第5章

类与对象

CHAPTER 5

本章学习目标

- 理解面向对象编程思想
- 掌握 Java 中创建类和对象的方法
- 掌握构造方法的概念
- 掌握 Java 的方法重载
- 掌握包的创建和使用方法
- 掌握 Java 访问修饰符的使用
- 掌握静态变量、静态方法的使用


视频讲解

5.1　面向对象思想

一般来说，计算机的基本功能就是数据处理，即编写程序，把现实世界中的事物表示为数据，然后对数据进行某种操作，得到人们需要的结果。

早期编程是非结构化编程，当程序执行到某一行，根据条件需要分支或循环操作时，就跳转（goto 行号）到相应的程序行继续执行。当程序中需要多次跳转时，程序的流程图中就会有很多的跳转线，像“一碗面条”，非常难以阅读，并且很容易出错。

结构化程序设计在 20 世纪 60 年代开始发展，并很快成为程序设计的主流。结构化程序由一些简单、有层次的程序流程架构所组成，可分为顺序、选择和循环结构。结构化编程放弃了 goto 语句，可以避免写出面条式代码，改善程序的明晰性、品质以及开发时间。但是，结构化编程在可维护性、扩展性和复用性方面存在一定的不足，而面向对象的思想可以很好地解决这些问题，逐渐成为程序设计的主流模式。

面向对象编程（Object Oriented Programming，OOP）就是以对象为单位进行编程设计。对象即现实世界中的事物，是可以明确标识的一个实体。例如，一个学生、一张桌子、一个圆、一个按钮甚至一笔贷款都可以看作一个对象。每个对象都有自己的特征和行为。对象的特征也称为状态或属性，即数据。对象的行为是指对数据的操作，Java 中称为方法。面向对象技术更容易将自然事物的逻辑转换为编程语言，有利于提高程序设计效率和准确性。

面向对象技术的关键性观念是它将数据及对数据的操作行为放在一起，作为一个相互依存、不可分割的整体——对象。对于相似的对象进行分类、抽象后，得出共同的特征而形成了类。一个类的实现实例被称作一个“对象”或者“实例”。一个类可以有多个实现对象，即类是一个范围，而对象则是类的一个具体的实体。

面向对象编程的方法过程：首先定义类，然后将类作为数据类型创建具体的对象，用“对象.变量”和“对象.方法”调用对象的数据和方法，获得结果。程序的执行表现为一组对象之间的交互通信。对象之间通过公共接口进行通信，从而完成系统功能。

面向对象编程的优点很多，随着学习的深入会慢慢了解和理解。现在先用一个简单的例子介绍一下。假设要对 10 个人的身体质量指数（BMI）进行比较，需要定义这 10 个人的姓名、身高、体重和 BMI 的变量，共 40 个。采用面向对象编程，定义一个表示人的类 Person，其中定义 4 个变量 name、height、weight、bmi，用 Person 作为类型定义 10 个对象，用“对象.变量”表示一个人的数据，相当于对数据进行了分组，非常容易区分，逻辑清晰，编程不容易出错，而变量个数的减少也使代码更简洁。

面向对象主要有三大特征：封装、继承和多态。

1. 封装

封装就是把对象的属性（状态）和方法（行为）结合在一起，并尽可能隐蔽对象的内部细节，成为一个不可分割的独立单位（即对象），对外形成一个边界，只保留有限的对外接口使之与外部发生联系。封装的原则是使对象以外的部分不能随意存取对象的内部数据，从而有效地避免了外部错误对它的“交叉感染”。数据隐藏特性提升了系统安全性，使软件错误

能够局部化，减少查错和排错的难度。

2. 继承

继承是软件重用的一种形式，它利用现有的类来构建新类，新类继承现有类的属性和方法，并可以增加新功能或修改现有类的功能。例如，有一个类 Person，定义了人的一般特性，现在需要定义一个新类表示学生，学生是一种特殊的人，既有人的特征，又有自己的特性，所以学生类 Student 可以通过继承 Person 类来定义，然后增加自己的特征。被继承的类称为父类或超类，派生的新类称为子类，子类对象拥有父类的属性和方法。继承有利于复用程序、共享代码，提高程序的可维护性、可扩展性和编程效率。

3. 多态

多态是指同一个实体同时具有多种实现形式，同一操作作用于不同的实现形式，可以有不同的解释，产生不同的执行结果。简单地说，多态就是调用相同的方法，得到不同的结果。Java 中可以通过子类对父类方法的重写实现多态，也可以利用在同一个类中重载方法实现多态。多态的引入提高了程序的抽象性、简洁性、灵活性和可替换性，降低了耦合性，提高了类模块的封闭性，有利于写出通用的代码，做出通用的编程，以适应需求的不断变化。


视频讲解

5.2 类的定义

类和对象是面向对象编程的核心和本质，是面向对象编程语言的基础。多个对象所共有的属性和行为组合成一个单元，称为类。类是具有相同属性和行为的一组对象的抽象。

类由属性和方法构成。对象的特征在类中表示为数据成员（成员变量），称为类的属性。对象的行为定义为类的方法，指定以何种方式操作对象的数据，是操作的实际实现。调用对象的一个方法就是要求对象完成一个动作。

Java 中类声明格式如下。

```
[访问符] [修饰符] class <类名>{
    [属性]
    [方法]
}
```

其中的相关概念解释如下。

访问符：用于声明类、属性或方法的访问权限，具体可取 public（公共）、protected（受保护）、private（私有）或省略。

修饰符：用于说明所定义的类有关方面的特性，可用的有 abstract（抽象）、static（静态）或 final（最终）等。

class：是 Java 语言中定义类的关键字。

类名：类名是一个字符串，必须符合标识符命名规则，习惯上以大写字母开头。

类头：访问符、修饰符、class 和类名部分，合起来称为类声明或类头。

类体：大括号及其内部的属性、方法等，称为类体。类体中可以没有任何语句，只有一对大括号，此时称类体为空。

属性：表现为数据变量，又称为成员变量或数据域(data field，或数据字段)。成员变量在类的内部是全局变量，可以被类中的任何方法调用(也有一些限制，如静态方法不能调用实例变量，后面会介绍)。属性的声明格式如下。

```
[访问符] [修饰符] <数据类型> 变量名;
```

或者

```
[访问符] [修饰符] <数据类型> 变量名 = 值;
```

例如：

```
private int age;
String name = "Zhang";
public static final PI = 3.14;
```

方法：也称为函数，是类的行为的体现，用于指定以何种方法操作对象及数据，是为完成一个操作而组合在一起的语句块，可以传入参数，返回结果。类的方法包括声明和实现两个部分，其语法格式如下。

```
[访问符] [修饰符] <返回类型> 方法名 ([参数列表]) {
    //方法体
}
```

方法定义中，大括号前的内容为方法的声明部分，也称为方法头。大括号及其中的语句是方法的实现部分，称为方法体。方法的返回类型是该方法运行后返回值的数据类型，如果没有返回值，则返回类型为 void。参数列表为该方法运行所需要的特定类型的参数。方法中声明的变量和方法的参数变量只在方法中起作用，是局部变量。如果局部变量与类的成员变量同名，则类的成员变量被隐藏。

注意，除了赋值可以和成员变量声明合并的语句，类的内部没有其他任何具体的语句(赋值、输出、分支、循环等)，具体语句必须放在方法中。

类一旦定义，就可以作为一种数据类型来声明变量和创建对象。类是对象的模板，而对象就是类的一个具体实例。创建对象的过程称为实例化。对象和实例两个概念经常互换使用，创建一个对象也称创建一个实例，可以从一个类创建多个实例。

