**信息科学与工程学院**

**2022－2023学年第一学期**

实 验 报 告

课程名称： Java编程技术

实验名称： 第四次实验（B）

专 业 班 级 通信工程三班

学 生 学 号 202000120202

学 生 姓 名 李鑫

实 验 时 间 2022年11月5日

实验报告

【实验目的】

1. 强化使用java 对继承、接口的使用

2. 深化对类和对象的理解

【实验要求】

1. 抽象类的练习：编写一个程序，在程序中定义一个抽象类shape，再定义两个抽象方法Area和printArea，定义两个shape类的子类Rectangle和Circle类，在子类中实现父类的抽象方法。

2. 编写下面程序写出运行结果，谈谈成员变量的继承与隐藏，方法的覆盖与重载

|  |
| --- |
| public **class** Homework4B\_2 **{**  public static **void** main**(**String**[]** args**)** **{**  B m=**new** B**()**;  m.sum1**()**;  m.sum2**()**;  m.sum2**(**50**)**;  m.sum3**(**50**)**;  **}**  **}**  **class** A**{**  int sum,num1,num2;  public A**()** **{**  sum=0;  num1=10;  num2=20;  **}**  **void** sum1**()** **{**  sum=num1+num2;  System.out.println**(**"sum="+num1+"+"+num2+"="+sum**)**;  **}**  **void** sum2**(**int n**)** **{**  num1=n;  sum=num1+num2;  System.out.println**(**"sum="+num1+"+"+num2+"="+sum**)**;  **}**  **}**  **class** B **extends** A**{**  int num2;  public B**()** **{**  num2=200;  **}**  **void** sum2**()** **{**  sum=num1+num2;  System.out.println**(**"sum="+num1+"+"+num2+"="+sum**)**;  **}**  **void** sum2**(**int n**)** **{**  num1=n;  sum=num1+num2;  System.out.println**(**"sum="+num1+"+"+num2+"="+sum**)**;  **}**  **void** sum3**(**int n**)** **{**  super.sum2**(**n**)**;  **}**  **}** |

3. 理解下面3个类的定义，分析它们之间的关系，写出运行结果

|  |
| --- |
| public class Homework4B\_3 {  public static void main(String[] args) {  SubClass subC=new SubClass();  subC.doSomething();  }  }  class SuperClass{  int x;  SuperClass() {  x=3;  System.out.println("in SuperClass:x="+x);  }  void doSomething() {  System.out.println("in SuperClass.doSomething()");  }  }  class SubClass extends SuperClass{  int x;  SubClass() {  super();  x=5;  System.out.println("in SubClass:x="+x);  }  void doSomething() {  super.doSomething();  System.out.println("in SubClass.doSomething()");  System.out.println("super.x="+super.x+" sub.x="+x);  }  } |

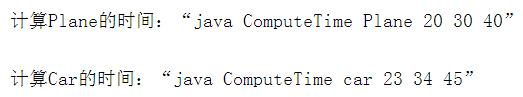
4.接口和继承的综合应用

首先定义一一个接口(IShapeArea)， 其中包含返回面积的方法 (getArea)。 然后定义一个矩形类 ( Rectangle) 和一-个圆类(Circle)， 并派生出一个正方形类(Square)， 即正方形类的父类为矩形类， 三者都要求实现接口IShapeArea， 自行扩充成员变量和方法(比如周长和面积)。

在主方法main中建一数组， 数组中放入一些上述类型的对象，并计算它们的周长和面积之和。

5. 接口的练习

为某研究所编写一个通用程序，用来计算每一种交通工具运行1000米所需的时间，已知每种交通工具的参数都是3个整数A、B、C的表达式。现有两种工具：Car和Plane，其中Car的速度运算公式为：A\*B/C，Plane的速度运算公式为：A+B+C。需要编写4类：ComputeTime、Plane、Car和接口（ICommon），要求在未来如果增加第3种交通工具的时候，不必修改以前的任何程序，只需要编写新的交通工具的程序即可，其运行过程如下，从命令行输入ComputeTime的四个参数，第一个是交通工具的类型，第二、 三、四个参数分别是整数A、B、C，我们在Dos下运行时输入如下：



