**Java程序设计 LAB04**

**实验目的：**

* 理解继承的必要性
* 理解继承的基本概念：单继承，多继承
* 掌握继承的使用
  + 子类属性、构造方法和一般成员方法的书写的编写
  + 变量隐藏
  + this和super的使用
  + 向上映射
  + 在继承的基础上加深理解权限控制符的使用
  + 保护成员的使用
  + 方法覆盖（初步，重点放在多态一章）
* 复合与继承的区别及使用

注意事项

建议建立一个自己的统一且良好的**代码风格**，比如**命名风格**（camelCase、snake\_case等）、缩进方式（空格数量、switch-case缩不缩进等）、开闭大括号换不换行等容易引发战争（迫真）的东西，以养成良好的编程习惯。

编程题最好为每一个类编写一个完备的测试类，覆盖尽可能多的输入、函数调用、输出，以证明代码正确实现了功能。

如果编程题使用了package语句，应当确保提交时目录结构和package语句表达的包结构相同。（IDE很多时候会帮你做）

编程题在给出了具体需求的情况下，**可以根据自己的需要添加额外的方法**。

实验题目

**根据教学进度，本次实验的题目在设计时均没有考虑interface。**

1. **Package**

浏览OO文件夹中的01目录，不允许变更目录结构和文件位置，只允许在文件中添加package和import语句，使根目录下build和test文件中的指令可以无报错顺利执行。

1. **权限**

阅读OO文件夹中的02目录，根据你对访问权限的理解。将文件中所有无法通过编译的语句注释掉。

你遇到的问题可能会集中在protected和默认访问权限。

从这次实验和上一次实验，可以引申出几个问题：

* + LAB03中的单例模式，或者说所有构造方法都是private时，可能有子类吗？
  + final类可以视为所有构造方法都是private的类吗？

**题外话：**

如果不知道怎么编译和运行，参见第一题

1. **Hide & Override**

阅读：

|  |
| --- |
| **class** Parent {  **int** num = 4;  **protected** **void** foo() {  System.out.println("foo() of Parent");  }  **static** **protected** **void** bar() {  System.out.println("bar() of Parent");  }  }  **class** Child **extends** Parent {  **int** num = 5;  **protected** **void** foo() {  System.out.println("foo() of Child");  }  **static** **protected** **void** bar() {  System.out.println("bar() of Child");  }  }  **public** **class** Test {  **public** **static** **void** main(String[] args) {  Parent f1 = **new** Parent();  System.out.println(f1.num);  Parent f2 = **new** Child();  System.out.println(f2.num);  Child c = **new** Child();  System.out.println(c.num);    f1.foo();  f2.foo();  c.foo();    f1.bar();  f2.bar();  c.bar();  }  } |

执行java Test，程序的运行结果是什么？

如果将子类的protected void foo()改为默认访问权限，程序还能通过编译吗？

如果将子类的protected void bar()改为默认访问权限，程序还能通过编译吗？

foo和bar的测试输出结果表现一致吗？

属性、静态方法、非静态方法，哪些可以覆盖（重写、override），哪些可以隐藏？

**题外话：**

在想要覆盖的时候，主动写一个@Override注解，可以提高代码可读性，也可以防止你想要覆盖但是写成了重载的情况发生。

不想让父类方法在子类中被覆盖的时候，将父类方法声明为final的。

1. **Shape**

阅读下面的二维形状类Shape的描述：

|  |
| --- |
| **public** **abstract class** Shape {  **protected** double a;  **protected** double b;  **public** Shape() { **this**(0.0, 0.0); }  **public** Shape(double a, double b) {  **this**.a = a;  **this**.b = b;  }  */\*\* calcArea*  *\* 计算形状的面积*  *\* @return 面积*  *\*/*  **abstract public** double calcArea();  */\* 其他必要的方法，比如getter和setter \*/*  } |

你的任务是：

* + 编写Rectangle类，a和b分别代表矩形两条边的长度（长和宽）；
  + 编写Rhombus类，a和b分别代表两条菱形对角线的长度；
  + 编写Ellipse类，a和b分别代表椭圆两个半轴（半长轴和半短轴）的长度；
  + 编写一个测试类，构造子类对象并测试

