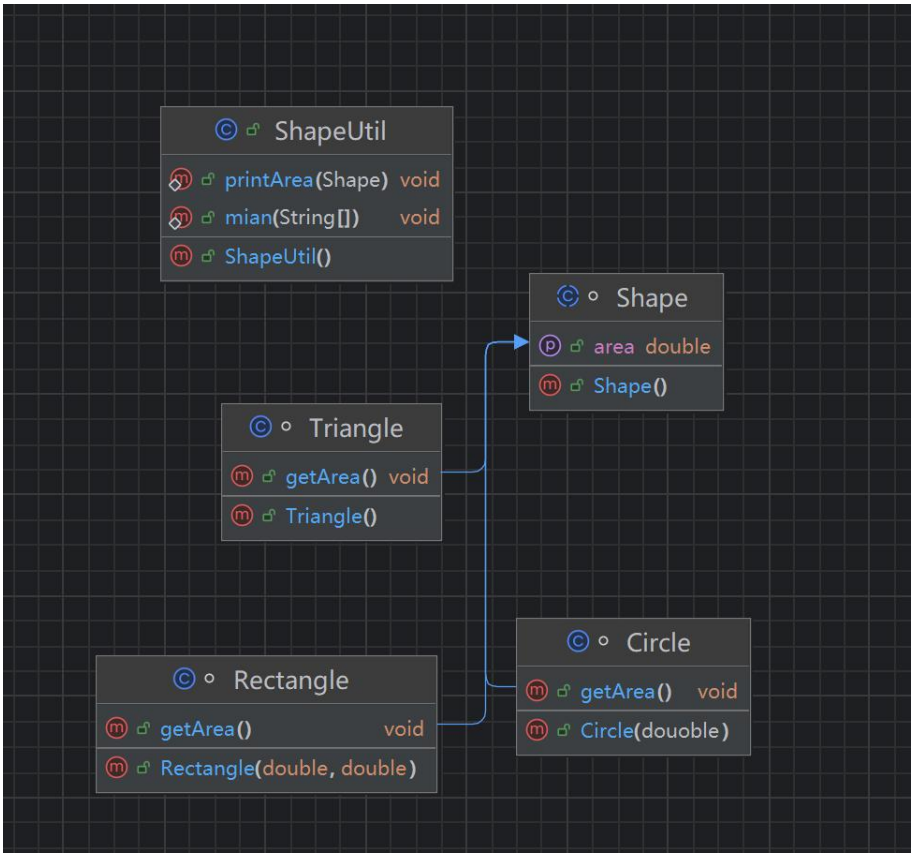
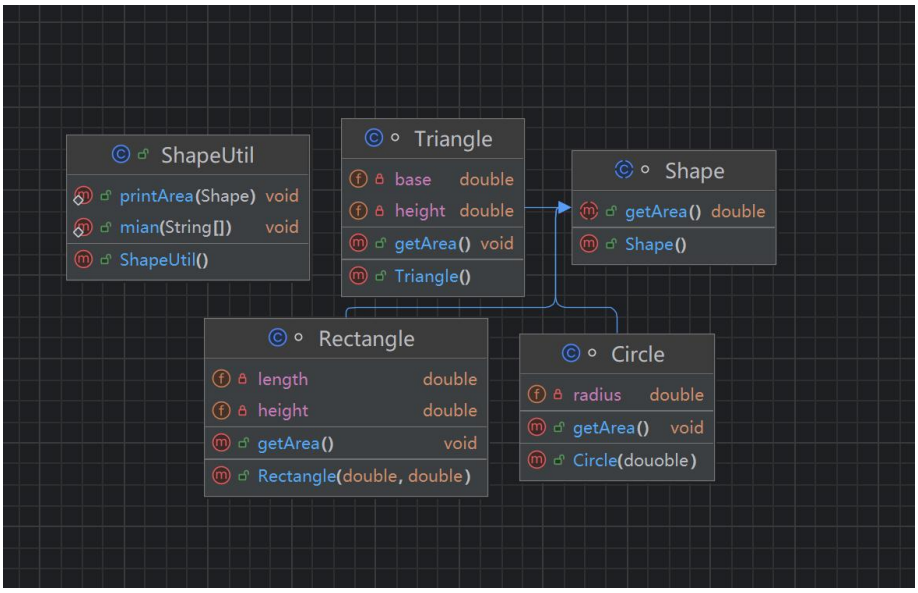
类图

