Java06-封装继承与多态

Task1.继承

何为继承

继承就是子类自动获得父类的特性和能力，就像孩子继承父母的某些特征一样。

- 代码格式：`public class 子类（派生类） extends 父类（基类或超类） {}`
- 好处：
  - 提高代码的复用性
  - 子类可以在父类基础上，增加其他功能，使子类更强大
- 特点：
  - Java只支持单继承，不支持多继承，支持多层继承
  - 每一个类都直接或间接地继承于`Object`

题目代码阅读及相关补充

```java
public class Penguin extends Animal {
    //子承父类
    public Penguin(String myName, int myid) {
        //定义一个构造方法来初始化
        super(myName, myid);
        //相当于：
        //super.myName=myName;
        //super.myid=myid;

    }
}
```

和 Task.5 里的`this`类似，`super`关键字的作用和补充如下：

- 作用：用于指向**直接父类对象。**它主要用于在子类中访问父类的成员（属性和方法）。
- 成员变量的访问特点：**就近原则**

  如果重名 -> `this`（本类 -> 子类 -> 父类）和`super`（父类）关键字
- **注意事项：**
  - **构造方法调用必须第一行**：在构造方法中，`this(...)` 和 `super(...)` 都必须是第一条语句，因此它们**不能同时出现**在一个构造方法中。
  - **静态上下文中无效**：`this` 和 `super` 都依赖于对象实例，因此不能在 `static` 方法（包括 `main` 方法）或 `static` 代码块中使用。

问题回答：java并不支持多继承

想象一个类 `FlyingCar`，它想同时继承 `Car`和 `Airplane`这两个类。其中，`Car` 类有一个 `run()` 方法，表示“在公路上行驶”，`Airplane` 类也有一个 `run()` 方法，但表示“在跑道上滑行”，现在 `FlyingCar` 类从两个父类那里都继承了一个 `run()` 方法。

那么，当 `FlyingCar` 的对象调用 `run()` 方法时，编译器会非常困惑，因为它**不知道应该执行哪个父类的 `run()` 方法。**

这就是著名的“钻石问题”（Dilemma of the Diamond），因为类的继承结构形状像一颗钻石。

所以为了避免上述情况发生，Java无法实现多继承，以此**简化语言，避免歧义，减少复杂性。**

Task2.多态

何为多态

一言以蔽之，多态就是**相同名字的方法，有不同的表现形式（状态）**

多态存在的必要条件：

1. **继承**：必须存在继承关系。
2. **重写**：子类必须对父类中的某些方法进行重写（Override）。
3. **向上转型**：父类引用指向子类对象：`Parent p = new Child();`

何为重写（@Override）

个人更喜欢另一个相似的中文翻译：**覆写**，是指**子类对从父类继承过来的方法进行重新实现。**

它的核心特点是：**方法名、参数列表、返回值类型必须与父类方法完全相同**。重写后，当通过子类对象调用该方法时，将执行子类中重写后的新方法体，而不是父类原有的方法体。我们将 `@Override` 作为重写的标志，告诉VM我们要开始重写了。

何为接口

说到接口我们必须要先谈一谈 `Abstract` 是什么。

Abstract抽象类

抽象类是一个不能被实例化（即不能直接创建对象）的类。

- 意义：实现代码复用和共性抽取，实现多态。
- 特点：
  - 不能被实例化。
  - **子类必须实现所有抽象方法。**
  - 可以有抽象方法和具体方法。

接口（interface）

接口是一种抽象类型，它定义了一种方法，但没有提供方法的具体实现。

- 类可以实现一个或多个接口，以提供接口定义方法的具体实现。
- 好处：实现代码的复用和多态性。
- 接口的实现：`implements` 关键字。

接口 vs. 抽象类

特性	抽象类	接口 (Java 8+)
实例化	不能	不能
方法类型	抽象方法 + 具体方法	抽象方法 + **默认方法** `default` + 静态方法
成员变量	可以是各种类型（甚至非静态、非final）	默认是 `public static final` (常量)
继承	**单继承** (一个类只能继承一个抽象类)	**多实现** (一个类可以实现多个接口)
设计理念	**“是一个（IS-A）”** 关系，表示类的本质	**“具有一种（CAN-DO）”** 能力，表示行为契约

