Day3: Introduction of OOPs

Object Orientation:

To develop a computer application, we have various programming paradigms, or styles of programming are there like:

- Procedural programming
- functional programming
- Object-Oriented Programming

Out of these all Object-oriented and functional are the two most popular paradigms we use these days.

These programming paradigms are ways or styles of writing code. they are not programming languages.

We classify the Programing language based on the Paradigm they support.

Java support multiple paradigms to develop an application or program.

Note: In a single application we may use the different paradigms to solve different problems.

In Object-oriented programming, everything is based on a concept called **objects**

These objects are the basic units going to contain some data which is called **state** and the **operation** on these data, called methods or behaviours. these methods modify the data (state).

So in Object-oriented programming, we bring both data and methods that operate on it together, in a single object. These objects interact with each other, to perform various tasks.

The benefits of Object-oriented programming:-

- Reduce complexity
- Easier maintenance
- Code reusability
- Faster Development

Difference between Class and Object:

Class is a template or a blueprint for the objects. whereas Object is an instance of the Class.

> Class is the virtual encapsulation of properties and behaviours. The object is the physical encapsulation of properties and behaviours.

Class:

The main purpose of the class is to represent all real-world entities in Java applications.
EX: Student, Employee, Product, Account, etc.

To represent entities data in Java applications we will use variables.
and to represent entities' behaviours we will use methods.

Basic syntax:

```
public class ClassName {

  --- variables---
  --- methods---
}
```

Instance variable in Java

A variable that is created inside the class but outside the method is known as an instance variable. The instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

Example:

```
package com.masai;
public class Main{

   int x=10; //instance variable

  public static void main(String[] args){

    int y=10; //local variable of main mehtod

    System.out.println(y); // 10



    System.out.println(x);//error
}
}
```

Explanations:

Inside a class, there will be two types of elements are there :

1. static elements (instance variables and methods)
2. non-static elements(instance variables and methods)

All the static elements will be loaded into RAM first, and all the non-static elements will be there inside the hard disk in the form of bytecode in the dot class file. now JVM searches for the main
method and control start the execution of a java program from the first line of the main method.

The Complete signature of the main method is :

```
public static void main(String[] args)
```

Note: since static elements are loaded into the RAM, they will be available for the CPU for the execution, but non-static members will not be loaded into the RAM initially. so they will not be available to the CPU for the execution.

We can not access the non-static elements from the static area directly.

By using some procedure if we can transfer the content of the hard disk to RAM (loading the non-static elements into the RAM) so that x will be loaded in the RAM and it will be available to the CPU for execution.

The procedure of loading the contents of the hard disk to the RAM dynamically at runtime
is done by creating an object of a class, this is the need for creating an object.

Requirement of creating Object in java

1. To load the non-static element of a class temporarily de the RAM so that will be available to the CPU for the execution.

2. To Access the non-static elements of any particular class we need to create an object of that class.

The object of a class is created in java by using the 'new' operator.

syntax of creating an object

```
Class_Name ref_Var = new Class_Name([param_List]);
```

Example:

```
Main obj = new Main();
```

```
package com.masai;
public class Main{

  int x=10;

  public static void main(String[] args){

    int y=10;
    System.out.println(y);

    Main obj=new Main();

    System.out.println(obj.x);
  }
}
```

‘new’ operator:

An important responsibility of the new operator is to transfer the contents of the .class file from the hard disk to the RAM.

As soon as the **'new'** operator is encountered by the JVM, JVM will allocate some memory space for all the non-static members present inside the dot class file and it will load all the non-static elements from the hard disk to the RAM and initializes with their default values if the value is not given, and if any non-static

method will be there then the signature and address of the method will be loaded, not the body of the method.
And then the address of this memory location will be assigned to the variable “d1”,
where d1 is a variable of class Demo type.

Note: the memory in the RAM which is reserved for the non-static elements of the dot class file is known as Object in java.

- Now we can say that “obj” is the variable of the class Main type which contains the address of the object.
- Now if we change the value of the x through the “obj” variable, like obj.x=20; then the change will be done only inside the RAM, but the value in the dot class file which is in the hard disk will not be affected.
- Since “obj” points to the object of class Main, it is not an object, it is only a reference to an object.
- After execution of the main method, the main method will also be deleted from the RAM and the reference variable created in the main method “obj” will also be deleted and the address to the object pointed out by “obj” will also be lost.
  the object which has no variables to point them treated like garbage.
- In Java, it is the duty of the garbage collector to clear this garbage and free the memory in the RAM.

