笔记总链接：http://bbs.itheima.com/thread-200600-1-1.html

4、继承

4.7 接口

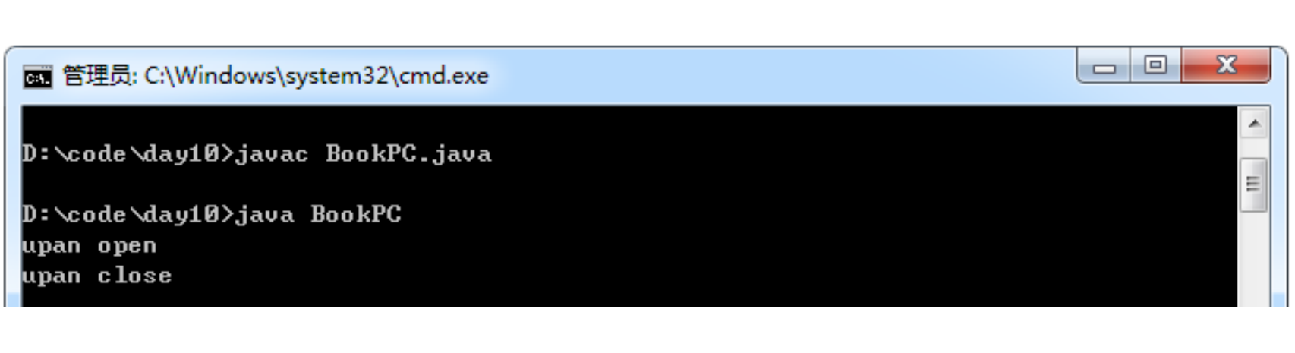
接口应用综合案例

代码：

```
/*
笔记本电脑使用。
为了扩展笔记本的功能，但日后出现什么功能设备不知道。
因此需要定义一个规则，只要日后出现的设备都符合这个规则就可以了。
规则在java中就是接口。
*/
interface USB{//暴露的原则
        public void open();
        public void close();
}

//实现原则
//这些设备和电脑的耦合性降低了
class UPan implements USB{
        public void open(){
             System.out.println("upan open");
        }
        public void close(){
             System.out.println("upan close");
        }
}

class UsbMouse implements USB{
        public void open(){
             System.out.println("usbMouse open");
        }
        public void close(){
             System.out.println("usbMouse close");
        }
}

class BookPC{
        public static void main(String[] args){
             //功能扩展了
             useUSB(new UPan());
        }
        //使用原则
        public static void useUSB(USB u){//接口类型的引用，用于接收（指向）接口的子类对象
             if(u != null ){
                  u.open();
                  u.close();
             }
        }
}
```

复制代码

运行结果：



```
D:\code\day10>javac BookPC.java

D:\code\day10>java BookPC
upan open
upan close
```

4.8 多态

定义：某一类事物的多种存在形态。

例：动物中猫，狗。

猫这个对象对应的类型是猫类型：猫 x = new 猫();

同时猫也是动物中的一种，也可以把猫称为动物：动物 y = new 猫();

