第5章 chapter 5

类和对象

编程语言为计算机解决问题提供了抽象机制。因此，编程语言所能提供的抽象化类型和方法决定了解决问题的复杂度和质量。面向对象编程语言以客观事物为本，以接近人类思维和客观事物本来面目的方法去解决问题。本章开始深入介绍面向对象编程，围绕面向对象程序的三个特征展开，即封装、继承和多态。

5.1 类与对象的关系

类(Class)和对象(Object)是面向对象方法的核心概念。类是对某一类事物的描述，是抽象的、概念上的定义。对象是实际存在的某类事物的个体，因而也称为实例(Instance)。例如，汽车类描述了汽车的属性和汽车具有的行为，它是所有汽车对象的模板、图纸。对象是一个个具体实在的个体，一个类可以对应多个对象。如果将对象比作具体的汽车，那么类就是汽车的设计图纸。类和对象的区别如下：

类是实体对象的模型，是抽象的、不具体的概念，对象是根据模型创造的具体实体。通过定义模型，可以规定实体对象所具有的属性和行为。

5.2 类的设计与 UML 建模

面向对象程序分析经常使用 UML 建模。UML(Unified Modeling Language，统一建模语言)由 OMG 组织(Object Management Group，对象管理组织)在 1997 年发布。UML 的目标之一就是为开发团队提供标准通用的设计语言。UML 提出了一套 IT 专业人员期待多年的统一的标准建模符号。使用 UML，IT 人员能够阅读和交流系统架构与设计规划，就像建筑工人多年来所使用的建筑设计图一样。UML 提供了一种适用于所有面向对象方法学的标准记号体系。

类是描述一类对象的属性(Attribute)和行为(Behavior)。在 UML 中，类用划分成三部分的矩形表示。图 5.1 所示为 UML 表示的 Car 类。

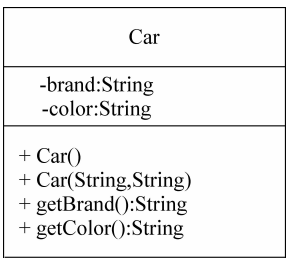

图 5.1 Car 类图

矩形的第一部分是类名。类名应尽量用领域中的术语，要明确、无歧义，利于开发人员与用户之间的相互理解和交流。一般而言，类的名字是名词。

矩形的第二部分是类的属性，用以描述对象的共同特点，该项可省略。图 5.1 中 Car 类有 brand、color 特性。类的属性能够描述并区分每个对象。根据图的详细程度，类的属性可以包括属性的可见性、属性名称、类型、默认值和约束特性。UML 规定类的属性的语法为：

可见性　属性名 ：　类型 =默认值

不同属性具有不同的可见性，即访问控制权限。常用的可见性有 public、private、protected 和默认 4 种，在 UML 中分别用＋、－、＃和“无”表示。图 5.1 中 brand 属性描述为“- brand ：string”，表示 brand 属性的访问范围是 private。

类的操作(Operation)项可省略。操作用于修改、查找类的属性或执行某些动作。操作通常也称为功能，它们被约束在类的内部，只能作用到该类的对象上。操作名、返回类型和参数表组成操作界面。UML 规定操作的语法如下：

可见性 操作名 (参数表) ：返回类型

如图 5.1 中的 getBrand 操作，其中“＋”表示该操作可被所有类访问，返回类型为字符串。

5.3 类

类将数据和方法封装在一起，数据表示属性，方法表示行为，程序的基本单元是类。对事物的所有描述都要在类中，类是面向对象的封装性的表现之一。Java 用关键字 class 表示类。

有了类的设计模型，就能产生代码。例 5.1 是根据汽车类的 UML 图开发汽车类的实现代码。

5.3.1 汽车类实例

【例 5.1】 汽车类。

```
//Car.java
public class Car {
  private String brand;     //私有属性
  private String color;     //私有属性

  Car(){
      brand="Test";
      color="gray"
  }
```

```
    Car(String brandIn, String colorIn) {
        brand=brandIn;
        color=colorIn;
    }
    public String getBrand(){
        return brand;
    }
    public String getColor(){
        return color;
    }
}
```

Car 类的设计是程序员对现实世界的抽象，而类的实现是将这种抽象以机器可理解的编程语言进行重新表示。好比翻译工作，将人类语言翻译为机器语言。类的设计直接决定了类的实现。

5.3.2 定义类

定义类的语法为：

```
[修饰符] class 类名
{
  属性
  方法
}
```

类名由用户自己定义，一般首字母大写，类名能够反映功能。Java 中程序都以类的形式存在。类中包括属性和方法，分别称为成员变量和成员方法，它们都要包含在“{}”内。上例中，Car 类有两个成员变量 brand 和 color，有两个成员方法 getBrand()和 getColor()。一个类的方法可以直接访问同类中的任何成员变量，因此 getBrand()方法可以直接访问 brand 成员变量。成员变量 brand 和 color 前面有 private 修饰，表示私有的，只能在类的内部访问。成员方法 getBrand()和 getColor()前面有 public 修饰，表示其他类可以访问。

