Object Oriented Programming (OOPs)

Object Oriented Progamming means working with real life Objects so first of all we need to know that "What is an Object ?"

What is an Object ?

An object is a physical entity which exist in the read world.
Example : Car, Mouse, Remote, Laptop and so on

An Object is having 3 characteristics :

1) Identification of the Object [Name of the Object]
2) State of an Object [Data OR Properties of an Object]
3) Functionality of the Object [What an object can perform]

An Object Oriented programming is a technique through which we can develop and design the programs using CLASS and OBJECT.

Writing Programs on real life Objects is known as Object Oriented Progamming.

In OOPs we concentrate on Objects rather than function(Method).

Advantages of OOPs :

There are 3 advantages :

1) Modularity (Dividing the bigger task into number of smaller task)
2) Reusability (We can reuse the BLC classes [WORA])
3) Flexibility (Easy to maintain [Interface])

Features Of OOPs :

There are six features :

1) Class
2) Object
3) Abstraction
4) Encapsulation
5) Inheritance
6) Polymorphism

11-02-2025

What is a class :

A class is a model/bluprint/template/prototype for creating the Object. Object creation is possible by using class template.

A class is a logical entity which does not take any space.

A class is user-defined data type which contains fields and methods.

Example :

```
public class Demo
{
  Fields
   +
  Methods
}
```

*A class is a component which is used to defined Object properties and Object behavior.

```
 public class Student
 {
   Fields [Student Properties]
    +
   Methdos [Student Behavior]
 }
```

WAP to provide student properties and student behavior by using OOP.
(Diagram is available)

Student.java

```
package com.ravi.oop;

//BLC
public class Student
{
        int rollNumber;         //Non static variable
        String studentName;     //Non static variable
        String studentAddress;  //Non static variable
```

```
	public void talk()
	{
		System.out.println("My name is :"+studentName);
		System.out.println("Roll number is :"+rollNumber);
		System.out.println("Address is :"+studentAddress);
	}

	public void writeExam()
	{
		System.out.println("Hello Everyone !!!, My name is :"+studentName+ " every
saturday we have weekly test");
	}


}
```

StudentDemo.java

```
package com.ravi.oop;

//ELC
public class StudentDemo
{
	public static void main(String[] args)
	{
		Student  raj = new Student();
		//Initializing the properties through reference variable
	raj.rollNumber = 111;
	raj.studentName = "Raj";
	raj.studentAddress ="Ameerpet";

	//calling the behavior
	raj.talk();
	raj.writeExam();

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

	Student priya = new Student();
	//Initializing the properties through reference variable
	priya.rollNumber = 222;
	priya.studentName = "Priya";
	priya.studentAddress = "S R Nagar";
```

```
        //calling the Behavior
        priya.talk();
        priya.writeExam();

           }

}
```

Steps to develop OOP :
---
1) Create BLC and ELC class.
2) Create an Objec for BLC class, inside the ELC class in the main method.
3) Think about Object properties and behavior and write inside the BLC class.
4) In the ELC class, With the help of Object reference initialize all the Object properties.
5) By using Object reference call the methods.

12-02-2025

Instance Variable OR Non static Field :

Instance variable we can declare at class level.

If a non static variable is declared inside the class but outside of the method then it is called Instance Variable OR Non static field.

Example :

```
public class Student
{
  int rollNumber;  //Instance Variable OR Non static field

  public void m1()
  {
  }
}
```

An instance variable is automatically created and initialized with default value at the time of Object creation. [We can't even think about instance variable without object creation]

```
public class Test
```

```
{
	int x = 200; //Instance Variable OR Non static filed

	public static void main(String[] args)
	{
		System.out.println(x); //error
	}
}
```

An instance variable is used to represent the properties of an object hence without object instance variable will not exist.

As far as it's scope is concerned, An instance variable can be accessible anywhere within the class as well as outside of the class also using valid access modifier.

Whenever we create an object, a separate copy of all the instance variables will be created.

Initializing the Object properties using Method without parameter (Using Scanner class) :

```
package com.ravi.method_initialization;

import java.util.Scanner;

public class Employee
{
  int employeeNumber;  //0
  String employeeName; //null
  double employeeSalary; //0.0

  public void setEmployeeData()
  {
		Scanner sc = new Scanner(System.in);
		System.out.print("Enter employee Number :");
		employeeNumber = sc.nextInt();
		System.out.print("Enter employee Name :");
		employeeName = sc.nextLine();
		employeeName = sc.nextLine();
		System.out.print("Enter employee Salary :");
		employeeSalary = sc.nextDouble();
  }

  public void getEmployeeData()
  {
```

```
            System.out.println("Employee Number is :"+employeeNumber);
            System.out.println("Employee Name is :"+employeeName);
            System.out.println("Employee Salary is :"+employeeSalary);
    }

}

package com.ravi.method_initialization;

public class EmployeeDemo
{
        public static void main(String[] args)
        {
                Employee scott = new Employee();
                scott.setEmployeeData();
                scott.getEmployeeData();

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

                Employee smith = new Employee();
                smith.setEmployeeData();
                smith.getEmployeeData();

        }

}
```

What is parameter variable ?:

If a variable is declared inside a method as a parameter (not inside a method body) then it is parameter variable.

It used to recieve the values from the outer world so the end user will provide the values and these values are recieved by
parameter variable in the program.

Initializing the object properties using parameter variable :

```
package com.ravi.initialization_through_parameter;

public class Customer
{
    int customerId;
```

```
    String customerName;

    public void setCustomerData(int id, String name)
    {
        customerId = id;
        customerName = name;
    }

    public void getCustomerData()
    {
        System.out.println("Customer Id is :"+customerId);
        System.out.println("Customer Name is :"+customerName);
    }

}

package com.ravi.initialization_through_parameter;

public class CustomerDemo
{
        public static void main(String[] args)
        {
                Customer martin = new Customer();
                martin.setCustomerData(111, "Mr Martin");
                martin.getCustomerData();

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

                Customer alen = new Customer();
                alen.setCustomerData(222, "Mr. Alen");
                alen.getCustomerData();

        }

}
```

Note : Upto here, We know total 3 ways to initialize the object properties (instance variable) which are as follows :

1) Using Object reference(raj.rollNumber = 122)
2) Using Method without parameter (Scanner class)
3) Using Method with Parameter (Park Story)

What is a constructor [Constructor Introduction]

If the name of the class and name of the method both are exactly same and it should not contain any return type then it is called constructor.

Example :

Case 1:

```
public class Demo
{
  public void Demo() //Method
  {
  }
}
```

Case 2:

```
public class Demo
{
  public  Demo() //Constructor
  {
  }
}
```

13-02-2025

Default constructor added by compiler :

In java, Whenever we write a class and If we don't write any kind of constructor then automatically compiler will add one
default no argument constructor in the class at the time of compilation.

Demo.java

```
public class Demo
{
  //Programmer has not provided any constructor
}
```

javac Demo.java (Compilation)
[After compilation Demo.class file will be generated]

Demo.class

```
public class Demo
{
  public Demo() //Default No Argument Constructor [Added by
  {                                    java compiler]
  }
}
```

*Every java class must have at-least one constructor either
explicitly written by user OR implicitly added by java compiler.

The access modifier of default no argument constructor [added by java compiler] will depend upon class access modifier that means, If the class is public then default no argument constructor will also be public otherwise not.

Example :

```
public class Test
{
}
```

javac Test.java [java compiler will compile and add default
constructor]

javap Test.class [To see the Constructor added by java compiler]

Why compiler is adding default constructor to our class :

We have 2 reasons that why compiler is adding default constructor :

1) Without default constructor, Object creation is not possible in java by using new keyword, if the class does not contain user-defined constructor.

