

# **Java & JEE Training**

**Day 8 – OOP with Java Contd.**

**MindsMapped Consulting**

# Review of last session...

---

- ✓ Object Oriented Concepts
- ✓ Introduction to OO Analysis and Design

# Agenda...

---

- ✓ Deep dive into coding OOP with Java... with practical examples.
  - ✓ How to create a class
  - ✓ How to create objects
  - ✓ How to create instance variables
  - ✓ How to create class variables
  - ✓ Constructors

# **Java & JEE Training**

**Object Oriented Programming with Java  
- Basic Class Demo**

**MindsMapped Consulting**

# Naming Conventions

---

Name	Convention
class name	should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.
interface name	should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.
method name	should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc.
variable name	should start with lowercase letter e.g. firstName, orderNumber etc.
package name	should be in lowercase letter e.g. java, lang, sql, util etc.
constants name	should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.

# Object and Class in Java - Demo

---

## A basic example of a class:

```
class Student1{  
    int id;//data member (also instance variable)  
    String name;//data member(also instance variable)  
  
    public static void main(String args[]){  
        Student1 s1=new Student1();//creating an object of Student  
        System.out.println(s1.id);  
        System.out.println(s1.name);  
    }  
}
```

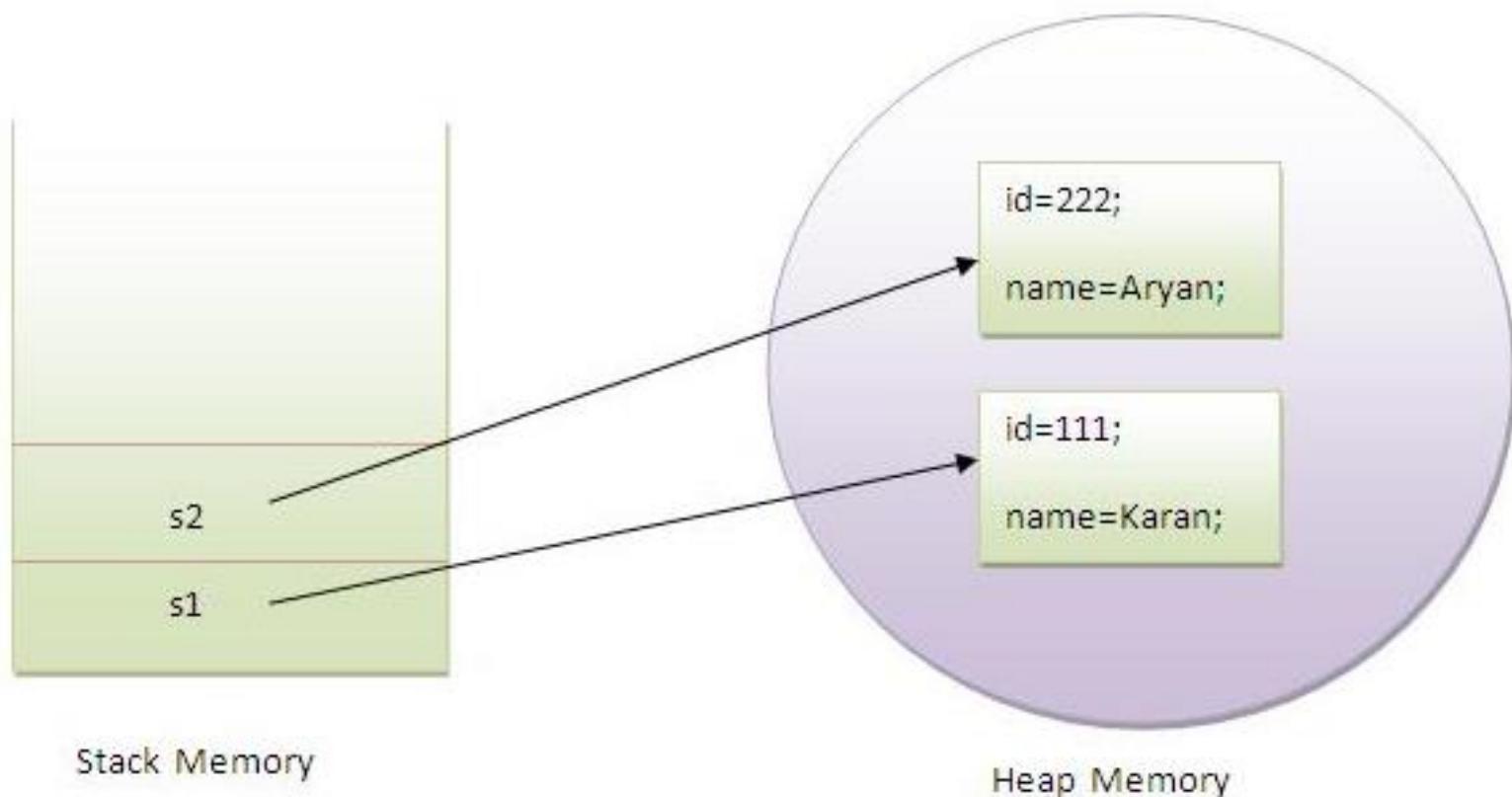
# Another Example of Objects and Classes in Java