【实验具体内容】

1. 抽象类的练习

编写一个程序，在程序中定义一个抽象类Shape，再定义两个抽象方法Area和printArea，定义两个Shape类的子类Rectangle和Circle类，在子类中实现父类的抽象方法。

【实验代码】

|  |
| --- |
| package Homework4b; *//注意文件层次，如果在src文件夹下，可以不用写package,但是如果把java文件放在了某一文件夹下， // 比如Homework4b文件夹下面，这时候就需要在最开始声明package Homework4b  // 主类，主要用于实例化其他类的调用其他类的方法* public class Homework4B\_1 {  public static void main(String[] args) {  *//实例化Rectangle的对象并传参width = 5, height = 6* Rectangle R = new Rectangle(5, 6);  *//调用Rectangle的方法printArea,打印实例化出来的R的矩形面积* R.printArea();  *//实例化Circle的对象并传参 radius =2* Circle C = new Circle(2);  *//调用Circle的方法printArea* C.printArea();  } }  *//抽象类Shape* abstract class Shape {  *//定义抽象方法Area计算面积* abstract double Area();   *//定义抽象方法printArea输出面积* abstract void printArea(); }  *//Shape类的子类Rectangle类，实现父类的抽象方法* class Rectangle extends Shape {  int width, height;   *//构造方法* public Rectangle(int width, int height) {  this.height = height;  this.width = width;  }   *//Area方法计算矩形面积* public double Area() {  return height \* width;  }   *//printArea方法输出矩形面积* public void printArea() {  System.*out*.println("矩形的面积为：" + Area());  } }  *//Shape类的子类Circle类，实现父类的抽象方法* class Circle extends Shape {  int radius;   *//构造方法* public Circle(int radius) {  this.radius = radius;  }   *//Area方法计算圆形面积* public double Area() {  return radius \* radius \* 3.1415926;  }   *//printArea方法输出圆的面积* public void printArea() {  System.*out*.println("圆的面积为：" + Area());  } } |

【实验流程图】

|  |
| --- |
| C:\Users\李鑫\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\3DB712AB.tmp |

【实验结果截图】

|  |
| --- |
|  |

2. 编写下面程序写出运行结果，谈谈成员变量的继承与隐藏，方法的覆盖与重载

【实验收获】

|  |
| --- |
| 1. copyof 方法是 Array 类的方法，所以在使用这个方法之前，要import java.util.Arrays。它可以指定复制数字元素的个数，也可以指定起止位置。使用例子如下： b1 = Arrays.*copyOf*(a1, 10); |
| 2.以 arrayCopy( arr1, 2, arr2, 5, 10)为例，就实现了 arr1 数组从位置为 2 的元素开始，复制到 arr2 中位置是 5 的地方，并且复制的元素的个数是 10 个，2为源操作位置，5为目标操作位置，10为操作长度。arr1为源操作对象,arr2为目的操作对象 |

【实验代码】

|  |
| --- |
| package Homework4b; *// 主类* public class Homework4B\_2 {  public static void main(String[] args) {  *//实例化B的对象m* B m=new B();  *//调用父类中的sum1方法,方法继承* m.sum1();  *//调用sum2方法* m.sum2();  *//传参50* m.sum2(50);  *//传参50* m.sum3(50);  } }  *// 定义A类* class A{  *//A类中定义sum,num1,num2* int sum,num1,num2;  *//构造方法* public A() {  sum=0;  num1=10;  num2=20;  }  *//定义sum1方法* void sum1() {  sum=num1+num2;  System.*out*.println("sum="+num1+"+"+num2+"="+sum);  }  *//定义sum2方法* void sum2(int n) {  num1=n;  sum=num1+num2;  System.*out*.println("sum="+num1+"+"+num2+"="+sum);  } } *//B类继承A类* class B extends A{  int num2;  *//构造方法,对num2重新赋值为200* public B() {  num2=200;  }  void sum2() {  sum=num1+num2;  System.*out*.println("sum="+num1+"+"+num2+"="+sum);  }  *//重写sum2()* void sum2(int n) {  num1=n;  sum=num1+num2;  System.*out*.println("sum="+num1+"+"+num2+"="+sum);  }  void sum3(int n) {  *//调用父类的sum2方法,传参n* super.sum2(n);  } } |

