Polymorphism

Abstract Classes, Abstract Methods, Override Methods

SoftUni Team

Technical Trainers

Software University

SoftUni Foundation

Software University

http://softuni.bg

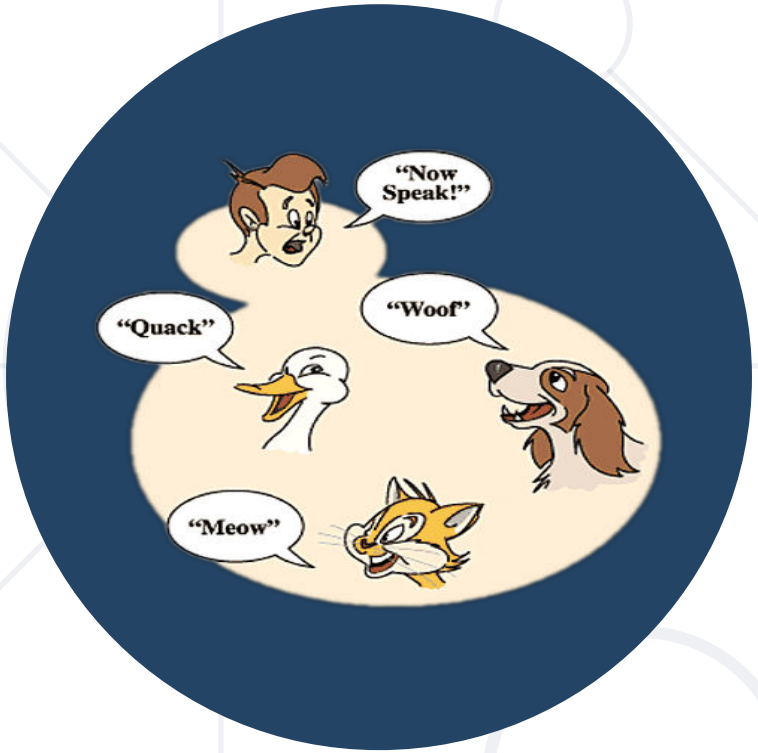
Table of Contents

Have a Question?

SoftUni Foundation

sli.do

#java-advanced



Polymorphism

What is Polimorphism?

- From the Greek

Polys *(many)*

Morphe *(shape/forms)*

Polymorphos

- This is something similar to a word having several different meanings depending on the context
- Polymorphism is often referred to as the third pillar of object-oriented programming, after encapsulation and inheritance

Polymorphism in OOP

SoftUni Foundation

- Ability of an **object** to take on **many forms**

```
public interface Animal {}
public abstract class Mammal {}
public class Person extends Mammal implements Animal {}
```

```
Person IS-A Person
```

```
Person IS-AN Animal
```

```
Person IS-A Mammal
```

```
Person IS-AN Object
```

Reference Type and Object Type

```
public class Person extends Mammal implements Animal {}
Animal person    = new Person();
Mammal personOne = new Person();
Person personTwo = new Person();
```

Reference Type

Object Type

- **Variables** are saved in **reference** type
- You can use only **reference methods**
- If you need **object method** you need to **cast it or override it**

Keyword - instanceof

SoftUni Foundation

- Check if **object** is an **instance** of a specific **class**

```
Mammal george = new Person();
Person peter = new Person();
```

Check object type of person

```
if (george instanceof Person) {
  ((Person) george).getSalary();
}
```

```
if (peter.getClass() == Person.class) {
  ((Person) peter).getSalary();
}
```

Cast to object type and use its methods

Types of Polymorphism

SoftUni Foundation

- **Runtime** polymorphism

```
public class Shape {}
public class Circle extends Shape {}
public static void main(String[] args) {
  Shape shape = new Circle();
}
```

Method overriding

- **Compile time** polymorphism

```
int sum(int a, int b, int c){}
double sum(Double a, Double b){}
```

Method overloading

Compile Time Polymorphism

- Also known as **Static Polymorphism**

```
static int myMethod(int a, int b) {}
static Double myMethod(Double a, Double b) {}
```

