COMSC-051

Java Programming Part 1

Part-Time Instructor: Joenil Mistal

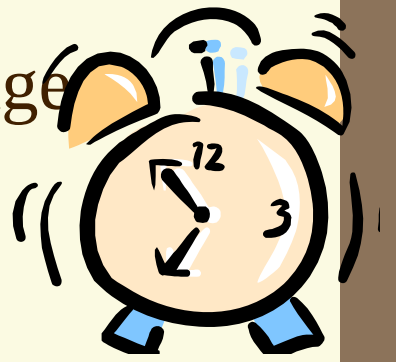
Chapter 7

Delving Further into Object-Oriented Concept

After completing this chapter, you will have the tools you will have the tools you need to create organized and advanced Java applications.

Chapter 7 Topics:
What you will learn in this chapter:

- What annotations are
- What overload methods are
- How to us the this keyword
- What information hiding or encapsulation is
- What class inheritance and polymorphism mean
- What interfaces are and what they to
- How to organize your classes into packages
- How garbage is collected in Java



Annotations page 222

- Annotations provide metadata about source code that's not part of the program itself.
- Annotations can provide information for the compiler and can be processed by some tools to generate code, documentation, or other files

@Deprecated – is used in a program, the compiler will give a warning so the programmer knows to use an alternative

@SuppressWarnings – instructs the compiler to suppress specific warnings that it would normally generate

Annotations page 222

@SuppressWarnings – instructs the compiler to suppress specific warnings that it would normally generate

MainApp.java | RoomsPainted.java | Cost.java

```
1  package residentialPaintJobs;
2  import java.util.Scanner;
3  @SuppressWarnings("resource")
4
5  public class RoomsPainted {
6      // declare the public variables that will be used in the subclass- Cost
7      public double numRooms, pricePerGallon;
8
9      RoomsPainted(){
10         // perform error handling if invalid data is entered
11         try {
12             // declare the scanner object that will be used to input the number of rooms to be painted
13             Scanner input1= new Scanner(System.in);
14             System.out.print("Enter the number of rooms to be painted: ");
15             numRooms = input1.nextDouble();  // assign the input value to the variable
16
17             // declare the scanner object that will be used to input the price of paint per gallon
18             Scanner input2= new Scanner(System.in);
19             System.out.print("Enter the price of the paint per gallon: $");
20             pricePerGallon = input2.nextDouble();  // assign the input value to the variable
21
22         } catch (Exception e) {
23             System.out.println("Invalid Data Entered");
24         }
25     }
26 }
```

Overloading Methods page 222

- **Overloading** simply refers to using the same name for more than one method in the same class
- Java can determine which method you're calling as long as the number or type of parameter is different in each method.

The this Keyword page 224

- The **this** keyword is placed inside an instance method or constructor in order to refer to the object whose method is being called.
- The **this** keyword is also used when you have overloaded the constructor method and want to explicitly call one constructor from within another constructor.

Information Hiding (Encapsulation) page 229

- Also known as **encapsulation**.
- Is an object-oriented practice that hides the internal representation of objects.
- The main idea is to make member variables private, so they are not directly accessible from outside of the class

Access Modifiers page 230

- The first step to restricting access to variables or methods is adjust the access modifiers.
- There are four access modifier that can be applied to methods and variables in Java

Public: **Can be accessed by any class**

Protected: **Can be accessed by subclasses or classes in the same package**

No modifier: **Can be accessed by classes in same package**

Private: **Can be accessed only from within the same class**

Getters page 231

- **Getters** are used to access variables values for viewing only
- You should a getter method for every private variable.
- Depending on the access level you want to give to the variable, you can set the access modifier of its getter method.