2) As we know only class level variables are having default values so, new keyword will initialize all the non static variable with the support of java compiler with default values as shown below.

Data type - Default value
byte - 0
short - 0
int - 0

long - 0
float - 0.0
double - 0.0
char - (space) '\u0000'
boolean - false
String - null
Object - null (For any class i.e reference variable the default value is null)

Program that describes new keyword is repsonsible to allocate
the default values for non static variables :

Student.java

```
package com.ravi.oop;

public class Student
{
  int roll;
  String name;

  public void show()
  {
        System.out.println(roll);  //0
        System.out.println(name);  //null
  }
}
```
StudentDemo.java

```
package com.ravi.oop;

public class StudentDemo
{
      public static void main(String[] args)
      {
            Student raj = new Student();
            raj.show();


      }

}
```

Here we will get output as default values i.e 0 , null

How to initialize the object properties with parameter variable as per our requirement :

Based on the requirement we can initailize the instance variable with parameter variable as shown in the program below :

```
package com.ravi.oop;

public class Employee
{
  int employeeId;
  String employeeName;
  double employeeSalary;
  char employeeGrade;

  public void setEmployeeData(int id, String name, double salary)
  {
        employeeId = id;
        employeeName = name;
        employeeSalary = salary;
  }

  public void calculateEmployeeGrade()
  {
        if(employeeSalary >=100000)
        {
              employeeGrade = 'A';
        }
        else if(employeeSalary >=75000)
        {
              employeeGrade = 'B';
        }
        else if(employeeSalary >=50000)
        {
              employeeGrade = 'C';
        }
        else
        {
              employeeGrade = 'D';
        }

  }
```

```
  public String getEmployeeData()
  {
            return "[Id is :"+employeeId+", Name is :"+employeeName+", Salary is
"+employeeSalary+", Grade is :"+employeeGrade+"]";
  }

}
package com.ravi.oop;

public class EmployeeDemo {

        public static void main(String[] args)
        {
                Employee scott = new Employee();
                scott.setEmployeeData(101, "Scott", 120000);
                scott.calculateEmployeeGrade();
                System.out.println(scott.getEmployeeData());

        }

}
```

14-02-2025

Role of instance variable OR Non static field (Object Properties)
while creating an object.

In java, Whenever we create an object, a separate copy of all the instance variables will be created with each and every object.

```
package com.ravi.role_of_instance_variable;

public class Test
{
  int x = 100;

  public static void main(String [] args)
  {
     Test t1 = new Test();
     Test t2 = new Test();
```

```
        ++t1.x;   --t2.x;

        System.out.println(t1.x); //101
        System.out.println(t2.x); //99
    }
}
```

What is the role of class variable OR static field in Object creation ?

What is a static field ?

It is a variable which we should declare at class level.

If we declare a variable inside a class with static modifier then it is called static field OR Class Variable.

In order to access the static field, Object is not required, We can access the static field with the help of class name.

A static field is automatically created and initialized with default value at THE TIME LOADING THE .CLASS FILE INTO JVM MEMORY.

Whenever we create an object in java then a single copy of static field will be created and the same single copy will be sharable by all the objects.

* It is sharable by all the objects hence if we modify the value
  by any of the Object reference then it will be mofifiable for
  all the objects.

```
package com.ravi.role_of_static_field;

public class Demo
{
        static int x = 100;

        public static void main(String[] args)
        {
                Demo d1 = new Demo();
                Demo d2 = new Demo();

                ++d1.x;   ++d2.x;
```

```
                System.out.println(d1.x); //102
                System.out.println(d2.x); //102
        }
}
```

Note : The main purpose of static field to save the memory.

When we should declare a variable as an instance and we should decalre as static variable ?

Instance Variable :

If the value of the variable is different with respect to object
then we should declare the variable as an instance variable.

Static Variable :

If the value of the variable will be common with respect to all the objects then we should use static variable.

Example 1:

```
public class Student
{
  int roll;
  String name;
  String address;
  static String collegeName = "NIT";
  static String courseName = "Java";
}
```

Example 2:

```
public class Customer
{
   int accountNumber;
   String customerName;
   String customerAddress;
   long mobileNumber;
   String emailId;
   static String ifscCode = "SBIHYD00123";
   static String branchLocation ="S.R Nagar";
}
```

//Programs :
---
2 files :

Student.java

```
package com.ravi.role_of_instance_variable;

public class Student
{
	int rollNumber;
	String studentName;
	String studentAddress;
	static String collegeName = "NIT";
	static String courseName = "Java";

	public void setStudentData(int roll, String name, String address)
	{
		rollNumber = roll;
		studentName = name;
		studentAddress = address;
	}

	public void showStudentData()
	{
		System.out.println("Roll Number is :"+rollNumber);
		System.out.println("Name is :"+studentName);
		System.out.println("Address is :"+studentAddress);
		System.out.println("College Name is :"+collegeName);
		System.out.println("Course is :"+courseName);
	}
}

package com.ravi.role_of_instance_variable;

public class VariableDeclaration {

	public static void main(String[] args)
	{
		Student raj = new Student();
		raj.setStudentData(101, "Raj", "Ameerpet");
		raj.showStudentData();
```

```
		System.out.println(".............");
		Student priya = new Student();
		priya.setStudentData(102, "Priya", "S.R Nagar");
		priya.showStudentData();

	}

}
```

==========

What is variable shadow ?

If class level variables and method level variables are having exactly same name then method level variable will hide class level variable inside the method body OR constructor OR block, This concept is known as Variable Shadow.

```
package com.ravi.role_of_instance_variable;

class Customer
{
	int customerId = 111;
	String customerName = "Scott";
	double customerBill = 12000;

	public void show(double customerBill)
	{
		int customerId = 222;
		String customerName = "Alen";

		System.out.println(customerId); //222
		System.out.println(customerName); //Alen
		System.out.println(customerBill); //18000

	}

}

public class VariableShadow {

	public static void main(String[] args)
	{
		Customer cust = new Customer();
		cust.show(18000);
```

```
        }

}
```
---
this keyword in java :

Whenever instance variable name and parameter variable name both are same then at the time of instance variable initialization our runtime environment will provide more priority to parameter variable/local variable, parameter variables are hiding the instance variables (Due to variable shadow)

To avoid the above said problen, Java software people introduced "this" keyword.

this keyword always refers to the current object and instance variables are the part of the object so by using this keyword we can represent instance variable.

We cannot use this (non static member) keyword from static area (Static context)[static method, static block and static nested ineer class]

2 files :

Manager.java

```
package com.ravi.this_keyword;

public class Manager
{
  int managerId;
  String managerName;

  public void setManagerData(int managerId, String managerName)
  {
          this.managerId = managerId;
          this.managerName = managerName;

  }

  public void getManagerData()
  {
         System.out.println("Manager Id is :"+managerId);
         System.out.println("Manager Name is :"+managerName);
  }
```

```
}

ManagerDemo.java
----------------
package com.ravi.this_keyword;

public class ManagerDemo {

	public static void main(String[] args)
	{
		Manager alen = new Manager();
		alen.setManagerData(111, "Alen");
       alen.getManagerData();
	}

}
-----------------------------------------------------------------
Program on Local search algorithm :[Diagram 15-FEB]

package com.ravi.this_keyword;

class Test
{
	static int a = 100;  //static Field
	int b = 200; //Non static Field

	public void accept(int c)   //Parameter Variable
	{
		int d = 400;  //Local Variable
		System.out.println("Static Field :"+Test.a);
		System.out.println("Non Static Field :"+this.b);
		System.out.println("Parameter Variable :"+c);
		System.out.println("Local Variable :"+d);
	}
}
public class LocalSearchAlgorithm
{
	public static void main(String[] args)
	{
		Test t1 = new Test();
		t1.accept(300);
	}
```

}
==========
17-02-2025

**What is Data Hiding ?

Data hiding is nothing but declaring our data members with private access modifier so our data will not be accessible from outer world that means no one can access our data directly from outside of the class.

