COMP 3550

5.3 — SOLID (PART 1: SRP, OCP, LSP)

Week 5: Design Principles & Refactoring

WHAT IS SOLID?

- *A Design Philosophy for Better Code*

- an acronym for five design principles that help you write:
  - Maintainable
  - Scalable
  - Understandable
  - Testable software

WHAT IS SOLID?

- *A Design Philosophy for Better Code*

- an acronym for five design principles that help you write:
  - Maintainable
  - Scalable
  - Understandable
  - Testable software

S Single Responsibility Principle

O Open Closed Principle

L Liskov Substitution Principle

I Interface Segregation Principle

D Dependency Inversion Principle

WHAT IS SOLID?

- *A Design Philosophy for Better Code*

- an acronym for five design principles that help you write:
  - Maintainable
  - Scalable
  - Understandable
  - Testable software
- Where did it come from?
  - Introduced by Robert C. Martin (Uncle Bob)
  - Popularized in the early 2000s in object-oriented programming circles
- **NOTE:**
  - These are guidelines, not laws.
  - They're tools to evaluate and improve design, not dogma.

S — SINGLE RESPONSIBILITY PRINCIPLE (SRP)

- A class should have one reason to change

Credit: Gabriel Schenker via Derick Bailey, CC-SA 3.0

S — SINGLE RESPONSIBILITY PRINCIPLE (SRP)

```
public class AreaCalculator {
    private ArrayList<Shape> shapes;
    public AreaCalculator() {
        shapes = new ArrayList<Shape> ();
        // fill List from data file
    }

    public double sumAreas() {
        double sum = 0;
        for(Shape s : shapes) {
            if(s instanceof Square) {
                sum += Math.pow(((Square)s).getLength(), 2);
            }
            else if(s instanceof Circle) {
                sum += Math.PI *
                    Math.pow(((Circle)s).getRadius(), 2);
            }
        }
        return sum;
    }

    public void printTotalSum() {
        System.out.println("Sum of all Shape Areas = " + sumAreas());
    }
}
```

S — SINGLE RESPONSIBILITY PRINCIPLE (SRP)

```java
public class AreaCalculator {
    private ArrayList<Shape> shapes;
    public AreaCalculator() {
        shapes = new ArrayList<Shape> ();
        // fill List from data file
    }

    public double sumAreas() {
        double sum = 0;
        for(Shape s : shapes) {
            if(s instanceof Square) {
                sum += Math.pow(((Square)s).getLength(), 2);
            }
            else if(s instanceof Circle) {
                sum += Math.PI *
                    Math.pow(((Circle)s).getRadius(), 2);
            }
        }
        return sum;
    }
}
```

```java
public class SumOutputter {
    private AreaCalculator calculator;
    public SumOutputter() {
        calculator = new AreaCalculator();
    }

    public void outputConsole() {
        System.out.println("Sum of all Shape Areas = "
                + sumAreas());
    }

    public void outputJSONFormat() {
        System.out.println("sum: " + sumAreas());
    }
}
```

O — OPEN/CLOSED PRINCIPLE (OCP)

- Open to extension, closed to modification

Credit: Maxim Ivanov

O — OPEN/CLOSED PRINCIPLE (OCP)

```java
public static void useTool(ArrayList<Tool> tools) {
    for(Tool tool : tools) {
        if(tool instanceof Hammer) {
            System.out.println("Bang! Bang!");
        }
        else if(tool instanceof ScrewDriver) {
            System.out.println("Twist! Twist!");
        }
        else if(tool instanceof Scissors) {
            System.out.println("Cut! Cut!");

        }
    }
}
```

O — OPEN/CLOSED PRINCIPLE (OCP)

```java
public static void useTool(ArrayList<Tool> tools) {
    for(Tool tool : tools) {
        System.out.println(tool.toolSound());
    }
}
```

L — LISKOV SUBSTITUTION PRINCIPLE (LSP)