动物是猫和狗具体事物中抽取出来的父类型。

父类型引用指向了子类对象。

多态性简单说就是一个对象对应着不同类型。

体现：

父类或者接口的引用指向或者接收自己的子类对象。

作用：

多态的存在提高了程序的扩展性和后期可维护性。

前提：

1. 需要存在继承或者实现关系。
2. 需要有覆盖操作。

好处：

提高了代码的扩展性，前期定义的代码可以使用后期的内容。

弊端：

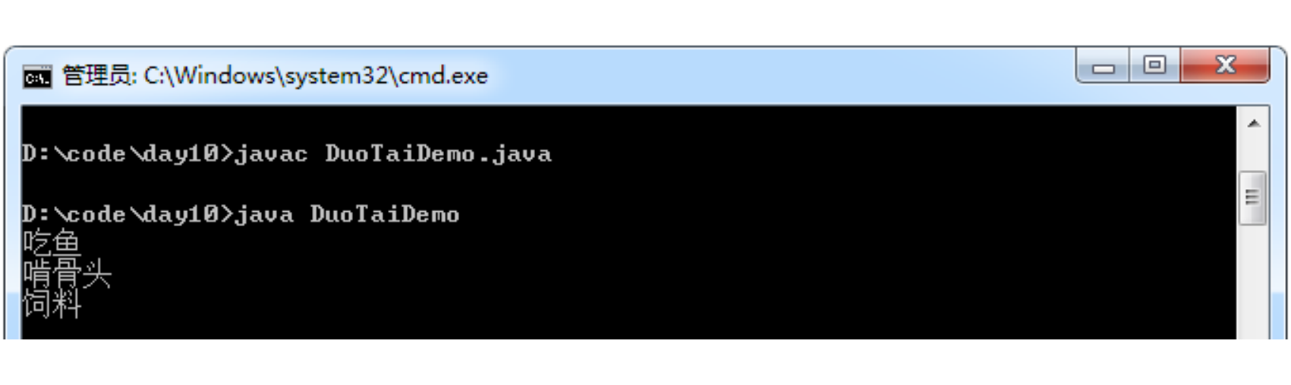
前期定义的内容不能使用（调用）后期子类的特有内容。

示例1：

```
abstract class Animal{
      abstract void eat();
}

class Dog extends Animal{
      void eat(){
          System.out.println("啃骨头");
      }
      void lookHome(){
          System.out.println("看家");
      }
}

class Cat extends Animal{
      void eat(){
          System.out.println("吃鱼");
      }
      void catchMouse(){
          System.out.println("抓老鼠");
      }
}

class Pig extends Animal{
      void eat(){
          System.out.println("饲料");
      }
      void gongdi(){
          System.out.println("拱地");
      }
}

class DuoTaiDemo{
      public static void main(String[] args){
          Cat c = new Cat();
          Dog d = new Dog();

           method(c);
           method(d);
           method(new Pig());
      }

      public static void method(Animal a){
          a.eat();
      }
}
```

复制代码

运行结果：



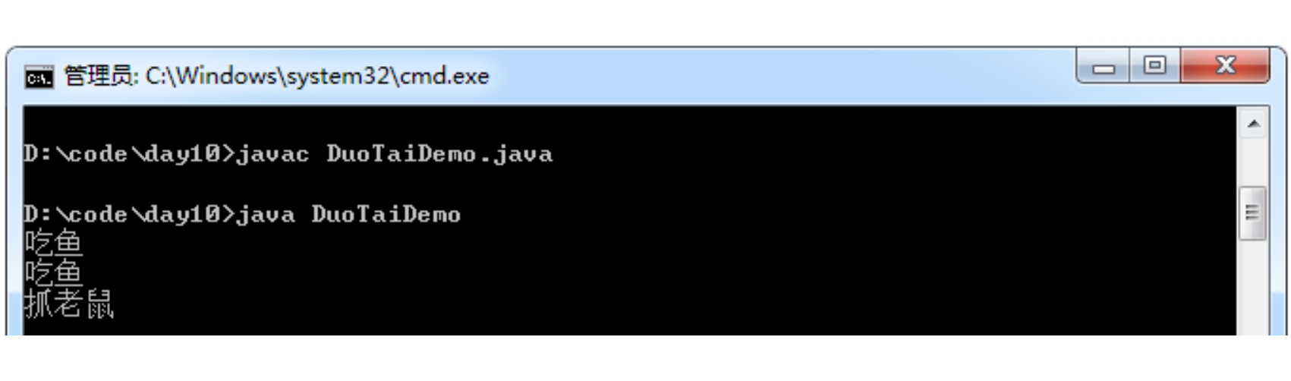
```
D:\code\day10>javac DuoTaiDemo.java

D:\code\day10>java DuoTaiDemo
吃鱼
啃骨头
饲料
```

示例2：

```
class DuoTaiDemo{
       public static void main(String[] args){
            //自动类型提升，猫对象提升到了动物类型。但是特有功能无法访问，作用就是限制对特有功能的访问。
            //专业讲：向上转型，将子类型隐藏。就不能使用子类的特有方法了。
            Animal a = new Cat();
            a.eat();
            //a.catchMouse();//报错

            //如果还想用具体动物猫的特有功能。
            //你可以将该对象进行向下转型。
            Cat c = (Cat)a; //向下转型的目的是为了能够使用子类中的特有方法。
            c.eat();
            c.catchMouse();

            //注意：对于转型，自始至终都是子类对象在做类型的变化。
            //Animal a = new Dog();
            //Cat c = (Cat)a;//但是类型不能随意转换，否则可能会报出ClassCastException的异常
       }

       public static void method(Animal a){
            a.eat();
       }
}
```

