Polymorphism

The Problem

```
public class Circle {
    private double radius;
    ...
    public double area(){
            return Math.PI * Math.pow(radius, 2);
    }
}
```

The Problem

```
public class Rectangle {

    double width;
    double height;
    ...
    public double area(){
          return height * width;
    }
}
```

The Problem

```
public class Drawing {

    ArrayList<Circle> circles = new ArrayList<Circle>();
    ArrayList<Rectangle> rectangles = new ArrayList<Rectangle>();

    public double calculateTotalArea(){
        double totalArea = 0;

        for (Circle circle : circles){
            totalArea += circle.area();    // totalArea = totalArea + circle.area();
        }

        for (Rectangle rect : rectangles){
            totalArea += rect.area();     // totalArea = totalArea + rect.area();
        }
        return totalArea;
    }
}
```

The Problem

- We are asked to introduce a new shape class to the application.
- How Drawing class will be affected?

New Shape: Square

```
public class Square {
    private double side;

    ...

    public double area(){
        return Math.pow(side, 2);
    }
}
```

Drawing

```
public class Drawing {

    ArrayList<Circle> circles = new ArrayList<Circle>();
    ArrayList<Rectangle> rectangles = new ArrayList<Rectangle>();
    ArrayList<Square> squares = new ArrayList<Square>();
    public double calculateTotalArea(){
        double totalArea = 0;
        for (Circle circle : circles){
            totalArea += circle.area();    // totalArea = totalArea + circle.area();
        }
        for (Rectangle rect : rectangles){
            totalArea += rect.area();     // totalArea = totalArea + circle.area();
        }
        for (Square sq : squares){
            totalArea += sq.area();
        }
        return totalArea;
    }
}
```

Design Principle

- Classes should be open for extension, but closed for modification
- Allow classes to be easily extended to add new behaviour without modifying existing code
- How can we accomplish this?

Drawing (Version 2)

```
public class DrawingV2 {

    ArrayList shapes = new ArrayList();

    public double calculateTotalArea(){
            double totalArea = 0;

            for (Object shape : shapes){
                    if (shape instanceof Circle){
                            Circle circle = (Circle) shape;
                            totalArea += circle.area();
                    }else if (shape instanceof Rectangle){
                            Rectangle rect= (Rectangle) shape;
                            totalArea += rect.area();
                    }
            }
            return totalArea;
    }
}
```

Problems with Casting

```
Rectangle r = new Rectangle(5, 10);
Circle c = new Circle(5);

Object s = c;


((Rectangle)s).changeWidth(4);
```

- Does this work?

Problems with Casting

- The following code compiles but an exception is thrown at runtime

```
Rectangle r = new Rectangle(5, 10);
Circle c = new Circle(5);
Object s = c;
((Rectangle)s).changeWidth(4);
```

- Casting must be done carefully and correctly
- If unsure of what type object will be then use the **`instanceof`** operator

instanceof

```
Rectangle r = new Rectangle(5, 10);
Circle c = new Circle(5);
Object s = c;
if(s instanceof Rectangle)
     ((Rectngle)s).changeWidth(4);
```

- syntax: **expression** **instanceof** **ClassName**

Casting

- It is always possible to convert a subclass to a superclass. For this reason, explicit casting can be omitted. For example,
  - `Circle c1 = new Circle(5);`
  - `Object s = c1;`

is equivalent to

  - `Object s = (Object)c1;`

- Explicit casting must be used when casting an object from a superclass to a subclass. This type of casting may not always succeed.
  - `Circle c2 = (Circle) s;`

Modification to handle Square

```
public class DrawingV2 {
	ArrayList shapes = new ArrayList();
	public double calculateTotalArea(){
		double totalArea = 0;
		for (Object shape : shapes){
			if (shape instanceof Circle){
				Circle circle = (Circle) shape;
				totalArea += circle.area();
			}else if (shape instanceof Rectangle){
				Rectangle rect= (Rectangle) shape;
				totalArea += rect.area();
			}else if (shape instanceof Square){
				Square sq= (Square) shape;
				totalArea += sq.area();
			}
		}
		return totalArea;
	}
}
```