以下展示了 Shape 类及其子类的属性和方法以及修饰符。





核心代码

构造方法

以下代码展示了抽象类和抽象方法 getArea 的设计，使用 abstract 修饰，让三个图形分别继承 Shape，在子类中创建字段、构造方法、覆写父类的 getArea

方法，以实现不同图形的面积计算。

```
abstract class Shape{
    public abstract double getArea();
}

class Circle extends Shape{
    private double radius;
    public Circle(double radius){
        this.radius = radius;
    }
    @Override
    public double getArea(){
        return Math.PI*radius*radius;
    }
}

class Rectangle extends Shape{
    private double length;
    private double height;
    public Rectangle(double length,double height){
        this.length=length;
        this.height=height;
    }
    @Override
    public double getArea(){
        return length*height;
    }
}

class Triangle extends Shape{
    private double base;
    private double height;
    public Triangle(double base,double height){
        this.base=base;
        this.height=height;
    }
    @Override
    public double getArea(){
        return base*height/2;
    }
}
```

以下代码展示了 ShapeUtil 类提供的 printArea 方法（保留了两位小数），以及主方法的设计编写，创建三个子类的对象，调用 printArea 方法实现。

```
public class ShapeUtil{
    public static void printArea(Shape shape){
        System.out.printf("面积是%.2f%n",shape.getArea());
    }
    public static void main(String[] args){
    Shape circle = new Circle(5.0);
    Shape rectangle = new Rectangle(10.0,20.0);
    Shape triangle = new Triangle(3.0,4.0);
    printArea(circle);
    printArea(rectangle);
    printArea(triangle);
    }
}
```

运行结果截图

正常流程

得到面积结果如下：

```
D:\java\w4>java ShapeUtil
面积是78.54
面积是200.00
面积是6.00
```

非法操作提示

```
D:\java\w4>javac Shape.java
Shape.java:35: 错误：类ShapeUtil是公共的，应在名为 ShapeUtil.java 的文件中声明
public class ShapeUtil{
       ^
Shape.java:6: 错误：找不到符号
        public Circle(douoble radius){
                      ^
  符号：   类 douoble
  位置：类 Circle
Shape.java:4: 错误：Circle不是抽象的，并且未覆盖Shape中的抽象方法getArea()
class Circle extends Shape{
^
Shape.java:9: 错误：Circle中的getArea()无法覆盖Shape中的getArea()
        public void getArea(){
                    ^
  返回类型void与double不兼容
Shape.java:7: 错误：不兼容的类型：意外的返回值
                return this.radius = radius;
                                   ^
Shape.java:10: 错误：不兼容的类型：意外的返回值
                return Math.PI*radius*radius;
                                     ^
Shape.java:13: 错误：Rectangle不是抽象的，并且未覆盖Shape中的抽象方法getArea()
class Rectangle extends Shape{
^
Shape.java:20: 错误：Rectangle中的getArea()无法覆盖Shape中的getArea()
        public void getArea(){
                    ^
  返回类型void与double不兼容
Shape.java:17: 错误：不兼容的类型：意外的返回值
                return this.length=length;
                                   ^
Shape.java:18: 错误：不兼容的类型：意外的返回值
                return this.height=height;
                                   ^
Shape.java:21: 错误：不兼容的类型：意外的返回值
                return length*height;
                                   ^
Shape.java:24: 错误：Triangle不是抽象的，并且未覆盖Shape中的抽象方法getArea()
class Triangle extends Shape{
```

```
D:\java\w4>javac ShapeUtil.java
ShapeUtil.java:8: 错误: 不兼容的类型: 意外的返回值
                return this.radius = radius;
                                   ^
ShapeUtil.java:20: 错误: 不兼容的类型: 意外的返回值
                return this.length=length;
                                   ^
ShapeUtil.java:21: 错误: 不兼容的类型: 意外的返回值
                return this.height=height;
                                   ^
ShapeUtil.java:33: 错误: 不兼容的类型: 意外的返回值
                return this.base=base;
                                  ^
ShapeUtil.java:34: 错误: 不兼容的类型: 意外的返回值
                return this.height=height;
                                   ^
5 个错误
```