复制代码

运行结果：



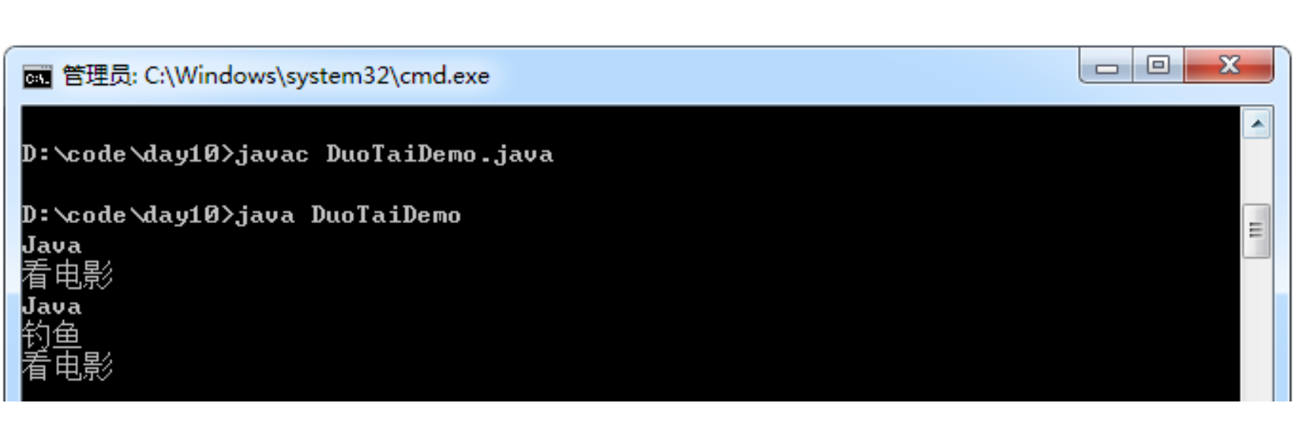
```
D:\code\day10>javac DuoTaiDemo.java

D:\code\day10>java DuoTaiDemo
吃鱼
吃鱼
抓老鼠
```

示例3：

```
/*
毕老师和毕姥爷的故事
*/
class 毕姥爷{
        void 讲课(){
             System.out.println("管理");
        }
        void 钓鱼(){
             System.out.println("钓鱼");
        }
}

class 毕老师 extends 毕姥爷{
        void 讲课(){
             System.out.println("Java");
        }
        void 看电影(){
             System.out.println("看电影");
        }
}

class DuoTaiDemo{
        public static void main(String[] args){
             毕老师 x = new 毕老师();
             x.讲课(); //Java
             x.看电影(); //看电影

             毕姥爷 y = new 毕老师();
             y.讲课(); //Java
             y.钓鱼(); //钓鱼

             毕老师 z = (毕老师)y;
             z.看电影(); //看电影
       }
}
```

复制代码

运行结果：



```
D:\code\day10>javac DuoTaiDemo.java

D:\code\day10>java DuoTaiDemo
Java
看电影
Java
钓鱼
看电影
```

P.S.

instanceof ：用于判断对象的具体类型，只能用于引用数据类型判断，通常在向下转型前用于健壮性的判断。

示例4：

```
class DuoTaiDemo{
        public static void main(String[] args){
        }

        public static void method(Animal a){
             a.eat();

              if(a instanceof Cat){
                   Cat c = (Cat )a;
                   c.catchMouse();
              }
              else if (a instanceof Dog){
                   Dog d = (Dog )a;
                   d.lookHome();
              }
        }
}
```

复制代码

多态时，成员的特点：

1. 成员变量

编译时：参考引用型变量所属的类中是否有调用的成员变量。有，编译通过，没有，编译失败。

运行时：参考引用型变量所属的类中是否有调用的成员变量，并运行该所属类中的成员变量。

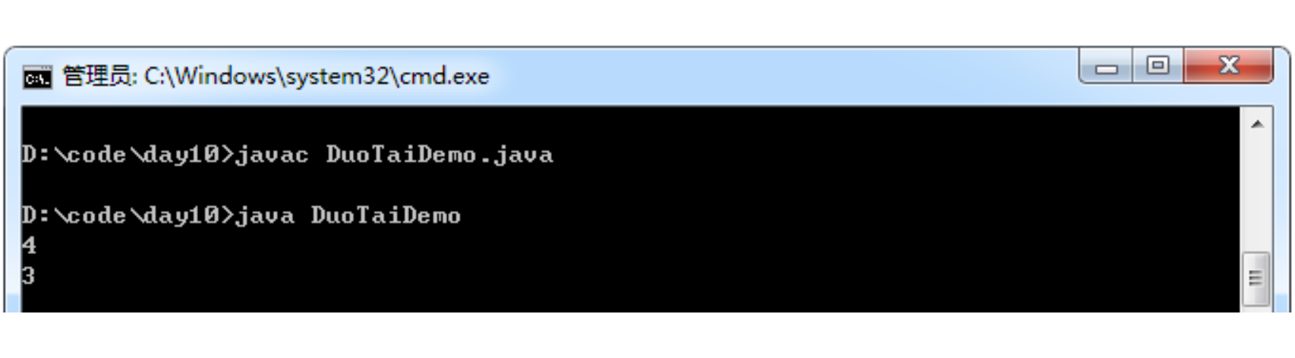
简单说：编译和运行都参考等号的左边。

示例：

```
class Fu{
        int num = 3;
}

class Zi extends Fu{
        int num = 4;
}

class DuoTaiDemo{
        public static void main(String[] args){
             Zi f1 = new Zi();
             System.out.println(f1.num);
             Fu f2 = new Zi();
             System.out.println(f2.num);
        }
}
```

