第 5 章　类与类的继承——个人通讯录(一)

5.1　个人通讯录(一)

设计简单的个人通讯录系统(一),用于存储联系人信息,实现对联系人按照基本联系人、家人、工作伙伴等进行分类存储,具体要求如下:

(1) 联系人的基本信息包括姓名、性别、电子邮件及若干个联系电话;

(2) 家人除具有联系人基本信息之外,还需要定义家庭地址及生日;

(3) 工作伙伴除具有联系人基本信息之外,还需要添加所在公司及职称/职务信息。

5.2　程序设计思路

实体对象可以看作是具有内部属性及功能的构件,通过封装技术隐藏实体内部的信息及功能的实现细节。个人通讯录(一)涉及 3 个基本类,分别是联系人类、家人类及工作伙伴类。

联系人类封装联系人的基本信息及输出功能;家人类复用联系人类并添加特有属性,描述联系人的家庭地址与生日;工作伙伴类也复用联系人类,同时通过组合技术描述工作伙伴所在的公司及职称/职务信息。

5.3　关键技术

5.3.1　面向对象的基本概念

Java 是面向对象的程序设计语言,其基本思想是通过对象、类、封装、继承、多态等基本概念进行程序设计,它是从现实世界中存在的客观事物(即对象)出发构建软件系统的。

对象:作为构成系统的基本单位,对象是由数据及其行为所构成的封装体。对象包含 3 个基本要素,分别是对象标识、对象状态和对象行为。每一个对象必须有一个名字,以区别于其他对象,这就是对象标识;状态用来描述对象的某些特征;对象行为用来封装对象所拥有的业务操作。

例如,联系人 Contract 对象包含姓名、性别、电子邮件及联系电话等基本状态信息,同时还具有打印输出的行为特征。

类:对同类对象进行抽象形成类,它为属于该类的所有对象提供了统一的抽象描述,其内部包括状态和行为两个主要部分。类也可以被认为是一种自定义的数据类型,可以使用类定义变量,所有通过类定义的变量都是引用变量,它们将会引用到类的实例,即对象。

封装：封装是一种信息隐蔽技术，通过封装将对象的状态和行为结合成一个独立的模块，尽可能隐藏对象的内部细节(包括对象内部的私有状态及行为的具体实现)。封装的目的在于把对象的设计者与使用者分开，作为使用者不必了解对象内部的实现细节，只需要使用设计者提供的行为方法实现功能即可。

继承：继承表示类之间的层次关系。继承关系使得子类对象可以共享父类对象的状态和行为。继承又可分为单继承(一个子类仅拥有一个父类)和多继承(一个子类可以拥有多个父类)，Java 语言支持类的单继承，而 C++ 允许多继承。在程序设计过程中通过继承性，一方面得到了类的层次等级结构；另一方面，通过类的继承关系可以使公共的状态和行为特性得到共享，提高了软件的重用性。

多态：多态性是指同名的行为方法可在不同的类中具有不同的实现。在子类继承父类的同时，类的方法实现可以进行扩充或者修改，使子类的同名方法更适合子类对象。如父类图形(Shape)的绘图方法 draw()，在其子类圆形(Circle)和正方形(Square)都具有同名的绘制方法 draw()，但是绘制的内容和方式都是不同的。

5.3.2 类的定义

类是一种自定义的引用型数据类型，可以采用如下格式实现类的定义：

```
[public|abstract|final] class 类名{
    [初始化块的定义;]
    [成员变量的定义;]
    [方法的定义;]
}
```

public、abstract、final 称为修饰符，分别用于定义类的访问权限、抽象类及最终类等属性。方括号([])表示可选项。类名必须是合法的 Java 标识符。为了提高程序的可读性，Java 的类名通常由若干个有意义的单词组成，每个单词首字母大写，其余字母全部小写。

大括号({})之间的内容称为类体，主要包括 3 部分：初始化块、成员变量的定义、方法的定义。类中各成员之间的定义顺序没有任何影响，各成员之间可以互相调用。初始化块用于对类对象进行初始化操作，详细内容见 5.3.8 节；成员变量用于定义同类对象具有的状态数据；方法则定义类的行为特征或功能实现。

1. 成员变量

成员变量通常用于描述类或者类实例的状态信息，其定义格式如下：

```
[修饰符] 数据类型 成员变量[=默认值];
```

成员变量的修饰符用于定义成员变量的属性，如 public、protected、private、static、final 等。成员变量的数据类型既可以为基本数据类型，也可以是引用数据类型，如数组、类及接口等。

