

第5章 类和对象

Java 是一门面向对象的语言，其重要的一个思想就是“万物皆对象”。而类是 Java 的核心内容，它是一种逻辑结构，定义了对象的结构，可以由一个类得到众多相似的对象。从某种意义上说，类是 Java 面向对象性的基础。Java 与 C++ 不同，它是一门完全的面向对象语言，它的任何工作都要在类中进行。这一章的内容主要讲解 Java 类和对象，包括类的使用以及类中的属性、方法、构造函数、对象、方法参数传递以及 Java 垃圾回收等内容。

5.1 类

类实际上是定义一个模板，而对象是由这个模板产生的一个实例。实际上前面的程序中也是在类中实现的，不过全在类中的 `main` 方法中演示程序的使用，没有体现面向对象编程的思想。这一节里主要讲解 Java 类的相关知识，包括类的形式、类包含的内容属性和方法。

5.1.1 类的一般形式

Java 的重要思想是万物皆对象，也就是说在 Java 中把所有现实中的一切人和物都看做对象，而类就是它们的一般形式。程序编写就是抽象出这些事物的共同点，用程序语言的形式表达出来。

例如，可以把某某人看做一个对象，那么就可以把人作为一个类抽象出来，这个人就可以作为人这个类的一个对象。类的一般形式如下。

```
class 类名{
    类型 实例变量名;
    类型 实例变量名;
    .....
    类型 方法名(参数){
        //方法内容
    }
    .....
}
```

人的一般属性包括姓名、性别、年龄、住址等，他的行为可以有工作、吃饭等内容。这样人这个类就可以有如下定义。

```
class Human
{
    //声明各类变量来描述类的属性
    String name;
    String sex;
    int age;
    String addr;
    void work(){
```

```

System.out.println("我在工作");
}
void eat(){
System.out.println("我在吃饭");
}
}

```

需要注意的是，在类名面前并没有向以前那样加上修饰符 `public`，在 Java 中是允许把许多类的声明放在一个 Java 中的，但是这些类只能有一个类被声明为 `public`，而且这个类名必须和 Java 文件名相同。这里主要讲解 Java 的一般形式，只使用类的最简单形式，便于读者理解。关于修饰符这里先做简单的说明。

- ❑ `private`：只有本类可见。
- ❑ `protected`：本类、子类、同一包的类可见。
- ❑ 默认（无修饰符）：本类、同一包的类可见。
- ❑ `public`：对任何类可见。

类的一些描述性的属性，如人这个类中的姓名、性别、年龄、住址这些内容，可以看做类的字段。本书对他们使用的称谓是实例变量，人这个类定义了以下 4 个实例变量：

```

String name;
String sex;
int age;
String addr;

```

当然，为了说明类的最简单形式，并没有对这些变量加上修饰符，这样的效果是使用默认修饰符。可以用以下形式声明这些变量。

```

public String name;
public String sex;
public int age;
public String addr;

```

实际上在类中的实例变量最好被声明为 `private`，这更符合面向对象的封装性原则，这是后面的内容，暂且先不考虑，只讲如何使用它们而暂且不涉及使用的合理性。

方法从返回类型上可分为有返回值和无返回值两类。当不需要返回值时把方法用关键字 `void` 修饰，表示该方法无返回值。如果有返回值的话，方法的类型定义必须和方法的返回值相同。如果想要方法返回一个字符串类型，就要有如下声明。

```

public String returnString(){
//方法体
return "a String";
}

```

当然修饰符是根据需要确定的。如果方法需要返回一个 `int` 型，方法的类型也必须为 `int` 型，否则程序编译会报错。有时候需要给方法传递参数，就需要使用带参数的方法，这种方法的一般形式如下。

```

类型 方法名(参数类型 参数 1,参数类型 参数 2...){
//方法体
}

```

下面的程序示意了一个带参数的方法。

```

class Human {

```

```

String name;
String sex;
int age;
String addr;
void work() {
    System.out.println("我在工作");
}
void eat() {
    System.out.println("我在吃饭");
}
//定义一个方法，返回值为 String 类型
String getState(int time) {
    String state = null;
    if (time >= 0 && 24>=time) {
        if (time > 8 && time < 17)
            state = "我在工作";
        else if (time > 17 && time < 22)
            state = "我在学习";
        else if (time > 22 || time < 7)
            state = "我在睡觉";
    } else
        state = "错误的时间";
    return state;
}
}
//测试类
public class MethodDemo {
    public static void main(String args[] ) {
        Human wangming = new Human();
        //设定对象的属性值
        wangming.name = "王明";
        wangming.age = 25;
        wangming.sex = "男";
        wangming.addr = "中国北京";
        System.out.println(wangming.name+"晚上 23 点钟你在干嘛");
        //调用 getState()方法，把返回值打印出来
        System.out.println(wangming.getState(23));
        System.out.println("下午 15 点呢");
        System.out.println(wangming.getState(15));
    }
}

```

程序给 `Human` 类添加了一个方法 `getState`，这个方法有一个整型参数 `time` 表示时间，当然可以用更准确的表达方式，但这里主要讲解参数的使用，用简单的示例更容易明白。在方法中，根据不同的调用传递的时间 `time`，返回不同的字符串表示当时的状态。在 `MethodDemo` 类中，首先声明了一个 `Human` 对象，然后对它的一系列实例变量赋值。调用 `getState` 方法，两次传递不同的参数过去。程序的运行结果如下。

```

王明晚上 23 点钟你在干嘛
我在睡觉
下午 15 点呢
我在工作