---

```
class Student2{  
    int rollno;  
    String name;  
  
    void insertRecord(int r, String n){ //method  
        rollno=r;  
        name=n;  
    }  
  
    void displayInformation(){System.out.println(rollno+" "+name);} //method  
  
    public static void main(String args[]){  
        Student2 s1=new Student2();  
        Student2 s2=new Student2();  
  
        s1.insertRecord(111,"Karan");  
        s2.insertRecord(222,"Aryan");  
  
        s1.displayInformation();  
        s2.displayInformation();  
    }  
}
```

# Memory Allocation

---



# **Java & JEE Training**

**Method Overloading**

**MindsMapped Consulting**

# Method Overloading

---

- If a class have multiple methods by same name but different parameters.
- **Advantage:** Increases readability of the program.
- **Two ways:**
  - By changing number of arguments
  - By changing the data type

# Method Overloading: Changing number of arguments

---

```
class Calculation{  
    void sum(int a,int b){System.out.println(a+b);}  
    void sum(int a,int b,int c){System.out.println(a+b+c);}  
  
    public static void main(String args[]){  
        Calculation obj=new Calculation();  
        obj.sum(10,10,10);  
        obj.sum(20,20);  
  
    }  
}
```

## Method overloading: Changing data type of argument

---

```
class Calculation2{  
    void sum(int a,int b){System.out.println(a+b);}  
    void sum(double a,double b){System.out.println(a+b);}  
  
    public static void main(String args[]){  
        Calculation2 obj=new Calculation2();  
        obj.sum(10.5,10.5);  
        obj.sum(20,20);  
  
    }  
}
```

## **Method overloading: Can it be done by changing return type of methods?**

---

```
class Calculation3{  
    int sum(int a,int b){System.out.println(a+b) ;}  
    double sum(int a,int b){System.out.println(a+b) ;}  
  
    public static void main(String args[]){  
        Calculation3 obj=new Calculation3();  
        int result=obj.sum(20,20);  
        /* Compile Time Error; Here how can java determine which  
           sum() method should be called */  
    }  
}  
//
```

# Method Overloading: Can we overload main() method?

---

```
class Overloading1{  
    public static void main(int a){  
        System.out.println(a);  
    }  
  
    public static void main(String args[]){  
        System.out.println("main() method invoked");  
        main(10);  
    }  
}
```

# Method Overloading and Type Promotion

---

One type is promoted to another implicitly **if no matching datatype is found**

```
class OverloadingCalculation1{
    void sum(int a,long b){System.out.println(a+b) ;}
    void sum(int a,int b,int c){System.out.println(a+b+c) ;}

    public static void main(String args[]){
        OverloadingCalculation1 obj=new OverloadingCalculation1();
        obj.sum(20,20); //now second int literal will be promoted to long
        obj.sum(20,20,20);

    }
}
```

## Method overloading: Ambiguous code

---

```
class OverloadingCalculation3{
    void sum(int a,long b){System.out.println("a method invoked");}
    void sum(long a,int b){System.out.println("b method invoked");}
}

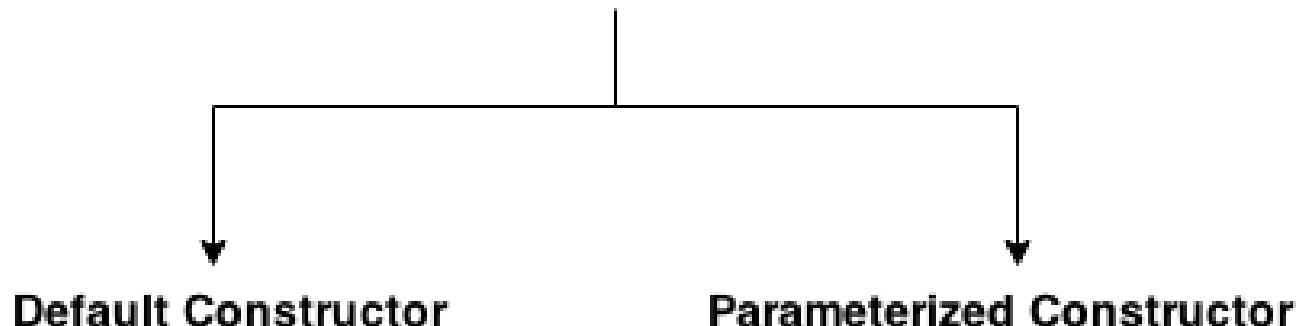
public static void main(String args[]){
OverloadingCalculation3 obj=new OverloadingCalculation3();
obj.sum(20,20); //now ambiguity; Compile time error.
}
}
```

