# 第 3 章　类与对象


- 类和对象的概念。
- 类的编写方法和对象的创建。
- 类的成员变量的定义。
- 类的成员方法的定义。
- 包的概念。
- 访问权限。
- 内部类及泛型类。


- 理解类和对象的基本概念。
- 掌握类和对象的创建与应用。
- 理解方法和构造方法的概念。
- 掌握方法的声明和调用。
- 理解包的概念。
- 掌握类及其成员访问权限控制。
- 理解内部类的基本概念。
- 编写简单的面向对象程序。

## 3.1　引例

**例 3.1**　用面向对象的思想来设计一个简单的程序，求一个矩形的周长和面积。

分析：以面向对象的程序设计方式来思考这一问题，我们可以通过以下几步来解决：

（1）用一个合适的名字（如 Rectangle）来标识我们要分析的客观实体（矩形）。

（2）分析客观实体的共有特征：长和宽，用 length 和 width 两个属性对其进行描述。

（3）分析客观实体的共有功能，用 perimeter()和 area()两个方法来实现其功能。

（4）设计矩形类和一个测试类。

（5）在测试类中，用矩形类产生一个矩形对象，并计算其周长和面积。

程序代码如下：

```
//文件名 Jpro3_1.java
class Rectangle                              //定义类
```

```
{
    float length;                          //定义属性
    float width;                           //定义属性
    double perimeter()                     //定义方法
    {
        return 2*(length+width);
    }
    double area()                          //定义方法
    {
        return length*width;
    }
}
public class Jpro3_1                                  //定义类
{
    public static void main(String args[])           //定义程序执行的入口方法
    {
        Rectangle r1=new Rectangle();                 //创建对象
        r1.length=5.5f;                               //给对象的属性赋值
        r1.width=3.5f;
        double zc= r1.perimeter();                    //调用 perimeter()方法计算周长
        System.out.println("矩形的周长是:"+zc);              //输出周长
        System.out.println("矩形的面积是:"+r1.area());     //计算并输出面积
    }
}
```

程序运行结果：

```
矩形的周长是:18.0
矩形的面积是:19.25
```

**程序分析：**

在上述 Java 源程序 Jpro3_1.java 中，定义了两个类。第一个类为矩形类，类名为 Rectangle，用于实现对矩形实体的共同的属性和方法的封装；第二个类的类名 Jpro3_1 与源程序名相同，该类的主要功能不是用来封装实体，而是用于编写算法对前面的 Rectangle 类的功能进行测试，因此，不妨称其为测试类。在测试类中首先定义了一个程序执行的入口方法 main()，在该方法中用矩形类定义矩形对象，再利用矩形对象调用其相应的属性和方法完成其相应的功能。

为了解决这个问题，在开始的分析中，我们提到了几个概念：类、属性、对象和方法，它们究竟有什么意义，下面将一一介绍。

## 3.2 类

将客观世界中的一个特定种类的实体放在一起，并抽取它们身上共性加以描述，这就得到了软件系统中的类。因此，通常从下面三个方面来描述一个类：

① 有一个名字来唯一标识它所描述的客观实体。

② 有一组属性来描述客观实体的共有特征。

③ 有一组方法来实现客观实体的共有行为。

类是组成 Java 程序的基本要素。类封装了一类对象的属性和方法。类是用来定义对象的模板，当使用一个类创建了一个对象时，我们也说给出了这个类的一个实例。

### 3.2.1 类的声明

Java 语句中类的定义通常包含两个部分：类声明和类体。其基本格式如下：

class 类名

{

    类体的内容

}

其中，“class 类名”是类的声明部分。

class 是关键字，用来定义类。类名指的是类的名称，类名的命名与标识符的命名一致。类名的命名规则是，类名的第一个字母通常要大写，如果类名是多个单词连接而成，每个单词首字母都大写，如 BeijingChangcheng、HelloChina、ComputerArea 等。类名最好能体现类的功能或作用。

当定义一个类时，我们可以在“class 类名”前加 public、abstract 和 final 等修饰符对所定义类的特征进行限制。还可以在其后加 extends <父类名>和 implements <接口名列表>来说明类的继承性。这些内容在后面的章节中将会陆续介绍。

### 3.2.2 类体的构成

定义类的目的是为了描述一类事物共有的属性和功能，即将数据和对数据的操作封装在一起，这一过程由类体来实现。类体通常有两种类型的成员：

① 成员变量——通过变量声明定义，来描述类创建的对象的属性。

② 成员方法——通过方法的声明定义，来描述类创建的对象的功能。

封装的思想就是将数据和对数据的操作封装在一起，当一个对象执行自己的操作时，它对外界隐藏了操作的细节。如当人们看电视时，通常大部分人都不关心电视机机壳里隐藏的复杂电子元器件，也不关心这些电子器件是如何工作来产生电视画面的。电视机做了自己要做的事并对我们隐藏了它的工作过程。

如何才能做到对类的合理封装呢？这要通过合理地定义类中的成员变量和成员方法来实现。

1. 成员变量的定义

定义成员变量最简单的格式为：

类型 变量名 1[，变量名 2，…]

如果我们把变量名前的所有关键字称为该变量的修饰符，那么变量的类型修饰符是必须有的。它决定该变量在内存中分配空间的大小。成员变量可以是简单类型，如 byte、int、long、boolean、float、double；也可以是数组、字符串或类等引用类型。

每个类中的成员变量类型的定义，要根据具体情况来定，不能一概而论。如将例 3.1 中 Rectangle 类中的 length 和 width 定义为 int 型是否可以？

```
class Rectangle
{
    int  length;
    int  width;
}
```

这样的定义对于程序来说完全可以，只不过在现实世界中，矩形的长和宽并非总是整数，

所以定义为 float 类型比 int 类型更具有实际意义。

那么，例 3.1 中关于矩形长和宽的定义是否合理呢？

```
class Rectangle
{
    float length;
    float width;
}
```

从类的封装性来看，上面的定义并不理想。由于 Rectangle 类中的成员变量的访问权限修饰符为缺省情况，于是在测试类 Jpro3_1 中，我们可以对 Rectangle 类的成员变量直接进行操作。

```
pubic class Jpro3_1
{
    public static void main(String args[])
    {
        Rectangle r1=new Rectangle();
        r1.length=5.5f;
        r1.width=3.5f;
     }
}
```

这就相当于一个电视机除去了机壳，任何人都可能对其里面的器件直接进行操作，没了任何封装性，其安全性就受到了威胁。

对于软件系统来说，封装有什么作用呢？在一个包含许多对象的系统中，对象之间以各种方式相互依赖。如果其中一个对象出现了故障，软件工程师不得不修改它的时候，对其他的对象隐藏这个对象的操作意味着只需修改这个对象而不需改变其他对象。如上面的 Rectangle 类的定义中，如果将成员变量 length 的类型由 float 改为 int，那么测试类 Jpro3_1 中的语句 r1.length=5.5f;，将不能通过编译。