复制代码

运行结果：



```
D:\code\day10>javac DuoTaiDemo.java

D:\code\day10>java DuoTaiDemo
4
3
```

2. 成员函数（非静态）

编译时：参考引用型变量所属的类中是否有调用的函数。有，编译通过。没有，编译失败。

运行时：参考的是对象所属的类中是否有调用的函数。

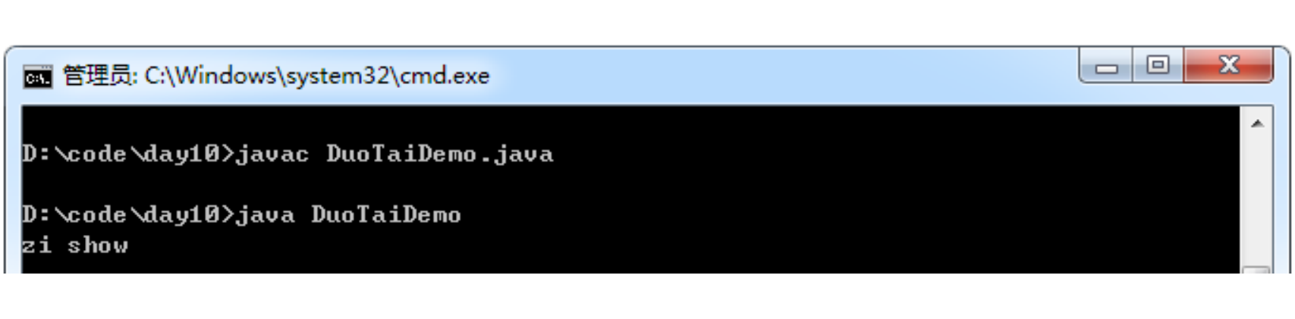
简单说：编译看左边，运行看右边。

示例：

```
class Fu{
        void show(){
             System.out.println("fu show");
        }
}

class Zi extends Fu{
        void show(){
             System.out.println("zi show");
        }
}

class DuoTaiDemo{
        public static void main(String[] args){
             Fu f = new Zi();
             f.show();
        }
}
```

复制代码

运行结果：



```
D:\code\day10>javac DuoTaiDemo.java

D:\code\day10>java DuoTaiDemo
zi show
```

3. 静态函数

编译时：参考的是对象所属的类中是否有调用的函数。

运行时：参考的是对象所属的类中是否有调用的函数。

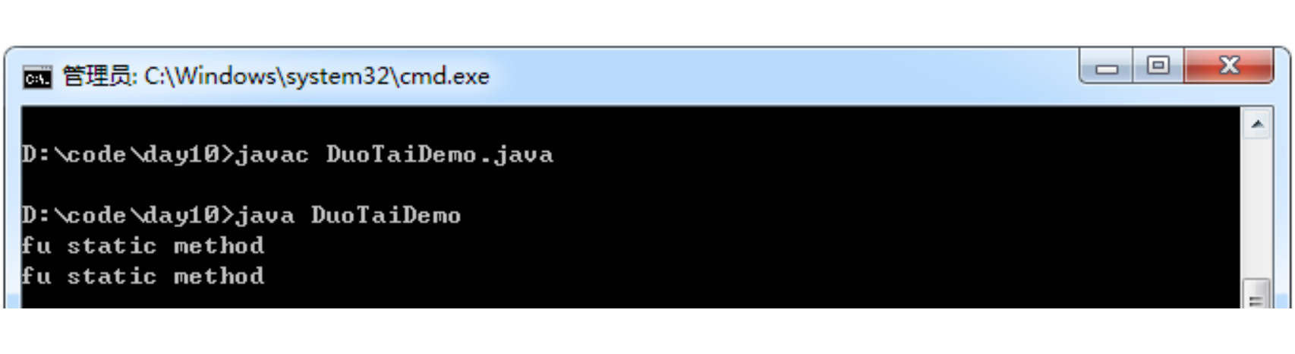
简单说：编译和运行看左边。

示例：

```
class Fu{
        static void method(){
             System.out.println("fu static method");
        }
}

class Zi extends Fu{
        static void method(){
             System.out.println("zi static method");
        }
}

class DuoTaiDemo{
        public static void main(String[] args){
             Fu f = new Zi();
             f.method();// fu static method
             Fu.method();
        }
}
```

复制代码

运行结果：



```
D:\code\day10>javac DuoTaiDemo.java

D:\code\day10>java DuoTaiDemo
fu static method
fu static method
```