选择依据：

- 如果要定义一些紧密相关对象的**核心、本质身份**（如：Dog **是一个** Animal），使用**抽象类。**
- 如果要定义一些**可选的的功能**（如：一个类可以同时 `CanFly` 和 `CanSwim`），使用**接口。**
- 以实际例子来说明：
  - 我们上学的流程都是“起床 -> 吃饭 -> 睡觉”，但是大家吃的饭不一样——使用Abstract。
  - 如上面所说，定义可选的功能，使用接口。

问题解决

Solution 1：继承和多态

首先写父类，运用了`Abstract`来定义面积和周长，便于子类的继承来实现多态。

```java
package Calculate;

public abstract class Shape {
    public static double PAI = 3.14;
    private double S;
    private double C;

    public Shape() {
    }

    public Shape(double s, double c) {
        S = s;
        C = c;
    }

    public abstract double getS();
    public abstract double getC();
}
```

```
19
```

接下来就是三个子类

圆形：

```java
package Calculate;

public class Circle extends Shape{
    private double r;

    public Circle(double r){
        this.r = r;
    }

    @Override
    public double getS() {
        return PAI * r * r;
    }

    @Override
    public double getC() {
        return 2 * PAI * r;
    }

}

```

三角形：

```java
package Calculate;

public class Tri extends Shape{
    private double x,y,z;
    public Tri(double x,double y,double z){
        this.x = x;
        this.y = y;
        this.z = z;
    }


    @Override
    public double getS() {
        //海伦公式
        double p = (x+y+z)/2;
        return Math.sqrt(p * (p - x) * (p - y) * (p - z));
    }

    @Override
    public double getC() {
        return x + y + z;
    }
}

```

矩形：

```
package Calculate;

public class Rec extends Shape{
    private double b,c;

    public Rec(double b,double c){
        this.b = b;
        this.c = c;
    }


    @Override
    public double getS() {
        return b * c;
    }

    @Override
    public double getC() {
        return 2 * (b + c);
    }
}

```

最后来写一个测试类并赋值：

```
package Calculate;

public class Test {
    public static void main(String[] args) {
        Circle c = new Circle(2);
        Tri tri = new Tri(4,5,3);
        Rec r = new Rec(6,7);

        System.out.println(c.getS());
        System.out.println(c.getC());
        System.out.println(tri.getS());
        System.out.println(tri.getC());
        System.out.println(r.getS());
        System.out.println(r.getC());
    }
}

```

输出结果如下：结果无误。

```
12.56
12.56
6.0
12.0
42.0
26.0
```

Solution 2：接口

把Solution 1的父类Shape改写成接口，其余子类代码只需将extends修改为implements即可。
为**方便审核~~（方便我偷懒）~~**此处仅放接口代码和其中一个子类代码。

接口Shape

```
package Calculate2;

public interface Shape {
    public static double PAI = 3.14;
    public abstract double getS();
    public abstract double getC();
}

```

子类Circle：

```
package Calculate2;

//只在这个地方修改一下就好了
public class Circle implements Shape {
    private double r;

    public Circle(double r){
        this.r = r;
    }

    @Override
    public double getS() {
        return PAI * r * r;
    }

    @Override
    public double getC() {
        return 2 * PAI * r;
    }
}
```

再来一个Test类：

```
package Calculate2;

public class Test {
    public static void main(String[] args) {
        Circle c = new Circle(2);
        System.out.println(c.getS());
        System.out.println(c.getC());
    }

}

