类的一个成员而存在的。

◆**成员内部类**可用静态修饰，也可用public，Protected 和private 修饰,但**局部内部类**和**匿名类**不可以用这些访问修饰符。

```
class  <外部类名>{

        [<成员的访问限制修饰符号>]  [static] <内部类名>{
                //内部类成员
          }
          //外部类的其他成员
}
```

# 1 内部类

内部类仅供它的外嵌类使用，其他类不可以用某个类的内部类声明对象。

➡ 外嵌类的成员变量在内部类中仍然有效，内部类中的方法也可以调用外嵌类中的方法。

➡ 内部类的类体中不可以声明类变量和类方法。外嵌类的类体中可以用内部类声明对象，作为外嵌类的成员。

## 2. 内部类的种类：

（1）成员内部类（熟知）
（2）静态内部类（了解）
（3）局部内部类（熟知）
（4）匿名内部类（掌握）

# 2.1 成员内部类

```
public class Outter{
        //内部类
          public class Inner{
                  //内部类的成员
                    int  i=12;
          }
           //外部类的普通成员
          int  count=0;
}
```

# 2.1 成员内部类的对象创建

```
public class Outter{
//内部类
    public class Inner{
        public void show(){
                System.out.println();
        }
    }
  //外部类中的方法，调用内部类
    public Inner getInstance(){
//在外部类中创建内部类的对象
                Inner in=new Inner();
                return Inner;
        }
}
```

外部类之内创建内部类对象

```
public static void main(String
args[]){
        Outter out=new Outter();
        Outter.Inner    in=out.new
Inner()
        in.show();
}
 public static void main(String
 args[]){
        Outter out=new Outter();
        Outer.Inner
 in=out.getInstance();
        in.show();
 }
```

## 2.2 静态内部类

当内部类前有static关键字时，该内部类为静态内部类

静态内部类是外部类的静态成员，其不依赖于外部类的对象存在，因此在创建该种内部类的对象时，不需要先把外部类的对象创建出来。

**<外部类类名>.<内部类类名> 引用变量=**
**new <外部类类名>.<内部类构造器>**

静态嵌套类**仅能访问外**部类的静态成员和方法。

# 2.2 静态内部类

```
class Outer{
    static double price=0.6d;
    String name="zhangsan";
    static class Inner{
        void print(){
            System.out.println(name); //Error
            System.out.println(price);   //correct
        }
    }
}
class Test {
    public static void main(String[] args){
        Outer.Inner n = new Outer.Inner();
    }
}
```

# 2.3 局部内部类

把类放在方法内

```
class Outer {
    public void doSomething(){
        class Inner{
            public void
            seeOuter(){      }
        }
    }
}
```

局部内部类的作用域与局部变量相同

局部内部类不能有成员的访问限制修饰符

# 2.3 局部内部类

**两个优点：**

（1）它对外面的所有类来说都是隐藏的

（2）它可以访问当前代码块内的**常量**和此外部类的所有成员

```
public class Outter{
        public void getInner(){
            //定义局部变量
            int x=100;
            //定义局部内部类

}
```

```
public class Outter{
        public void getInner(){
            //定义局部变量
            int x=100;   //error
            //定义局部内部类
             class Inner{
                  public void show(){
                      System.out.println(
                          “visit x,x=”+x;
                       )
                  }
              }

}
```

final int x=100;

# 2.4 匿名内部类

## 没有名字的内部类

```
new 父类构造器 | 实现接口{
    //匿名内部类类体
}
```

既声明了一个匿名内部类，又同时创建了一个匿名内部类的对象

# 2.4 匿名内部类

## 没有名字的内部类

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（推荐是在4行代码以下）

# 2.4 匿名内部类

## 继承式匿名内部类

```
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();
    }
}
```

# 2.4 匿名内部类

接口式匿名内部类

```
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();
    }
}
```

# 2.4 匿名内部类

## 参数式匿名内部类

```
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");}
                         });
     }
   }
```

# 2.4 匿名内部类

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。

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的外围类的对象。所以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或一个（接口的）实现，所以无论外围类是否已经继承了或某个（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“多继承”。-》例子

```
class Guider{
    //向导
    public void work(String name) {
    System.out.println(name+" 向导工作");
    }
}
class Repairer{
    public void work(String name) {
    System.out.println(name+" 维护工作");
    }
}
```

含有同名方法的类或者接口

InnerMultiExtends.java

```
class Robot extends Guider{
    public void doGuiderWork(String name) {
        work(name);
    }

    public class RepairRobot extends Repairer{
        public void doRepairWork(String name) {
        work(name);
        }
     }

    public void doRepairWork(String name) {
            new RepairRobot().doRepairWork(name);
    }
}
```

其它方案？

改成接口

参数接口回调的方式，传实现该接口的不同对象。。。。。