```

5.1.2 方法的重载

在 Java 中支持有两个或多个同名的方法，但是它们的参数个数和类型必须有差别。这种情况就是方法重载（overloading）。重载是 Java 实现多态的方式之一。

当调用这些同名的方法时，Java 根据参数类型和参数的数目来确定到底调用哪一个方法，注意返回值类型并不起到区别方法的作用。下面是一个方法重载的示例程序。

```
public class OverloadDemo{
    //定义一系列的方法，这些方法的参数是不同的，通过参数来区别调用的方法
    void method(){
        System.out.println("无参数方法被调用");
    }
    void method(int a){
        System.out.println("参数为 int 类型被调用");
    }
    void method(double d){
        System.out.println("参数为 double 方法被调用");
    }
    void method(String s){
        System.out.println("参数为 String 方法被调用");
    }
    public static void main(String args[ ]){
        OverloadDemo ov=new OverloadDemo();
        //使用不同的参数调用方法
        ov.method();
        ov.method(4);
        ov.method(4.5D);
        ov.method("a String");
    }
}
```

程序的运行结果如下。

```
无参数方法被调用
参数为 int 类型被调用
参数为 double 方法被调用
参数为 String 方法被调用
```

当参数类型并不能完全匹配时候，Java 的自动类型转换会起作用。示例如下。

```
public class OverloadDemo2{
    void method(){
        System.out.println("无参数方法被调用");
    }
    void method(double d){
        System.out.println("参数为 double 方法被调用");
    }
    void method(String s){
        System.out.println("参数为 String 方法被调用");
    }
    public static void main(String args[ ]){
        OverloadDemo2 ov=new OverloadDemo2();
        ov.method();
        ov.method(4);
        ov.method(4.5D);
    }
}
```

```

    ov.method("a String");
}
}

```

程序跟 OverloadDemo 基本相同，只是去掉了参数为 `int` 类型的方法，程序的运行结果如下。

```

无参数方法被调用
参数为 double 方法被调用
参数为 double 方法被调用
参数为 String 方法被调用

```

当调用 `method(4)` 时候找不到 `method(int)`, Java 就找到了最为相似的方法 `method(double)`。这种情况只有在没有精确匹配时才发生。

5.2 对象

还是那句话“万物皆对象”。前面已经把描述对象的类抽象出来，已经有了建立对象的模板，接下来的工作就是创造这些类的实例，既对象。在本节中就来学习如何创建对象，如何使用创建后的对象，以及其他一些相关内容。

5.2.1 对象的创建和使用

要想使用一个对象，需要首先创建它。创建一个对象实际上分为两步来完成。首先，声明一个该类类型的变量，这个变量并不是对象本身，而是通过它可以引用一个实际的对象。然后，获得类的一个实例对象把它赋值给该变量，这个过程是通过 `new` 运算符完成的。`new` 运算符完成的实际工作是为对象分配内存。

在 Java 程序中，上边的两个过程如下所示，这里使用前一节讲过的 `Human` 类来创建对象，假设有一个人叫“王明”，创建一个对象存放他的信息。

```

Human wangming;
wangming=new Human();

```

第一行声明一个 `Human` 类的变量 `wangming`，第二行通过 `new` 运算符获得一个对象实例并为其分配内存，获得对象实例赋值给 `wangming`。上面的过程可以合并为一个语句。

```

Human wangming=new Human();

```

这两种形式使用的效果是相同的，这样获得的对象就包含 4 个实例变量。描述这个对象的信息，可以调用的方法有 `eat` 和 `work`，调用方法是通过“.”运算符实现的，如下所示。

```

wangming.eat();

```

该语句执行的结果是在控制台打印出“我在吃饭”。下面的程序简单地演示了对象的声明以及使用。

```

public class HumanDemo{

```

```

public static void main(String args[] ){
    //创建一个对象
    Human wangming;
    wangming=new Human();
    //对对象的实例变量赋值
    wangming.name="王明";
    wangming.age=25;
    wangming.sex="男";
    wangming.addr="中国北京";
    System.out.println("姓名: "+wangming.name);
    System.out.println("性别: "+wangming.sex);
    System.out.println("年龄: "+wangming.age);
    System.out.println("地址: "+wangming.addr);
    System.out.println("在干什么?");
    wangming.eat();
}
}

```

程序的运行结果如下。

```

姓名: 王明
性别: 男
年龄: 25
地址: 中国北京
在干什么?
我在吃饭

```

对象之间也是可以进行赋值运算的，如下所示。

```

Human zhangsan;
Human lisi;
zhangsan=new Human();
zhangsan.name="张三";
zhangsan.age=30;
zhangsan.sex="男";
zhangsan.addr="中国北京";
lisi=zhangsan;

```

首先是声明了两个 `Human` 变量 `zhangsan` 和 `lisi`，然后创建 `zhangsan` 并初始化它的一系列实例变量，最后把 `zhangsan` 赋值给 `lisi`。注意最后一句的作用仅仅是把 `lisi` 也指向原来 `zhangsan` 指向的对象，也就是现在 `zhangsan` 和 `lisi` 指向同一个对象，改变 `lisi` 的话，`zhangsan` 也跟着改变。下面的程序演示了这一点。

```

class Human{
    String name;
    String sex;
    int age;
    String addr;
    void work(){
        System.out.println("我在工作");
    }
    void eat(){
        System.out.println("我在吃饭");
    }
}

public class HumanDemo2 {