*We should provide the accessibility of our data through methods so we can perform VALIDATION ON DATA which are coming from outer world.

```
package com.ravi.data_hiding;

public class Customer
{
  private double balance = 10000;  //Data Hiding

  public void deposit(double amount)
  {
        //Data Validation
        if(amount <=0)
        {
              System.err.println("Amount can't deposited");
              System.exit(0);
        }
        else
        {
              this.balance = this.balance + amount;
              System.out.println("Balance After Deposit is :"+this.balance);
        }
  }

  public void withdraw(double amount)
  {
        if(amount > this.balance)
        {
              System.err.println("Withdraw is not possible, Low Balance");
        }
        else
        {
```

```
                this.balance = this.balance - amount;
                System.out.println("Balance After Withdraw is :"+this.balance);
            }

    }

}

package com.ravi.data_hiding;

public class DataHidingDemo {

        public static void main(String[] args)
        {
                Customer scott = new Customer();
                scott.deposit(1000);
                scott.withdraw(5000);
        }

}
```

How to print Object properties (Non static field) by using toString() method :

In java, There is a class called Object class available in java.lang pacakage. This class is by default the super class for all the classes in java.

Any predefined OR user-defined class in java, can use the method of Object class directly because Object is the super class.

In Object there is a predefined non static method called toString(), Any java class can override this toString() methdo to perform some common operation.

```
        public String toString();
```

* If we want to print our object properties i.e instance variables in String format then we can override toString() method in the respective class.

* This toString() method is available in Object class and we can generate using Eclipse IDE in our program

Right click on the Program -> source -> generate toString()

How to call toString() method :

If we pass our object reference to the System.out.println() statement then automatically it will call toString() method of the corresponding class.

Example :

```
Manager raj = new Manager();
System.out.println(raj); //It will call toString() method of
                         Manager class


Player virat = new Player();
System.out.println(virat); //It will call toString() method of
                         Player class
```

Programs :

```
package com.ravi.to_string;

public class Player {
	private int playerId;
	private String playerName;
	private double basePrice;

	public void setPlayerData(int playerId, String playerName, double basePrice) {
		this.playerId = playerId;
		this.playerName = playerName;
		this.basePrice = basePrice;
	}

	@Override
	public String toString() {
		return "Player [playerId=" + playerId + ", playerName=" + playerName + ",
basePrice=" + basePrice + "]";
	}

}
```

```
package com.ravi.to_string;

public class PlayerDemo {
```

```
	public static void main(String[] args)
	{
		Player virat = new Player();
		virat.setPlayerData(18, "Virat Kohli", 2000000);
		System.out.println(virat);

		System.out.println(".....");
		Player rohit = new Player();
		rohit.setPlayerData(45, "Rohit Sharma", 3999990);
		System.out.println(rohit);

	}

}
```

Constructor :

What is the advantage of writing constructor in our class ?

If we don't write a constructor in our program then variable initialization and variable re-initialization both are done in two different lines.

If we write constructor in our program then variable initialization and variable re-initialization both are done in the same line i.e at the time of Object creation. [17-FEB]

With Constructor approach, we need not to depend on method to re-initialize our instance variable with user value.

18-02-2025

Constructor Defination :

It is used to construct the object that is the reason it is known as Constructor.

If the name of the class and name of the method both are exactly same and it does not contain any return type then it is called Constructor.

Example :

```
public class Student
{
  public Student()  //Constructor
  {
```

```
  }

  public void Student() //Method
  {
  }
}
```

Program :

```
---------
package com.ravi.constructor;
public class Student
{
 public void Student()
 {
          System.out.println("It is a method");
 }
}
```

```
package com.ravi.constructor;

public class ConstructorDemo {

        public static void main(String[] args)
        {
                System.out.println("Main");

                Student s1 = new Student();
                s1.Student(); //Calling the Method
        }

}
```

The main purpose of constructor to initialize the instance variable i.e to initialize the Object properties.

A constructor never contain any return type including void also.

Every class in java must have at-least one constructor, either
explicitly written by developer OR implicitly added by java compiler.

Whenever we create an object in java by using new keyword, at-least one constructor must be invoked.

A constructor is automatically called and executed at the time of creating the Object. [We need not to call manually like method]

A default constructor will be added by compiler in the class, in case user has not provided any type of constructor in the class.

The access modifier of default constructor will depend upon the class access modifier.

A Constructor is called only once per object that means if we create multiple objects then multiple time constructor will be invoked.

Types of Constructor :

In java we have total 3 types of Constructor :

1) Default No Argument Constructor [Added by java Compiler]

2) No Argument OR Non parameterized OR Zero Argument
   constructor [Written by user without parameter]

3) Parameterized Constructor.

Default Constructor :

Whenever we write a class and if we don't write any type of constructor then automatically a default no argument constructor will be added by java compiler.

The main purpose of this default no argument constructor to accept default values with the help of new keyword and java compiler.

Example :

Example.java

```
public class Example
{

}
```

javac Example.java

```
public class Example
{
   public Example() //Default No Argument Constructor
   {
   }
}
```

No Argument Constructor :

If a user is writing constructor in the class without any parameter then it is called No Argument Constructor.

Employee.java

```
public class Employee
{
  public Employee() //No Argument Constructor
  {
  }
}
```

In this approach, all the objects will be initialized with same value (as shown in the program) so, it is not a recommended way because every object must have unique values.

```
package com.ravi.no_args_constr;

public class Person {
	private int personId;
	private String personName;
	private int personAge;

	public Person() // No Argument Constructor
	{
		this.personId = 101;
		this.personName = "Scott";
		this.personAge = 24;
	}

	@Override
	public String toString() {
		return "Person [personId=" + personId + ", personName=" + personName + ",
personAge=" + personAge + "]";
	}
```

```
}


package com.ravi.no_args_constr;

public class PersonDemo {

	public static void main(String[] args)
	{
		Person scott = new Person();
		System.out.println(scott);

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

		Person smith = new Person();
		System.out.println(smith);


	}

}
```

Note : Here both the objects will be initialized with same value so better to use Parameterized Constructor.

Parameterized Constructor :

If we pass one or more argument to the constructor then it is called parameterized constructor.

The main purpose of paramterized constructor to initialize the instance variable of the class with different values.

