第5章

继承与多态

本章学习要点

- 全面掌握 Java 中面向对象的基本特征的实现
- 掌握在 Java 中如何使用继承性来达到软件的重用
- 深入掌握继承过程中域的隐藏和方法的覆盖技巧
- 深入掌握抽象类和抽象方法的定义
- 深入掌握接口的定义和使用技巧

5.1 基本知识点

5.1.1 继承

继承最大的好处是只需花较少的工夫就能从已经存在的类中扩展出一个新类，而这个新类具有其父类的功能。

1. 继承的概念

继承是一个类与其他类相关联的机制，它使一个类能具有其他类的特征，被继承的类称为超类或父类，继承出来的类称为子类或派生类。继承用关键字 extends 表示，其语法如下：

```
[modifier] class SubClassName extends ClassName
{ /* 类的实现 */ }
```

子类自动拥有超类的所有非私有成员方法和成员变量，它可用关键字 super 访问超类的成员。此外，在子类的构造方法的第 1 行可以使用 super(参数)来引用超类的构造方法。

2. 继承的层次性

图 5.1 显示了 7 个相关类的可能构成层次。

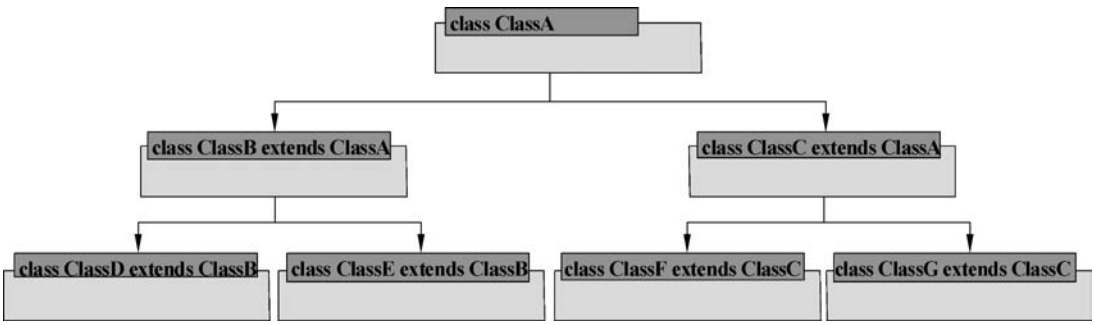

图 5.1 7 个相关类的可能构成层次

继承性允许子类使用祖先的非私有成员变量和成员方法，这种特性的好处是可以减少代码，易于维护，修改祖先类的代码，与它相关的派生类也同时被修改。

3. 继承的使用

在编写一个类的时候，如果一个类“有什么”，意味着该类需要一个成员变量；要“做什么”隐含着该类需要增加一个成员方法。而且，如果已经有一个相近的类，就可以从该类中扩展出所需要的新类。但要注意，子类所需的成员在超类中必须是非私有的，因为无法继承私有成员。另一个解决办法是在超类中使用公有的 set/get 方法访问其私有成员变量。

在使用继承手段时，关键字 extends 暗示子类比超类更加特殊，实际上子类是超类的一种特殊类别。例如，桌子比课桌更普通，使用面向对象的技术，课桌应从桌子中扩展而来。

在创建一个类层次的时候，应当先抽象出超类，再通过超类衍生出子类。

4. 继承与构造方法

当一个类被实例化的时候，它的构造方法被自动调用。当一个子类被实例化的时候，虽然没有加入任何调用构造方法的语句，但子类的构造方法和超类的构造方法都被自动地调用。

当子类被调用的时候，隐含地调用超类的构造方法，此时没有为构造方法输入任何参数，除非在子类的第 1 行明确地调用超类的构造方法。如果超类没有无参数的构造方法，子类又不明确地调用超类的构造方法，编译器将无法编译子类。

在涉及构造方法的继承时有一个要注意的编程技巧，就是避免隐蔽地调用超类的构造方法。用户可使用下面的两种办法解决：

(1) 每个子类都应当明确地调用超类的一个构造方法，而且必须在第 1 行调用。

(2) 为每个可能成为超类的类提供一个无参构造方法，如果还有其他需要，则重载构造方法。

5.1.2 覆盖

有时超类的方法和变量不一定适合子类，在这种情况下，子类可以对这些变量和方法进行重新定义，即覆盖原来的定义。

如果在子类中重新定义了一个与它的超类完全相同的方法，子类的方法将覆盖超类中的同名方法，此时超类中的该方法仍可在子类中使用，但要加上关键字 super 和圆点操作符。

如果在子类中重新定义了一个与它的超类完全相同的成员变量，子类的成员变量将覆盖超类的成员变量，此时超类中的该成员变量仍可在子类中使用，还是使用关键字 super 和圆点操作符。

覆盖可以使超类中的成员方法被重新定义，这将很容易增强一个类的功能。

覆盖成员方法比重载方法更普遍，但是要尽量避免覆盖成员变量，通常的原则如下：

(1) 如果一个超类中的方法不适合子类，可以在子类中使用与超类中一样的方法头部进行覆盖，如果继承过来的方法还能使用，可以通过使用 super 关键字避免代码重复。

(2) 如果重定义的方法的参数列表与超类中该方法的列表不一致，那么应该在超类中重载而不是在子类中覆盖。

(3) 覆盖一个类的基本功能可能会造成一些问题，所以在覆盖前用户一定要了解继承

来的所有方法。

(4) 尽量避免覆盖成员变量,否则会使代码难以理解。如果有很多变量需要覆盖,则意味着不应该从该超类扩展出该子类。

5.1.3 抽象

从前面可以看出,继承是非常有用的。在有些场合需要一个超类,但是不能非常明确地指明要实现的方法。这样的超类很抽象,抽象到只有某些行为方面的概念,不知道其具体实现的方法。在这种情况下,类可以只提供一个方法声明,但是没有具体实现,这样的类就是抽象类。

1. 抽象方法和抽象类

抽象方法只有方法的声明,没有方法体。用户可以使用限定词 abstract 定义一个方法为抽象类型。拥有抽象方法的类是一个抽象类,这样的类同样需要用限定词 abstract 来声明:

```
[public] abstract class ClassName [extends Name]
   {  [modifier] abstract returnType methodName(inputParameters); }
```

一个从抽象类扩展而来的类,如果没有把抽象方法具体化,则它也是一个抽象类。抽象类不能进行实例化,也就是不能直接产生对象。

在抽象类中声明抽象方法可以迫使子类实现继承而来的抽象方法。在声明一个方法为抽象方法时,无须实现该方法,但要确保它的所有子类实现该方法。

在对实际应用问题进行抽象时,通常按照下述原则创建 Java 类:

(1) 把一个现实世界的实体当作一个类来处理,每个待处理的实体作为单独一个类。

(2) 对于每个实体(即类)使用“is-a”(是什么)、“has-a”(有什么)和“does-a”(做什么)短语来描述其名称属性、成员变量、成员方法以及它们的关系。

(3) 确保每个类尽可能小,而且方法尽可能简单。

(4) 通过已有的类,用继承、重载和覆盖技术减少代码重复。

(5) 以包含 main()方法的类作为主类,但实际上大多数工作应该是其他类完成。

(6) 一个好的面向对象的程序就像一个现代公司,每个雇员(普通类)接受良好的训练,他们的行动尽可能独立,执行官(主类)了解每个雇员的职能和专长,制订全盘计划,分配资源,明确职责,检查效果,但实际的工作都是雇员去做。

2. 最终方法和最终类

最终方法是在派生类中不能被覆盖的方法,最终类是不能被扩展的类,一个类可以含有最终方法。最终类及最终方法用 final 声明,格式如下:

```
[public] [final] className [extends ClassName]
{  [modifier] [final] returnType methodName(inputParameters)
     { /* 方法的实现 */ }
}
```

使用最终类和最终方法是为了保护类的完整性。不难想象,如果用户可以使用一个自己设计不好的 String 类代替 Java 的 String 类,一旦出了问题可能会导致整个 Java 系统出

问题。因此,Java 的 String 类是一个最终类,用户不能从 Java 的 String 类派生出子类。

最常使用 final 限定词的情况是常数,又因为 static 使得该变量只存在一个副本,final 使得它不能改变,所以 final 和 static 经常联合使用,以后可以用这些变量名代替具体数值。