【实验结果截图】

|  |
| --- |
|  |

3.理解下面3个类的定义，分析它们之间的关系，写出运行结果。

【类间关系】

|  |
| --- |
| 从代码中可以看出  先创建了一个主类SuperClass，在里面又实现了两个方法分别是SuperClass()和doSomething()  之后通过继承SuperClass得到子类SubClass，在其中重写了父类的方法doSomething |

【实验代码】

|  |
| --- |
| *//package声明* package Homework4b; *//主类* public class Homework4B\_3 {  public static void main(String[] args) {  *//实例化无参的对象，调用无参构造方法* SubClass subC=new SubClass();  *//调用对象subC对象所在类的方法doSomething()* subC.doSomething();  } } *//定义SuperClass类作为父类* class SuperClass{  int x;  *//父类的无参构造方法  //调用输出“in SuperClass:x=3”* SuperClass() {  x=3;  System.*out*.println("in SuperClass:x="+x);  }  *//父类方法  //调用输出"in SuperClass.doSomething()"* void doSomething() {  System.*out*.println("in SuperClass.doSomething()");  } } *// SubClass继承于SuperClass类* class SubClass extends SuperClass{  int x;  *//子类的无参构造方法* SubClass() {  *//调用父类的无参构造函数* super();  x=5;  *//调用输出“in SubClass:x=5”* System.*out*.println("in SubClass:x="+x);  }  *//定义子类的方法，对父类方法进行重写* void doSomething() {  *//调用父类的doSomething方法* super.doSomething();  System.*out*.println("in SubClass.doSomething()");  System.*out*.println("super.x="+super.x+" sub.x="+x);  }  } |

【实验结果截图】

|  |
| --- |
|  |

4. 接口和继承的综合应用

首先定义一一个接口(IShapeArea)， 其中包含返回面积的方法 (getArea)。 然后定义一个矩形类 ( Rectangle) 和一-个圆类(Circle)， 并派生出一个正方形类(Square)， 即正方形类的父类为矩形类， 三者都要求实现接口IShapeArea， 自行扩充成员变量和方法(比如周长和面积)。

在主方法main中建一数组， 数组中放入一些上述类型的对象，并计算它们的周长和面积之和。

【实验收获】

|  |
| --- |
| 该实验和我们C#实验的一段代码很像,所以在写代码的时候我很多都是参考的之前写的C#代码，通过将C#代码转换为Java代码使得我对面向对象有了更深一层的理解，同时对Java和C#的理解也上了一层。 |
|  |