Example :

```
public class Customer
{
  private int customerId;
  private String customerName;

  public Customer(int customerId, String customerName)
  {
    this.customerId = customerId;
```

```
    this.customerName = customerName;
  }
}

//Program on Parameterized Constructor :
----------------------------------------
Dog.java
---------
package com.ravi.parameterized_constructor;

public class Dog
{
	private String dogName;
	private int dogAge;
	private double dogHeight;
	private String dogColor;


	public Dog(String dogName, int dogAge, double dogHeight, String dogColor)
	{
		super();
		this.dogName = dogName;
		this.dogAge = dogAge;
		this.dogHeight = dogHeight;
		this.dogColor = dogColor;
	}

	@Override
	public String toString()
	{
		return "Dog [dogName=" + dogName + ", dogAge=" + dogAge + ", dogHeight=" +
dogHeight + ", dogColor=" + dogColor
				+ "]";
	}

}


ParameterizedConstructor.java
-----------------------------
package com.ravi.parameterized_constructor;

public class ParameterizedConstructor {
```

```
	public static void main(String[] args)
	{
		Dog tommy = new Dog("Tommy", 3, 2.3, "Grey");
		System.out.println(tommy);

		System.out.println(".............");
		Dog tiger = new Dog("Tiger", 5, 4.4, "Black");
		System.out.println(tiger);
	}

}
```

Upto here, by using following ways we can initialize the Object Properties :

---

1) By using Object reference
2) By using Method without parameter
3) By using Method With Parameter
4) At the time of variable declaration
5) Using No Argument Constructor
6) Using parameterized Constructor [Best Approach]

IQ :

---

return keyword added by java compiler :

Whenever we write a method with void keyword OR we are writing constructor then at the last statement of method as constructor, automatically compiler will add one return statement so the control will return back to the caller method.

Demo.java

```
public class Demo
{
	public void m1()
	{

	}
}
```

javac Demo.java

Demo.class

```
public class Demo
{
	public void m1()
	{

		return;
	}
}
```

The following command is used to verify the return statement provided by java compiler.

```
javac Demo.java

javap -c Demo.class
```

Constructor with return statement :

```
package com.ravi.no_args_constr;

class Student
{
	public Student()
	{
		System.out.println("Student Constructor");
		return;
	}

	public int m1()
	{
		System.out.println("It is a method");
		return 10;
	}

}


public class ConstructorWithReturn {

	public static void main(String[] args)
	{
		Student s1 = new Student();
```

```
        System.out.println(s1.m1());
           }

}
```

How to modify the existing Object data ?

In order to modify the existing Object data, we should use setter concept, On the other hand to read the private variable value from BLC class to any other class we should use getter concept.

How to write setter and getter for instance variable ?

```
public class Employee
{
  private double salary;   //[Data hiding]

  public Employee(double salary)
  {
    this.salary = salary;
  }

  public void setSalary(double salary)  //setter                    {
    this.salary = salary;
  }

  public double getSalary() //getter
  {
     return this.salary;
  }
}
```

FINAL CONCLUSION :

Parameterized Constructor : To initialize the Object properties with user values.

Setter : To modify the existing object data.[Only one data at a time] OR Writing/Overriding Operation

Getter : To read/retrieve private data value outside of BLC class. [Reading Operation]

toString() : To print Object properties (Instance Variable)

*** What is Encapsulation

[Accessing our private data with public methods like setter and getter]

Binding the private data with its associated method in a single unit is called Encapsulation.

Encapsulation ensures that our private data (Object Properties) must be accessible via public methods like setter and getter.

It provides security because our data is private (Data Hiding) and it is only accessible via public methods WITH PROPER DATA VALIDATION.

In java, class is the example of encapsulation.

How to achieve encapsulation in a class :

In order to achieve encapsulation we should follow the following two techniques :

1) Declare all the data members with private access modifiers (Data Hiding OR Data Security)

2) Write public methods to perform read(getter) and write(setter) operation on these private data like setter and getter.

Note : If we decalre all our data with private access modifier then it is called TIGHTLY ENCAPSULATED CLASS. On the other hand if we declare our data other then private access modifier then it is called Loosely Encapsulated class.

Program on Encapsulation :

```
package com.ravi.encapsulation;

public class Employee
{
	private int employeeId;
	private String employeeName;
	private double employeeSalary;

	public Employee(int employeeId, String employeeName, double employeeSalary)
	{
		super();
```

```
		this.employeeId = employeeId;
		this.employeeName = employeeName;
		this.employeeSalary = employeeSalary;
	}

	@Override
	public String toString()
	{
		return "Employee [employeeId=" + employeeId + ", employeeName=" +
employeeName + ", employeeSalary="
				+ employeeSalary + "]";
	}

	public int getEmployeeId()
	{
		return this.employeeId;
	}

	public void setEmployeeId(int employeeId)
	{
		this.employeeId = employeeId;
	}

	public String getEmployeeName()
	{
		return this.employeeName;
	}

	public void setEmployeeName(String employeeName)
	{
		this.employeeName = employeeName;
	}

	public double getEmployeeSalary()
	{
		return this.employeeSalary;
	}

	public void setEmployeeSalary(double employeeSalary)
	{
		this.employeeSalary = employeeSalary;
	}
```

```
}

package com.ravi.encapsulation;

import java.util.Scanner;

public class EncapsulationDemo {

	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);
		System.out.print("Enter Employee Id :");
		int id = sc.nextInt();
		System.out.print("Enter Employee Name :");
		String name = sc.nextLine();
		name = sc.nextLine();
		System.out.print("Enter Employee Salary :");
		double salary  = sc.nextDouble();

		Employee scott = new Employee(id, name, salary);
		System.out.println("Original Data :"+scott);

		System.out.print("Enter the updated Salary :");
		double increment = sc.nextDouble();

		if(increment <= 0)
		{
			System.err.println("Increment amount must be positive");
		}
		else
		{
		scott.setEmployeeSalary(scott.getEmployeeSalary() + increment);
		System.out.println("After Salary Increment :"+scott);
		}

		/*Based on the Salary print whether the Employee is Developer, Designer
		  OR Tester
		   Salary >= 50000  -> Developer
		   Salary >= 35000  -> Designer
		   else Tester
		*/

		double employeeSalary = scott.getEmployeeSalary();
```

```
		if(employeeSalary >=50000)
		{
			System.out.println(scott.getEmployeeName()+" is a Developer");
		}
		else if(employeeSalary >=35000)
		{
			System.out.println(scott.getEmployeeName()+" is a Designer");
		}
		else
		{
			System.out.println(scott.getEmployeeName()+" is a Tester");
		}
		sc.close();
	}

}
```

Method return type as a class :

While declaring a method in java, return type is compulsory.
As a method return type we have following options

1) void as a retutn type of the Method

2) Any primitive data type as a return type of the method.

3) Any class name/interface / enum / record we can take as a return type of the method.

Program on Method return type as a class :

```
package com.ravi.method_return_type;

public class Demo
{
	public Demo(int x)
	{

	}


   public Demo m1()
   {
```

```
        //return null;

        //return new Demo(10);

        Demo d1 = new Demo(100);
        return d1;
    }
}
```

Note : In the above program, in order to return the value we depend upon current class constructor.

What is a Factory Method ?

If a method return type is class that means that method will return current object then that type of method is known as Factory Method.

Example :

```
public class Sample
{
   public Sample getSampleObject()
   {

      return new Sample(); //to return this value we depend
   }                       on Sample class default constr
 }
```

//Program on Static Factory Method :

---

2 files :

Book.java

```
package com.ravi.static_factory_method;

public class Book
{
        private String author;
        private String title;
        private double price;
```

```
	public Book(String author, String title, double price)
	{
		this.author = author;
		this.title = title;
		this.price = price;
	}

	public String toString()
	{
		return "Book [author=" + author + ", title=" + title + ", price=" + price + "]";
	}


	//Static Factory Method
	public static Book getBookObject()
	{
		Book b1 = new Book("James", "Java", 1200);
		return b1;
	}

}
```

BookDemo.java

```
package com.ravi.static_factory_method;

public class BookDemo {

	public static void main(String[] args)
	{
	  Book book = Book.getBookObject();
      System.out.println(book);
	}

}
```

Note : In the above program, getBookObject() method returns the Book object but as we know a method is mainly used for
reusability purpose but here we are receiving only object

The following program explains, how to get multiple object from the static factory method.

