



Riza Satria Perdana, S.T., M.T.

Teknik Informatika - STEI ITB

Solid

Solid Principles (Bag. 2)

Pemrograman Berorientasi Objek

Liskov's Substitution Principle

a subclass should behave in such a way that it will not cause problems when uses instead of the superclass

- references to some object (base class) must be able to use objects of classes derived from the base without any knowledge specialized to the derived classes



Liskov's Substitution Principle

```
1 public class CasualEmployee extends Employee {  
2     public String getProjectDetails(int empId) {  
3         super.getProjectDetails(empId);  
4         System.out.println("Casual Employee Project Detail");  
5     }  
6 }  
7  
8 public class ContractEmployee extends Employee {  
9     public String getProjectDetails(int empId) {  
10        // broken the base class  
11        System.out.println("Contract Employee Project Detail");  
12    }  
13 }
```



Substitution Failures

Deriving a square from a rectangle implies that one of the state variables, height or width, is redundant

The behavior of a square - change its height and you change its width - does not apply to rectangles and so square objects are not rectangle objects

```
package com.solidtutorial;
```

```
public class Rectangle {  
    protected int width;  
    protected int height;  
  
    public Rectangle(int width, int height) {  
        this.width = width;  
        this.height = height;  
    }  
  
    public double area() {  
        return width * height;  
    }  
  
    public int getWidth() {  
        return width;  
    }  
  
    public int getHeight() {  
        return height;  
    }  
  
    public void setWidth(int width) {  
        this.width = width;  
    }  
  
    public void setHeight(int height) {  
        this.height = height;  
    }  
}
```

```
package com.solidtutorial;
```

```
public class OtherSquare extends Rectangle {  
    public OtherSquare(int side) {  
        super(side, side); // Calls Rectangle constructor with same value  
                             for width and height  
    }  
  
    @Override  
    public void setWidth(int width) {  
        this.width = width;  
        this.height = width;  
    }  
  
    @Override  
    public void setHeight(int height) {  
        this.width = height;  
        this.height = height;  
    }  
}
```



```

package com.solidtutorial;

import java.util.List;

public class Main {
    public static void processRectangle(Rectangle r) {
        r.setWidth(width:5);
        r.setHeight(height:4);
        System.out.println(r.area());
    }

    Run | Debug
    public static void main(String[] args) {
        AreaCalculator areaCalculator = new AreaCalculator();
        Circle circle = new Circle(radius:5);
        Square rectangle = new Square(length:5);
        ShapesPrinter shapesPrinter = new ShapesPrinter();
        Cube cube = new Cube(length:5);

        List<Shape> shapes = List.of(circle, rectangle, cube);

        int sum = areaCalculator.sum(shapes);
        System.out.println(shapesPrinter.json(sum));
        System.out.println(shapesPrinter.csv(sum));

        Rectangle rectangle2 = new Rectangle(width:5, height:4);
        OtherSquare otherSquare = new OtherSquare(side:5);
        processRectangle(rectangle2);
        processRectangle(otherSquare);
    }
}

```

```

> cd /Users/galuhdipabharata/Developer/00P/SOLID ; /usr/bin/env /Users/galuhdipabharata/.sdkman/candidates/java/17.0.9-amzn/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/galuhdipabharata/Library/Application\ Support/Code/User/workspaceStorage/1569b9166e28afe13ee1e403fad766e3/redhat.java/jdt_ws/SOLID_d6c6c0ea/bin com.solidtutorial.Main {shapes_sum: 253}
shapes_sum,253
20.0
16.0

```

```
~/Developer/00P/SOLID ..... 17:20:34
```

```
> []
```

```
package com.solidtutorial;

public class Rectangle implements Shape{
    private int width;
    private int height;

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

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

    public int getWidth() {
        return width;
    }

    public int getHeight() {
        return height;
    }

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

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