```

```

public static void main(String args[] ) {
//创建两个对象
Human zhangsan = new Human();
    Human lisi=new Human();
    //对 zhangsan 赋值
    zhangsan.name="张三";
    zhangsan.sex="男";
    zhangsan.age=25;
    zhangsan.addr="中国北京";
    //把 zhangsan 赋值给 lisi
    lisi=zhangsan;
    //打印出赋值后的结果
    System.out.print("张三的姓名: ");
    System.out.println(zhangsan.name);
    System.out.print("李四的姓名: ");
    System.out.println(lisi.name);
    System.out.println("改变李四的名字");
    lisi.name="李四";
    System.out.print("现在李四的名字为: ");
    System.out.println(lisi.name);
    System.out.print("现在张三的名字为: ");
    System.out.println(zhangsan.name);
}
}

```

程序首先对 zhangsan 的实例变量进行了赋值；然后把 zhangsan 赋值给 lisi，访问 lisi 的属性发现跟 zhangsan 的是一样的；最后改变 lisi 的名字发现张三的名字也改变了，说明了这两个变量都指向同一个对象。程序的运行结果如下。

```

张三的姓名: 张三
李四的姓名: 张三
改变李四的名字
现在李四的名字为: 李四
现在张三的名字为: 李四

```

5.2.2 构造函数

创建对象使用的语句如下。

```
Human zhangsan=new Human();
```

实际上是调用了一个方法，不过这个方法是系统自带的方法，由于这个方法被用来构造对象，所以把它称为构造函数。构造函数的作用是生成对象，并对对象的实例变量进行初始化。系统自带的默认构造函数把所有的数字变量设为 0，把所有的 boolean 型变量设为 false，把所有的对象变量都设为 null。Human 类的默认构造函数的实际效果如下。

```

Human(){
name=null;
age=0;
sex=null;
addr=null;
}

```

通过下面的程序进行验证。

```
class Human{
    String name;
    String sex;
    int age;
    String addr;
    void work(){
        System.out.println("我在工作");
    }
    void eat(){
        System.out.println("我在吃饭");
    }
}

public class ConstructorDemo1
{
    public static void main(String args[ ])
    {
        //创建一个 zhangsan 对象
        Human zhangsan=new Human();
        //打印出 zhangsan 的属性的默认值
        System.out.println("姓名默认值"+zhangsan.name);
        System.out.println("性别默认值"+zhangsan.sex);
        System.out.println("年龄默认值"+zhangsan.age);
        System.out.println("地址默认值"+zhangsan.addr);
    }
}
```

程序的运行结果如下。

```
姓名默认值 null
性别默认值 null
年龄默认值 0
地址默认值 null
```

构造函数有一个很明显的特点是它的名字必须跟类名相同，并且没有返回值类型。下面是 Human 类一个简单的构造函数。

```
public Human(){
    name=null;
    age=0;
    sex=null;
    addr=null;
}
```

把构造函数加入类中，类的完整定义如下。

```
class Human
{
    //属性
    String name;
    String sex;
    int age;
    String addr;
    //构造方法
    public Human()
    {
        name=null;
    }
}
```



```

    age=0;
    sex=null;
    addr=null;
}
//work 方法
void work(){
    System.out.println("我在工作");
}
//eat 方法
void eat(){
    System.out.println("我在吃饭");
}
}

```

构造函数的主要作用是用来对对象的变量进行初始化。如果不想把它们都初始化为默认值，就需要自己编写构造函数，通过有参数的构造函数可以把值传递给对象的变量。可以把 Human 类的构造函数定义如下。

```

public Human(String hName,String hSex,int hAge,String hAddr){
    name=hName;
    sex=hSex;
    age=hAge;
    addr=hAddr;
}

```

创建对象时可以使用如下语句，这样生成对象 zhangsan 的属性就被初始化为相应的值。

```
Human zhangsan=new Human("张三","男",25,"中国北京");
```

Human 类的完整定义如下。

```

class Human{
    String name;
    String sex;
    int age;
    String addr;
    public Human(String hName,String hSex,int hAge,String hAddr){
        name=hName;
        sex=hSex;
        age=hAge;
        addr=hAddr;
    }
    void work(){
        System.out.println("我在工作");
    }
    void eat(){
        System.out.println("我在吃饭");
    }
}

```

由于已经定义了自己的构造函数，所以 Java 不再提供默认的构造函数。这时候如果再使用默认的构造函数，即 `Human lisi=new Human();` 编译时会报错。只能使用程序中提供的构造函数来创建对象。演示构造函数用的 demo 程序如下。

```

public class HumanDemo3
{
    public static void main(String args[ ])

```

```

{
//创建两个对象
    Human zhangsan=new Human("张三","男",25,"中国北京");
    Human lisi=new Human("李四","男",20,"中国上海");
//Human asu=new Human();//该语句会报错
//分别打印出两个对象的信息
    System.out.println("张三的信息");
    System.out.println("姓名: "+zhangsan.name);
    System.out.println("性别: "+zhangsan.sex);
    System.out.println("年龄: "+zhangsan.age);
    System.out.println("地址: "+zhangsan.addr);
    System.out.println("李四的信息");
    System.out.println("姓名: "+lisi.name);
    System.out.println("性别: "+lisi.sex);
    System.out.println("年龄: "+lisi.age);
    System.out.println("地址: "+lisi.addr);
}
}