要解决上述问题，我们可以在定义成员变量时再加上访问权限修饰符，其格式如下：

[访问权限] 类型 变量名 1[，变量名 2，…]

成员变量常用的访问权限修饰符有 4 种：public（公共的）、protected（受保护的）、private（私有的）和 default（缺省的）。Rectangle 类中的成员变量没有使用访问权限修饰符，也就是访问权限修饰符缺省，此时，同一个包（具体见 3.8.2 节）中的其他类就能对其进行访问。要实现类的成员变量在类的外部不可见，就必须使用 private 修饰符对其进行限定。用 private 修饰的成员变量只在本类中有效，因此，可以实现数据最严密的封装。如可将例 3.1 中关于矩形长和宽的定义修改成如下形式：

```
class Rectangle
{
    private float length;
    private float width;
}
```

这样，外部类就不能直接访问其成员变量了。如将 Rectangle 类的两个成员变量定义为私有，那么测试类 Jpro3_1 中两条访问成员变量的语句将无效。如：

```
public class Jpro3_1
{
    public static void main(String args[])
    {
        Rectangle r1=new Rectangle();
```

```
        r1.length=5.5f;          //非法
        r1.width=3.5f;           //非法
    }
}
```

接下来的问题是：如果外部类无法访问其他类私有的成员变量，那么，每个类私有的成员变量又该如何赋值呢？我们有三种途径来解决这个问题：

① 在定义成员变量时赋初值。

② 在类中定义成员方法时给成员变量赋值。

③ 利用构造方法给成员变量赋初值。

2. 成员变量的初始化

成员变量定义时如果没有赋值，则其初值是它的默认值。例如，byte、short、int 和 long 类型的默认值为 0，float 类型的默认值为 0.0f，double 类型默认值为 0.0，boolean 类型的默认值为 false，char 类型默认值为“\u0000”，引用类型默认值为 null。但有时我们需要变量具有其他初值，那么可以在定义的同时给变量赋值。如在定义 Rectangle 类的成员变量时，直接给长和宽赋初值：

```
class Rectangle
{
    private float length=5.5f;
    private float width=3.5f;
}
```

注意以下的写法是错误的：

```
class Rectangle
{
    private float length;
    length=5.5f;             //非法
}
```

上述程序中的错误反映了这样一个问题：对成员变量的操作应放在方法中进行，当程序执行过程中要改变成员变量的值，应设计相应的方法，在方法体内通过相应的语句来修改成员变量的值。

3. 成员方法

在 Java 中，方法只能作为类的成员，也称为成员方法。方法操作类所定义的数据，以及提供对数据的访问的代码。大多数情况下，程序都是通过类的方法与其他类的实例进行交互的。

方法包括方法声明和方法体。创建成员方法最简单的格式为：

```
返回值类型  方法名（[参数列表]）
{
    方法体
}
```

第一行为方法声明，大括号中的是方法体。方法体可以包含一个或多个语句，每个方法执行一项任务。每个方法只有一个名称，通过使用这个名称方法才能被调用。方法名一般用小写字母表示，如例 3.1 中定义的 perimeter ()方法和 area()方法。但当方法名由多个英文单词组成时，一般第一个单词用小写，后面每个单词的首字母都大写，如 computePerimeter()、computeArea()等。

返回值类型是方法返回值的数据类型。若方法不返回任何值，则返回值类型为关键字 void。除构造方法外，所有的方法都要求有返回值类型。方法名的定义与标识符定义一致，最好能够体现方法的含义，达到“见名知义”的程度。如例 3.1 中，可以定义专门操作成员变量的方法，一般命名为 setXxx()和 getXxx()。如对 Rectangle 类进行修改，增加 4 个专门用于操作成员变量的方法：

```
class Rectangle
{
    private float length=5.5f;
    private float width=3.5f;
    void setLength(float l)
    {length=l;}
    void setWidth(float w)
    { width =w;}
    float getLength()
    {return length; }
    float getWidth()
    {return width; }
}
```

在增加的 4 个方法中，setLength(float l)和setWidth(float w)方法分别用来设置两个成员变量 length 和 width 的值。getLength()和getWidth()方法分别用来获取两个成员变量 length 和 width 的值。由于两个成员变量的访问权限被定义为 private，从 Rectangle 类的外部无法对这两个变量进行访问，程序中提供了两组 set()和 get()方法用于对两个成员变量进行读写操作。如只设置 get()方法，那么该变量对外来说是只读的；如只设置 set()方法，那么该变量对外来说是只写的。

方法的参数列表是可选的。列表中的参数称为形式参数，简称为形参。当方法被调用时，形参被数据或变量替换，这些数据或变量称为实际参数，简称为实参。这种在方法调用时用实参代替形参的形式，称为参数传递。

如果希望方法有返回值，则在方法体的最后使用 return xxx;语句，终止方法并返回一个值给该方法的调用者。

在类体中，有一些方法的设置是为了实现类的相应功能，如例 3.1 中，perimeter()方法和 area()方法分别用来计算矩形的周长和面积。这些方法往往是类对象的功能体现。

成员方法也可加访问权限修饰符，用来限定该方法的使用范围。成员方法的访问权限修饰符与成员变量相同，共有 4 种，其意义也相似。其具体内容将在 3.8 节中介绍。

4. *构造方法*

构造方法是一种特殊方法，它的名字必须与它所在类的名字完全相同，并且不返回任何数据类型。Java 程序中的每个类允许定义若干个构造方法，但这些构造方法的参数必须不同。

每个类都有一个默认的构造方法（它没有任何参数），如果类没有重新定义构造方法，则创建对象时系统自动调用默认的构造方法。否则，创建对象时调用自定义的构造方法。

在例 3.1 中的 Rectangle 类中，在定义成员变量的同时给变量赋了初值，这就意味着用此类创建的任何矩形对象，其初始的长和宽都是一样的，如果我们希望每次能得到长和宽都不一样的矩形对象，又该如何来设计呢？请看下面的程序：

```
class Rectangle
{
    private float length;
    private float width;
   public Rectangle()
   {
      length=5.5f;
      width=3.5f;
    }
   public Rectangle(float x, float y)
   {
      length=x;
      width=y;
    }
   ……

}
```

在 Rectangle 类中，增加了两个构造方法，分别用于创建类的对象。那么，在测试类 Jpro3_1 中，就能调用这两个构造方法来创建对象，程序代码如下：

```
public class Jpro3_1
{
    public static void main(String args[])
    {
        Rectangle r1=new Rectangle();
        Rectangle r2=new Rectangle();
        Rectangle r3=new Rectangle(2.4f,3.2f);
        Rectangle r4=new Rectangle(2.0f,3.5f);
        ……
    }
}
```