类名前也有 public 修饰，表示类可以被所有其他类访问，是公有的。public 修饰符也叫访问权限修饰符，表示访问范围，可以修饰类、方法或属性。

类的成员的访问权限修饰符主要有 public、private、protected 和默认的 4 种。访问权限表示属性、方法或类的可见性，即可以被其他类或方法访问(调用)的范围。Java 的成员除了成员方法和成员变量外，还有语句块、内部类等。4 种访问权限分别代表了不同的访问范围。

(1) 公共类型：public。

类的成员声明是 public 时，所有其他类都可以访问该成员。

(2) 私有类型：private。

private 关键字修饰的成员是私有的，不能被该成员所在类之外的任何类访问。如例 5.1 中，属性被修饰为 private，则只能在 Car 类中访问这些属性，其他类不能访问。

(3) 默认类型：default。

成员前没有写任何访问修饰符时，其访问权限是默认的包访问类型，即在同一个包(文件夹)中的类是可见的。也就是说，同一个包中的类，默认类型相当于 public；而包外的类则相当于 private。

(4) 保护类型：protected。

标识为保护类型的成员用 protected 关键字修饰，成员的访问权限相当于包加继承关系的类。继承关系将在后续章节介绍。

表 5.1 所示为成员的访问修饰符和对应的可见性。

表 5.1 访问限制修饰符

可 见 性	public	protected	默认(包访问)	private
对同一个类	是	是	是	是
对同一个包中的任何类	是	是	是	否
对包外所有非子类	是	否	否	否
对同一个包中的子类基于继承访问	是	是	是	否
对包外的子类基于继承访问	是	是	否	否

访问限制修饰符不能用来修饰局部变量(方法内定义的变量)，否则将会报错。而且局部变量的作用域是局部，也没有必要使用访问修饰符。

5.4 对 象

5.4.1 创建对象

类是对现实世界的抽象。例如定义了汽车类，但要开车时使用的是某一辆具体的汽车，而不能是抽象的模型。对象是类的实例，类是对象的抽象表示。

创建类的对象使用 new 关键字和类名。类名就像 byte、int 等基本数据类型一样，表示数据的类型。对象名就是变量名。创建对象形式如下：

```
类名 对象名 =new 类名( [参数列表] );
```

new 关键字是创建对象的重要操作，表示在内存中开辟空间，空间大小根据具体类确定。对象类型是已经定义好的类，例如创建汽车对象有如下操作：

```
Car myCar =new Car();
```

创建的对象名为 myCar，它的类型是 Car。Car 类的属性有颜色和型号，因此对象 myCar 就拥有这两个属性。对象名 myCar 是指向 new 开辟空间的引用句柄，如图 5.2 所示。在 Java 内存中有“栈”和“堆”。在方法中定义的基本类型的变量和对象的引用变量都是在栈内存中分配。堆内存用于存放由 new 创建的对象。

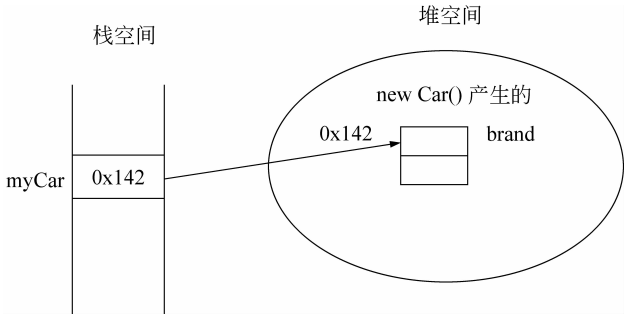

图 5.2 创建对象示意图

有时也可以不定义对象的句柄，而直接调用这个对象的方法。这样的对象叫作匿名对象，如“new Car().getBrand();”。

如果一个对象只使用一次就可以使用匿名对象。匿名对象经常作为参数传递给方法调用。

前面介绍过，方法内部的变量必须进行初始化赋值，否则编译无法通过。当对象被创建时，虚拟机会自动对成员变量进行初始化，默认值如表 4.1 所示。因此，程序中可以不对成员变量初始化。

创建新的对象之后，就可以使用“对象名.成员”的格式来访问对象的成员属性和方法。例 5.2 演示了 Car 类的对象的创建和调用对象成员。

5.4.2 使用对象

【例 5.2】 对象的使用。

```
//TestCar.java
class TestCar {
  public static void main(String args[]){
    Car carOne =new Car("audiA8", "red");
    Car carTwo =new Car("porsche", "white");

    carOne.getColor();       //对象.方法操作
    carTwo.getBrand();
  }
}
```