```

程序的运行结果如下。

```

张三的信息
姓名: 张三
性别: 男
年龄: 25
地址: 中国北京
李四的信息
姓名: 李四
性别: 男
年龄: 20
地址: 中国上海

```

可以看到，生成的对象的信息已经由提供的参数进行了初始化，这时候程序编写可能提出进一步的要求，需要每次创造对象的模板是不同的。可有时候并不需要对全部的变量进行初始化，而只对其中的一部分初始化即可。前面已经讲过方法的重载，Java 也提供了构造函数的重载，读者可以灵活地根据不同的需要利用不同的构造函数进行对象的创建。增加几个构造函数的 Human 类定义如下。

```

class Human
{
    String name;
    String sex;
    int age;
    String addr;
    public Human()
{
        name=null;
        age=0;
        sex=null;
        addr=null;
    }
    public Human(String hName,String hSex){
        name=hName;
        sex=hSex;
    }
}

```

```

    }
    public Human(String hName, String hSex, int hAge, String hAddr) {
        name = hName;
        sex = hSex;
        age = hAge;
        addr = hAddr;
    }
    void work() {
        System.out.println("我在工作");
    }
    void eat() {
        System.out.println("我在吃饭");
    }
}

```

在 `Human` 类中提供了 3 个不同的构造函数以满足构造对象是不同的需要，演示程序如下。

```

public class HumanDemo4 {
    public static void main(String args[] ) {
        Human zhangsan=new Human();
        Human qq=new Human("青青","女");
        Human lisi=new Human("李四","男",20,"中国上海");
        System.out.println("张三的信息:");
        System.out.println(" 姓 名 :"+zhangsan.name+"\n 性 别 : "+zhangsan.sex+"\n 年 龄 : "+zhangsan.age+"\n 地址: "+zhangsan.addr);
        System.out.println("青青的信息:");
        System.out.println("姓名:"+qq.name+"\n 性别: "+qq.sex+"\n 年龄: "+qq.age+"\n 地址: "+qq.addr);
        System.out.println("李四的信息:");
        System.out.println("姓名:"+lisi.name+"\n 性别: "+lisi.sex+"\n 年龄: "+lisi.age+"\n 地址: "+lisi.addr);
    }
}

```

程序的运行结果如下。

```

张三的信息:
姓名:null
性别: null
年龄: 0
地址: null
青青的信息:
姓名:青青
性别: 女
年龄: 0
地址: null
李四的信息:
姓名:李四
性别: 男
年龄: 20
地址: 中国上海

```

由程序的运行结果可以看出，创建 3 个对象分别使用了 3 个不同的构造函数，起到了不同的效果，尽管演示程序的使用意义不大，但这里主要演示构造函数的重载，Java 的这个功能在很多情况下非常有用。在 Java 中还有一项重要的功能，可以在构造函数中调用其他的构造函数。重新定义 `Human` 类，如下所示。

```

class Human {
    String name;
    String sex;
    int age;
    String addr;
    public Human() {
        name = null;
        age = 0;
        sex = null;
        addr = null;
    }
    public Human(String hName, String hSex) {
        name = hName;
        sex = hSex;
        System.out.println("第二个构造函数被调用");
    }
    public Human(String hName, String hSex, int hAge, String hAddr) {
        this(hName, hSex);
        age = hAge;
        addr = hAddr;
        System.out.println("第三个构造函数被调用");
    }
    void work() {
        System.out.println("我在工作");
    }
    void eat() {
        System.out.println("我在吃饭");
    }
}

```

执行下面的演示程序。

```

public class HumanDemo5 {
    public static void main(String args[] ) {
        Human lisi = new Human("李四", "男", 20, "中国上海");
        System.out.println("李四的信息:");
        System.out.println("姓名:" + lisi.name + "\n 性别: " + lisi.sex + "\n 年龄: "
            + lisi.age + "\n 地址: " + lisi.addr);
    }
}

```

程序的运行结果如下。

```

第二个构造函数被调用
第三个构造函数被调用
李四的信息:
姓名:李四
性别: 男
年龄: 20
地址: 中国上海

```

可以看到在通过第三个构造函数创建对象时，首先通过 `this` 关键字调用了第二个构造函数。

5.3 static 关键字

在编程的过程中，有一些是对象公用的数据，如果每个对象都需要这个数据，那么数据的同步将是很大的问题。假设在一个学生类中，要描述学生的信息，需要有一个 `int` 型的数据类型来描述当前学生的个数，如果给每一个学生这样一个值，并且每增加一个人都需要更新所有对象的属性值，显然造成了很大的不便，所以 `Java` 提供了静态变量的概念来解决这个问题。静态变量是用 `static` 来描述的。

