LISKOV SUBSTITUTION PRINCIPLE

the long road to better code

CONTENT

- What is LSP?
- Why we need that?
- How we do that?
- Example

? WHAT ?


BARBARA LISKOV
Developed the Liskov substitution principle

? WHAT ?

> *Subtypes must be behaviorally substitutable for their base types.*
> *Barbara Liskov, 1988*

- Introduced by Barbara Liskov 1988
- Part of the SOLID-Principles

? WHY ?

Class inheritance and subtype polymorphism are primary mechanisms for supporting the open-close principle.

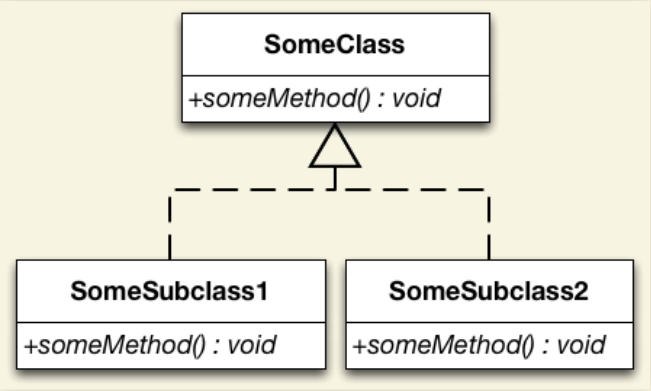
? WHY ?

LSP …

- … gives us a way to characterize good inheritance hierarchies
- … helps us to avoid violations of the open-close principle

? HOW ?

```
void someClientMethod(SomeClass sc)
     ...
     sc.someMethod();
     ...
}
```

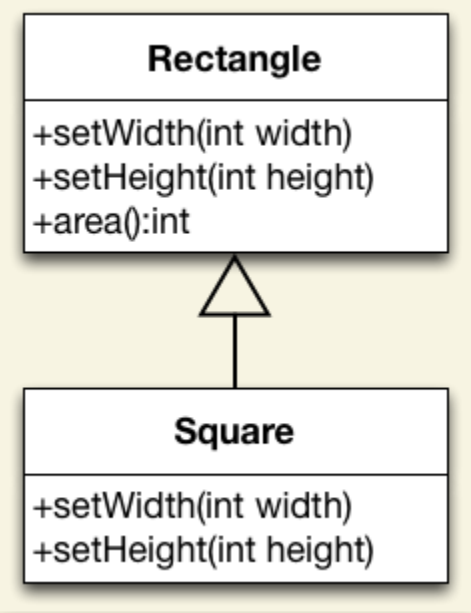


LSP additionally requires behavioral substitutability.

? HOW ?

someClientMethod should not be able to distinguish objects of **SomeSubclass1** and **SomeSubclass2** from objects of **SomeClass**

EXAMPLE




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

    public void setHeight(int height) {
        super.setWidth(height);
        super.setHeight(height);
    }
    ...
}
```

PROBLEMS ?

 Do you see any problems?

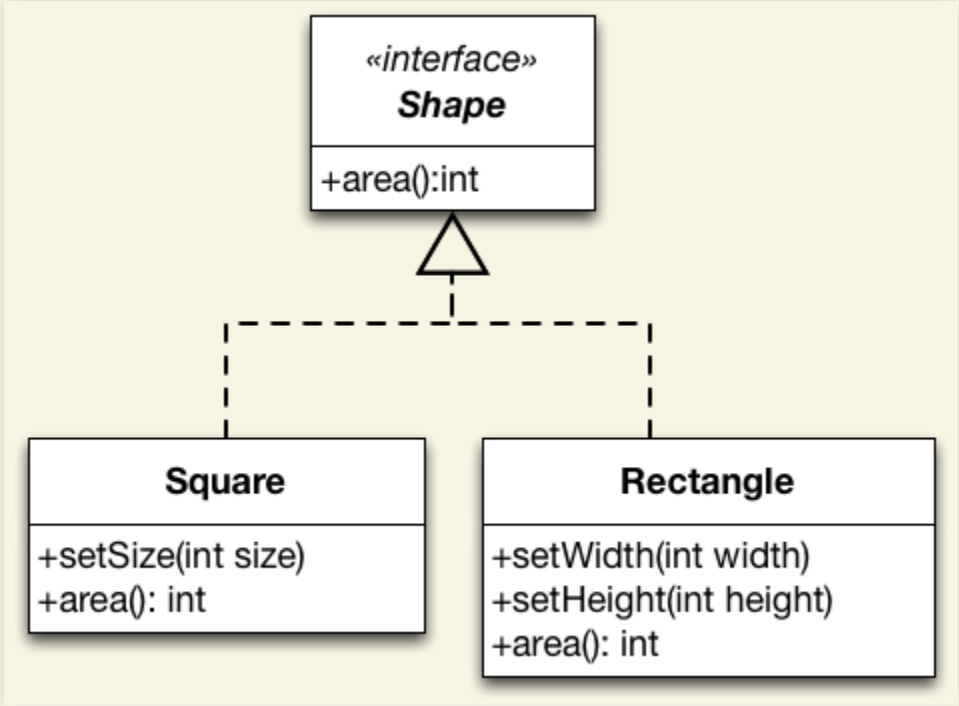
PROBLEMS !

