

第 5 章 进一步讨论对象和类

5.1 详细说明类是如何定义的,解释类的特性及它的几个要素。

解: Java 程序设计就是定义类的过程,Java 程序中的所有代码都包含在类中。类可以看作是数据的集合及操作这些数据所需的方法的整合。

Java 中的类分两种,一种是系统预定义的类,这些类组成 Java 类库。Java 类库是一组由软件供应商编写好的程序模块,完成常用的基本功能和任务,可由程序编写人员直接调用。正是因为有了这些类库,程序员才有很好的辅助工具,不必将精力浪费在一些简单常见的功能实现上。基本类库提供的这些功能,使得程序员站在了一个较高的起点上,可以把主要精力关注在更加复杂的工作上。这些定义好的类根据实现功能的不同,划分成不同的集合,每个集合称为一个包。Sun 公司提供的 JDK 中共有 43 个大包。

除去系统预定义的类之外,还有一种是用户程序自己定义的类,当然这其中又包括其他程序员定义的类和自己定义的类。这些类都显式或隐式地派生于 Java 中某个预定义的类。不论是预定义的类,还是程序员自己定义的类,每个类中一般都包含属性和方法。属性即是数据,属性值表明一个对象的状态;方法决定类有哪些可利用的手段,即可通过哪些函数来操作这些数据。

类的具体格式如下:

```
修饰符 class 类名 [extends 父类名] {  
    修饰符 类型 成员变量 1;  
    修饰符 类型 成员变量 2;  
    .....  
    修饰符 类型 成员方法 1(参数列表) {  
        类型 局部变量;  
        方法体  
    }  
    修饰符 类型 成员方法 2(参数列表) {  
        类型 局部变量;  
        方法体  
    }  
    .....  
}
```

类定义的第一行是类头,class 关键字表明这里定义的是一个类。class 前的修饰符允许有多个,用来限定所定义的类的使用方式。

类名是用户为该类所起的名字,它应该是一个合法的标识符,并尽量遵从命名约定。

extends 是关键字。如果所定义的类是从某一个父类派生而来,那么,父类的名字应写在 extends 之后。如果不写的话,则隐式表明继承自 Object 类。Java 不允许多重继承,因此如果有父类的话,只能有一个父类。

类头后面的部分称为类体,类体用一对大括号括起来,含有两部分,一部分是数据

成员变量,另一部分是成员方法。数据成员变量可以含有多个,这是类的静态属性,表明类的实例目前所处的状态。类的不同实例对应各自不同的属性值,因此有些属性值可用来标识不同的实例。成员变量前面的类型是该变量的类型。成员方法也可以有多个,其前面的类型是方法返回值的类型。方法对应类的行为和操作。方法体是要执行的真正语句。在方法体中还可以定义该方法内使用的局部变量,这些变量只在该方法内有效。

类可以是 public 的,表明任何对象都可以使用或扩展这个类;也可以是 friendly 的,表明它可以被同一个包中的对象使用。类还可以是 final 的,表明它不可以再有子类,与之相对的,使用 abstract 修饰的类必须要有子类。

5.2 给出 3 个类的定义:

```
class ParentClass {}  
class SubClass1 extends ParentClass {}  
class SubClass2 extends ParentClass {}
```

并分别定义 3 个对象:

```
ParentClass a=new ParentClass ();  
SubClass1 b=new SubClass1();  
SubClass2 c=new SubClass2();
```

若执行下面的语句:

```
a=b;  
b=a;  
b=(SubClass1) c;
```

会有什么结果? 分别从下面的选项中选择正确的答案。

- (1) 编译时出错。
- (2) 编译时正确,但执行时出错。
- (3) 执行时完全正确。

解: 3 行类定义分别定义了 3 个类,一个父类 ParentClass 及它的两个子类 SubClass1 和 SubClass2。后面的 3 行则分别为每个类说明了一个实例,其中,a 是 ParentClass 类的实例,b 是 SubClass1 类的实例,c 是 SubClass2 类的实例。因为 SubClass1 和 SubClass2 都是派生于 ParentClass 的子类,所以 b 和 c 也同时是 ParentClass 类的实例。

Java 中允许使用对象之父类类型的一个变量指示该对象,称为转换对象(casting)。关于转换对象的使用,遵从对象引用的赋值兼容原则。所谓对象引用的赋值兼容原则是指允许把子类的实例赋给父类的引用,但不允许把父类的实例赋给子类的引用。实际编程时,可以使用 instanceof 运算符来判明一个引用指向的是哪个类的实例。如果父类的引用指向的是子类实例,就可以转换该引用,恢复对象的全部功能。