【实验代码】

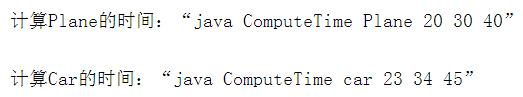
|  |
| --- |
| package Homework4b;  public class Homework4B\_4 {  public static void main(String[] args) {  *//数组A存放参数对象* double A[]= {2,4,6,8};  *//实例化对象,传参A[0]和A[1]中的值* Rectangle1 R = new Rectangle1(A[0],A[1]);  *//实例化对象,传参A[2]中的值* Circle1 C = new Circle1(A[2]);  *//实例化对象,传参A[3]中的值* Square S = new Square(A[3]);  *//输出面积* R.printArea();  C.printArea();  S.printArea();  *//计算面积之和* double sumArea = R.getArea()+C.getArea()+S.getArea();  *//输出面积之和* System.*out*.println("面积之和为：" + sumArea);  *//换行* System.*out*.println();  *//输出周长* R.printPerimeter();  C.printPerimeter();  S.getPerimeter();  *//计算周长之和* double sumPerimeter = R.getPerimeter()+C.getPerimeter()+S.getPerimeter();  *//输出周长之和* System.*out*.println("周长之和为：" + sumPerimeter);  } } *//抽象接口* abstract interface IShapeArea {  abstract double getArea(); } *//类Rectangle1实现接口IShapeArea* class Rectangle1 implements IShapeArea {  double width,height;  *//含参构造方法* public Rectangle1(double width,double height) {  this.width = width;  this.height = height;  }  *//返回面积* public double getArea() {  return width \* height;  }  *//返回周长* public double getPerimeter() {  return (width + height)\*2;  }  *//输出面积* public void printArea() {  System.*out*.println("矩形的面积为：" + this.getArea());  }  *//输出周长* public void printPerimeter() {  System.*out*.println("矩形的周长为：" + this.getPerimeter());  } } *//类Circle1实现接口IShapeArea* class Circle1 implements IShapeArea {  double radius;  *//含参构造方法* public Circle1(double radius) {  this.radius = radius;  }  *//返回面积* public double getArea() {  return radius \* radius \* 3.14159265;  }  *//返回周长* public double getPerimeter() {  return radius\*3.14\*2;  }  *//输出面积* public void printArea() {  System.*out*.println("圆的面积为：" + this.getArea());  }  *//输出周长* public void printPerimeter() {  System.*out*.println("圆的周长为：" + this.getPerimeter());  } } *//类Square继承类Rectangle1* class Square extends Rectangle1 {  double side;  *//含参构造方法* public Square(double side) {  super(side, side);  this.side = side;  }  *//返回面积* public double getArea() {  return side \* side;  }  *//返回周长* public double getPerimeter() {  return (side + side)\*2;  }  *//输出面积* public void printArea() {  System.*out*.println("正方形的面积为：" + this.getArea());  }  *//输出周长* public void printPerimeter() {  System.*out*.println("正方形的周长为：" + this.getPerimeter());  } } |

【实验运行截图】

|  |
| --- |
|  |

5. 接口的练习

为某研究所编写一个通用程序，用来计算每一种交通工具运行1000米所需的时间，已知每种交通工具的参数都是3个整数A、B、C的表达式。现有两种工具：Car和Plane，其中Car的速度运算公式为：A\*B/C，Plane的速度运算公式为：A+B+C。需要编写4类：ComputeTime、Plane、Car和接口（ICommon），要求在未来如果增加第3种交通工具的时候，不必修改以前的任何程序，只需要编写新的交通工具的程序即可，其运行过程如下，从命令行输入ComputeTime的四个参数，第一个是交通工具的类型，第二、 三、四个参数分别是整数A、B、C，我们在Dos下运行时输入如下：



【涉及知识点】

|  |
| --- |
| 接口并不是类，编写接口的方式和类很相似，但是它们属于不同的概念。类描述对象的属性和方法。接口则包含类要实现的方法。  除非实现接口的类是抽象类，否则该类要定义接口中的所有方法。  接口无法被实例化，但是可以被实现。一个实现接口的类，必须实现接口内所描述的所有方法，否则就必须声明为抽象类。另外，在 Java 中，接口类型可用来声明一个变量，他们可以成为一个空指针，或是被绑定在一个以此接口实现的对象。  当类实现接口的时候，类要实现接口中所有的方法。否则，类必须声明为抽象的类。  类使用implements关键字实现接口。在类声明中，Implements关键字放在class声明后面。  一个接口能继承另一个接口，和类之间的继承方式比较相似。接口的继承使用extends关键字，子接口继承父接口的方法 |

