第3章

面向对象编程

Java是一门纯面向对象的语言。本章将正式步入面向对象程序开发中，说明面向对象程序设计的基本理论、类、对象、继承、重载、覆盖和多态等特征，实现使用Java语言进行面向对象的程序设计。

3.1 面向对象程序设计思想

面向对象是一种将对象(Object)、类(Class)、封装(Encapsulation)、继承(Inheritance)、抽象(Abstract)、聚合(Aggregation)、消息传递(Communication with Messages)和多态(Polymorphism)等概念运用到软件开发中的方法。它实现了软件工程中重用性、灵活性和扩展性的3个目标，所以被广泛应用到软件开发的各个阶段，包括面向对象分析(OOA，Object Oriented Analysis)，面向对象设计(OOD，Object Oriented Design)和面向对象程序设计(OOP，Object Oriented Programming)。

1. 面向对象程序设计

面向对象程序设计是一种以“对象”为中心的程序语言模型，主要包括对象、类、数据抽象、数据封装、继承、动态绑定、多态性、消息传递等概念，使面向对象的思想得到了具体体现。面向对象开发领域流行的一句话是“万物皆对象”，即软件是由各种对象构成的一个动态运行的系统。每个对象都属于某一类特定类型，不同类型决定了属于它的对象可以接收不同的消息。

(1) 类(Class)。

用于定义某类事物的抽象特征和行为。通俗地说，类定义了某类事物的属性和它所具有的行为(或方法)。例如，在教学管理系统中，常常会设计“学生”类，类中包含“学号”“姓名”“入学时间”“年级”和“专业”等属性，同时也会设计“入学注册”“选课”等基本行为。在设计阶段，类一般采用统一建模语言(Unified Modeling Language，UML)图来描述。类由3部分组成：类名、属性和方法，如图3-1所示。其中左边为类的一般描述，右边为Student类的UML描述。

类是面向对象程序设计的基础，通过提供类机制，供开发人员构建适合应用所需的类型。面向对象程序设计中的核心问题是类从哪里来，类的属性和行为如何决定。实际上，类来源于对应业务领域的分析，从具体用例(Use Case)中提取出来。类的属性和方法则

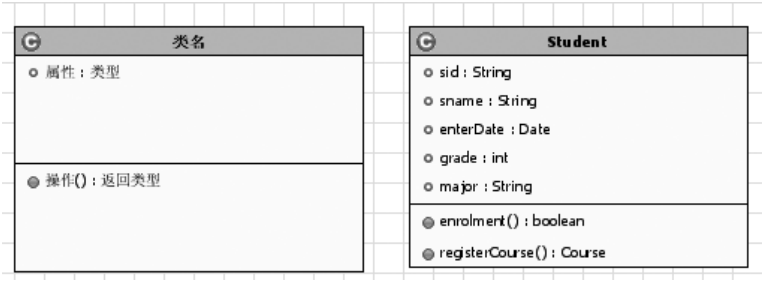

图 3-1 类图

根据业务领域的应用范围来决定。例如，“学生”类中一般会设计“性别”属性，但是作为教学管理系统，一般与“性别”没有紧密的关系或应用，所以教学管理系统中“学生”类可以不设计“性别”属性。

(2) 对象(Object)。

对象是类的实例(Instance)，是一个动态的概念，用于构成系统的具体执行逻辑。例如，“学生”类定义了教学管理系统中学生的抽象特征，是一个用来声明具体对象的类型。“李凯”是一个具体的学生，他是 2020 年入学的软件工程专业的一年级学生。他具有“学生”类中各个属性对应的值。所以，“李凯”是“学生”类的一个实例，他的属性值被称作“状态”。实际上，当用户编写的程序运行时，JVM 会为对象动态地分配内存空间。因此，类是抽象的，即用户自定义的数据类型，对象则是具体的，是类型的变量。

在面向对象软件开发中，软件开发人员首先从问题领域中寻找对象，并将同类对象的共有属性抽取出来，设计成类(即所说的抽象成类)，然后将类实例化为对象，由各种不同类型对象的相互作用构成软件。

(3) 消息传递(Message Passing)。

对象通过接受消息、处理消息、传出消息或使用其他类的方法来实现系统功能。对象的相互作用实现了系统的业务功能。例如，“李凯”发出“选课”消息。

2. 面向对象程序设计的特征

面向对象的软件开发过程更适于设计大型和复杂的系统，主要原因是实现了数据和功能的封装，易于代码维护和复用。面向对象程序设计包括 3 个主要特征：封装(Encapsulation)、继承(Inheritance)和多态(Polymorphism)。