图 5.1 显示名为 Circle 的类和它的三个对象。

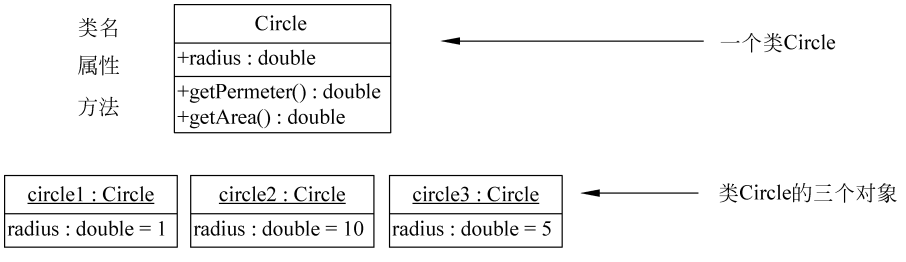

图 5.1　类及其对象

图 5.1 所示的表示方法称为 UML 类图，简称类图。类的类图有 3 个格，分别指明类名、属性和方法。对象的类图有两个格，分别指明对象名和属性值。类图中，在变量或方法名前面用“+”或“−”表示可见性为 public 或 private，数据类型放在变量或方法名的后面，用“:”隔开。UML 类图可以方便地表示程序和类的框架，是一种优秀的建模工具。

【例 5.1】　定义一个类表示圆形，有属性半径，以及求周长和面积的方法。

程序 5.1　Circle.java

```
public class Circle {
    double radius;
    public double getPerimeter() {
        return 2 * Math.PI * radius;
    }
    public double getArea() {
        return Math.PI * radius * radius;
    }
}
```

上述代码实现了对圆形类的定义，类名为 Circle，圆的属性半径定义为一个变量 radius。求周长的方法定义为 getPerimeter()，返回 double 类型的数据 2 * Math.PI * radius。求面积的方法定义为 getArea()，返回 double 类型的数据 Math.PI * radius * radius。其中，Math.PI 是一个常量，在 java.lang.Math 类中定义。

可以看到，Circle 类中没有定义 main()方法，因此是不能运行的，它只是对圆对象的定义。通常将含有 main()方法的类称为主类。面向对象编程的一般步骤是：定义一些表示对象的普通类和一个主类，在主类的 main()方法中，用普通类作为数据类型声明并创建对象，然后调用对象的属性和方法，得到想要的结果。


视频讲解

5.3　对象的创建

当创建完一个类时，就创建了一种新的数据类型。此时可以通过使用 new 关键字来声明该种类型的对象，为对象动态分配内存空间，并返回对它的一个引用，且将该内存初始化为默认值。

获得一个类的对象一般经过以下两步。

第一步：声明该类类型的一个变量，即定义一个该类的对象，给对象命名。

第二步：创建该对象(即在内存中为该对象分配地址空间)，并把该对象的引用赋给声明好的变量。这是通过使用 new 运算符实现的。

以 Circle 类为例，可以使用下面的语句创建一个 Circle 类的对象。

```
Circle circle;
circle = new Circle();
```

或者将两个步骤合并在一个语句中。

```
Circle circle = new Circle();
```

在 Java 中，所有的类对象都是动态分配内存空间。以创建 Circle 对象为例，在内存中动态创建的对象如图 5.2 所示。

执行声明语句“Circle circle”后，就定义了 Circle 的一个变量 circle，其值为空，即图 5.2 中 circle 后的方格(为变量 circle 分配的内存)中是空的。执行创建对象的语句“new Circle()”后，就创建了 Circle 的一个对象，该对象被分配了内存，内存的引用地址码为“0x1b1c6”，内存中对该对象的 radius 属性赋了初值 0。执行“=”赋值语句，将对象的引用“0x1b1c6”赋值

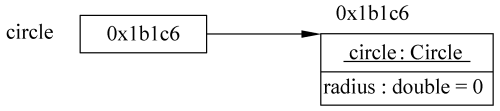

图 5.2　创建 Circle 对象

给变量 circle，circle 后的方格中存放了对象的引用“0x1b1c6”。所以，用一个类作为数据类型定义的变量，它的值是该类的对象内存的引用。

声明对象后，如果不想给对象分配内存空间，可以使用 null 关键字给对象赋值。null 关键字表示“空”，用于标识一个不确定的对象，即没有分配内存空间的对象。

null 可以赋值给引用型变量，不能赋值给基本数据类型变量。null 虽然代表一个不确定的空对象，但它本身不是一个对象，也不是类的实例。可以用等号“==”来判断一个引用型变量是否为 null。

null 的另一个用途是释放内存。Java 中没有释放内存的语句，但是提供了垃圾自动回收机制，可以自动回收不被任何变量引用的对象所占用的内存。当一个非 null 的引用型变量指向的对象不再被使用时，将 null 赋值给这个引用型变量，就断开了它对对象的引用，如果没有其他的变量引用该对象，该对象就成为内存垃圾，JVM 垃圾自动回收机制会将其占用的内存回收释放。

一个类的对象创建后，可以用“对象.变量”和“对象.方法”访问该对象的变量和方法，传递参数，获取方法运行的结果。

【例 5.2】　创建圆形类 Circle 的对象，为属性半径赋值，输出周长和面积。

程序 5.2　TestCircle.java

```
public class TestCircle {
    public static void main(String[] args) {
        //创建 Circle 对象 circle1
        Circle circle1 = new Circle();
        System.out.println("半径 = " + circle1.radius);
        //为对象的属性赋值
        circle1.radius = 10;
        System.out.println("半径 = " + circle1.radius);
        //调用对象的方法
        System.out.println("周长 = " + circle1.getPerimeter());
        System.out.println("面积 = " + circle1.getArea());
    }
}
```

运行结果如图 5.3 所示。

图 5.3　创建和使用 Circle 对象

上述代码定义了一个主类 TestCircle，在 main()方法中，声明了 Circle 类的对象

circle1，并用 new 关键字创建了对象，然后使用对象 circle 的 radius 属性，设置半径为 10，调用 radius 属性、getPerimeter()和 getArea()方法获得圆的半径、周长和面积并显示。

注意，类的成员变量有默认初始值，整数类型的自动赋值为 0，带小数点的自动赋值为 0.0，boolean 类型自动赋值为 false，其他各种引用类型自动赋值为 null。例如，在程序 5.2 中，在未使用赋值语句 circle1.radius=10；为半径赋值之前，输出的半径为 0.0。


视频讲解

5.4 构造方法

在使用 new 关键字创建对象时，new 关键字之后是“类名()”，这是调用了类的构造方法。构造方法是与类名相同的方法，用于创建类的对象并初始化属性值。构造方法是一种特殊的方法，它的名字必须和类名完全相同，并且没有返回值类型，即使 void 类型也没有。定义构造方法的格式如下。

```
[访问符] <类名称> ([参数列表]){
    //初始化语句;
}
```

在例 5.1 对 Circle 类的定义中，并没有构造方法，在这种情况下，编译器会在类中隐式地自动定义一个没有参数的方法体为空的构造方法，称为默认的构造方法。该方法不存在于源程序中，但可以使用，因此不会影响创建对象时对构造方法的调用。

构造方法常用来在创建对象时初始化属性值，以增加代码的简洁性。例如，可以在 Circle 类中定义一个带参数的构造方法 Circle(double r)，在调用这个构造方法时传入一个值 r 为属性 radius 赋值，代码如下。

```
public class Circle {
    private double radius;
    public Circle(double r){
        radius = r;
    }
    //省略
}
```

这样，如果使用 circle1=new Circle(10) 语句，就可以在创建对象的同时为对象 circle1 的半径属性 radius 赋值为 10。

注意，如果类中显式地定义了任何构造方法，Java 就不为该类提供默认(无参数)的构造方法了。这时就只能调用类中显式定义的构造方法来创建对象。因此，如果在类中定义了构造方法，一般也显式地定义一个无参的构造方法，以方便使用并避免出现一些问题(例如，基于类的封装性，类的调用者可能不知道类中没有无参的构造方法而进行了调用，就会出现编译错误)。


视频讲解

5.5 方法重载

在 Java 程序中，方法名和参数列表一起构成方法签名。如果同一个类中存在两个方法具有相同的签名，将无法编译通过。但只要保证方法签名不同，在同一个类中是允许多个方