每次调用无参构造方法所创建的矩形对象的长和宽都有一个固定的初值，如 r1 和 r2 的长为 5.5，宽为 3.5。而每次调用带参构造方法时，只要实参不同，就能构造出不同的矩形对象。如 r3 的长为 2.4，宽为 3.2；r4 的长为 2.0，宽为 3.5。

5. main()方法

一般情况下，要使用一个类，就必须创建这个类的对象。那么，对象的创建是应该设计在同一个类中还是设计在另一个类中呢？答案是两者都可以，但最好是在另一个类中。这样，没有对象定义的纯粹的类设计部分就可以单独保存在一个文件中，就不会影响该类的重复使用。

在例 3.1 中，矩形类和测试类分别放在两个类中来进行设计，就是类重复使用思想的体现。由于 main()方法是每个 Java 应用程序执行的入口方法。因此，在 Jpro3_1 类中设计了 main()方法。main()方法的定义格式如下 ：

```
public static void main(String args[])
```

public 修饰符说明 main()方法可以为所有类访问。static 修饰符表明 main()方法是静态方法。main()方法用于启动 Java 应用程序，因为是用 static 定义的，当应用程序启动后，实际上系统中并不存在任何对象，可以直接调用。因此，main()方法的主要工作就是创建启动程序所需的对象。它的返回值类型为 void，即无实际返回值。args[]是形式参数。

## 3.3 对象

对象（object）是以类作为“模板”创建的，类是一种复杂数据类型，是对象定义的前提。类是具有共同特性的实体抽象，而对象又是现实世界中实体的表现。对象是类的实例化，对象和实例（instance）两个词语通常可以互换。当然，实例也可理解为类的具体实现。类和对象的关系是一般与个别的关系，可以比作一张图纸和多幢楼房之间的关系。

### 3.3.1 对象的创建

对象的创建过程实际上就是类的实例化过程。创建对象须使用操作符 new，其格式可以有两种：

类名　对象名=new 类名([参数 1,参数 2,…]);

如：Rectangle r1=new Rectangle(5.5f,3.5f);

或者

类名　对象名;

对象名=new 类名([参数 1，参数 2，…]);

如：Rectangle r1;

　　r1=new Rectangle(5.5f,3.5f);

第一种方式，将对象的声明和创建合并在一起，其功能是为对象分配内存空间，然后执行构造方法中的语句为成员变量赋值，最后将所分配存储空间的首地址赋给对象变量。存储空间相当于一个抽屉，数据放在抽屉中，对象变量中所放的是打开这个抽屉的钥匙。其内存模型如图 3-1（b）所示。

第二种方式，先声明对象变量，对象变量声明后，该对象变量还没有引用任何实体，我们称这时的对象为空对象，其内存模型如图 3-1（a）所示。空对象必须再用 new 运算符分配实体后才能使用，其内存模型如图 3-1（b）所示。

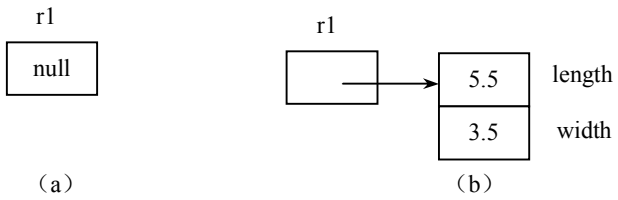


图 3-1　对象的内存模型

使用 new 运算符结果是返回新创建的对象的一个引用。new 为指定的类创建一个对象时，首先为该对象在内存中分配内存空间，然后以类为模板构造该对象，最后把该对象在内存中的首地址返回给对象名。这样我们就可以像使用一个普通变量一样通过对象名来使用对象。同时，使用 new 运算符创建对象时，也调用了该类的构造方法实现对象的初始化。

一个类使用 new 运算符可以创建多个不同的对象，这些对象被分配不同的内存空间，因此改变一个对象的内存状态不会影响其他对象的内存状态。例如，我们使用 Rectangle 类创建两个对象 r1 和 r2，其内存模型如图 3-2 所示。

Rectangle r1=new Rectangle(5.5f,3.5f);
Rectangle r2=new Rectangle(2.4f,3.2f);


图 3-2　多个对象的内存模型

### 3.3.2　对象的使用

一旦创建了对象，就可以使用对象编写程序，完成相应的功能了。对象的使用主要有以下三种情况：

1. 使用对象的成员变量和成员方法

对象不仅可以操作自己的成员变量来改变状态，而且还可以使用类中的方法，对象通过这些方法产生一定的行为。对象通过运算符“.”来引用自己的成员。如例 3.1 中，main()方法中的语句：

```
r1.length=5.5f;
r1.width=3.5f;
double zc= r1.perimeter();
System.out.println("矩形的周长是:"+zc);
System.out.println("矩形的面积是:"+r1.area());
```

在上面的程序中，当方法有返回值时，可以将返回值赋给相同类型的变量，也可以直接输出返回值。

2. 对象间的赋值

相同类型的变量可以互相赋值。如果两个对象有相同的值，那么它们就具有相同的实体，即指向同一个内存空间。如以下语句：

```
Rectangle r1=new Rectangle(5.5f,3.5f);
Rectangle r2=new Rectangle(2.4f,3.2f);
r1=r2;
```

执行赋值语句 r1=r2 后，r1 和 r2 引用的实体就一样了，即 r1 和 r2 指向同一个存储空间，r1 原先引用的存储空间失去了引用对象，变成了一块垃圾内存，其内存模型如图 3-3 所示。此时 r1 和 r2 的成员完全相同，其值也相同。

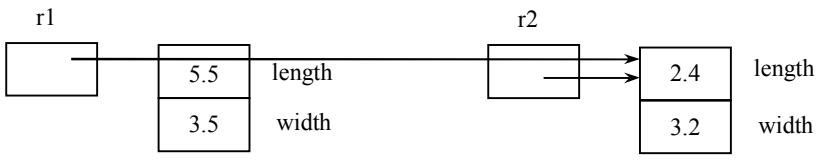


图 3-3　对象赋值后的内存模型

3. 把对象作为方法的参数

对象也可以像变量一样，作为方法的参数使用。

**例 3.2** 分别定义两个类：点类和圆类，将点类对象作为圆类的成员变量，并编写测试类进行功能测试。

```
//文件名 Jpro3_2.java
class Point
{
   int x;
   int y;
   Point(int a,int b)
   {
     x=a;
     y=b;
   }
 }
 class Circle
 {
    int radius;
    Point point;              //定义引用型成员变量
    Circle(int r,Point p)     //形参为引用型的变量
    {
      radius=r;
      point=p;                //引用型变量赋值
    }
    void output()
    {
     System.out.println("圆的半径是:"+radius);
     System.out.println("圆的圆心是:"+"("+point.x+","+point.y+")");
    }
 }
 public class Jpro3_2
 {
    public static void main(String args[])
    {
        Point p1=new Point(1,2);
        Circle c=new Circle(5,p1);  //实参为对象变量
        c.output();
    }
}
```

