**信息科学与工程学院**

**2020－2021学年第一学期**

实 验 报 告

课程名称： Java编程技术

实验名称： 实验四 **(B)**

专 业 班 级 通信工程 二班

学 生 学 号 201800121050

学 生 姓 名 孟麟芝

实 验 时 间 2020年11月13日

实验报告

【实验目的】

1.继续熟练使用Java实现一些简单的算法。

2.掌握Java中类、对象等概念。

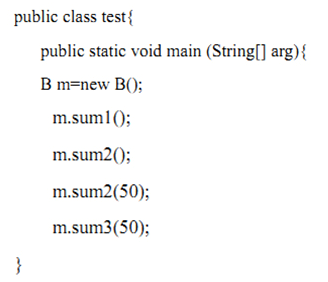
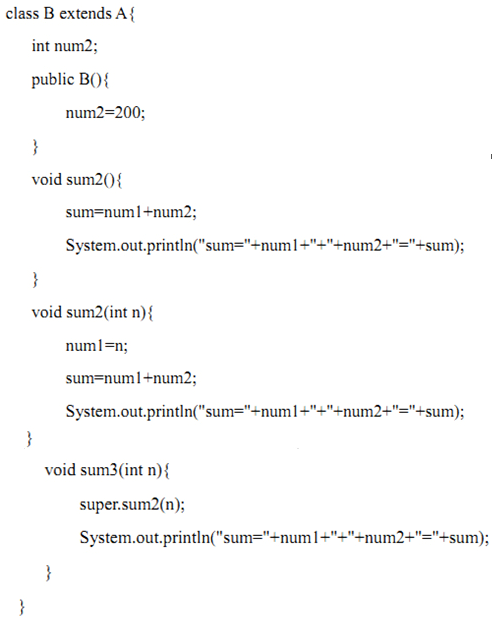
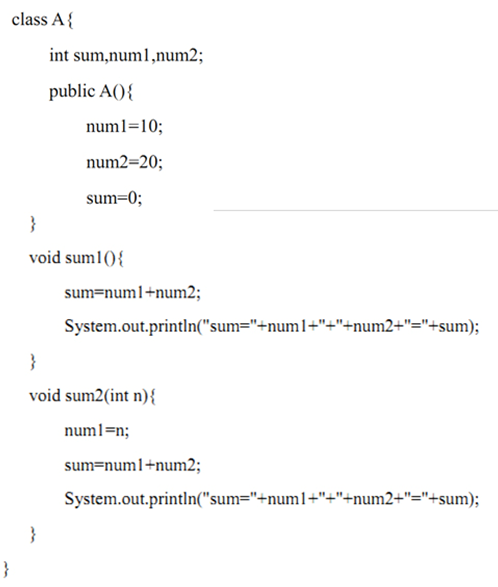
3.掌握Java中接口、抽象类的概念