```
void someClientMethod(Rectangle rec) {
    rec.setWidth(5);
    rec.setHeight(4);
    assert(rec.area() == 20);
}
```

PROBLEMS !

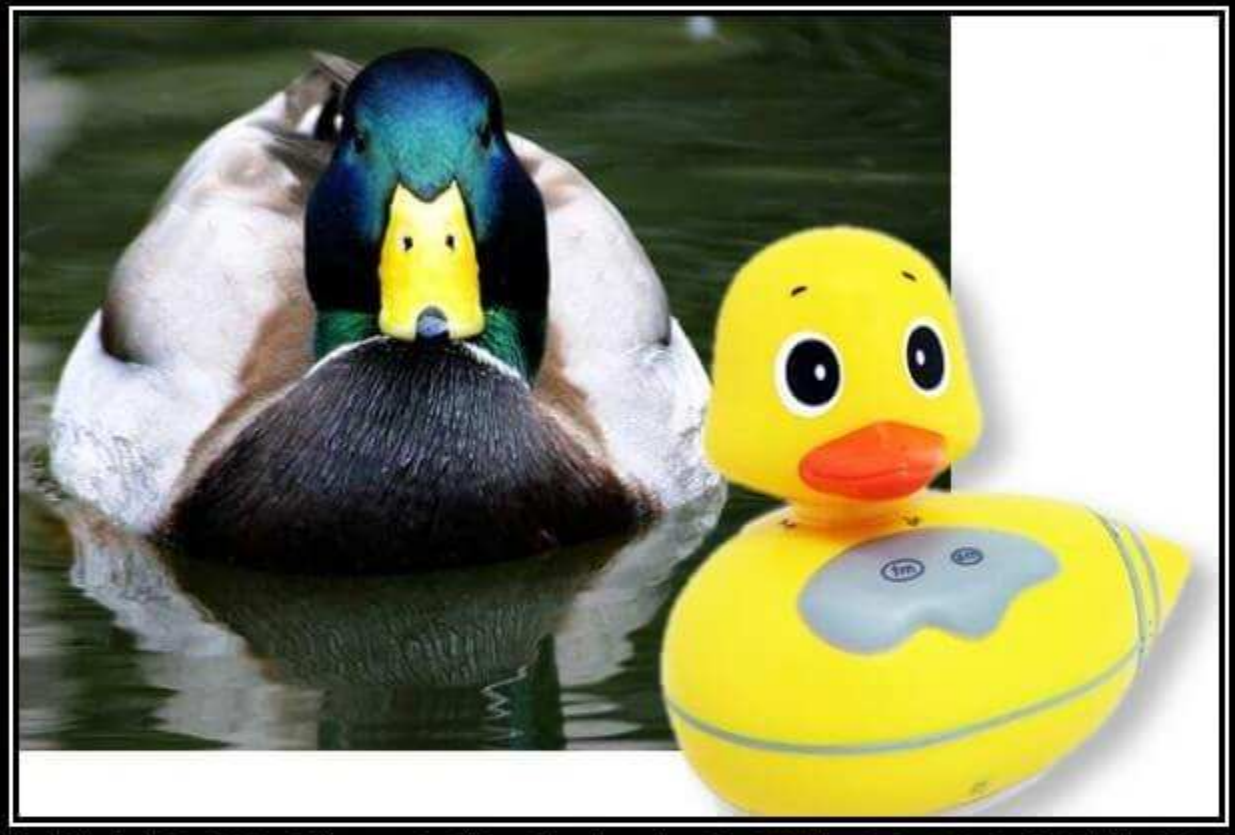
- The behavior of a Square object is not consistent with the expectations of someClientMethod on the behavior of a Rectangle.
- The Rectangle/Square hierarchy violates LSP! Square is **NOT BEHAVIORALLY SUBSTITUTABLE** for Rectangle.

SOLUTION


«interface»
Shape
+area():int
Square
+setSize(int size)
+area(): int
Rectangle
+setWidth(int width)
+setHeight(int height)
+area(): int

CONCLUSION

- LSP violations are difficult to find
- examples could be found everywhere in our code (i guess)
- **Refactoring is hard**



LISKOV SUBSTITUTION PRINCIPLE

If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction