Object Oriented Programing 1

Lecture 9: Java Class Attributes and Methods

By

Elubu Joseph

MSci.IS

Email: josebulinda@gmail.com

or

jose@kumiuniversity.ac.ug

Agenda

1. Java Attributes
2. Java Class Methods

Java Class Attributes

In the previous chapter(s), we used the term "variable“ in a number of examples we handled. In the example (as shown below) It is actually an **attribute** of the class. You can therefore say that class attributes are variables within a class:

Example

Create a class called "`KumuClass`" with two attributes: `x` and `y of int type`.

```
public class KumuClass{
  int x = 5;
  int y = 3;
}
```

Note! Another term for class attributes is **fields**.

Example 2

Create a class called "KClass" having two attributes of int and String types: age and name initialized to 20 and “KUMU” respectively

```
public class KClass{

   int age = 5;

   String name =“KUMU”;

}
```

Accessing Attributes

You can access attributes by creating an object of the class where the attributes have been created, and by using the dot syntax (.). However, static attributes may be accessed without class objects.

The following example will create an object of the KumuClass class, with the name myObj. We use the x attribute on the object to print its value.

In Java, an object is created from a class. We have already created the class named KumuClass, so now we can use this to create objects.

Class Object Creation

To create an object of any class, you have to specify the class name, followed by the object name, assignment operator, the new keyword and the class name again followed by the parenthesis:

Create an object called "myObj" and print the value of x:

```
public class KumuClass {

int x = 5;

public static void main(String[] args){
        KumuClass myObj = new KumuClass();
        System.out.println(myObj.x);
}
}
```

OUTPUT: 5

Accessing Both Static and None static attributes

Example

Create an object called "myObj" and print the value of x:

```
public class KumuClass {
  int x = 5;
  static int y=19;

  public static void main(String[] args) {

    KumuClass myObj = new KumuClass();

    System.out.println(myObj.x);

    System.out.println(y);
  }
}
```

OUTPUT:

5
19

Multiple Attributes

You can specify as many attributes as you want.

Example

```
public class Person {
      String fname = “Okello";
      String lname = “James";
      int age = 30;
public static void main(String[] args) {
    Person myObj = new Person();
    System.out.println("Name: " + myObj.fname + " " + myObj.lname);
    System.out.println("Age: " + myObj.age);
}
}
```

OUTPUT:

```
Name: Okello James
Age:30
```

Multiple Objects

Example

Create two objects of `KumuClass`:

```
public class KumuClass {
       int x = 20;
       String s =“Students”;

    public static void main(String[] args) {

       KumuClass myObj1 = new KumuClass();  // Object 1
       KumuClass myObj2 = new KumuClass();  // Object 2
       System.out.print(myObj1.x);
       System.out.println(myObj2.s);
  }
}
```

OUTPUT:

20 Students

Exercises : 1

Create an Object of OtherClass.java called oc.

OtherClass oc = new OtherClass();

Exercise: 2

Create an object of `KumuClass` called `myob`.

KumuClass myob = new KumuClass();

Exercise:

Use myob to access and print the value of the x attribute of KumuClass.

```
public class KumuClass {
     double x = 25.5;
     public static void main(String[] args) {

     KumuClass myob = new KumuClass();
     double k =  myob.x;
     System.out.println(k);
}
}
```

OUTPUT

25.5

Modify Attributes

You can also modify attribute values:

Example

Set the value of x to 40:

```
public class KumuClass{
      int x;
public static void main(String[] args){
      KumuClass myObj = new KumuClass();
      myObj.x = 40;
      System.out.println(myObj.x);
}
}
```

OUTPUT

40

Or override existing values:

Or override existing values:

Example

Change the value of x to 25:

```
public class KumuClass {
  int x = 10;

  public static void main(String[] args) {
    KumuClass myObj = new KumuClass();
    myObj.x = 25; // x is now 25
    System.out.println(myObj.x);
  }
}
```

What will be the output of the code above?

If you don't want the ability to override existing values, declare the attribute as **final**:

Example

```
public class KumuClass {
  final int x = 10;

  public static void main(String[] args) {
    KumuClass myObj = new KumuClass();
    myObj.x = 25; // will generate an error:
cannot assign a value to a final variable
    System.out.println(myObj.x);
  }
}
```

The final keyword is useful when you want a variable to always store the same value, like PI (3.14159...).
The final keyword is called a "modifier". You will learn more about these in the Java Modifiers Chapter.

Multiple Objects

If you create multiple objects of one class, you can change the attribute values in one object, without affecting the attribute values in the other:

Example

Change the value of `x` to 25 in `myObj2`, and leave `x` in `myObj1` unchanged:

```
public class KumuClass {
int x = 5;
public static void main(String[] args) {
    KumuClass myObj1 = new KumuClass();  // Object 1

    KumuClass myObj2 = new KumuClass();  // Object 2

    myObj2.x = 25;
    System.out.println(myObj1.x);  // Outputs 5
    System.out.println(myObj2.x);  // Outputs 25
}
}
```

Java Methods

Java Methods

A **method** is a block of code which only runs when it is called. You can set data, known as parameters, into a method.

Methods are used to perform certain actions, and they are also known as **functions**.

Why use methods?

To reuse code: define the code once, and use it many times.

Create a Method

A method must be declared within a class. It is defined with the name of the method, followed by parentheses **()**.

Java provides some pre-defined methods, such as System.out.println(), but you can also create your own methods to perform certain actions:

Example
Create a method called myMethod inside KumuClass:

```
public class KumuClass {
  static void myMethod() {
    // code to be executed
  }
}
```

Example Explained

`myMethod()` is the name of the method

`static` means that the method belongs to KumuClass class and not an object of the KumuClass class.

`void` means that this method does not have a return value. You will learn more about return values later in this chapter

Call a Method

To call a method in Java, write the method's name followed by two parentheses **()** and a semicolon**;**

In the following example, `myMethod()` is used to print a text (I am in action!), when it is called:

Example

Inside `main`, call the `myMethod()` method:

```
public class KumuClass {

  static void myMethod() {
    System.out.println("I am in action!");
  }

  public static void main(String[] args) {
   myMethod();
}
}

// Outputs "I am in action!"
```

A method can also be called multiple times:

Example

```
public class KumuClass {

  static void myMethod() {
    System.out.println("I am in action!");
  }

  public static void main(String[] args) {
    myMethod();
    myMethod();
    myMethod();
  }
}

// I am in action!
// I am in action!
// I am in action!
```

Exercise:

Call myMethod on the object.

```
public class KumuClass {
public void myMethod() {
System.out.println("Hello World");

}

public static void main(String[] args) {
KumuClass myObj = new KumuClass();

  myObj      .  myMethod      ();

}
}
```

Method Parameters

Information can be passed to methods as parameter. Parameters act as variables inside the method.

Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

The following example has a method that takes a String called **fname** as parameter. When the method is called, we pass along a first name, which is used inside the method to print the full name:

Example

```
public class KumuClass1 {
	static void myMethod(String fname) {
		System.out.println(fname + " is a student");
	}
	public static void main(String[] args) {
		myMethod(“Lemi");
		myMethod(“Anywar");
		myMethod(“Ambrose");
	}
}
```

OUTPUT

```
Lemi is a student
Anywar is a student
Ambrose is a student
```

Return Values

The void keyword, used in the examples above, indicates that the method should not return a value.