4.9 内部类

定义：

将一个类定义在另一个类的里面，里面那个类就称为内部类（内置类，嵌套类）。

访问特点：

内部类可以直接访问外部类中的成员，包括私有成员。

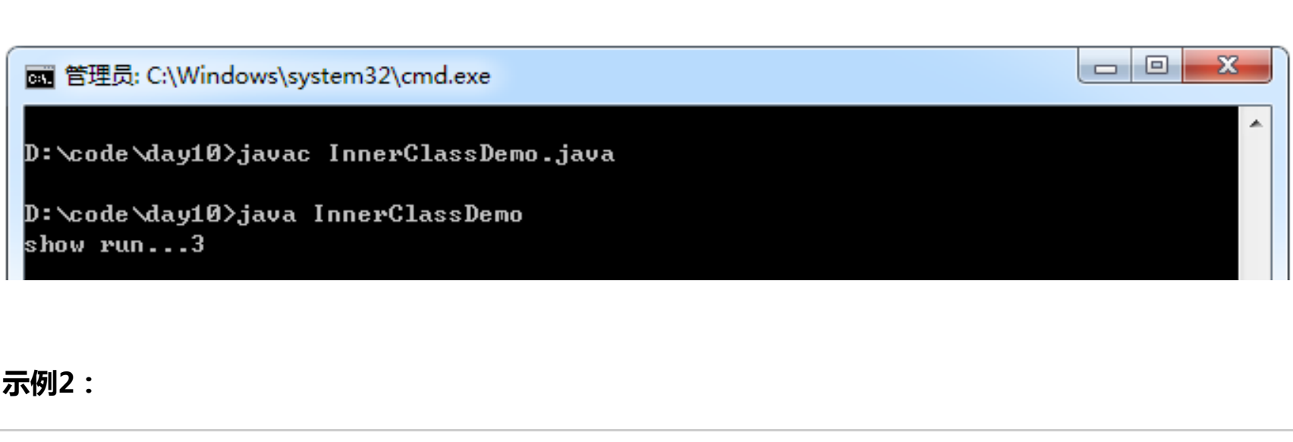
而外部类要访问内部类中的成员必须要建立内部类的对象。

示例1：

```
/*
内部类的设计：
分析事物时，发现该事物描述中还有事物，而且这个事物还在访问被描述事物的内容，这时候就定义内部类。
*/
class Outer{
      private int num = 3;

      class Inner //内部类
      {
           void show(){
               System.out.println("show run..." + num);
           }
      }

      public void method(){
          Inner in = new Inner();
          in.show();
      }
}

class InnerClassDemo{
      public static void main(String[] args){
          Outer out = new Outer();
          out.method();
      }
}
```

复制代码

运行结果：



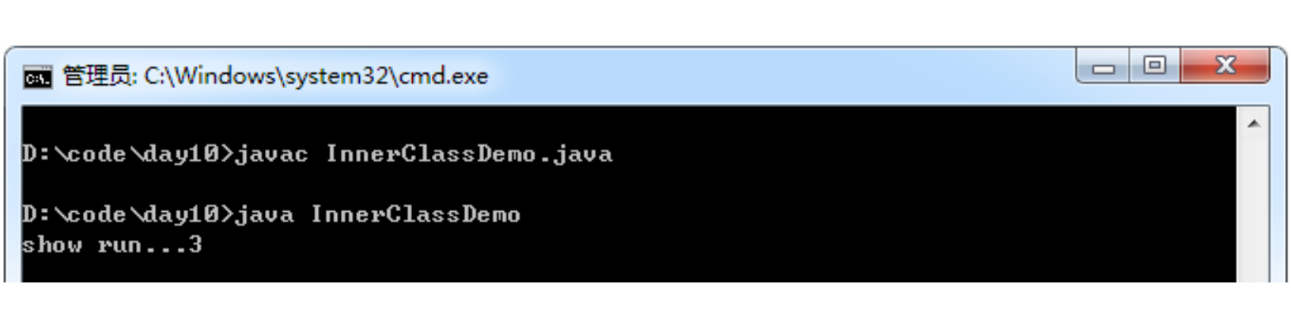
```
D:\code\day10>javac InnerClassDemo.java

D:\code\day10>java InnerClassDemo
show run...3
```

示例2：

```
class Outer{
        private int num = 3;

        class Inner
        {
              void show(){
                   System.out.println("show run..." + num);
              }
        }
}

class InnerClassDemo{
```

```
        public static void main(String[] args){
              //直接访问外部类中的内部类中的成员
             Outer.Inner in = new Outer().new Inner();
             in.show();
        }
}
```
复制代码

运行结果：


```
D:\code\day10>javac InnerClassDemo.java

D:\code\day10>java InnerClassDemo
show run...3
```