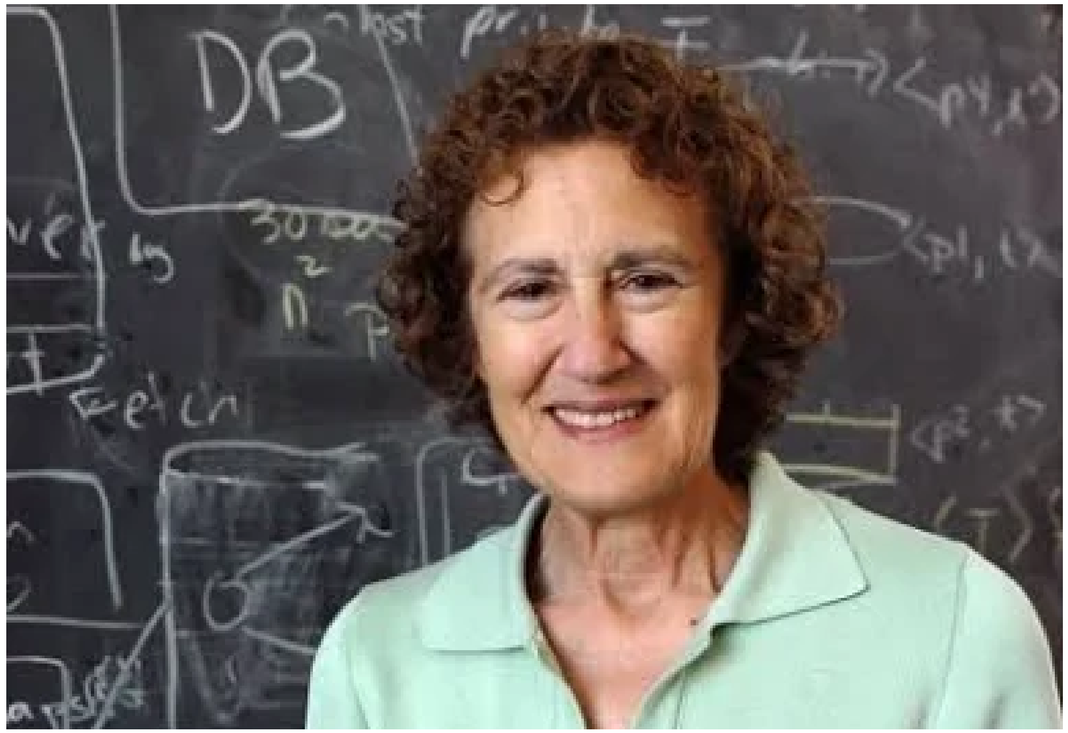
- Subclass should behave like parent class

Credit: Derick Bailey, CC-SA 3.0

L — LISKOV SUBSTITUTION PRINCIPLE (LSP)

- Barbara Liskov of the LSP

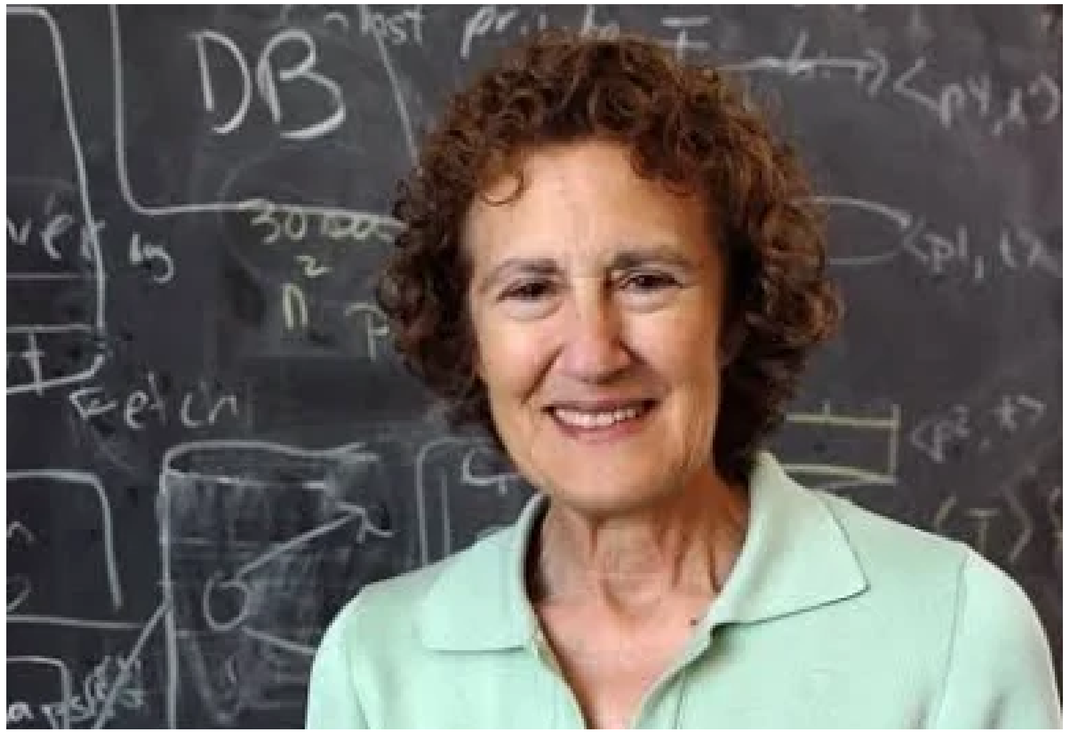
- First Woman in the U.S. to Earn a PhD in Computer Science
  - Her thesis?