【实验代码】

|  |
| --- |
| package Homework4b.work5; // 导包 import javax.swing.\*; import java.lang.reflect.Constructor; import java.lang.reflect.Field; import java.lang.reflect.InvocationTargetException; import java.lang.reflect.Method; // 定义接口 interface ICommon {  float getSpeed(); }  class Plane implements ICommon {   float A,B,C;  public Plane(float a, float b, float c) {  A = a;  B = b;  C = c;  }   public float getA() {  return A;  }   public void setA(float a) {  A = a;  }   public float getB() {  return B;  }   public void setB(float b) {  B = b;  }   public float getC() {  return C;  }   public void setC(float c) {  C = c;  }  //方法重载  @Override  public float getSpeed() {  return A+B+C;  } }  //从Plane继承得到子类Car class Car extends Plane {   public Car(float a, float b, float c) {  super(a,b,c);  }   @Override  public float getA() {  return super.getA();  }   @Override  public void setA(float a) {  super.setA(a);  }   @Override  public float getB() {  return super.getB();  }   @Override  public void setB(float b) {  super.setB(b);  }   @Override  public float getC() {  return super.getC();  }   @Override  public void setC(float c) {  super.setC(c);  }   @Override  public float getSpeed() {  return A\*B/C;  } } public class ComputeTime {  @SuppressWarnings("unchecked")  public static void main(String args[])  // 捕获异常  throws ClassNotFoundException, NoSuchMethodException, IllegalAccessException, InvocationTargetException, InstantiationException, NoSuchFieldException {  String s = "Homework4b.work5."+args[0];  float Atemp = Float.*parseFloat*(args[1]);  float Btemp = Float.*parseFloat*(args[2]);  float Ctemp = Float.*parseFloat*(args[3]);  *//获取类* Class Trans = Class.*forName*(s);  *//获取类的指定构造器* Constructor constructor = Trans.getDeclaredConstructor(float.class,float.class,float.class);  *//调用类的有参构造* ICommon trans = (ICommon) constructor.newInstance(Atemp,Btemp,Ctemp);   Method getA = Trans.getDeclaredMethod("getA");  Method getB = Trans.getDeclaredMethod("getB");  Method getC = Trans.getDeclaredMethod("getC");  Method getSpeed = Trans.getDeclaredMethod("getSpeed");  float A = (float)getA.invoke(trans);  float B = (float)getB.invoke(trans);  float C = (float)getC.invoke(trans);  float speed = (float) getSpeed.invoke(trans);  System.*out*.println("A:"+A+"\tB:"+B+"\tC"+C);  System.*out*.println("Speed of "+args[0]+" is "+speed);  } } |

【实验运行结果截图】

|  |
| --- |
|  |

Class.forName：返回与给定的字符串名称相关联类或接口的Class对象。

Class.forName是一个静态方法，可以用来加载类。该方法有两种形式：1.Class.forName(String name, boolean initialize, ClassLoader loader)

name表示的是类的全名；initialize表示是否初始化类；loader表示加载时使用的类加载器。

2.Class.forName(String className)。

默认相当于设置了参数initialize的值为true，loader的值为当前类的类加载器。

使用Class.forName( )静态方法的目的是为了动态加载类，如果在类中有静态初始化器的话，JVM必然会执行该类的静态代码段。通常在加载完成后，还要调用Class下的newInstance( )静态方法来实例化对象以便操作。newInstance()实际上是把new这个方式分解为两步，即首先调用Class加载方法加载某个类，然后实例化。

Class.forName(“”)返回的是类。  
Class.forName(“”).newInstance()返回的是object 。

在指定类给 class.forName() 方法后，如果找不到指定的类，会抛出ClassNotFoundException 异常，所以在使用时要进行异常处理，否则编译器会提示“Unhandled exception type Exception”错误警告。



可以用try-catch-finally或者throw throws进行一场处理，我选的是catch。一般要catch详细的异常，不同的异常有不同的解决方法。因为我对各种异常不是非常熟悉，所以选择用try–catch(Exception e)解决所有异常。

