SOLID Principles: Assignment — Liskov Substitution Principle (LSP) in Shapes

**Goal**: identify an LSP violation, explain why it breaks substitutability, redesign the model so every shape follows the original contract, and state the architecture/pattern applied.

#### **Starter code (keep unchanged in src/main/java/legacy/)**

package legacy;

public class Shape { public double getArea() { return 0.0; } }

package legacy;

public class Rectangle extends Shape {

private double width, height;

public Rectangle(double w,double h){this.width=w;this.height=h;}

@Override public double getArea(){return width\*height;}

public void setWidth(double w){this.width=w;}

public void setHeight(double h){this.height=h;}

}

package legacy;

public class Square extends Rectangle {

public Square(double side){super(side,side);}

@Override public void setWidth(double w){super.setWidth(w);super.setHeight(w);}

@Override public void setHeight(double h){super.setWidth(h);super.setHeight(h);}

}

#### **Tasks**

1. Analyse the violation: list every broken contract in analysis/lsp\_problems.md and show a short code snippet that reveals wrong behaviour.
2. Write a failing JUnit test (SquareBehavesLikeRectangleTest) that replaces a Rectangle with a Square and asserts area expectations.
3. Refactor under src/main/java/clean/:  
   * Introduce an **abstract** Shape contract (getArea).
   * Implement independent Rectangle and Square classes, each honouring its own invariants.
   * Use **composition instead of inheritance**; this is the *pattern* you must document.
4. Create a new shape Circle without touching existing classes; prove polymorphic use in Main.java.
5. Reflection: in reflection.md answer  
   * How did the original design break LSP?
   * Why does composition restore substitutability?
   * What architecture pattern did you use and where might you apply it elsewhere?

#### **Deliverables**

analysis/lsp\_problems.md

src/main/java/clean/\*\* ← refactored code + Circle

src/test/java/\*\* ← failing then passing tests

src/main/java/Main.java ← demo: Rectangle, Square, Circle polymorphism

reflection.md

README.md ← build/run instructions

#### **Solution reference (clean module)**

package clean;

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

package clean;

public final class Rectangle extends Shape {

private double width, height;

public Rectangle(double w,double h){this.width=w;this.height=h;}

@Override public double getArea(){return width\*height;}

// setters follow SRP; do not link width/height

public void setWidth(double w){this.width=w;}

public void setHeight(double h){this.height=h;}

}

package clean;

public final class Square extends Shape {

private double side;

public Square(double side){this.side=side;}

@Override public double getArea(){return side\*side;}

public void setSide(double side){this.side=side;}

}

package clean;

public final class Circle extends Shape {

private double radius;

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

@Override public double getArea(){return Math.PI\*radius\*radius;}

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

}

package clean;

public class Main {

public static void main(String[] args){

Shape rect=new Rectangle(4,5);

Shape sq=new Square(4);

Shape circle=new Circle(3);

System.out.println(rect.getArea()); //20

System.out.println(sq.getArea()); //16

System.out.println(circle.getArea()); //28.27…

}

}

*Architecture pattern used*: **Composition over Inheritance** (a strategy to satisfy LSP by modelling independent behaviours instead of forcing an “is-a” hierarchy).