4.了解Java中的反射机制。

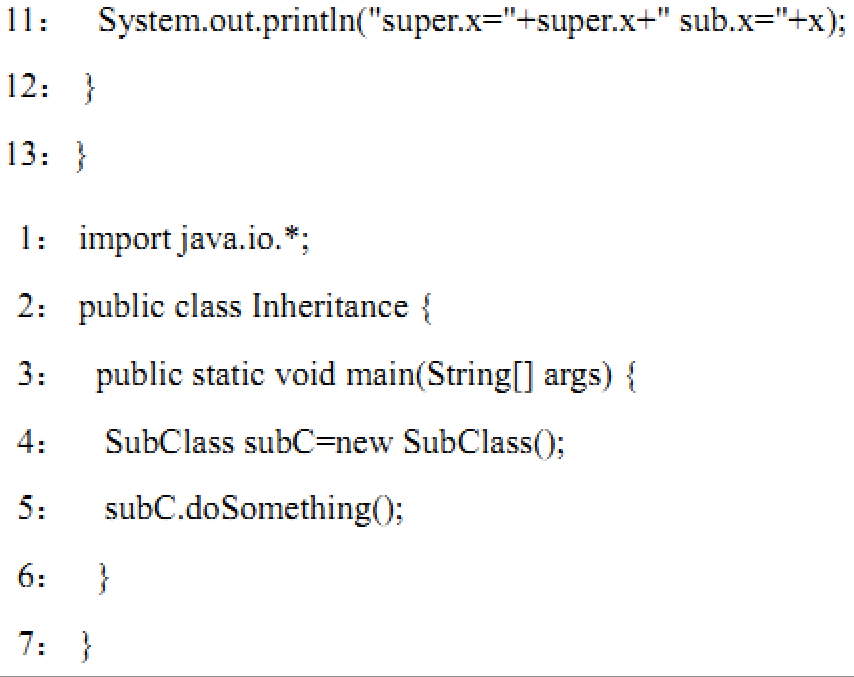
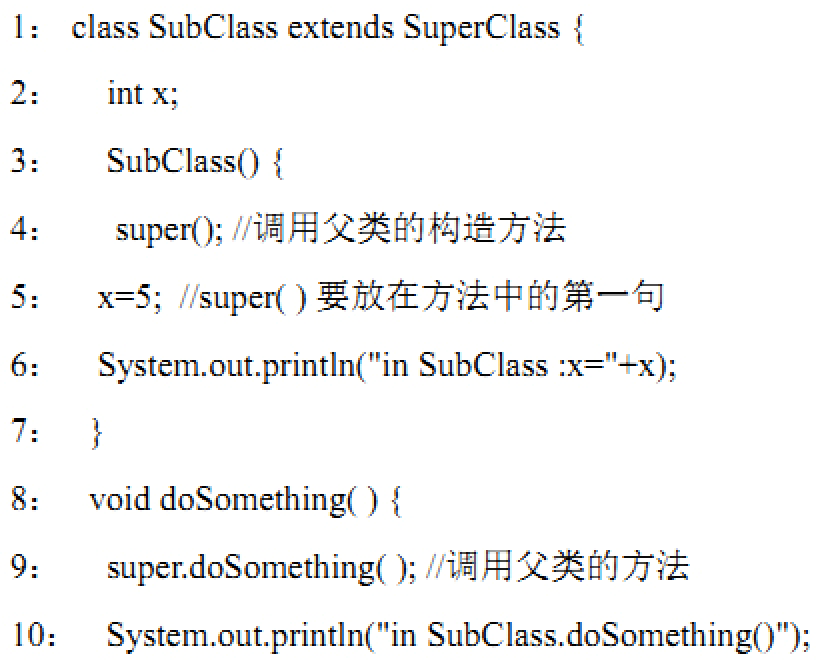
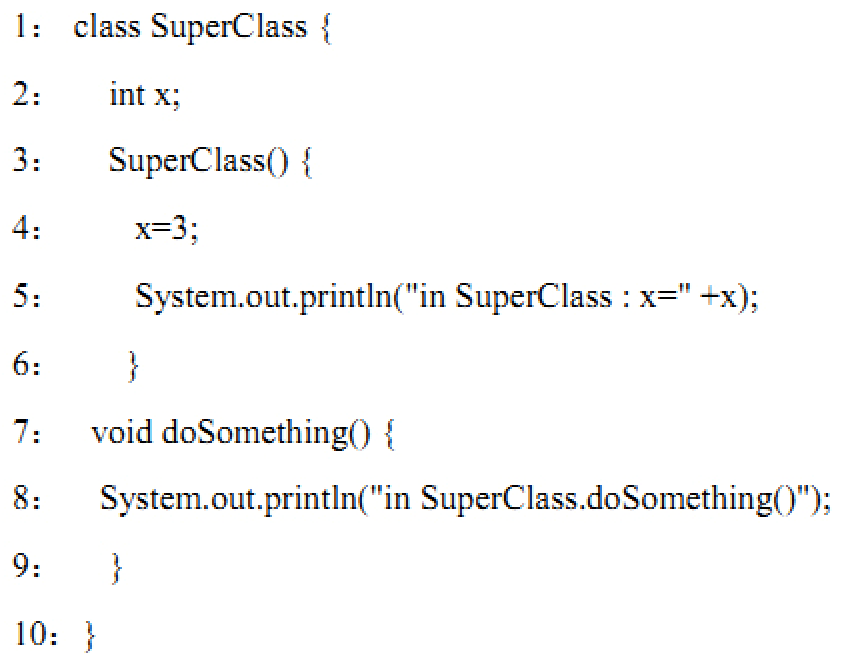
## 【实验要求】