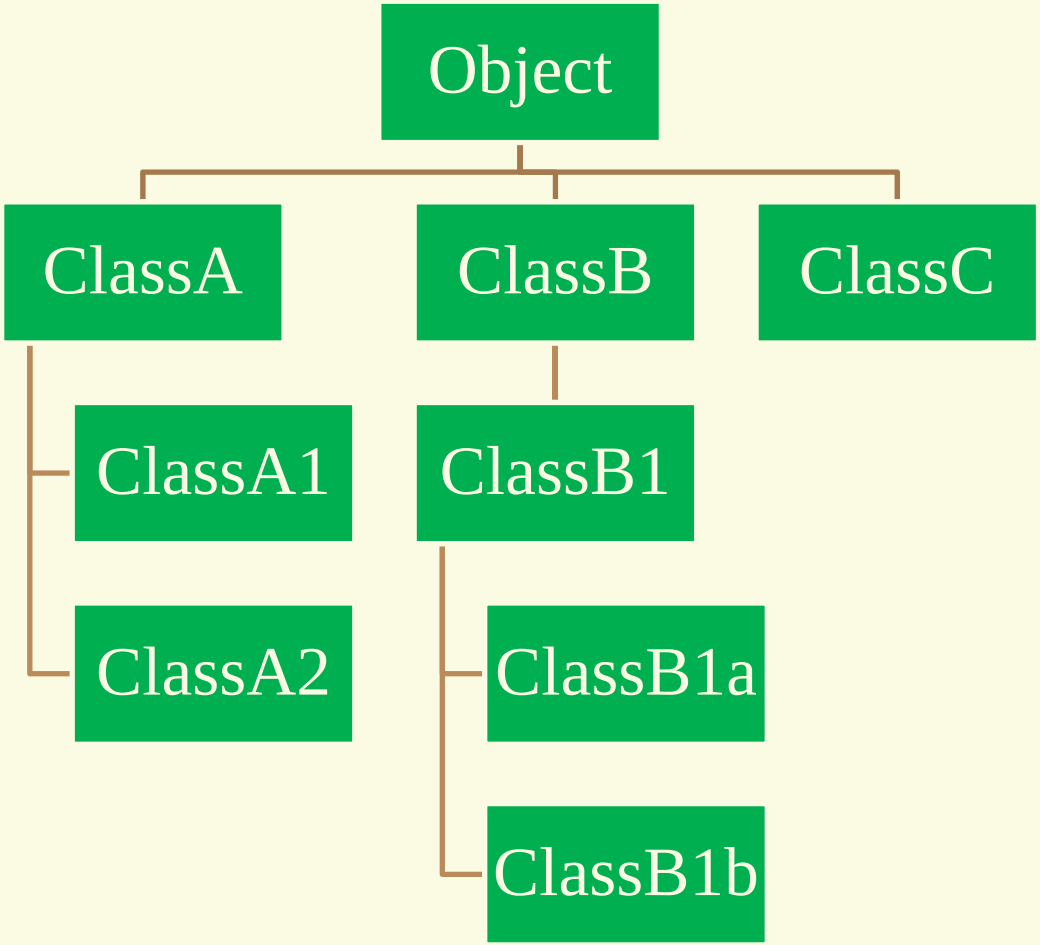
Setters page 232

- **Setters** are the methods you use to modify the values of variables.
- Just like with getters, you should create the setters for every variable.
- The setters here are very simple; they don’t include any of the validity check or access restrictions

Class Inheritance page 240

- Java classes are organized in a hierarchical inheritance structure.
- **Subclasses** are derived from **Superclasses**.
- The class Object is the highest-level structure.
- All of the other classes are either subclasses of Object or subclass of subclasses of Object.
- Inheritance represents “is” a relationship

Class Inheritance page 240



Class Inheritance page 240

Superclass **Subclasses**

MainApp.java | Shape.java | Rectangle.java | Triangle.java | Square.java

```
1  // the Rectangle subclass will inherit the methods and variables from the Shape parent class
2  // add the keyword "extends Shape" after the name of class
3  public class Rectangle extends Shape{
4
5      // method the get the dimensions and display the area size
6      void getDimensions(){
7
8          System.out.println("Rectangle Shape Dimensions");
9
10         // use "this" keyword to call the parent class (Shape) method - getData
11         this.getData();
12
13         // calculate the area of the shape
14         double area = height * width;
15
16         // display the results on the screen
17         System.out.print("Area of the " + color + " Rectangle shape (inches): ");
18         System.out.format("%.02f", area);
19         System.out.println();
20         System.out.println();
21     }
22 }
```

The Keyword ***super*** page 241

- Another keyword introduced in this chapter is ***super***. Similar to the ***this*** keyword.
- Super is a way of referring to the superclass.
- It is used in constructors to invoke the constructor of the superclass.
- It is also used to access the variables and methods of the superclass

Polymorphism page 243

- **Polymorphism** is a key concept in Object-Oriented Programming and is closely related to inheritance.
- Because inheritance models an “is a” relationship, one instance can take on the behaviors and attributes of more than one class.