上面的代码在 main 方法中创建了两个 Car 类的对象。在内存中，每一次 new 都开辟一个新的内存空间。因此，两个对象分别指向不同的存储空间保存各自属性。carOne

和 carTwo 是两个完全独立的对象，类中定义的成员变量在每个对象中都单独实例化，不会被共享。例 5.2 中程序运行的内存示意图如图 5.3 所示。

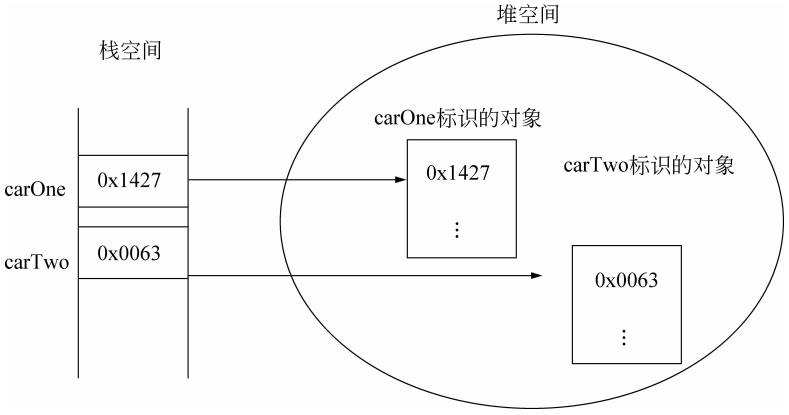

图 5.3　创建两个对象示意图

对象是具体的实例，上例中汽车类的对象有奥迪车，奥迪车是红色的。对象的属性有具体值。而类是抽象的，类中定义的属性、操作只是概念，没有具体数值。

5.5　成员变量与局部变量

声明和使用变量要遵循一个原则，就是变量只在其作用域的范围内有效。根据作用域不同，变量可以划分为两种：

（1）成员变量。在类中声明，在类中任何位置可以被访问。

（2）局部变量。在方法的内部或者代码块中声明，在该方法或者代码块内部可以访问，超出该范围则无法访问。

【例 5.3】　成员变量和局部变量实例。

```
//Car.java
public class Car {
    private String brand;          //成员变量 brand
    private String color;          //成员变量 color
    //局部变量 brandIn,colorIn
    Car(String brandIn, String colorIn) {
      String  getColor ="red";     //局部变量 getColor
      brand =brandIn;
      color =colorIn;
    }
    void print(){
      System.out.println(color);
      //将会编译出错,colorIn、getColor 超出了作用域
      System.out.println(colorIn);
```

```
        System.out.println(getColor);
    }
}
```

成员变量在声明时可以不进行初始化赋值，因为系统会自动赋默认值。而局部变量在声明时系统不会赋默认值，因此局部变量必须初始化。

5.6 构造方法

5.6.1 为什么需要构造方法

例 5.1 中有两个特殊的方法 Car()和 Car(String brandIn,String colorIn)。这两个方法的特殊之处在于方法名与类名一样，它们是构造方法(Constructor)。为什么要在类中定义构造方法呢？思考一下：如果创建某个类的对象，需要对属性赋值，有了具体数值，对象才是确定的实例而非抽象的概念。例如，红色奥迪车，计算机专业的张三同学。对象一定是具体的实体。具体就体现在它们的属性是有数值的。那么什么时候给属性赋值呢？当然是越早越好，最早的时刻是对象创建时，由此就有了构造方法，它在对象创建时调用。为便于记忆，构造方法的名字同类名。对象创建的语句是：

类名 对象名 =new 类名(); ⟵ 构造方法

new 后面的方法就是构造方法。创建对象时是通过调用构造方法为对象的成员变量赋初始值。当构造方法没有参数时，虚拟机会自动给成员变量赋默认值；当构造方法有参数时，将按照传递的参数初始化成员变量。可见，构造方法的作用是初始化成员变量。当一个类的对象用 new 产生时，构造方法就会自动调用。可以在构造方法中加入要完成初始化工作的代码。

总之，构造方法是类的一种特殊方法，它的特殊性主要体现在如下几个方面：

(1) 构造方法的方法名与类名相同。

(2) 构造方法没有返回值，不加 void 修饰，也不能在方法中使用 return 语句。

(3) 构造方法的主要作用是初始化成员变量。

(4) 构造方法的调用比较特殊，是在创建类的新对象时由系统自动调用该类的构造方法，并且只调用一次。

Java 中可以不定义构造方法，系统会生成一个默认的构造方法。这个构造方法的名字与类名相同，它没有任何参数。但如果用户自定义了构造方法，系统将不再生成无参数的构造方法。