```
package com.solidtutorial;

public class OtherSquare implements Shape {
    private int side;

    public OtherSquare(int side) {
        this.side = side;
    }

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

    @Override
    public double area() {
        return side * side;
    }
}
```



```
package com.solidtutorial;
```

```
import java.util.List;
```

```
public class Main {
    public static void processRectangle(Shape r) {
        // r.setWidth(5);
        // r.setHeight(4);
        System.out.println(r.area());
    }
}
```

Run | Debug

```
public static void main(String[] args) {
    AreaCalculator areaCalculator = new AreaCalculator();
    Circle circle = new Circle(radius:5);
    Square rectangle = new Square(length:5);
    ShapesPrinter shapesPrinter = new ShapesPrinter();
    Cube cube = new Cube(length:5);

    List<Shape> shapes = List.of(circle, rectangle, cube);

    int sum = areaCalculator.sum(shapes);
    System.out.println(shapesPrinter.json(sum));
    System.out.println(shapesPrinter.csv(sum));

    // Rectangle rectangle2 = new Rectangle(5, 4);
    // OtherSquare otherSquare = new OtherSquare(5);
    // processRectangle(rectangle2);
    // processRectangle(otherSquare);

    Rectangle rect = new Rectangle(width:5, height:4);
    OtherSquare otherSquare = new OtherSquare(side:5);
    processRectangle(rect);
    processRectangle(otherSquare);
}
```

```
> cd /Users/galuhdipabharata/Developer/OOP/
SOLID ; /usr/bin/env /Users/galuhdipabharata
.sdkman/candidates/java/17.0.9-amzn/bin/jav
a -XX:+ShowCodeDetailsInExceptionMessages -c
p /Users/galuhdipabharata/Library/Applicatio
n\ Support\Code\User\workspaceStorage\1569b9
166e28afe13ee1e403fad766e3/redhat.java/jdt_w
s/SOLID_d6c6c0ea/bin com.solidtutorial.Main
```

```
{shapes_sum: 253}  
shapes_sum,253  
20.0  
25.0
```

```
~/Developer/00P/SOLID ..... 19:12:41
> █
```


Interface Segregation Principle

Clients should not be forced to depend upon interfaces they do not use

- When a component has several different clients it is tempting to provide a large interface that satisfies the needs of all clients
- It is much better design to have the component support multiple interfaces, one appropriate for each client



Interface Segregation Principle

```
1 public interface IEmployee {  
2     public String getProjectDetails*int empId);  
3     public String getEmployeeDetails*int empId);  
4 }  
5  
6 public class ContractEmployee extends IEmployee {  
7     public String getProjectDetails(int empId) {  
8         // code for specific project detail  
9         System.out.println("Contract Employee Project Detail");  
10    }  
11    public String getEmployeeDetails*int empId) {  
12        // we don't need this methode  
13        throw new NotImplementedException();  
14    }  
15 }
```



```
package com.solidtutorial;

public class Cube implements Shape {
    private final int length;