**try** {  
需要检查的代码（**可能会抛出异常，也可能不会抛出异常**）  
}  
**catch**(异常的类型 异常类型的变量) {  
捕获异常后要处理异常  
**没有抛出异常就不执行catch**（可以抛出多个catch）  
}  
**finally** {  
**一定会被执行的代码**（不管异常抛不抛出都会执行），一般收尾  
}

在查资料时了解到了JAVA的反射机制，反射是一种动态的方法使得在任意位置能够动态获取类信息，从而通过类信息获取类的成员变量和调用成员方法。

Java反射就是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意方法和属性；并且能改变它的属性。这也是Java被视为动态语言的一个关键性质。反射机制允许程序在运行时取得任何一个已知名称的class的内部信息，包括包括其modifiers(修饰符)，fields(属性)，methods(方法)等，并可于运行时改变fields内容或调用methods。得到class的三种方法：

1、通过对象调用 getClass() 方法来获取

2、直接通过 类名.class 的方式得到

3、通过 Class 对象的 forName() 静态方法来获取，用的最多，但可能抛出 ClassNotFoundException 异常。

## 1.接口

接口的作用是实现多重继承，一个类只能继承一个父类，但是可以实现一个或多个接口。接口的目的是指明相关或者不相关类的多个对象的共同行为，跟抽象类很相似，可以说接口是更加抽象的抽象类。

注意：

1. 接口不能被实例化。
2. 接口中所有方法不能有主体。
3. 一个类可以实现多个接口。
4. 接口可以有变量，但一定是public static final 修饰，如果不修饰 JVM也会默认是public static final。
5. 接口中的方法只能是public修饰。
6. 一个接口不能继承其他类，但是可以继承别的接口

好处：

在项目中，有一些功能，不仅仅是一个类去实现，即有很多地方有用到，大家需要统一标准。所以需要用接口，这样大家同时实现一个接口，就能够统一规范。在接口中只是定义了基本的方法名，具体的实现交给实现类。接口的使用可以使代码简洁、同一方法的实现可以多种多样、降低代码的耦合性，同为规范化可以降低维护成本。

假设张三、李四同时开发一个模块。张三负责写收银员类和方法，李四负责写计算器类和方法。如果没有接口，那么张三写好了之后，需要调用李四的方法，想当然的认为方法名是jisuan（），结果却发现李四的方法名是countMoney（），这个时候就需要去修改代码了，带来了不必要的麻烦。在日常生活中，两个实体之间进行连接的部分称为接口。如电脑和U盘连接的标准USB接口。接口可以确保不同实体之间的顺利连接。不同的电脑厂家和U盘厂家只要按照相同的USB接口进行生产，那么所有的电脑和U盘就可以顺利的连接起来。

【接口和抽象类的区别】

1.语法层面上的区别

抽象类:  
         1）抽象类不可以实例化，只能够用来继承  
         2）抽象类的子类应提供对其所有抽象方法的实现。如果子没有实现它的全部方法 ，可以将子类定义为抽象类  
         3）抽象类中可以包含非抽象的方法  
         4）抽象类的引用可指向子类的实例  
    
    接口：  
         1）接口用interface关键修饰。接口中所有的方法为抽象方法 ,注明：jdk8之后提供了默认方法和静态方法（且这两种方法可以存在方法体）  
         2）接口中的变量都为常量  
         3）接口不能实例化，只被实现,还能被接口继承  
         4）接口的实现类必须全部实现接口中的方法，如果不实现，可以将子类变成一个抽象类  
         5）接口可以多实现  
         6）接口可多继承  
         7）接口的引用指向实现类的实例