法重名的，这种特性称为重载(Overload)。对于重载的方法，JVM 会在调用时根据签名进行动态绑定。

因此，在同一个类中，多个方法具有相同的名字，但方法签名不同，即参数的个数、类型或顺序不同，称为方法的重载。

注意，方法的返回类型不是方法签名的一部分，因此进行方法重载时，不能将返回类型不同当成两种方法的区别。参数的名称也不是签名的一部分，对于同一个参数，定义为“int a”和“int b”，对方法签名来说没有区别。

方法重载是同一个类中多态性的表现，经常用来完成功能相似的操作。

【例 5.3】　创建一个名为 MyMath 的类，其中定义加法运算。

程序 5.3　MyMath.java

```
public class MyMath {
    public int add(int a, int b) {
        return a + b;
    }
    public float add(float a, float b) {
        return a + b;
    }
    public double add(double a, double b) {
        return a + b;
    }
    public static void main(String[] args) {
        //定义一个 MyMath 的对象
        MyMath mymath = new MyMath();
        //求两个 int 型数的和，并输出
        System.out.println("3 + 5 = " + mymath.add(3, 5));
        //求两个 float 型数的和，并输出
        System.out.println("3.5 + 5.0 = " + mymath.add(3.5F, 5.0F));
        //求两个 double 型数的和，并输出
        System.out.println("2.71828 + 5.2 = " + mymath.add(2.71828, 5.2));
    }
}
```

运行结果如图 5.4 所示。

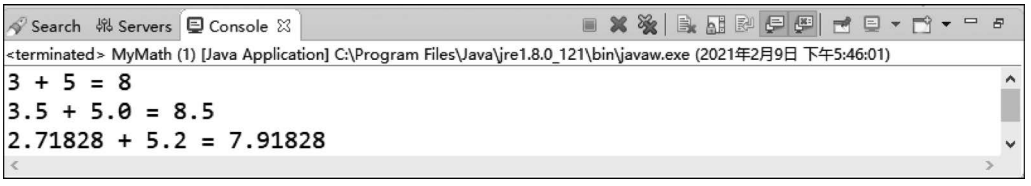

图 5.4　方法重载

上述代码中定义的三个方法名称都为 add，但是三个方法的参数是不一样的，分别传入整型、浮点型和双精度型数据时，JVM 通过参数匹配找到了对应的方法，执行后返回了正确的结果，这就是方法重载的应用。

除了普通方法，构造方法也可以重载。也就是说，在类中可以定义多个构造方法，只要保证方法签名不同。例如，可以在 Circle 类中同时定义一个没有参数的构造方法 Circle() 和一个有参数的构造方法 Circle(double r)，就形成构造方法的重载。每个构造方法都可以

用来创建对象，丰富了创建对象的方法，提高了编程灵活性。

需要注意的是，如果重载方法定义不当，可能会在调用一个方法时有多个可能的匹配，但是编译器无法判断哪个是更精确的匹配。这种情况称为歧义调用。歧义调用会产生一个编译错误。例如：

```
public class MyMathAmbiguous {
    public double add(double a, int b) {
        return a + b;
    }
    public double add(int a, double b) {
        return a + b;
    }
    public static void main(String[] args) {
        MyMathAmbiguous m = new MyMathAmbiguous();
        System.out.println(m.add(2, 1));
    }
}
```

方法 add(double a, int b)和 add(int a, double b)都可以与 add(2, 1)匹配，但是不能确定哪个更精确，所以这个调用是有歧义的，程序编译时，会在输出语句 System.out.println(m.add(2, 1))处提示“The method double add(double, int) is ambiguous”，表示 add()方法的调用不明确。


视频讲解

5.6 参数传递

定义在方法参数列表中的变量称为方法的形式参数，简称形参。调用方法时，给参数传递的数据值，称为实际参数，简称实参。实参必须与方法签名中的形参数量相同且对应的数据类型兼容，这称为参数匹配。类型兼容是指不需要经过显式的类型转换，实参的值就可以传递给形参，例如，将 int 类型的实参值传递给 double 型的形参。

在 Java 中，给方法传递参数的方式有两种：按值传递和按引用传递。

5.6.1 按值传递参数

按值传递是将要传递的参数(实参)的“值”传递给被调方法的参数(形参)，被调方法通过创建一份新的内存拷贝来存储传递的值，然后在内存拷贝上进行数值操作。也就是说，实参和形参在内存中占用不同的空间，当实参的值传递给形参后，两者之间互不影响。所以按值传递不会改变原始参数的值。

在 Java 中，当传递基本数据类型的参数给方法时，是按值传递的。

【例 5.4】 创建一个名为 PassValue 的类，演示按值传递参数。

程序 5.4 PassValue.java

```
public class PassValue {
    public static void main(String[] args) {
        int num = 5;
        System.out.println("调用 change 方法前 : num = " + num);
        //创建一个 PassValue 类型的对象
```

```
        PassValue passValue = new PassValue();
        //调用 change()方法,num 作为实参
        passValue.change(num);
        System.out.println("调用 change 方法后 : num =" + num);
    }
    //声明 change()方法,num 作为形参
    public void change(int num) {
        num += 5;
        System.out.println("change 方法中 : num = " + num);
    }
}
```

运行结果如图 5.5 所示。

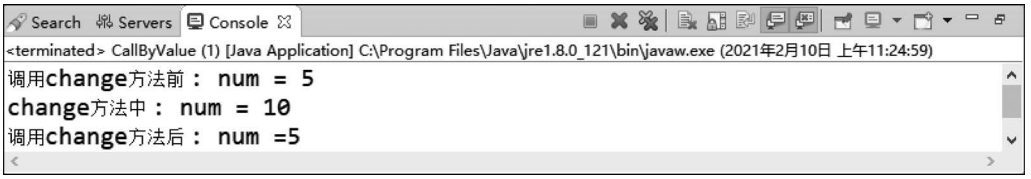

图 5.5　按值传递

从运行结果可以看出,虽然在 change(int num)方法中将实参 num 值加 5 后输出为 10,但实参的变量 num 在调用 change(int num)方法前后没有变化,都是 5。说明在这个程序中,实参和形参分别占用内存空间,互不影响,是按值传递的。

按值传递有时可能给编程带来困惑,例如下面的程序,定义了一个 swap()方法,希望把两个变量的值交换,结果却未能如愿。

【例 5.5】　创建一个名为 PassValue2 的类,定义一个 swap()方法,尝试按值传递交换变量。

程序 5.5　PassValue2.java

```
public class PassValue2 {
    public static void main(String[] args) {
        int num1 = 5, num2 = 10;
        System.out.println("调用 swap 方法前 : num1 = " + num1 + "  num2 = " + num2);
        //创建一个 PassValue2 类型的对象
        PassValue2 passValue2  = new PassValue2();
        //调用 swap()方法,num1、num2 作为实参
        passValue2.swap(num1,num2);
        System.out.println("调用 swap 方法后 : num1 = " + num1 + "  num2 = " + num2);
    }
    public void swap(int num1, int num2){       //交换 num1、num2 的值
        int temp;
        temp = num1; num1 = num2; num2 = temp;
        System.out.println("swap 方法中 : num1 = " + num1 + "  num2 = " + num2);
    }
}
```

运行结果如图 5.6 所示。

解决这个问题的一种方法是将数据放在数组中,因为数组作为参数是按引用传递的。

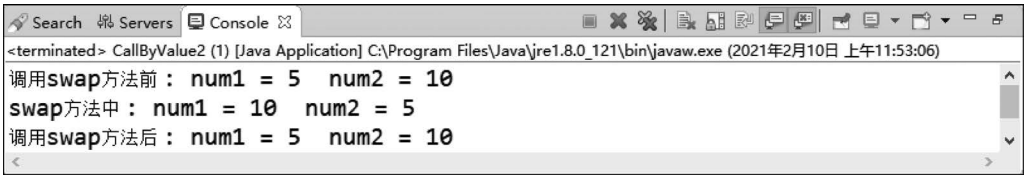

图 5.6　按值传递交换变量

5.6.2　按引用传递参数

按引用传递，是将参数的引用(类似于 C 语言的内存指针)传递给被调方法，被调方法通过传递的引用值获取其指向的内存空间，即实参和形参指向的是同一内存空间。这样，在方法中对形参进行修改操作，就是在实参的内存空间直接进行操作，将导致实参内存空间状态的改变。Java 中，对象作为参数一般是按引用传递。