DrawingV2

- Still requires modification to handle new Shapes

- It is possible to add other Objects to the shape list.
  - drawing.add(new String("abc"));

- The common super class for Rectangle, Circle and Square is java.lang.Object

Shape Class

```
public class Shape {

    public double area(){
        return 0;       //default implementation
    }

    public double perimeter(){
        return 0;       //default implementation
    }

}
```

Circle extends Shape

```
public class Circle extends Shape{
    private double radius;
    ...
    public double area(){
        return Math.PI * Math.pow(radius, 2);
    }
}
```

Rectangle extends Shape

```
public class Rectangle extends Shape{

    double width;
    double height;
    ...
    public double area(){
        return height * width;
    }
}
```

Drawing (Version 3)

```
public class DrawingV3 {

    ArrayList<Shape> shapes = new ArrayList<Shape>();

    public void addShape(Shape shape){
            shapes.add(shape);
    }

    public double calculateTotalArea(){
            double totalArea = 0;
            for (Shape shape : shapes){
                    totalArea += shape.area();
            }
            return totalArea;
    }
}
```

DrawingV3

- Does not need modification to handle new Shapes
- Only Shape typed objects can be added. Following is not possible now

```
drawing.add(new String("  ")); //compile-time error
```

- What happens if a developer forgets to override area method in a new Shape class?

```
public class Square extends Shape{
    private double side;

    public Square(double side){
        this.side = side;
    }

}
```

Abstract Shape

```
public abstract class Shape {

   public abstract double area();

   public abstract double perimeter();

}
```

Polymorphism

- The term *polymorphism* literally means "having many forms"
- A *polymorphic reference* is a variable that can refer to different types of objects at different points in time
- The method invoked through a polymorphic reference can change from one invocation to the next

Polymorphism

- Suppose we create the following reference variable:

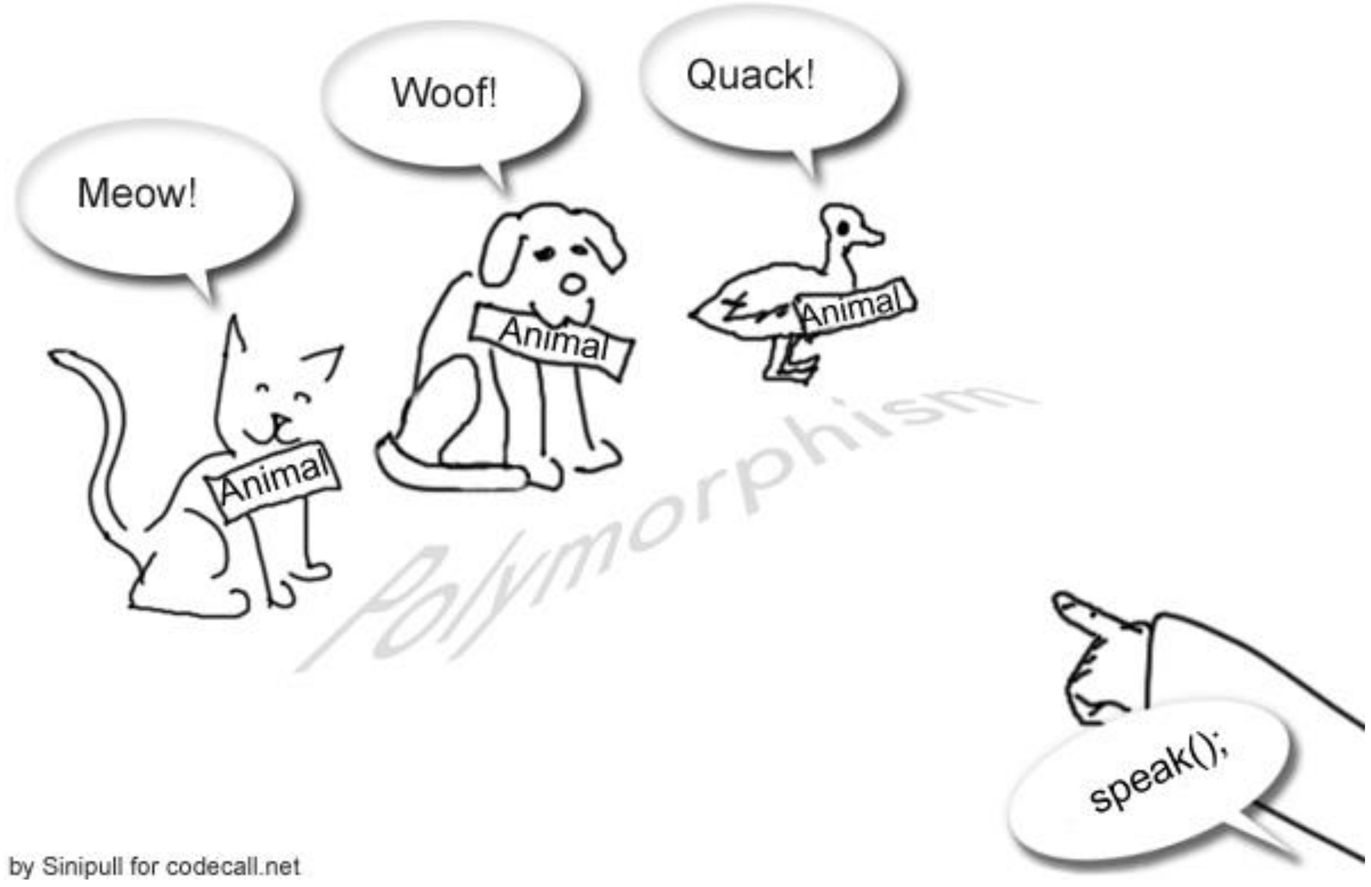
```
        Shape shape;
```

- Java allows this reference to point to an `Shape` object, or to any object of <u>any compatible type</u>

- This compatibility can be established using inheritance or using interfaces

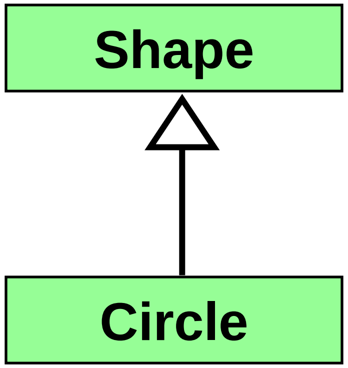
- Careful use of polymorphic references can lead to elegant, robust software designs

Polymorhism



References and Inheritance

- An object reference can refer to an object of its class, or to an object of any class related to it by inheritance
- For example, if the `Shape` class is used to derive a class called `Circle`, then a `Shape` reference could be used to point to a `Circle` object



```
Shape shape;
shape = new Circle(5);
```

References and Inheritance

- Assigning a child object to a parent reference is called upcasting, and can be performed by simple assignment

```
Shape shape;

shape = new Circle(5);
```

- Assigning a parent object to a child reference can be done also, but it is called downcasting and must be done manually

```
Circle c2 = (Circle) shape;
```

Polymorphism via Inheritance

- It is the type of the object being referenced, not the reference type, that determines which method is invoked
- Suppose the `Shape` class has a method called `area`, and the `Circle and Rectangle` classes override it
- Now consider the following invocation:

```
shape.area();
```

- If `shape` refers to a `Circle` object, it invokes the `Circle` version of `area`; if it refers to a `Rectangle` object, it invokes the `Rectangle` version

References

- http://math.hws.edu/javanotes/
- http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-092-introduction-to-programming-in-java-january-iap-2010/lecture-notes/
- https://www.tiobe.com/tiobe-index/
- Head First Java 2nd Edition
- Java a Beginner's guide 7th Edition