程序运行结果为：

```
圆的半径是:5
圆的圆心是:(1,2)
```

**程序分析：**

例 3.2 定义了一个点类 Point，一个圆类 Circle，一个测试类 Jpro3_2。Circle 类的成员变量 point 的类型为 Point，因此其构造方法中必须有这个相应的参数 Point p 来给成员变量 point 赋初值。测试类 Jpro3_2 首先要创建一个 Point 对象 p1，然后将 p1 作为构造方法 Circle(5,p1)的实参去创建 Circle 对象 c。那么实参 p1 传递给形参 p 是什么呢？是对象变量本身所存储的内容，即所引用对象的内存空间的首地址，而不是所引用的实体的内容。

### 3.3.3 垃圾对象的回收

当对象被创建时，就会在 Java 虚拟机的堆区中拥有一块内存，在 Java 虚拟机的生命周期中，Java 程序会陆续地创建多个对象，如果所有的对象都永久占有内存，那么内存有可能很快被消耗，引发内存空间不足。因此，必须采取一种措施及时回收那些无用对象占用的内存，以保证内存可以被重复利用。

Java 虚拟机提供了一个系统级的垃圾回收器线程，它负责自动回收那些无用对象所占用的内存，这种内存回收的过程被称为垃圾回收。

图 3-3 中，r1 和 r2 是 Rectangle 类的两个对象，分别指向两个实体，占用不同的内存空间，如果执行语句：

```
r1=r2;
```

则 r1 与 r2 均指向 r2 所引用的内存空间，而 r1 以前指向的内存空间将成为垃圾内存。Java 虚拟机会自动回收这个没用的对象空间。

## 3.4 成员变量

### 3.4.1 实例变量和类变量

类有两种不同类型的成员变量：实例变量和类变量。类变量又称为静态变量。用关键字 static 修饰的成员变量称为类变量，而没有用关键字 static 修饰的成员变量称为实例变量。例如下面 A 类中，x 是实例变量，而 y 是类变量。

```
class A
{
    float x;         //实例变量定义
    static int y;    //类变量定义
}
```

具体来说，实例变量和类变量的主要区别为：

① 在内存分配的空间上。不同对象的同名实例变量分配不同的内存空间，变量之间的取值互不影响；不同对象的同名类变量分配相同的内存空间，也就是说多个对象共享类变量，改变其中一个对象的类变量的值会影响其他对象中相应的类变量的值。

② 在内存分配的时间上。当类的字节码文件被加载到内存时，类变量就分配了相应的内存空间；实例变量是当类的对象创建时才会被分配内存。

③ 访问方式不同。实例变量必须用对象名访问；类变量可以用类名访问，也可以用对象名访问。

**例 3.3** 编写一个矩形类，用类变量统计所创建的矩形对象的个数。

```
//文件名 Jpro3_3.java
class Rectangle
{
    private float length;
    private float width;
    static int number=0;    //定义类变量
    Rectangle(float l, float w)
```

```
        {
            length=l;
            width=w;
            number++;        //改变类变量的值
        }
    }

    public class Jpro3_3
    {
        public static void main(String args[])
        {   //用两种方式访问类变量
            System.out.println("当前矩形对象的个数为:"+Rectangle.number);
            Rectangle r1=new Rectangle(1.0f,2.0f);
            System.out.println("当前矩形对象的个数为:"+r1.number);
            Rectangle r2=new Rectangle(2.0f,2.5f);
            System.out.println("当前矩形对象的个数为:"+r1.number);
            System.out.println("当前矩形对象的个数为:"+r2.number);
            System.out.println("当前矩形对象的个数为:"+Rectangle.number);

        }
    }
```

程序运行结果：

```
当前矩形对象的个数为:0
当前矩形对象的个数为:1
当前矩形对象的个数为:2
当前矩形对象的个数为:2
当前矩形对象的个数为:2
```

**程序分析：**

main()方法的第一条语句被执行时，系统首先将 Rectangle 类加载到内存，并为类变量 number 分配了内存空间，此时，就可以用类名 Rectangle 直接引用类变量 number；当执行了下面两句：

```
Rectangle r1=new Rectangle(1.0f,2.0f);
Rectangle r2=new Rectangle(2.0f,2.5f);
```

系统创建了矩形对象 r1 和 r2 后，才用对象 r1 和 r2 来引用类变量 number，但这种方式并不提倡使用，因为其可读性较差。从最后三条语句的输出结果也可看出，r1、r2 和 Rectangle 共享类变量 number。

### 3.4.2 常量

如果一个类的成员变量前加 final 修饰符，该成员变量就为常量，常量的名字习惯用大写字母表示，如

final double PI=3.14159;

常量不占用内存，这意味着在声明常量时，必须初始化。对象可以使用常量，但不能更改它的值。

**例 3.4** 已知圆的半径，求圆的面积。

```
//文件名 Jpro3_4.java
class Circle
```

```
{
    float radius;
    final float  PI=3.14f;  //定义常量
    Circle (float r)
    {
        radius=r;
    }
    float area()
    {
        return PI*radius*radius;
    }
}
public class Jpro3_4
{
    public static void main(String args[])
    {
        Circle t=new Circle (0.5f);
        System.out.println("圆面积是:"+t.area());
    }
}
```

程序运行结果为：

```
圆面积是:0.785
```

## 3.5 成员方法

### 3.5.1 实例方法和类方法

类有两种不同类型的成员方法：实例方法和类方法。类方法又称为静态方法。用关键字 static 修饰的成员方法称为类方法，而没有用关键字 static 修饰的成员方法称为实例方法。在例 3.3 中，如将静态变量 number 定义为私有成员，则在其测试类 Jpro3_3 中不能直接访问 number，若要在 Rectangle 类定义一个方法来访问 number，该方法必须定义为静态方法。

**例 3.5** 定义静态方法访问静态变量。

```
//文件名 Jpro3_5.java
class Rectangle
{
    private float length;
    private float width;
    private static int number;
    Rectangle(float l, float w)
     {
        length=l;
        width=w;
        number++;
     }
     public static void print()  //定义静态方法访问静态变量
     {
        System.out.println("当前矩形对象的个数为:"+number);
     }
}
```

```
public class Jpro3_5
{
    public static void main(String args[])
    {
        Rectangle.print();
        Rectangle r1=new Rectangle(1.0f,2.0f);
        Rectangle r2=new Rectangle(2.0f,2.5f);
        Rectangle.print();
    }
}
```