【例 5.6】　创建一个名为 PassRef 的类，定义一个 changeRadius()方法，按指定倍数放大圆的半径。

程序 5.6　PassRef.java

```
public class PassRef {
    //按指定倍数放大圆的半径
    static void changeRadius(Circle c, int times) {
        c.radius = c.radius * times;
        System.out.println("在 changeRadius 方法中:radius = " + c.radius);
        System.out.println(c);
    }
    public static void main(String[] args) {
        Circle circle = new Circle();
        circle.radius = 1;
        System.out.println("调用 changeRadius 方法前:radius = " + circle.radius);
        System.out.println(circle);
        //circle 对象作为实参,按引用传递
        changeRadius(circle,10);
        System.out.println("调用 changeRadius 方法后:radius = " + circle.radius);
        System.out.println(circle);
    }
}
```

运行结果如图 5.7 所示。

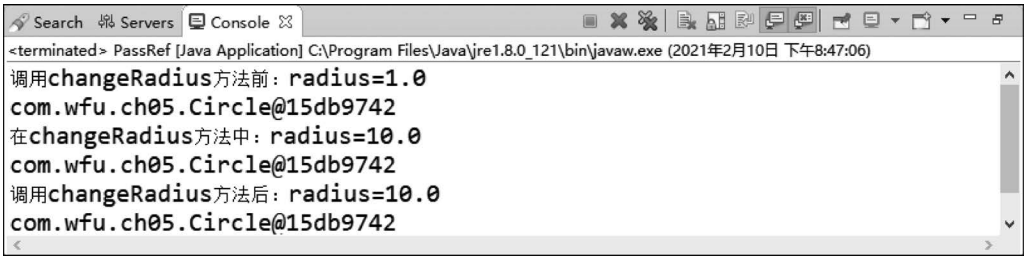

图 5.7　按引用传递参数

程序中将 Circle 对象传递给方法 changeRadius()，在方法中将半径放大了 10 倍，调用

方法后，Circle 对象的半径也发生同样的改变。程序中同时输出了 Circle 对象的字符串，其中@符号后边是对象的引用，可以看到在调用方法前后及方法中都是一样的。这是因为使用了按引用传递，形参和实参指向了相同的内存空间。按引用传递的内存模型如图 5.8 所示。

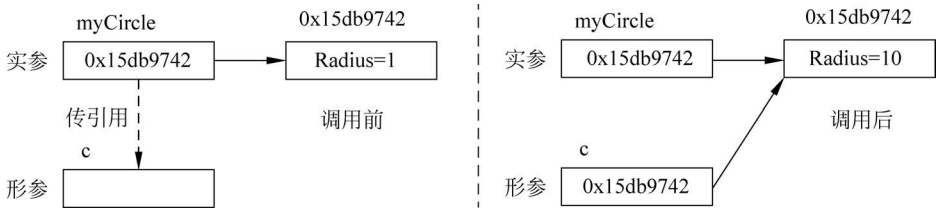

图 5.8　按引用传递的内存模型

注意，基本数据类型的封装类的对象作为方法参数时，是按值传递的，将例 5.4 程序中的 int 类型全部改为 Integer 类型进行验证，可以看到其运行结果与原来是一样的。

5.7　this 关键字

视频讲解

this 关键字代表当前所在类的对象，即本类对象。当方法的参数或者方法中的局部变量与类的属性同名时，类的属性变量就被屏蔽，此时访问类的属性需要使用“this.属性名”的方式，以区别于局部变量。

this 关键字也可以表示一个构造方法，用于在构造方法内部调用同一个类的其他构造方法。

【例 5.7】　定义一个表示矩形的类，有属性宽和高，有求周长和面积的方法。

程序 5.7.1　Rectangle.java

```
public class Rectangle {
    private double width;
    private double height;
    public Rectangle() {
        this(4,5);
    }
    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }
    public double getWidth() {
        return width;
    }
    public void setWidth(double width) {
        this.width = width;
    }
    public double getHeight() {
        return height;
    }
    public void setHeight(double height) {
        this.height = height;
    }
```

```
    public double getPerimeter() {
        return 2 * (width + height);
    }
    public double getArea() {
        return width * height;
    }
}
```

程序 5.7.2　TestRectangle.java

```
public class TestRectangle {
    public static void main(String[] args) {
        Rectangle rect = new Rectangle();
        System.out.println("周长 = " + rect.getPerimeter());
        System.out.println("面积 = " + rect.getArea());
    }
}
```

运行结果如图 5.9 所示。

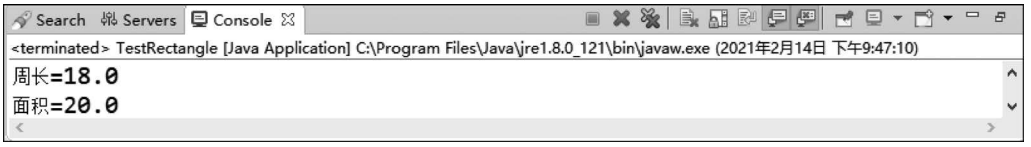

图 5.9　this 关键字的使用

在 Rectangle 类中，定义了一个带参数的构造方法 Rectangle(double width, double height)，用于给属性赋值。在这个构造方法中，两个参数的名称与类的两个属性名对应相同，使用变量 width 和 height 时，代表的是传入的参数变量，对应的属性变量被隐藏。这时，必须使用关键字 this 代表当前类的对象，用 this.width 和 this.height 表示类的属性变量，否则就不能对属性变量进行正确的赋值。

在无参的构造方法 Rectangle()中，用 this(4,5)调用了另一个构造方法为成员变量 width 和 height 赋值，这是 this 关键字的另一种用法。

关键技术：

(1) 在类的构造方法中调用另一个构造方法，只能使用 this 关键字，不能直接使用构造方法名调用。例如，将 this(4,5)改为 Rectangle(4,5)就会发生编译错误。

(2) 使用 this 关键字调用构造方法，必须放在另一个构造方法的第一行，不能在构造方法中调用自己，也不能在普通方法中调用构造方法。这是因为构造方法是用来创建对象的，要保证先创建对象，然后才能进行其他操作。

(3) 习惯上，将类的属性定义为私有变量，用 private 修饰，然后提供公开的获取和设置该变量值的方法，称为 setter/getter(修改器/访问器)方法。在上述 Rectangle 类的定义中，成员变量 width 和 height 都定义为 private 的，提供了公开的修改和访问的方法 setRadius(double radius)和 getRadius()。使用修改器和访问器有利于对属性的封装，控制对属性的访问。


视频讲解

5.8　static 关键字

Java 语言中提供了一个 static 关键字，可以用来修饰类的成员，包括成员变量、常用方

法以及代码块等。用 static 修饰的类成员具有一些特殊的属性，下面分别进行介绍。

5.8.1　静态变量

通过前面的学习我们知道，类是对象的模板，对象是类的实例。类只负责描述某类事物的特征和行为，并没有产生具体的数据。只有使用 new 关键字创建对象后，系统才会为对象分配内存空间，存储数据。一个类可以创建多个对象，不同的对象分配不同的内存空间，数据依附于对象而存在，对某个对象的属性值操作，不影响其他对象的数据。

程序员有时希望某些特定的数据在内存中只有一份，并且能够被一个类的全部实例所共享。例如，某个学校的学生共享同一个学校名称，就可以在学生类中定义一个由 static 关键字修饰的变量来表示学校名称，让学生类的所有对象来共享。

在一个 Java 类中，用 static 关键字来修饰的成员变量称为静态变量。静态变量在内存中只有一份，被该类所有的实例共享，可以使用“类名.变量名”的形式访问。静态变量与类的实例无关，只与类相关，所以又称为类变量。

【例 5.8】　定义一个表示学生的类 Student，用静态属性 schoolName 演示变量的共享。

程序 5.8.1　Student.java

```
public class Student {
    static String schoolName = "志远学校";
    String name;
    public Student(String name) {
        this.name = name;
    }
}
```