1.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。

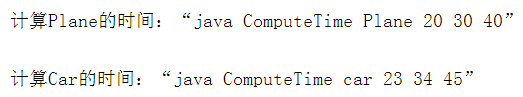


3.理解下面3个类的定义，分析它们之间的关系，写出运行结果。



4.首先定义一个接口(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.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：



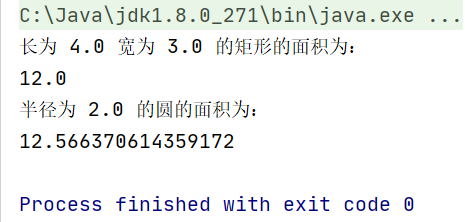
## 【第一个实验】

1.实验源代码

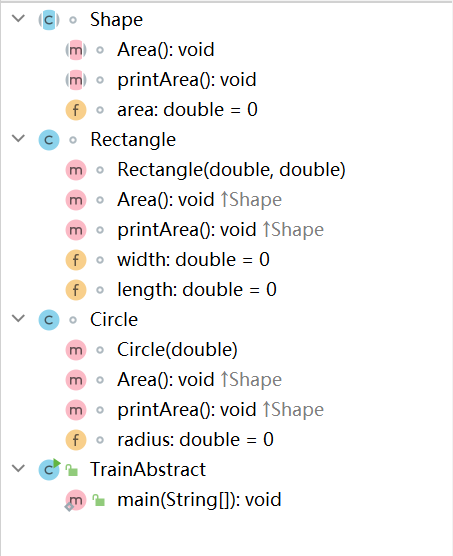
package classes.b;  
  
abstract class Shape {  
 double area = 0;  
  
 abstract void Area();  
  
 abstract void printArea();  
}  
  
class Rectangle extends Shape {  
 double width = 0;  
 double length = 0;  
  
 Rectangle(double width, double length) {  
 this.length = length;  
 this.width = width;  
 }  
  
 void Area() {  
 this.area = this.length \* this.width;  
 }  
  
 void printArea() {  
 System.*out*.println("长为 " + this.length + " 宽为 " + this.width + " 的矩形的面积为： ");  
 System.*out*.println(this.area);  
 }  
}  
  
class Circle extends Shape {  
 double radius = 0;  
  
 Circle(double radius) {  
 this.radius = radius;  
 }  
  
 void Area() {  
 this.area = Math.*PI* \* this.radius \* this.radius;  
 }  
  
 void printArea() {  
 System.*out*.println("半径为 " + this.radius + " 的圆的面积为： ");  
 System.*out*.println(this.area);  
 }  
}  
  
public class TrainAbstract {  
 public static void main(String[] args) {  
 Rectangle rect = new Rectangle(3, 4);  
 Circle cir = new Circle(2);  
 rect.Area();  
 rect.printArea();  
 cir.Area();  
 cir.printArea();  
 }  
}

2.实验记录与说明分析

下面是实验的结果，可见程序能实现相应的功能。



程序的结构如下：



在面向对象的概念中，所有的对象都是通过类来描绘的，但是反过来，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。

抽象类除了不能实例化对象之外，类的其它功能依然存在，成员变量、成员方法和构造方法的访问方式和普通类一样。抽象类中也可以声明非抽象的方法和成员变量，抽象类中甚至可以不含抽象成员，但含有抽象方法一定是抽象类。

继承抽象方法的子类必须重写（或者说成实现更容易理解）该方法。否则，该子类也必须声明为抽象类。最终，必须有子类实现该抽象方法，否则，从最初的父类到最终的子类都不能用来实例化对象。