最终类或最终方法不能被扩展和覆盖,这是面向对象程序设计的又一个基本机制。

5.1.4 Java 的基类 Object

在 Java 中定义的每一个类都来源于一个叫 Object 的基类,并且继承该类的所有非私有的属性和方法。因此 Java 中所定义的每一个类都有 Object 所提供的一些最基本的特征。

如果要弄清 Object 类的全部定义,必须查看 Java API。注意它的某些方法被定义为 final,因此不能被重载。Object 类主要包含如下方法:

```
public class java.lang.Object
{    public Object();
     public boolean equals(Object obj);
     protected void finalize();
     public final void notifyAll();
     public String toString();
     public final void wait(long timeout);
}
```

每个 Java 类都有一个祖先,这种做法有以下两个好处。

(1) 每个类继承 Object 类的方法。例如,每个 Java 类都有一个 toString()方法。

(2) 每个类既是所定义的类型,又属于 Object 类型。

例如,如果定义了一个 Object 类型的数组,意味着它可以包含任何对象。如果要使用 Object 数组中实际的对象类型,首先要把它们强制转换为实际类型。如果所使用的方法是所有对象的通用方法,则不必将对象进行强制转换。

如果要在一个数组中存储多种相关对象,可以创建一个类作为这些对象的父类,并且定义该数组为父类类型的数组。如果在父类中定义了一个抽象方法,则所有扩展该父类的类必须实现该方法,而且不必对对象进行强制转换就可以调用这些方法。

5.1.5 接口

在前面章节中介绍了类、实例、封装、重载和继承等概念,下面对接口和多重继承进行介绍。

1. 多级继承与多重继承

多级继承即一个超类的子类被另一个类继承。Java 支持多级继承。

多重继承即一个类同时继承多个超类。Java 不支持多重继承。

多重继承在 Java 中不允许使用,但 Java 提供了一个接口机制来代替它。

2. 接口的声明

接口是由一些抽象方法和常量组成的一个集合。它好比一个契约,保证每个实现它的类都包含该接口所声明的方法。接口中的方法都默认为 abstract,常量都默认为 final。一个接口不能用来实例化一个对象,它不是一个类。接口一般应是 public 的,而且能够扩展

其他接口。声明接口的一般格式如下：

```
public interface InterfaceName [extends InterfaceList]
```

一个类可通过关键字 implements 来继承一个或多个接口。该类继承了接口中的抽象方法和常量，并且除了自身的类型外，它还属于所继承的接口类型。该类必须实现所继承的方法或者被声明为 abstract 类。类使用接口的语法格式如下：

```
[public] class ClassName [extends Class] implements Interface1 [,Interface2]
```

3. 接口的使用技巧

使用接口通常包括以下 3 步：

(1) 如果有需要完成抽象的动作或具有概念性的属性，那么在接口中定义该动作或概念。抽象方法或概念性的属性在一个接口中声明。

(2) 描述上一步中抽象方法的具体算法或流程。该算法和流程在需要实现该接口的类中完成。

(3) 在某个或多个类中使用第(1)步中的方法或属性以及第(2)步中提供的算法或流程。

在某种属性是抽象的而且因对象不同所使用的算法或流程也不同的情况下，使用这些步骤可以使编程更方便。

5.1.6 多态性

多态性是面向对象程序设计中又一个重要的概念，它使得用户很容易在从超类继承来的新类上添加新的功能。

多态性是对象自动根据实际情况调用不同类层上适当的同名方法的一种现象。如果要使多态性在类层上起作用，类层至少要有一个公共行为特征，比如说共享的方法声明。

无论什么时候创建有层次关系的一些类，如果这些类有可能共享一个方法声明，就应当考虑使用多态性，分别在每个类中实现不同的方法。在超类中使用抽象(或普通)的方法，然后在子类中实现(或覆盖)该方法，这样就能产生多态性。

如果在类中有多个被覆盖的方法，多态性会选择与对象类型最相近的方法。

5.2 教材习题与解答

1. 什么是继承？继承的意义是什么？如何定义继承关系？

答：继承是一个类与其他类相关联的机制，它使一个类能具有其他类的特征，被继承的类称为超类或父类，继承的类称为子类或派生类。

继承用关键字 extends 表示，其语法形式为：

```
[modifier] class SubClassName extends ClassName
{ /* 类的实现 */ }
```

2. “子类的域和方法的数目一定大于或等于父类的域和方法的数目”，这种说法是否正确？为什么？

答：不正确，因为父类可能有私有的域和方法，所以子类的域和方法的数目不一定大于或等于父类的域和方法的数目。

3. 设有小学生、中学生、大学生类，为它们设计状态与行为，并利用第4章的学生类设计类的继承程序，分别创建这些学生对象并输出其信息。

参考答案：

```
public class AllKindStudent extends Student {
  protected String kind;

  protected AllKindStudent() {
      super();
      this.kind = "small";
  }

  protected AllKindStudent(String id, String name, String sex, int age) {
      super(id, name, sex, age);
      this.kind = "small";
  }

  @Override
  public String printInfo() {
      System.out.print("Kind: " + this.kind);
      return super.printInfo();
  }

  public static void main(String[] args) {
      AllKindStudent ss = new AllKindStudent("2019202030111", "张三", "男", 20);
      ss.printInfo();
   }
}
  //小学生
public class PupilStudent extends AllKindStudent {
  protected PupilStudent() {
      super();
      this.kind = "Pupil";
  }

   protected PupilStudent(String id, String name, String sex, int age) {
      super(id, name, sex, age);
      this.kind = "Pupil";
  }

  public void display() {
      System.out.println("PupilStudent:my name is " + this.name + ", I'm" + this.age);
  }
}

//中学生
public class MiddleStudent extends AllKindStudent {
  protected MiddleStudent() {
```

```
        super();
        this.kind = "middle";
    }

     protected MiddleStudent(String id, String name, String sex, int age) {
        super(id, name, sex, age);
        this.kind = "middle";
    }
    public void display() {
        System.out.println("MiddleStudent:my name is " + this.name + ", I'm" + this.age);
    }

}

//大学生
public class UndergraduateStudent extends AllKindStudent {
    public UndergraduateStudent() {
        super();
    }

    public UndergraduateStudent(String name, Integer age) {
        this.name = name;
        this.age = age;
    }

    public void display() {
        System.out.println("UndergraduateStudent:my name is " + this.name + ", I'm" + this.
age);
    }
}
```

4. 什么是域的隐藏？什么是方法的覆盖？方法的覆盖与域的隐藏有何不同？方法的覆盖与方法的重载有何不同？

答：子类重新定义一个与从父类那里继承来的域变量完全相同的变量，称为域的隐藏。即子类中定义了与父类同名的域变量，就是子类变量对同名父类变量的隐藏。

方法的覆盖是指子类重定义从父类继承来的一个同名方法，此时子类将清除父类方法的影响。子类可以通过重新定义与父类同名的方法实现自身的行为。

方法的覆盖与域的隐藏的不同之处在于：子类隐藏父类的域只是使其在子类中不可见，父类的同名域仍然占有自己独立的内存空间；而子类方法对父类同名方法的覆盖将清除父类方法占用的内存空间，从而使父类方法在子类对象中不复存在。

5. 如何实现方法的覆盖？实现方法的覆盖应注意什么？

答：如果在子类中重新定义了一个与它的超类完全同名的方法，子类的方法将覆盖超类中的同名方法。

在方法的覆盖中需要注意的问题是：子类在重新定义父类已有的方法时应保持与父类完全相同的方法头部声明，即应保持与父类有完全相同的方法名、返回类型和参数列表，否则就不是方法的覆盖，而是子类定义了一个与父类无关的方法，父类的方法在子类中仍然存

在。另外，在访问权限上只能放宽，而抛出异常不能抛出更多，否则编译时会出错。

6. 什么是多态？面向对象程序设计为什么要引入多态的特性？使用多态有什么优点？

答：多态就是多重状态，一个实体可以呈现多种状态。多态性允许不同类的对象对同一消息做出不同响应。

面向对象程序设计引入多态的特性，使语言具有灵活、抽象、行为共享、代码共享的优势，很好地解决了应用程序函数同名的问题。

使用多态可以解决继承性下的多样性问题，符合人们处理问题的思维习惯，使得软件开发时间短、效率高、可靠性高，开发的程序更易于维护、更新和升级。

7. Java 程序如何实现多态？有哪些实现方式？

答：Java 是面向对象的语言，多态性是面向对象程序设计代码重用的一个最强大的机制。Java 程序实现多态的方式是“一个名字，多个方法”。Java 实现运行时多态性的基础是动态方法调用，它是一种在运行时而不是在编译期调用重载方法的机制，主要体现在重载和覆盖两种实现方式上。