程序 5.8.2　TestStudent.java

```
public class TestStudent {
    public static void main(String[] args) {
        Student stu1 = new Student("张帅");
        Student stu2 = new Student("李丽");
        System.out.println(Student.schoolName);
        System.out.println(stu1.name + ":" + stu1.schoolName);
        System.out.println(stu2.name + ":" + stu2.schoolName);
        stu1.schoolName = "强国学校";
        System.out.println(Student.schoolName);
        System.out.println(stu1.name + ":" + stu1.schoolName);
        System.out.println(stu2.name + ":" + stu2.schoolName);
    }
}
```

运行结果如图 5.10 所示。

Student 类中定义了一个静态变量 schoolName，用于表示学生所在的学校，它被所有的实例共享。由于 schoolName 是静态变量，因此可以直接用 Student.schoolName 来访问，也可以使用 Student 的对象进行调用。将 stu1 的 schoolName 修改为“强国学校”后，两个学生对象 stu1 和 stu2 的 schoolName 都变为“强国学校”，用类名调用的 schoolName 也是一样。

注意，static 关键字只能修饰成员变量，不能修饰局部变量，否则会出现编译错误。

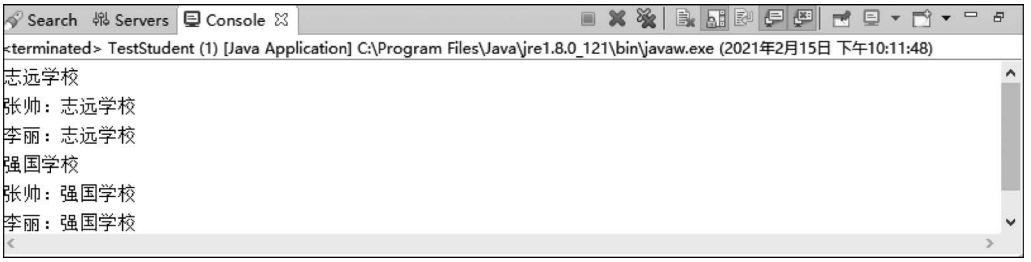

图 5.10　静态变量

5.8.2　静态方法

被 static 关键字修饰的方法称为静态方法，可以使用“类名.方法名”的形式来调用，也可以使用“对象名.方法名”的形式来访问。静态方法在不创建对象的情况下就可以被调用。在实际开发中，程序员可以将一些经常用的方法定义为静态方法放在一个公共类中，以方便使用。

Java API 中提供了一个 java.lang.Math 类，其中定义了一些静态方法来实现数学上常用的算法。例如，对 5 开平方，可以调用 Math.sqrt(5)；求两个数 a 和 b 的较大值，可以调用 Math.max(a,b)。不过，Math 类中没有求最大公约数和最小公倍数的方法，下面编写一个程序来实现它。

【例 5.9】　定义一个数学类 MathCommon，用静态方法求最大公约数和最小公倍数。

程序 5.9.1　MathCommon.java

```
public class MathCommon {
    public static int gcd(int m, int n) {
        if (m * n <= 0) return 0;
        int temp;
        if (n > m) {
            temp = n;
            n = m;
            m = temp;
        }
        while (m % n > 0) {
            temp = n;
            n = m % n;
            m = temp;
        }
        return n;
    }

    public static int lcm(int m, int n) {
        return (m * n) / gcd(m,n);
    }
}
```

程序 5.9.2　TestMathCommon.java

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

```
        int a = 8;
        int b = 6;
        System.out.println(a + "和" + b + "的最大公约数:" + MathCommon.gcd(8, 6));
        System.out.println(a + "和" + b + "的最小公倍数:" + MathCommon.lcm(8, 6));
    }
}
```

运行结果如图 5.11 所示。

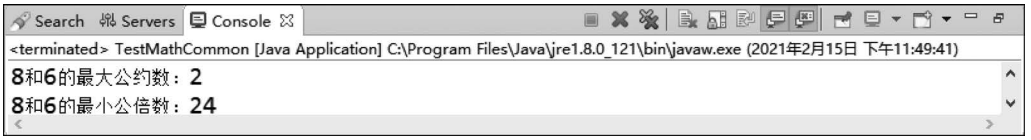

图 5.11　静态方法

MathCommon 类中定义了求最大公约数和最小公倍数的静态方法，可以用“类名.方法名”在任何类中调用，而不需要创建 MathCommon 类的实例对象。

关键技术：

(1) 静态方法可以在不创建对象的情况下访问。

(2) Java 推荐使用“类名.方法名”和“类名.静态变量”访问静态方法和静态变量。这样容易识别类中的静态方法和变量。使用对象调用静态方法和静态变量也可以，但在编译时会提示警告信息。

(3) 未用 static 修饰的方法和成员变量，必须创建对象后才能使用，分别称为实例方法和实例变量。

(4) 实例方法可以访问实例方法和实例变量，也可以访问静态方法和静态变量。静态方法只能访问静态方法和静态变量，不能访问实例方法和实例变量，因为静态方法和静态变量不属于某个特定的对象。

5.8.3　静态代码块

在 Java 类中，使用一对大括号包围起来的若干代码行称为一个代码块，用 static 关键字修饰的代码块被称为静态代码块。当类被加载时，静态代码块被执行，由于类只加载一次，因此静态代码块只执行一次。在程序中，通常用静态代码块对类的成员变量进行初始化。

【例 5.10】　在 Person 类和测试类中分别定义静态代码块，测试执行情况。

程序 5.10.1　Person.java

```
public class Person {
    String name;
    static {
        System.out.println("执行 Person 类中的静态代码块");
    }
}
```

程序 5.10.2　TestStaticBlock.java

```
public class TestStaticBlock {
    static {
```

```
        System.out.println("执行测试类中的静态代码块");
    }

    public static void main(String[] args) {
        System.out.println("执行 main 方法");
        Person p1 = new Person();
        Person p2 = new Person();
    }
}
```

运行结果如图 5.12 所示。

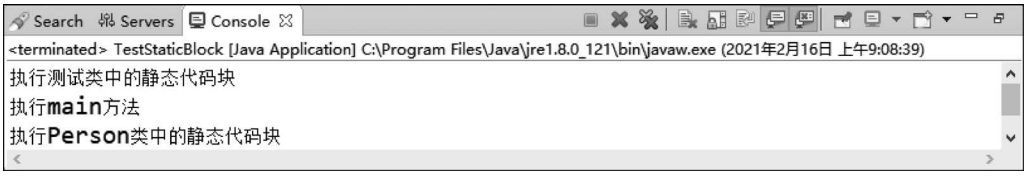

图 5.12　静态块测试

可以看到，Person 类和测试类中的两段代码块都执行了。运行程序时，Java 虚拟机会先加载主程序 TestStaticBlock，在加载类的同时就会执行该类中的静态代码块，输出“执行测试类中的静态代码块”，然后调用 main()方法，输出“执行 main 方法”，接着在 main()方法中创建了两个 Person 对象。创建第一个 Person 对象时，虚拟机加载了 Person 类，类中的静态代码块被执行，输出“执行 Person 类中的静态代码块”。因为类只加载一次，所以静态代码块的内容只输出了一次。


视频讲解

5.9　类的组织

在大型项目中，往往会有很多人参与开发，会定义很多的类，难免会有同名的类，而同一个目录下不允许有同名的文件，在同一个目录下有很多文件也不易查找。因此，Java 中引入了“包”的概念，利用包(package)可以将用途不同的类存储在不同的目录中。不同的包中，类名可以相同，使用“包名.类名”调用一个类，可以在同一段代码中使用多个同名的类创建对象而互不影响。

1. 声明包

使用 package 关键字可以指定类所属的包，语法格式如下。

```
package 包名;
```

下面是一个简单例子。

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

这里声明了一个包，名称为 mypackage。Java 用文件系统目录来存储包，任何声明了“package mypackage”的类，编译后形成的字节码文件(.class)都被存储在一个 mypackage