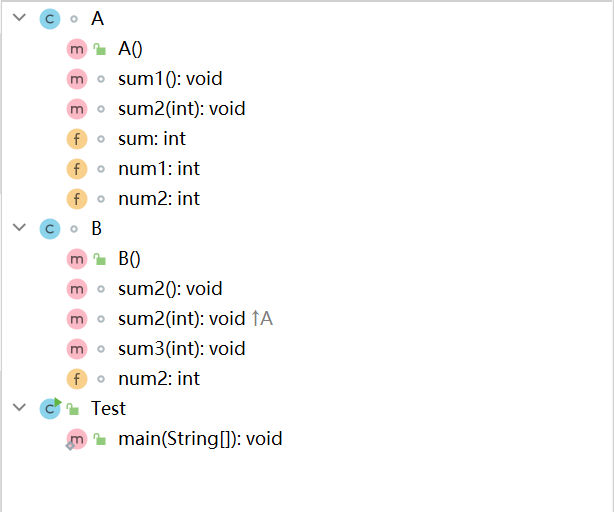
由于抽象类不能实例化对象，所以抽象类必须被继承，才能被使用。也是因为这个原因，通常在设计阶段决定要不要设计抽象类。

## 【第二个实验】

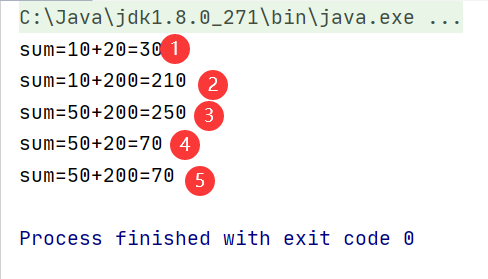
1.实验源代码

package classes.b;  
  
class A {  
 int sum, num1, num2;  
  
 public A() {  
 num1 = 10;  
 num2 = 20;  
 sum = 0;  
 }  
  
 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);  
 System.*out*.println("sum=" + num1 + "+" + num2 + "=" + sum);  
 }  
}  
  
public class Test {  
 public static void main(String[] args) {  
 B m = new B();  
 *//这里的sum1()是从父类继承而来的,未进行重写和重载，调用了父类中的属性和方法* m.sum1();  
 *//这里对sum2()进行了重载* m.sum2();  
 *//此方法是对父类里方法的重写* m.sum2(50);  
 *//这是一个新方法* m.sum3(50);  
 }  
}

2.实验记录与说明分析



程序结构如上图。



实验结果如上图所示，若子类中定义了和父类中一样名字的成员变量，则父类的成员则会被“隐藏”（而非覆盖），如何将其调出？则使用super关键字即可使之浮出水面。

或者还有另一种方法，如①的结果所示，若子类调用了继承自父类的方法，且该方法没有被重写或重载，这一方法也相当于是由父类来执行（相当于使用了super）的，使用的成员变量也是父类中的。

除去①的其他结果则都比较好理解。②是子类中重载父类方法的结果，③是子类中重写父类方法的结果，④则是对父类方法的调用，⑤则是由于在子类中调用成员变量且未加任何修饰符，那么调用的也是子类中的成员变量。

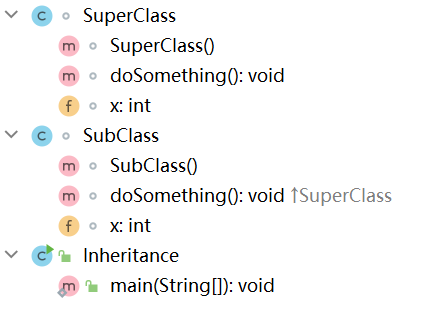
## 【第三个实验】

1.实验源代码

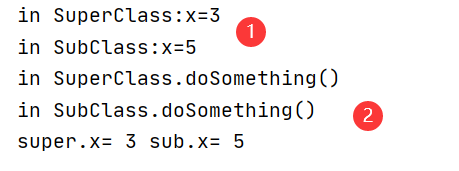
package classes.b;  
  
class SuperClass {  
 int x;  
  
 SuperClass() {  
 x = 3;  
 System.*out*.println("in SuperClass:x=" + x);  
 }  
  
 void doSomething() {  
 System.*out*.println("in SuperClass.doSomething()");  
 }  
}  
  
class SubClass extends SuperClass {  
 int x;  
  