【例 5.4】 构造方法实例。

定义一个圆类：有半径属性；有一个构造方法，构造方法中有半径参数，为成员变量赋值。创建两个半径分别为 3 和 5 的圆。

```
//Circle.java
public class Circle {
  double r;
  public  Circle (int rIn) {            //带参数的构造方法
      r =  rIn;
      System.out.println(r);
  }
  public static void main(String as[]) {
      //创建对象时调用构造方法,注意带参数
      Circle c1 =new Circle(5);
      Circle c2 =new Circle(3);
  }
}
```

实例中,构造方法有一个参数——半径 r。创建对象时使用的语句为“Circle c1＝new Circle(5);”。可见,创建新对象是通过调用构造方法实现的。由于构造方法带一个参数,调用时必须有参数。如果不带参数,由于不存在不带参数的构造方法,因此编译将出错。

注意,构造方法定义时必须与类名相同,没有返回值,在方法内部不允许用 return 语句。构造方法的参数可以有 0 个或多个。一个类也可以定义多个构造方法,方法的名字相同,但参数不同,这种情况叫作方法重载。

5.6.2 构造方法重载

【例 5.5】 构造方法重载实例。

```
//Person.java
class Person {
  private String name ="unknown";
  private    int age  =-1;

  //不带参数的构造方法
  public Person(){
    System.out.println("constructor1 is calling");
  }

  //有一个参数的构造方法
   public Person(String nameIn){
    name =nameIn;
    System.out.println("constructor2  is calling");
    System.out.println("name is:" +name);
  }

  //有多个参数的构造方法
  public Person(String nameIn, int ageIn){
```

```
        name =nameIn;
        age =ageIn;
        System.out.println("constructor3 is calling");
        System.out.println("name and age is:" +name +";" +age);
      }

      public void shout(){
        System.out.println("I'm happy.");
      }
    }

    TestPerson.java
    class TestPerson {
      public static void main(String ags[]){
        //调用不带参数的构造方法创建 P1 对象
        Person p1 =new Person();
        p1.shout();

        //调用带一个参数的构造方法创建 P2 对象
        Person p2 =new Person("Tommy");
        p2.shout();

        //调用带多个参数的构造方法创建 P3 对象
        Person p3 =new Person("Jenny", 20);
        p3.shout();
      }
    }
```

Person 类有三个构造方法，分别是不带参数、带一个参数、带两个参数。方法名相同，但参数类型不同或个数不同的情况称为方法重载。构造方法重载是比较常见的。调用构造方法时，编译器根据参数情况判断用哪个。

每个类里至少有一个构造方法，如果程序没有定义构造方法，系统会自动为类产生一个默认的构造方法。默认的构造方法没有参数，在方法体内也没有代码，即什么也不做。下面程序中 Construct 类的两种写法是一样的效果。

```
class Construct{

}
```

```
class Construct{
  public Construct()
  {    }
}
```

第一种写法虽然没有声明构造方法，但也可以用 new Construct()语句来创建类的实例对象，因为系统生产了默认的构造方法。

由于系统提供的默认构造方法往往不能满足要求，因此需要自定义类的构造方法。一旦为类定义了构造方法，系统就不再提供默认的构造方法。

```
class Car {
    String brand;
    public Car(String brandIn) {
        brand =brandIn;
    }
    public static void main(String args[]) {
        Car mycar =new Car();  //调用构造方法时出错
    }
}
```

编译上面的程序将会报错。错误的原因就在于 new Car()创建 Car 类的实例对象时要调用没有参数的构造方法,而程序没有定义无参数的构造方法,定义了一个有参数的构造方法,系统将不再自动生成无参数的构造方法。针对这种情况,应注意:只要定义有参数的构造方法,都需要再定义一个无参数的构造方法。

5.7 this 关键字

构造方法的作用是对成员变量进行初始化,在初始化变量时,有时会出现命名冲突问题。

【例 5.6】 命名冲突问题。

```
class Car{
    String brand;              //成员变量
    public Car(String brand ){
        brand =brand;          //初始化成员变量
    }
}
```

方法体中语句意图很好,右边代表形式参数,左边代表成员变量,可是一条赋值语句中完全相同的变量会代表两个不同的变量吗?这是不可以的。

这个问题涉及变量的作用域。变量的作用域表示变量的有效范围。在例 5.6 的构造方法中,成员变量和局部变量都处于作用域的范围之内,如何进行区分呢?解决方案是使用关键字 this。关键字 this 表示当前类,它有三种用法,下面分别介绍。

(1) 表示类中的成员。

(2) 使用 this 调用构造方法。

(3) 表示当前对象。

5.7.1 this 表示类的成员

this 表示类的成员时,使用方法为:

```
this.成员
```