目录中。

定义包需要注意以下几点。

- package 语句必须放在类的源文件的第一行。package 语句的前面只能有注释或空行。
- 在一个 Java 源文件中，最多只能有一条 package 语句。
- 如果源文件中同时定义了多个类，则它们都在同一个包中。
- 多个 Java 源文件可以声明相同的包。
- 包对应文件系统目录，包的名字有层次关系，各层之间以“.”分隔，每个层次对应一个文件目录，例如 package com.wfu.ch05。

注意，虽然包的表现形式是目录，但二者并不相同。如果仅将一个类的文件放在某个目录下，而不在源文件中使用 package 声明包，在程序中是不能按“包名.类名”引用类的。

2. 使用包

在同一个包中的类，相互访问时可以不指定包名，直接引用即可。一个类如果要访问另一个包中的类，可以用“包名.类名”来引用，即直接在类名前添加完整的包名。例如：

```
java.util.Scanner sc = new java.util.Scanner();
java.util.Date now = new java.util.Date();
```

使用“包名.类名”的方法创建多个对象时，书写比较复杂，因此，Java 提供了 import 语句，可以导入其他包中的类，然后就可以直接使用类名称来声明和创建对象了。import 语句的语法格式如下。

```
import 包名.类名;
```

或者

```
import 包名.*;
```

例如：

```
import java.util.*;
import mypackage.Student;
```

使用“import 包名.类名”的方式，可以导入一个具体的类；使用“import 包名.*”的方式，可以导入整个包中所有的类。

当程序中导入两个或多个包中有同名的类时，如果使用不限定包名的类，编译器将无法区分，会产生编译错误。此时，就必须使用完全限定包名的形式。例如，类中使用了下列导入语句。

```
import java.util.*;
import java.sql.*;
```

如果类中有“Date now = new Date();”，就会产生编译错误，因为两个包中都有 Date 类，无法确定使用哪一个，在这种情况下，就必须使用完全限定包名的形式，例如：

```
java.util.Date now = new java.util.Date();
java.sql.Date sqlNow = new java.sql.Date();
```

关键技术：

（1）指明导入包中的所有的类，不能使用类似于“java. *”的语句来导入以 java 为前缀的所有包中的类。

（2）如果一个包中还有下级包，如“com. wfu. ch05”，那么，使用导入语句“import com. wfu. *;”只能导入直接在上级包 com. wfu 中的类，不能导入 com. wfu 的下级包中的任何类。

（3）一个包中只能有一条 package 语句，但可以有多条 import 语句。


视频讲解

5.10　访问修饰符

为了将类中的数据有效地保护起来，Java 提供了访问修饰符来控制属性、方法和类的访问，以隐藏类的实现细节，防止对数据进行未经授权的访问，这种方式称为封装。

引入封装，使用者只能通过事先制定好的方法访问数据，可以方便地加入控制逻辑，限制对属性的不合理操作，有利于保证数据的完整性和安全性。实现封装的关键是限制外界直接与对象交互，而要通过指定的方法操作对象的属性。

Java 中定义了 private（私有的）、protected（受保护的）、public（公开的）访问修饰符，同时定义了一个默认的（friendly，友好的）服务级别，用于声明类、属性和方法的访问权限。明确访问修饰符的限制是用好“封装”的关键。

- 使用 public 访问修饰符，类的成员可被同一包或不同包中的所有类访问，也就是说，public 访问修饰符可以使类的特性公用于任何类。
- 使用 protected 访问修饰符允许类本身、同一包中的所有类和不同包中的子类访问。
- 如果一个类或类的成员前没有任何访问修饰符时，它们获得默认的（友好的）访问权限，默认的可以被同一包中的其他类访问。
- private 访问修饰符是限制性最大的一种访问修饰符，被声明为 private 的成员只能被此类中的其他成员访问，不能在类外看到。

Java 中访问修饰符的用法如表 5.1 所示。

表 5.1　访问修饰符

访问控制	private 成员	默认成员	protected 成员	public 成员
同一类中成员	√	√	√	√
同一包中其他类	×	√	√	√
不同包中子类	×	×	√	√
不同包中非子类	×	×	×	√

【例 5.11】　在包 p1 中定义 public 类 MyClass1，其中定义一些用不同的访问符修饰的变量和方法，定义一个无修饰符的类 MyClass2，其中定义一个 public 方法。分别在包 p1、p2、p3 中定义 Test 类，测试访问符的限制范围。

程序 5.11.1　MyClass1. java

```
package p1;
public class MyClass1 {
    public int a = 5;
    private int b = 10;
```

```
    protected int c = 20;
    int d = 30;
    public void func1() {
        System.out.println("func1");
    }
    private void func2() {
        System.out.println("func2");
        System.out.println(b);
    }
    protected void func3() {
        System.out.println("func3");
    }
    void func4() {
        System.out.println("func4");
    }
}
```

程序 5.11.2　MyClass2.java

```
package p1;
class MyClass2 {//同一包中被访问
    public void func1() {
        System.out.println("func1 of MyClass2");
    }
}
```

程序 5.11.3　Test.java

```
package p1;
public class Test {
    public void func() {
        MyClass1 obj1 = new MyClass1();
        //公共属性,任何地方都可以访问
        System.out.println(obj1.a);
        //Error,b 为私有属性,类外无法访问
        //System.out.println(obj1.b);
        //c 是受保护属性,同包的类可以访问
        System.out.println(obj1.c);
        //d 是默认属性,同包的类可以访问
        System.out.println(obj1.d);
        //func1()是公共方法,任何地方都可以访问
        obj1.func1();
        //Error,func2()为私有方法,类外无法访问
        //obj1.func2();
        //func3()是受保护方法,同一包中的类可以访问,其他包中的子类也可以访问
        obj1.func3();
        //func4()是默认方法,同一包中的类可以访问
        obj1.func4();
        //同一包中的默认类可以访问
        MyClass2 obj2 = new MyClass2();
    }
    public static void main(String[] args){
        Test t = new Test();
        t.func() ;
    }
}
```

程序 5.11.4 Test.java

```
package p2;
import p1.MyClass1;
//Error,不能导入不同包中的默认类
//import p1.MyClass2;
public class Test {
    public void func() {
        MyClass1 obj1 = new MyClass1();
        //公共属性,任何地方都可以访问
        System.out.println(obj1.a);
        // Error,b 为私有属性,类外无法访问
        //System.out.println(obj1.b);
        // Error,c 是受保护属性,不同包中的非子类无法访问
        //System.out.println(obj1.c);
        // Error,d 是默认属性,不同包中的类不能访问
        //System.out.println(obj1.d);
        // func1()是公共方法,任何地方都可以访问
        obj1.func1();
        // Error,func2()为私有方法,类外无法访问
        //obj1.func2();
        // Error,func3()是受保护方法,不同包中的非子类无法访问
        //obj1.func3();
        // Error,func4()是默认方法,不同包中的类不能访问
        //obj1.func4();
        // Error,不可以访问不同包中的默认类
        //MyClass2 obj2 = new MyClass2();
    }
}
```

程序 5.11.5 Test.java

```
package p3;
import p1.MyClass1;
public class Test extends MyClass1 {
    public void func() {
        //公共属性,任何地方都可以访问
        System.out.println(a);
        // Error,b 为私有属性,类外无法访问
        //System.out.println(b);
        // c 是受保护属性,子类可以访问
        System.out.println(c);
        // Error,d 是默认属性,不同包中的类不能访问
        //System.out.println(d);
        // func1()是公共方法,任何地方都可以访问
        func1();
        // Error,func2()为私有方法,类外无法访问
        //func2();
        // func3()是受保护方法,子类可以访问
        func3();
        // Error,func4()是默认方法,不同包中的类不能访问
        //func4();
    }
}
```

包 p1 中的 Test 类，可以访问同一个包中的公开类 MyClass1 和默认类 MyClass2，可以

访问类中除了私有成员之外的其他成员。

包 p2 中的 Test 类可以访问包 p1 中的公开类 MyClass1，可以访问其中的 public 成员。但是，包 p2 中的 Test 类不能访问包 p1 中的默认 MyClass2。