# Constructors

---

- **Constructor in java** is a *special type of method* that is used to initialize the object.
- Java constructor is *invoked at the time of object creation*.
- Two rules:
  - Constructor name must be same as its class name
  - Constructor must have no explicit return type
  - **If there is no constructor in a class, compiler automatically creates a default constructor.**

## Types of Java Constructor



# **Java & JEE Training**

**Constructors**

**MindsMapped Consulting**

## Example of default constructor that displays the default values

---

```
class Student3{  
    int id;  
    String name;  
  
    void display(){  
        System.out.println(id+" "+name);  
    }  
  
    public static void main(String args[]){  
        Student3 s1=new Student3();  
        Student3 s2=new Student3();  
        s1.display();  
        s2.display();  
    }  
}
```

## Example of parameterized constructor

---

```
class Student4{  
    int id;  
    String name;  
  
    Student4(int i, String n){ //Parameterized Constructor  
        id = i;  
        name = n;  
    }  
    void display(){System.out.println(id+" "+name);}  
  
    public static void main(String args[]){  
        Student4 s1 = new Student4(111,"Karan");  
        Student4 s2 = new Student4(222,"Aryan");  
        s1.display();  
        s2.display();  
    }  
}
```

# Example of Constructor Overloading

---

```
class Student5{  
    int id;  
    String name;  
    int age;  
  
    Student5(int i,String n){  
        id = i;  
        name = n;  
    }  
  
    Student5(int i,String n,int a){  
        id = i;  
        name = n;  
        age=a;  
    }  
    void display(){System.out.println(id+" "+name+" "+age);}  
  
    public static void main(String args[]){  
        Student5 s1 = new Student5(111,"Karan");  
        Student5 s2 = new Student5(222,"Aryan",25);  
        s1.display();  
        s2.display();  
    }  
}
```

# Constructor vs Method

---

Java Constructor	Java Method
Constructor is used to initialize the state of an object.	Method is used to expose behaviour of an object.
Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.
The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in any case.
Constructor name must be same as the class name.	Method name may or may not be same as class name.

# Constructor Example: Used for cloning

---

```
class Student6{
    int id;
    String name;
    Student6(int i, String n){
        id = i;
        name = n;
    }

    Student6(Student6 s){
        id = s.id;
        name = s.name;
    }
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
        Student6 s1 = new Student6(111,"Karan");
        Student6 s2 = new Student6(s1);
        s1.display();
        s2.display();
    }
}
```

## Point to ponder...

---

Does constructor return any value?

# **Java & JEE Training**

**"Static"**

**MindsMapped Consulting**

## Java “Static”

---

- The **static keyword** in java is used for memory management mainly.
- We can apply java static keyword with variables, methods, blocks and nested class.
- The static keyword belongs to the class than instance of the class.
- The static can be:
  - variable (also known as class variable)
  - method (also known as class method)
  - block
  - nested class

# Java Static Variable

---

- If you declare any variable as static, it is known static variable.
  - The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees, college name of students etc.
  - The static variable gets memory only once in class area at the time of class loading.
- 
- Advantage:  
It makes your program **memory efficient**

