Design Patterns - Hallmarks of Good Architecture

SOLID Principles of **Object-Oriented Programming**

Practice Session

Open/Closed Principle (OCP)

Design Patterns - Hallmarks of Good Architecture

```
class Shape {
  String type;
   Shape(this.type);
}

class AreaCalculator {
  double calculateArea(Shape shape) {
    if (shape.type == 'circle') {
      // calculate area of circle
    } else if (shape.type == 'square') {
      // calculate area of square
    }
    // ...
  }
}
```

Open/Closed Principle (OCP)

Hints:

1. Identify the parts of the code **that change** when a new type of object is introduced.
2. Create an **abstraction** (interface or abstract class) for these objects and define the behavior that varies in this abstraction.
3. **Implement this abstraction** in each of the object classes, providing their own implementation of the behavior.
4. Use the **abstraction instead of the concrete classes** where the behavior is needed.

Design Patterns - Hallmarks of Good Architecture

```
abstract class Shape {
  double calculateArea();
}

class Circle extends Shape {
  double radius;
  Circle(this.radius);

  @override
  double calculateArea() {
    return 3.14 * radius * radius;
  }
}
```

```
class Square extends Shape {
  double side;
  Square(this.side);

  @override
  double calculateArea() {
    return side * side;
  }
}

class AreaCalculator {
  double calculateArea(Shape shape) {
    return shape.calculateArea();
  }
}
```

Design Patterns - Hallmarks of Good Architecture

Open/Closed Principle (OCP)

1. In the refactored solution, we have an abstract `Shape` class with an abstract `calculateArea` method.
2. Then, we have the `Circle` and `Square` classes, which are concrete implementations of the `Shape` class.
3. Each of these classes overrides the `calculateArea` method to provide its own implementation.
4. Finally, we have the `AreaCalculator` class that uses the `Shape` class to calculate the area, so it doesn't need to know the specific type of shape.
5. The bad code violated the Open/Closed Principle because the `AreaCalculator` class was not closed for modification.
6. Every time we wanted to add a new shape, we had to modify the `AreaCalculator` class.
7. This made the class difficult to maintain, and also increased the risk of introducing bugs.