Method overloading

- Argument lists could differ in:
  - Number of parameters
  - Data type of parameters
  - Sequence of Data type of parameters

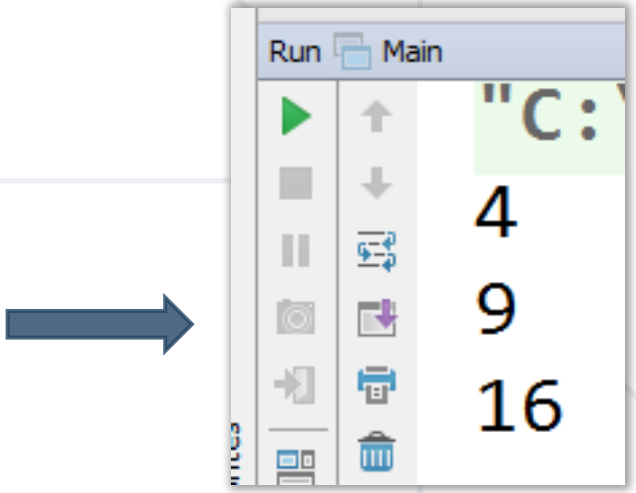
Problem: MathOperation

- Create a class **MathOperation**, which should have method **add()**
- That has to be invoked with **two**, **three** or **four integers**

MathOperation
+add(int a, int b): int +add(int a, int b, int c): int +add(int a, int b, int c, int d): int

```
MathOperation mathOperation = new MathOperation();

System.out.println(mathOperation.add( a: 2,  b: 2));
System.out.println(mathOperation.add( a: 3,  b: 3,  c: 3));
System.out.println(mathOperation.add( a: 4,  b: 4,  c: 4,  d: 4));
```



Check your solution here :https://judge.softuni.bg/Contests/1592/Polymorphism-Lab

Solution: Math Operation

```
public class MathOperation {
  public int add(int a, int b) {
    return a + b;
  }
  public int add(int a, int b, int c) {
    return a + b + c;
  }
  public int add(int a, int b, int c, int d) {
    return a + b + c + d;
  }
}
```

Check your solution here :https://judge.softuni.bg/Contests/1592/Polymorphism-Lab

Rules for Overloading Method

SoftUni Foundation

- **Overloading** can take place in the **same class** or in its **sub-class**
- **Constructors** in Java can be **overloaded**
- Overloaded methods must have a **different argument list**
- Overloaded method should always be the part of the same class (can also take place in sub class), with **same name** but **different parameters**
- They may have the **same** or **different return types**

Runtime Polymorphism

- Using of **override** method

```
public static void main(String[] args) {
  Rectangle rect = new Rectangle(3.0, 4.0);
  Rectangle square = new Square(4.0);

  System.out.println(rect.area());
  System.out.println(square.area());
}
```

Method overriding

Runtime Polymorphism

- Also known as **Dynamic Polymorphism**

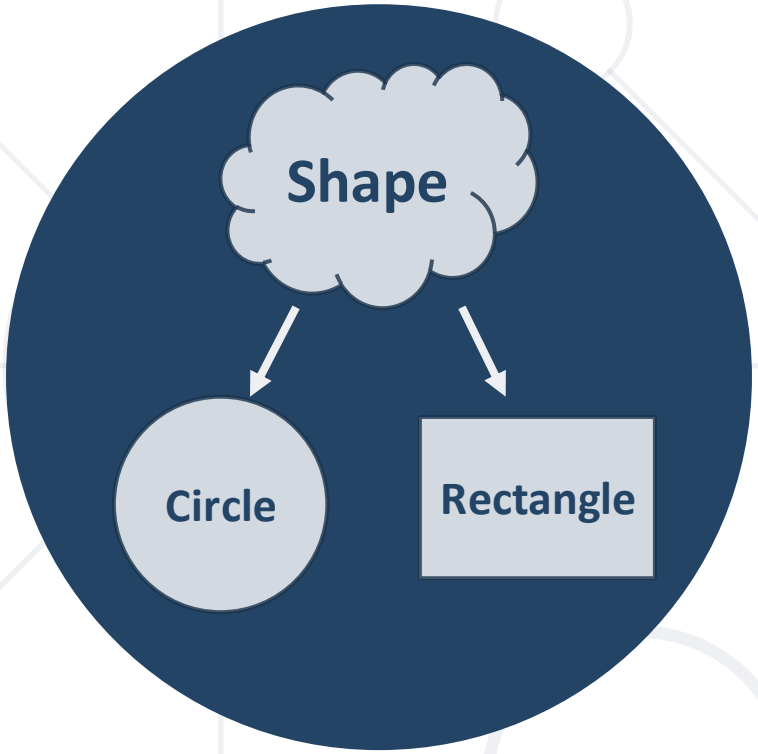
```
public class Rectangle {
  public Double area() {
    return this.a * this.b;
  }
}
```

```
public class Square extends Rectangle {
  @Override
  public Double area() {
    return this.a * this.a;
  }
}
```

