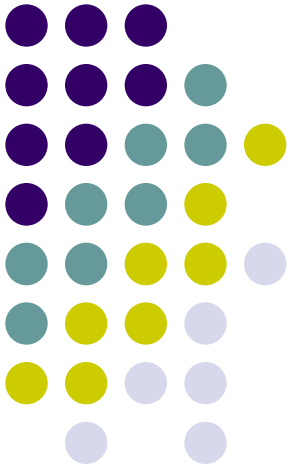
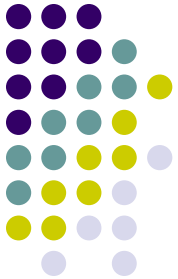
# Software Architecture

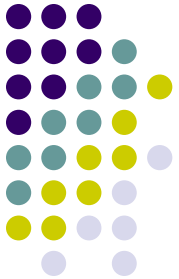
Design Principles – OCP & LSP



# Content

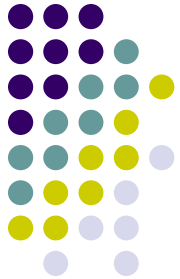


- The Open-Closed Principle (OCP)
- The Liskov Substitution Principle (LSP)



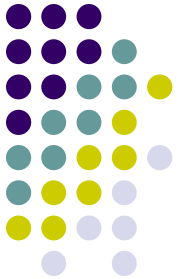
# OCP – The Open-Closed Principle

# OCP – The Open-Closed Principle



- **The open-closed principle:**
  - **Open** for Extension
  - **Closed** for Modification
- What this really means is that you should (re)design so that change leads to extending, not modifying existing code