程序运行结果为：

```
当前矩形对象的个数为:0
当前矩形对象的个数为:2
```

具体来说，实例方法和类方法的主要区别为：

① 在内存分配的时间上。当类的字节码文件被加载到内存时，类方法就分配了相应的入口地址；实例方法是当类的对象创建时才会被分配入口地址。

② 访问方式不同。实例方法必须用对象名访问；类方法一般用类名访问，也可用对象名访问。

③ 操作的对象不同。类方法只能操作类变量，不能操作实例变量；而实例方法既可以操作类变量也可以操作实例变量。

④ 另外，实例方法中可以调用实例方法和类方法，而类方法中只能调用类方法，不能调用实例方法。

### 3.5.2 方法中的参数传递

当方法被调用时，如果方法有参数，参数变量必须有具体的值。如例 3.2 中的两个构造方法：

```
Point(int a,int b)
   {
    x=a;
    y=b;
   }
Circle(int r,Point p)
    {
     radius=r;
     point=p;
    }
```

其调用语句为：

```
Point p1=new Point(1,2);
Circle c=new Circle(5,p1);
```

方法声明中设计的参数称为形式参数或形参，如 int a、int b 和 int r、Point p。调用方法时所传递的参数称为实际参数或实参。实参可以是一个常量，如 Point(1,2)中的 1、2，也可以是一个已赋值的基本数据类型的变量，如：

```
int a=1;
int b=2;
Point p1=new Point(a,b);
```

实数还可以是一个引用类型的对象，如 Circle(5,p1)中的 p1。调用方法时，实参的值传递给形参。不管参数是何类型，传递时都是按值传递，下面分两种情况进行说明。

1. 基本数据类型参数的传值

对于基本数据类型的参数，实参数据类型的级别不能高于形参的级别。比如，不能向 int 类型的形参传递一个 float 类型的值，但可以向 double 类型形参传递一个 float 类型的值。如将例 3.2 中的语句：

```
Point p1=new Point(1,2);
```

改为

```
Point p1=new Point(1.5f,2);
```

则程序不能通过编译。

另外，改变形参变量的内容，实参的内容不会跟着变化。

**例 3.6** 当方法的类型为基本数据类型时，改变形参变量的内容，实参的内容不会跟着变化。

```
//文件名 Jpro3_6.java
class A
{
    void a(int a)
    {
        a=10;
        System.out.println("a="+a);
    }
}
public class Jpro3_6
{
    public static void main(String args[])
    {
        A a1=new A();
        int x=5;
        a1.a(x);
        System.out.println("x="+x);
    }
}
```

程序运行结果为：

```
a=10
x=5
```

**程序分析：**

实参 x 的值为 5，当对象 a1 调用方法 a()时，x 将 5 传递给形参 a，a 的值就为 5。在方法 a()中，将 a 赋值为 10，此时，x 的值并没有改变，仍为 5。

2. 引用类型参数的传值

引用类型数据包括对象、数组以及接口等。当方法的参数是引用类型时，实参传递的是对象的引用，而不是对象的内容。

当实参的值传递给形参时，实参和形参对象都指向同一个存储空间。如果改变形参所引用实体的内容，实参所引用实体的内容也会跟着变化。

**例 3.7** 引用类型参数的传值。

```
//文件名 Jpro3_7.java
```

```
class Point
{
   int x;
   int y;
   Point(int a,int b)
   {
     x=a;
     y=b;
   }
 }
 class Circle
 {
    int r;
    Point point;
    Circle(int r1,Point p1)
    {
      r=r1;
      //p1.x=10;      //改变 p1 的坐标
      //p1.y=10;
      point=p1;
    }
    void output()
    {
     point.x=10;      //改变 point 的坐标
     point.y=10;
     System.out.println("圆心的坐标是:"+"("+point.x+","+point.y+")");
    }
 }
public class Jpro3_7
 {
    public static void main(String args[])
    {
        Point p=new Point(1,2);
        System.out.println("点的坐标是:"+"("+p.x+","+p.y+")");
        Circle c=new Circle(5,p);
        c.output();
        System.out.println("点的坐标是:"+"("+p.x+","+p.y+")");
    }
 }
```

程序的运行结果为：

```
点的坐标是:(1,2)
圆心的坐标是:(10,10)
点的坐标是:(10,10)
```

**程序分析：**

首先定义了一个坐标为(1,2)的点对象 p，把 p 作为实参创建一个圆对象 c，将实参 p 的值传递给了形参 p1，不管是在构造方法中改变 p1 的坐标，还是在 output()方法中改变 point 的坐标，p 的坐标都会随之改变。

## 3.6 关键字 this

this 是 Java 中的关键字，代表本类对象，下面从两个方面介绍它的应用。

1. 使用 this 区分成员变量和局部变量

在方法体中声明的变量以及方法的参数称为局部变量，方法的参数在整个方法内有效，方法内定义的局部变量从它定义的位置之后开始有效。成员变量在整个类内有效。

在一个类中，如果出现局部变量的名字与成员变量的名字相同，则成员变量被隐藏，即这个成员变量在这个方法内暂时失效。例如：

```
class Rectangle
{
    private double length;
    private double width;
    Rectangle(double length,double width)
     {
        length= length; //成员变量被隐藏
        width=width;
     }
}
```

如果希望成员变量 length 被成功赋值，必须在成员变量前加 this，this.length 表示当前对象的成员变量 length，而不是局部变量 length。例如：

```
class Rectangle
{
    private double length;
    private double width;
    Rectangle(double length,double width)
     {
        this.length= length;    //给成员变量 length 赋值
        this.width=width;       //给成员变量 width 赋值
     }
}
```

2. 用 this 调用本类中的其他构造方法

在构造方法中用 this 调用本类中的其他构造方法，调用时要放在构造方法的首行。如：

```
class Rectangle
{
    private float length;
    private float width;
   public Rectangle()
   {
      this(5.5f,3.5f);  //调用了带参构造方法
    }
   public Rectangle(float x, float y)
   {
      length=x;
      width=y;
    }
 }
```

3. 实例方法中可以使用 this，类方法中不可使用 this

实例方法中使用 this 来引用的成员表示为当前对象的成员，通常情况省略不写，其含义相同。类方法中不可使用 this，因为类方法可以通过类名直接调用，这时可能还没有创建任何对象。例如：

```
class R
{
    int x;
    void f()
    {
        this.x=100;     //this 可以省略
        this.g();       //this 可以省略
    }
    void g()
    {
        x=200;
    }
 }
```

## 3.7 内部类

类可以嵌套定义，即在一个类的类体中可以嵌套定义另外一个类。被嵌套的类称为内部类，它的上级称为外部类。内部类中还可以再嵌套另一个类，在最外层的类被称为顶层类。内部类的创建方法与外部类相似。