5.3.1 静态变量

普通的各种变量都是属于某个对象的，有一个对象就有一个这个数据的副本。静态变量则是整个类只有一个变量，它跟对象是没有关系的，跟它相关联的是类而不是对象。示例如下。

```
public class StaticDemo1
{
    int commanint=0;
    //声明 static 变量
    static int staticint=0;
    StaticDemo1(int x)
    {
        commanint=x;
    }
    public static void main(String args[ ])
    {
        //创建两个对象
        StaticDemo1 s1=new StaticDemo1(1);
        StaticDemo1 s2=new StaticDemo1(2);
        //下面的语句是对比静态变量和普通变量的使用
        System.out.println("s1.commanint="+s1.commanint);
        System.out.println("s2.commanint="+s2.commanint);
        System.out.println("s1.staticint="+s1.staticint);
        System.out.println("s2.staticint="+s2.staticint);
        s1.commanint++;
        System.out.println("改变 commanint 的值");
        System.out.println("s1.commanint="+s1.commanint);
        System.out.println("s2.commanint="+s2.commanint);
        System.out.println("s1.staticint="+s1.staticint);
        System.out.println("s2.staticint="+s2.staticint);
        s1.staticint++;
        System.out.println("通过 s1 改变 staticint 的值为:"+s1.staticint);
        System.out.println("s2 的 staticint 的值为:"+s2.staticint);
    }
}
```

在程序中首先声明了该类的两个对象，然后通过其中的一个对象分别改变 `commanint`、`staticint` 的值，最后打印出两个对象的对象值。程序的运行结果如下。

```

s1.commanint=1
s2.commanint=2
s1.staticint=0
s2.staticint=0
改变 commanint 的值
s1.commanint=2
s2.commanint=2
s1.staticint=0
s2.staticint=0
通过 s1 改变 staticint 的值为:1
s2 的 staticint 的值为:1

```

通过一个对象改变静态变量的值，可以看到两个对象的值都改变了。其实该变量作为类变量，所有的对象只是共有一个值，通过一个变量改变该值，其他的对象访问它自然也是改变了之后的值。实际上上面的程序的访问静态变量的方法是不规范的，规范的方法是使用“类名.变量名”的方式来访问的，这样能增加程序的可读性。示例如下。

```

public class Student
{
    int id=0;
    static int studentNum=0;
    Student(int x)
    {
        id=x;
        studentNum++;
    }
    public static void main(String args[ ])
    {
        Student s1=new Student(1000);
        Student s2=new Student(1001);
        Student s3=new Student(1002);
        Student s4=new Student(1003);
        System.out.println("s1 的学号:"+s1.id);
        System.out.println("s2 的学号:"+s2.id);
        System.out.println("s3 的学号:"+s3.id);
        System.out.println("s4 的学号:"+s4.id);
        System.out.println("学生的数目:"+Student.studentNum);
    }
}

```

程序中 id 是学生的学号，studentNum 用来计算学生的数量，Student.studentNum 用来访问该变量。程序的运行结果如下。

```

s1 的学号:1000
s2 的学号:1001
s3 的学号:1002
s4 的学号:1003
学生的数目:4

```

5.3.2 静态方法

还可以用 static 来描述方法，用 static 描述的方法称为静态方法。访问静态方法也是使用

“类名.方法名”的方式来访问。一般在工具类中定义一些静态方法来处理一些事情，这样可以方便地使用这些静态方法。在 Java 常用的数学工具类 **Math** 类中的方法大多数是静态的，可以很方便地访问。示例如下。

```
public class StaticMethod
{
    public static void main(String args[ ])
    {
        System.out.println("用静态方法打印出信息");
        MyMethod.printString("str 类型");
        MyMethod.printInt(5);
    }
}
class MyMethod
{
    //静态方法的声明
    static void printString(String str)
    {
        System.out.println(str);
    }
    static void printInt(int i)
    {
        System.out.println(i);
    }
}
```

上面的程序中定义了一个 **MyMethod** 类，在这个类中定义了两个静态方法，这两个方法会把传递的参数打印出来。程序的运行结果如下。

```
用静态方法打印出信息
str 类型
5
```

使用静态方法的时候需要注意以下两点。

- ❑ 静态方法不能直接访问非静态变量。
- ❑ 非静态方法可以直接访问静态变量。

静态方法访问非静态变量的示例如下。

```
public class Demo1
{
    int x=0;
    public static void main(String args[ ])
    {
        //在非静态方法中调用静态变量，非法
        System.out.println(x);
    }
}
```

程序在编译的时候会报错，如下所示。

```
Demo1.java:6: 无法从静态上下文中引用非静态 变量 x
        System.out.println(x);
                        ^
1 错误
```

5.3.3 静态常量

在实际应用中，静态变量不是很常用，最常用的是静态常量，用来存储一些在程序中不会改变的信息。静态常量的定义如下所示。

```
public static final int X=123;
```

该变量 X 在程序中是不可以改变的，不能通过赋值改变它的值。静态变量、静态常量的初始化是调用构造函数之前完成的，示例如下。

```
public class Demo2
{
    static int x=0;
    static
    {
        x=100;
    }
    Demo2()
    {
        System.out.println(x);
    }
    public static void main(String args[ ])
    {
        //创建一个对象，创建对象之前会先执行 static 代码块，注意程序输出
        Demo2 d=new Demo2();
    }
}
```

程序中的静态变量 x 的初始值为 0，在一个静态语句块中对其进行了赋值，然后调用了该类的构造函数，构造函数是打印出 x 的值。程序的运行结果如下。

```
100
```

可以看到程序的输出是 100，也就是说 static 语句块在程序执行之前就已经执行过了。静态语句块的执行实际上是在类加载的时候执行，在程序执行之前加载工作已经完成了，可以把静态变量的初始化放在静态语句块中，这样就能提前完成它们的初始化。

5.4 参数传递

在各种程序设计语言中，参数传递一般有两种，一种是“传值”，另一种是“传地址”。传值是指在调用方法时，把参数的值直接传递给方法，而传地址则是给方法提供参数的地址。Java 的参数传递方法都为传值调用，但是当涉及到对象的传递时，它又不是简单的传值，所以比较复杂，下面将通过一些实例进行讨论，主要分为简单数据类型和对象引用两类。