If you want the method to return a value, you can use a primitive data type (such as int, char, etc.) instead of void, and use the return keyword inside the method:

Example

```
public class KumuClass {

        public static void main(String[] args) {
                System.out.println(myMethod(3));
        }

        static int myMethod(int x) {
        return 5 + x;
        }
}

// Outputs 8
```

This example returns the sum of a method's **two parameters**:

Example

```
public class KumuClass {

      static int myMethod(int x, int y) {
      return x + y;
      }

      public static void main(String[] args) {
      System.out.println(myMethod(5, 10));
      }
}
// Outputs 15
```

You can also store the result in a variable (recommended):

Example

```
public class KumuClass {
  static int myMethod(int x, int y) {
    return x + y;
  }

  public static void main(String[] args) {
    int z = myMethod(5, 3);
    System.out.println(z);
  }
}
// Outputs 8 (5 + 3)
```

A Method with If...Else

It is common to
use `if...else` statements
inside methods:

Example

```
public class KumuClass {

  // Create a checkAge() method with an integer variable called age
  static void checkAge(int age) {

    // If age is less than 18, print "access denied"
    if (age < 18) {
      System.out.println("Access denied - You are not old enough!");

    // If age is greater than 18, print "access granted"
    } else {
      System.out.println("Access granted - You are old enough!");
    }

  }

  public static void main(String[] args) {
    checkAge(20); // Call the checkAge method and pass along an age of 20
  }
}
// Outputs "Access granted - You are old enough!"
```

Exercise:

Insert the missing part to call `myMethod` from `main`.

```
static void myMethod() {
System.out.println("I just got executed!");
}

public static void main(String[] args) {

  myMethod();

}
```

Java Class Methods

You learned from the Java Methods above that methods are declared within a class, and that they are used to perform certain actions:

Static or Public

You often see Java programs that have either static or public attributes and methods.

In the example above, we created a static method, which means that it can be accessed without creating an object of the class, unlike public, which can only be accessed by objects:

Example 2

An example to demonstrate the differences between `static` and `public` **methods**:

```
public class KumuClass4 {
    // Static method
   static void myStaticMethod() {
      System.out.println("Static methods can be called without creating objects");
   }

   // Public method
   public void myPublicMethod() {
      System.out.println("Public methods must be called by creating objects");
   }

   // Main method
   public static void main(String[] args) {
      myStaticMethod(); // Call the static method
      // myPublicMethod(); This would compile an error

     KumuClass4 myObj = new KumuClass4(); // Create an object of KumuClass4
     myObj.myPublicMethod(); // Call the public method on the object
   }
}
```

OUTPUT

```
Static methods can be called without creating objects
Public methods must be called by creating objects
```

Note: We will learn more about these keywords (called modifiers) in the Java Modifiers chapter.

Access Methods With an Object

Example

Create a Car object named ob. Call the fullSpeed() and speed() methods on the ob object, and run the program:

```
public class Car {

   // Inside main, call the methods on the ob object
   public static void main(String[] args) {
     Car ob = new Car();       // Create a ob object
     ob.fullSpeed();         // Call the fullSpeed() method
     ob.speed(200);            // Call the speed() method
  }

   // Create a fullSpeed() method
   public void fullSpeed(){
     System.out.println("The car is moving as fast as it can!");
  }

   // Create a speed() method and add a parameter
  public void speed(int maxSpeed) {
    System.out.println("Max speed is: " + maxSpeed);
  }

}
```

Example explained

1) We created a custom Car class with the class keyword.

2) We created the fullSpeed() and speed() methods in the Car class.

3) The fullSpeed() method and the speed() method printed out some text, when they were called.

4) The speed() method accepts an int parameter called maxSpeed - we will use this in **8)**.

5) In order to use the Car class and its methods, we needed to create an **object** of the Car Class.

6) Then, go to the main() method, which you know by now is a built-in Java method that runs your program (any code inside main is executed).

7) By using the new keyword we created a Car object with the name ob.

8) Then, we call the fullSpeed() and speed() methods on the ob object, and run the program using the name of the object (ob), followed by a dot (.), followed by the name of the method (fullSpeed(); and speed(200). Notice that we add an int parameter of **200** inside the speed() method.

Remember that..

The dot (.) is used to access the object's attributes and methods.

To call a method in Java, write the method name followed by a set of parentheses **()**, followed by a semicolon (;).

A class must have a matching filename (Car and **Car.java**).