8. 解释 this 和 super 的意义和作用。

答：this 是当前对象的引用，可以指向当前对象的方法、变量。super 是父类的引用，在子类中可以通过 super 关键字来调用父类的域和方法。

当成员函数中定义了和成员变量中相同的变量时，引用成员变量要用 this；当构造函数中调用同一个类的其他构造函数时也用 this；当子类中覆盖了父类成员变量或成员函数时，在子类中调用父类的变量或函数要用 super；在子类的构造函数中调用父类的构造函数也用 super。

9. 什么是构造方法的重载？如何实现重载的构造方法之间的调用？

答：构造方法的重载是指在同一个类中定义了具有不同参数的多个构造方法，以完成不同情况下对象的初始化。

一个类的若干个构造方法之间可以相互调用。当类中的一个构造方法需要调用另一个构造方法时，可以使用关键字 this 加括号，在括号中可带也可不带参数，并且这个调用语句应该是该构造方法的第一个可执行语句。

10. 构造方法的继承原则有哪些？继承方式下子类的构造方法如何设计？

答：子类继承父类的构造方法遵循以下原则。

(1) 子类无条件地继承父类的无参数的构造方法。

(2) 如果子类没有定义构造方法，则它将继承父类的无参数构造方法作为自己的构造方法；如果子类定义了构造方法，则在创建新对象时，将先执行来自继承父类的无参数构造方法，然后再执行自己的构造方法。

(3) 对于父类的带参数构造方法，子类可以通过在自己的构造方法中使用 supper 关键字来调用它，但这个调用语句必须是子类构造方法的第一个可执行语句。

设计子类的构造方法可选择如下方式。

(1) 仅调用父类的无参数构造方法。

(2) 定义自己的一个(或多个)构造方法，并通过不同的参数选择调用父类的带参数的构造方法。

11. 以第 4 章 15 题中定义的 Address 类作为超类，定义一个子类，它拥有另外一个成员变量来存储电话号码。

参考答案:

```
class Address{
  String name, email;
  void showInfo(){
     System.out.println(name + ":" + email);
  }
}

public class AddressWithPhoneNum extends Address{
   long phoneNumber;
   void showInfo(){
      System.out.println("name:" + name  + "\nemail:" + email
                          + "\nphonenumber:" + phoneNumber);
   }

   public static void main(String[] args) {
      AddressWithPhoneNum person = new AddressWithPhoneNum();
      person.name = "Wu Da Lang";
      person.email = "liyuna.guo850412@163.com";
      person.phoneNumber = 68764848;
      person.showInfo();
   }
}
```

12. 下面的程序创建一个名为 SuperClass 的类,它的构造方法有一个 String 类型的参数;另一个名为 SubClass 的子类,它从 SuperClass 类扩展而来,也有一个参数为 String 类型的构造方法。如下的 Test 类在实例化 SubClass 的一个对象时出现错误,试改正之。

```
public class SuperClass
{  public SuperClass(String msg)
   {  System.out.println("SuperClass constructor: " + msg); }
}
public class SubClass extends SuperClass
{  public SubClass()
   {  System.out.println("SubClass constructor"); }
}
public class Test
{  public static void main(String args[])
   {  SubClass descendent = new SubClass(); }
}
```

参考答案:

以上程序不能恰当地调用超类构造方法,在编译时出现了错误,改正如下。

改正一:重载 SuperClass 构造方法,确保它含有无参数的版本。

```
public class SuperClass
{  public SuperClass()
   {  System.out.println("SuperClass constructor"); }
   public SuperClass(String msg)
   {  System.out.println("SuperClass constructor: " + msg); }
}
```

改正二：在 SubClass 构造方法的第 1 行加入调用 SuperClass 构造方法的语句。

```
public class SubClass extends SuperClass
{  public SubClass()
   {  super("Input message");
      System.out.println("SubClass constructor");
   }
}
```

13. 用最终成员变量和抽象方法为某州立大学的学生建立账单系统，州内外的学生收费不同，州内每学分收费 $750，州外为 $2000，每个学生的账单上有学校名称、学生姓名、信用卡使用的时间以及账单的总数。

参考答案：

```
public abstract class Student
{  protected final static double INSTATE_RATE = 750;
   protected final static double OUTSTATE_RATE = 2000;
   protected String name;
   protected int hours;

   public abstract void showStudent();

   public final void showSchoolName()
   {  System.out.println("Java State University");
      System.out.println("**********************");
   }
}

public class OutStateStudent extends Student
{  public OutStateStudent(String _name, int _hours)
   {  name = _name;
      hours = _hours;
   }

   public void showStudent()
   {  showSchoolName();
      System.out.println(name + " takes " + hours + " credits.");
      System.out.println("OutState bill: " + hours * OUTSTATE_RATE);
   }
}

public class InStateStudent extends Student
{  public InStateStudent(String _name, int _hours)
   {  name = _name;
      hours = _hours;
   }

   public void showStudent()
   {  showSchoolName();
      System.out.println(name + " takes " + hours + " credits.");
```

```
        System.out.println("InState bill: " + hours * INSTATE_RATE);
    }
}
public class BursarsOffice
{   public static void main(String args[])
    {   InStateStudent resident = new InStateStudent("John Doe", 24);
        OutStateStudent alien = new OutStateStudent("Joan Smith", 26);
        resident.showStudent();
        alien.showStudent();
    }
}
```

14. 定义一个 Object 数组,它可以存储一个矩形、一个圆、一个双精度数或一个整数。参考答案:

```
public class Shape
{   protected String name;
    protected double area, perimeter;

    public Shape()
    {   name = "undetermined";
        area = perimeter = 0;
    }

    public String toString()
    {   return name + " [area = " + area + ", perimeter = " + perimeter + "]";
    }

    public void display()
    {   System.out.println(toString());
    }
}

public class Rectangle extends Shape
{   protected double length, width;

    public Rectangle(double _length, double _width)
    {   name = "Rectangle";
        length = _length;
        width = _width;
    }

    public void computeArea()
    {   area = length * width;
    }

    public void computePerimeter()
    {   perimeter = 2 * (length + width);
    }

    public void display()
```

```
    {  super.display();
       System.out.println("Length: " + length);
       System.out.println("Width: " + width);
    }
}

public class Square extends Rectangle
{  public Square(double _side)
    {  super(_side, _side);
       name = "Square";
    }
}
public class Circle extends Shape
{  protected double radius;
    public Circle(double _radius)
    {  name = "Circle";
       radius = _radius;
    }

    public void computeArea()
    {  area = Math.PI * radius * radius;
    }

    public void computePerimeter()
    {  perimeter = 2 * Math.PI * radius;
    }
}

public class ObjectArray
{  public static void display(Object obj)
    {  if (obj instanceof Rectangle)
       {  System.out.println( ( (Rectangle) obj).toString());
       }
       else if (obj instanceof Circle)
       {  System.out.println( ( (Circle) obj).toString());
       }
       else if (obj instanceof Double)
       {  System.out.println( ( (Double) obj).toString());
       }
       else if (obj instanceof Integer)
       {  System.out.println( ( (Integer) obj).toString());
       }
    }

    public static void main(String args[])
    {  Object objects[] = new Object[4];
       objects[0] = new Rectangle(3, 4);
       objects[1] = new Circle(5);
       objects[2] = new Double(1.0);
       objects[3] = new Integer(2);
       for (int i = 0; i < objects.length; i++)
```

```
        {  display(objects[i]);
        }
    }
}
```