Method overriding

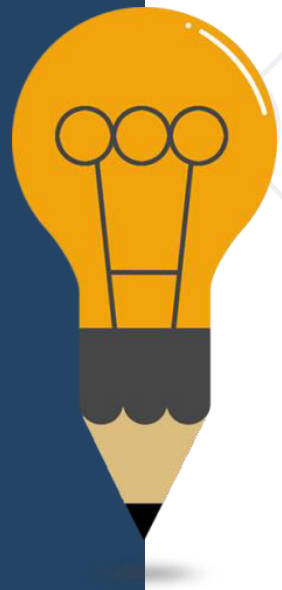
Rules for Overriding Method

- **Overriding** can take place in **sub-class**
- **Argument list** must be the **same** as that of the **parent method**
- The overriding method must have **same return type**
- **Access modifier** cannot be more **restrictive**
- **Private**, **static** and **final** methods can **NOT** be overriden
- The overriding method **must not** throw new or broader **checked exceptions**



Abstract Classes

Abstract Classes

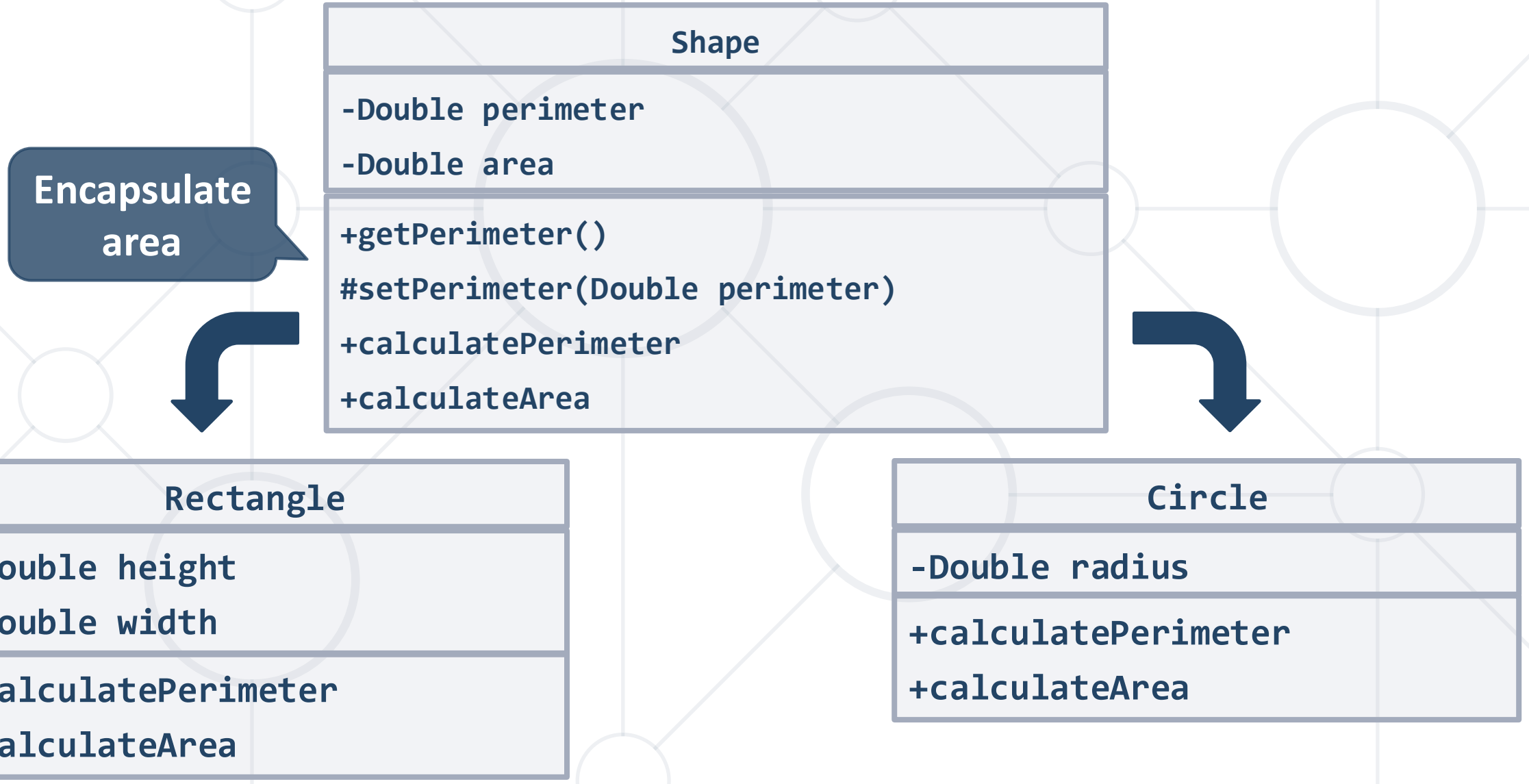


- Abstract class **can NOT be instantiated**

```
public abstract class Shape {}
public class Circle extends Shape {}
Shape shape = new Shape(); // Compile time error
Shape circle = new Circle(); // polymorphism
```

- An **abstract** class may or may not include **abstract methods**
- If it has at least one abstract method, it must be declared **abstract**
- To use an **abstract class**, you need to **inherit it**

Problem: Shapes



Solution: Shapes

SoftUni Foundation

```
public abstract class Shape {
  private Double perimeter;
  private Double area;
  protected void setPerimeter(Double perimeter) {
    this.perimeter = perimeter;
  }
  public Double getPerimeter() { return this.perimeter; }
  protected void setArea(Double area) {this.area = area; }
  public Double getArea() { return this.area; }
  protected abstract void calculatePerimeter();
  protected abstract void calculateArea();
}
```

Solution: Shapes

SoftUni Foundation

```
public class Rectangle extends Shape {