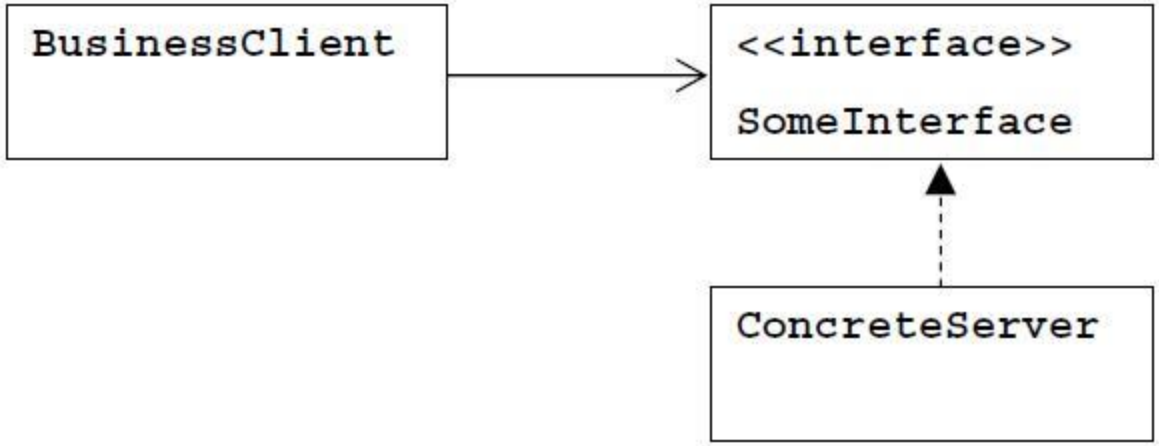
# OCP – An example of violating OCP



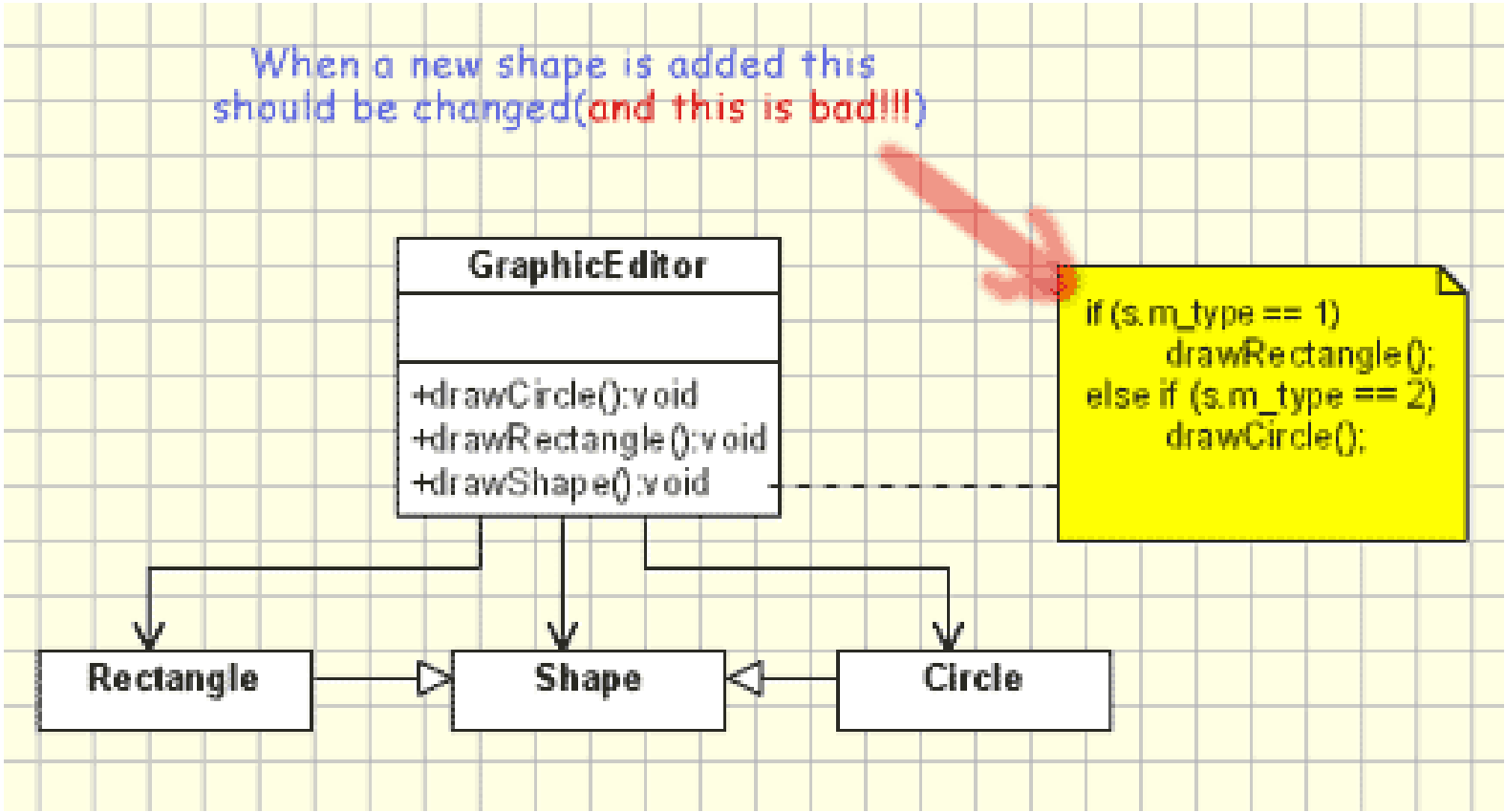
- If the client has a reference to a concrete server-class, replacing the server leads to modification of the client.

# OCP – Conforming to OCP

- If the client has a reference to an interface, replacing the server will not lead to modification of the client. The client still references the interface.



# Example 2


When a new shape is added this
should be changed(and this is bad!!!)
GraphicEditor
+drawCircle():void
+drawRectangle():void
+drawShape():void
if (s.m_type == 1)
drawRectangle();
else if (s.m_type == 2)
drawCircle();
Rectangle
Shape
Circle