除外部类外，其他类无法访问内部类。当一个类只在某个类中使用，并且不允许除外部类外的其他类访问时，可考虑把该类设计成内部类。

**例 3.8** 内部类的使用。

```
//文件名 Outer.java
class Outer
{
    private int index=100;                              //外部类成员
    class Inner
    {
        private int index=50;                           //内部类成员
        void print()
        {
            int index=30;                               //内部类局部变量
            System.out.println(index);                  //输出内部类局部变量
            System.out.println(this.index);             //输出内部类成员变量
            System.out.println(Outer.this.index);       //输出外部类成员变量
        }
    }
    void print()
    {
        Inner inner=new Inner();                        //创建内部类对象
        inner.print();                                  //调用内部类方法
    }
    Inner getInner()
    {
        return new Inner();                             //创建匿名内部类对象
```

```
    }
    public static void main(String args[])
    {
        Outer outer=new Outer();                    //创建外部类对象
        outer.print();
        Inner inner=outer.getInner();               //创建内部类对象
        inner.print();
    }
}
```

程序运行结果为：

```
30
50
100
30
50
100
```

**程序分析：**

在外部类 Outer 中定义了一个内部类 Inner，外部类对象 outer 调用的 print()方法是自己的方法。要调用内部类的 print()方法，必须先构造内部类对象。内部类调用 getInner()方法，该方法用匿名的方式创建了一个内部类对象。

若将 main()方法放在另一个类 Jpro3_8 中，那么，创建内部类对象的方式将会改变，基本格式为：

外部类.内部类 内部类对象=外部类对象.new 内部类();

程序如下：

```
//文件名 Jpro3_8.java
……
public class Jpro3_8
{
    public static void main(String args[])
    {
        Outer outer=new Outer();
        outer.print();
        Outer.Inner inner=outer.new Inner(); //创建内部类对象
        inner.print();
    }
}
```

还有一种类称为匿名内部类，是指可以利用内部类创建无名对象，并利用它访问类里的成员。匿名内部类的创建不同于普通的内部类的创建，不需要定义类名，直接用 new 创建对象。如上例中的 getInner()方法中的语句：

```
Inner getInner()
{
    return new Inner();          //创建匿名内部类对象
}
```

匿名内部类的应用主要是简化程序代码。在 Java 的窗口程序设计中，常会利用匿名内部类的技术编写“事件”的程序代码，具体应用请参见第 11 章。

## 3.8 自定义包

包（package）是 Java 提供的类的组织方式。一个包对应一个文件夹，一个包中可以放置许多类文件和子包。

Java 语言可以把类文件存放在不同层次的包中，其目的是在设计软件系统时，当系统中的类较多，就可以分类存放不同的类文件，从而大大方便软件的维护和资源的重用。Java 语言规定：同一个包中的文件名必须唯一，不同包中的文件名可以相同。包的组织方式和表现方式与 Windows 中的文件和文件夹完全相同。

JDK 中提供许多系统包，只要正确安装了 JDK 文件，在 Java 环境下就可以使用系统包中的文件，关于系统包中的相关内容将在第 7 章中介绍。

### 3.8.1 创建包

定义包语句的格式为：

package <包名>;

其中，package 是包的关键字，<包名>是包的标识符。package 语句指出该语句所在的 Java 源文件中的所有类编译后所存放的位置。

Java 文件规定，如果一个 Java 源程序中有 package 语句，那么 package 语句必须写在 Java 源程序的第一行。如：

```
package mypackage;
public class a
{
    ……
}
```

如果源程序中省略了 package 语句，那么源文件中的类经编译后放在与源程序相同的一个无名包中。

一个包中还可以定义子包，可由标识符加“.”分割而成，如：

```
package china.anhui.hefei;
package sun.com.cn;
```

如果在 china.anhui.hefei 包中存放一个名叫 Student 的类，则该类的全名应为：china.anhui.hefei.Student。

### 3.8.2 使用包

包中存放的是编译后的字节码文件。用户可以在编程时，通过 import 语句导入包中的类，从而直接使用导入的类。

import 语句的使用分两种情况：

① 导入某个包中的所有类，如：

import mypackage.*;

② 导入某个包中的一个类，如：

import mypackage.Student;

**注意：**导入的类是要占用内存空间的，当某包中的类很多，而用到的类也很多时，就用

方式①导入；当某包中的类很多，而要用的类却很少时，就用方式②导入。当用方式①导入类时，如包中还有子包，则子包中的类不会被导入。

例 **3.9** 修改例 3.2，将 Point 类放到 point 包中，并用文件名 Point.java 保存；将 Circle 类放到 circle 包中，并用文件名 Circle.java 保存；将 Jpro3_9 类用文件名 Jpro3_9.java 保存。修改后的程序如下：

```
//Point.java
package point;        //定义包
public class Point
{
   public int x;
   public int y;
   public Point(int a,int b)
   {
     x=a;
     y=b;
   }
 }
```

Point.java 编译后，产生文件夹 point，在 point 文件夹中生成类文件 Point.class。

```
//Circle.java
package circle;             //定义包
import point.Point;         //导入 Point 类
public class Circle
 {
    int r;
    Point p;
    public Circle(int r1,Point p1)
    {
      r=r1;
      p=p1;
    }
    public void output()
    {
     System.out.println("圆的半径是:"+r);
     System.out.println("圆的圆心是:"+"("+p.x+","+p.y+")");
    }
 }
```

Circle.java 编译后，产生文件夹 circle，在 circle 文件夹中生成类文件 Circle.class。

```
// Jpro3_9.java
import point.Point;         //导入 Point 类
import circle.Circle;       //导入 Circle 类
public class Jpro3_9
 {
    public static void main(String args[])
    {
        Point p=new Point(1,2);
        Circle c=new Circle(5,p);
        c.output();
    }
 }
```

**程序分析：**

程序的输出同例 3.2，但由于三个类不在同一个包中，被导入的类的访问权限必须是

public，被导入的类的成员要想被访问到，其访问权限也要随之改变。

**注意**：在实际操作时，要将 Point.java、Circle.java、Jpro3_9.java 放在同一个文件夹下。

## 3.9 访问权限

当类存放到不同的包中时，对类及其成员的访问将受到其访问权限的限制。Java 中的访问权限修饰符有：private（私有的）、protected（受保护的）、public（公有的）和缺省的（不加任何访问修饰符）。

### 3.9.1 类与构造方法的访问权限

1. public 类和友好类

对类的访问权限的控制只有两种：一种是加 public 修饰符。例如：

```
public class A
{
   ……
}
```

用 public 修饰符修饰的类称为公共类，公共类可以被任何包中的类访问。

另一种是不加任何访问权限修饰符，例如：

```
class A
{
   ……
}
```

这样的类被称为友好类。如果在另一个类中使用友好类，一定要保证它们在同一个包中。

2. 构造方法的访问权限

类中默认构造方法的访问权限和类的访问权限保持一致。当用户自定义构造方法时，也要保证其访问权限与类相同。因此，构造方法一般只用 public 和缺省两种权限修饰符。当 public 类的构造方法的访问权限缺省时，在不同包的类中，是不能用此构造方法来创建对象的。