Static Binding

Dynamic Binding

Static Binding page 244

- Polymorphism applies to instances rather than to classes, so static methods can be bound to their implementation at compile time.
- This is referred to as **static binding** or compile-time binding.
- Java restricts static methods from being overridden.

Static Binding

Dynamic Binding page 244

- **Dynamic binding**, also called virtual method invocation, is the binding used for instance methods to allow for polymorphism.
- Dynamic binding means binding of instance methods to the appropriate implementations is resolved at runtime, not at compile time.

Dynamic Binding

The Superclass Object page 245

- All classes in Java descend from the class Object.
- The Object class has several methods that are, therefore inherited by every other class.
- You may choose to use them directly or override them in your own class
  - protected Object close():
  - public Boolean equals (Object obj):
  - protected void finalize():
  - public final Class getClass()
  - public int hashCode():
  - public String toString():

Abstract Classes and Methods

page 246

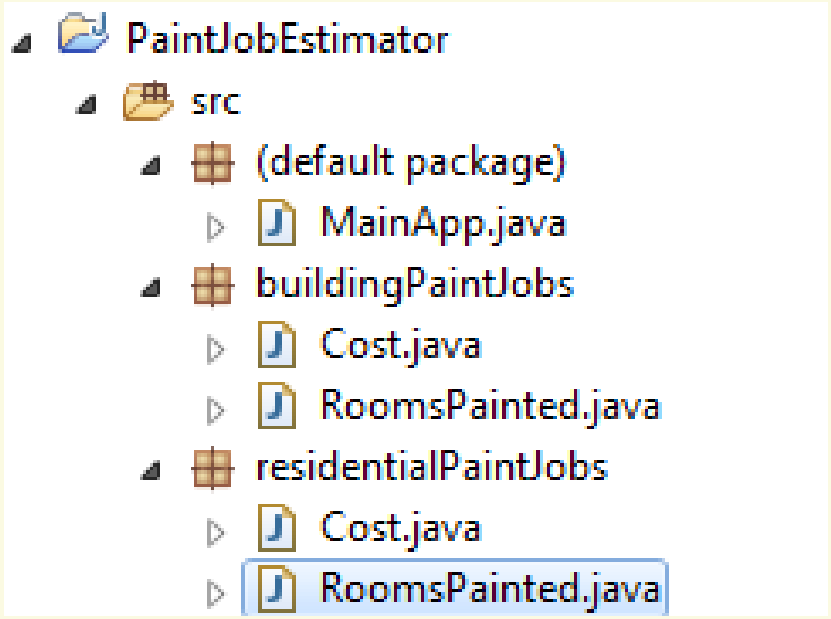
- **Abstract classes** can not be instantiated.
- They are used as superclasses when you want all instances to belong to a subclass
- The keyword abstract is used in the class declaration
- **Abstract methods** are declared without an implementation.
- All abstract methods must eventually be implemented in one of the subclasses.

Packages page 251

- **Packages** are Java construct that function something like folders to organized classes.
- Packages server as organizational tool to keep related classes together and less related classes separate.
- Packages provide unique namespaces that allow several classes with the same name to exist without conflict as long as they are located in different packages.

Packages page 251

- **Packages** serves as organizational tool to keep related classes together and less related classes separate.



Interfaces page 252

- An interface defines a protocol, or contrast, of behavior.
- It resembles something like a template for a class, as it specifies what a class must do, but not how to do it.
- Instead of a public class ClassName heading, you should use the *public interface InterfaceName* heading
- Interfaces must have either the public access modifier or no access modifier, indicating they are only accessible within the same package.

Garbage Colletion page 259

- Garbage collection is a way to reclaim memory from objects they are no longer in use.
- By removing unused objects from memory, more is available for new and existing objects.
- Java incorporates automatic garbage collection, so the programmer does not need to manage memory as much as themselves
- An object is eligible for garbage collection when it is no longer accessible through any variable.