SOLID

Liskov Substitution Principle

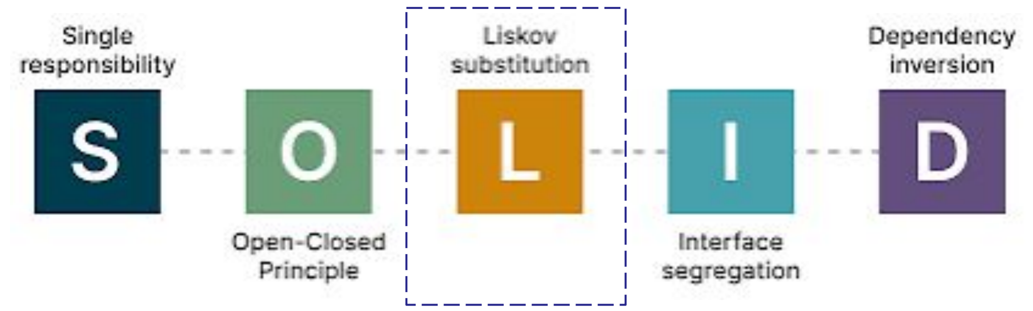
Upcode Software
Engineer Team

-2023-

CONTENT

SINGLE RESPONSIBILITY
OPEN CLOSE
LISKOV SUBSTITUTION
INTERFACE SEGREGATION
DEPENDENCY INVERSION

1. What is LSP?



- **Robert Martin** introduced them in the book Agile Software Development, Principles, Patterns, and Practices
- Simply put, the **Liskov Substitution Principle (LSP)** states that objects of a superclass should be replaceable with objects of its subclasses without breaking the application.

2. WHY needs to LSP ?

- The Liskov Substitution Principle helps us model good inheritance hierarchies.
- The software is **easier and cheaper to maintain,** easier to understand, faster to develop in a team, and easier to test.
- Following those **principles doesn’t guarantee success,** but avoiding them will lead in most cases to at least **sub-optimal results in terms of functionality**, cost, or both.
- It's now time for the L in SOLID, known as the **Liskov Substitution principle.**

3. HOW to use LSP ? (1/n)

When extending a class, remember that you should be able to pass objects of the subclass in place of objects of the parent class without breaking the client code.

This means that the subclass should remain compatible with the behavior of the superclass. When overriding a method, extend the base behavior rather than replacing it with something else entirely

3. HOW to use LSP? (2/n)

- Parameter types in a method of a subclass should match or be more abstract than parameter types in the method of the superclass.
- The return type in a method of a subclass should match or be a subtype of the return type in the method of the superclass
- A method in a subclass shouldn’t throw types of exceptions which the base method isn’t expected to throw.
- A subclass shouldn’t strengthen pre-conditions.
- A subclass shouldn’t weaken post-conditions.
- Invariants of a superclass must be preserved.

4. WHERE to use LSP? (1/n)

- The Liskov Substitution Principle helps us model good inheritance hierarchies. It helps us prevent model hierarchies that don't conform to the Open/Closed principle.
- Any inheritance model that adheres to the Liskov Substitution Principle will implicitly follow the Open/Closed principle.

4. WHERE to use LSP? (2/n)

- Remember in 5th grade when your maths teacher taught you the formula of squares for rectangles and squares? I always remembered square as the more “formula-friendly” rectangle. It is after all a rectangle with equal sides, isn’t it?
- Now suppose you have a base class Rectangle and I tell you to create a new class Square and maximize reuse. Will you inherit the Rectangle class inside the Square class?
- No matter how tempting it might sound or how well you intend to handle it, this small “reuse” will be totally **WRONG!**

5. SOURCE code (1/n)

```java
public class Rectangle {
   private int height;
   private int width;

   public Rectangle(){
   }
   public Rectangle(int height, int width){
       this.height = height;
       this.width = width;
   }

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

   public void setWidth(int width) {
       this.width = width;
   }
```

5. SOURCE code (2/n)

```
public class Square extends Rectangle{
   @Override
   public void setHeight(int height) {
       super.setHeight(height);
       super.setWidth(height);
   }

   @Override
   public void setWidth(int width) {
       super.setWidth(width);
       super.setHeight(width);
   }
}
```

5. SOURCE code (3/n)

```
public class Application {
   private final Rectangle rectangle;

   public Application(Rectangle rectangle) {
       this.rectangle = rectangle;
   }

   public void calculateArea() {
       System.out.println("Area : " + rectangle.getHeight() * 
rectangle.getWidth());
   }
}
```

5. SOURCE code (4/n)

```
public class ClientCode {
   public static void main(String[] args) {

       Rectangle rectangle = new Rectangle();
       Application application;
       rectangle.setHeight(5);
       rectangle.setWidth(4);

       application = new Application(rectangle);
       application.calculateArea();


       System.out.println("-------------------------------------------------------");
```

5. SOURCE code (6/n)

```
Rectangle rectangle1 = new Rectangle();
        Application application1;
        rectangle1.setHeight(5);
        rectangle1.setWidth(4);

        application1 = new Application(rectangle);
        application1.calculateArea();
    }
}
```

5. Summary

The Liskov Substitution Principle is the third of Robert C. Martin's SOLID design principles. It extends the Open/Closed principle and enables you to replace objects of a parent class with objects of a subclass without breaking the application. This requires all subclasses to behave in the same way as the parent class. To achieve that, your subclasses need to follow these rules:

- Don't implement any stricter validation rules on input parameters than implemented by the parent class.
- Apply at the least the same rules to all output parameters as applied by the parent class.

Reference Resources

1. What is Liskov Substitution Principle?
2. Dive Into Design Patterns (Book)
3. Liskov Substitution Principle explained
4. SOLID Design Principles Explained: The Liskov Substitution Principle with Code Examples

Thank you !

Prepared By Abbos Fayziboev

abbosfayziboev@gmail.com