## Example: Problem without static variable

---

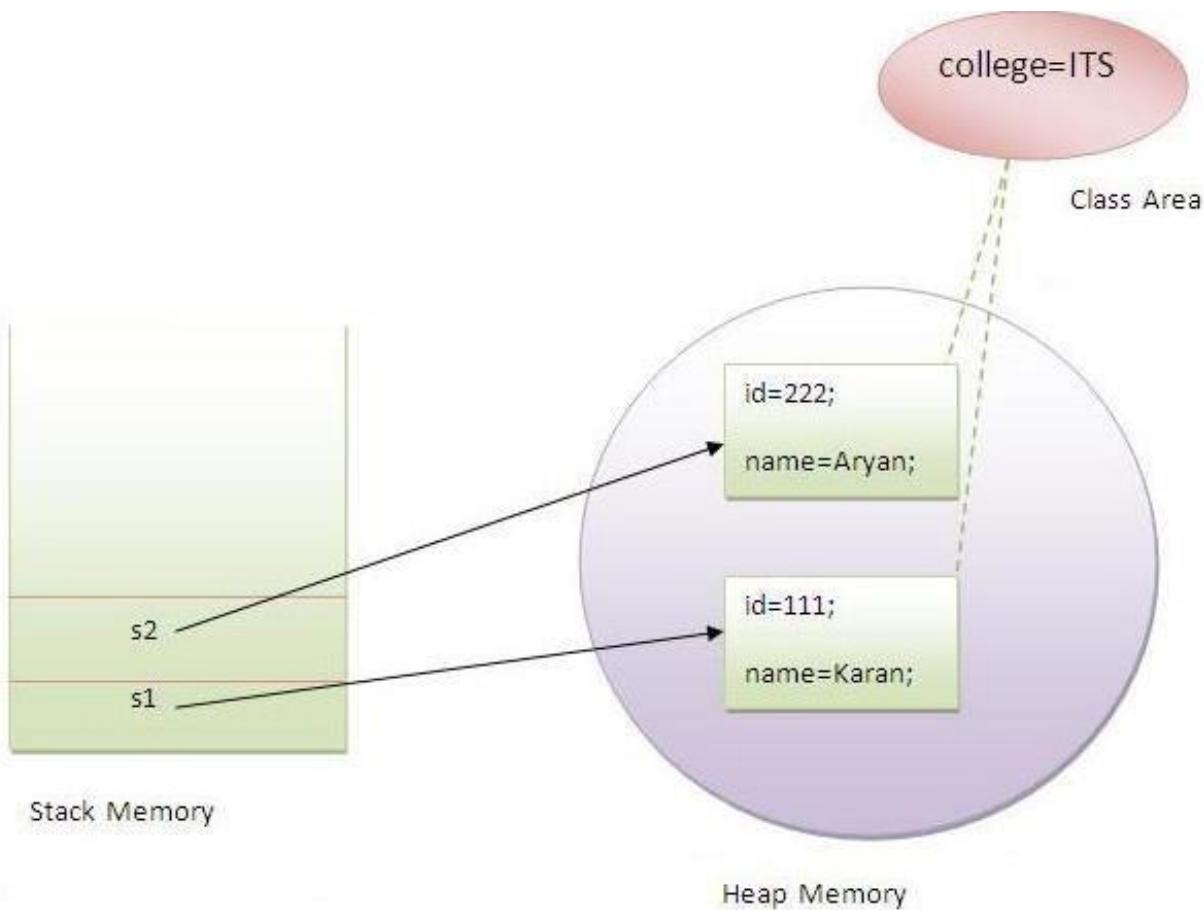
```
class Student{  
    int rollno;  
    String name;  
    String college="ITS";  
}
```

## Example: Static Variable solution

---

```
class Student8{  
    int rollno;  
    String name;  
    static String college ="ITS";  
  
    Student8(int r, String n){  
        rollno = r;  
        name = n;  
    }  
    void display (){System.out.println(rollno+" "+name+" "+college);}  
  
    public static void main(String args[]){  
        Student8 s1 = new Student8(111,"Karan");  
        Student8 s2 = new Student8(222,"Aryan");  
  
        s1.display();  
        s2.display();  
    }  
}
```

# Static Variable Memory Allocation



## Example: Counter with static variable

---

```
class Counter2{  
    static int count=0;//will get memory only once and retain its value  
  
    Counter2() {  
        count++;  
        System.out.println(count);  
    }  
  
    public static void main(String args[]) {  
  
        Counter2 c1=new Counter2();  
        Counter2 c2=new Counter2();  
        Counter2 c3=new Counter2();  
    }  
}
```

## Java Static Method

---

- A static method belongs to the class rather than object of a class.
  - A static method can be invoked without the need for creating an instance of a class.
  - static method can access static data member and can change the value of it.
- 
- There are two main restrictions for the static method. They are:
    - The static method can not use non static data member or call non-static method directly.
    - this and super cannot be used in static context.