### 3.9.2 成员变量和成员方法的访问权限

1. 私有的变量和方法

用关键字 private 修饰的成员变量和成员方法被称为私有变量和私有方法。如：

```
class A
{
   private int x;
   private void printX()
     {
         ……
      }
   ……
}
```

私有的成员只在本类中有效，只有在本类中创建该类的对象时，这个对象才能访问自己的私有成员。从类的封装性来说，成员变量大多定义为私有，而成员方法往往是类的对外访问接口，定义为私有就失去了意义。

2. 公有的变量和方法

用关键字 public 修饰的成员变量和成员方法被称为公有变量和公有方法。如：

```
public class A
{
   public int x;
   public void printX()
   {
        ……
    }
   ……
}
```

公有的变量和方法通常定义在公共类中，不管是否处于同一个包，公共类对象能访问自己的公有的变量和方法。

3. 友好的变量和方法

不使用任何访问权限修饰符修饰的成员变量和成员方法被称为友好变量和友好方法。如：

```
class A
{
   int x;
   void printX()
     {
        ……
      }
   ……
}
class B
{
    void g()
     {
        A a=new A();
        a.x=10;
        a.printX();
     }
}
```

友好成员通常定义在友好类中，友好成员的有效范围是同包中的类。若类 A 与类 B 定义在同一个源文件中，那么编译后，类 A 与类 B 为同一个包中的类。此时，在 B 类中用 A 类创建的对象可直接引用其友好的成员。

4. 受保护的变量和方法

用关键字 protected 修饰的成员变量和成员方法被称为受保护的变量和受保护的方法。如：

```
public class A
{
   protected int x;
   protected void printX()
     {
        ……
      }
   ……
}
```

受保护的成员通常用在父类与子类之间，体现了继承的概念。对于同一个包中的类，受保护的成员的用法与友好成员相同，对于不同包中的类，只有子类对象才能访问受保护的成员。

其具体用法将在第 4 章中介绍。

## 3.10 泛型类

泛型（Generics）是 JDK 1.5 的新特性，泛型的本质是参数化类型，也就是说所操作的数据类型被指定为一个参数。这种参数类型可以用在类、接口和方法的创建中，分别称为泛型类、泛型接口、泛型方法。Java 语言引入泛型的好处是安全简单。

在 JDK1.5 之前，在没有泛型的情况下，是通过对类型 Object 的引用来实现参数的“任意化”。“任意化”带来的缺点是要做显式的强制类型转换，而这种转换是要求开发者对实际参数类型可以预知的情况下进行的。对于强制类型转换错误的情况，编译器可能不提示错误，在运行的时候才出现异常，这是一个安全隐患。

1. 泛型类声明

声明泛型类的一般格式为：

class 类名<泛型列表>

如：

class A<E,F>

其中，A 是泛型类的名称，E、F 是泛型类的参数，也就是说，泛型类的参数类型没有指定。它可以是任何引用类型，但不能是基本数据类型。泛型类声明时，其参数类型可能作为类成员变量和成员方法的类型。

2. 泛型类的使用

**例 3.10** 计算锥体的体积。

分析：要计算锥体的体积，必须知道锥体底的形状。假设锥体的底是圆、正方形和矩形等形状，其底面的面积可以计算出来。于是设计一个计算锥体体积的泛型类，其参数为底的形状。

```
//文件名 Jpro3_10.java
class Cone<E>       //定义泛型类
{
   double height;
   E  bottom;       //定义泛型变量
   public Cone(E e)
   {
      bottom=e;
   }
   public void volume()
   {
      String s=bottom.toString();
      double area=Double.parseDouble(s);
      System.out.println("体积是:"+1.0/3.0*area*height);
   }
}
class Circle
{
   double radius;
   Circle(double r)
   {
```

```
        radius=r;
    }
    public String toString()  //覆盖 toString 方法
    {
        return ""+radius*radius*Math.PI;
    }
}
class Square
{
    double side;
    Square(double s)
    {
        side=s;
    }
    public String toString()  //覆盖 toString 方法
    {
        return ""+side*side;
    }
}
public class Jpro3_10
{
    public static void main(String args[])
    {
        Circle circle=new Circle(10);
        Cone<Circle> coneOne=new Cone<Circle>(circle);  //创建一个(圆)锥对象
        coneOne.height=30;
        coneOne.volume();
        Square square=new Square(10);
        Cone<Square> coneTwo=new Cone<Square>(square); //创建一个(方)锥对象
        coneTwo.height=10;
        coneTwo.volume();
    }
}
```

程序运行结果为：

```
体积是：3141.59
体积是：333.33
```

**程序分析：**

每个类都是 Object 类的子类，Object 类中有一个 toString()方法，因此，所有的类都有一个从 Object 类继承的 toString()方法，本例正是运用了 toString()方法返回锥底的面积。

## 3.11 实例

**例 3.11** 打印某个日期，并判断该年是否是闰年。

分析：

日期有年、月、日三个基本属性，因此所设计的日期类应有三个私有的成员变量，要构造日期对象，带参的构造方法是必不可少的。针对私有的成员变量，应设计一组 get 和 set 方法来存取私有的成员变量的值。另外，输出一个日期的方法和判断某年是否是闰年的成员方法也是必需的。

```
//文件名 Jpro3_11.java
class Date
{
    private int year;                          //成员变量，表示年
    private int month;                         //成员变量，表示月
    private int day;                           //成员变量，表示日
    public Date(int y, int m, int d)           //构造方法
    {
       year = y;
       month = m;
       day = d;
    }
    public void setDate(int y, int m, int d)    //设置日期值
    {
        year = y;
        month = m;
        day = d;
    }
    public int getYear()
    {
    return year;
    }
    public int getMonth()
    {
    return month;
    }
    public int getDay()
    {
    return day;
    }
    public void Print()                        //输出日期值
    {
       System.out.println("date is "+year+'-'+month+'-'+day);
    }

    public boolean isLeapYear()                //判断是否闰年
    {
       return (year%400==0) | (year%100!=0) & (year%4==0);
    }
}
public class Jpro3_11
{
    public static void main(String args[])
    {
        Date a = new Date(2010,10,1);          //创建对象
        a.Print();
        if(a.isLeapYear())
            System.out.println(a.getYear()+"是闰年");
        else
            System.out.println(a.getYear()+"不是闰年");
    }
}
```

程序运行结果为：

```
date is 2010-10-1
2010 不是闰年
```

**程序分析：**

程序中所创建的日期对象，其成员变量都有一个初值。当对象创建后，再要改变其成员变量的值，可用 setDate(int y, int m, int d)方法进行修改。getYear()、getMonth()、getDay()三个方法可以获取三个成员的值。