(1) 封装。

将数据和方法封装在一个类中，对数据实现了隐藏，对公开方法通过消息传递实现调用(自身的具体实现，实现了隐藏)，并使程序的调试实现了局部化的目标。例如，“李凯”具有“选课”的方法，程序设计中实现“选课”功能时，只需给“李凯”这个对象传递“选课”消息，而不需要知道具体的“选课”流程。在程序调试过程中，一旦选课的业务不符合客户的逻辑需求，只需要修改“选课”方法的内部逻辑，实现调试的局部化。

(2) 继承。

是一种数据和方法共享的机制，实现了代码重用、易扩展的目标；在软件开发中，业务

系统中的类会出现一种概念分层的情况,例如“教学管理系统”中出现了“本科生”“研究生”和“预科生”等学生类别,可以将他们抽象出一张类图,如图 3-2 所示。其中“学生”称为父类,具有更加普遍意义上的属性和方法;“本科生”“研究生”和“预科生”称为子类,比父类具有要更加具体化的属性和方法。

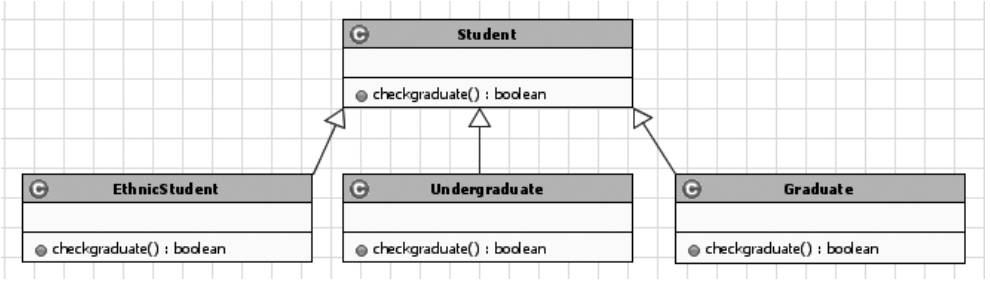

图 3-2 继承类图

(3) 多态性。

在程序执行时,对象会根据当前具体对应的消息执行的一种机制。实现了不同对象的消息执行不同动作的过程,具有动态绑定的灵活性,将面向对象开发技术具有的这一特征称为多态。多态性的实现可以通过“类”或“接口”实现。例如,在“学生”示例中,对于学生(Student 类)对象,向其发送“毕业检查”(checkgraduate)的消息,当该对象的具体实例是“研究生”时,将执行研究生的“毕业检查”;是“预科生”时,将执行预科生的“毕业检查”。这就是使用类实现的多态过程。

3.2 类

3.2.1 类的定义

根据前面的知识,写程序时首先要写类。为什么面向对象不首先写对象,而是写类呢?实际上,面向对象程序设计方法写程序前,首先是要对研究问题的领域进行对象寻找,然后再把具有相同或相似属性和行为的对象抽象成一个类。

Java 语言对类的一般定义格式如下。

```
[<modifiers>] class <class_name>[extends superclass]
[implements interfacea,...]{
    [<attribute_declarations>]                    //定义属性
    [<constructor_declarations>]                  //定义构造方法
    [<method_declarations>]                       //定义方法
}
```

类定义包括如下内容。

(1) modifiers:修饰符。修饰符包含访问修饰符和非访问修饰符。访问修饰符可以是 public、private 和默认(default)等值,用于说明类的访问权限。非访问修饰符可以是

abstract、final 或 static,用于说明定义的类的特性。

(2) class <class_name>: 类定义。class 为类关键字,class_name 为类名,是一个由用户定义的合法 Java 标识符。按照命名规范,类名的首字符一般大写。

(3) extends superclass: 类继承。extends 为关键字,superclass 为父类标识符。

(4) implements interfacea,...: 接口实现列表。implements 为关键字,interfacea 为接口标识符。

(5) { }: 为类结构体。

3.2.2 属性

完成类结构的定义后,要进行类属性的定义。类的属性定义包括访问修饰符、类型和标识符。格式如下。

```
[<access modifiers>] <type><attribute _name>;
```

(1) access modifiers。

指属性访问修饰符。public、protected、private 和 default。为了实现类的封装特性,一般情况下属性的访问修饰符设为 private,即只有类自身可以访问它,其他类都不能访问。为了能够让这些属性被访问,需要设置 public 的方法。这将在方法定义中说明。