2. 方法

方法通常描述类或者类实例的行为及功能，其定义格式如下：

```
[修饰符] 返回值类型 方法名(形参列表){
//方法体
}
```

修饰符用于定义成员方法的属性，如 public、protected、private、static、final 等。方法的返回值类型既可以为基本数据类型，也可以是引用数据类型。

下面的例 5-1 示范了 Student 类的定义。

例 5-1

```
public class Student {
    String id;                              //学号
    String name;                            //姓名
    Date birth;                             //生日
    float[] score;                          //成绩
    public float computeAverage() {         //计算平均成绩
        float sum = 0.f;
        for(float s:score)
            sum+=s;
        return sum/score.length;
    }
    public String toString() {              //对象的字符串表示
        return id+"\t"+name;
    }
}
```

类 Student 用于描述学生信息，包括 4 个成员变量，其中，成员变量 birth、score 是引用类型变量；定义了两个公有(public)方法：public float computeAverage()和 public String toString()。

3. 成员变量与局部变量

在 Java 语言中，根据定义变量位置的不同，可以将变量分成两大类：成员变量与局部变量。成员变量、局部变量的命名必须遵守 Java 语言关于标识符的命名规则，通常由多个有意义的单词组成，首单词小写，后面每个单词首字母大写。

局部变量根据定义形式的不同，又可以分为形参变量(方法的形参变量)、方法局部变量(在方法体内定义的局部变量)及代码块局部变量(代码块中定义的局部变量)。

成员变量与局部变量在 Java 程序中存在如下差异。

(1) 在类中的位置不同。

成员变量：定义位于类体中，方法的外面；

局部变量：在方法或者代码块中，或者方法的声明上(即在参数列表中)。

(2) 生命周期不同。

成员变量：随着对象的创建而存在，随着对象的消失而消失，其作用域为类中的所有

方法。

局部变量：随着方法的调用或者代码块的执行而存在，随着方法的调用完毕或者代码块的执行完毕而消失，局部变量在其所定义的方法或者代码块中有效。

(3) 初始值。

成员变量：无须显式初始化，有默认初始值，如表5-1所示。

表5-1 成员变量默认的初始值

数据类型	byte、short、int、long	float、double	char	boolean	类、接口等引用类型
初始化值	0	0.0	'\u0000'	false	null

局部变量：没有默认初始值，使用之前必须初始化。

Student类的id、name、score、birth都是成员变量，可以被类的所有成员方法访问；成员方法computeAverage()中定义的变量sum、s是局部变量，sum在computeAverage()的方法体中有效，局部变量s仅在foreach循环语句中可以被访问。局部变量没有初始值，需要先初始化，然后才能使用。

注意：Java语言允许局部变量与成员变量同名。如果局部变量与成员变量同名，那么方法内的变量名默认表示局部变量；如果需要在方法中访问被隐藏的成员变量，则需要使用this关键字。

例如：

```
public void setName(String name) {
      this.name = name;
}
```

方法setName()中赋值运算的左侧通过this关键字访问成员变量name；而赋值运算的右操作数name则是同名的局部变量(形参变量)。

5.3.3 对象

对象是类的实例，拥有具体的状态与行为。例如Student是类，描述同类对象共同的状态及行为，而对象是指一个具体的Student个体，例如姓名是“李白”的Student对象。对象是动态的，每个对象都拥有一个从创建、运行到消亡的动态过程。同时，对象也要占用内存空间，存储对象的状态数据，即成员变量。

1. 声明对象

对象必须先声明再使用，声明对象的一般格式为：

```
类名 对象名;
```

例如：“Student stu;”通过声明定义对象变量stu，表明该变量可以指向一个Student类型的对象。

2. 创建对象

创建对象就是通过 new 运算符调用类的构造方法(构造方法的详细内容见 5.3.5 节),创建类的实例对象。一般格式为：

```
对象名 = new 类名([参数列表]);
```

例如："stu = new Student();"创建了一个学生对象,并将其引用赋值给变量 stu。可以将对象的声明与创建合并成语句,如"Student stu = new Student();"。

3. 使用对象

创建对象之后,就可以通过以下两种方式使用对象。

(1) 访问对象的成员变量。

```
对象变量.成员变量
```

(2) 调用对象的方法。

```
对象变量.成员方法([参数列表])
```

4. 对象的运算

Java 语言支持对象的"==="与"!="关系运算,用于比较两个对象变量是否指向同一内存地址。

5. 对象的内存模型

本节分析对象在创建过程中的内存模型。上述代码行"Student stu = new Student();"在运行的过程中不仅生成一个 Student 对象,而且会同时产生一个 Student 类型的变量 stu。那么生成的对象和变量 stu 在内存中是如何存储的呢?