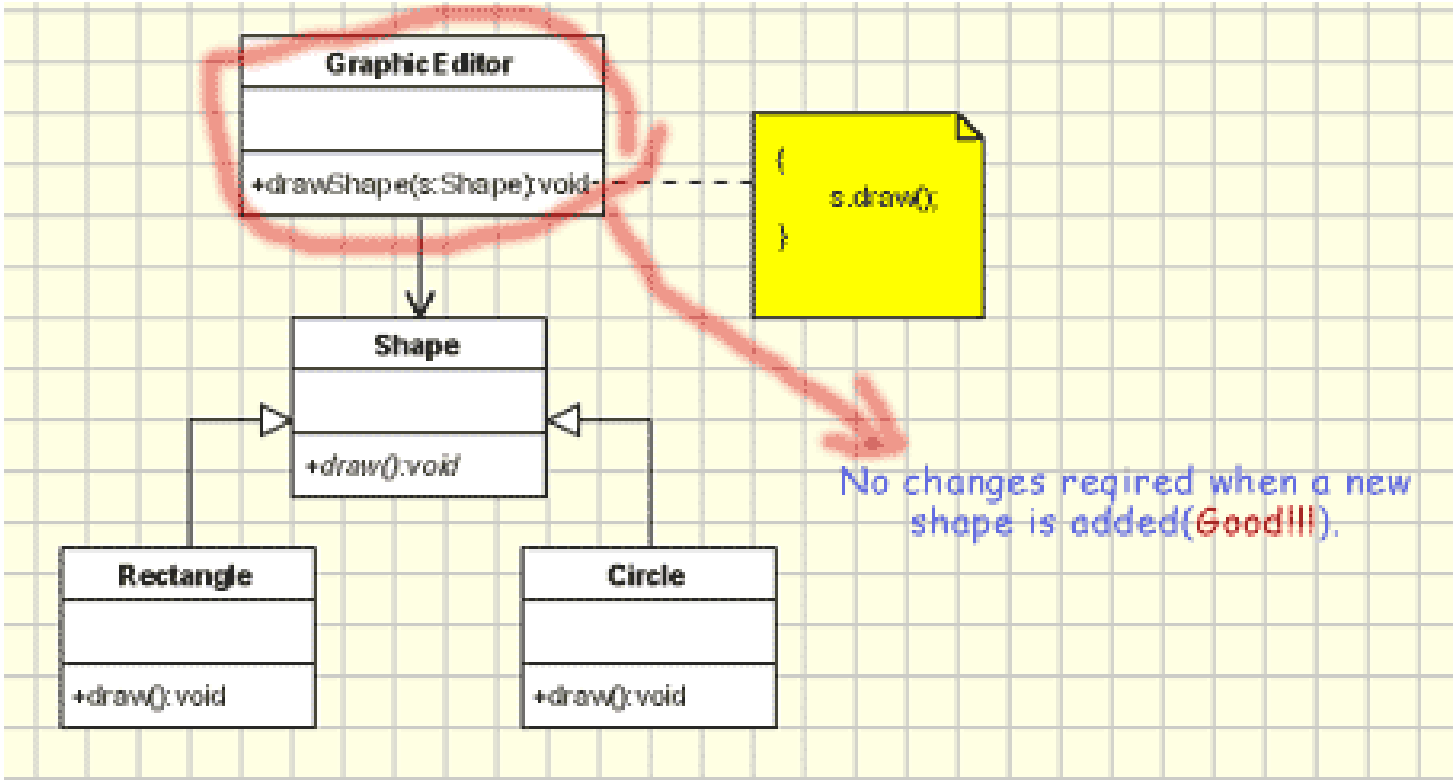
# Example 2

```
// Open-Close Principle - Bad example
class GraphicEditor {
        public void drawShape(Shape s) {
                if (s.m_type==1)
                drawRectangle(s);
                else if (s.m_type==2)
                        drawCircle(s);
        }
        public void drawCircle(Circle r) {....}
        public void drawRectangle(Rectangle r) {....}
}
```

# Example 2

```
class Shape {
        int m_type;
 }
class Rectangle extends Shape {
      Rectangle() {
            super.m_type=1;
      }
}
class Circle extends Shape {
      Circle() {
            super.m_type=2;
      }
}
```

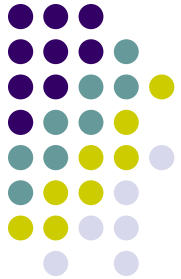
# Example 2


GraphicEditor
+drawShape(s:Shape):void
{
s.draw();
}
Shape
+draw():void
Rectangle
+draw():void
Circle
+draw():void
No changes reqired when a new shape is added(Good!!!).

# Example 2

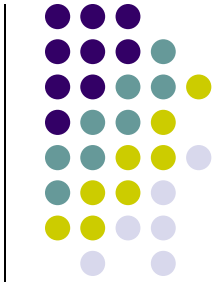
```
// Open-Close Principle - Good example
class GraphicEditor {
        public void drawShape(Shape s) {
        s.draw();
        }
}
class Shape {
    abstract void draw();
}
class Rectangle extends Shape {
    public void draw() {   // draw the rectangle   }
}
```

# OCP



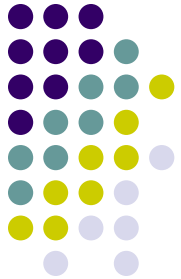
- Conforms to the OCP
- To add a new shape:
  - Open for extension – Add new subclass for new shape
  - Closed for modification – No mod. In drawShape()
- What about the smells?
  - Rigidity – Just add new shape-classes
  - Fragility – No if's or switches to maintain
  - Opacity, Needless Repetition, Immobility, … No problem!
- The code is closed against this particular change

# OCP – Shape may have more issues

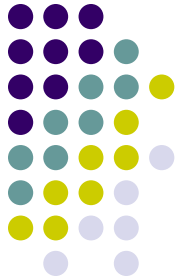


- What if a new requirement states that the shapes must be drawn in some sorted order, e.g. all Circles must be drawn before all Squares:
- No matter how “closed” a module is, there will always be some kind of change against which it is not closed.
- The designer must choose the kinds of changes against which to close his design. Which changes are more likely?
- Plan for OCP, but wait until the change happens!
- To avoid needless complexity, take the first bullet.