2.设计层面上的区别

　　1）抽象类是对一种事物的抽象，即对类抽象，而接口是对行为的抽象。抽象类是对整个类整体进行抽象，包括属性、行为，但是接口却是对类局部（行为）进行抽象。举个简单的例子，飞机和鸟是不同类的事物，但是它们都有一个共性，就是都会飞。那么在设计的时候，可以将飞机设计为一个类Airplane，将鸟设计为一个类Bird，但是不能将飞行这个特性也设计为类，因此它只是一个行为特性，并不是对一类事物的抽象描述。此时可以将飞行设计为一个接口Fly，包含方法fly( )，然后Airplane和Bird分别根据自己的需要实现Fly这个接口。然后至于有不同种类的飞机，比如战斗机、民用飞机等直接继承Airplane即可，对于鸟也是类似的，不同种类的鸟直接继承Bird类即可。从这里可以看出，继承是一个 "是不是"的关系，而接口实现则是 "有没有"的关系。如果一个类继承了某个抽象类，则子类必定是抽象类的种类，而接口实现则是有没有、具备不具备的关系，比如鸟是否能飞（或者是否具备飞行这个特点），能飞行则可以实现这个接口，不能飞行就不实现这个接口。

　　2）设计层面不同，抽象类作为很多子类的父类，它是一种模板式设计。而接口是一种行为规范，它是一种辐射式设计。什么是模板式设计？最简单例子，大家都用过ppt里面的模板，如果用模板A设计了ppt B和ppt C，ppt B和ppt C公共的部分就是模板A了，如果它们的公共部分需要改动，则只需要改动模板A就可以了，不需要重新对ppt B和ppt C进行改动。而辐射式设计，比如某个电梯都装了某种报警器，一旦要更新报警器，就必须全部更新。也就是说对于抽象类，如果需要添加新的方法，可以直接在抽象类中添加具体的实现，子类可以不进行变更；而对于接口则不行，如果接口进行了变更，则所有实现这个接口的类都必须进行相应的改动。

## 2.上转型和下转型

转型是基于继承，所以先实现一个子类继承父类。

1. 向上转型upcasting

Father f1 = new Son();//f1引用指向一个Son对象

将基类（父类）引用转换为导出类（子类）引用的动作。通俗地讲即是将子类对象转为父类对象。此处父类对象可以是接口。

【问题】

子类在向上转型的过程中，容易出现子类方法丢失。因为向上转型之后，使用基类（父类）对象去调用方法，只能使用子类重写父类的方法、被子类隐藏的父类变量、子类未重写的父类方法、未被隐藏的父类成员变量，而不能使用子类新增成员(包括成员变量,成员方法)。

1. 向下转型

Father f1 = new Son();

Son s1 = (Son)f1; // 现在f1还是指向Son对象

向下转型必须是在已经向上转型的基础上进行，否则报错。

进行下转型后，对象能够调用子类的所有方法。即是把父类对象转为子类对象。为了保证向下转型的顺利完成,在java中提供了一个关键字:instanceof,通过instanceof可以判断某对象是否是某类的实例,如果是则返回true,否则为false。

【二者的理解】

内存中，因为向上转型是父类的引用指向为子类的对象，所以，它只是指向了父类应该拥有的属性和方法，而子类的独有的方法和属性就不指向了（或者说隐藏了），当再强转到子类时。又重新指向了子类对象，那属于它的东西又恢复了。

【好处】

代码变得简洁，体现了JAVA的抽象编程思想，以及继承、多态，有利于程序扩展，写出更易维护，简洁的代码，最大的用处是java的泛型编程。很多时候，我们需要把很多种类的实例对象，全部扔到一个集合。但是肯定不可能给他们每个种类都用一个独立的集合去存放吧，这个时候我们应该寻找到一个标准，接口就是一个标准。把很多种类的子类实例对象全部扔到存放父类实例的集合。经过了这个过程，子类实例已经赋值给了父类引用（即完成了向上转型），但很遗憾的丢失了子类扩展的方法。Java语言的向下转型的特性让我们可以重新获得丢失的方法，即强转回子类。所以我们需要用到子类实例的时候，就从那个父类集合里拿出来向下转型就可以了，一样可以使用子类实例对象。