15. 创建一个名称为 List 的类,它可存储任何类型的对象,并可以在任何时候增加或删除对象。

参考答案:

```
public class List
{  private int maxItems = 100;
   private int numItems = 0;
   private Object[] list = null;
   public List()
   {  list = new Object[maxItems];
   }

   public List(int _maxItems)
   {  maxItems = _maxItems;
      list = new Object[maxItems];
   }

   public void add(Object obj)
   {  /* 假设还有空间增加一个对象,即假设 numItems < maxItems */
      list[numItems] = obj;
      numItems++;
   }

   public void delete(int pos)
   {  /* 假设 pos 为 0～numItems */
      for (int i = pos + 1; i < numItems; i++)
      {
         list[i - 1] = list[i];
      }
      numItems--;
   }

   public Object get(int pos)
   {  return list[pos];
   }

   public int getSize()
   {  return numItems;
   }

   public boolean isFull()
   {  return (numItems >= maxItems);
   }

   public String toString()
   {  String s = new String();
```

```
        for (int i = 0; i < numItems; i++)
        {  s += "\n" + list[i].toString();
        }
        return s + "\n";
    }
}

public class ListTest
{  public static void main(String args[])
    {  List list = new List();
        list.add(new Double(10.0));
        list.add(new String("Java by Definition"));
        list.add(new Integer( -10));
        System.out.println(list);
        System.out.println("Position 0:" + list.get(1));
        list.delete(2);
        list.delete(0);
        System.out.println("List Size: " + list.getSize());
    }
}
```

16. 定义一个接口 OneToN,在接口体中包含一个抽象方法 disp()。定义 Sum 和 Pro 类,并分别用不同代码实现 OneToN 中的 disp()方法,在 Sum 的方法中计算 1～n 的和,在 Pro 的方法中计算 1～n 的乘积。

参考答案:

```
//接口 OneToN
  public interface OneToN {
    int disp();
  }
//Sum 和 Pro 类
public class Sum implements OneToN {
    private int n;
    public Sum(int n) {
        this.n = n;
    }
    @Override
    public int disp() {
        return(1 + n) / 2 * n;
    }

  public class Pro implements OneToN {
    private int n;
    public Pro(int n) {
        this.n = n;
    }
    @Override
    public int disp() {
        if (n < 0) {
            return -1;
```

```
        }

        int result = 1;
        for (int i = 2; i < n + 1; i++) {
            result * = i;
        }
        return result;
    }
```

17. 编写一个类,使用第 16 题中接口类型衍生的类。

参考答案:

```
public class UseInterface
  public static void main(String[] args) {
      OneToN sum = new Sum(4);
      OneToN pro = new Pro(4);
      System.out.println("sum.disp():" + sum.disp());
      System.out.println("pro.disp():" + pro.disp());
  }
}
```

18. 给定下面的类定义:

```
class Test1 {
   float aMethod(float a, float b) { return a;}
}
class Test2 extends Test1 {
   //…
}
```

请问下面哪些方法可以加到类 Test2 中的注释行处?

A. float aMethod(float b,float a){ return a; }

B. public int aMethod(int a,int b) { return a; }

C. protected float aMethod(float p,float q){ return a; }

D. private float aMethod(float p,float q) { return a; }

答:只有 D 不行,因为覆盖的方法不能比被覆盖的方法有更严格的访问权限。

19. 给定以下代码,请选择一个正确的答案替换其中的注释行。

```
class A {
  A(int i) { }
}
public class B extends A {
 B() {
    // …
 }
 public static void main(String args[]) {
    B b = new B();
 }
}
```

A. super(100); B. this(100); C. super(); D. this();

答案：A

解答：A 是调用父类带参数的构造方法。

20. 编译、运行下面的代码将发生什么？

```
class Base{}
class Sub extends Base{}
public class BaseToSub{
   public static void main(String argv[]) {
      Base b = new Base();
      Sub s = (Sub) b;
   }
}
```

A. 编译和运行都不会出错 B. 编译出错

C. 运行时抛出例外 D. 以上选项都不对

答案：C

解答：运行时出现“Exception in thread "main" java.lang.ClassCastException：Base cannot be cast to Sub at BaseToSub.main(BaseToSub.java：6)”。

21. 考虑类定义：

```
class BaseWidget extends Object{
   String name = "BaseWidget";
   void speak() { System.out.println("I am a " + name); }
}
class TypeAWidget extends BaseWidget{
   TypeAWidget() { name = "TypeA"; }
}
```

基于以上类定义，下面哪个代码片段是正确的？

A. Object A=new BaseWidget()；A.speak()；

B. BaseWidget B=new TypeAWidget()；B.speak()；

C. TypeAWidget C=new BaseWidget()；C.speak()；

答案：B

解答：A、C 都无法通过编译。

5.3 补充习题与解答

5.3.1 单选题

1. 试分析下列代码：

```
class Aclass{
   Aclass(){
        System.out.print("Aclass ");
    }
}
```

```
public class Bclass extends Aclass{
  Bclass(){
    System.out.print("Bclass ");
  }
  public static void main(String args[]){
    Aclass a = new Aclass();
    Aclass a1 = new Bclass();
  }
}
```

其执行的结果为(　　)。

A. 编译失败

B. 编译成功且输出“Aclass Bclass”

C. 编译成功且输出“Aclass Aclass Bclass”

D. 编译成功且输出“Aclass Bclass Bclass”

答案：B

解答：在 main()方法中创建了 Aclass 和 Bclass 两个对象，而在调用构造方法时，Aclass()的执行结果是“Aclass”。对于 Bclass 类的无参构造方法，按照构造方法的继承性，要首先调用它的父类的无参构造方法，再执行自己的方法体，因此调用构造方法 Bclass()的结果是“Aclass Bclass”。

2. 分析下列程序：

```
class Super{
  public int i = 0;
  public Super(String text){
    i = 1;
  }
}
public class Sub extends Super{
  public Sub(String text){
     i = 2;
  }
  public static void main(String args[]){
    Sub sub = new Sub("Hello");
    System.out.println(sub.i);
   }
}
```

该程序的结果是(　　)。

A. 编译失败　　B. 编译成功且输出“0”

C. 编译成功且输出“1”　　D. 编译成功且输出“2”

答案：A

解答：因为超类中缺少无参构造方法，所以编译错误。

3. 分析以下程序段：

```
abstract class AbstractIt {
    abstract float getFloat();                //第 2 行
```

```
}
public class AbstractTest extends AbstractIt {
    private float f1 = 1.0f;
    private float getFloat() {return f1;}       //第 6 行
}
```

下面(　　)结果正确。

A. 可编译成功　　B. 在第 6 行运行失败

C. 在第 6 行编译失败　　D. 在第 2 行编译失败

答案：C

解答：在覆盖父类的方法时不能加上更加严格的访问修饰符。父类的方法的访问修饰符为 package，而子类的为 private。

4. 分析下列程序：

```
1  class Super {
2     public int getLength() {return 4;}
3  }
4
5  public class Sub extends Super {
6     public int getLength() {return 5;}
7
8     public static void main (String[]args) {
9        Super sooper = new Super();
10       Sub sub = new Sub();
11       System.out.println(
12       sooper.getLength() + "," + sub.getLength() );
13  }
14 }
```

该程序的输出是(　　)。

A. 4,4　　B. 4,5　　C. 5,4　　D. 5,5

答案：B

解答：本题中子类覆盖了父类的 getLength()方法，所以分别用父类对象和子类对象调用同名方法时执行的是各自的方法，故该程序的执行结果为 B。

5. 分析下列程序：

```
1  class A {
2     public byte getNumber() {
3       return 1;
4     }
5  }
6
7  class B extends A {
8     public short getNumber() {
9       return 2;
10    }
11
12    public static void main(String args[]) {
```

```
13        B b = new B();
14        System.out.println(b.getNumber());
15    }
16 }
```

该程序的结果是(　　)。

A. 编译成功并输出 1

B. 编译成功并输出 2

C. 编译在第 8 行引起错误

D. 编译在第 14 行引起错误

E. 编译成功但执行时在第 14 行抛出异常

答案:C

解答:子类和父类的两个 getNumber()方法不能构成覆盖关系,但又同名且参数相同,因此出现语法错误。子类覆盖父类的同名方法要求方法的返回类型、方法名、方法的参数完全相同,而本题中父类的方法的返回类型是 byte,子类的方法的返回类型是 short,这是出现语法错误的原因。