注意：

* + **上述三个类都继承自Shape类，都必须要override方法calcArea；**
  + a和b都不能是负数，当通过constructor或者setter设定为负数时，将对应的值置为0.0，或throw异常；
  + a和b的大小关系没有任何约束，没有“a必须不小于b”的说法。

**题外话：**

我们知道属性不设置为public是为了保护数据，如果a和b在Shape中的访问权限被设置为private，在不添加新的属性的情况下，子类想要实现自己的calcArea()该怎么办？

1. **xx岁，是学生**

定义一个Person类，属性有姓名、性别、生日；

定义一个Student类，除了作为人的属性，还有学号（常量）；

定义一个研究生类，除了学生的属性，他还有专业、导师；

描述上述关系并在测试类中测试功能。

1. **车车**

定义一个Vehicle类，在其中声明一个属性代表这个交通工具有多少个轮子，提供对这个属性的setter。提供轮子类、引擎类。创建几个Vehicle的导出类：Motorbike、Car、Tank。自由发挥，在测试类中构造这几种交通工具，并测试功能。

1. **初始化II**

阅读上一次实验的题目《初始化I》以及《面向对象上机LAB04反馈》，证明加粗部分:

1. JVM执行类装载时，在加载某类的.class文件后，**如果其父类还没有被装载过，会先完成父类的装载（加载文件和静态初始化），之后才执行本类的静态初始化**。
   * 证明点1： 变量：父类有没有被装载过
   * 证明点2： 顺序：父类优先
2. 调用构造器构造对象时，**先调用父类构造器，等父类构造器执行结束后，才会按类定义的顺序初始化显式赋值的实例属性，最后执行本类构造器的其他代码**。
   * 证明点： 顺序：先父类，再属性，最后其他代码

**题外话：**

还记得高中物理实验吗？对照试验+控制变量。

有兴趣可以尝试证明：

* + - 装载某类A，不会导致其子类的装载，除非父类初始化时要访问子类。
    - 装载某类A时，即使此类的所有方法都会使用未装载的类B、C、D等，也不会导致类装载；除非类A初始化时使用了这些方法或者这些类。

1. **文件**

操作系统中，有一个概念叫做文件系统。以Windows为例，其文件系统中的文件可以粗略地分为以下几种：

* + 文件夹（包含一个子目录）
  + 快捷方式（打开它等价于打开其所指向的文件）
  + 可执行文件（.exe这种不需要借助其他程序就能直接运行的文件）
  + 不可执行文件（本身无法运行，需要借助其他程序（打开方式）才能正确使用）

以上四种，都是“文件”。

所有文件的属性都有：

* + 名称（不能为空，字符集是ASCII可打印字符）
  + 创建时间（年月日时分秒）
  + 大小（以byte为单位，是整数）
  + 内容（字节序列）
  + 位置（直接包含此文件的文件夹）

文件夹：

不考虑OS层面的内存对齐，文件夹的大小等于其内部所有文件的大小的加和；

文件夹的额外属性是，他的直接子目录中包含的文件夹数和其他文件数；

打开文件夹，会切换当前的工作路径到该文件夹下，初始工作路径我们认为是C盘；

可以在文件夹中添加新的文件；

快捷方式：

本质是一个类型为.lnk的文件；

快捷方式的额外属性是其指向的文件（可以是任何文件，包括快捷方式和文件夹）；

打开快捷方式，等价于打开其指向的文件；指向的文件被删除后，无法打开快捷方式；

注意，快捷方式的所有属性，都代表其本身，不代表其指向的文件；

可执行文件：

打开就是运行其内容。

不可执行文件：

没有给出打开方式的情况下，无法被正确打开。

**任务：**

抽象出上述文本中的属性、方法、继承和组合信息，编写多个类描述上述关系。

不用实现很具体的功能，甚至不用实现可以运行的程序。但是不能有逻辑硬伤和编译错误。要求实现两种写法：不用继承和使用继承。

**题外话：**

你可能需要回忆一下C中的结构体，学习一下Java中的enum类。