



# Polymorphism

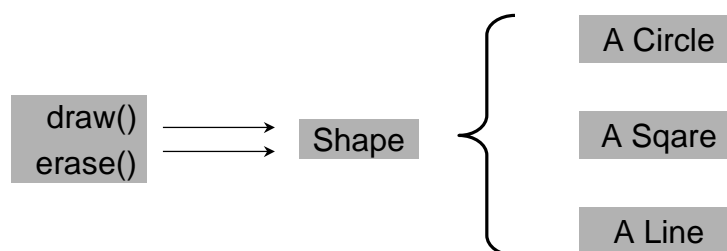
- Polymorphism
- Abstract Classes
- Interfaces

OOAD 1998/99

Claudia Niederée, Joachim W. Schmidt  
Software Systems Institute

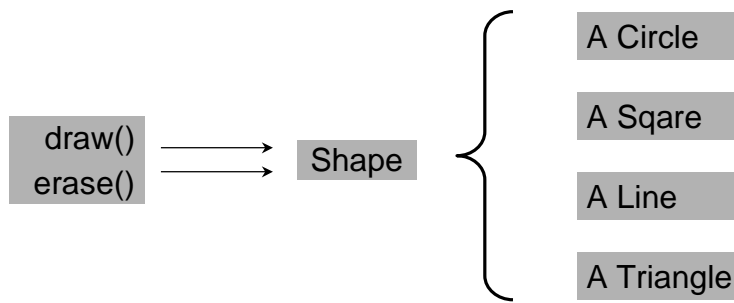
c.niederee@tu-harburg.de <http://www.sts.tu-harburg.de>

# Polymorphism



“Substitutability“

= You may use a subclass object wherever a base class object is expected



“Extensibility“

## A problem?

```
Shape s = new Circle();
```

You might think

`s.draw()`

calls this `draw()`

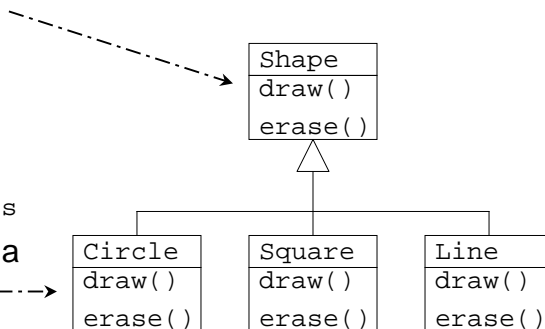
since `s` is a `Shape`

We want it to call

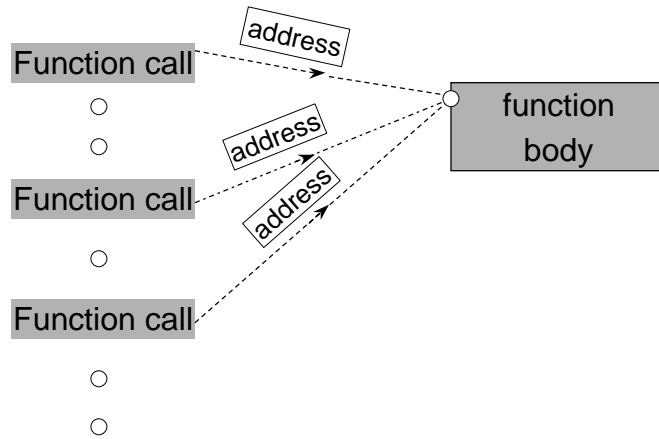
this `draw()` since `s`

actually points to a

`Circle`



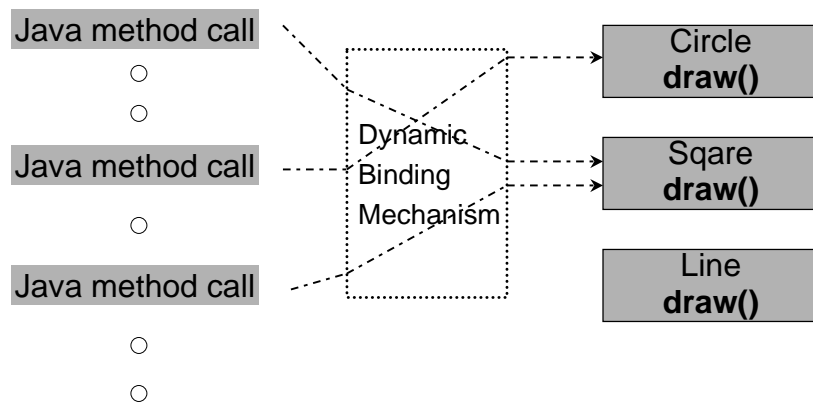
## Early Function Call Binding



OOAD98/99-STS-Polymorphism

5

## Dynamic Binding in Java



OOAD98/99-STS-Polymorphism

6

```

//: Shapes.java
import java.util.*;
class Shape {
    void draw() {}
    void erase() {}
}
class Circle extends Shape {
    void draw() {
        System.out.println("Circle.draw()"); }
    void erase() {
        System.out.println("Circle.erase()"); }
}
class Square extends Shape {
    void draw() {
        System.out.println("Square.draw()"); }
    void erase() {
        System.out.println("Square.erase()"); }
}
class Triangle extends Shape {
    void draw() {
        System.out.println("Triangle.draw()"); }
    void erase() {
        System.out.println("Triangle.erase()"); }
}

