JOHANNES KEPLER
UNIVERSITY LINZ

Übung Softwareentwicklung 2
für Wirtschaftsinformatik

Übung 03 B
Polymorphism & Dynamic Binding

Ismail Khalil
ismail.khalil@jku.at

https://moodle.jku.at/jku/course/view.php?id=8729

Polymorphism

- Polymorphic = “of many forms”
- Polymorphism is an OOP feature that enables an object to determine which method implementation to invoke upon receiving a method call.
- A **polymorphic method** is one that has the same name for different classes of the same family but has different implementations for the various classes

Overloading methods & Constructors

- Overloading refers to the ability to allow different methods or constructors of a class to share the same name
- If two methods or constructors in the **same class** have **different signatures**, then they may share the same name

Method	**Signature**
void move(int x, int y)	move(int, int)
void move(double x, double y)	move(double, double)
boolean move(int x, int y)	move(int, int)

- Methods of **different classes** can have the **same signature**

An example - Point class

```
class Point {
  private double x, y;
  public Point() { x = 0.0; y = 0.0; }
  public Point(double x, double y) {
    this.x = x; this.y = y;
  }
  public double distance(Point other) {
    double dx = this.x - other.x;
    double dy = this.y - other.y;
    return Math.sqrt(dx * dx + dy * dy);
  }
  public double distance(double x, double y) {
    double dx = this.x - x;
    double dy = this.y - y;
    return Math.sqrt(dx * dx + dy * dy);
  }
}
```

a constructor with a different signature

a method with a different signature

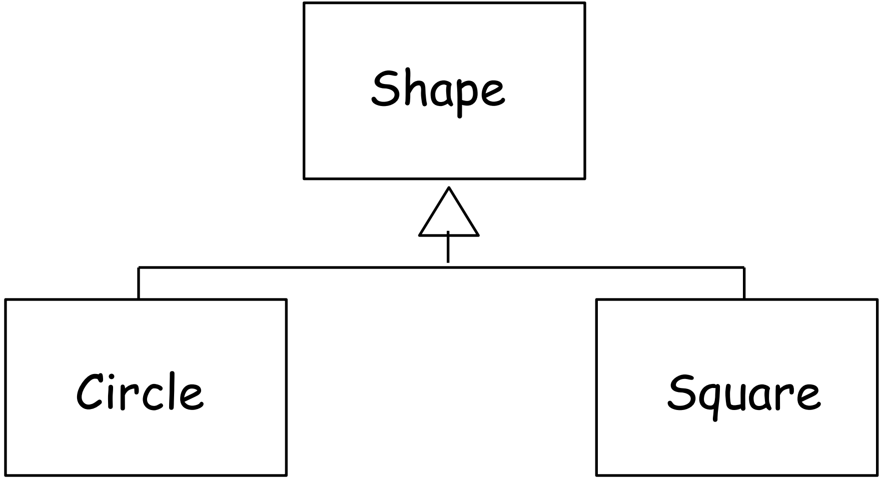
Example … cont.

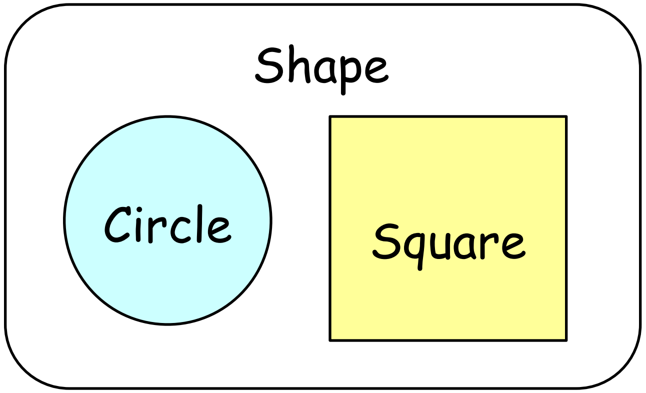
- when an **overloaded method** is called, the number and the types of the arguments are used to determine the method that will be invoked

```
… …
Point p1 = new Point();
Point p2 = new Point(20.0, 30.0);
p2.distance(p1);
p2.distance(50.0, 60.0)
… …
```

Subtypes

- A subclass is a specialization of its superclass
- Every instance of the subclass is an instance of the superclass