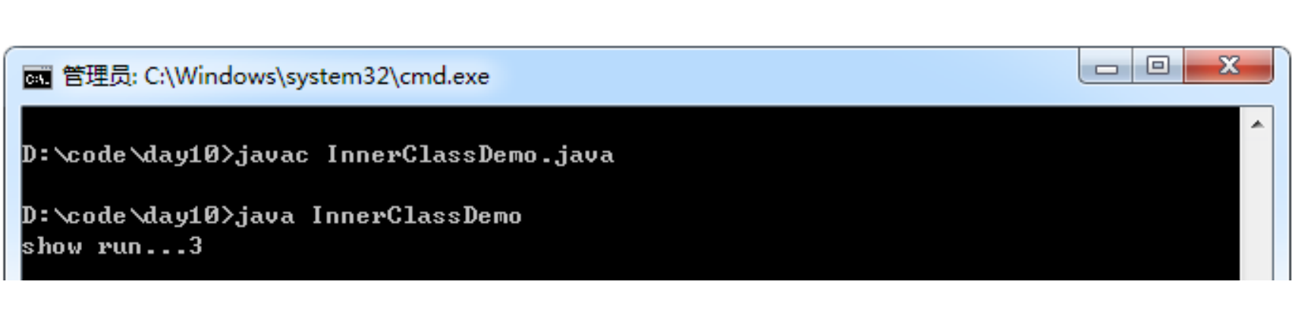
内部类的位置：

内部类定义在成员位置上，可以被private、static成员修饰符修饰。被static修饰的内部类只能访问外部类中的静态成员。

示例1：

```
class Outer{
        private static int num = 3;

        static class Inner
        {
              void show(){
                   System.out.println("show run..." + num);
              }
        }
}

class InnerClassDemo{
        public static void main(String[] args){
              //如果内部类是静态的，相当于一个外部类
             Outer.Inner in = new Outer.Inner();
             in.show();
        }
}
```
复制代码

运行结果：


```
D:\code\day10>javac InnerClassDemo.java

D:\code\day10>java InnerClassDemo
show run...3
```

示例2：如果内部类是静态的，内部类成员也是静态的，可以不用创建内部类对象，直接调用。

```
class Outer{
        private static int num = 3;

        static class Inner
        {
              static void show(){
                   System.out.println("show run..." + num);
              }
        }
}

class InnerClassDemo{
        public static void main(String[] args){
             Outer.Inner.show();
        }
}
```
复制代码

运行结果：


```
D:\code\day10>javac InnerClassDemo.java

D:\code\day10>java InnerClassDemo
show run...3
```

P.S.

1、如果内部类中定义了静态成员，该内部类也必须是静态的！

示例：

```
class Outer{
      private static int num = 3;

      static class Inner
      {
           static void show(){
                System.out.println("show run..." + num);
           }
      }
}
```
复制代码

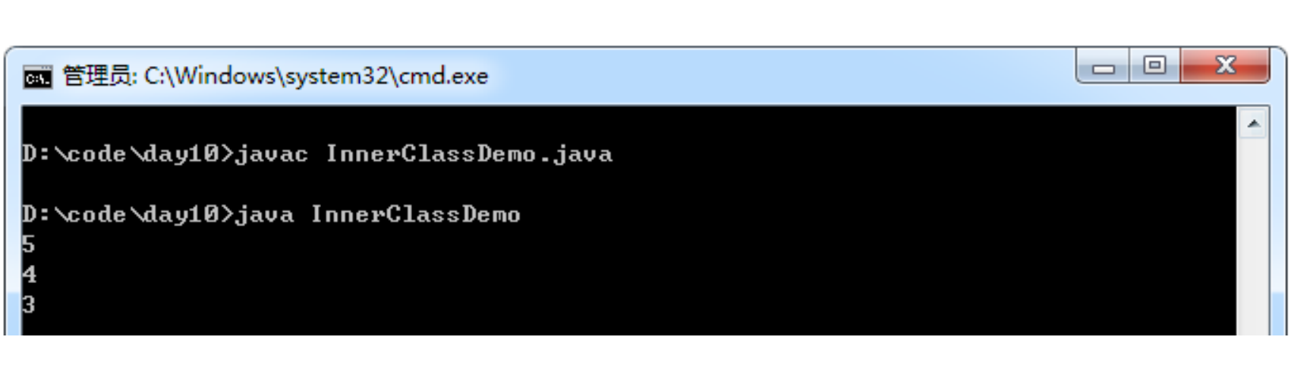
2、为什么内部类能直接访问外部类中的成员呢？

那是因为内部类持有了外部类的引用，外部类名.this。

示例：

```
class Outer{
        int num = 3;
        class Inner{
              int num = 4;
              void show(){
                    int num = 5;
                    System.out.println(num);
                    System.out.println(this.num);
                    System.out.println(Outer.this.num);
              }
        }
        void method(){
              new Inner().show();
        }
}

class InnerClassDemo{
        public static void main(String[] args){
              new Outer().method();
        }
}
```
复制代码