(2) type。

类型名称。既可以是基本类型名,也可以是复合类型名。

(3) attribute _name。

属性名称。是合法的 Java 标识符。

【例 3-1】 根据图 3-3 所示的 Student(学生)类,使用 Java 语言描述属性部分。

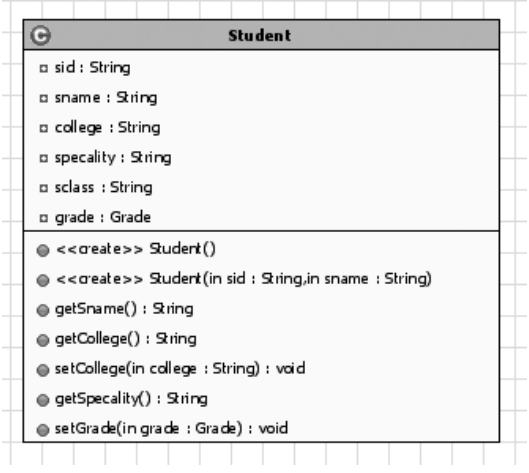

图 3-3 学生类结构

```
class Student{
    private String sid;                          //学号
    private String sname;                        //姓名
```

```
    private String sclass;                          //班级
    private String specialty;                       //专业
    private String college;                         //学院
    private Grade mark;                             //Grade 用户自定义成绩类,成绩
}
```

【例 3-2】 学生的成绩类包括学号、课名、学分和成绩,使用 Java 语言描述。

```
class Grade{
    private String sid;                             //学号
    private String cname;                           //课名
    private float credit;                           //学分
    private float score;                            //成绩
}
```

上面介绍的属性是必须实例化后才能访问的属性,称为成员变量(fields),还有一种属性是不需要实例化就可以使用的属性,而且是一种共享变量,称为类变量(class variable)。定义类变量的办法就是使用 static 修饰符对变量进行修饰。

【例 3-3】 定义一个共享变量。

```
public static int total =0;
```

还可以使用 final 修饰符定义类间共享的符号常量。

【例 3-4】 假设学生毕业的最低学分是 210 分,这个数据被所有学生对象共享,而且不能修改,所以,可以定义为类的符号常量。

```
class Student{
    public final static int CREDIT_MIN =210;
    private String sid;                             //学号
    private String sname;                           //姓名
    private String sclass;                          //班级
    private String specialty;                       //专业
    private String college;                         //学院
    private Grade mark;                             //Grade 用户自定义成绩类,成绩
}
```

3.2.3 方法

1. 方法的定义

方法是类向外提供的接口,也可以为类内部使用提供服务。方法的定义如下。

```
[<modifiers>] <return_type><method _name>(parameter lists) [throws exception
lists]{
        //method body (方法体);
}
```

每个方法包括以下 6 部分中的几部分。

(1) modifiers：访问修饰符，包括 public、protected、private 和 default。

(2) return_type：返回类型，方法运行后返回值的数据类型，可以是任何有效的类型，包括全部基本类型和复合类型，如果方法没有返回值，则返回类型为 void。

(3) method _name：方法名，符合规范的标识符。

(4) parameter lists：参数列表，表示传输的数据，包括数据类型和方法变量，使用“,”对多个参数进行分割，方法定义时也可以没有参数。

(5) exception lists：异常列表，在第 7 章中介绍。

(6) 方法体：方法的功能定义。

方法的使用根据定义也分为两种：一种是必须被实例化后才能使用的方法，称为成员方法；另一种是通过类名进行访问的方法，称为类方法。使用 static 进行定义。下面演示成员方法的定义，类方法将在 3.7.1 节中详细介绍。

【例 3-5】 实现一个将成绩类对象中的属性逐一输出显示的方法。

程序 Grade.java 如下。

```
public class Grade {
    private String sid;
    private String cname;
    private float credit;
    private float score;
    public void showInfo() {
        System.out.println("sid=" +sid +", cname=" +cname +", credit=" +
                           credit +", score=" +score );
    }
```

【例 3-5】中定义的 showInfo 方法没有返回值，因此方法类型为 void，方法无参数，小括号中的内容为空，方法体为一条输出语句，输出对象的属性及对应的属性值。

由于类将属性进行了封装，特别是将属性设置为私有类型(private)，其他对象无法通过“对象名.属性名”的方式访问它们，所以需要设置方法提供对它们的访问。对它们的访问包括两类方法：读取和设置属性的值。为了方便用户定义与使用，Java 定义了统一的方法名。读取属性值为 getXxx()，设置属性值为 setXxx()方法。

【例 3-6】 对成绩类属性进行读取和设置方法的定义。

程序 Grade.java 如下。

```
public class Grade {
    private String sid;
    private String cname;
    private float credit;
    private float score;
    public void showInfo() {
        System.out.println("sid=" +sid +", cname=" +cname +", credit=" +
                           credit +", score=" +score );
```

```
    }
    public String getSid() {
        return sid;
    }
    public void setSid(String sid) {
        this.sid =sid;
    }
    public String getCname() {
        return cname;
    }
    public void setCname(String cname) {
        this.cname =cname;
    }
    public float getCredit() {
        return credit;
    }
    public void setCredit(float credit) {
        this.credit =credit;
    }
    public float getScore() {
        return score;
    }
    public void setScore(float score) {
        this.score =score;
    }
}
```