本题中,可以进行下面的测试:

```
boolean tagb1=b instanceof ParentClass;  
boolean tagc1=c instanceof ParentClass;
```

tagb1 和 tagc1 的值都是 true,表明 b 和 c 是子类实例的同时,也是父类的实例。反过来,

父类的实例不是子类的实例,例如下面的测试:

```
boolean taga2=a instanceof SubClass1;
boolean taga3=a instanceof SubClass2;
```

taga2 和 taga3 的值都是 false。b 和 c 是不同子类的实例,因此如下测试:

```
boolean tagb3=b instanceof SubClass2;
boolean tagc2=c instanceof SubClass1;
```

将出现编译错误。下面针对题目中的 3 条语句分别进行测试。

(1) 执行 `a=b`; 时, `a` 指向父类的实例, `b` 指向子类的实例。由于是将子类实例赋给父类实例, 因此编译及执行都正确。该语句执行子类中的方法, 如果子类中没有重写父类中的方法, 则执行父类中的方法。例如, 下面程序的父类和子类中都定义了 `value` 成员和 `getValue()` 方法, 将子类的实例赋给父类引用后, 此时 `a` 的值是子类的实例, 再执行语句:

```
taga2=a instanceof SubClass1;
```

则 `taga2` 的值应为 `true`。在给 `a` 分配的内存中既包括子类中 `value` 的值, 也含有父类中 `value` 的值。调用 `a.getValue()` 方法时, 先在子类中查找这个方法是否存在, 如果有, 则返回子类中 `value` 的值 1; 若没有, 则查找父类中的同名方法, 并返回父类中的值 0。

完整的测试代码如下:

```
import java.util.*;

class ParentClass
{   public ParentClass()           //父类的构造方法
    {   value=0;
    }
    public int getValue()           //父类的求值方法,返回父类中 value 的值
    {   return value;
    }
    public void setValue(int y)     //给父类的属性 value 赋值
    {   value=y;
    }

    private int value;
}

class SubClass1 extends ParentClass
{   public SubClass1()             //子类 1 的构造方法, value 值为 1
    {   value=1;
    }
    public int getValue()           //子类的同名求值方法,返回子类 1 中 value 的值
    {   return value;
    }
    public int getClassValue1()     //子类的特殊求值方法,返回 classvalue1 的值
    {   return classvalue1;
    }
    private int value;
    private int classvalue1=11;
}
```



源代码

```

class SubClass2 extends ParentClass
{
    public SubClass2()                //子类 2 的构造方法,value 值为 2
    {
        value=2;
    }
    public int getValue()              //子类的同名求值方法,返回子类 2 中 value 的值
    {
        return value;
    }
    public int getClassValue2()        //子类的特殊求值方法,返回 classvalue2 的值
    {
        return classvalue2;
    }
    private int value;
    private int classvalue2=22;
}

public class Test2
{
    public static void main(String[] args)
    {
        ParentClass a=new ParentClass ();        //父类实例
        SubClass1 b=new SubClass1();              //子类 1 的实例
        SubClass2 c=new SubClass2();              //子类 2 的实例

        a=b;                                       //a 指向子类 1 的实例
        System.out.println("a="+a.getValue());   //返回子类 1 中的属性值
    }
}

```

程序的执行结果如下：

```
a=1
```

(2) 执行 `b=a;` 时,由于是将父类的实例赋给子类的变量,因此会出现编译错误,错误类型是变量类型不匹配。

(3) 执行 `b=(SubClass1)c;` 时,由于 `b` 和 `c` 是不同类的实例,因此也会出现编译错误,错误类型是变量类型不能转换。

【拓展思考】

(1) `new` 运算符执行什么动作?

解: `new` 运算符创建指定类的一个新实例(对象)。然后调用类的构造方法设置新生成的对象。

(2) `null` 引用的作用是什么?

解: `null` 引用是不指向任何对象的引用。用保留字 `null` 来检查空引用,以避免对空引用的访问。

(3) 什么是别名?