2 files :

Customer.java

```
package com.ravi.static_factory_method;

import java.util.Scanner;

public class Customer
{
	private int customerId;
	private String customerName;
	private double customerBill;

	public Customer(int customerId, String customerName, double customerBill) {
		super();
		this.customerId = customerId;
		this.customerName = customerName;
		this.customerBill = customerBill;
	}

	@Override
	public String toString()
	{
		return "Customer [customerId=" + customerId + ", customerName=" +
customerName + ", customerBill="
				+ customerBill + "]";
	}

	public static Customer getCustomerObject()
	{
		Scanner sc = new Scanner(System.in);
		System.out.print("Enter Customer Id :");
		int id = sc.nextInt();

		System.out.print("Enter Customer Name :");
		String name = sc.nextLine();
		name = sc.nextLine();

		System.out.print("Enter Customer Bill :");
		double bill = sc.nextDouble();

		Customer c1 = new Customer(id, name, bill);
		return c1;
```

```
	}



}
```

CustomerDemo.java

```
package com.ravi.static_factory_method;

import java.util.Scanner;

public class CustomerDemo {

	public static void main(String[] args)
	{
	  Scanner sc = new Scanner(System.in);
	  System.out.println("How many objects you want to create :");
	  int noOfObj = sc.nextInt();

	  for(int i=1; i<=noOfObj; i++)
	  {
		Customer customer = Customer.getCustomerObject();
		System.out.println(customer);
	  }

	  sc.close();

	}

}
```

What is Shallow and Deep copy in java :

Shallow Copy :

In Shallow copy, Only one Object will be created but the same object will be refered by multiple reference variables.

If we modify the object properties by any of the reference variable then original object will be modified as shown in the program :

2 files :

```
package com.nit.shallow_copy;

public class Product {
	private int productId;
	private String productName;
	private double productPrice;

	public Product(int productId, String productName, double productPrice)
	{
		super();
		this.productId = productId;
		this.productName = productName;
		this.productPrice = productPrice;
	}

	@Override
	public String toString() {
		return "Product [productId=" + productId + ", productName=" + productName + ",
productPrice=" + productPrice
				+ "]";
	}

	public int getProductId() {
		return productId;
	}

	public void setProductId(int productId) {
		this.productId = productId;
	}

	public String getProductName() {
		return productName;
	}

	public void setProductName(String productName) {
		this.productName = productName;
	}

	public double getProductPrice() {
		return productPrice;
	}
```

```
	public void setProductPrice(double productPrice) {
		this.productPrice = productPrice;
	}

}
```

```
package com.nit.shallow_copy;

public class ProductDemo {

	public static void main(String[] args)
	{
		Product p1 = new Product(111, "M Series", 22000);
		System.out.println(p1);

		Product p2 = p1;
		System.out.println(p2);

		System.out.println(".....");
		p2.setProductId(222);
		p2.setProductName("X Series");
		p2.setProductPrice(25000);

		System.out.println("After Modification");
		System.out.println(p1);
		System.out.println(p2);

	}
}
```

Note : Here in shallow copy by using p2 reference variable we are modifying the content of object by using setter method, p1 and p2 both are pointing to the same object so original object will modify (20-FEB-25)

Deep copy :

In deep copy we will create more than one object.

Here 2nd object will copy the content of first object.

Since, Objects are created in two different memory location so, if we modify the content of one object then another object content will remain unchangd.

2 files :

```
package com.ravi.deep_copy;

public class Laptop
{
	private String laptopBrand;
	private double laptopPrice;

	public Laptop(String laptopBrand, double laptopPrice) {
		super();
		this.laptopBrand = laptopBrand;
		this.laptopPrice = laptopPrice;
	}

	public Laptop()
	{
	}


	@Override
	public String toString() {
		return "Laptop [laptopBrand=" + laptopBrand + ", laptopPrice=" + laptopPrice +
"]";
	}

	public String getLaptopBrand() {
		return laptopBrand;
	}

	public void setLaptopBrand(String laptopBrand) {
		this.laptopBrand = laptopBrand;
	}

	public double getLaptopPrice() {
		return laptopPrice;
	}

	public void setLaptopPrice(double laptopPrice) {
		this.laptopPrice = laptopPrice;
	}

}
```

```
package com.ravi.deep_copy;

public class DeepCopy
{
  public static void main(String[] args)
  {
        Laptop obj1 = new Laptop("HP", 90000);
        Laptop obj2 = new Laptop();

        obj2.setLaptopBrand(obj1.getLaptopBrand());
        obj2.setLaptopPrice(obj1.getLaptopPrice());

        System.out.println(obj1);
        System.out.println(obj2);

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

        //Modifying Object1 data
        obj1.setLaptopBrand("Apple");
        obj1.setLaptopPrice(150000);

        System.out.println(obj1);
        System.out.println(obj2);


  }
}
```

Note : Here when we modify the object 1 data (Apple, 1500000)
the 2nd object data will remain unchanged because two different Objects are created in two different memory location.

***Pass by Value :

Java does not support pointers so java only works with pass by value.

Pass by value means we are sending the copy of orginal data to the method.

```
package com.ravi.pass_by_value;

public class PassByValueDemo1 {
```

```
	public static void main(String[] args)
	{
		int x = 100;
		m1(x);
		System.out.println("x value is :"+x);
	}

	public static void m1(int val)
	{
		val = 200;
	}

}
```

Output is : 100

```
package com.ravi.pass_by_value;

public class PassByValueDemo2
{
	public static void main(String[] args)
	{
       int x = 100;
       int y = accept(x);
       System.out.println(x +" : "+y);
	}
	public static int accept(int val)
	{
		val = 500;
		return val;
	}

}
```

```
package com.ravi.pass_by_value;

class Product
{
	private double price = 12000;

	public double getPrice()
	{
		return price;
```

```
	}

	public void setPrice(double price)
	{
		this.price = price;
	}
}

public class PassByValueDemo3 {

	public static void main(String[] args)
	{
		Product p1 = new Product();
		System.out.println(p1.getPrice()); //12000
		accept(p1);
		System.out.println("After Method calling :");
		System.out.println(p1.getPrice());//18000
	}

	public static void accept(Product prod)
	{
		prod.setPrice(18000);
	}
}
-----------------------------------------------------------------
package com.ravi.pass_by_value;

class Customer
{
	private double customerBill = 12000;

	public double getCustomerBill()
	{
		return customerBill;
	}

	public void setCustomerBill(double customerBill)
	{
		this.customerBill = customerBill;
	}
}

public class PassByValueDemo4
```

```
{
	public static void main(String[] args)
	{
		Customer c1 = new Customer();
		System.out.println(c1.getCustomerBill()); //12000

		accept(c1);
		System.out.println("After Method Calling :");
		System.out.println(c1.getCustomerBill()); //12000
	}

	public static void accept(Customer cust)  //cust = c1
	{
		cust = new Customer();
		cust.setCustomerBill(25000);
	}
}
```

Output is 12000 and 12000

==========

22-02-2025

In java we have immutable objects are also available.

Immutable objects means it will not modify (Un-modifiable) so we if try modify the objects then modification will not effect on
the existing object it will create another object in another memory location.