6. 分析下列程序:

```
class A{
  public int getNumber(int a){
     return a + 1;
   }
}
class B extends A{
   public int getNumber(int a, char c){              //第 7 行
      return a + 2;
   }
   public static void main(String[] args){
      B b = new B();
      System.out.println(b.getNumber(0));          //第 14 行
  }
}
```

该程序的执行结果是(　　)。

A. 编译成功并输出 1　　B. 编译成功并输出 2

C. 在第 7 行出现编译错误　　D. 在第 14 行出现编译错误

答案:A

解答:此题与第 5 题比较,答案完全不同。此题中子类方法的方法名与父类的方法名相同,好像是覆盖。但子类方法的参数与父类方法的参数不同,因此算不上是覆盖。当在子类调用中没有合适的方法时(这里是指没有具有适合方法的参数),调用超类中可以匹配的同名方法,所以执行结果为 A。

注意:子类和父类的方法,如果方法名相同且参数相同,但返回类型不同,则为语法错误。若子类和父类方法名相同,但参数不同,则语法正确,通过参数的不同可以确定是调用父类还是子类的方法。

7. 分析下列程序：

```
1  public class SuperClass{
2      class SubClassA extends SuperClass{}
3      class SubClassB extends SuperClass{}
4      public void test(SubClassA foo){
5          SuperClass bar = foo;
6      }
7  }
```

以下对该程序的说法正确的是(　　)。

A. 第5行的赋值语句是非法的

B. 第5行的赋值语句是合法的，但执行时抛出一个ClassCastException异常

C. 程序的语法是正确的，在使用时不会抛出异常

D. 程序的语法不正确，不允许内部类继承外部类

答案：C

解答：该程序的语法是正确的，Java允许内部类继承外部类，所以选项D错；子类的对象可以被赋值给超类的对象，所以选项A、B都不正确。

8. 分析下列程序：

```
1  class A {
2       public int getNumber(int a) {
3         return a + 1;
4       }
5  }
6
7  class B extends A {
8       public int getNumber(int a) {
9         return a + 2;
10       }
11
12       public static void main(String args[]) {
13         A a = new B();
14         System.out.println(a.getNumber(0));
15       }
16 }
```

该程序的执行结果是(　　)。

A. 编译成功并输出1　　B. 编译成功并输出2

C. 在第8行出现编译错误　　D. 在第13行出现编译错误

E. 在第14行出现编译错误

答案：B

解答：在main()方法中创建的对象是B类对象，因为调用的是B类的构造方法，创建的A类的引用来引用B类的对象。在方法覆盖时，父类的方法在子类中不复存在，因此用这个引用调用的是B类的方法，其执行结果是2，故选择B。

9. 分析下列程序：

```
1  class Person {
2    public void printValue(int i, int j) {//... }
3    public void printValue(int i){//... }
4  }
5  public class Teacher extends Person {
6    public void printValue() {//... }
7    public void printValue(int i) {//...}
8    public static void main(String args[]){
9      Person t = new Teacher();
10     t.printValue(10);
11   }
12 }
```

第 10 行语句将调用(　　)语句。

A. 第 2 行　　B. 第 3 行　　C. 第 6 行　　D. 第 7 行

答案：D

解答：本题较上一题看起来增加了一些复杂性，那就是父类和子类中都有关于 printValue()方法的重载，但本质是一样的。在 main()方法中创建的对象是子类的对象，因为调用的是子类的构造方法，但创建的父类类型的引用来引用子类的对象，所以用这个引用调用方法调用的是子类的方法，再根据其参数进行区分，调用的应该是第 7 行的 printValue()方法，故选择 D。

10. 分析下列程序：

```
1  interface Foo{
2    int k = 0;
3  }
4  public class Test implements Foo{
5   public static void main(String args[]){
6      int i;
7      Test test = new Test();
8      i = test.k;
9      i = Test.k;
10     i = Foo.k;
11  }
12 }
```

该程序的结果是(　　)。

A. 编译成功　　B. 编译时在第 2 行出现错误

C. 编译时在第 8 行出现错误　　D. 编译时在第 9 行出现错误

E. 编译时在第 10 行出现错误

答案：A

解答：在第 2 行声明的整型变量实际隐含这个变量是 static 的且是 final 的，因为它是在接口中声明的。实现这个接口的类具有这两个属性，因此第 8、9、10 行的使用都是对的。第 8 行是因为 Test 类实现了 Foo 接口，也就自动继承了变量 k，所以可以使用对象名来引用变量 k；第 9 行是因为 Test 类自动继承了变量 k，而 k 是 static 的，所以可以使用类名来

引用静态变量 k；第 10 行是因为对于接口来说可以使用接口名来引用它包含的常量 k。

5.3.2 多选题

1. 下列选项中，(　　)可以创建一个数组实例。

A. int[] ia = new int[15];　　B. float fa = new float[20];

C. char[] ca = "Some String";　　D. Object oa = new float[20];

E. int ia[][] = (4,5,6) (1,2,3)

答案：A、D

解答：A 可以创建一个整型数组，这是创建数组的标准语法；B 不对，它的左端是声明一个浮点数，右端才是声明一个数组；C 不对，因为左端是声明字符数组，而右端是声明一个字符串，两边类型不对；D 对，因为数组也是一个引用类型，任何引用类型都是 Object 的子类；E 错，右边的初始化格式不对，正确的格式应该是{{4,5,6},{1,2,3}}。

2. 假定有程序段：

```
1  class BaseClass {
2     private float x = 1.0f;
3     protected float getVar () { return x;}
4  }
5  class Subclass extends BaseClass {
6     private float x = 2.0f;
7        //此处插入代码
8  }
```

下列选项中，(　　)方法是对父类方法的有效覆盖。

A. float getVar() { return x; }

B. public float getVar() { return x; }

C. double getVar() { return x; }

D. protected float getVar() { return x+1; }

E. public float getVar(float f) { return f; }

答案：B、D

解答：A、C 不对，因为 A、C 声明的方法的访问权限是包可访问，弱于父类方法的访问权限 protected，并且 C 的返回类型不一致；B 是有效覆盖，除了返回类型、方法名和参数相同外，访问权限声明 public，比父类更宽松，这是合法的；D 是标准的覆盖，方法首部声明完全一样，只是方法体不一样而已；E 在语法上是正确的，但不是覆盖，因为参数不一样。

3. 下列方法中，(　　)方法能够防止被覆盖。

A. final void methoda(){}　　B. void final methoda(){}

C. static void methoda(){}　　D. static final void methoda(){}

E. final abstract void methoda(){}

答案：A、D

解答：A、D 符合语法规范。被声明为 final 的方法不可以被覆盖，B 错在顺序问题，final 不能写在 void 后面；C 没有用 final 关键字声明，所以可以被覆盖；E 错，因为声明为 abstract 的类必须要被覆盖，而 final 是防止覆盖，两个关键字不能连用。

4. 假设 AnInterface 是一个接口；AnAdapter0 是一个非抽象(non-abstract)、非最终(non-final)并具有无参数构造方法的类；AnAdapter1 是一个非抽象(non-abstract)、非最终(non-final)并具有一个 int 型参数构造方法的类。

下面可以创建一个匿名内部类的两项是(　　)。

A. method1(new AnAdapter0(){})

B. method2(new AnAdapter1(5){})

C. AnAdapter1 aa=new AnAdapter1(){}

D. AnAdapter0 aa=new AnAdapter0(5){}

E. AnInterface ai=new AnInterface(5){}

答案：A、B

解答：定义匿名内部类一般可以使用 new 关键字，后面跟一个 class 类型作为一个方法的输入参数。

```
someMethod(new ClassType([constructorInputList])
{ /* 定义的内部类,包括域和方法 */ }
```

内部类通常用来定义一个急于处理的事件类，或者用于代码比较短并且只用于某一个类的情况。匿名内部类很容易定义，但是它在被嵌入类的内部不能重用，而且匿名类没有构造方法。有名内部类要更灵活一些，它在被嵌入类的内部可重用。因此用户应尽量使用有名内部类，而不是匿名内部类。