Note: we can create multiple objects of the same class, each object contains a separate copy of the data.

Example:

```
package com.masai;
public class Main{

  int x=10;
```

```
  public static void main(String[] args){

    Main obj1=new Main();
    obj1.x=50;
    System.out.println(obj1.x); //50

    Main obj2=new Main();
    System.out.println(obj2.x); //10

  }
}
```

Note: In Java, a single object can be referenced by multiple reference variables simultaneously, but at a time one reference variable can refer to only one object.

Example

```
package com.masai;
public class Main{

  int x=10;

  public static void main(String[] args){

    Main obj1=new Main();
    obj1=50;
    System.out.println(obj1.x); //50

    Main obj2=obj1;
    System.out.println(obj2.x); //50
  }
}
```

Note: the default value of a reference variable is null. and if a variable is holding the null value and, from that variable if we try to access any non-static static element, then it will throw a “NullPointerException” at runtime.

Example:

```
package com.masai;
public class Main{
```

```
  int x=10;

  public static void main(String[] args){

    Main obj=new Main();
    obj.x=50;
    System.out.println(obj.x); //50

    obj=null; //here garbage collector will destroy the Demo object

    System.out.println(obj.x); //NullPointerException
  }
}
```

Encapsulation:

This is the basic principle of object-oriented programming, according to this principle, we bundle the data and methods, that operate on that data in a single unit.

State and Behaviours of an Object:

Data present inside the object at that instance of time is called the state of an object.

it is the current value of the instance variable of an object.

whereas, functionality that is applied to the object is known as the behaviours of an object.

We Problem:

```
package com.masai;
public class Song{

  String artist;
  String title;

  void play(){
      System.out.println(artist+" is singing "+title);
  }

    public static void main(String[] args){

      Song track1=new Song();
      track1.artist="Lata";
      track1.title="wande matram";
      track1.play();

      Song track2=new Song();
      track2.artist="Sukhwindar";
      track2.title="Jai Ho";
      track2.play();


    }
}
```

static elements inside a class:

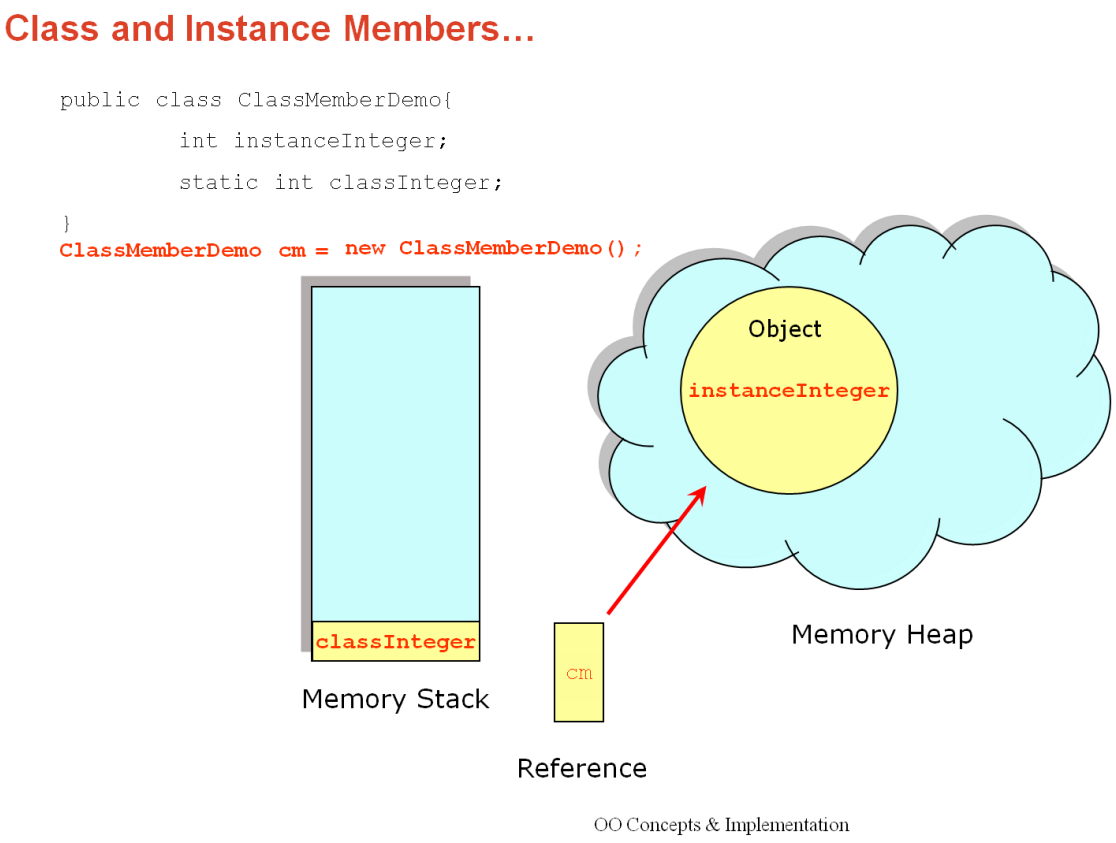
Using non-static variables and methods, we represent the state and behaviours of an object.

