5.1 例题解析

例 5.1.1 详细解析 Java 中的抽象类。

【例题解析】

在 Java 语言中, abstract class 和 interface 是支持抽象类定义的两机制。在面向对象的概念中,所有的对象都是通过类来描绘的,但反过来却不是这样,并不是所有的类都是用来描绘对象的。如果一个类中没有包含足够的信息来描绘一个具体的对象,这样的类就是抽象类。

抽象类往往用来表征在对问题领域进行分析、设计中得出的抽象概念,是对一系列看上去不同但本质上相同的具体概念的抽象。例如,在进行一个图形编辑软件的开发时,会发现问题领域存在着圆、三角形这样一些具体概念,它们是不同的,但它们又都属于形状这样一个概念。形状这个概念在问题领域中是不存在的,它是一个抽象概念。正是因为抽象的概念在问题领域没有对应的具体概念,所以用于表征抽象概念的抽象类是不能够实例化的。

例 5.1.2 总结 Java 中 abstract class、interface、final 和 static 的概念。

【例题解析】

在语法层面,Java 语言对于 abstract class 和 interface 给出了不同的定义方式,下面以定义一个名为 Demo 的抽象类为例来说明这种不同。使用 abstract class 定义 Demo 抽象类的方式如下。

```
abstract class Demo{
    abstract void method1();
    abstract void method2();
    :
}
```

使用 interface 定义 Demo 抽象类的方式如下。

```
interface Demo{
    void method1();
}
```

```
void method2();  
:  
}
```

首先,在 abstract class 方式中,Demo 可以有自己的数据成员,也可以有非 abstract 的成员方法;而在 interface 方式的实现中,Demo 只能有静态的不能被修改的数据成员,也就是必须是 static final 的。不过,在 interface 中一般不定义数据成员,所有的成员方法都是 abstract 的。从某种意义上说,interface 是一种特殊形式的 abstract class。

abstract class 在 Java 语言中表示的是一种继承关系,一个类只能使用一次继承关系。但是,一个类却可以实现多个 interface。这也是 Java 语言的设计者在考虑 Java 对于多重继承的支持方面的一种折中处理。

其次,在 abstract class 的定义中,我们可以赋予方法默认行为。但是在 interface 的定义中,方法却不能拥有默认行为。

abstract class 和 interface 所反映出的设计理念不同,abstract class 表示的是 is-a 关系,interface 表示的是 like-a 关系。

实现抽象类和接口的类必须实现其中的所有方法。抽象类中可以有非抽象方法,接口中则不能有实现方法。

关于抽象类 abstract class:

- (1) 只要是有一个或一个以上抽象方法的类必须用 abstract 声明为抽象类。
- (2) 抽象类中可以有具体的实现方法。
- (3) 抽象类中可以没有抽象方法。
- (4) 抽象类中的抽象方法必须被它的子类实现,如果子类没有实现,则该子类继续为抽象类。
- (5) 抽象类不能被实例化,但可以由抽象父类指向的子类实例来调用抽象父类中的具体实现方法,通常作为一种默认行为。
- (6) 要使用抽象类中的方法,必须有一个子类继承于这个抽象类并实现抽象类中的抽象方法,通过子类的实例去调用。

关于接口 interface:

- (1) 接口中可以有成员变量,且接口中的成员变量必须初始化。
- (2) 接口中的成员方法只能是方法原型,不能有方法主体。
- (3) 接口的成员变量和成员方法只能是 public(或默认不写,效果一样,都表示 public)。
- (4) 实现接口的类必须全部实现接口中的方法。

关于关键字 final:

- (1) final 可用于修饰成员变量、非抽象类(不能与 abstract 同时出现)、非抽象的成员方法以及方法参数。
- (2) final 方法不能被子类的方法重写,但可以被继承。
- (3) final 类表示该类不能被继承,没有子类;final 类中的方法也无法被继承。
- (4) final 变量表示常量,只能赋值一次,赋值后不能被修改。final 变量必须初始化。

(5) final 不能用于修饰构造方法。

(6) 对于 final 参数,只能使用该参数,不能修改该参数的值。

关于关键字 static:

(1) static 可以修饰成员变量和成员方法,但不能修饰类以及构造方法。

(2) 被 static 修饰的成员变量和成员方法独立于该类的任何对象。也就是说,它不依赖类特定的实例,被类的所有实例共享。

(3) static 变量和 static 方法一般通过类名直接访问,但也可以通过类的实例来访问(不推荐这种访问方式)。

(4) static 变量和 static 方法同样适用 Java 修饰符。用 public 修饰的 static 变量和 static 方法在任何地方都可以通过类名直接访问,用 private 修饰的 static 变量和 static 方法只能在声明的本类方法及静态块中访问,但不能用 this 访问,因为 this 属于非静态变量。

关于 static 和 final 同时使用:

(1) static final 用来修饰成员变量和成员方法,可简单理解为“全局常量”。

(2) 对于变量,表示一旦赋值就不可修改,并且通过类名可以访问。

(3) 对于方法,表示不可覆盖,并且可以通过类名直接访问。

例 5.1.3 分析下列程序片段,指出程序中的错误。

```
(1) abstract class Name {  
    private String name;  
    public abstract boolean isStupidName(String name) {}  
}
```

【例题解析】

abstract method 必须以分号结尾,且不带花括号。

```
(2) public class Something {  
    void doSomething () {  
        private String s="";  
        int l=s.length();  
    }  
}
```