- The type defined by the subclass is a subtype of the type defined by its superclass

Rule of Subtype

- **A value of a subtype can appear wherever a value of its supertype can appear**

```
class Shape { … … }
class Circle extends Shape { … … }
class Square extends Shape { … … }

Shape shape1, shape2;
shape1 = new Circle();
shape2 = new Square();
```

Circle and Square are subclasses of Shape

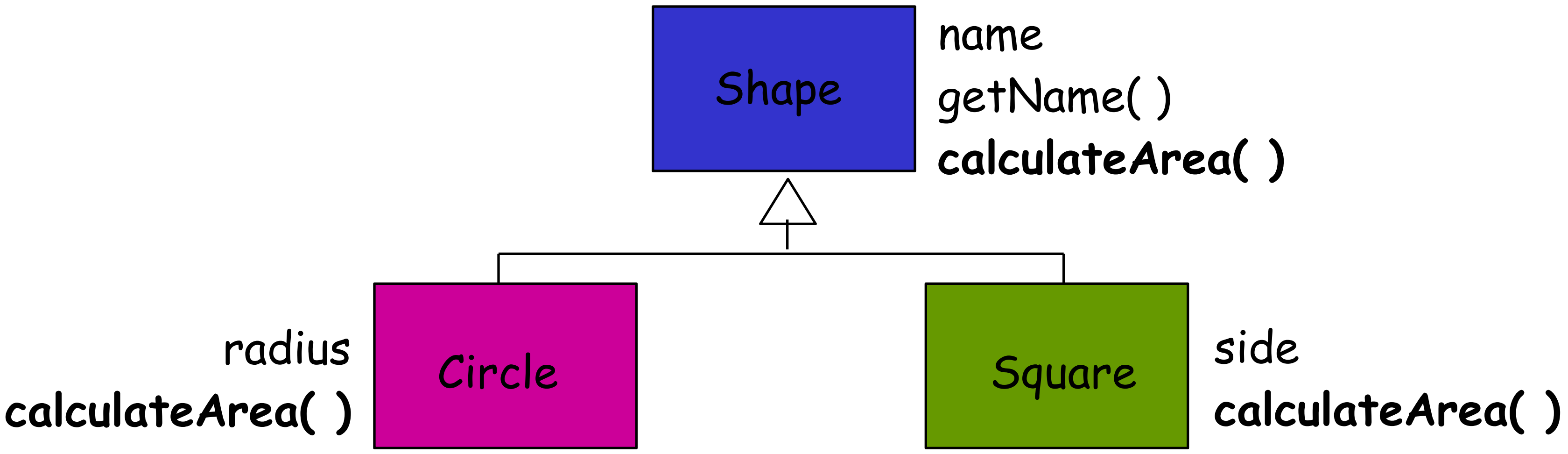
- If class E extends class B, any instance of E can act as an instance of B

Static vs dynamic binding

- **Binding** refers to the **association of a method invocation** and the code to be executed on behalf of the invocation.
- In **static binding** (early binding), all the associations are determined at **compilation time.**
  - conventional function calls are statically bound
- In **dynamic binding** (late binding), the code to be executed in response to a method invocation (i.e., a message) will not be determined until **runtime**.
  - method invocations to reference variable **`shapeArray[i]`** (in the following example) are dynamically bound

Polymorphism

- The ability of different objects to perform the appropriate method in response to the same message is known as polymorphism
- The selection of the appropriate method depends on the class used to create the object



Example

```
class Shape {
  private String name;
  public Shape(String aName) { name=aName; }
  public String getName( ) { return name; }
  public float calculateArea( ) { return 0.0f; }
} // End Shape class
```

Example ... cont.,

inheritance

```
class Circle extends Shape {
  private float radius;
  public Circle(String aName) { super(aName); radius = 1.0f; }
  public Circle(String aName, float radius) {
    super(aName); this.radius = radius;
  } public float calculateArea() { return
(float)3.14f*radius*radius; }
} // End Circle class
```

overloading

overriding

```
class Square extends Shape {
  private float side;
  public Square(String aName) { super(aName); side = 1.0f; }
  public Square(String aName, float side) {
    super(aName); this.side = side;
  }
  public float calculateArea() { return (float) side*side; }
} // End Square class
```