5.4.1 基本类型的参数传递

对于基本类型的参数传递，前面的方法已经使用过，但是并没有详细的讲解它的过程。由于 Java 中使用的都是传值引用，所以参数得到的都是参数值的拷贝。示例如下。


```
void amethod(int i){
    i=i*5;
}
```

通过对象调用本方法。

```
int money=100;
对象.amethod(money);
```

执行这个操作的时候有下面几个过程：首先 `i` 被初始化为一个值与 `money` 相等的变量，在方法中把这个变量变为原来的 5 倍，这时候 `i` 的值为 500，而 `money` 的值不变，方法执行完成后，变量 `i` 由于是临时变量被抛弃。下面的程序演示了这个状态。

```
public class ParamTransfer
{
    public int money;
    void amethod(int i){
        //注意对比形参 i 和 money 的值的变化
        System.out.println("方法得到的 i 的值为: "+i);
        i=i*5;
        System.out.println("方法执行语句 i=i*5 后 i 的值为: "+i);
        System.out.println("money 的值为: "+this.money);
    }
    public static void main(String[ ] args)
    {
        ParamTransfer pt=new ParamTransfer();
        pt.money=100;
        //把 money 作为参数传递给方法
        pt.amethod(pt.money);
    }
}
```

在 `ParamTransfer` 中有一个 `int` 型实例变量 `money`，在执行方法的时候把它作为参数传递给方法 `amethod`。仔细看语句的执行过程，在 `amethod` 中首先展示了变量 `i` 的变化过程，最后打印出了 `money` 的值。注意语句 `this.money`，这个语句的意思是获得当前对象实例变量 `money` 的值。`this` 表示调用该方法的对象。总结 `this` 关键字的用途如下。

- 在构造函数中调用其他的构造函数。
- 在方法中获得调用该方法的对象。

回到原来讨论的话题，打印出结果会发现 `money` 的值没有随着 `i` 变化，说明 `i` 得到的不过是 `money` 的一个拷贝。程序的运行结果如下。

```
方法得到的 i 的值为: 100
方法执行语句 i=i*5 后 i 的值为: 500
money 的值为: 100
```

5.4.2 对象类型的参数传递

“Java 的参数传递都是传值传递”，这句话容易引起误解，如果是简单类型还好说，不过是传递一个原来的值的拷贝给参数就可以了，但是对于对象型的参数呢，是把对象拷贝一份传递过去还是把对象的引用直接传递过去呢？。

实际上这两种方式都不对，Java 采取的方法比较特殊，它是把对象的引用拷贝一份传递

给参数。虽然也是传值，但是需要读者仔细理解，以免误会，下面通过程序来演示对于对象类型的参数传递 Java 值怎么处理的。

```
class Time
{
    public int hour;
    public int minute;
    public int second;
}

public class ObjectParamTransfer {
    Time time;
    public static void main(String[] args) {
        ObjectParamTransfer opt=new ObjectParamTransfer();
        opt.time = new Time();
        opt.time.hour = 12;
        opt.time.minute = 45;
        opt.time.second = 20;
        System.out.println("time 的属性值: ");
        System.out.println("hour="+opt.time.hour);
        System.out.println("minute="+opt.time.minute);
        System.out.println("second="+opt.time.second);
        //将对象作为参数传递给方法 objectMethod
        opt.objectMethod(opt.time);
        //对比执行方法后的变化
        System.out.println("执行方法之后的 time 的属性值: ");
        System.out.println("hour="+opt.time.hour);
        System.out.println("minute="+opt.time.minute);
        System.out.println("second="+opt.time.second);

    }
    void objectMethod(Time t) {
        System.out.println("参数 t 的属性值: ");
        System.out.println("hour="+t.hour);
        System.out.println("minute="+t.minute);
        System.out.println("second="+t.second);
        System.out.println("对 t 和 time 进行==比较, 结果为: "+(t==this.time));
        System.out.println("对 t 和 time 进行 equals 比较, 结果为: "+(t.equals(this.time)));
        System.out.println("改变 t 的实例变量值");
        t.hour=8;
        t.minute=12;
        t.second=24;
    }
}
```

程序的运行结果如下。

```
time 的属性值:
hour=12
minute45
second20
参数 t 的属性值:
hour=12
minute=45
second=20
对 t 和 time 进行==比较, 结果为: true
对 t 和 time 进行 equals 比较, 结果为: true
改变 t 的实例变量值
```

执行方法之后的 time 的属性值:

```
hour=8
minute=12
second=24
```

程序中首先定义了一个简单的 Time 类, 有 3 个 int 型的实例变量 hour、minute、second。在 ObjectParamTransfer 类中有一个 Time 实例变量 time。首先程序初始化一个 ObjectParamTransfer 对象, 然后对其 time 进行一系列的赋值, 打印出它调用方法前的值, 把 time 作为参数传递给 objectMethod 方法。