Java 程序在运行时,需要在内存中分配空间。为了提高运算效率,Java 程序对内存空间进行了不同的划分,其中就包括堆内存与栈内存。栈内存主要存放一些基本类型的变量(int、short、long、byte、float、double、boolean、char)和对象变量;而堆内存则用来存储 Java 中的对象。

(1) 创建一个对象的内存模型。

代码行：

```
Student stu1 = new Student();
stu1.name = "李白";
stu1.id = "17010";
```

在内存中的存储示意图如图 5-1 所示。

图 5-1 指示了栈内存与堆内存空间。通过运算符 new 实例化 Student 对象,系统在堆内存中分配一段空间,用于存储 Student 对象的多个成员变量。同时,声明的对象变量 stu1 被存储在栈内存中,它指向刚刚创建的存储在堆内存中的 Student 对象,也就是变量

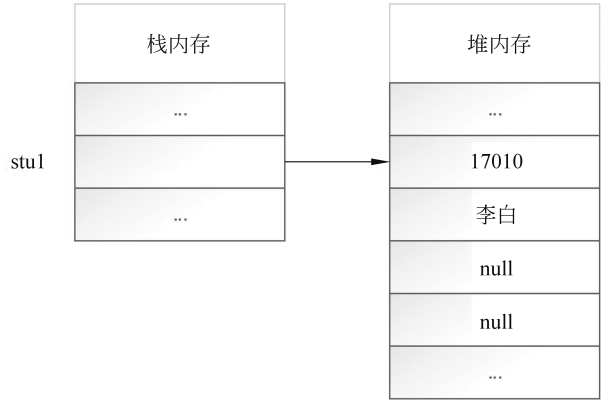

图 5-1　一个对象的内存模型

stu1 中存储了 Student 对象的引用。

（2）创建多个对象的内存模型

代码行：

```
Student stu1 = new Student();
stu1.name = "李白";
stu1.id = "17010";
Student stu2 = new Student();
stu2.name = "李白";
stu2.id = "17010";
```

在内存中的存储示意图如图 5-2 所示。

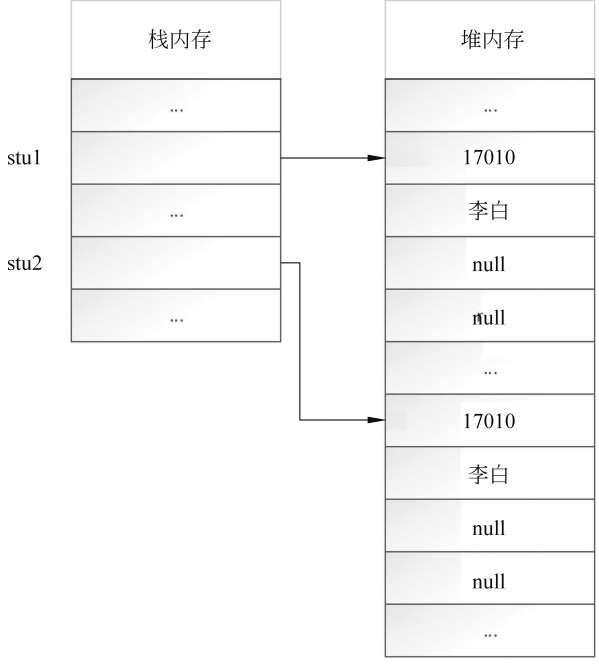

图 5-2　多个对象的内存模型

通过运算符 new 先后在堆内存中实例化两个 Student 对象。与对象相对应的对象变量 stu1 与 stu2 被存储在栈内存中。虽然堆内存保存的两个 Student 对象属性相同，但是其位于不同的内存空间，也就是具有不同的引用。

若执行代码行“stu1＝null;”，这时，stu1 变量指向的存放于堆内存中的对象就变成了“垃圾”，Java 的垃圾回收机制将回收该对象，释放该对象所占用的堆内存空间。

5.3.4 方法的重载

Java 语言允许在一个类中定义多个同名的方法，但是方法的参数必须不同。同名的方法被称为重载方法。

重载方法通过不同的参数个数或参数类型进行区别，返回值类型及方法的修饰符定义都与重载无关。

例 5-2 示范了方法的重载。

例 5-2

```
public class Calculator{
    public int add(int a, int b) {
        System.out.println("参数是 int 类型");
        return a+b;
    }
    public float add(float a, float b) {
        System.out.println("参数是 float 类型");
        return a+b;
    }
    public double add(double a, double b) {
        System.out.println("参数是 double 类型");
        return a+b;
    }
    public byte add(byte a, byte b) {
        System.out.println("参数是 byte 类型");
        return (byte)(a+b);
    }
    public staticvoid main(String[] args) {
        Calculator cal =  new Calculator();
        //传递整数参数,所以调用 add(int,int)方法
        System.out.println(cal.add(1, 3));
    }
}
```