运行结果：


```
D:\code\day10>javac InnerClassDemo.java

D:\code\day10>java InnerClassDemo
5
4
3
```

3、内部类定义在局部位置上，也可以直接访问外部类中的成员。

同时可以访问所在局部中的局部变量，但必须是被final修饰的。

示例：

```
class Outer{
        int num = 3;
        void method(final int y){
              final int x = 9;
              class Inner{
                    void show(){
                         System.out.println("show..." + x + "," + y);
                    }
              }
             Inner in = new Inner();
             in.show();
        }
}

class InnerClassDemo{
        public static void main(String[] args){
              new Outer().method(4);
        }
}
```
复制代码

运行结果：


```
D:\code\day10>javac InnerClassDemo.java

D:\code\day10>java InnerClassDemo
show...9,4
```

匿名内部类

定义：

就是内部类的简化写法。

前提：

内部类可以继承或实现一个外部类或者接口。

格式：

new 外部类名或者接口名(){覆盖类或者接口中的代码，(也可以自定义内容。)}

简单理解：

就是建立一个带内容的外部类或者接口的子类匿名对象。

什么时候使用匿名内部类呢？

通常使用方法是接口类型参数，并且该接口中的方法不超过三个，可以将匿名内部类作为参数传递。

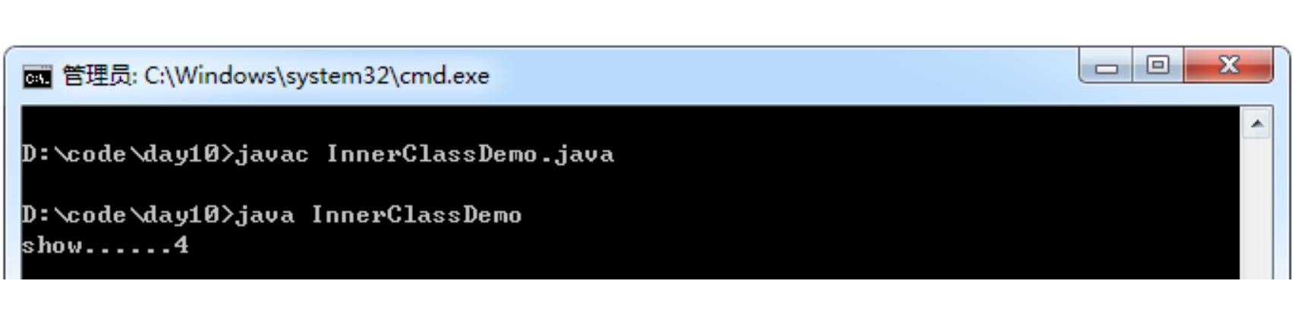
好处：

增强阅读性。

示例1：

```
abstract class Demo{
        abstract void show();
}

class Outer{
        int num = 4;

        void method(){
              new Demo(){//匿名内部类
                    void show(){
                         System.out.println("show......" + num);
                   }
              }.show();
        }
}

class InnerClassDemo{
        public static void main(String[] args){
              new Outer().method();
        }
}
```
复制代码

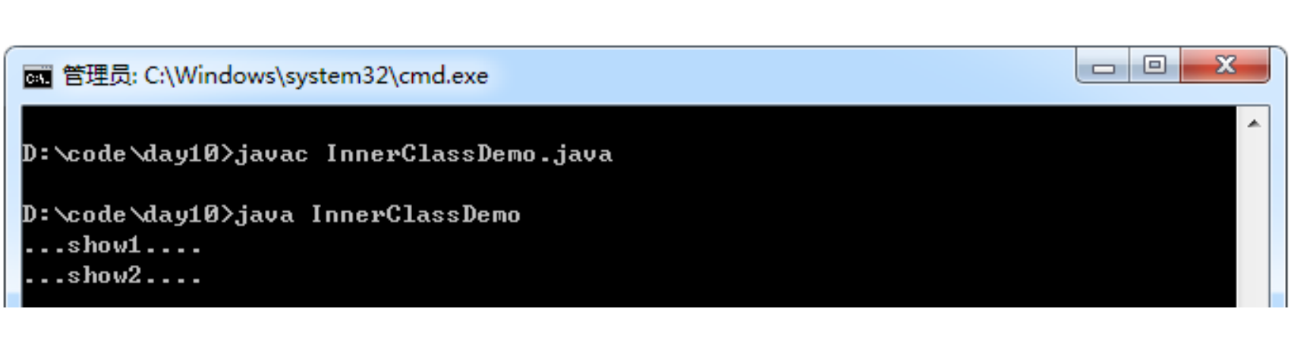
运行结果：


```
D:\code\day10>javac InnerClassDemo.java

D:\code\day10>java InnerClassDemo
show......4
```