Using Multiple Classes

Using Multiple Classes

Like we learnt before, a class is a blue print of and object. For us to be able to build bigger programs, we need to create many classes with methods that does specific tasks. After creating these classes, we will have to make sure they communicate to each other.

To allow classes communicate, we will always need to instantiate(create the class object) of the class we need to access its attributes and methods.

Using Multiple Classes+

You can create an object of a class and access it in another class. This is often used for better organization of classes (one class has all the attributes and methods, while the other class holds the main() method (code to be executed)).

Remember that the name of the java file should match the class name. In this example, we have created two files in the same package.

1. KumuClass.java
2. OtherClass.java

KumuClass.java

```
public class KumuClass {
  int x = 25;
}
```

OtherClass.java

```
class OtherClass {

  public static void main(String[] args){
      KumuClass myObj = new KumuClass();
      System.out.println(myObj.x);
  }
}
```

OUTPUT:

25

Accessing Another class methods and attributes using objects

In this example, we have created two files in the same directory:

- Car.java
- OtherClass2.java

```
public class Car {
  public void fullSpeed() {
    System.out.println("The car is moving as fast as it can!");
  }

  public void speed(int maxSpeed) {
    System.out.println("Max speed is: " + maxSpeed+ " KPH");
  }
}
```

OtherClass2.java

```
import java.util.Scanner;
class OtherClass2 {
      public static void main(String[] args) {

      Scanner sob = new Scanner(System.in);//Create Scanner object

      System.out.println("Enter Car Speed and press Enter");
      int speeder = sob.nextInt(); //Reader user input

      Car ob1 = new Car();// Create a ob1 object
     ob1.fullSpeed();// Call the fullSpeed() method
     ob1.speed(speeder);              // Call the speed() method
   }
}
```

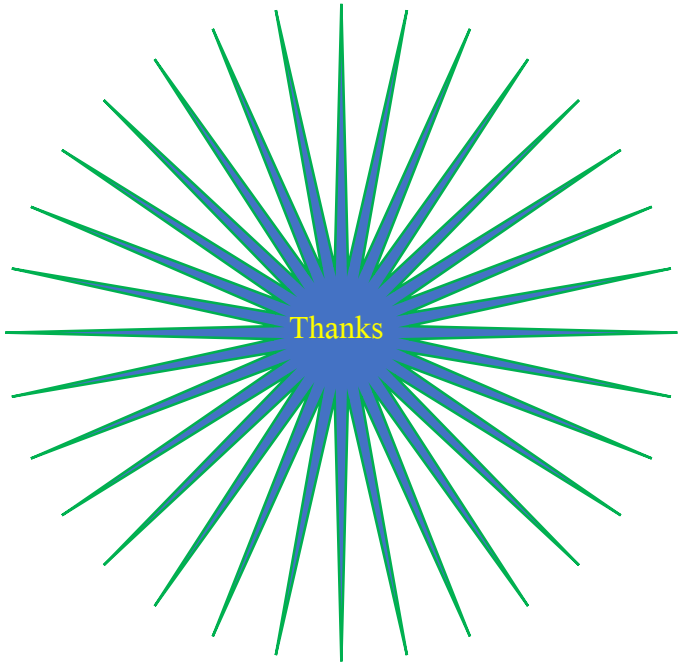
OUTPUT

```
Enter Car Speed and press Enter
180
The car is moving as fast as it can!
Max speed is: 180 KPH
```

Summary

1. Java Attributes(Creating attributes, how to access them from various methods)
2. Java Class Methods (working with multiple classes)

Thank you for Listening



Reference

Java Class Attributes. Java class attributes. (n.d.). Retrieved May 20, 2022, from https://www.w3schools.com/java/java_class_attributes.asp

Java - Methods. Tutorials Point. (n.d.). Retrieved May 28, 2022, from https://www.tutorialspoint.com/java/java_methods.htm

Java Class Methods. Java class methods. (n.d.). Retrieved May 30, 2022, from https://www.w3schools.com/java/java_class_methods.asp