L — LISKOV SUBSTITUTION PRINCIPLE (LSP)

- Barbara Liskov of the LSP

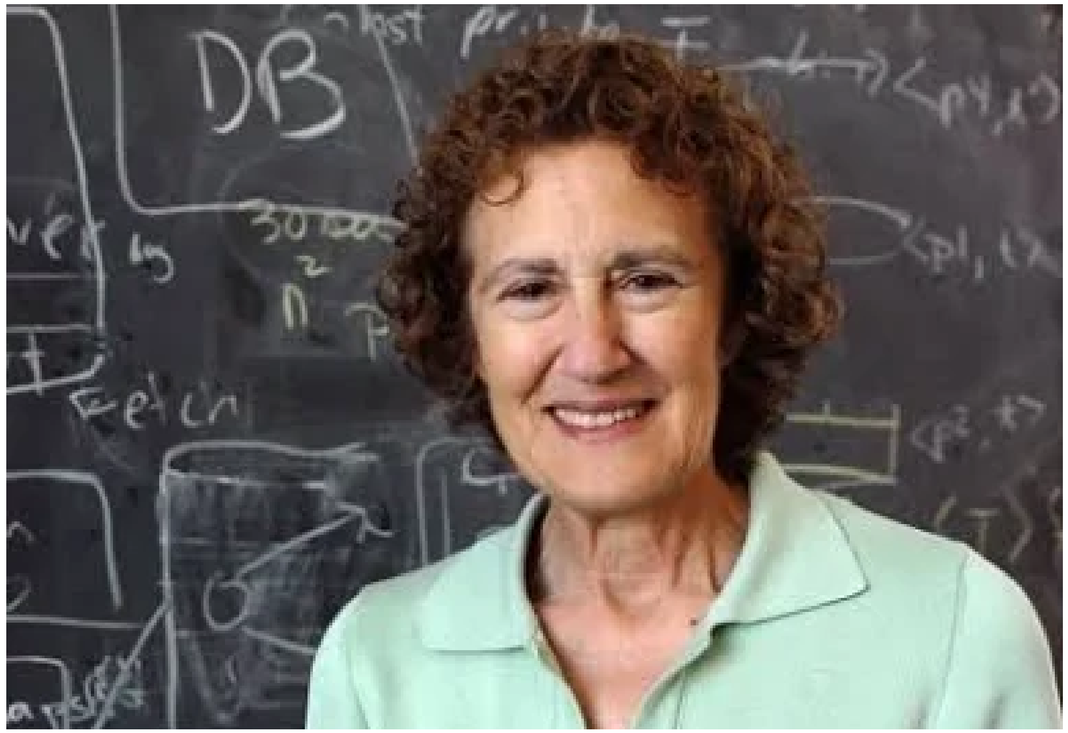
- First Woman in the U.S. to Earn a PhD in Computer Science
  - Her thesis?
  - On artificial intelligence — before it was trendy.