    public Cube(int length) {
        this.length = length;
    }

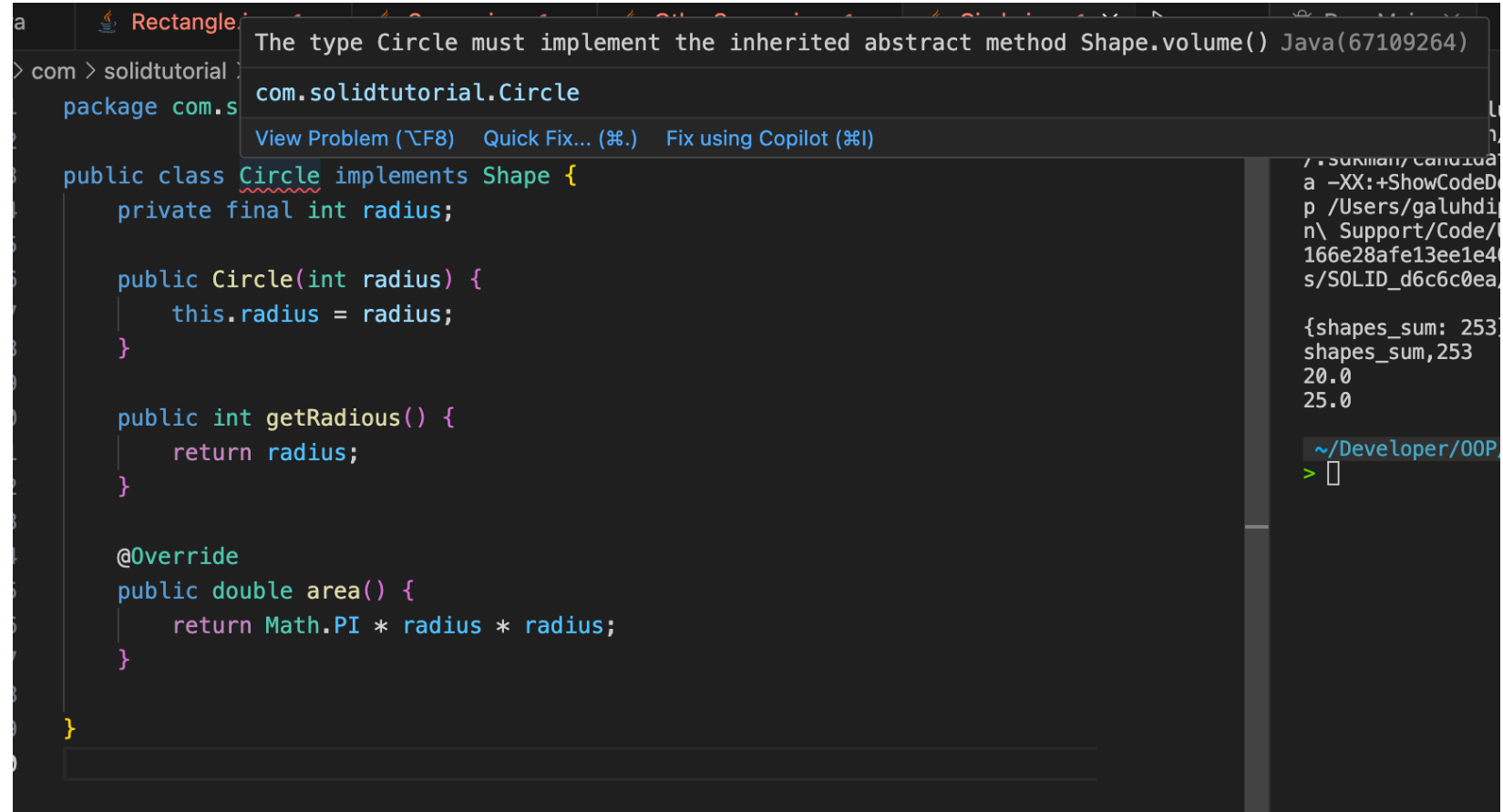
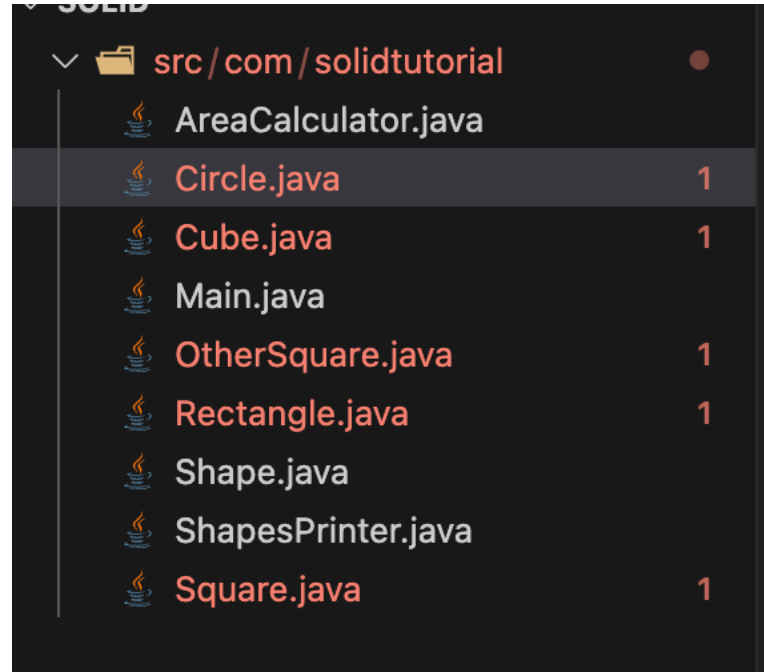
    public int getLength() {
        return length;
    }