# Stimulate the changes



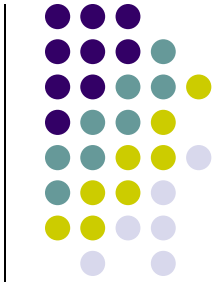
- Stimulate the changes
  - Write tests first.
  - Use short developing cycles
  - Develop features before infrastructure and frequently show those features to stakeholders.
  - Develop the most important features first.
  - Release the software early and often.



# OCP - Summary

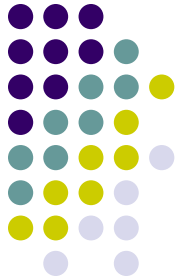
- **Principle**: Software entities should be
  - *Open* for extension – new functions are added
  - *Closed* for modification – existing code is unchanged
- **Implementation**:
  - Find an *abstraction* for what is common in the variation
  - Use *polymorphism* to add varying behavior
  - Abstractions can also be applied in non-OO languages!

# OCP - Summary

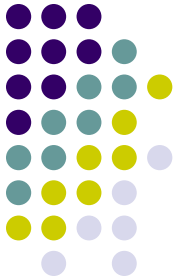


- OCP cannot be achieved for all possible contexts!
  - *Strategic* choices
  - Apply only for *actual changes*

# Question

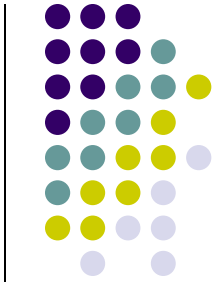


- Please give an example that violates OCP and explain why? How to modify it to conform to OCP?



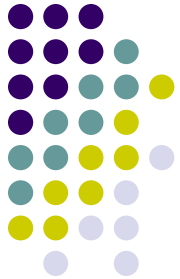
# LSP - Liskov Substitution Principle

# LSP - Liskov Substitution Principle



- The key of OCP: Abstraction and Polymorphism
  - Implemented by inheritance
  - How do we measure the quality of inheritance?
- If for each object ob1 of type S there is an object ob2 of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when ob1 is substituted for ob2 then S is a subtype of T. ***B. Liskov****, 1988*

# LSP - Liskov Substitution Principle

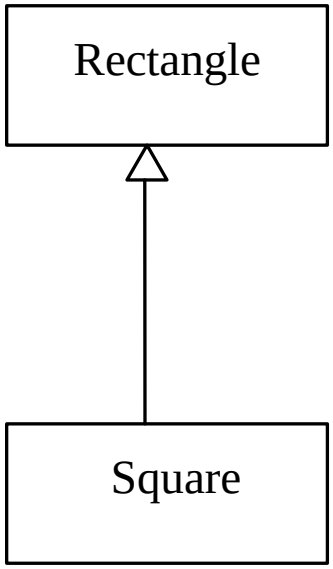


- LSP: Subtypes must be substitutable for their base types.

# Example

```
class Rectangle {
        protected int m_width;
        protected int m_height;
        public void setWidth(int width){
                m_width = width;    }
        public void setHeight(int height){
                m_height = height; }
        public int getWidth(){ return m_width; }
        public int getHeight(){ return m_height; }
        public int getArea(){ return m_width * m_height; }
   }
```

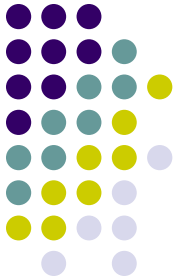
# Example



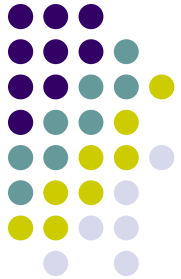

```
class Square extends Rectangle {
    public void setWidth(int width){
        m_width = width;
        m_height = width;
    }
    public void setHeight(int height){
        m_width = height;
        m_height = height;
    }
}
```

# Example

```
class LspTest {
    private static Rectangle getNewRectangle() {
        // it can be an object returned by some factory ...
        return new Square(); }
    public static void main (String args[]) {
        Rectangle r = LspTest.getNewRectangle();
        r.setWidth(5);
        r.setHeight(10);
        // user knows that r it's a rectangle.
        System.out.println(r.getArea());
        // now he's surprised to see that the area is 100
    instead of  //50.
    }
}
```

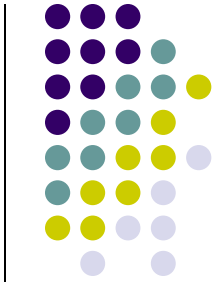


# LSP Conclusion



- Conclusion:
  - A model, viewed in isolation, can’t be meaningfully validated.
  - ISA is about behavior.
    - Behaviorally, a square is not a rectangle.

# Exercise



- Please give an example that violates LSP and explain why? How to modify it to conform to LSP?