In Java, String class and all the Wrapper classes are immutable so un-modifiable.

```
package com.ravi.unmodifiable;

public class ImmutableDemo1 {

	public static void main(String[] args)
	{
		String str = new String("Hello");
		System.out.println("Before Method call :"+str);
      accept(str);
      System.out.println("After Method call :"+str);
	}

	public static void accept(String s1)
```

```
	{
		s1 = "Hello World";
	}

}
```

Here : str is pointing to "Hello" object but Strins are immutable so original object will not be modified hence we will get the output as Hello Hello.

```
package com.ravi.unmodifiable;

public class ImmutableDemo2 {

	public static void main(String[] args)
	{
		Integer i = new Integer(100);
		System.out.println("Before Method Call :"+i);
		accept(i);
		System.out.println("After Method Call :"+i);

	}

	public static void accept(Integer val)
	{
		val = 500;
	}

}
```

Output : 100 100

HEAP Memory Management by using Garbage Collector :

Whenever we create an object in java then with the help of new keyword memory is allocated for the Object (Non static members)
in the HEAP area. [Diagram 22-FEB-25]

In older languages like C++, Memory allocation and memory de-allocation is the responsibility of programmer

If C++ programmer forgot to delete the memory then there is a chance of getting OutOfMemoryError.

In Java, A programmer is only responsible to allocate the memory in the HEAP area, Memory de-allocation is automatically done by
a component of JVM called Garbege Collector.

What is Garbage Collector in java [Deleting the unused objects]

It is a autometic memory management in java. It is used to delete un-used objects from the HEAP memory.

Garbage Collector is a daemon thread, It will scan the HEAP area,
identify which objects are eligible for Garbage Collector (Object which does not contain any references) then delete those objects from the HEAP memory so we can create another object inside the HEAP memory.

Garbage Collector internally uses an algorithm called Mark and Sweep algorithm to delete the objects from the HEAP memory.

JVM will give the instruction to Garbage Collector to visit the heap memory if the heap memory is 60-70% filled up. A developer can also call Garbage Collector explicitly by using the following static method of System class

System.gc(); //Calling the Garbage Collector explicitly

How many ways we can make an object eligible for Garbage Collector :

There are 3 ways we can make an object eligible for GC.

1) Assigning null literal to existing reference variable :

```
Employee e1 = new Employee(111,"Ravi");
        e1 = null;
```

2) Creating an Object inside a method :

```
public void createObject()
{
   Employee e2 = new Employee();
}
```

Here we are creating Employee object inside the method so, once the method execution is over then e2 will be deleted from the Stack Frame and the employee object will become eligible for GC.

3) Assigning new Object to the old existing reference variable:

```
Employee e3 = new Employee();
e3 = new Employee();
```

Earlier e3 variable was poting to Employee object after that a new Employee Object is created which is pointing to another Object in new memory location so the first object is eligible for GC.
==========

Memory in java :

In java, whenever we create an object then Object and its content (properties and behavior) are stroed in a special memory called HEAP Memory. Garbage collector visits heap memory only.

All the local variables and parameters variables are executed in Stack Frame and available in Stack Memory.

24-02-2025

HEAP and STACK Digram :

Program 01:

HEAP and STACK Diagram for CustomerDemo.java

```
class Customer
{
	private String name;
	private int id;

	public Customer(String name , int id)
	{
		super();
		this.name=name;
		this.id=id;
	}

	public void setId(int id)
	{
		this.id=id;
	}

	public int getId()
	{
```

```
		return this.id;
	}
}

public class CustomerDemo
{
	public static void main(String[] args)
	{
		int val = 100;

		Customer c = new Customer("Ravi",2);

       m1(c);

		//GC

		System.out.println(c.getId());
	}

	public static void m1(Customer cust)
	{
		cust.setId(5);

	    cust = new Customer("Rahul",7);

		cust.setId(9);
		System.out.println(cust.getId());
	}
}

//9 5
-----------------------------------------------------------------
HEAP and STACK Diagram for Employee.java
----------------------------------------
public class Employee
{
	int id = 100;

	public static void main(String[] args)
	{
		int val = 200;

		Employee e1 = new Employee();
```

```
                e1.id = val;

                update(e1);

                System.out.println(e1.id);

        Employee e2 = new Employee();

                e2.id = 900;

                switchEmployees(e2,e1); //3000x, 1000x

                  //GC [2 Objects, 2000x and 4000x are eligible]

                        System.out.println(e1.id);
                    System.out.println(e2.id);
            }

           public static void update(Employee e)
           {
        e.id = 500;
                e = new Employee();
                e.id = 400;
                System.out.println(e.id);
           }

           public static void switchEmployees(Employee e1, Employee e2)
            {
                    int temp = e1.id;
                    e1.id = e2.id; //500
                    e2 = new Employee();
                    e2.id = temp;
            }
    }

//Output 400 500 500 500
```

25-02-2025

HEAP and STACK Diagram for Sample.java

---

public class Sample

```
{
	private Integer i1 = 900;

	public static void main(String[] args)
	{
		Sample s1 = new Sample();

		Sample s2 = new Sample();

	   Sample s3 = modify(s2);

		s1 = null;

	//GC [4 objects, 1000x, 2000x, 5000x and 6000x are eligible]

		System.out.println(s2.i1);
	}
   public static Sample modify(Sample s)
	{
		s.i1=9;
		s = new Sample();
		s.i1= 20;
      System.out.println(s.i1);
		s=null;
		return s;
	}
}

//20 9
-----------------------------------------------------------------
HEAP and STACK Diagram for Test.java
-----------------------------------
public class Test
{
	Test t;
	int val;

	public Test(int val)
	{
		this.val = val;
	}

	public Test(int val, Test t)
```

```
	{
		this.val = val;
		this.t = t;
	}

	public static void main(String[] args)
	{
		Test t1 = new Test(100);

		Test t2 = new Test(200,t1);

		Test t3 = new Test(300,t1);

		Test t4 = new Test(400,t2);

		t2.t = t3;
		t3.t = t4;
		t1.t = t2.t;
		t2.t = t4.t;

	System.out.println(t1.t.val);
	System.out.println(t2.t.val);
	System.out.println(t3.t.val);
	System.out.println(t4.t.val);
	}

}
```

HEAP and STACK Digarm for Demo.java

```
class Demo
{
int x;
int y;

void m1(Demo d)
{
x=x+1;
y=y+2;

d.x=d.x+3;
d.y=d.y+4;
}
```

```
public static void main(String[] args)
{
Demo d1=new Demo();
Demo d2=new Demo();

d1.m1(d2);

System.out.println(d1.x+"... "+d1.y);
System.out.println(d2.x+"... "+d2.y);

d2.m1(d1);
System.out.println(d1.x+"... "+d1.y);
System.out.println(d2.x+"... "+d2.y);

d1.m1(d1);
System.out.println(d1.x+"... "+d1.y);
System.out.println(d2.x+"... "+d2.y);

d2.m1(d2);
System.out.println(d1.x+"... "+d1.y);
System.out.println(d2.x+"... "+d2.y);
 }
}
```

==========

Passing an Object reference to the Constructor :(Copy Constructor)

In java, We can pass an object reference to the Constructor. The main purpose of passing an object reference to the constructor to copy the content of one object to another object.