【例题解析】

局部变量前不能放置任何修饰符(private、public 和 protected)。final 可以用来修饰局部变量,但它是非修饰符。

```
(3) abstract class Something {  
    private abstract String doSomething ();  
}
```

【例题解析】

abstract 方法不能以 private 修饰。abstract 方法是让子类实现具体细节的,不可以用 private 把 abstract 方法封锁起来。同理,abstract 方法前也不能加 final。

```
(4) public class Something {  
    public int addOne(final int x) {  
        return ++x;  
    }  
}
```

【例题解析】

int x 被修饰成 final 意味着 x 不能在 addOne 方法中被修改。

5.2 习题解答

1. 接口与抽象类有哪些异同点？

参考答案：

在面向对象的概念中,我们知道所有的对象都是通过类来描绘的,但是反过来却不是这样。并不是所有的类都是用来描绘对象的,如果一个类中没有包含足够的信息来描绘一个具体的对象,这样的类就是抽象类。抽象类往往用来表征我们在对问题领域进行分析、设计时得出的抽象概念,是对一系列看上去不同、但是本质上相同的具体概念的抽象。正是因为抽象的概念在问题领域没有对应的具体概念,所以用以表征抽象概念的抽象类是不能够实例化的。

接口与抽象类的主要异同点如下。

(1) 接口定义了一组特定功能的对外接口与规范,而并不真正实现这种功能,功能的实现留给实现这一接口的各个类来完成。抽象类一般作为公共的父类为子类的扩展提供基础,这里的扩展包括属性上的和行为上的。而接口一般来说不考虑属性,只考虑方法,使得子类可以自由地填补或扩展接口所定义的方法。抽象类表示的是 is-a 关系,接口着重表示的是 can-do 关系。

(2) 抽象类在 Java 语言中表示的是一种继承关系,一个类只能使用一次继承。但是,一个类却可以实现多个接口,接口可以解决多重继承问题。

(3) 接口是抽象方法和常量值的定义的集合,从本质上讲,它是一种只包含常量与抽象方法的特殊的抽象类,这种抽象类中只包含常量和方法的定义,而没有变量和方法的实现。接口里面不能有私有的方法或变量,接口中的所有常量必须是 public static final 的,且必须给其初值(接口的实现类中不能重新定义,也不能改变其值)。接口中的方法必须是 public abstract 的,这是系统默认的,在定义接口时写不写修饰符都是一样的。抽象类中可以有私有方法或私有变量,抽象类中的变量默认是友元型,其值可以在子类中重新定义,也可以重新赋值。

(4) 实现抽象类和接口的类必须实现其中的所有方法。在抽象类中可以有自己的数据成员,也可以有非 abstract 的成员方法。而在接口中,只能够有静态的不能被修改的数据成员,所有的成员方法都是 abstract 的。实现接口时一定要实现接口中定义的所有方法,而实现抽象类时可以有选择地重写需要用到方法。在一般的应用中,顶级的是接口,然后是抽象类实现接口,最后才到具体类实现。

2. 区分接口与抽象类分别在什么场合使用。

参考答案:

如果预计要创建类的多个版本,则创建抽象类,因为抽象类提供简单的方法来控制类版本。如果创建的功能将在大范围的异类对象间使用,则使用接口;如果要设计小而简练的功能块,则使用接口;如果要设计大的功能单元,则使用抽象类。如果要向类的所有子类提供通用的已实现功能,则使用抽象类,抽象类主要用于关系密切的对象;而接口适合为不相关的类提供通用功能。接口多定义对象的行为;抽象类多定义对象的属性。

3. 一个类如何实现接口? 实现某接口的类是否一定要重载该接口中的所有抽象方法?

参考答案:

一个类使用关键字 implements 实现某接口。实现某接口的类如果不是抽象类,则需要通过重载来实现该接口中的所有抽象方法;如果这个类是抽象类,则它可以不必实现该接口中的所有抽象方法。

4. 对于以下程序,运行 java StaticTest 后得到的输出结果是什么?

```
public class StaticTest {
    static {
        System.out.println("Hi there");
    }
    public void print() {
        System.out.println("Hello");
    }
    public static void main(String args []) {
        StaticTest st1=new StaticTest();
        st1.print();
        StaticTest st2=new StaticTest();
        st2.print();
    }
}
```

参考答案:

```
Hi there
Hello
Hello
```

5. 编写程序,要求创建一个抽象类 Father,其中有姓名、身高、体重等属性及爱好(唱歌)等方法。创建子类 Son 类继承 Father 类,并且增加性格这个属性,改写父类的方法(爱好)。

参考答案:

```
public class test {
    public static void main(String args[]) {
        Son son=new Son("乖儿子",1.78f,61f, "篮球");
    }
}
```

```
        son.showInfo();
        son.singsong();
    }
}
abstract class Father {
    float high, weight;
    protected String name;
    Father(String name, float high, float weight) {
        this.name=name;
        this.high=high;
        this.weight=weight;
    }
    abstract void singsong();
    abstract void showInfo();
}
class Son extends Father {
    String specialty;
    Son(String name, float high, float weight, String specialty) {
        super(name, high, weight);
        this.specialty=specialty;
    }
    void singsong() {
        System.out.println(name+"is singging loudly!");
    }
    void showInfo() {
        System.out.println("姓名:"+name+";身高:"+high+";体重:"+weight+";爱好:"+
specialty);
    }
}
```