



# Coding-bootcamps.com

## Introduction to Java Programming Language

By Jim Sullivan  
from [Coding Bootcamps](https://coding-bootcamps.com)



# Increasing Convenience by Using Polymorphism

Session 9



# Brief Recap

- Re-hash what we've covered in our last class

# Objective - Discuss The Following

- Purpose of Polymorphic Behavior
- The Concept of a Signature
- Abstract Classes and Methods
- final Methods and Classes
- Purpose of Interfaces
- Using and Creating Interfaces
- Common Interfaces of the Java API

# Materials

- These Powerpoint Slides

# OOP - Polymorphism

- Polymorphism is the ability of an object to take on many forms.
  - Polymorphism occurs when a parent class reference is used to refer to a child class object

# OOP - Polymorphism

- How to tell if something is polymorphic
  - It can pass the “IS-A” test
  - In JAVA all objects are polymorphic since any java object will pass the IS-A test for their own type for the class Object
- For example
  - A deer IS-A Animal
  - A deer IS-A Vegetarian
  - A deer IS-A Object

# OOP - Polymorphism

- Example 1

```
public class Employee {
    private String name;
    private String address;
    private int number;

    public Employee(String name, String address, int number) {
        System.out.println("Constructing an Employee");
        this.name = name;
        this.address = address;
        this.number = number;
    }

    public void mailCheck() {
        System.out.println("Mailing a check to " + this.name + " " + this.address);
    }

    public String toString() {
        return name + " " + address + " " + number;
    }

    public String getName() {
        return name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String newAddress) {
        address = newAddress;
    }

    public int getNumber() {
        return number;
    }
}
```



# OOP - Polymorphism

- Example 2

```
public class Salary extends Employee {
    private double salary; // Annual salary

    public Salary(String name, String address, int number, double salary) {
        super(name, address, number);
        setSalary(salary);
    }

    public void mailCheck() {
        System.out.println("Within mailCheck of Salary class ");
        System.out.println("Mailing check to " + getName()
            + " with salary " + salary);
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double newSalary) {
        if(newSalary >= 0.0) {
            salary = newSalary;
        }
    }

    public double computePay() {
        System.out.println("Computing salary pay for " + getName());
        return salary/52;
    }
}
```

# OOP - Abstract Classes

- Abstract classes are classes that contain one or more abstract methods.
- An abstract method is a method that is declared, but contains no implementation.
- Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods.

# OOP - Abstract Classes

- An abstract class is a class that is declared ***abstract***—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.
- An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

```
abstract void moveTo(double deltaX, double deltaY);
```

# OOP - Abstract Classes

- If a class includes abstract methods, then the class itself must be declared abstract, as in:

```
public abstract class GraphicObject {  
    // declare fields  
    // declare nonabstract methods  
    abstract void draw();  
}
```

# OOP - Abstract Classes

- Example

```
public abstract Animal
{
    public void eat(Food food)
    {
        // do something with food....
    }

    public void sleep(int hours)
    {
        try
        {
            // 1000 milliseconds * 60 seconds * 60 minutes *
            hours
            Thread.sleep ( 1000 * 60 * 60 * hours);
        }
        catch (InterruptedException ie) { /* ignore */ }
    }

    public abstract void makeNoise();
}
```

# OOP - Abstract Classes (Continued)

```
public Dog extends Animal
{
    public void makeNoise() { System.out.println ("Bark!
Bark!"); }
}

public Cow extends Animal
{
    public void makeNoise() { System.out.println ("Moo!
Moo!"); }
}
```

# OOP - Final Classes & Methods

- You can declare some or all of a class's methods final. You use the final keyword in a method declaration to indicate that the method cannot be overridden by subclasses.
- The Object class does this—a number of its methods are final.
- You might wish to make a method final if it has an implementation that should not be changed and it is critical to the consistent state of the object. For example, you might want to make the `getFirstPlayer` method in this `ChessAlgorithm` class final:

# OOP - Final Classes & Methods

```
class ChessAlgorithm {  
    enum ChessPlayer { WHITE, BLACK }  
    ...  
    final ChessPlayer getFirstPlayer() {  
        return ChessPlayer.WHITE;  
    }  
    ...  
}
```



# OOP - Interfaces

- As you've already learned, objects define their interaction with the outside world through the methods that they expose.
- Methods form the object's interface with the outside world
- In its most common form, an interface is a group of related methods with empty bodies.

# OOP - Interfaces

```
interface Bicycle {  
  
    // wheel revolutions per minute  
    void changeCadence(int newValue);  
  
    void changeGear(int newValue);  
  
    void speedUp(int increment);  
  
    void applyBrakes(int decrement);  
}
```

# Assignment

Suppose a class **Shape** has been defined like this:

Define a **Rectangle** subclass from the **Shape** class that contains the methods/data members below and any other **necessary** methods. Assume the upper left hand corner of the rectangle is (x,y) which is defined in Shape.

- Additional data members to contain the height and width of the rectangle
- A perimeter method to return the perimeter of the rectangle
- Override the toString() method so that it returns something like:

**"Rectangle: upper left corner: (x, y)                  width: w  
                                 height: h"** where x, y, w, and h are replaced by the values in the data members of Shape and Rectangle.

Suppose we have a second class **Circle** that also inherits from **Shape**. Also suppose we have a **Shape** array, **arrayOfShapes**, and it has already been filled with a collection of circles and rectangles. Write code to display only the elements in **arrayOfShapes** that are of type **Rectangle**.

```
public abstract class Shape
{
    private int x;    //the x coordinate of the shape
    private int y;    //the y coordinate of the shape
    public String toString()
    {
        return "(" + x + ", " + y + ")";
    }
    public abstract double area ();
    public abstract String getName(); // returns the shape's name
}
```

# Summary

# Live private coaching sessions for Java

- [Private tutoring sessions for software design and engineering- Weekly and monthly plans](#)
- [Java programming language- Private tutoring sessions](#)



Coding-bootcamps.com

Thank You



**Coding**  
Bootcamps