```

OOAD98/99-ST5-Polymorphism

7

```

public class Shapes {
    public static Shape randShape() {
        switch((int)(Math.random() * 3)) {
            default: // To quiet the compiler
            case 0: return new Circle();
            case 1: return new Square();
            case 2: return new Triangle();
        }
    }
    public static void main(String args[]) {
        Shape s[] = new Shape[9];
        // Fill up the array with shapes:
        for(int i = 0; i < s.length; i++)
            s[i] = randShape();
        // Make polymorphic method calls:
        for(int i = 0; i < s.length; i++)
            s[i].draw();
    }
}

```

OOAD98/99-ST5-Polymorphism

8

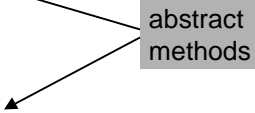
## Output:

```
Circle.draw()  
Triangle.draw()  
Circle.draw()  
Circle.draw()  
Circle.draw()  
Square.draw()  
Triangle.draw()  
Square.draw()  
Square.draw()
```

## Abstract Classes

- May Contain abstract methods
- Some methods and data may be defined

```
abstract class ColouredShape {  
    private Colour c; // storage allocated for each  
    public abstract void draw();  
    public Colour getColour() {  
        return c;  
    }  
    public abstract void erase();  
}
```



- No instances allowed
- Subclasses must implement all abstract methods or they are also abstract

## Interface

- Contains no implementation
- Method signatures + static final data members  
Method signature =  
return type + method name + argument list
- Establishes a protocol
- public or friendly

```
interface Shape {  
    Point zero = Point(0,0);  
    // Cannot have method implementations:  
    void erase();  
    void draw();  
}
```

automatically static & final

automatically public

OOAD98/99-ST5-Polymorphism

11

## Interfaces and Classes

- A class may implement one or more interfaces

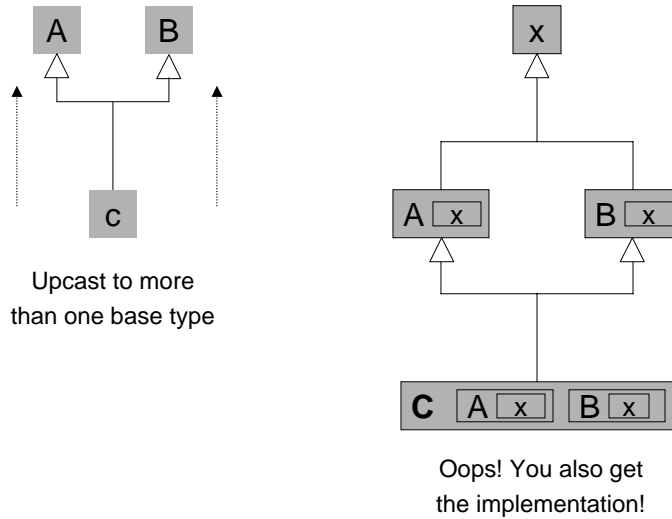
```
class Circle implements Shape {  
    private int radius;  
    public void erase() {  
        ...  
    }  
    public String draw() {  
        ...  
    }  
    public int getRadius(){  
        ...  
    }  
}
```

- An interface may extend one or more other interface

OOAD98/99-ST5-Polymorphism

12

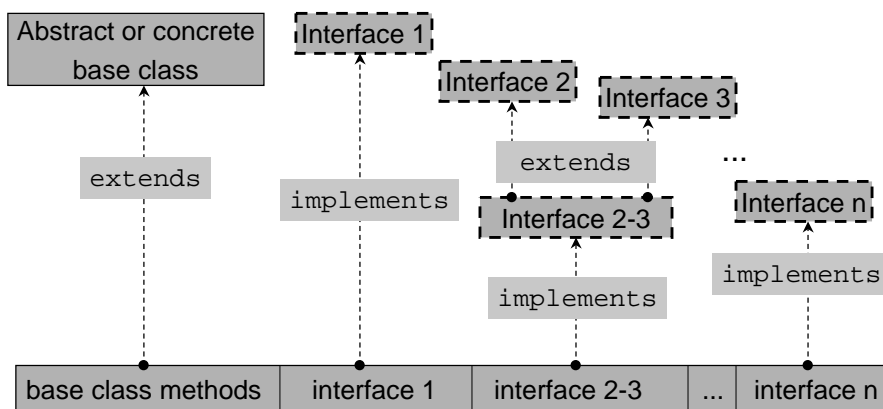
## C++ Multiple Inheritance



OOAD98/99-ST5-Polymorphism

13

## “Multiple Inheritance” in Java



New class has combined interfaces of all types, but only one physical implementation: that of the base class.

OOAD98/99-ST5-Polymorphism

14

```

interface CanFight {
    void fight();
}
interface CanSwim {
    void swim();
}
interface CanFly {
    void fly();
}
interface CanFlySwimDive extends CanFly, CanSwim{
    void dive();
}

class ActionCharacter {
    public void fight() {...}
}

```

OOAD98/99-ST5-Polymorphism

15

```

class Hero extends ActionCharacter
    implements CanFight, CanSwim, CanFly {
    public void swim() {...}
    public void fly() {...}

public class Adventure {
    static void t(CanFight x) { x.fight(); }
    static void u(CanSwim x) { x.swim(); }
    static void v(CanFly x) { x.fly(); }
    static void w(ActionCharacter x) { x.fight(); }
    public static void main(String args[]) {
        Hero i = new Hero();
        t(i); // Treat it as a CanFight
        u(i); // Treat it as a CanSwim
        v(i); // Treat it as a CanFly
        w(i); // Treat it as an ActionCharacter
    }
}

```

OOAD98/99-ST5-Polymorphism

16