L — LISKOV SUBSTITUTION PRINCIPLE (LSP)

- Barbara Liskov of the LSP

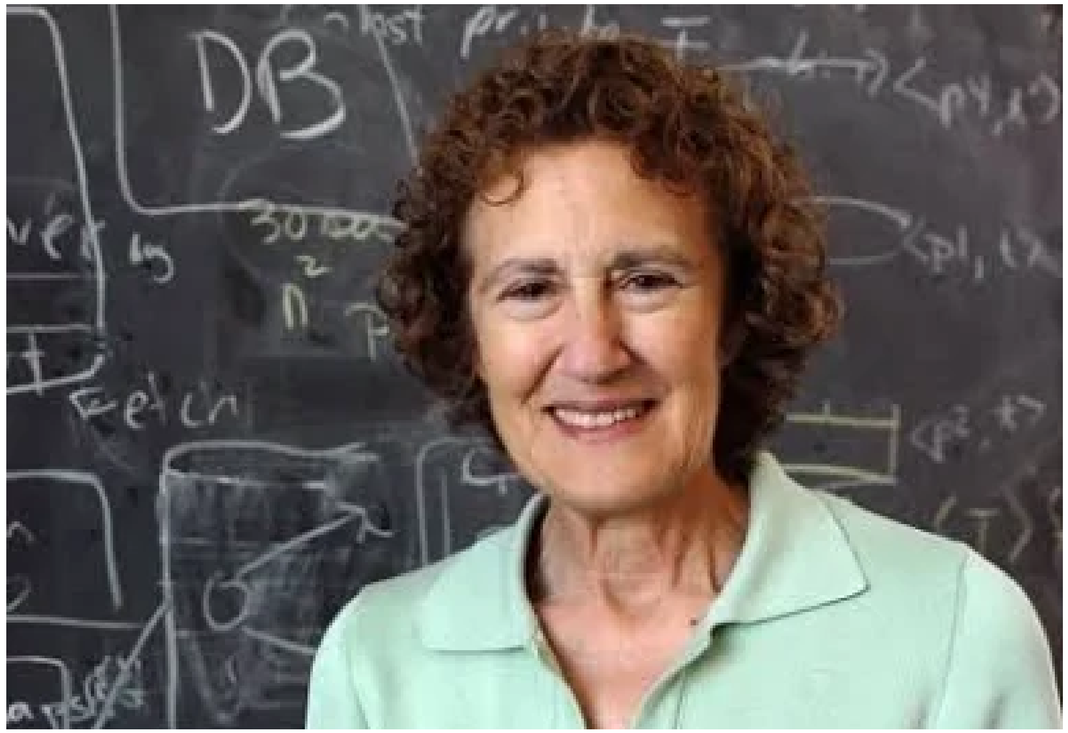
- First Woman in the U.S. to Earn a PhD in Computer Science
  - Her thesis?
  - On artificial intelligence — before it was trendy.
- Did she *really* name a principle after herself?



L — LISKOV SUBSTITUTION PRINCIPLE (LSP)

- Barbara Liskov of the LSP

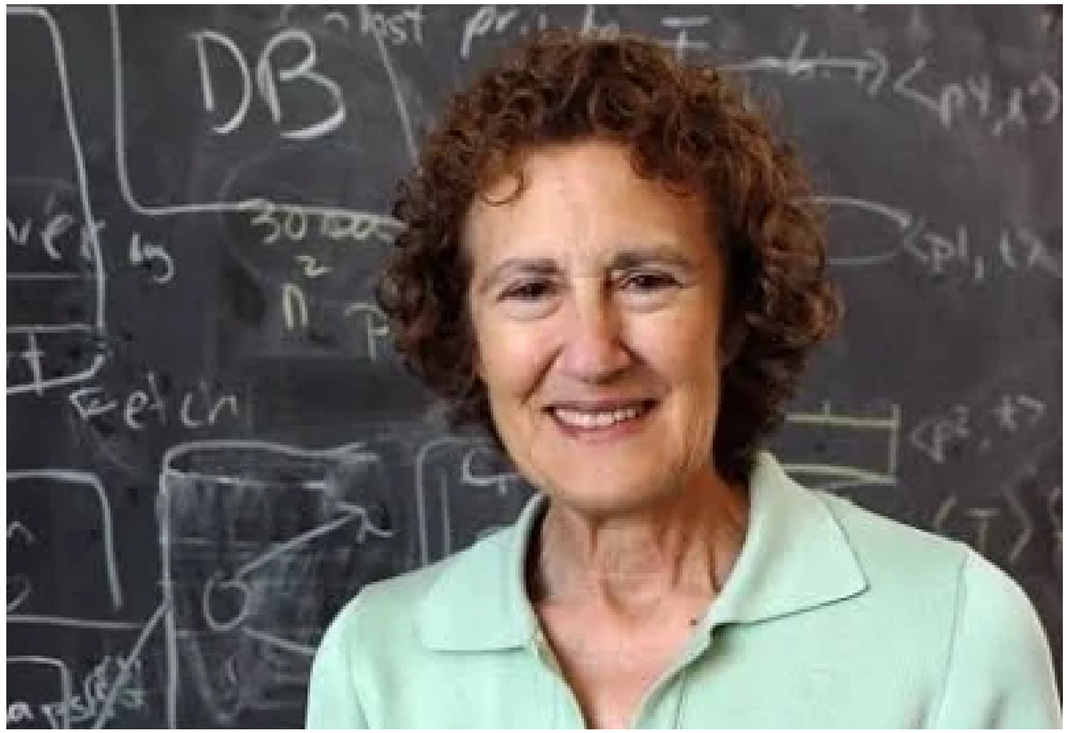
- First Woman in the U.S. to Earn a PhD in Computer Science
  - Her thesis?
  - On artificial intelligence — before it was trendy.
- Did she *really* name a principle after herself?
  - **Nope!**
  - She never coined “Liskov Substitution Principle” — that was Robert C. Martin, naming it in honor of her work on behavioral subtyping and robust interface design.