Example :

```
class Employee
{
}
class Manager
{
  public Manager(Employee emp)
  {
  }
}

//Program that describes how to copy the content of one object from another object
```

3 files :

Employee.java

```
package com.ravi.passing_object_ref;

public class Employee
{
	private int employeeId;
	private String employeeName;

	public Employee(int employeeId, String employeeName)
	{
		super();
		this.employeeId = employeeId;
		this.employeeName = employeeName;
	}

	public int getEmployeeId() {
		return employeeId;
	}

	public String getEmployeeName() {
		return employeeName;
	}
}
```

Manager.java

```
package com.ravi.passing_object_ref;

public class Manager
{
	private int managerId;
	private String managerName;

	public Manager(Employee emp)
	{
		this.managerId = emp.getEmployeeId();
		this.managerName = emp.getEmployeeName();
	}

	@Override
```

```
	public String toString() {
		return "Manager [managerId=" + managerId + ", managerName=" +
managerName + "]";
	}

}
```

CopyConstructor.java

```
package com.ravi.passing_object_ref;

public class CopyConstructor
{
	public static void main(String[] args)
	{
		Employee e1 = new Employee(111, "Scott");

		Manager m1 = new Manager(e1);
	   System.out.println(m1);
	}

}
```

Note : In the above program we have copied the data of Employee object to initialize the Manager Object.

The following program explains how to copy the same object data to another object of same type.

2 files :

```
package com.ravi.passing_object_ref;

public class Product
{
	private int productId;
	private String productName;

	public Product(int productId, String productName)
	{
		super();
		this.productId = productId;
		this.productName = productName;
```

```
	}

	public Product(Product prod) //prod = p1
	{
		this.productId = prod.productId;
		this.productName = prod.productName;
	}

	@Override
	public String toString() {
		return "Product [productId=" + productId + ", productName=" + productName +
"]";
	}
}

package com.ravi.passing_object_ref;

public class CopyConstructorDemo {

	public static void main(String[] args)
	{
		Product p1 = new Product(111, "Laptop");
		Product p2 = new Product(p1);

		System.out.println(p1);
		System.out.println(p2);

	}

}
```

==========

26-02-2025

Scenario Based Programs :

Lab Program :(Method return type as a class + Passing Object ref)

A BLC class called Customer is given to you.

The task is to find the applicable Credit card Type and create CardType object based on the Credit Points of a customer.

Define the following for the class.

Attributes :

customerName : String,private

creditPoints: int, private

Constructor :

parameterizedConstructor: for both cusotmerName & creditPoints in that order.

Methods :

Name of the method : getCreditPoints

Return Type : int

Modifier : public

Task : This method must return creditPoints

Name of the method : toString, Override it,

Return type : String

Task : return only customerName from this.

Create another class called CardType. Define the following for the class

Attributes :

customer : Customer, private

cardType : String, private

Constructor :

parameterizedConstructor: for customer and cardType attributes in that order

Methods :

Name of the method : toString Override this.

Return type : String

Modifier : public

Task : Return the string in the following format.

The Customer 'Rajeev' Is Eligible For 'Gold' Card.

Create One more class by name CardsOnOffer and define the following for the class.

Method :

Name Of the method : getOfferedCard

Return type : CardType

Modifiers: public,static

Arguments: Customer object

Task : Create and return a CardType object after logically finding cardType from

creditPoints as per the below rules.

creditPoints		cardType
100 - 500	-	Silver
501 - 1000	-	Gold
1000 >	-	Platinum
< 100	-	EMI

Create an ELC class which contains Main method to test the working of the above application.

4 files :

Customer.java

```
package com.ravi.scenario;

public class Customer
{
	private String customerName;
	private int creditPoints;

	public Customer(String customerName, int creditPoints)
	{
		super();
		this.customerName = customerName;
		this.creditPoints = creditPoints;
	}


	public int getCreditPoints()
	{
		return this.creditPoints;
   }


	@Override
	public String toString()
	{
		return this.customerName;
	}


}
```

CardType.java

```
package com.ravi.scenario;

public class CardType
{
	private Customer customer;
	private String cardType;

	public CardType(Customer customer, String cardType)
	{
		super();
		this.customer = customer;
		this.cardType = cardType;
	}

	@Override
	public String toString()
	{

		return "The Customer '"+ this.customer+"' Is Eligible For '"+this.cardType+"'
Card";
	}


}
```

CardsOnOffer.java

```
package com.ravi.scenario;

public class CardsOnOffer
{
  public static CardType getOfferedCard(Customer cust) //cust = c1
  {
	int creditPoint = cust.getCreditPoints();

	if(creditPoint >=100 && creditPoint <=500)
	{
		return new CardType(cust, "Silver");
	}
	else if(creditPoint >500 && creditPoint <=1000)
	{
```

```
                return new CardType(cust, "Gold");
        }
        else if(creditPoint > 1000)
        {
                return new CardType(cust, "Platinum");
        }
        else
        {
                return new CardType(cust, "EMI");
        }

 }
}
```

CreditCardType.java

```
package com.ravi.scenario;

import java.util.Scanner;

public class CreditCardType
{
        public static void main(String[] args)
        {
                Scanner sc = new Scanner(System.in);
                System.out.print("Enter Customer Name :");
                String name = sc.nextLine();
                System.out.print("Enter Customer Credit Point :");
                int creditPoint = sc.nextInt();

                Customer c1 = new Customer(name, creditPoint);

                CardType offeredCard = CardsOnOffer.getOfferedCard(c1);
                System.out.println(offeredCard);

                sc.close();
        }

}
```

==========

Constructor Overloading :

We can write multiple constructors in a single class but arguments must be different.

Example :

```
public class Calculate
{
   public Calculate(int x, int y)
   {
   }

   public Calculate(int x, int y, int z)  //Constructor Overloading
   {
   }
}
```

In order to call all the constructors in the same class, we need not to create multiple objects, with single object we can call all the constructors with the help of this() [this of].

**THE FIRST LINE OF ANY CONSTRUCTOR IS RESERVED EITHER FOR this() OR super() [this of OR super of] that means the first line of any constructor is used to call another constructor either from same class (this()) OR from super class (super()).

this() [this of] must be the first line of any constructor, we can't write in the 2nd line.

this() we can use only for constructor calling but not method calling.

Calculate.java

```
package com.ravi.constructor_overloading;

public class Calculate
{
  public Calculate(int x) //2
  {
            this("Data", "Base");
            System.out.println("Square of "+x+" is :"+(x*x));
  }

  public Calculate(int x, int y)  //1
  {
            this(5);
            System.out.println("Sum of "+x+" and "+y+" is :"+(x+y));
  }
```

```
  public Calculate(String x, String y) //3
  {
            System.out.println("String after concatenation :"+(x+y));
  }

}
```

ConstructorOverloading.java

```
package com.ravi.constructor_overloading;

public class ConstructorOverloading {

        public static void main(String[] args)
        {
                new Calculate(10,20); //Anonymous OR Nameless object

        }

}
```

27-02-2025

Modifiers on Constructor :

Constructor can accept all types of access modifiers which are apllicable for accessibility level like private, default, protected and public.

Example :

```
public class Test   //Valid    private  protected  public
{                   //Invalid  static final abstract synchronized
   public Test()
   {

   }
}
```

Note : We can't apply static, final, abstract and synchronized modifier on constructor.

IQ :

Why we should declare a constructor with private Access modifier ?

We can declare a constructor with private access modifier due to the following two reasons :

1) If we want to declare only static variables and static methods inside a class then we should decalre constructor with private access modifier [Object is not required]

Example : java.util.Arrays and java.lang.Math class has private constructor because It contains only static methods and will not allow anyone to create the object.

2) If we want to develop singleton class, for singleton class we should create only one object by the class itself then we should declare the constructor with private access modifier.

```
package com.ravi.constructor;

public class Foo
{
  private Foo() //Private Constructor
  {
            System.out.println("It is a private Constructor");
  }

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

}
```

==========Instance Block OR Instance Initializer OR Non static block :

An instance block is a very special block in java which is automatically executed at the time of creating the Object.

Example :

```
{
   System.out.println("It is a Non static block");
}
```

Every time we will create an Object then instance block will be executed.

