Task1.继承

java并不支持多继承，这是为什么?

1. 避免混乱的继承关系：当一个子类有多个父类，不同的父类都定义了相同的方法，但以不同的具体方式实现。此时，子类的该方法就会产生冲突，导致二义性和逻辑混乱。所以，java不支持多继承。
2. 存在替代方案：为了实现多继承的替代方案，java提出了多接口+单继承的模式。因为接口不会具体实现方法，而只是定义了方法签名，所以不会出现上述混乱。
3. 简化了java的语言设计，提高了编译器的运行效率，降低学习难度。

Task2.多态

package `polymorphism1`完成了抽象类的实现方式

package `polymorphism2`完成了接口的实现方式

详见上述两个package

Task3.封装

`public`

作为访问权限修饰符

常用于类，接口，方法，常量。

e.g.

```
public class Student {
    public String name;
    public int number;
    public void sayHello() {
        System.out.println("Hello, I'm " + name + "and my number is" + number);
    }
}
```

```
public interface Shape {
    double getArea();
    double getPerimeter();
}
```

项目中的任何地方都可以访问`public`修饰的内容。

`private`

作为访问权限修饰符

常用于JavaBean类的成员变量，防止其被外部直接访问或修改。

与此同时，会提供相应的getter和setter实现安全访问和修改，维护了数据的安全性。

```
public class Circle {
    private double radius;

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

     public double getRadius() {
        return radius;
    }

    public void setRadius(double radius) {
        this.radius = radius;
    }

}
```

`protected`

作为访问权限修饰符

通常用于父类的变量，方法的定义中，此时子类就可以直接访问到父类的变量与方法。

```
public abstract class AbstractDocument {
    protected final String identifier;
    protected int accessCount;

    public AbstractDocument(String identifier, int accessCount) {
        this.identifier = identifier;
        this.accessCount = accessCount;
    }

    protected void increaseAccessCount() {
        accessCount++;
    }
}
```

```
public final class Image extends AbstractDocument {
    public Image(String identifier, int accessCount) {
          super(identifier, accessCount);
    }

    public void display() {
          System.out.println("Displaying image: " + identifier);
          increaseAccessCount();
          System.out.println("Access count: " + accessCount);
    }
}
```

`default`

作为访问权限修饰符

只能在同一个`package`中的文件能够访问

`final`

防止修改或继承

1. 用于修饰变量，则该值只能在声明或构造方法中初始化，后续无法更改
2. 用于修饰方法，则继承该类的子类不能再Override该方法
3. 用于修饰类，则该类为最终实现类，无法被继承

`static`

作为静态声明

1. 常用于修饰变量，则该变量为该类所共享，以该类创建的所有对象都能访问，常用于全局常量

   e.g.`public static final double PI =3.14159;`
2. 常用于修饰方法，则该方法为静态方法，不需要创建具体对象就可以使用，常见于工具类

   e.g.

   ```
   public final class MathUtil {

       private MathUtil() {
           throw new UnsupportedOperationException("工具类不能被实例化");
       }

       public static int max(int a, int b) {
           return (a > b) ? a : b;
   ```

```
    }

    public static int min(int a, int b) {
        return (a < b) ? a : b;
    }
}
```

代码实现

详见BankAccount类