2. 不定参数个数的方法定义

使用方法时,常有一些不定参数个数的广义方法的使用。如 C 语言中 printf("%d,%s",a,b)的使用,根据格式参数来决定对应参数的个数使用。Java 也提供了这种方法的定义。参数的定义使用 3 个点(…),格式为(类型 …参数变量)。

【例 3-7】 不定参数个数方法的定义。

程序 TestMethod.java 如下。

```
class TestMethod{
      static void myprint(String format,Object …objects){
      //不定参数个数的方法定义
          String temp =format;
          String result ="";
          int index =0;
          int num =0 ;
          while((index =temp.indexOf('%'))>=0 && index <format.length()){
              char ch =temp.charAt(index +1);
```

```
                switch(ch){
                case 's':
                    result +=objects[num].toString();
                    num++;
                    break;
                case 'd':
                    result +=objects[num].toString();
                    num++;
                    break;
                }
                temp =temp.substring(index+1);
            }
            System.out.println(result);
    }
}
```

这里定义了一个不定参数的 myprint 方法，根据 format 参数中%的个数来确定后面参数的个数，用%后面的 s 和 d 的字符区分具体的数据类型。如 myprint("%d",12345)；myprint("%s","hello!")；myprint("%s,%d","hello!",12345)；myprint("%s,%d,%s","hello!",12345,"234")；等都是合法的使用。读者可以根据需要，对上面方法的数据类型和功能进行扩充。

3. 方法重载

在实际开发中，有时可能会在同一个类中定义几个功能相同但参数不同的方法。如定义一个计算面积的方法，不同形状的面积计算方法是不同的，需要的参数也不相同，如果为三角形、圆形、长方形等各种形状定义一个计算面积的方法，需要定义多个方法，而且方法的命名也会枯燥无意义。因此，Java 提供了方法重载(overloading)机制来实现同一个类中存在多个同名的方法。对于重载的多个方法，应具有相同的方法名，但为了使编译器能够区分这些方法，必须彼此间有不同的成分。因此，Java 语言要求进行方法重载时，必须遵循以下原则。

(1) 在同一个类中，方法名必须相同的多个方法。

(2) 方法的参数列表必须不同：表现在参数个数不同、参数类型不同或参数顺序不同。

(3) 方法的返回类型可以相同，也可以不同。

【例 3-8】 使用方法的重载实现三角形、圆形、矩形、正方形的面积计算。

程序 OverLoadTest.java 如下。

```
public class OverLoadTest {
        /**
         * 方法重载
         */
        public static void main(String[] args) {
```

```
        OverLoadTest olt =new OverLoadTest();
        double area =olt.area(3, 4, 5);
        System.out.println("area="+area);
    }
    //求三角形的面积
    public double area(double a,double b,double c){    //a、b、c为三角形三边长
        double p =(a+b+c)/2;
        double area =Math.sqrt(p*(p-a)*(p-b)*(p-c));
                                                        //海伦公式求解三角形面积
        return area;
    }
    //正方形面积
    public double area(double d){
        return d*d;
    }
    //矩形面积
    public double area(double a,double b){
        return a*b;
    }
    //圆的面积
    public double area(float r){
        return 3.14*r*r;
    }
    /* 下面方法如果定义在类中,则编译错误
     * public float area(float r){
       return (float)3.14*r*r;
    }*/
}
```

【例3-8】中在同一个类中定义了3个同名方法,方法名都为area,但参数个数、类型不同,实现了方法的重载。在程序的编译过程中,根据变量类型来找相应的方法,因此方法重载实现了Java的“编译时多态”。编译之后的方法名实际加上了各参数类型成为方法的签名,编译器也是通过方法的签名来识别方法的。

3.2.4 构造方法

实例化对象时,需要同时初始化属性。Java提供了专门方法,称为构造方法(Constructor Method)。构造方法是一个特殊的方法,方法名必须与类名相同,并且没有返回类型。注意:没有返回类型是指不能定义返回类型,而不是void类型。

【例3-9】 定义成绩类的构造方法。

程序Grade.java如下。

```
public class Grade {
    private String sid;
```

```
    private String cname;
    private float credit;
    private float score;
    //构造方法,完成成员变量的初始化
    public Grade(String sid, String cname, float credit, float score) {
        this.sid =sid;
        this.cname =cname;
        this.credit =credit;
        this.score =score;
    }
    //省略属性的读取和设置方法及 showInfo 方法
}
```

Java 提供构造方法的目的是在创建对象时初始化成员变量。定义类时,可以显式地定义构造方法,也可以不定义。不显式定义构造方法时,Java 编译器会为每一个类提供一个缺省的无参构造方法。用默认的构造方法初始对象时,由系统用默认值来初始化对象的成员变量。Java 对各种数据类型都指定了默认的初始值,数值型默认初始值为 0,boolean 类型的默认初始值为 false,char 类型的默认初始为\0,引用类型默认初始值为 null。需要注意,一旦类中显式定义了一个或多个构造方法,系统将不再提供默认的构造方法。