解: 如果两个引用指向同一个对象,则它们互为别名。通过一个引用改变对象的状态,也就改变了另一个引用指向的对象,因为实际上只有一个对象。仅当对象再没有被引用指向时,才被垃圾收集所标记。

5.3 什么是抽象类？它如何定义？下面的哪些定义是正确的？

(1)

```
class alarmclock {  
    abstract void alarm();  
}
```

(2)

```
abstract alarmclock {  
    abstract void alarm();  
}
```

(3)

```
class abstract alarmclock {  
    abstract void alarm();  
}
```

(4)

```
abstract class alarmclock {  
    abstract void alarm();  
}
```

(5)

```
abstract class alarmclock {  
    abstract void alarm() {  
        System.out.println("alarm!")  
    };  
}
```

解：如果一个方法只有方法的声明，而没有方法的实现，则称为抽象方法（abstract method）。含有抽象方法的类通常称为抽象类（abstract class）。在 Java 中可以通过 abstract 关键字把一个类定义为抽象类，每一个未被定义具体实现的抽象方法也应标记为 abstract。抽象类是表示一般概念的类。在抽象类中可定义公共特征及方法签名，然后由其子类来继承它们。

在抽象类中可以包括被它的所有子类共享的公共行为，也包括被它的所有子类共享的公共属性。因为抽象类中含有抽象方法，所以不能将抽象类作为模板来创建对象，必须生成抽象类的一个非抽象的子类后才能创建实例。这是因为一个实例的任何方法都必须已被具体实现了。抽象类可以包含常规类能够包含的任何元素，当然也包括构造方法，因为子类可能需要继承这种方法。抽象类中当然包含抽象方法，这种方法只有方法的声明，而没有方法的实现。这些方法将在抽象类的子类中被具体实现。只有实现了所有抽象方法的子类才能创建对应的实例。除了抽象方法，抽象类中当然也可以包含非抽象方法，反之，不能在非抽象的类中声明抽象方法。也就是说，只有抽象类才能具有抽象方法。

根据抽象方法和抽象类的定义和规则，再来分析题目中的 5 个语句说明。第 1 个示例中，alarm() 方法被定义为抽象方法，那么方法所在的类也必须被声明为抽象类型；第 2 个示例中，虽然方法和类都被说明是抽象的，但缺少 class 关键字；第 3 个示例中，各关键

字的次序不正确,abstract 关键字要放在 class 之前;第 5 个示例中,虽然 alarm()方法被说明为抽象方法,但方法体不为空,这显然有矛盾。只有第 4 个示例的说明才是正确的。

【拓展思考】

父类的所有成员都能被子类继承吗?请解释理由。

解: 如果父类的成员是私有可见的,则不能被子类继承。这意味着在子类中不能通过名字来直接引用父类的私有成员。但对子类来说,这样的成员确实存在,可以间接引用。

5.4 什么叫方法重载?什么叫方法重写?它们之间的区别是什么?

解: 同一个类中,可以定义同名的多个方法。它们的不同之处在于参数列表不同,这其中包括参数的个数不同或是对应的参数类型不完全相同。这就是方法的重载(overload)。当对该类实例调用相应的方法时,系统将依据所带参数的个数及类型从同名的多个方法中来选择参数列表满足要求的方法。不只如此,在不同的类中也可以定义有相同方法名的方法。

使用类的继承关系,可以从已有的类产生一个新类,在原有特性基础上,增加新的特性。原类及新类分别称为父类及子类。如果父类中原有的方法不能满足子类的要求,可以在子类中对父类的方法重新编写代码。这称为方法重写(override),也称为方法的隐藏,意思是子类中看不到父类中的实现代码。子类中定义的方法所用的名字、返回类型及参数表和父类中方法使用的完全一样,从逻辑上看就是子类中的成员方法隐藏了父类中的同名方法。

从对方法重载和方法重写的分析可以看出,它们之间的区别主要有以下几点。

- 方法重载时参数列表必须不同,才能使系统区别出到底调用哪个方法,而方法重写时参数列表可以相同。
- 方法重载时,方法的返回类型可能不同,也可能相同;而方法重写时,子类中方法的返回类型和父类中同名方法的返回类型完全一样。
- 方法重载多出现在同一个类中,方法重写必须是在父子类中。

【拓展思考】

(1) 什么是多态?

解: 多态是指引用变量在不同时刻指向不同类型对象的一种能力。通过这样的引用,调用的方法可以在不同的时刻,根据对象引用的类型与不同的方法绑定。

(2) 继承如何支持多态?