在 objectMethod 方法中, 首先把参数 t 的一系列值打印出来, 可以发现跟 time 的值是一样的; 再将 t 和 opt 的 time 进行 “==” 和 “equals” 比较, 打印出比较结果都为 true, 可以知道 t 和 time 都指向同一个对象; 然后改变 t 的属性值, 返回到主方法中, 把 time 的属性值打印出来, 发现 time 的属性值随着 t 的改变而改变了, 验证了 t 和 time 指向同一个对象的说法。那么 t 和 time 是不是同一个引用呢? 这一点在程序中并不能得到验证, 可以再写一段代码验证。

```
public class ObjectParamTransfer2
{
    Time time1;
    Time time2;
    public static void main(String[] args)
    {
        ObjectParamTransfer2 opt = new ObjectParamTransfer2();
        //创建两个对象
        opt.time1=new Time();
        opt.time2=new Time();
        //设置对象属性值
        opt.time1.hour = 12;
        opt.time2.hour = 23;
        System.out.println("交换前的属性值: ");
        System.out.println("time1.hour=" + opt.time1.hour);
        System.out.println("time2.hour=" + opt.time2.hour);
        //调用 swap 方法
        opt.swap(opt.time1, opt.time2);
        System.out.println("交换后的属性值: ");
        System.out.println("time1.hour=" + opt.time1.hour);
        System.out.println("time2.hour=" + opt.time2.hour);
    }
    //swap 方法试图交换两个对象
    void swap(Time t1, Time t2) {
        Time temp;
        temp = t1;
        t1 = t2;
        t2 = temp;
    }
}
```

程序的运行结果如下。

```
交换前的属性值:
time1.hour=12
time2.hour=23
交换后的属性值:
```

```
time1.hour=12
time2.hour=23
```

可以看到对 Time 对象的交换操作并不成功。在操作中交换的只是 t1 和 t2，而对于原来的对象并没有改变。

实际上 Java 的对象参数的传递比较特殊，它不是传引用（很多人都说 Java 对于对象参数是传引用，这是错误的认识），而是得到对象引用的一个拷贝，把它传递给参数，所以参数跟原来的对象引用都指向同一个对象，对对象的操作都改变实际的对象。但是对参数的操作却不能影响原来的对象引用，它们之间是一个拷贝的关系。

5.5 包

在大型的项目中，可能需要上千个类甚至上万个类，如果都放在一起，是非常乱的，而且要对这上万个类都起不同的名字，显然这样是很复杂的。Java 提供了一种有效的类的组织结构，这就是包。标准的 Java 类库就是由多个包组织的，例如前面使用到的 java.util 就是其中一个。

5.5.1 包的使用

包的概念是比较抽象的，但是定义和使用包的时候却非常简单，在程序的最前面声明如下。

```
package pkg;
```

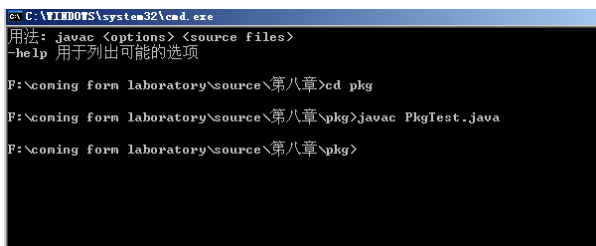
这里还是用最简单的程序演示包的使用，首先编写如下程序。

```
package pkg;
public class PkgTest
{
    public static void main(String args[]){
        System.out.println("PkgTest success");
    }
}
```

然后将其放入名为 pkg 的文件里面，在命令行中进入 pkg 文件夹执行编译命令：

```
javac PkgTest.java
```

运行结果如图 5-1 所示。



```
C:\WINDOWS\system32\cmd.exe
用法: javac <options> <source files>
-help 用于列出可能的选项

F:\coming from laboratory\source\第八章>cd pkg
F:\coming from laboratory\source\第八章\pkg>javac PkgTest.java
F:\coming from laboratory\source\第八章\pkg>
```

图 5-1 运行结果 1

然后返回 pkg 的上一级目录，执行运行编译出的字节码的命令。

```
java pkg.PkgTest
```

屏幕上输出：

```
PkgTest success
```

表示程序成功地运行，运行结果如图 5-2 所示。

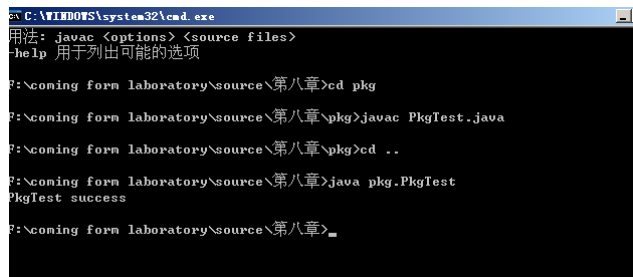


图 5-2 运行结果 2

一般情况下包的命名比上面的实例复杂很多，例如：

```
package com.lancy.qq;
```

把上面的类放入这个包中，则类的定义如下。

```

package com.lancy.qq;
public class PkgTest{
    public static void main(String args[]){
        System.out.println("PkgTest success");
    }
}
  
```

文件 PkgTest.java 的存放路径应改为“...source\第八章\com\lancy\qq”，其中 com、lancy、qq 都是文件夹。进入 qq 文件夹目录执行如下编译命令。

```
javac PkgTest.java
```