L — LISKOV SUBSTITUTION PRINCIPLE (LSP)

- Barbara Liskov of the LSP

- First Woman in the U.S. to Earn a PhD in Computer Science
  - Her thesis?
  - On artificial intelligence — before it was trendy.
- Did she *really* name a principle after herself?
  - **Nope!**
  - She never coined “Liskov Substitution Principle” — that was Robert C. Martin, naming it in honor of her work on behavioral subtyping and robust interface design.
- Turing Award Winner
  - 2008, pioneering work in programming languages and system design, including the creation of the CLU language



L — LISKOV SUBSTITUTION PRINCIPLE (LSP)

```
class FileReader {
    public String readFile(String path) throws IOException {
        return "File contents";
    }
}
```

```
class SecureFileReader extends FileReader {
    @Override
    public String readFile(String path) throws SecurityException {
        throw new SecurityException("Access Denied!");
    }
}
```

```
FileReader reader = new SecureFileReader();
try {
    reader.readFile("secret.txt");
} catch (IOException e) {
    System.out.println("Handled IO error");  // This won't catch it!
}
```

L — LISKOV SUBSTITUTION PRINCIPLE (LSP)

LSP isn't just about inheritance — it's about behavioral compatibility.

- **Fix 1:** Throw IOException Instead
  - Use IOException or a subclass of it to stay within the expected contract

```
class SecureFileReader extends FileReader {
    @Override
    public String readFile(String path) throws IOException {
        if (!hasPermission(path)) {
            throw new IOException("Access denied for: " + path);
        }
        return super.readFile(path);
    }

    private boolean hasPermission(String path) {
        // fake logic
        return false;
    }
}
```

L — LISKOV SUBSTITUTION PRINCIPLE (LSP)

LSP isn't just about inheritance — it's about behavioral compatibility.

- **Fix 2:** Wrap the Unexpected Exception

```java
@Override
public String readFile(String path) throws IOException {
    try {
        if (!hasPermission(path)) {
            throw new SecurityException("Access denied!");
        }
        return super.readFile(path);
    } catch (SecurityException se) {
        throw new IOException("Security violation: " + se.getMessage(), se);
    }
}
```

L — LISKOV SUBSTITUTION PRINCIPLE (LSP)

```
class Rectangle {
    protected int width;
    protected int height;

    public void setWidth(int w) { width = w; }
    public void setHeight(int h) { height = h; }

    public int getArea() { return width * height; }
}
```

```
class Square extends Rectangle {
    @Override
    public void setWidth(int w) {
        width = w;
        height = w; // Surprise! Sets both
    }

    @Override
    public void setHeight(int h) {
        height = h;
        width = h; // Surprise! Sets both
    }
}
```

```
Rectangle r = new Square();
r.setWidth(5);
r.setHeight(10);
System.out.println(r.getArea());
```

L — LISKOV SUBSTITUTION PRINCIPLE (LSP)

LSP isn't just about inheritance — it's about behavioral compatibility.