构造方法是为了方便用户实例化对象时同时进行成员变量的初始化。【例 3-9】中的构造方法只提供了一种方式实现成员变量的初始化。在不同的应用情况下,需要进行不同的初始化,这需要通过构造方法重载来实现。构造方法和普通方法一样,也可以重载,通过多个参数不同的构造方法为对象成员变量的初始化提供方便。

【例 3-10】 重载成绩类的构造方法。

程序 Grade.java 如下。

```
public class Grade {
    private String sid;
    private String cname;
    private float credit;
    private float score;
    public Grade() {
        //无参构造方法
    }
    public Grade(String sid, String cname, float credit) {
        this.sid =sid;
        this.cname =cname;
        this.credit =credit;
        this.score =60.0f;
    }
    public Grade(String sid, String cname, float credit, float score) {
        this.sid =sid;
```

```
        this.cname =cname;
        this.credit =credit;
        this.score =score;
    }
  //省略属性的读取和设置方法
}
```

【例 3-10】给出了 3 个重载的构造方法,一个是不带参数的构造方法,且方法体为空,但调用时会根据成员变量的类型设置默认初始值;另外两个带参数,且参数个数不同,从而实现了构造方法的重载。

3.2.5 this 关键字

在【例 3-9】和【例 3-10】定义的构造方法中都使用了 this 关键字,那么 this 的作用究竟是什么呢? 在对象实例化方法和构造方法中,this 代表当前对象。主要用在实例化方法时对象属性的使用和构造方法的引用。

this 关键字有以下两种使用方法。

1. this 区分成员变量与局部变量

用法如下。

```
this.varName(成员变量)                                    //引用对象的成员变量
```

典型的情况是用在区分构造方法的参数与成员变量的标识符相同的情况下。【例 3-9】和【例 3-10】定义的构造方法中都使用了 this 来实现区分成员变量与局部变量。

```
public Grade(String sid, String cname, float credit, float score) {
    this.sid =sid;
    this.cname =cname;
    this.credit =credit;
    this.score =score;
}
```

方法中参数的名字 sid、cname、credit 和 score 与类中成员变量的名字是完全一致的。在构造方法中进行赋值时,为了区分成员变量与局部变量(方法的参数),使用了 this 关键字来实现,this.sid 指定是当前对象的 sid 成员变量,赋值运算符右侧的 sid 是局部变量,其他几条赋值语句同样使用 this 来指代当前对象,实现区分构造方法的参数与成员变量。

2. this 用于构造方法的调用

用法如下。

```
this(参数列表)                                            //调用本类的构造方法
```

用在构造方法的重载实现中,目的是调用其他构造方法。但是 this 语句必须放在第一行。

【例 3-11】 this 作为构造方法的使用。

程序 Grade.java 如下。

```
public class Grade {
    private String sid;
    private String cname;
    private float credit;
    private float score;

    public Grade() {
        //无参构造方法
    }
    public Grade(String sid, String cname, float credit) {
        this.sid =sid;
        this.cname =cname;
        this.credit =credit;
        this.score =60.0f;
    }
    public Grade(String sid, String cname, float credit, float score) {
        this(sid,cname,credit);
        this.score =score;
    }
    public String getSid() {
        return sid;
    }
    public void setSid(String sid) {
        this.sid =sid;
    }
    public String getCname() {
        return cname;
    }
    public void setCname(String cname) {
        this.cname =cname;
    }
    public float getCredit() {
        return credit;
    }
    public void setCredit(float credit) {
        this.credit =credit;
    }
    public float getScore() {
        return score;
    }
    public void setScore(float score) {
        this.score =score;
    }
```

```
    public void showInfo() {
        System.out.println("sid=" +sid +", cname=" +cname +", credit=" +
                           credit +", score=" +score );
    }
}
```

【例 3-11】修改了【例 3-10】中 4 个参数的构造方法,在方法体中使用了 this(sid, cname,credit),实现对 public Grade(String sid, String cname, float credit) 方法的调用。

3.3 对　　象

对象(Object)是软件建模中的基本单位,是一个运行状态(state)和行为(behavior)的综合体,对象之间通过消息传递(调用方法)相互作用。在面向对象程序设计中,类是对象的模板,依据类来创建一个对象。对象具有不同的方法,可以使用对象的不同状态和不同方法,使程序在运行中呈现出不同功能。

