Section 6: OOP Part 1 - Classes, Constructors and Inheritance

15 July 2021 19:23

Section 6: OOP Part 1 - Classes, Constructors and Inheritance

Class : A Class is like an object constructor, or a "blueprint" for creating objects or prototype from which objects are created.
Every class we create is inherited from base java class.

Object : An object in Java is the physical as well as a logical entity.

Class_Name Object_Name = new Class_Name();

Put this before private data member to modify then in public function on concur same name convention. **Example :**

```
public class Car {

   private int doors;
   private int wheels;
   private String model;
   private String engine;
   private String colour;

   public void setModel(String model)
   {
      this.model = model;
   }
}
```

Use like this in Main : porsche.setModel("Carrera");

Concept of Encapsulation : not allowing people to access the field directly. Can validate the data as well.

Default (Empty) Constructor always called when a new object is created.

Instead of calling set and get initially, there is another way of doing this when we are creating an object for the first time using a class, that's using constructors.

General rule : don't call setter from inside a construstor. Call an constructor from inside the constructor is fine.

Inheritance :
Super Keyword : Super means is to call the constructor that is for the class that we are extending from (superclass)

Reference vs **Object** vs **Instance** vs **Class** :

A **Class** is basically a blueprint(plan) for a house, using this we can build as many houses.

Each house we build (instantiate using **new** operator) is an **object** also known as **instance**.

Each house build has an address. If we want to tell somewhere where we live. This is known as **reference**. Can copy reference as many times, but it remain as one house.

We can pass **references** as **parameters** to **constructors** and **methods**.

In java we always have **references** to an **object** in memory, there is no way to access an **object** directly everything is done using a **reference**.

This vs super :

Super is used to access/call the parent class members (variables & methods)
Commonly used with method overriding, when we call a method with the same name from the parent class.

This is used to call the current class members (variables and methods).
Commonly used with constructors and setters, optionally in getters

This() call : to call a constructor from another overloaded constructor in the same class, must be first statement in constructor.

Super() call : only way to call a parent constructor. This calls parent constructor, , must be first statement in each constructor.

Note : A constructor can have a call to **super()** or **this()** but never both.

Methods Overriding vs Overloading :

Method overloading : provides 2 or more separate methods in a class with same name but different parameters. (java developers often refer overloading as Compile Time Polymorphism)

Overriding : means defining a method in a child class that already exist in the parent class with same signature (same name, same argument). Method overriding is also known as Runtime Polymorphism and Dynamic Method Dispatch.
@Override immediately above the method definition.
We can't override static methods only instance methods. And methods can be overridden only in child classes.

Static vs Instance Methods :

Static methods are declared using static modifier and can't access instance methods and instance variables directly.
Static methods don't require an instance to be created. Just type class name dot method name.

Instance methods belong to an instance of a class.
We have to instantiate the class first usually by new keyword

Static Variable share between all its instance. One instance change all will change.

Instance Variables belong to an instance of a class.

Main class automatically inherited from object class in java.