 SubClass() {  
 super();*//调用父类的构造方法，其实不写也将会对父类构造* x = 5;*//super()要放在方法中的第一句* System.*out*.println("in SubClass:x=" + x);  
 }  
  
 void doSomething() {  
 super.doSomething();*//调用父类的方法* System.*out*.println("in SubClass.doSomething()");  
 System.*out*.println("super.x= " + super.x + " sub.x= " + x);  
 }  
}  
  
public class Inheritance {  
 public static void main(String[] args) {  
 SubClass subC = new SubClass();  
 subC.doSomething();  
 }  
}

2.实验记录与说明分析

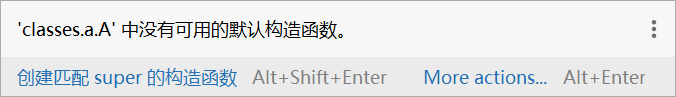
下面是程序的结构图，可以看出，SubClass与SuperClass是子类与父类的关系，Inheritance则与它们都无关，是用来测试的类，SubClass中的doSomething与父类中方法签名一致，是对父类方法的重写。



实验结果如下图所示：



从结果可以看出①体现了构造函数调用的规则，若调用子类的构造函数，则会隐式调用其父类的无参构造函数，这时，若父类中定义了有参构造函数却未定义无参构造函数则会报错：



父类还有父类则要调用父类的父类的构造函数，依次类推。若要显式的写出调用，则要写在子类构造方法的第一句。

## 【第四个实验】

1.实验源代码

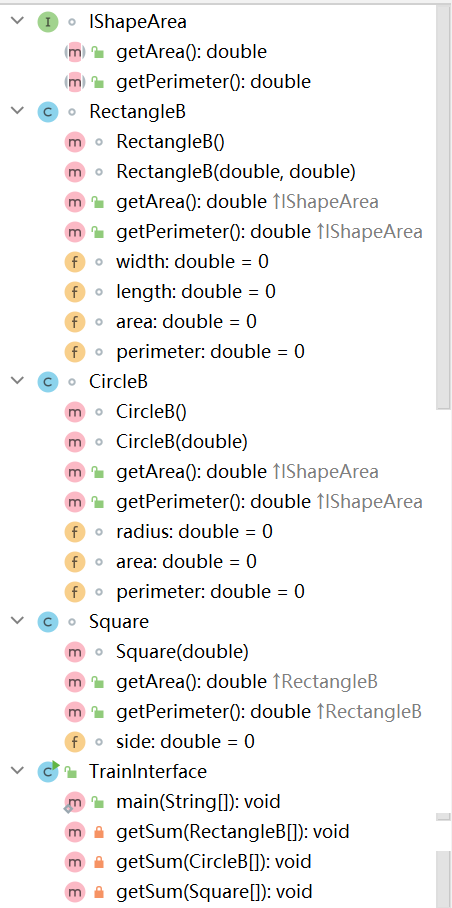
package classes.b;  
  
import java.util.Random;  
  
interface IShapeArea {  
  
 double getArea();  
  
 double getPerimeter();  
}  
  
class RectangleB implements IShapeArea {  
 double width = 0;  
 double length = 0;  
 double area = 0;  
 double perimeter = 0;  
  
 RectangleB() {  
 }  
  
 *//由于同包中有Rectangle类，故命名为了RectangleB* RectangleB(double length, double width) {  
 this.length = length;  
 this.width = width;  
 }  
  
 public double getArea() {  
 this.area = this.width \* this.length;  
 return this.area;  
 }  
  
 public double getPerimeter() {  
 this.perimeter = 2 \* (this.width + this.length);  
 return perimeter;  
 }  
}  
  
class CircleB implements IShapeArea {  
 double radius = 0;  
 double area = 0;  
 double perimeter = 0;  
  
 CircleB() {  
 }  
  
 CircleB(double radius) {  
 this.radius = radius;  
 }  
  
 *//实现接口的方法必须为public* public double getArea() {  
 this.area = Math.*PI* \* this.radius \* this.radius;  
 return this.area;  
 }  
  
 public double getPerimeter() {  
 this.perimeter = 2 \* Math.*PI* \* radius;  
 return this.perimeter;  
 }  
}  
  
class Square extends RectangleB {  
 double side = 0;  
  