- **Fix 3:** Composition Over Inheritance
  - Instead of extending Rectangle, Square wraps a Rectangle or defines its own behavior separately.

```
class Rectangle {
    protected int width;
    protected int height;

    public void setWidth(int w) { width = w; }
    public void setHeight(int h) { height = h; }

    public int getArea() { return width * height; }
}

class Square {
    private int side;

    public void setSide(int s) { side = s; }
    public int getArea() { return side * side; }
}
```

L — LISKOV SUBSTITUTION PRINCIPLE (LSP)

LSP isn't just about inheritance — it's about behavioral compatibility.

- **Fix 4:** Generalize a Shape Interface or Abstract Class
  - If you want polymorphism, share like behaviours or properties **only**

```
interface Shape {
    int getArea();
}
```

L — LISKOV SUBSTITUTION PRINCIPLE (LSP)

LSP isn't just about inheritance — it's about behavioral compatibility.

- **but why not...**

```
public class Rectangle extends Shape {
    private int width, height;
    // ...
}

public class Square extends Shape {
    private int side;
    // ...
}

public class Circle extends Shape {
    private double radius;
    // ...
}
```

L — LISKOV SUBSTITUTION PRINCIPLE (LSP)

LSP isn't just about inheritance — it's about behavioral compatibility.

- **but why not...**

```
public abstract class Shape {
    protected int width;
    protected int height;

    public abstract int getArea();
}
```

Argument?
YAGNI?

L — LISKOV SUBSTITUTION PRINCIPLE (LSP)

LSP isn't just about inheritance — it's about behavioral compatibility.

- **What about Circles? Triangles?**
- What if you **know** you will never need them?
- Even in a rectangles-only world, squares don't behave like rectangles if you override setters or rely on width/height being independent. That's **still** the LSP trap, regardless of YAGNI.

```
Shape s = new Square();
s.setWidth(5);
s.setHeight(10);
System.out.println(s.getArea());
// Still gives wrong/astonishing result
```

So while YAGNI says "don't design for shapes you don't have," LSP says "don't lie about the ones you do."

L — LISKOV SUBSTITUTION PRINCIPLE (LSP)

A Summary

- **Fix 1:** Throw IOException Instead
  - Use IOException or a subclass of it to stay within the expected contract
- **Fix 2:** Wrap the Unexpected Exception
  - If you must use SecurityException, you can wrap it in an IOException:
- **Fix 3:** Composition Over Inheritance
  - Instead of extending Rectangle, Square wraps a Rectangle or defines its own behavior separately.
- **Fix 4:** Generalize a Shape Interface or Abstract Class
  - If you want polymorphism

Just because "A is a B" in real life doesn’t mean A extends B is safe in code.

SUMMARY TABLE

Principle	Core Idea	Example of Violation
SRP – Single Responsibility	A class should have **one reason to change**	A class that handles both UI layout **and** database access
OCP – Open/Closed	Software entities should be **open for extension, closed for modification**	Adding a new payment type requires editing a core `PaymentProcessor` class
LSP – Liskov Substitution	Subtypes should be **fully substitutable** for their base types	A `Square` class that breaks `Rectangle` logic by linking width & height

SPOT THE VIOLATION!

- Read each and identify which principle is being violated: SRP, OCP, or LSP.

Scenario 1:

The ReportGenerator class formats reports, writes them to disk, and sends emails.

Scenario 2:

You add a new type of user role and have to modify the AccessController class to add an if-else block.

Scenario 3:

A subclass overrides a method to throw a different exception than its superclass declares, breaking calling code. add an if-else block.

SPOT THE VIOLATION!

- Read each and identify which principle is being violated: SRP, OCP, or LSP.

Scenario 1:

The ReportGenerator class **formats** reports**, writes** them to disk**, and sends** emails.

Scenario 2:

You add a new **type of user role** and have to **modify** the **AccessController** class to add an if-else block.

Scenario 3:

A **subclass** overrides a method to t**hrow a different exception** than its **superclass** declares, breaking calling code. add an if-else block.

PAUSE & PROJECT REFLECT

- Look for any classes in your code that violate SRP.
- How could you split them up?

- Even if you dont do the refactor for the above code right now
- Make a ticket to not lose track. bring it up at the next meeting and brainstorm ways to fix it