上转型和下转型其实实际输出都是输出导出类（子类）的内容，不同的是，调用的对象不同，还有就是上转型的对象对造成方法丢失的情况。

java 转型问题其实并不复杂，只要记住一句话：父类引用指向子类对象。

1。父类引用指向子类对象，而子类引用不能指向父类对象。

2。把子类对象直接赋给父类引用叫upcasting向上转型，通过子类对象(小范围)实例化父类对象(大范围),向上转型不用强制转换。

3。把指向子类对象的父类引用赋给子类引用叫向下转型(downcasting)，通过父类对象(大范围)实例化子类对象(小范围),要强制转换。

【实验反思】

通过从本次实验我对面向对象编程又有了更新一层的认识：

当解决一个问题的时候，面向对象会把事物抽象成对象的概念，就是说这个问题里面有哪些对象，然后给对象赋一些属性和方法，然后让每个对象去执行自己的方法，问题得到解决。易维护、易复用、易扩展，由于面向对象有封装、继承、多态性的特性，可以设计出低耦合的系统，使系统 更加灵活、更加易于维护。

也了解到了面向对象编程的三大基本特性：封装，继承，多态

封装，就是把客观事物封装成抽象的类，并且类可以把自己的数据和方法只让可信的类或者对象操作，对不可信的进行信息隐藏。一个类就是一个封装了数据以及操作这些数据的代码的逻辑实体。在一个对象内部，某些代码或某些数据可以是私有的，不能被外界访问。通过这种方式，对象对内部数据提供了不同级别的保护，以防止程序中无关的部分意外的改变或错误的使用了对象的私有部分。

继承，指可以让某个类型的对象获得另一个类型的对象的属性的方法。它支持按级分类的概念。继承是指这样一种能力：它可以使用现有类的所有功能，并在无需重新编写原来的类的情况下对这些功能进行扩展。 通过继承创建的新类称为“子类”或“派生类”，被继承的类称为“基类”、“父类”或“超类”。继承的过程，就是从一般到特殊的过程。要实现继承，可以通过 “继承”（Inheritance）和“组合”（Composition）来实现。继承概念的实现方式有二类：实现继承与接口继承。实现继承是指直接使用父类的属性和方法而无需额外编码的能力；接口继承是指仅使用属性和方法的名称、但是子类必须提供实现的能力。

多态，是指一个类实例的相同方法在不同情形有不同表现形式。多态机制使具有不同内部结构的对象可以共享相同的外部接口。这意味着，虽然针对不同对象的具体操作不同，但通过一个公共的类，它们（那些操作）可以通过相同的方式予以调用。

接口（英文：Interface），在JAVA编程语言中是一个抽象类型，是抽象方法的集合，接口通常以interface来声明。一个类通过继承接口的方式，从而来继承接口的抽象方法。

接口并不是类，编写接口的方式和类很相似，但是它们属于不同的概念。类描述对象的属性和方法。接口则包含类要实现的方法。

除非实现接口的类是抽象类，否则该类要定义接口中的所有方法。

接口无法被实例化，但是可以被实现。一个实现接口的类，必须实现接口内所描述的所有方法，否则就必须声明为抽象类。另外，在 Java 中，接口类型可用来声明一个变量，他们可以成为一个空指针，或是被绑定在一个以此接口实现的对象。

接口与类相似点：

一个接口可以有多个方法。

接口文件保存在 .java 结尾的文件中，文件名使用接口名。

接口的字节码文件保存在 .class 结尾的文件中。

接口相应的字节码文件必须在与包名称相匹配的目录结构中。

接口与类的区别**：**

接口不能用于实例化对象。

接口没有构造方法。

接口中所有的方法必须是抽象方法，Java 8 之后 接口中可以使用 default 关键字修饰的非抽象方法。

接口不能包含成员变量，除了 static 和 final 变量。

接口不是被类继承了，而是要被类实现。

接口支持多继承。

同时在网上看到了之前有个学姐在写第五道题的时候报错，我感觉好像是因为她实例化的这个接口，然后产生了报错。