解: 在 Java 中,使用父类声明的引用变量可以指向子类的对象。两个类包含有相同签名的方法时,父类引用是多态的。

(3) 与多态相关的重写如何实现?

解: 当子类重写父类方法的定义时,这个方法就有了两个版本。用多态引用调用这个方法时,调用的方法版本取决于所用对象的类型,而非引用变量的类型。

(4) 如何使用接口完成多态?

解: 接口名可作为引用类型使用。这样的引用变量可指向实现该接口的任一类的任一对象。因为所有的类实现同一个接口,有公共的签名,所以可以动态绑定。

(5) 单重继承和多重继承之间的差别是什么?

解：在单重继承中，只能从一个父类派生子类；而在多重继承中，一个类可从多个父类中派生，并继承每个父类的特性。多重继承的问题是，必须解决当两个或多个父类中有同名的属性或方法时引起的冲突。Java 只支持单重继承。

5.5 什么是 null 引用？

解：null 引用是不指向任何对象的引用。用保留字 null 来检查空引用，以避免对空引用的访问。

在 Java 中，当执行 new 为一个对象分配内存时，Java 自动初始化所分配的内存空间。对于引用，即对象类型的任何变量，使用一个特殊的值 null 进行初始化，它表示引用不指向任何对象。运行过程中，一旦系统发现使用了这样一个引用时，可以立即停止进一步的访问，不会带来任何危险。

5.6 this 关键字和 super 关键字在成员方法中的特殊作用是什么？

解：在 Java 中，this 引用总是指向当前对象，即 this 引用所在的对象。如果在类的成员方法中访问类的成员变量，可以使用 this 关键字指明要操作的对象。在构造方法中，this 还有一个用法，即作为构造方法的第一个语句，它的形式是 this(参数表)，这个构造方法会调用同一个类的另一个构造方法。

如果子类已经重写了父类中的方法，但在子类中还想使用父类中被隐藏的方法，或者子类中定义了和父类中同名的成员变量，还想使用父类中隐藏的成员变量，可以使用 super 关键字。

5.7 仿照书中的例子，构造一个类，并使其具有多个相互调用的构造方法；然后构造它的子类，在构造方法中利用 super 关键字来调用父类的构造方法。

解：在习题 2.10 中，定义了教师类及其子类。父类及子类中有多个相同的属性，对这些属性的访问方法也存在于父类及子类中。在习题 2.10 的实现中，父子类中的构造方法是独立的，它们之间没有关系。实际上，若父子类中有多个相同的属性，子类的构造方法就可以借用父类的构造方法，以简化代码。除构造方法外，子类的任何成员方法都可以借用父类同名的成员方法，并在此基础上，增加自己的特殊处理部分。

修改习题 2.10 的实现，包括修改其中定义的类及成员方法。先定义一个 SchoolTeacher 类，其中包括 3 个互相调用的构造方法。在此基础上派生 ResearchSchoolTeacher 子类，其构造方法中使用 super 来调用父类的构造方法，对父类及子类共有的属性进行赋值。除了可以在构造方法中使用 super 调用父类的构造方法外，还可以在一般的方法中调用父类的方法。例如本题中，当输出 ResearchSchoolTeacher 类实例的信息时，因前 6 个属性是与父类实例相同的属性，所以可以借用父类中的输出方法：

```
super.print(); //使用父类的输出方法
```

上述语句调用父类的输出方法，先输出前 6 个属性的值，然后再使用下面的两条语句：

```
System.out.print("The SchoolTeacher research field is: ");  
System.out.println(this.getResField()); //特殊属性的输出
```