5.3.5 构造方法

在类的定义中，有一类特殊的方法称为构造方法。构造方法与类同名，没有返回值，

可以通过构造方法实现实例对象的初始化操作。

1. 构造方法的定义

一般格式如下：

```
class 类名{
    public 构造方法名([参数列表]){
        方法体
    }}
```

2. 构造方法的调用

new 运算符将自动调用构造方法，实现对象实例的初始化操作。例如：

```
public class Student {
    String id;              //学号
    String name;            //姓名
    public Student(String strId, String strName) {      //定义构造方法
        id = strId;
        name = strName;
    }
    public static void main(String[] args) {
        Student stu = new Student("17010","李白");      //自动调用构造方法
        System.out.println(stu.name);                   //输出"李白"
    }
}
```

3. 默认的构造方法

如果一个类中没有自定义构造方法，则系统会自动生成一个无参数且方法体为空的构造方法，称为默认的构造方法；反之，如果类中已经显式定义了构造方法，则系统不会再自动生成默认的构造方法。

4. 构造方法的重载

构造方法也可以重载，即提供多种逻辑以实现类实例对象的初始化。例如：

```
public class Student {
    String id;                          //学号
    String name;                        //姓名
    public Student() {                  //定义默认构造方法
    }
    public Student(String strId) {      //定义一个 String 参数的构造方法
        id = strId;
    }
```

```
    public Student(String strId, String strName) {
                                                    //定义两个 String 参数的构造方法
        id = strId;
        name = strName;
    }
    public static void main(String[] args) {
        Student stu1 = new Student();          //调用默认的构造方法创建 Student 对象
        System.out.println(stu1.name);
        Student stu2 = new Student("17010");
                                              //调用单参数的构造方法创建 Student 对象
        System.out.println(stu2.name);
        Student stu3 = new Student("17010","李白");
                                                    //调用两个参数的构造方法创建对象
        System.out.println(stu3.name);
    }
}
```

注意：Student 类不能再定义。

```
public Student(String strName) {
    name = strName;
}
```

编译时，系统提示“Duplicate method Student(String) in type Student”，即出现方法重复定义的错误。

5.3.6 this 关键字

Java 语言提供了 this 关键字，代表当前对象的引用。this 关键字有以下 3 种典型应用。

1. 在类的实例方法中使用 this 访问本类的其他成员

例如：

```
public class Monkey {
    float height;              //身长
    float weight;              //体重
    Container container;       //装载猴子的笼子
    public String toString() {
        return "Monkey height:"+this.height+";weight:"+this.weight;
    }
    void print() {
        System.out.println("当前对象信息:"+this.toString());
    }
```

```
}
class Container{
    Monkey monkey;
    Container(Monkey monkey){
        this.monkey = monkey;
    }
}
```

上述类 Monkey 的实例方法 print 中使用了 this 关键字调用了另一个实例方法 toString();在实例方法 toString()中,又通过关键字 this 访问了当前对象的 height 实例变量。通常情况下,以上两处可以省略"this.",默认指代当前对象的引用。但是,在下面构造方法中:

```
Monkey(float height, floatweight) {
        this.height = height;
        this.weight = weight;
}
```

由于实例变量(height)与局部变量同名,所以必须使用 this 关键字区别实例变量。

2. 在类的构造方法中使用 this 调用本类的另一个构造方法

通过 this 关键字调用构造方法只能用在构造方法中,且它必须是第一行语句。例如,在 Monkey 类中增加无参的构造方法:

```
Monkey() {
        this(50.0f,10.0f);
}
```

上述构造方法中使用 this(float,float),实际上是调用另一个构造方法 Monkey(float,float)实现 Monkey 对象的初始化操作。

3. 在类的实例方法中 this 指代当前对象的引用

例如,在 Monkey 类中定义实例方法:

```
void loadContainer() {
    container= new Container(this);      //将当前 Monkey 对象装到 container 中
}
```

通过 this 关键字指代当前对象,将当前的 Monkey 对象装到 container 对象中。

5.3.7 static 关键字

在 Java 类的定义中,static 关键字可用于声明成员变量、方法与初始化块(关于初始化块的详细内容参见 5.3.8 节)。