  //TODO: Add fields
  public Rectangle(Double height, Double width) {
    this.setHeight(height); this.setWidth(width);
    this.calculatePerimeter(); this.calculateArea(); }
  //TODO: Add getters and setters
  @Override
  protected void calculatePerimeter() {
    setPerimeter(this.height * 2 + this.width * 2); }
  @Override
  protected void calculateArea() {
    setArea(this.height * this.width); } }
```

Solution: Shapes

SoftUni Foundation

```
public class Circle extends Shape {
  private Double radius;
  public Circle (Double radius) {
    this.setRadius(radius);
    this.calculatePerimeter();
    this.calculateArea();
  }
  public final Double getRadius() {
    return radius;
  }

  //TODO: Finish encapsulation
  //TODO: Override calculate Area and Perimeter
}
```

Summary

- Polymorphism - **Definition** and **Types**
- Override Methods
- Overload Methods
- Abstraction
  - **Classes**
  - **Methods**

Questions?

SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

SoftUni Diamond Partners

SoftUni Foundation

XSsoftware

SBTech
we know sports

telenor

SoftwareGroup
doing it right

NETPEAK

SmartIT

Postbank
Решения за твоето утре

SUPER HOSTING .BG

INDEAVR
Serving the high achievers

INFRAGISTICS

STEMO
Computer Systems & Software

SoftUni Organizational Partners

SoftUni Foundation

Trainings @ Software University (SoftUni)

SoftUni Foundation

- Software University – High-Quality Education and Employment Opportunities
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license