包 p3 中，Test 类是 MyClass1 的子类，可以访问 MyClass1 中的 public 和 protected 成员。

程序中对访问权限不足的语句都加了注释符进行屏蔽，并进行了注释说明，读者可以放开注释进行测试。关于子类和继承的知识，请参见第 6 章。

5.11　综合案例

2020 年 6 月 23 日 9 时 43 分，我国在西昌卫星发射中心用长征三号乙运载火箭，成功发射北斗系统第五十五颗导航卫星，暨北斗三号最后一颗全球组网卫星，至此北斗三号全球卫星导航系统星座部署比原计划提前半年全面完成。

从立项论证到启动实施，从双星定位到区域组网，再到覆盖全球，我国卫星导航系统建设历经 30 多年探索实践、三代北斗人接续奋斗，走出了一条自力更生、自主创新、自我超越的建设发展之路，建成了我国迄今为止规模最大、覆盖范围最广、服务性能最高、与百姓生活关联最紧密的巨型复杂航天系统，成为我国第一个面向全球提供公共服务的重大空间基础设施，为世界卫星导航事业发展做出了重要贡献，为全球民众共享更优质的时空精准服务提供了更多选择，为我国重大科技工程管理现代化积累了宝贵经验。

1. 案例描述

北斗系统至今发展共有三代，其中第一代也被称为“北斗卫星导航实验系统”，属于实验性质，自第二代开始的北斗系统被正式称为“北斗卫星导航系统”。北斗系统从开始建设到全面组网一共发射了 59 颗卫星，包含 4 颗北斗导航实验卫星，实际提供信号的数量为 55 颗。北斗导航卫星全球星座由地球静止轨道(GEO)卫星、倾斜地球同步轨道(IGSO)卫星和中圆地球轨道(MEO)卫星三种轨道组成。

使用所学知识编写一个北斗系统信息展示程序，展示三代北斗系统的启动时间、建成时间以及所发射的卫星信息。

2. 问题分析

(1) 确定案例中的实体对象：通过案例描述，程序中包括北斗和卫星两个对象，可以分别定义类表示。

(2) 定义卫星类 Satellite：卫星类需要定义卫星编号、发射日期、发射火箭、轨道类型以及卫星类型等属性。

(3) 定义北斗类 Beidou：北斗类需要定义编号、启动时间、建成时间以及包含的卫星等属性。

(4) 定义测试类 BeidouSystem：创建北斗对象，添加包含的卫星，显示北斗系统信息。

3. 实现代码

程序 5.12.1 Satellite.java

```
import java.time.LocalDate;
//卫星类 Satellite
public class Satellite {
    private int number;                          //卫星编号
    private LocalDate LaunchDate;                //发射日期
    private String rocket;                       //运载火箭
    private String orbit;   /*轨道类型,北斗导航卫星全球星座由地球静止轨道(GEO)卫星、倾斜地球同步轨道(IGSO)卫星和中圆地球轨道(MEO)卫星三种轨道卫星组成*/
    private String type;   /*卫星类型,包括北斗一号系统,北斗二号系统,北斗三号实验系统,北斗3号系统*/
    public Satellite() {

    }
    public Satellite(int number, LocalDate launchDate, String rocket, String orbit, String type) {
        super();
        this.number = number;
        LaunchDate = launchDate;
        this.rocket = rocket;
        this.orbit = orbit;
        this.type = type;
    }
    public int getNumber() {
        return number;
    }
    public void setNumber(int number) {
        this.number = number;
    }
    public LocalDate getLaunchDate() {
        return LaunchDate;
    }
    public void setLaunchDate(LocalDate launchDate) {
        LaunchDate = launchDate;
    }
    public String getRocket() {
        return rocket;
    }
    public void setRocket(String rocket) {
        this.rocket = rocket;
    }
    public String getOrbit() {
        return orbit;
    }
    public void setOrbit(String orbit) {
        this.orbit = orbit;
    }
    public String getType() {
        return type;
    }
    public void setType(String type) {
```

```
        this.type = type;
    }
    /**
     * @return 卫星名称
     */
    public String getName() {
        /*北斗一号系统的四颗卫星都称为北斗导航实验卫星,北斗二号发射的第一颗卫星通常称为北斗导航一号卫星,人们常说的北斗 55 颗星是指北斗二号系统和北斗三号系统运行和建设过程中发射卫星的总数*/
        String name = "";
        if(number < 5) {
            name = "第" + number + "颗北斗导航实验卫星";
        }
        else {
            name = "第" + (number - 4) + "颗北斗导航卫星";
        }
        return name;
    }
    public void showInfo() {
        System.out.println(this.getName() + "\t" + this.getLaunchDate() + "\t" + this.getRocket() + "\t" + this.getOrbit());
    }
}
```

程序 5.12.2　Beidou.java

```
package com.wfu.ch05.shopping;
/*北斗类 Beidou*/
public class Beidou {
    private int number;                          //编号
    private String start;                        //启动时间
    private String finish;                       //建成时间
    private Satellite[] satellites;              //包含的卫星
    private int count = 0;                       //卫星数量
    public Beidou() {

    }
    public Beidou(int number, String start, String finish) {
        super();
        this.number = number;
        this.start = start;
        this.finish = finish;
    }
    public Beidou(int number, String start, String finish, Satellite[] satellites) {
        this.number = number;
        this.start = start;
        this.finish = finish;
        this.satellites = satellites;
    }
    public int getNumber() {
        return number;
    }
    public void setNumber(int number) {
        this.number = number;
    }
```

```
    public String getStart() {
        return start;
    }
    public void setStart(String start) {
        this.start = start;
    }
    public String getFinish() {
        return finish;
    }
    public void setFinish(String finish) {
        this.finish = finish;
    }
    public Satellite[] getSatellites() {
        return satellites;
    }
    public void setSatellites(Satellite[] satellites) {
        this.satellites = satellites;
    }
    public void addSatellites(Satellite satellite) {
        this.satellites[count++] = satellite;
    }
    public void showInfo() {
        System.out.println("北斗" + this.getNumber() + "号系统从" + this.getStart() + "开始
建设到" + this.finish + "建成,包括" + this.getSatellites().length + "颗卫星.");
        System.out.println("部分卫星信息如下:");
        System.out.println("卫星名称\t\t" + "发射时间\t" + "\t 运载火箭\t" + "轨道\t");
        int i = 0;
        while(this.getSatellites()[i]!= null) {
            this.getSatellites()[i].showInfo();
            i++;
        }
        System.out.println();
    }
}
```

程序 5.12.3 BeidouSystem.java

```
package com.wfu.ch05.shopping;
/*北斗系统类 BeidouSystem*/
public class BeidouSystem {
    public static void main(String[] args) {
        Beidou[] beidous = new Beidou[3];
        //北斗系统从开始建设到全面组网一共发射了 59 颗卫星,其中北斗一号系统发射了 4 颗卫
        //星,北斗二号系统发射了 20 颗,北斗三号系统发射了 35 颗。
        Satellite[] satellites = new Satellite[59];
        beidous[0] = new Beidou(1,"1994 年","2003 年",new Satellite[4]);
        beidous[1] = new Beidou(2,"2004 年","2012 年",new Satellite[20]);
        beidous[2] = new Beidou(3,"2009 年","2020 年",new Satellite[35]);
        //创建北斗一号系统的 4 颗卫星
        satellites[0] = new Satellite(1,LocalDate.of(2000,10,31),"CZ-3A","GEO","北斗一号
系统");
        satellites[1] = new Satellite(2,LocalDate.of(2003,05,25),"CZ-3A","GEO","北斗一号
系统");
        satellites[2] = new Satellite(5,LocalDate.of(2007,04,14),"CZ-3A","MEO","北斗二号
系统");
```

```
        satellites[3] = new Satellite(6,LocalDate.of(2009,04,15),"CZ-3C","GEO","北斗二号
系统");
        satellites[4] = new Satellite(22,LocalDate.of(2015,07,25),"CZ-3B","MEO","北斗三
号实验系统");
        satellites[5] = new Satellite(23,LocalDate.of(2015,07,25),"CZ-3B","MEO","北斗三
号实验系统");
        satellites[6] = new Satellite(28,LocalDate.of(2017,11,05),"CZ-3B","MEO","北斗三
号系统");
        satellites[7] = new Satellite(29,LocalDate.of(2017,11,05),"CZ-3B","MEO","北斗三
号系统");
        int i = 0;
        while(satellites[i]!= null) {
            int x = satellites[i].getType().charAt(2) - '1';
            beidous[x].addSatellites(satellites[i]);
            i++;
        }
        beidous[0].showInfo();
        beidous[1].showInfo();
        beidous[2].showInfo();
    }
}
```