返回到 com 的上一级目录，即第八章文件夹，运行程序。

```
java com.lancy.qq.PkgTest
```

屏幕上输出：

```
PkgTest success
```

表示程序运行成功。如果程序没有成功运行，检查文件夹是否正确，程序中包的定义语句是不是完整。

5.5.2 导入包

包可以对类进行良好的管理，但是这样的话包就位于不同的文件夹下面了，不能直接在一个文件夹中调用需要的类。Java 的解决方案是导入需要的包。在前面使用 Scanner 类来获得用户输入时已经使用了导入包的语句，程序如下。

```

import java.util.*;
public class AverageTemperatures{
    public static void main(String args[] ){
        int count;
        double next,sum,average;
        sum=0;
        //创建一个 Scanner 对象，通过它可以获得用户输入
        Scanner sc=new Scanner(System.in);
        System.out.println("请输入七天的温度：");
        for(count=0;count<7;count++)
        {
            next=sc.nextDouble();
            sum+=next;
        }
        System.out.println(sum);
        average=sum/7;
        System.out.println("平均气温为： "+average);
    }
}

```

程序的第一行 `import java.util.*;`，这句话的意思是把 Java 包中的 `util` 包下的所有类都导入进来，这样就能使用 `util` 下面的类了。Java 包定义了所有的标准 Java 类，`util` 包是其中之一，而 `Scanner` 类是 `util` 包中的一个类。

导入这个包之后，就可以使用包中的 `Scanner` 类了，程序中获得了一个该类的对象，然后执行相关的操作：

```
Scanner sc=new Scanner(System.in);
```

在程序编写中，读者需要自己定义一些包，然后在其他的程序中需要使用这些包。下面演示一个简单的示例，在原来的包 `com.lancy.qq` 中添加一个类 `Output`。

```

package com.lancy.qq;
public class Output
{
    public void output(Object o){
        System.out.println(o.toString());
    }
}

```

在 `com` 目录的上一层目录中定义类 `TestImport`，如下所示。

```

import com.lancy.qq.*;
import java.util.*;
public class TestImport{
    public static void main(String args[] ){
        //创建 Output 对象
        Output out=new Output();
        //创建 Date 对象
        Date Date=new Date();
        out.output(Date);
    }
}

```

在这个类中导入自己定义的包 `com.lancy.qq`，同时导入了 `java.util` 包，并创建了一个自定义类 `Output` 类的对象 `out` 和 `java.util` 下面的 `Date` 类型的对象，通过 `out` 的 `output` 方法把该对象转换为字符串输出。程序的运行结果如下。

Sat Sep 20 16:07:31 CST 2008

当然程序会根据不同的运行时间产生不同的运行结果。

5.5.3 在 Eclipse 中使用包

包的组织是比较复杂的，尤其是当包和类的数量都很大的时候，会产生比较大的工作量。在 eclipse 下面使用包来组织就比较容易。下面就按步骤简单地讲解在 Eclipse 下包的使用方法。

(1) 新建一个工程“testPackage”，选中源代码文件夹 src，新建一个 Package，如图 5-3 所示。

(2) 单击“Next”按钮，在弹出对话框的“Name”文本框中输入包的名字，这里输入“com.lancy.qq”，如图 5-4 所示。

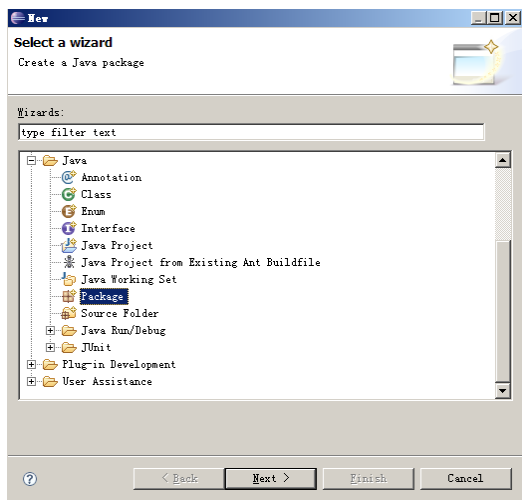


图 5-3 新建包

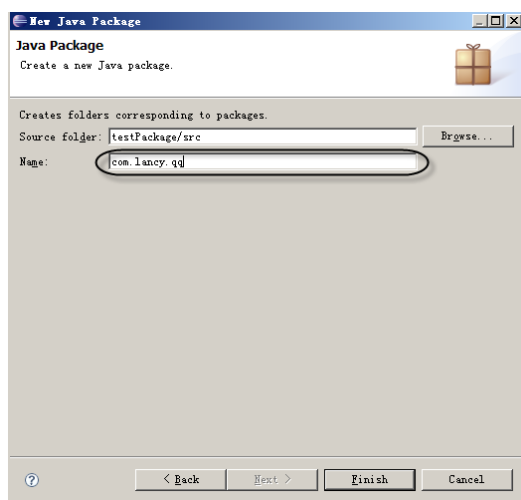


图 5-4 输入包名

(3) 单击“Finish”按钮，完成包的输入。选中新建的包，在下面新建一个类“PackageTest”，自动生成如下代码，可以看到包名已经自动加上了。

```
package com.lancy.qq;  
public class PackageTest {  
}
```

(4) workspace 中该工程下面的文件组织结构如图 5-5 所示。

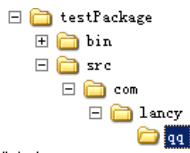


图 5-5 包的文件组织结构

可以看到，在 eclipse 下面使用包是比较简单的。

5.6 小结

本章首先对 Java 的类和对象以及对象和构造函数的知识进行了简单介绍，然后介绍了 static 关键字的使用，最后介绍了参数传递和包的使用方法。通过本章的学习，可使读者对 Java 面向对象特性有初步的了解。