对象的使用包括对象的声明、实例化、使用和销毁。

3.3.1 对象的声明

在 Java 语言中,变量需要先定义,后使用。所以要想使用一个对象,必须先进行声明才能使用。凡是符合变量声明的地方,都可以用来声明对象。实际上,对象声明的是保存该对象引用的变量,将来可以通过这个变量对对象进行操作。

声明格式如下。

```
类名 对象名;
类名 对象名列表;
```

【例 3-12】 声明成绩类对象。

```
Grade grade;                          //声明了一个 Grade 对象 grade
Grade grade1,grade2;                  //声明了两个 Grade 对象 grade1 和 grade2
```

在 Java 中声明对象后,对象变量中存储的值不是实际对象属性的值,而是对象的一个引用(即地址)。在【例 3-12】中声明的 grade、grade1 和 grade2 中存储的都不是具体的对象,而是代表某个 Grade 对象的引用。它们的类型是 Grade,是一个类的类型,属于复合(引用)类型变量。声明对象时并没有创建对象,所以对象变量 grade、grade1 和 grade2 的初始值为空,如图 3-4 所示。

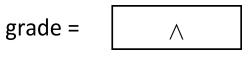

图 3-4　空对象 grade

3.3.2 对象的实例化

声明对象后,还不能真正使用对象。因为对象的声明并没有实现对象的创建,只有创建了对象才能使用对象。对象的创建也就是用一个类来实例化一个对象,所以对象的创建也叫作对象的实例化。创建对象主要是因为声明时没有为对象分配存储空间,所以必

须使用运算符 new 创建对象实例，才能使用对象。

实例化格式如下。

```
对象名 =new 类名();
```

【例 3-13】 实例化 Grade 类对象 grade。

```
grade=new Grade("20200101001","高数",3,65.0f);  //实例化对象
```

在实际开发中，常常是声明和实例化一起完成。

【例 3-14】 声明和实例化一起完成，下面的 Grade 类使用了【例 3-11】中的定义。

```
Grade grade1=new Grade("20200101002","面向对象程序设计",3,68.0f);
Grade grade2=new Grade("20200101003","面向对象程序设计",3,70.0f);
```

【例 3-13】和【例 3-14】中使用关键字 new 进行了对象的实例化，对象实例化的过程是为对象分配存储空间，并执行构造方法，完成属性的初始化。

new 关键字的作用如下。

① 为对象分配内存空间。

② 引起对象对应构造方法的调用。

③ 为对象返回一个引用。

在 grade 对象的实例化过程中，内存分配情况如图 3-5 所示。

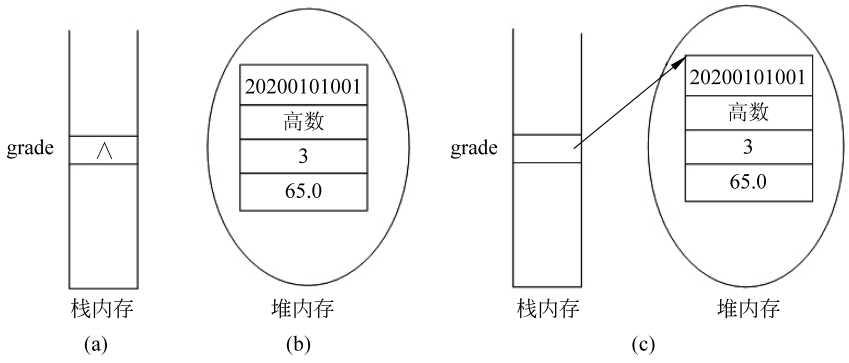

图 3-5 grade 对象的实例化过程

Java 中内存的逻辑分区情况如下。

① 栈内存：特定程序专用，采用先进后出的分配原则，存储空间分配连续，存储容量小，速度快。通常存放局部变量。

② 堆内存：所用应用程序公用，存储空间分配不连续，存储容量大，速度慢。通常存放成员变量、对象。

③ 代码区：专门存放方法、代码块。

④ 静态、常量存储区：存放静态变量、常量。

在对象的实例化过程中，为对象变量 grade 在栈内存中分配空间，为对象本身在堆内存中分配内存空间，调用构造方法执行其中的代码，实现属性值的设置，最后将堆内存中

对象的引用返回赋值给栈内存中的 grade,使得 grade 中存放堆内存中对象的“字符串表示”值,可以理解为是 grade 中存放对象的首地址或指针。

3.3.3 对象的使用

对象经过实例化后,就可以进行属性和方法的访问,但是能否访问这些内容与类中定义他们的访问属性有关,这将在后面介绍。

访问属性格式如下。

```
对象名.属性名;
```

访问方法格式如下。

```
对象名.方法名(参数列表);
```

【例 3-15】 属性和方法的访问。

```
System.out.println(grade.sid);
                          //grade 的访问修饰符为 public,将 grade 对象的 sid 属性输出
System.out.println(grade.getSid());
                          //grade 的访问修饰符为 private,使用 get 方法访问 sid
```