**例 3.12** 设计一个名为 Fan 的类来模拟风扇，属性为 speed、on、radius 和 color。假设风扇有 3 种固定的速度，用常数 1、2、3 表示慢、中、快速。写一个用户程序，程序中创建一个 Fan 对象，具有最大速度、半径为 10、黄色、打开状态这 4 个属性值。要求返回包含类中所有属性值的字符串。

分析：这个问题已经给了类名和属性名，我们需要设计的是如何获得和修改 4 个属性值，以及如何将 4 个属性转换为字符串。

```
//文件名 Jpro3_12.java
class Fan
{
  public static int SLOW=1;
  public static int MEDIUM=2;
  public static int FAST=3;
  private int speed;
  private boolean on;
  private double radius;
  private String color;
  public Fan()
    {
    speed=SLOW;
    on=false;
    radius=5;
    color="white";
    }
    public int getSpeed()
    {
    return speed;
    }
    public void setSpeed(int newSpeed)
    {
      speed=newSpeed;
    }
    public boolean isOn()
    {
    return on;
    }
    public void setOn(boolean trueOrFalse)
     {
    on=trueOrFalse;
     }
    public double getRadius()
    {
      return radius;
```

```
    }
    public void setRadius(double newRadius)
   {
    radius=newRadius;
    }
  public String getColor()
  {
  return color;
    }
  public void setColor(String newColor)
  {
    color=newColor;
    }
  public String toString(int sp)
  {
     String s="";
     switch (sp)
     {
        case 1:s="SLOW";
        case 2:s="MEDIUM";
        case 3:s="FAST";
        }
      return s;
     }
}
public class Jpro3_12
{
    public static void main(String[] args)
    {
        Fan ff=new Fan();
        String s1,s2,s3,s4,s5,s6;
        ff.setSpeed(3);
        s1=ff.toString(ff.getSpeed());
        ff.setRadius(10.0);
        s2=String.valueOf(ff.getRadius());
        ff.setColor("yellow");
        s3=String.valueOf(ff.getColor());
        ff.setOn(true);
        s4=String.valueOf(ff.isOn());
        System.out.println("The fan speed is:"+s1+"  The fan radius is:"+s2);
        System.out.println("The fan color is:"+s3+"  The fan status is:"+s4);
    }
}
```

程序运行结果为：

```
The fan speed is:FAST  The fan radius is:10.0
The fan color is:yellow  The fan status is:true
```

**程序分析：**

类 Fan 中有一个构造方法，用来初始化 4 个成员变量。类中的其他方法：setSpeed()、setRaidus()、setOn()和 setColor()分别是修改风扇的速度、半径、当前状态和颜色；getSpeed()、getRaidus()、isOn()和 getColor()分别是获得风扇的当前速度、半径、状态和颜色。String 类中

的方法 valueOf()可以将不同的数据类型转换为字符串，因此这 4 个成员变量的值可以通过方法 valueOf()转换成字符串。我们希望速度也能以定义的常量标识符表示，因此专门设计了方法 toString()用于实现速度的转换。


1．一个可以独立运行的 Java 应用程序，_____。

A．可以有一个或多个 main()方法　　B．最多只能有两个 main()方法

C．可以没有 main()方法　　D．只能有一个 main()方法

2．以下不属于构造方法特征的是_____。

A．构造方法名与其类名相同　　B．构造方法有返回值类型

C．构造方法在创建对象时自动执行　　D．每一个类可以有多个构造方法

3．下列哪个关键字可用来定义 Java 常量_____。

A．public　　B．static　　C．final　　D．void

4．this 关键字的含义是_____。

A．本类　　B．本类对象　　C．这个类　　D．父类对象

5．下列关于 Java 变量的描述，错误的是_____。

A．在 Java 程序中要使用变量，必须先对其进行声明

B．类变量可以使用对象名进行调用

C．变量不可以在其作用域之外使用

D．成员变量必须写在成员方法之前

6．类 A 有 3 个 int 型成员变量 a、b、c，则_____是类 A 的正确构造方法。

A．void A(){a=0; b=0; c=0; }

B．public void A(){ a=0; b=0; c=0;}

C．public int A (int x, int y, int z){ a=x; b=y; c=z; }

D．public A(int x,int y, int z) { a=x; b=y; c=z; }

7．指出下列程序中的非法语句。

```
class A
{
    int a;
    float b;
    a=12;
    b=12.56f;
    void f()
    {
    }
}
```

8．假设 A 类定义如下，则 main()方法中哪些语句是非法的？

```
class A
 {
   int x;
   private int y;
```

```
    static String s;
    void methodA1()
   {
   }
    static void methodA2()
   {
    }
}
public class Test
{
  public static void main(String args[])
  {
    A a=new A();
    System.out.println(a.x);
    System.out.println(a.y);
    System.out.println(a.s);
    a.methodA1();
    a.methodA2();
    System.out.println(A.x);
    System.out.println(A.y);
    System.out.println(A.s);
    A.methodA1();
    A.methodA2();
  }
}
```

9．下列程序的输出结果是什么？

```
class A
{
  void f(int x,double y)
   {
      x=x+1;
      y=y+1;
      System.out.printf("参数 x 和 y 的值分别是:%d,%f\n",x,y);
   }
}
public class Test
{
   public static void main(String args[])
   {
     int x=10;
     double y=12.58;
     A a=new A();
     a.f(x,y);
     System.out.printf("main 方法中 x 和 y 的值仍然分别是:%d,%f\n",x,y);
   }
}
```

10．写一个名为 Rectangle 的类表示矩形。其成员变量有宽 width、高 height、颜色 color，width 和 height 是 double 类型，color 是 String 类型。假定所有矩形颜色相同，用一个类变量表示颜色。要求提供构造方法和计算矩形面积的 computeArea()方法。

类的框架如下，只给出方法名，方法体自行设计。

```
class Rectangle
{
  private double  width=1;
  private double  height=1;
```

```
    private static String color=”white”;
    public Rectangle(){ }
    public Rectangle(double width,double height,String color){ }
    public double getWidth(){ }
    public void setWidth(double width){ }
    public double getHeight(){ }
    public void setHeight(double height){ }
    public static String getColor(){ }
    public static void setColor(String color){ }
    public double computeArea(){ }
}
```

写一个用户程序测试 Rectangle 类。在用户程序中，创建两个 Rectangle 对象。对两个对象设置任意的宽和高。设第一个对象为红色，第二个为黄色。显示两个对象的属性并求面积。

11．设计一个日期类及其测试类（要求：把日期类放在 MyPackage 包中，测试类和日期类不在同一包中）。

12．定义一个图书类，其成员变量包括：书号、书名、作者，单价，出版社；定义带参构造方法初始化成员变量，定义成员方法实现借书和还书功能；另外，定义一个成员变量用以记录一本书被借的次数。