NSBDemo1.java

```
package com.ravi.non_static_block;

class Test
{
	{
		System.out.println("NSB");
	}
}

public class NSBDemo1
{
	public static void main(String[] args)
	{
		new Test();
		new Test();

	}

}
```

A non static block is executed before the Constructor body execution.

The first line of any constructor is reserved for this() OR super(). If a contsructor contains super() as a first line of constructor then that constructor 2nd line is reserved for Non static block.

NSBDemo2.java

```
package com.ravi.non_static_block;

class Demo
{
	public Demo()
	{
		System.out.println("No Argument constructor");
	}

	{
		System.out.println("Non static Block");
	}
}
```

```
public class NSBDemo2
{
	public static void main(String[] args)
	{
		new Demo();  new Demo();
	}

}
```

NSBDemo3.java

```
package com.ravi.non_static_block;

class Sample
{
	public Sample()
	{
		super();
		//2nd line is reserved for NSB
		System.out.println("No Argument Constructor");
	}

	public Sample(int x, int y)
	{
	  this();
	  System.out.println("Parameterized Constructor and Sum is :"+(x+y));
	}

	{
		System.out.println("Non static Block");
	}


}

public class NSBDemo3 {

	public static void main(String[] args)
	{
		new Sample(10,20);
	}

}
```

The main purpose of non static block to initialize the non static data member of the class that is the reason It is also known as Instance Initializer, It is also used to provide some common message OR common logic execution for all the Objects.

```
package com.ravi.non_static_block;

class Foo
{
	private int x;

	public Foo()
	{
		System.out.println("x value is :"+x);
	}

	{
		x = 100;
		System.out.println("Object creation is in progress....");
	}
}

public class NSBDemo4 {

	public static void main(String[] args)
	{
		new Foo();
		System.out.println(".....");
		new Foo();

	}

}
```

If our class contains multiple non static blocks then it will be executed according to the order.[Top to bottom]

```
class Test
{
	int x;

	public Test()
```

```
	{
		System.out.println("x value is :"+x);
	}

	{
		x = 100;
	}

	{
		x = 200;
	}

	{
		x = 300;
	}


}

public class NSBDemo5
{
	public static void main(String[] args)
	{
		Test t1 = new Test();
	}
}
```

If a user explicitly writes any non static block inside the constructor body then it will be executed as it is, Here compiler will not perform any action.

```
package com.ravi.instance_initializer;

class Foo
{
	public Foo()
	{
		System.out.println("No Argument Constructor..");

		{
			System.out.println("Non static block..");
		}
	}
}
```

```
public class NSBDemo6
{
	public static void main(String[] args)
	{
		new Foo();

	}

}
```

We cannot write return statement inside a non static block because all the initializers must be executed normally.

```
class Sample
{
	{
		System.out.println("NSB");
		return; //error
	}

}
public class NSBDemo7
{
	public static void main(String[] args)
	{
		new Sample();
	}

}
```

==========

Non static variable initialization order :

The order of non static variable initialization :

1) Default value by using new keyword [init method is working internally]

```
package com.ravi.initialization_order;

class Demo
{

	int x = 100 ;
```

```
}
public class NSVInitializationOrder
{
	public static void main(String[] args)
	{
		Demo d1 = new Demo();

		System.out.println(d1.x);

	}

}
```

2) Variable initialization at the time of NSV declaration OR
Variable initialization inside non static block [Both are having equal priority, variable will initialized based on the order {top to bottom}]
[Remaining part : Illegal Forward reference, JVM Architecture]

Case 1:

```
package com.ravi.initialization_order;

class Demo
{

	int x = 100 ;

	{
		x = 200;
	}

}
public class NSVInitializationOrder
{
	public static void main(String[] args)
	{
		Demo d1 = new Demo();

		System.out.println(d1.x);

	}
```

```
}
```

Case 2:

```
package com.ravi.initialization_order;

class Demo
{
	{
		x = 200;
	}

	int x = 100 ;
}
public class NSVInitializationOrder
{
	public static void main(String[] args)
	{
		Demo d1 = new Demo();

		System.out.println(d1.x);
	}

}
```

3) We can initialize the NSV in the Constructor body.

```
package com.ravi.initialization_order;

class Demo
{
	int x = 100 ;	// Demo d1 = new Demo();

	{
		x = 200;
	}

	public Demo()
	{
		super();
		x = 300;
```

```
	}
}
public class NSVInitializationOrder
{
	public static void main(String[] args)
	{
		Demo d1 = new Demo();

		System.out.println(d1.x);
	}

}
```

5) We can initialize through Methods but it is not a recommended way because Object creation is already completed.

```
package com.ravi.initialization_order;

class Demo
{
	int x = 100 ;	// Demo d1 = new Demo();

	{
		x = 200;
	}

	public Demo()
	{
		super();
		x = 300;
	}

	public void setData()
	{
		x = 400;
	}

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

```
		Demo d1 = new Demo();
		d1.setData();
		System.out.println(d1.x);
	}

}
```
==========

What is a blank final field in java ?

A final field must be initialized by user only once.

If a final non static variable is not initialized at the time of declaration then it is called Blank final field.

Example :

```
class Student
{
  final int x;  //Blank final field
}
```

A blank final field cannot be initialized inside the default constructor.

```
class Test
{
	final int x; //Blank final field  [error]
}
class BlankFinalFieldDemo1
{
	public static void main(String[] args)
	{
		Test t1 = new Test();
		System.out.println(t1.x);
	}
}
```

A blank final field cannot be initialized inside the method body.

```
class Test
{
	final int x;
```

```
	public void set()
	{
		x = 100;
	}


}
class BlankFinalFieldDemo1
{
	public static void main(String[] args)
	{
		Test t1 = new Test();
		t1.set();
		System.out.println(t1.x);
	}
}
```

A blank field must be initialized by the developer expilicitly in the following two places only [Till the Object creation]

a) Inside a non static Block OR
b) Inside the constructor body

Note : A blank final field must be initialized by the
developer till Object creation.

```
class Test
{
	final int x;
	/*
	{
		x = 200;
	}
	*/
	public Test()
	{
		x = 300;
	}

}
class BlankFinalFieldDemo1
{
```

```
	public static void main(String[] args)
	{
		Test t1 = new Test();
		System.out.println(t1.x);
	}
}
```

A blank final field also have default value that means all the class level variables are having default value even the variable is declared as final.

```
class Test
{
	final int x;

	{
		print();
		x = 100;
	}

	public void print()
	{
		System.out.println("Default value :"+x);
	}

}
class BlankFinalFieldDemo1
{
	public static void main(String[] args)
	{
		Test t1 = new Test();
		System.out.println("User Value is :"+t1.x);
	}
}
```

A blank final filed must be initialized in all the constructors available in the class.

```
class Student
{
  final String name;

  public Student()
  {
```

```
            name = "Scott";
  }

  public Student(String name)
  {
            this.name = name;
  }
}

class BlankFinalFieldDemo1
{
          public static void main(String[] args)
          {
                    Student s1 = new Student();
                    System.out.println(s1.name);

                    Student s2 = new Student("Smith");
                    System.out.println(s2.name);
          }
}
-------------------------------------------------------------
class Student
{
  final String name;

          {
                    name = "James";
          }

  public Student()
  {

  }

  public Student(String name)
  {

  }
}

class BlankFinalFieldDemo1
{
          public static void main(String[] args)
```

```
	{
		Student s1 = new Student();
		System.out.println(s1.name);

		Student s2 = new Student("Smith");
		System.out.println(s2.name);
	}
}
```

==========

Relationship between the classes :