    @Override
    public double area() {
        return 6 * length * length;
    }
}
```

```
package com.solidtutorial;

public interface Shape {
    double area();
    double volume();
}
```





```
src > com > solidtutorial > ☕ ThreeDimensionalShape.java > ...
```

```
1 package com.solidtutorial;
2
3 public interface ThreeDimensionalShape {
4     double volume();
5 }
6
```

```
package com.solidtutorial;

public class Cube implements Shape, ThreeDimensionalShape {
    private final int length;

    public Cube(int length) {
        this.length = length;
    }

    public int getLength() {
        return length;
    }

    @Override
    public double area() {
        return 6 * length * length;
    }

    @Override
    public double volume() {
        return length * length * length;
    }
}
```

Dependency Inversion Principle

High level components should not depend upon low level components. Instead, both should depend on abstractions

Abstractions should not depend upon details. Details should depend upon the abstractions

Dependency Inversion Principle

```
1 public class Notification {  
2     private Email email;  
3     public Notification() {  
4         email = new Email();  
5     }  
6     public void promoNotification() {  
7         email.send();  
8     }  
9 }
```



Dependency Inversion Principle

```
1 public class Notification {  
2     private Messenger mess;  
3     public Notification(Messenger mess) {  
4         this.mess = mess;  
5     }  
6     public void promoNotification() {  
7         mess.send();  
8     }  
9 }
```




```
package com.solidtutorial;

import java.util.List;

public class ShapesPrinter {

    private AreaCalculator areaCalculator = new AreaCalculator();



    public String json(List<Shape> shapes) {
        return "{shapes_sum: %s}".formatted(areaCalculator.sum(shapes));
    }

    public String csv(List<Shape> shapes) {
        return "shapes_sum,%s".formatted(areaCalculator.sum(shapes));
    }

}
```

src > com > solidtutorial >  IAreaCalculator.java > ...

```
1  package com.solidtutorial;
2
3  import java.util.List;
4
5  public interface IAreaCalculator {
6      int sum(List<Shape> shapes);
7  }
8
```

src > com > solidtutorial >  AreaCalculator.java >  AreaCalculator

```
1  package com.solidtutorial;
2  import java.util.List;
3
4  public class AreaCalculator implements IAreaCalculator {
5      public int sum(List<Shape> shapes) {
6          int sum = 0;
7          for (int i = 0; i < shapes.size(); i++) {
8              sum += shapes.get(i).area();
9          }
10         return sum;
11     }
12 }
```

```
package com.solidtutorial;

import java.util.List;

public class ShapesPrinter {

    private final IAreaCalculator areaCalculator;

    public ShapesPrinter(IAreaCalculator areaCalculator) {
        this.areaCalculator = areaCalculator;
    }

    public String json(List<Shape> shapes) {
        return "{shapes_sum: %s}".formatted(areaCalculator.sum(shapes));
    }

    public String csv(List<Shape> shapes) {
        return "shapes_sum,%s".formatted(areaCalculator.sum(shapes));
    }
}
```



Terima Kasih