示例2：

```
interface Inter{
        void show1();
        void show2();
}

class Outer{
        public void method(){
             Inter in = new Inter(){
                    public void show1(){
                           System.out.println("...show1...." );
                    }
                    public void show2(){
                           System.out.println("...show2...." );
                    }
             };
             in.show1();
             in.show2();
        }
}

class InnerClassDemo{
        public static void main(String[] args){
              new Outer().method();
        }
}
```
复制代码

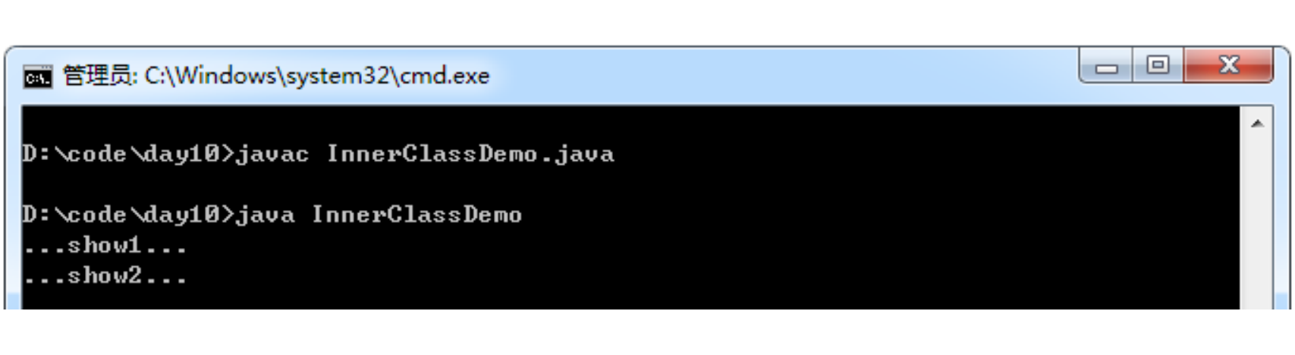
运行结果：


```
D:\code\day10>javac InnerClassDemo.java

D:\code\day10>java InnerClassDemo
...show1....
...show2....
```

示例3：

```
interface Inter{
        void show1();
        void show2();
}

/*
通常的使用场景之一：
当函数参数是接口类型时，而且接口中的方法不超过三个。
可以用匿名内部类作为实际参数进行传递。
*/
class InnerClassDemo{
       public static void main(String[] args){
              show(new Inter(){
                    public void show1(){
                         System.out.println("...show1..." );
                    }
                    public void show2(){
                         System.out.println("...show2..." );
                    }
             });
        }
        public static void show(Inter in){
             in.show1();
             in.show2();
        }
}
```
复制代码

运行结果：

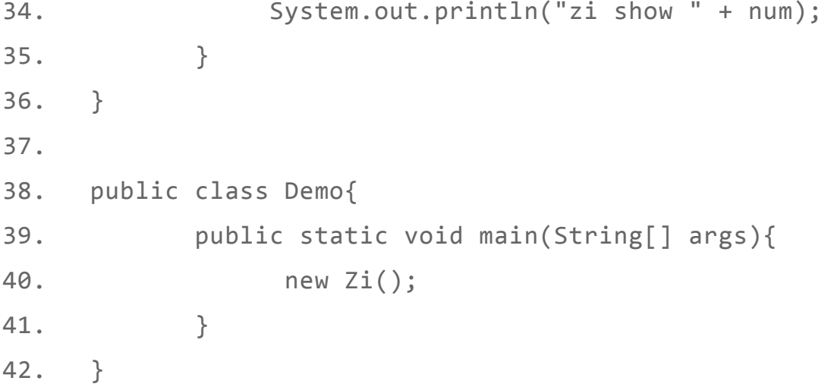

```
D:\code\day10>javac InnerClassDemo.java

D:\code\day10>java InnerClassDemo
...show1...
...show2...
```

对象的初始化过程

示例：

```
class Fu{
        int num = 9;

        {
             System.out.println("Fu" );
        }

       Fu(){
              super();//Object
              //显示初始化
              //构造代码块初始化
             show();
        }
        void show(){
             System.out.println("fu show " + num);//被覆盖，运行子类的
        }
}

class Zi extends Fu{
        int num = 8;

       {
             System.out.println("Zi" );
       }

       Zi(){
              super();
              //显示初始化
              //构造代码块初始化
             show();
       }

        void show(){
             System.out.println("zi show " + num);
        }
}

public class Demo{
        public static void main(String[] args){
              new Zi();
        }
}
```
复制代码

运行结果：


```
D:\code\day10>javac Demo.java

D:\code\day10>java Demo
Fu
zi show 0
Zi
zi show 8
```

~END~

~爱上海，爱黑马~