whereas using static variable and static method we represent class level members.

Only one copy of that member would be shared by all objects.

Non-static variables and non-static methods are also known as instance members of a class, whereas static variables and static methods are known as class level members.

static members belong to the class, whereas non-static members belong to the object.



Difference between static and non-static members:

Static Members	Non-Static Members
Belongs to class	Belong to Object
Can be accessed using ClassName.memberName	Can be accessed using objectName.memberName
Static variable initializes with default values at the time of class loading.	Non-static variable initializes with default values at the time of object creation
All objects of a class share a single copy of static variables	Each object has one local copy of the non-static variables.

Example :

For a Banking Application:

Static variables: bankName, branchName, ifscCode, phone, bankAddress

Non-static Variables: (each customer's) accountNumber, name, phone, address

Static method vs Non-static method

- The Common functionality of all objects in the application must be defined as static.
- Non-static functionality belongs to a particular object.

Example:

For an ATM application:

- Static method: To Launch a welcome screen
- Non-static method: To Launch a transaction screen

```
package com.masai;
public class Main{

  static int x;
  int y;
```

```
  public static void main(String[] args){

    Main obj1 = new Main();
    obj1.x=10;
    obj1.y=20;

    Main obj2 = new Main();
    obj2.x=50;
    obj2.y=80;

    System.out.println(obj1.x); //50
    System.out.println(obj1.y); //20

    System.out.println(obj2.x); //50
    System.out.println(obj2.y); //80

  }

}
```

Accessing static members of a class:

we can access static members of a class by using the following 3 ways:

1. directly by their name (only inside the same class)
2. by using the object
3. **by using the class name (even inside another class also).**

```
package com.masai;
public class Demo
{
  static int i=10;
```

```
  public static void fun1(){
    System.out.println("inside fun1 of Demo");
  }

  public static void main(String[] args){

    //using directly
    System.out.println(i);
    fun1();

    //by using class name
    System.out.println(Main.i);
    Main.fun1();

    //using by an object
    Main obj=new Main();
    System.out.println(obj.i);
    obj.fun1();
  }
}
```

Note:- whenever we use any class name in our application, if that class is in the current path, then that class context will be created, and all its static members will be loaded. context of a class will be created only one time for the entire JVM, whereas we can create an object of a class multiple times, whenever we require.

We Problem:

```
package com.masai;
public class Account
{
  static String bankName;

  long accountNumber;
  String accountHolderName;
  double balance;

  public static void main(String[] args){

      Account.bankName="SBI";
```

```
        Account ac1=new Account();
        ac1.accountHolderName="Ramesh";
        ac1.accountNumber=13422323432L;
        ac1.balance=5000;

        Account ac2=new Account();
        ac2.accountHolderName="Amit";
        ac2.accountNumber=25422323432L;
        ac2.balance=6000;


        System.out.println("Account 1 details");
        System.out.println(ac1.bankName);
        System.out.println(ac1.accountHolderName);
        System.out.println(ac1.balance);

        System.out.println("======================");

        System.out.println("Account 2 details");
        System.out.println(ac2.bankName);
        System.out.println(ac2.accountHolderName);
        System.out.println(ac2.balance);
    }
}
```

References:

https://javaconceptoftheday.com/non-static-members-in-java/

https://www.geeksforgeeks.org/difference-between-static-and-non-static-method-in-java/