Java06

Task1.继承

经过查阅资料，得知：这是一个非常核心的Java面向对象问题。Java的设计者们有意地不支持类的多继承（即一个类不能直接继承多个父类），主要是为了避免由多继承引发的著名问题——“**菱形问题**”，也为了保持**语言的简洁性与清晰性**，同时，java中有能够实现类似功能的**更优的解决方案**——接口。

- 菱形问题

这是最核心、最常被提及的原因。菱形问题是一个典型的歧义场景，假设Java支持多继承，考虑以下代码：

```
// 假设Java支持多继承
class A {
    public void method() {
        System.out.println("A's method");
    }
}

class B extends A {
    @Override
    public void method() {
        System.out.println("B's method");
    }
}

class C extends A {
    @Override
    public void method() {
        System.out.println("C's method");
    }
}

// 类D同时继承了B和C，那么它应该使用哪个 method()?
class D extends B, C { // 实际上这是不合法的Java代码
    public void test() {
        method(); // 歧义！是调用B的method还是C的method?
    }
}
```

在上面这段代码中，B、C分别重写了继承自A的method方法，若D同时继承了B和C，那么当D创建的对象调用method方法时，编译器无法判断该调用B还是C中重写的method方法。 **此外**，如果类A有一个成员变量value，那么由于B和C都继承了它，类D中会包含两份value的副本，这同样会造成混乱。

- 语言设计的简洁性与清晰性

通过强制使用单一继承，能避免程序设计方面的一些潜在的巨大复杂性，例如，在单一继承中，`super()`明确指向唯一的父类构造函数，而在多继承中，需要额外添加参数以明确它指向具体哪个父类。 **同时**，通过强制使用单一继承，Java中的父类与子类的关系将会是一条不会分岔的线，简单易懂。

- 接口作为更优的替代方案

Java并没有完全放弃"多继承"的概念，而是将其转化为了一个更安全、更强大的形式——**一个类可以实现多个接口。**

> 不过，虽然在Java中，类的多继承是不合法的，但接口允许多继承。

Task2.多态

使用继承实现：

```
import java.util.Scanner;
import java.lang.Math;

abstract class VariousShapes{
    //通过使用抽象类强制要求子类实现抽象方法
    double pi=3.1415926;
    double radius;
    double length1,length2,length3;
    abstract public double getPerimeter();
    abstract public double getArea();
}
class Circle extends VariousShapes{
    public Circle(double r){
        radius=r;
    }
    @Override
    public double getPerimeter(){
        return 2.0*pi*radius;
    }
    @Override
    public double getArea(){
        return pi*radius*radius;
    }
}
class Rectangle extends VariousShapes{
    public Rectangle(double l1,double l2){
        length1=l1; length2=l2;
    }
    @Override
    public double getPerimeter() {
        return 2.0*(length1+length2);
    }
    @Override
    public double getArea() {
        return length1*length2;
    }
```

```
}
class Triangle extends VariousShapes{
    public Triangle(double l1,double l2,double l3){
        length1=l1; length2=l2; length3=l3;
    }
    @Override
    public double getPerimeter(){
        return length1+length2+length3;
    }
    @Override
    public double getArea(){
        double tmp=(length1+length2+length3)/2;
        return Math.sqrt(tmp*(tmp-length1)*(tmp-length2)*(tmp-length3));
    }
}

public class java06 {
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        String type;
        VariousShapes shape;
        while(true){
            type=sc.nextLine();
            if(type.equals("圆形")){
                System.out.print("r= ");
                shape =new Circle(sc.nextDouble());
                System.out.printf("周长=%.4f 面积=%.4f \n",
shape.getPerimeter(),shape.getArea());
            } else if(type.equals("矩形")){
                System.out.print("a b= ");//输入的数据以空格分隔
                shape =new Rectangle(sc.nextDouble(),sc.nextDouble());
                System.out.printf("周长=%.4f 面积=%.4f \n",
shape.getPerimeter(),shape.getArea());
            } else if(type.equals("三角形")){
                System.out.print("a b c= ");//输入的数据以空格分隔
                shape =new
Triangle(sc.nextDouble(),sc.nextDouble(),sc.nextDouble());
                System.out.printf("周长=%.4f 面积=%.4f \n",
shape.getPerimeter(),shape.getArea());
            } else if(type.equals(("退出"))){
                //设置特定的标识以停止循环，若只是else，则输入缓冲区中的'\n'会导致循环
提前停止
                break;
            }
        }
    }
}
```