C、D 错，是因为有参数的构造方法就应该传递参数，有无参的构造器就不能传递参数。E 错是因为接口没有构造方法。

5. 假定有程序段：

```
public interface Foo{
   int k = 4;
}
```

下面与以上接口的第 2 行等价的是(　　)。

A. final int k= 4;　　B. public int k= 4;

C. static int k= 4;　　D. private int k= 4;

E. abstract int k= 4;　　F. volatile int k= 4;

G. transient int k= 4;　　H. protected int k= 4;

答案：A、B、C

解答：接口中声明的变量都等价于 static、public 和 final 的变量，因此 A、B、C 都对。其他选项都不完全具备这些属性。

5.3.3 填空题

1. 阅读下列程序：

```
class A{
     public String toString(){
       return "4";
     }
```

```
}
class B extends A{
   public String toString(){
      return super.toString() + "3";
   }
 }
 public class Test{
    public static void main(String[] args){
       B b = new B();
       System.out.println(b.toString());
    }
 }
```

该程序的执行结果是(　　)。

答案：43

解答：对象 b 是 B 类的实例，理所应当调用 B 类的方法，B 类方法再调用超类 A 中的 toString()方法。

2. 阅读下列程序：

```
public class SuperClass{
   String a = "hello";
   class SubClassA extends SuperClass{}
   static class SubClassB extends SuperClass{
       String a = "aaa";
   }
   public static void main(String args[]){
      SuperClass bar = new SubClassB();
      System.out.println(bar.a);
   }
}
```

该程序的执行结果是(　　)。

答案：hello

解答：可以和选择题的第 8、9 题比较，在那里是方法的覆盖。当方法覆盖时，父类的方法在子类中不复存在；而对于父类和子类的同名域，在子类中只是对父类的同名域进行隐藏，在子类中还是可以访问到父类的域，因此用父类的引用访问到的是父类的域，所以 bar.a 的内容是“hello”。

当子类不是父类的内部类时，对于父类和子类的同名域，在子类中还是可以访问到父类的域。于是用父类的引用访问到的还是父类的域。验证程序如下：

```
class SuperClass{
    public String a = "hello";
}

class SubClassB extends SuperClass{
    public String a = "aaa";

    public static void main(String args[]){
```

```
        SuperClass bar = new SubClassB();
        System.out.println(bar.a);
    }
}
```

3. 阅读下列程序:

```
class SuperClass{
   String a = "hello";
}
public class SubClassB extends SuperClass{
   String a = "aaa";
   String b = "bbb";
   public static void main(String args[]){
       SuperClass bar = new SubClassB();
       System.out.println(bar.a);
       System.out.println(bar.b);
   }
}
```

该程序的结果是(　　)。

答案:编译出错

解答:这里创建的 SuperClass 的引用,它引用的是 SubClassB 对象的父对象,因此访问 bar 的域 b 时不存在这个域,出现了未知符号,所以编译出错。

4. 阅读下列程序:

```
public class SuperClass{
   String a = "hello";
   class SubClassA extends SuperClass{}
   static class SubClassB extends SuperClass{
       String a = "aaa";
   }
   public static void main(String args[]){
      SubClassB bar = new SubClassB();
      System.out.println(bar.a);
   }
}
```

该程序的执行结果是(　　)。

答案:aaa

解答:bar 是 SubClassB 类对象的引用,当利用该引用访问成员变量 a 时,在子类中隐藏父类的成员变量 a,因此访问子类的成员变量 a,所以输出的值为 aaa。

5. 分析下列程序:

```
class SuperClass{
      String a = "hello";
void test(){
         System.out.print(a);
      }
}
```

```
public class SubClass extends SuperClass{
 String a = "aaa";
       public static void main(String args[]){
   SubClass bar = new SubClass();
   bar.test();
   System.out.println(bar.a);
 }
}
```

该程序的结果是(　　)。

答案：hello aaa

解答：在 main()方法中创建子类 SubClass 对象的应用，因此使用 bar.test()调用 bar 的 test 方法时，由于子类没有定义方法，调用的是从父类继承过来的方法，它访问的是父类的域 a，所以输出“hello”，而直接用 bar.a 时访问子类对象的域 a，所以输出“aaa”。

5.3.4 简答题

1. Java 中实现多态的机制是什么？

答：方法的重写(Override)和重载(Overload)是(Java)多态性的不同表现。重写(Override)是父类与子类之间多态性的一种表现，重载(Overload)是一个类中多态性的一种表现。

2. 方法覆盖的基本原则是什么？

答：方法覆盖的基本原则如下。

(1) 覆盖方法的返回类型必须与它所覆盖的方法相同；

(2) 覆盖方法不能比它所覆盖的方法的访问性差；

(3) 覆盖方法不能比它所覆盖的方法抛出更多的异常。

3. 有继承关系的父类对象和子类对象之间可以在一定条件下互相转换，这种转换要遵循的原则是什么？

答：父类对象和子类对象之间互相转换要遵循的原则如下。

(1) 子类对象可以被视为父类的一个对象，反之不可；

(2) 若一个方法的形式参数的类型是父类类型，则调用该方法的实际参数可以使用子类对象；

(3) 若父类对象的引用指向的是一个子类对象，则这个父类对象的引用可以强制类型转换成子类对象的引用。

4. 下列程序有错吗？

```
interface A{
   int x = 0;
}
class B{
   int x =1;
}
class C extends B implements A {
   public void pX(){
      System.out.println(x);
```

```
    }
    public static void main(String[] args) {
        new C().pX();
    }
}
```

答：有错。在编译时会发生错误(错误描述不同的 JVM 有不同的信息，意思就是未明确的 x 调用，两个 x 都匹配，就像同时在 import java.util 和 java.sql 两个包中直接声明 Date 一样)。如果是使用父类的变量，可以用 super.x 来明确，而接口的属性默认隐含为 public static final，可以通过 A.x 来明确是否使用接口中的 x 变量。

5. 下列程序有错吗？

```
interface Playable {
    void play();
}
interface Bounceable {
    void play();
}
interface Rollable extends Playable, Bounceable {
    Ball ball = new Ball("PingPang");
}
class Ball implements Rollable {
    private String name;
    public String getName() {
        return name;
    }
    public Ball(String name) {
        this.name = name;
    }
    public void play() {
        ball = new Ball("Football");
        System.out.println(ball.getName());
    }
}
```

答：有错，这个错误不容易发现。“interface Rollable extends Playable，Bounceable”没有问题，因为 interface 可继承多个 interface。问题出在 interface Rollable 中的如下语句：

```
Ball ball = new Ball("PingPang");
```

因为任何在 interface 中声明的 interface variable(接口变量，也可称成员变量)默认为 public static final。也就是说：

```
Ball ball = new Ball("PingPang");
```

实际上是

```
public static final Ball ball = new Ball("PingPang");
```

在 Ball 类的 play()方法中，“ball = new Ball("Football")”改变了 ball 的引用，而这里的 ball 来自 Rollable interface，Rollable interface 里的 ball 是 public static final 的，final 的对

象是不能被改变引用的。因此,编译器将在“ball = new Ball("Football")”处显示有错。

6. Overload 和 Override 的区别是什么? Overload 的方法是否可以改变返回值的类型?

答:方法的重写(Override)和重载(Overload)是 Java 多态性的不同表现。重写(Override)是父类与子类之间多态性的一种表现,重载(Overload)是同一个类中多态性的一种表现。如果在子类中定义某方法与其父类有相同的名称和参数,即为该方法覆盖(Override)父类的同名方法,当子类的对象使用这个方法时将调用子类中的定义,对它而言,父类中的定义如同被“屏蔽”了。

如果在一个类中定义了多个同名的方法,它们或有不同的参数个数或有不同的参数类型,则称为方法的重载(Overload)。多个 Overload 的方法可以有不同的返回值类型。

7. abstract class 和 Interface 有什么区别?

答:一个类声明它的方法而不去实现,即包含抽象方法的类被叫作抽象类(abstract class),它用于要创建一个体现某些基本行为的类,并为该类声明方法,但不能在该类中实现该类的情况。注意,不能直接通过构造方法创建 abstract 类的实例,然而可以创建一个变量,其类型是一个抽象类,并让它指向具体子类的一个实例;不能有抽象构造方法或抽象静态方法。abstract 类的子类为它们父类中的所有抽象方法提供实现,否则它们也是抽象类,如果子类是抽象类,则在这些子类的子类中实现该方法,只要是继承了其抽象方法的类就可以在类中实现这些方法。

接口(Interface)是抽象类的变体。在接口中,所有方法都是抽象的。多继承性可通过实现这样的接口来获得。接口中的所有方法都是抽象的,没有一个有程序体。接口只可以定义 static final 成员变量。接口的实现与子类相似,除了该实现类不能从接口定义中继承行为以外。

当类实现特殊接口时,将实现(即将给予这些方法的方法体)所有这种接口中的方法,然后它就可以在实现了该接口的类的任何对象上调用接口的方法。由于有抽象类,它允许使用接口名作为引用变量的类型。通常的动态联编将生效。引用可以转换到接口类型或从接口类型转换到相关的引用类型,instanceOf 运算符可以用来决定某对象的类是否实现了接口。

8. 接口是否可继承接口? 抽象类是否可实现(implements)接口? 抽象类是否可继承实体类(concrete class)?