 Square(double side) {  
 this.side = side;  
 }  
  
 public double getArea() {  
 this.area = this.side \* this.side;  
 return this.area;  
 }  
  
 public double getPerimeter() {  
 this.perimeter = 4 \* this.side;  
 return perimeter;  
 }  
}  
  
public class TrainInterface {  
 public static void main(String[] args) {  
 TrainInterface train = new TrainInterface();  
 double[][] sum = new double[3][2];  
 RectangleB[] rect = new RectangleB[2];  
 Square[] sqr = new Square[3];  
 CircleB[] cir = new CircleB[4];  
 *//随机生成图形并求其面积、周长和* train.getSum(rect);  
 train.getSum(cir);  
 train.getSum(sqr);  
 }  
  
 private void getSum(RectangleB[] rectangle) {  
 System.*out*.println("-----------------矩形-----------------");  
 Random rand = new Random();  
 double[][] sum = new double[1][2];  
 for (int i = 0; i < rectangle.length; i++) {  
 rectangle[i] = new RectangleB(10 \* rand.nextDouble(), 10 \* rand.nextDouble());  
 double area = rectangle[i].getArea();  
 double perimeter = rectangle[i].getPerimeter();  
 System.*out*.println("第 " + (i + 1) + " 个矩形：长为： " + String.*format*("%.2f", rectangle[i].length) + " 宽为： " + String.*format*("%.2f", rectangle[i].width));  
 System.*out*.println(" 面积为： " + String.*format*("%.2f", area) + " 周长为： " + String.*format*("%.2f", perimeter));  
 sum[0][0] += area;  
 sum[0][1] += perimeter;  
 }  
 System.*out*.println("这 " + rectangle.length + " 个矩形的面积和为： " + String.*format*("%.2f", sum[0][0]));  
 System.*out*.println("这 " + rectangle.length + " 个矩形的周长和为： " + String.*format*("%.2f", sum[0][1]));  
 }  
  
 private void getSum(CircleB[] circle) {  
 System.*out*.println("-----------------圆形-----------------");  
 Random rand = new Random();  
 double[][] sum = new double[1][2];  
 for (int i = 0; i < circle.length; i++) {  
 circle[i] = new CircleB(10 \* rand.nextDouble());  
 double area = circle[i].getArea();  
 double perimeter = circle[i].getPerimeter();  
 System.*out*.println("第 " + (i + 1) + " 个圆形：半径为： " + String.*format*("%.2f", circle[i].radius));  
 System.*out*.println(" 面积为： " + String.*format*("%.2f", area) + " 周长为： " + String.*format*("%.2f", perimeter));  
 sum[0][0] += area;  
 sum[0][1] += perimeter;  
 }  
 System.*out*.println("这 " + circle.length + " 个圆形的面积和为： " + String.*format*("%.2f", sum[0][0]));  
 System.*out*.println("这 " + circle.length + " 个圆形的周长和为： " + String.*format*("%.2f", sum[0][1]));  
 }  
  
 private void getSum(Square[] square) {  
 System.*out*.println("-----------------正方形-----------------");  
 Random rand = new Random();  
 double[][] sum = new double[1][2];  
 for (int i = 0; i < square.length; i++) {  
 square[i] = new Square(10 \* rand.nextDouble());  
 double area = square[i].getArea();  
 double perimeter = square[i].getPerimeter();  
 System.*out*.println("第 " + (i + 1) + " 个正方形：边长为： " + String.*format*("%.2f", square[i].side));  
 System.*out*.println(" 面积为： " + String.*format*("%.2f", area) + " 周长为： " + String.*format*("%.2f", perimeter));  
 sum[0][0] += area;  
 sum[0][1] += perimeter;  
 }  
 System.*out*.println("这 " + square.length + " 个正方形的面积和为： " + String.*format*("%.2f", sum[0][0]));  
 System.*out*.println("这 " + square.length + " 个正方形的周长和为： " + String.*format*("%.2f", sum[0][1]));  
 }  
}