使用接口实现：

```java
import java.util.Scanner;
import java.lang.Math;

interface Shapes{
    static double pi = 3.1415926;
    double getPerimeter();
    double getArea();
}
class Circles implements Shapes{
    double r;
    public Circles(double r){
        this.r=r;
    }
    @Override
    public double getPerimeter() {
        return 2.0*pi*r;
    }
    @Override
    public double getArea() {
        return pi*r*r;
    }
}
class Rectangles implements Shapes{
    double a,b;
    public Rectangles(double a,double b){
        this.a=a; this.b=b;
    }
    @Override
    public double getPerimeter() {
        return 2.0*(a+b);
    }
    @Override
    public double getArea() {
        return a*b;
    }
}
class Triangles implements Shapes{
    double a,b,c;
    public Triangles(double a,double b,double c){
        this.a=a; this.b=b; this.c=c;
    }
    @Override
    public double getPerimeter() {
        return a+b+c;
    }
    @Override
    public double getArea() {
        double t=(a+b+c)/2;
        return Math.sqrt(t*(t-a)*(t-b)*(t-c));
    }
}

public class java06_1 {
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input;
        while(true){
            input= sc.nextLine();
            if(input.equals("circle")){
                System.out.print("r= ");
                Circles shape = new Circles(sc.nextDouble());
                System.out.printf("Perimeter=%.4f
Area=%.4f",shape.getPerimeter(),shape.getArea());
            } else if(input.equals("rectangle")){
                System.out.print("a b= ");
                Rectangles shape = new
Rectangles(sc.nextDouble(),sc.nextDouble());
                System.out.printf("Perimeter=%.4f
Area=%.4f",shape.getPerimeter(),shape.getArea());
            } else if(input.equals("triangle")){
                System.out.print("a b c= ");
                Triangles shape = new
Triangles(sc.nextDouble(),sc.nextDouble(),sc.nextDouble());
                System.out.printf("Perimeter=%.4f
Area=%.4f",shape.getPerimeter(),shape.getArea());
            } else if(input.equals("exit")){
                break;
            }
        }
    }
}
```

Task3

不同访问修饰符的用法

1.private

- **含义**：私有的。
- **访问权限**：仅在定义它的类内部可以访问。类外部（包括子类）无法直接访问。
- **用途**：这是实现封装的关键。通常将类的属性（成员变量） 声明为 private，以防止外部代码随意修改，从而保护数据的完整性和安全性。

1.public

- **含义**：公共的。
- **访问权限**：可以被任何其他类访问。
- **用途**：通常将类对外提供的方法（接口） 声明为 public，作为外部与对象交互的“窗口”。

3.protected

- **含义**：受保护的。

- **访问权限**：可以被同一个包中的类以及它的子类（即使子类在不同包）访问。
- **用途**：主要用于继承中，允许子类访问父类的某些成员，同时仍对无关的外部类隐藏。

4.默认（不加任何修饰符）

- **含义**：包级私有。
- **访问权限**：可以被同一个包中的类访问。
- **用途**：用于包内部的协作，对包外部的类隐藏。

代码补全如下：

```
public class BankAccount {
    private String accountNumber;
    private String accountHolder;
    private double balance;
    private String password; // 敏感信息，需要严格保护

    BankAccount(String accountNumber, String accountHolder, double initialBalance,
String password) {
        this.accountNumber=accountHolder;
        this.accountHolder=accountHolder;
        this.balance=initialBalance;
        this.password=password;
    }

    void deposit(double amount) {
        if(amount>0){
            balance+=amount;
        } else{
            System.out.println("金额格式错误");
        }
    }

    boolean withdraw(double amount, String inputPassword) {
        if(password.equals(inputPassword)){
            if(amount>0){
                if(balance>=amount){
                    balance-=amount;
                    return true;
                } else{
                    System.out.println("余额不足");
                    return false;
                }
            } else{
                System.out.println("金额格式错误");
                return false;
            }
        } else{
            System.out.println("密码错误");
            return false;
        }
    }
```

```
    boolean transfer(BankAccount recipient, double amount, String inputPassword) {
        if(!password.equals(inputPassword)){
            System.out.println("密码错误");
            return false;
        }
        if(amount<=0){
            System.out.println("金额格式错误");
            return false;
        }
        if(amount>balance){
            System.out.println("余额不足");
            return false;
        }
        recipient.balance+=amount;
        balance-=amount;
        return true;
    }

    double getBalance() {
        return balance;
    }

    String getAccountInfo() {
        return accountNumber;
    }
    // 只需修改可见性
    private boolean validatePassword(String inputPassword) {
        return true;
    }
    // 只需修改可见性
    private boolean validateAmount(double amount) {
        return true;
    }
}
```