```

输出结果如下：结果无误。

```
12.56
12.56

进程已结束，退出代码为 0
```

Task3.封装

何为封装

引用题干，封装是指一种**将抽象性函式接口的实现细节部分包装、隐藏起来的方法**，可以被认为是一个保护屏障，防止该类的代码和数据被外部类定义的代码随机访问，从而保证数据的安全性、灵活性和可操作性。

封装最主要的功能在于我们能修改自己的实现代码，而不用修改那些调用我们代码的程序片段。

public和private关键字

在 **Task.5** 里我们已经学习了**访问修饰符**：（该表格来自于本人Task.5的md文档）

修饰符	当前类	同一包内	不同包的子类	不同包的非子类	说明
`public`	✅	✅	✅	✅	项目内完全公开
`protected`	✅	✅	✅	❌	主要提供给子类使用
`default`	✅	✅	❌	❌	包内可见，默认选项
`private`	✅	❌	❌	❌	仅当前类内部可见

在封装中`private`是非常重要的，因为被private修饰的成员**只能在本类中才能访问**，且针对private修饰的成员变量，如果被其他类使用，应该提供相应的操作，从而实现了封装，保证代码和数据的安全性。

问题解决

这是一个模拟银行账户进行一些操作时的代码。

需求分析&思路分析

`void deposit(double amount)`：存钱，amount是存进去的金额，对balance进行修改。
`boolean withdraw(double amount, String inputPassword)`：取钱，需要判断密码，并对balance进行修改。
`boolean transfer(BankAccount recipient, double amount, String inputPassword)`：转钱，需要判断密码。
`double getBalance()`：获取余额。
`String getAccountInfo()`：获取账户信息。
`boolean validatePassword(String inputPassword)`：判断是否为有效的密码，就是判断密码对不

对，在上面**取钱**和**转钱**需要。
`boolean validateAmount(double amount)`：判断是否为有效的金额。

代码实现

保留了题干的注释，自己在过程中也加了一些~~不必要的~~注释。

```java
package BankAccount;

public class BankAccount {
    // TODO 修改属性的可见性
    //已修改为private
    private String accountNumber;
    private String accountHolder;
    private double balance;
    private String password; // 敏感信息，需要严格保护

    //构造方法实现初始化
    BankAccount(String accountNumber, String accountHolder, double initialBalance, String password) {
        //TODO
        this.accountNumber = accountNumber;
        this.accountHolder = accountHolder;
        this.balance = initialBalance;
        this.password = password;
    }

    void deposit(double amount) {
        //TODO
        balance = balance + amount;
    }

    boolean withdraw(double amount, String inputPassword) {
        //TODO
        if (validatePassword(inputPassword)) {
            balance = balance - amount;
            return true;
        }
        else{
            return false;
        }
    }

    boolean transfer(BankAccount recipient, double amount, String inputPassword) {
        //TODO
        if (validatePassword(inputPassword)) {
            balance = balance - amount;
            recipient.balance = recipient.balance + amount;
            return true;
        }
        else{
            return false;
        }

```

```
    }

    double getBalance() {
        //TODO
        //额额报错的时候想起来没有return
        //System.out.println(balance);
        return balance;
    }

    String getAccountInfo() {
        //TODO
        //同上
        //System.out.println("Account: " + accountNumber + ", Holder: " + accountHolder + ", Balance: " + balance);
        return "Account: " + accountNumber + ", Holder: " + accountHolder + ", Balance: " + balance;
    }

    // 只需修改可见性
    //已修改为private
    private boolean validatePassword(String inputPassword) {
        return true;
    }

    // 只需修改可见性
    //已修改为private
    private boolean validateAmount(double amount) {
        return true;
    }
}
```

来写一个简单的测试类：

```
package BankAccount;

public class Test {

    public static void main(String[] args) {
        BankAccount b = new BankAccount("666","Glimmer",0,"123456");
        b.deposit(999);
        System.out.println(b.getBalance());
    }
}

```

运行结果如下：无误

```
999.0

进程已结束，退出代码为 0
```