小结

（1）面向对象思想：将数据及对数据的操作行为放在一起，作为一个相互依存、不可分割的整体——对象。对相似的对象进行分类、抽象后，定义为类。类是对象的模板，对象是类的一个具体实例。程序的执行表现为一组对象之间的交互通信。

（2）面向对象主要有三大特征：封装、继承和多态。

（3）Java 语言中定义类的关键字是 class。

（4）获得一个类的对象一般经过以下两步：首先使用类作为数据类型声明一个变量，即定义一个该类的对象，给对象命名。然后，使用 new 运算符创建该对象，在内存中为该对象分配地址空间，并把该对象的引用赋给声明好的变量。

（5）使用“对象.变量”和“对象.方法”访问类的属性和方法。

（6）类中与类名相同且没有返回类型的方法为构造方法。构造方法用于创建并初始化对象。

（7）如果类中没有显式地定义任何构造方法，Java 就为该类提供默认（无参数）的构造方法，否则就没有默认的构造方法。

（8）在同一个类中，多个方法具有相同的名字，但方法签名不同，即参数的个数、类型或顺序不同，称为方法的重载。构造方法也可以重载。

（9）方法的返回类型不是方法签名的一部分，在方法重载时，不能将返回类型不同当成两个方法的区别。

（10）按值传递是将要传递的参数（实参）的“值”传递给被调方法的参数（形参），实参和形参在内存中占用不同的空间，当实参的值传递给形参后，两者之间互不影响。所以按值传递不会改变原始参数的值。基本数据类型的参数是按值传递的。

（11）按引用传递是将参数的引用传递给方法，实参和形参指向的是同一内存空间。在方法中对形参进行修改操作，将导致实参状态的改变。Java 中对象作为参数一般是按引用传递。

（12）this 关键字代表当前所在类的对象。this()方法表示类的一个构造方法，用于在构造方法中调用同一个类的其他构造方法。this()必须放在构造方法的第一行。

（13）static 关键字可以用来修饰类的成员，包括成员变量、常用方法以及代码块等，表示该成员是静态的。

（14）Java 推荐使用“类名.方法名”和“类名.静态变量”访问静态方法和静态变量，静态方法又称为类方法，静态变量又称为类变量。

（15）实例方法可以访问实例方法和实例变量，也可以访问静态方法和静态变量。静态方法只能访问静态方法和静态变量，不能访问实例方法和实例变量。

（16）使用 package 关键字可以指定类所属的包。package 语句必须放在类的源文件的第一行。包的层次对应文件目录的层次。

（17）可以使用“包名.类名”方式调用不同包中的类。使用 import 关键字导入类以后，可以直接使用该类声明和创建对象。

（18）Java 中定义了 private（私有的）、protected（受保护的）、public（公开的）访问修饰符，同时定义了一个默认的（friendly，友好的）服务级别，用于声明类、属性和方法的访问权限。

习题

一、单选题

1. 以下代码在类的构造方法中调用了 xMethod()，则 xMethod()在类中是(　　)。

```
public MyClass() {
    xMethod();
}
```

A. 静态方法　　B. 实例方法　　C. 静态方法或实例方法

2. 如下所示，在 main()方法中调用了 xMethod()方法，则 xMethod()在类中是(　　)。

```
public static void main(String[] args) {
    xMethod();
}
```

A. 静态方法　　B. 实例方法　　C. 静态方法或实例方法

3. 分析以下代码，正确的选项是(　　)。

```
public class Test {
    private int t;
    public static void main(String[] args) {
        int x;
        System.out.println(t);
    }
}
```

A. 变量 t 未初始化，因此会导致错误

B. 变量 t 是私有的，因此不能在 main()方法中访问

C. 变量 t 是非静态的，不能在静态的 main()方法中引用它

D. 变量 x 未初始化，因此会导致错误

E. 程序编译和运行良好

4. Java 的访问修饰符中，限制性最高的是(　　)。

A. public　　B. protected　　C. private　　D. friendly

二、多选题

1. 以下哪些可能是类 Orange 的构造方法？(　　)

A. Orange(){…}　　B. Orange(…){…}

C. public void Orange(){…}　　D. public Orange(){…}

E. public int Orange(…){…}　　F. private Orange(…){…}

2. 假设包 p1 中包含包 p2，类 A 直接属于包 p1，类 B 直接属于包 p2，在类 C 中要使用类 A 和类 B，需要如何导入？(　　)

A. import p1. *；　　B. import p1. p2. *；

C. import p2. *；　　D. import p1. p2；

三、判断题

1. 类中与类名相同的方法就是构造方法。(　　)

2. 如果类中没有定义无参的构造方法，Java 就会为该类自动提供无参数的构造方法。(　　)

3. 在同一个类中，多个方法具有相同的名字，但方法签名不同，即参数的个数、类型或顺序不同，称为方法的重载。(　　)

4. 方法的返回类型是方法签名的一部分，返回类型不同的同名方法构成重载。(　　)

5. 按引用传递是将参数的引用传递给方法，实参和形参指向的是同一内存空间。在方法中对形参进行修改操作，将导致实参状态的改变。(　　)

6. this 关键字代表当前所在类的对象。this()方法用于在构造方法中调用同一个类的其他构造方法。this()必须放在构造方法的第一行。(　　)

7. 静态方法可以访问静态方法和静态变量，也能访问实例方法和实例变量。(　　)

8. 用 package 关键字可以指定类所属的包，必须放在类的源文件的第一行。(　　)

四、编程题

1. 请按照以下要求设计一个 Person 类，并进行测试。要求如下。

(1) Person 类中定义一个成员变量 name。

(2) Person 类中创建两个构造方法，其中一个构造方法是无参的，该方法中使用 this 关键字调用有参的构造方法，在有参的构造方法 Person(String name)中使用 this 关键字为成员变量赋值，并输出成员变量 name 的值。

(3) 在 main()方法中创建 Person 对象，调用无参的构造方法。

2. 编写一个程序，计算箱子的体积。表示箱子的类为 Box，有构造方法可以传入箱子的长、宽、高，有计算箱子体积的方法 getVolumn()。

3. 编写一个程序，计算两点之间的距离。表示点的类 Point，有两个属性 x、y 表示坐标，一个方法 distance(Point p1, Point p2)计算两点间的距离。

4. 设计一个类 OverLoadDemo，其中有求两个数的最大值的方法 getMax()，要求有两个重载方法，一个方法的参数为两个整数，另一个方法的参数为两个 double 型数。

5. 编写司机开车的程序。车类 Car 有名称属性 name 和运行方法 run()，司机类 Driver 有姓名属性 name 和开车方法 drive()。

6. 在包 p02 中定义一个类 FoundMin，使用下面的格式编写一个方法，求一个 double 型数组中的最小元素：

```
public static double min(double[] array)
```

在包 p02 中定义一个类 Test，提示用户输入 5 个 double 型数，并存放到一个数组中，然后调用这个方法返回最小值。

7. 创建学生类和教师类，分别给出两个类的特征定义、行为定义。创建学生对象和教师对象，并显示学生信息或教师信息。创建一个学生数组，容纳一个宿舍的学生，并显示各名学生的基本信息。

8. 修改超市购物程序，在商品类 Product 中添加价格和数量属性，在购物者类 Person 中添加购物清单，适当修改 Person 类购物方法 shopping()和 Market 类的售卖方法 sell()，测试购物过程，输出购物结果(购物小票)。