【例 3-16】 演示 Grade 类对象的实例化及使用过程。

程序 GradeTest.java 如下。

```
public class GradeTest {
    public static void main(String[] args) {
        //声明及实例化对象
        Grade grade=new Grade("20200101001","高数",3,65.0f);
        //对象使用
        grade.showInfo();
        //调用 setScore 重新设置分数
        grade.setScore(70.0f);
        System.out.println("重新设置分数后,grade 对象信息: ");
        grade.showInfo();
    }
}
```

程序运行结果如下。

```
sid=20200101001, cname=高数, credit=3.0, score=65.0
```

重新设置分数后,grade 对象信息如下。

```
sid=20200101001, cname=高数, credit=3.0, score=70.0
```

3.3.4 对象的销毁

Java 语言采用 JVM 自动垃圾回收机制来清理对象。JVM 会按照某种策略(如定时、

内存使用情况)使用垃圾回收机自动回收系统中不再使用的对象。

对应 C++ 语言的析构方法,Java 也提供了一个 finalize()方法。但是,本方法的工作原理是假设垃圾回收机要释放本对象的存储空间,回收前首先执行该方法。所以,可以使用 finalize()方法做垃圾回收时的清理工作,但是,finalize 方法不会在用户指定的时间执行清理工作。对于垃圾的回收,也可以使用 System.gc()对系统中的无效对象进行清理,但也并不保证立即执行。

实际上,不论上面哪种对象的销毁办法,都是在垃圾回收机工作时才开始进行清理工作的,而垃圾回收机什么时候开始工作,由 JVM 设定的策略决定。

3.3.5 对象的传递

对象也可以作为方法参数进行传递,但是,传递时使用的是对象的引用(对象变量)。定义方法时,参数的类型既可以是基本类型,如整型、实型、双精度型等,也可以是复合类型,如数组、对象等。

使用方法时,Java 进行参数传递是按照“值”传递的。传递基本类型参数时,复制实参(调用方法时程序给方法传递的实际数据)的值,传递给形参(方法声明时声明的参数),实参和形参在内存中占用不同的空间,两者互不影响,修改形参的值不会改变实参的值。传递复合类型参数时,也是按照“值”的方式传递,实参值传递给形参,但是引用类型变量实参中存放的是对象的引用,因此,形参得到的也是对象的引用,形参和实参指向了同一片内存空间。如果方法通过引用改变了所指对象的内容将会同时改变实参所指向对象的值。

【例 3-17】 演示参数传递的例子。

程序 PersonTest.java 如下。

```
public class PersonTest {
    public void m1(int n){
        n =1000;
    }
    public void m2(Person p){
        p.setAge(12);
        p.setName("Jack");
        p.setSalary(1234.56);
    }
    public void m3(Person p){
        p =new Person("Tom",23,9876.54);
    }
    /**
     * 方法的传值方式
     */
    public static void main(String[] args) {
        PersonTest pt =new PersonTest();
        int n =10;
```

```
        System.out.println("m1 方法调用前,n="+n);
        pt.m1(n);
        System.out.println("m1 方法调用后,n="+n);

        Person p1 =new Person("zhangsan",32,6666.66);
        //打印一个引用变量,则输出引用变量所指向对象的字符串表示,即打印该对象的
        //toString 方法的返回值
        System.out.println("m2 方法调用前,p1="+p1);
        pt.m2(p1);
        System.out.println("m2 方法调用后,p1="+p1);

        Person p2 =new Person("lisi",41,3333.02);
        System.out.println("m3 方法调用前, p2="+p2);
        pt.m3(p2);
        System.out.println("m3 方法调用后, p2="+p2);
    }
}
class Person{
    String name;
    int age;
    double salary;
    public Person(String name, int age, double salary) {
        this.name =name;
        this.age =age;
        this.salary =salary;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name =name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age =age;
    }
    public double getSalary() {
        return salary;
    }
    public void setSalary(double salary) {
        this.salary =salary;
    }
```

```
    //打印对象引用时默认调用该方法,表示打印出对象的字符串表示
    public String toString() {
        return "Person1[name=" +name +", age=" +age +", salary=" +salary
              +"]";
    }
}
```

程序运行结果如下。

```
m1方法调用前,n=10;
m1方法调用后,n=10;
m2方法调用前,p1=Person1[name=zhangsan, age=32, salary=6666.66];
m2方法调用后,p1=Person1[name=Jack, age=12, salary=1234.56];
m3方法调用前,p2=Person1[name=lisi, age=41, salary=3333.02];
m3方法调用后,p2=Person1[name=lisi, age=41, salary=3333.02]。
```