答:接口可以继承接口。抽象类可以实现(implements)接口。抽象类可继承实体类的前提是实体类必须有明确的构造方法。

9. 简述 Java 的接口的主要特点。

答:由于 Java 不支持多继承,而某个类或对象有可能要使用分别在几个类或对象里面的方法或属性,现有的单继承机制不能满足要求。与继承相比,接口有更高的灵活性,因为接口中没有任何实现代码。当一个类实现了接口以后,该类要实现接口里面所有的方法和属性,并且接口里面的属性在默认情况下都是 public static,所有方法在默认情况下都是 public。一个类可以实现多个接口。

10. Anonymous Inner Class(匿名内部类)是否可以 extends(继承)其他类? 是否可以 implements(实现)interface(接口)?

答:可以继承其他类或实现其他接口,在 Swing 编程中常用此方式。

5.3.5 编程题

1. 以 Point 类为例,尝试对 Object 类的 clone()和 equals()方法进行覆盖。

参考答案:

```
class Point
{  private double x,y;
   public Point(double a,double b){
     x = a;
     y = b;
   }

   public Point(Point p){
     x = p.x;
     y = p.y;
   }

   public Object clone(){
     return new Point(x,y);
   }

    public boolean equals(Point p){
      if (p instanceOf Point)
           return (x == ((Point)p).x && y == ((Point)p).y);
     else
       return false;
    }
    public String toString(){
    return new String("(" + x + "," + y + ")");
    }
}
class TestPoint
{  public static void main(String[] args)
    {  Point p = new Point(2, 3);
       System.out.println("p = " + p);

       Point q = (Point)p.clone();              //由 p 复制出 q
       System.out.println("q = " + q);

       if (q.equals(p))System.out.print("q equals p");
       else System.out.print("q does not equals p");
       if (q == p)System.out.println(", and q == p");
       else System.out.println(", and q != p");
    }
}
```

2. 定义一个有抽象方法的超类 SuperClass,并提供实现其抽象方法的子类 SubClassA 和 SubClassB,且创建一个测试类 PolyTester,测试类有一个具有两个元素的 SuperClass 对象数组,数组元素分别指定为 SubClassA 和 SubClassB。循环调用每个数组元素的共享方

法 display(),看看它们会调用哪个版本的 display()。

参考答案:

```
public abstract class SuperClass
{  public abstract void display();
}

public class SubClassA extends SuperClass
{  public void display()
   {  System.out.println("This is subclass A");
   }
}

public class SubClassB extends SuperClass
{  public void display()
   {  System.out.println("This is subclass B");
   }
}

public class PolyTester
{  public static void main(String args[])
   {  SuperClass array[] = new SuperClass[2];
      array[0] = new SubClassA();
      array[1] = new SubClassB();
      for (int i = 0; i < array.length; i++)
      {  array[i].display();
      }
   }
}
```

3. 创建 Address、Rectangle、Length 和 Check 类,在从 Object 类继承 toString()方法的基础上实现新的 toString()方法,然后为这些类创建一个 Object 数组,并依次显示转换出来的字符串。

参考答案:

```
public class Address
{  private String name, email;
   public Address(String _name, String _email)
   {  name = _name;
      email = _email;
   }

   public String toString()
   {  return name + " (" + email + ")";
   }
}

public class Length
{  private String scale;
   private double value;
```

```
    public Length(double _value, String _scale)
    {  value = _value;
       scale = _scale;
    }
    public String toString()
    {  return value + " " + scale;
    }
}

public class Check
{  private String descr;
   private double amount;

   public Check(String _descr, double _amount)
   {  descr = _descr;
      amount = _amount;
   }

   public String toString()
   {  return descr + " $ " + amount;
   }
}

public class Rectangle
{  private double width, height;
   public Rectangle(double _width, double _height)
   {  width = _width;
      height = _height;
   }

   public String toString()
   {  return width + " x " + height;
   }
}

public class TestAutoString
{  public static void main(String args[])
   {  Object array[] = new Object[4];
      array[0] = new Address("Bert Wachsmuth", "wachsmut@shu.edu");
      array[1] = new Check("Regular salary", 6500);
      array[2] = new Length(5.11, "feet");
      array[3] = new Rectangle(3, 4);
      for (int i = 0; i < array.length; i++)
      {  System.out.println(array[i]);
      }
   }
}
```

4. 在主教材 5.2.5 节的 Shape 类层中添加 RightTriangle,把 RightTriangle 当作一种特殊形式的 Rectangle,并修改 ShapeTestWithPolymorphism。

参考答案：

```
public abstract class Shape
{/* 同教材 5.2.5 节 */}

public class Square extends Rectangle
{/* 同教材 5.2.5 节 */}

public class Rectangle extends Shape
{/* 同教材 5.2.5 节 */}

public class Circle extends Shape
{/* 同教材 5.2.5 节 */}

class RightTriangle extends Rectangle
{  RightTriangle(double _width, double _height)
   {  super(_width, _height);
      type = "Triangle";
   }
   public void computeArea()
   {  area = 0.5 * width * height;
   }
   public void computePerimeter()
   {  perim = width + height + Math.sqrt(width * width + height * height);
   }
}

public class ShapeTestWithPolymorphism
{  public static void main(String args[])
   {  Shape shapes[] = {
          new Rectangle(2, 4), new Circle(3),
          new Square(4), new RightTriangle(3, 4)};
      for (int i = 0; i < shapes.length; i++)
      {  shapes[i].computeArea();
         shapes[i].computePerimeter();
         System.out.println(shapes[i]);
      }
   }
}
```

5. 编写 3 个接口，它们之间具有继承关系，在每个接口中包含一个常量字符串。试通过一个类继承这些接口，通过显示接口中的常量字符串来展示接口的继承性。

参考答案：

```
interface A{
    String a = "接口 A 中的常量";
    void showA();
}

interface B extends A{
    String b = "接口 B 中的常量";
```

```
    void showB();
}

interface C extends B{
    String c = "接口 C 中的常量";
    void showC();
}
class ImpInterfaceABC implements C{
    public void showA(){System.out.println(a);}
    public void showB(){System.out.println(b);}
    public void showC(){System.out.println(c);}
};

class InterfaceInheritanceTest
{ public static void main(String[] args)
    { ImpInterfaceABC intf = new ImpInterfaceABC();
        intf.showA();
        intf.showB();
        intf.showC();
    }
}
```

5.4 实践指导

5.4.1 继承性的使用

1. 设计一个表示多种形状的 Shape 类

假如需要写一个处理几何形状面积和周长的 Shape 类,要描述的几何形状可能是矩形、圆形或其他形状。要设计这些类,可以使用“是什么”“有什么”和“做什么”的方法决定成员变量、成员方法和类的层次。