2.实验记录与说明分析



以上是程序运行的结果，综合运用了接口、方法的重写与重载、类的继承、对象数组等知识，运行的结果经检验是正确的。

下面放出其结构图，可以清晰的看出其关系：



## 【第五个实验】

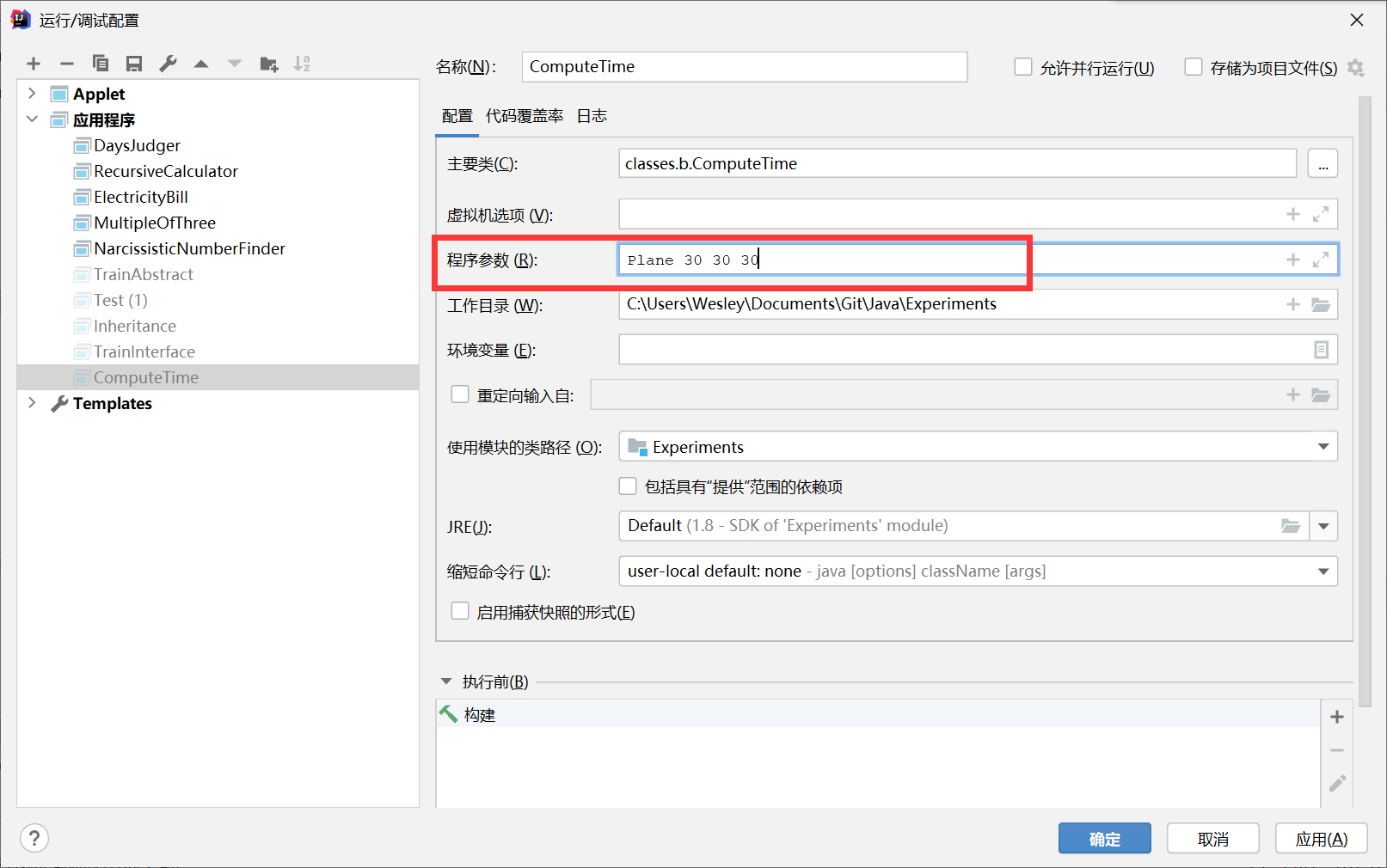
1.实验源代码

package classes.b;  
  
import java.lang.\*;  
  
interface Common {  
 double getSpeed(double A, double B, double C);  
}  
  
class Car implements Common {  
 public double getSpeed(double A, double B, double C) {  
 return (A \* B / C);  
 }  
}  
  