PersonTest 类中的 m1(int n)方法参数为基本数据类型 int,在参数传递过程中,实参 $n=10$ 复制给形参 n,在方法 $m1$ 中改变的是形参 n 的值,对实参 n 没有影响,因此在方法调用前后,n 的值不发生改变;m2(Person p)方法参数为复合类型(引用类型),并且在方法中通过引用变量改变了所指向对象的属性值,由于实参和形参所指对象相同,所以实参指向对象的属性值发生了改变(引用变量的值不发生改变);m3(Person p)方法参数为复合类型(引用类型),在方法中通过对形参重新赋值直接改变了引用变量的值,使得形参指向了其他对象。在这个过程中,实参值不会发生变化,所指对象的属性也没有改变。

3.4 继　　承

继承是面向对象程序设计的基本特征之一,是实现多态的基础。继承的主要目的是实现系统中代码的复用。

3.4.1 继承的定义

继承(Inheritance)是指在已有类型的属性和方法的基础上扩充和改造,得出新的类型,也就是定义一个类时可以继承一个已经存在的类。继承可以简化代码,实现代码复用,是多态的基础;满足“is a”关系就是类继承。

被继承的类称为父类(superclass),定义的新类称之为子类(subclass)。Java 中类的继承定义的语法如下。

```
[<modifiers>] class <subclass_name>extends superclass {
    [<attribute_declarations>]
    [<constructor_declarations>]
    [<method_declarations>]
}
```

其中,extends 就是类继承的关键字。创建一个新类时,系统中已经存在一个具有新

类的部分属性和方法的类,可以使用继承创建这个新类。这样就可以复用已经定义的属性和方法。

【例 3-18】 在实际成绩管理中,成绩不仅包括百分制,还有二分制和五分制。其中二分制的成绩为“不及格”和“及格”;五分制的成绩为“不及格”“及格”“中”“良”和“优秀”。那么成绩类如何设计?

(1) 解决方案 1:修改成绩类。

根据前面定义的成绩类,成绩类包括的属性为学号、课程名、学分和成绩。为了实现二分制、五分制和百分制,修改成绩类,增加成绩类型属性。1 表示百分制;2 表示二分制;3 表示五分制。其中成绩在百分制下使用 0～100 的实数;二分制下使用 0～1 的实数;五分制使用 0～4 的实数。

```
class Grade{
    public final static int CENTI_SCALE =1;
    public final static int BINARY_SCALE =2;
    public final static int FIVE_SCALE =3;
    private String sid;                                    //课程编号
    private String cname;                                  //课程名
    private float credit;                                  //学分
    private int gType;                                     //成绩类型
        private float score;                               //成绩
        //构造方法
    ⋮
        //读取成绩
        public String getScore() {
            String result =null;
            switch(gType){
            case CENTI_SCALE:
                result =Float.toString(score);
                break;
            case BINARY_SCALE:
                if(score ==0){
                    result ="不及格";
                }else{
                    result ="及格";
                }
                break;
            case FIVE_SCALE:
                if(score ==0){
                    result ="不及格";
                }else if(score ==1){
                    result ="及格";
                }else if(score ==2){
                    result ="中";
```

```
                }else if(score ==3){
                result ="良";
                }else if(score ==4){
                    result ="优秀";
            }
            break;
        }
        return result;
    }
}
```

成绩类的使用，如姚明参加了“高数”“IT 日语”和“Java EE 框架技术”课程的学习（其中“高数”是百分制，公共选修课“IT 日语”是二分制，选修课“Java EE 框架技术”是五分制），下面实例化这些课程的成绩对象，并使用它们。

```
class TestGrade{
    public static void main(String[] args){
        Grade grade;
        //编号“01”,高数,6 学分,百分制,89 分
        grade =new Grade("01","高数",6, Grade.CENTI_SCALE,89);
        System.out.println("成绩"+grade.getScore());
        //编号“02”,IT 日语,3 学分,二分制,1-表示“及格”
        grade =new Grade("02","IT 日语",3, Grade.BINARY_SCALE,1);
        System.out.println("成绩"+grade.getScore());
        //编号“03”,Java EE 框架技术,4 学分,五分制,3-表示“良”
        grade =new MyGrade("03","Java EE 框架技术",4, Grade.FIVE_SCALE,3);
        System.out.println("成绩"+grade.getScore());
    }
}
```

(2) 解决方案 2：使用继承。

在原有类型的基础上，继承实现二分制和五分制的成绩表示，百分制仍然由原来的成绩类来表示。类的继承模型如图 3-6 所示。

```
class BinaryGrade extends Grade {                                    //二分制成绩类
    private String biscore;
    public String getBiscore() {
        return biscore;
    }
    public void setBiscore(String biscore) {
        this.biscore =biscore;
    }
}
class FiveGrade extends Grade {                                      //五分制成绩类
    private String fscore;
    public String getFscore() {
```