完成对该类实例 resField 属性的输出。

程序代码实现如下：



源代码

```
import java.lang.*;

class Date //定义日期类
{
    int day;
    int month;
    int year;
    Date (int day, int month, int year)
    {
        this.day=day;
        this.month=month;
        this.year=year;
    }
    Date ()
    {
        this.day=8;
        this.month=11;
        this.year=2012;
    }
    public int getYear() //返回年
    {
        return year;
    }
    public int getMonth() //返回月
    {
        return month;
    }
    public int getDay() //返回日
    {
        return day;
    }
    public void setDate(Date SpeDate) //设置日期
    {
        year=SpeDate.getYear();
        month=SpeDate.getMonth();
        day=SpeDate.getDay();
    }
}

public class SchoolTeacher //定义教师类,这是基类
{
    private String name; //教师名字
    private boolean sex; //性别,true 表示男性,false 表示女性
    private Date birth; //教师出生日期
    private String salaryID; //教师工资号
    private String depart; //教师所属系
    private String posit; //教师职称

    String getName() //返回教师名字
    {
        return name;
    }
    void setName (String name) //记录教师名字
    {
        this.name=name;
    }
    boolean getSex() //返回教师性别
    {
        return sex;
    }
    void setSex (boolean sex) //记录教师性别
    {
        this.sex=sex;
    }
    Date getBirth() //返回教师出生日期
```

```

    {    return birth;
    }
    void setBirth (Date birth)                //记录教师出生日期
    {    this.birth=birth;
    }
    String getSalaryID()                      //返回教师工资号
    {    return salaryID;
    }
    void setSalaryID (String salaryID)        //记录教师工资号
    {    this.salaryID=salaryID;
    }
    String getDepart()                        //返回教师所属系所名
    {    return depart;
    }
    void setDepart (String depart)            //记录教师所属系所名
    {    this.depart=depart;
    }
    String getPosit()                         //返回教师职称
    {    return posit;
    }
    void setPosit (String posit)              //记录教师职称
    {    this.posit=posit;
    }

    public SchoolTeacher(String name)          //只含一个属性参数的构造方法
    {    this.name=name;
    }

    //含有 3 个属性参数的构造方法
    public SchoolTeacher(String name, boolean sex, Date birth)
    {    this(name);                          //调用只含一个参数的构造方法
      this.sex=sex;                          //对其余的两个属性赋值
      this.birth=birth;
    }

    //含有全部 6 个属性参数的构造方法
    public SchoolTeacher(String name, boolean sex, Date birth,
        String salaryid, String depart, String posit)
    {    this(name, sex, birth);              //调用含 3 个参数的构造方法
      this.salaryID=salaryid;                //对其余的 3 个属性赋值
      this.depart=depart;
      this.posit=posit;
    }
    public void print ()                      //输出教师基本信息
    {    System.out.print("The SchoolTeacher name: ");
      System.out.println(this.getName());
      System.out.print("The SchoolTeacher sex: ");
      if (this.getSex()==false)
      {    System.out.println("女");
      }
      else
      {    System.out.println("男");
      }
    }

```

```

        System.out.print("The SchoolTeacher birth: ");
        System.out.println(this.getBirth().year + "-" + this.getBirth().month
            + "-" + this.getBirth().day);
        System.out.print("The SchoolTeacher salaryid: ");
        System.out.println(this.getSalaryID());
        System.out.print("The SchoolTeacher posit: ");
        System.out.println(this.getPosit());
        System.out.print("The SchoolTeacher depart: ");
        System.out.println(this.getDepart());

    }

    public static void main (String [] args)
    {
        Date dt1=new Date(12, 2, 1985);        //创建日期实例,作为教师的出生日期
        Date dt2=new Date(2, 6, 1975);
        Date dt3=new Date(11, 8, 1964);
        Date dt4=new Date(10, 4, 1975);
        Date dt5=new Date(8, 9, 1969);

        //创建两个教师实例,一个为父类的实例,另一个为子类的实例
        SchoolTeacher t1=new SchoolTeacher("zhangsan", false,dt1, "123",
            "CS", "Professor");
        ResearchSchoolTeacher rt=new ResearchSchoolTeacher("lisi", true, dt2, "421",
            "software engineering", "associate professor", "Software");

        //分别调用各自的输出方法,输出相应的信息
        System.out.println("-----");
        t1.print();                    //输出普通教师的信息
        System.out.println("-----");
        rt.print();                    //输出研究系列教师的信息
        System.out.println("-----");
    }
}

class ResearchSchoolTeacher extends SchoolTeacher    //研究系列教师类的定义
{
    private String resField;                          //增加的研究领域属性
    String getResField()                              //返回研究领域属性
    {
        return resField;
    }
    void setResField (String resField)                 //记录研究领域属性
    {
        this.resField=resField;
    }
    public ResearchSchoolTeacher(String name, boolean sex, Date birth,
        String salaryid, String depart, String posit, String resField)
    {
        //使用父类的构造方法,对共有的 6 个属性进行赋值
        super(name, sex, birth, salaryid, depart, posit);
        this.resField=resField;                        //特殊属性的赋值
    }
    public void print()
    {
        System.out.println("One of Research SchoolTeachers' info is ");
        super.print();                                //使用父类的输出方法
    }
}

```