class Plane implements Common {  
 public double getSpeed(double A, double B, double C) {  
 return (A + B + C);  
 }  
}  
  
public class ComputeTime {  
 public static void main(String[] args) {  
 double speed, time;  
 *//下面寻找类时需要使用到包名* String packageName = ComputeTime.class.getPackage().getName() + ".";  
 *//这里是运行程序时传进来的参数* System.*out*.println("交通工具类型: " + args[0]);  
 System.*out*.println(" 参数A: " + args[1]);  
 System.*out*.println(" 参数B: " + args[2]);  
 System.*out*.println(" 参数C: " + args[3]);  
 double A = Double.*parseDouble*(args[1]);  
 double B = Double.*parseDouble*(args[2]);  
 double C = Double.*parseDouble*(args[3]);  
 try {  
 *//Class.forName("") 的作用是要求JVM查找并加载指定的类  
 //加载类后再使用newInstance得到实例化对象  
 //这里要求这些类必须属于Common这个接口* Common newObject = (Common) Class.*forName*(packageName + args[0]).newInstance();  
 speed = newObject.getSpeed(A, B, C);  
 time = 1000 / speed;  
 System.*out*.println("平均速度为: " + speed + " km/h");  
 System.*out*.println("运行时间为：" + time + " 小时");  
 } catch (ReflectiveOperationException e) {  
 System.*out*.println("尚未编写" + args[0] + "类的程序！");  
 }  
 }  
}

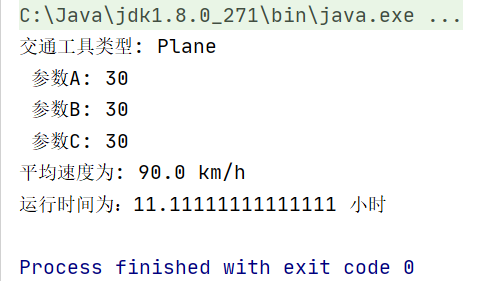
2.实验记录与说明分析

程序的核心在于对main函数参数的使用以及newInstance()方法的使用，这一方法的使用属于Java中的反射机制，可以动态的对类进行加载。

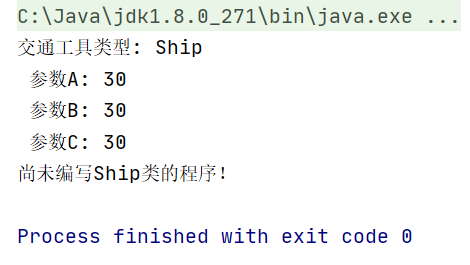
由于该程序的main函数使用了传进来的参数，所以可以在命令行运行，但比较麻烦，想在IDE中运行需要运行前配置一下参数。



运行得到如下结果：

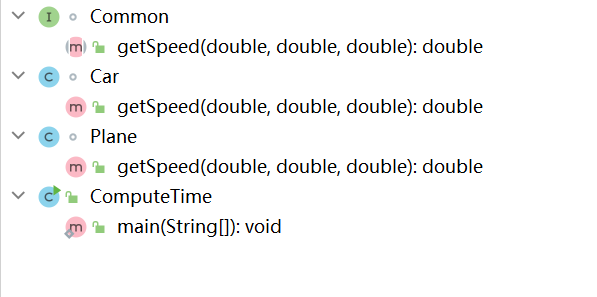


若参数所给出的交通工具还没有实现，则会触发异常。



只需要编写该类的代码，不需要修改main函数，即可实现对Ship类的计算 。

下面是程序的结构：



## 【实验心得】

IDEA这一集成开发环境是非常人性化的，它有很多功能等待我们去探索，比如只需要打开Structure选项卡即可看到整个程序的结构，其中类之间的关系清晰展现；代码修复功能十分强大， 有助于我们对Java的学习和理解，一个优秀的IDE将大大方便我们编写代码。

这一次实验对接口、抽象类等有了更好的理解，它们可以有效地提高代码的复用性，还了解了Java的一个高级属性——反射（尽管现在还不是很理解），它可以降低代码的耦合程度。希望这些知识在今后的学习生活中能够派上用场