1)一种几何形状

- “有什么”:名称、面积和周长;
- “做什么”:显示它的属性。

2)矩形

- “是什么”:一种称为矩形的形状;
- “有什么”:长和宽;
- “做什么”:计算面积和周长。

3)圆

- “是什么”:一种称为圆的形状;
- “有什么”:半径;
- “做什么”:计算面积和周长。

与其创建毫不相关的几个类,不如先好好探索这些类之间是否有相关之处。稍作分析,可以得到如图 5.2 所示的层次结构。

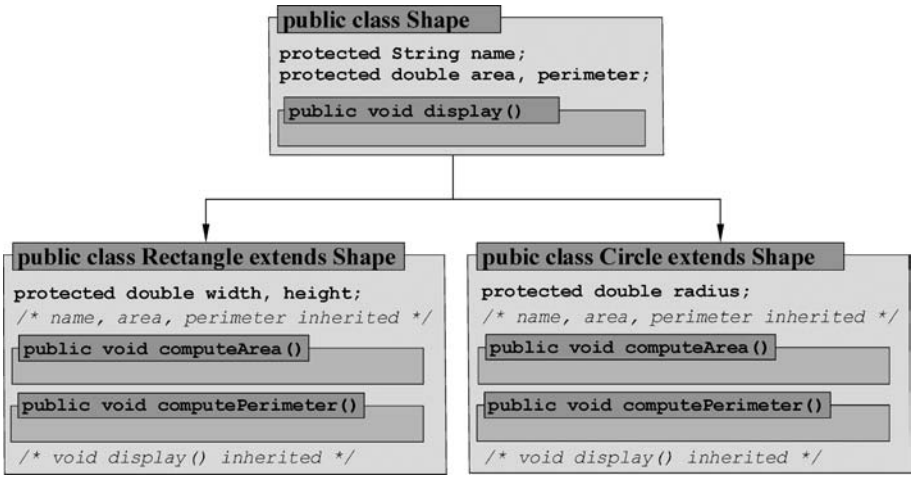

图 5.2 Shape 类的层次结构

因此,Shape 类作为超类,它有 3 个成员变量(名称、面积、周长)和两个成员方法(构造方法和显示方法)。为了能够实现继承性,这些成员都不能是私有的,因此把成员变量作为保护类型,把成员方法作为公有类型。

```
public class Shape
{  protected String name;
   protected double area, perimeter;
   public Shape()
   {  name = "undetermined";
      area = perimeter = 0;
   }
   public void display()
   {  System.out.println("Name: " + name);
      System.out.println("Area: " + area);
      System.out.println("Perimeter: " + perimeter);
   }
}
```

当不知道具体的形状时,不能计算面积和周长。矩形和圆形都是几何形状,把它们作为 Shape 类的子类,两个类都在构造方法中初始化自己的成员变量,在成员方法中计算面积和周长。

```
public class Rectangle extends Shape
{  protected double length, width;
   public Rectangle(double _length, double _width)
   {  name = "Rectangle";
        length = _length;
        width = _width;
   }
   public void computeArea()
    {  area = length * width; }
   public void computePerimeter()
    {  perimeter = 2 * (length + width); }
}
```

```
public class Circle extends Shape
{  protected double radius;
   public Circle(double _radius)
   {  name = "Circle";
      radius = _radius;
   }
   public void computeArea()
     {  area = Math.PI * radius * radius; }
   public void computePerimeter()
     {  perimeter = 2 * Math.PI * radius; }
}
```

用下面的 ShapeTester 类测试以上这些类,图 5.3 显示了 ShapeTester 类的执行结果。

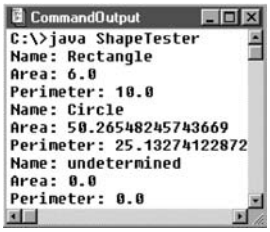

图 5.3 ShapeTester 类的执行结果

```
public class ShapeTester
{  public static void main(String args[])
   {  Shape s = new Shape();
      Rectangle r = new Rectangle(2.0, 3.0);
      Circle c = new Circle(4.0);
      r.computeArea();
      r.computePerimeter();
      c.computeArea();
      c.computePerimeter();
      r.display();c.display(); s.display();
   }
}
```

继承的一个好处是容易生成新的子类。在 Shape 类的基础上很容易实现正方形的类 Square。但正方形是一种特殊的矩形,所以它应当从 Rectangle 类中继承而不是 Shape 类。

2. 为 Shape 类的层次添加一个 Square 类

现在从 Rectangle 类中扩展出 Square 类。因为子类所需的成员都已经具备了,只需要适当地调用 Rectangle 的构造方法即可。

```
public class Square extends Rectangle
{  public Square(double _side)
   {  super(_side, _side);
      name = "Square";
   }
}
```

5.4.2 覆盖的使用

在超类的方法和变量不适合子类时,子类可以对这些变量和方法进行重新定义,即覆盖原来的定义。因为覆盖可以使超类中的成员方法被重新定义,它将很容易增强一个类的功能。

在前面的 Rectangle 和 Circle 类都使用从 Shape 类中继承而来的 display()方法,但是该方法不能显示新类中的特殊信息,下面要重新定义 Rectangle 类中的 display()方法,以便

能够输出矩形的长和宽。

Rectangle 类已经从 Shape 类中继承了 display()方法，现在使用同样的方法名在新类中重新定义 display()方法，显示名称、面积和周长。为了避免代码重复，在覆盖版本的方法中使用它，并增加代码显示矩形的长和宽。

```
public class Rectangle extends Shape
{  protected double length, width;
   public Rectangle(double _length, double _width)
   { /* 参考前面例子 */ }
   public void computeArea()
   { /* 参考前面例子 */ }
   public void display()
   {  super.display();
      System.out.println("Length: " + length);
      System.out.println("Width: " + width);
   }
}
```

5.4.3 抽象类和抽象方法的使用

1. 带抽象方法的 Shape 类

前面定义 Shape 类有成员变量 area，但是没有计算面积的方法，因为它没有具体几何形状的信息。在此将定义一个要计算面积的抽象方法。

定义抽象方法是一件很容易的事，因为它不要求有方法体。当一个抽象方法被添加到一个类中以后，该类也必须声明为抽象类。

```
public abstract class Shape
{  protected String name;
   protected double area, perimeter;
   public void display()
   {  /* 参考前面例子 */ }
   public abstract void computeArea();
}
```

现在不能实例化 Shape 类的对象了，因此前面的 ShapeTester 类无法通过编译，还需进一步修改。

2. 在 Shape 类层中使用多态性

重新设计 Shape、Rectangle、Circle 和 Square 类，使它们能利用多态性计算面积，并显示出来。

现在要利用多态性，确保每种形状分别用不同的方法来计算它们的面积和周长，因此超类 Shape 包含抽象方法 computeArea()，然后在不同的子类中实现和覆盖这个方法，同时添加 toString()方法来显示几何形状的一些基本属性。下面是这些类的实现。

```
public abstract class Shape
{  protected String type;
   protected double area, perim;
   public abstract void computeArea();
```

```
    public String toString()
    {  return type + ":\n\tArea = " + area + "\n\tPerimeter = " + perim;
    }
}
public class Rectangle extends Shape
{  protected double width, height;
    public Rectangle(double _width,
                     double _height)
    {  type = "Rectangle";
       width = _width;
       height = _height;
    }
    public void computeArea()
    {  area = width * height; }
    public void computePerimeter()
    {  perim = 2 * width + 2 * height; }
    public String toString()
    {  return super.toString() +
          "\n\tWidth = " + width +
          "\n\tHeight = " + height;
    }
}
public class Circle extends Shape
{  protected double radius;
    public Circle(double _radius)
    {  type = "Circle";
       radius = _radius;
    }
    public void computeArea()
    {  area = Math.PI * radius * radius; }
    public void computePerimeter()
    {  perim = 2.0 * Math.PI * radius; }
    public String toString()
    {  return super.toString() +
             "\n\tRadius = " + radius;
    }
}
public class Square extends Rectangle
{  public Square(double _side)
    { super(_side, _side);
       type = "Square";
    }
}
```

有了上面这些类,就能很容易创建它们的实例,并多态地选择适当的方法进行计算。

```
public class ShapeTestWithPolymorphism
{  public static void main(String args[])
    {  Shape shapes[] = { new Rectangle(2, 4), new Circle(3), new Square(4)};
       for (int i = 0; i < shapes.length; i++)
       {  shapes[i].computeArea();
```

```
            System.out.println(shapes[i]);
        }
    }
}
```

执行 ShapeTestWithPolymorphism 的输出如图 5.4 所示。

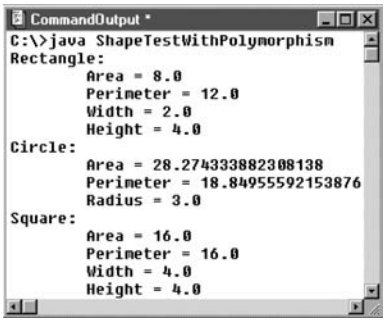

图 5.4 ShapeTestWithPolymorphism 的执行结果