原始代码存在多处关键错误：首先，抽象类 Shape 中的 getArea() 方法被错误地定义为带参数且返回 void，而正确做法应是无参并返回 double 类型；其次，子类（如 Circle、Rectangle、Triangle）在重写该方法时也添加了参数并使用 void 返回类型，导致方法签名不匹配，无法构成有效重写，从而引发“未实现抽象方法”的编译错误；此外，构造函数中错误地使用了 return this.radius = radius; 等语句，而构造函数没有返回值，不应使用 return 表达式，只需直接赋值即可；同时，面积计算逻辑混乱（如长方形用了三角形公式），且 public class ShapeUtil 被放在 Shape.java 文件中，违反了 Java “公共类必须与文件同名”的规则。针对这些问题，统一修正如下：将 Shape 中的抽象方法定义为 public abstract double getArea();，所有子类通过构造函数初始化属性并在无参的 getArea() 方法中返回正确的面积值（如圆用 Math.PI * r²，长方形用 长×宽，三角形用 底×高÷2），构造函数仅做赋值操作，不使用 return，并将 ShapeUtil 改为非公共类以允许与主类共存于同一文件，最后在输出时使用 System.out.printf("面积是：%.2f%n", shape.getArea()); 实现保留两位小数。修正后代码结构清晰、符合面向对象设计原则，可正确编译运行并输出格式化结果。

组合 VS 继承？

组合与继承是面向对象编程中实现代码复用的两种基本方式，各有优劣。继承体现“is-a”关系，子类可直接复用父类的属性和方法，并天然支持多态，适用于具有明确层级结构的场景（如 Circle 是 Shape 的一种），但其缺点在于紧耦合、破坏封装性，且 Java 不支持多继承，一旦父类变更可能引发连锁问题，容易因误用“为复用而继承”导致设计脆弱。相比之下，组合体现“has-a”关系，通过在一个类中持有另一个类的实例来复用功能，具有松耦合、高灵活性、良好封装性和支持多源复用等优势，虽需手动委托调用且代码略多，但更安全、更易维护。因此，现代软件设计普遍遵循“优先使用组合而非继承”的原则，在确保语义合理的前提下，仅在真正存在稳定“is-a”关系且需要多态时才使用继承，其余情况优先选择组合以提升系统的可扩展性与健壮性。

实验总结

本次实验围绕面向对象编程中的抽象类与多态机制展开，通过设计一个图形面积计算系统，深入理解了抽象类的定义、子类对抽象方法的重写规则以及 Java 语言的语法规范。在初始编码过程中，出现了多个典型错误：包括抽象方法误加参数和使用 void 返回类型、子类未正确重写父类方法（方法签名不一致）、构

造函数中错误使用 return 赋值语句、面积计算逻辑混淆，以及公共类命名与文件名不匹配等问题。这些问题导致程序无法编译通过。

通过对错误信息的逐条分析，我们明确了 Java 对抽象方法实现、构造函数语法和类文件组织的严格要求。最终采用规范的面向对象设计进行修正：定义无参且返回 double 的抽象方法 getArea()；各子类通过构造函数初始化自身属性，并在重写的 getArea() 方法中返回正确的面积值；移除构造函数中的 return 语句；使用 System.out.printf("%.2f") 实现保留两位小数的格式化输出；并将工具类设为非公共类以适配单文件结构。修正后的程序成功运行，准确输出各类图形的面积。

本实验不仅巩固了抽象类、继承与多态的核心概念，也提升了调试能力和对 Java 编码规范的理解，为后续复杂面向对象系统的设计奠定了坚实基础。