# Static Method Example

---

```
class Student9{
    int rollno;
    String name;
    static String college = "ITS";

    static void change(){
        college = "BBDIT";
    }
    Student9(int r, String n){
        rollno = r;
        name = n;
    }
    void display (){System.out.println(rollno+" "+name+" "+college);}
    public static void main(String args[]){
        Student9.change();

        Student9 s1 = new Student9 (111,"Karan");
        Student9 s2 = new Student9 (222,"Aryan");
        Student9 s3 = new Student9 (333,"Sonoo");

        s1.display();
        s2.display();
        s3.display();
    }
}
```

## Example of static method that performs normal calculation

---

```
//Program to get cube of a given number by static method
```

```
class Calculate{  
    static int cube(int x){  
        return x*x*x;  
    }  
  
    public static void main(String args[]){  
        int result=Calculate(cube(5);  
        System.out.println(result);  
    }  
}
```

## What is the output?

---

```
class A{
    int a=40;//non static

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

## Point to ponder

---

- Why is main() method static?

## Java Static Block

---

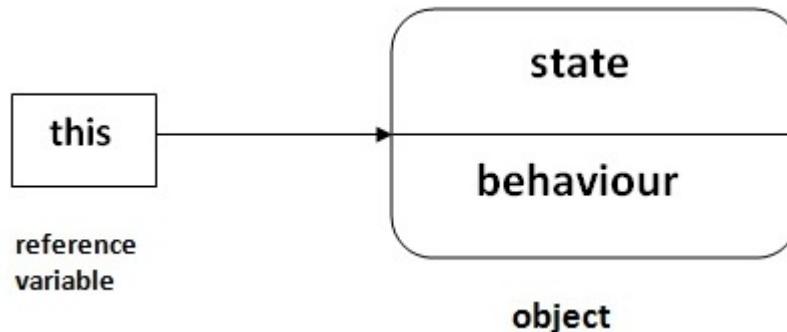
- Is used to initialize the static data member.
- It is executed before main method at the time of classloading.

```
class A2{  
    static{System.out.println("static block is invoked");}  
    public static void main(String args[]){  
        System.out.println("Hello main");  
    }  
}
```

# “this” keyword

---

- In java, this is a **reference variable** that refers to the current object.
- Uses:
  - this keyword can be used to refer current class instance variable.
  - this() can be used to invoke current class constructor.
  - this keyword can be used to invoke current class method (implicitly)
  - this can be passed as an argument in the method call.
  - this can be passed as argument in the constructor call.
  - this keyword can also be used to return the current class instance.



## What's the problem with the below code?

---

```
class Student10{  
    int id;  
    String name;  
  
    Student10(int id, String name) {  
        id = id;  
        name = name;  
    }  
    void display() { System.out.println(id+" "+name); }  
  
    public static void main(String args[]) {  
        Student10 s1 = new Student10(111, "Karan");  
        Student10 s2 = new Student10(321, "Aryan");  
        s1.display();  
        s2.display();  
    }  
}
```

## Solution:

---

```
//example of this keyword
class Student11{
    int id;
    String name;

    Student11(int id, String name) {
        this.id = id;
        this.name = name;
    }

    void display() {System.out.println(id+" "+name);}

    public static void main(String args[]) {
        Student11 s1 = new Student11(111, "Karan");
        Student11 s2 = new Student11(222, "Aryan");
        s1.display();
        s2.display();
    }
}
```

Note: If local variables(formal arguments) and instance variables are different, there is no need to use this keyword like in the following program:

## this() for invoking current class constructor: Constructor Chaining

---

```
//Program of this() constructor call (constructor chaining)

class Student13{
    int id;
    String name;
    Student13(){System.out.println("default constructor is invoked");}

    Student13(int id,String name){
        this(); //it is used to invoked current class constructor.
        this.id = id;
        this.name = name;
    }
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
        Student13 e1 = new Student13(111,"karan");
        Student13 e2 = new Student13(222,"Aryan");
        e1.display();
        e2.display();
    }
}
```

Rule: Call to this() must be the first statement in constructor.

## this keyword can be used to invoke current class method (explicitly)

---

```
class S{
    void m() {
        System.out.println("method is invoked");
    }
    void n() {
        this.m(); //no need because compiler does it for you.
    }
    void p() {
        n(); //compiler will add this to invoke n() method as this.n()
    }
    public static void main(String args[]) {
        S s1 = new S();
        s1.p();
    }
}
```

this keyword can be passed as an argument in the method.

---

```
class S2{  
    void m(S2 obj){  
        System.out.println("method is invoked");  
    }  
    void p(){  
        m(this);  
    }  
  
    public static void main(String args[]){  
        S2 s1 = new S2();  
        s1.p();  
    }  
}
```

## What is the output? What does the below code do?

---

```
class A5{  
    void m(){  
        System.out.println(this); //prints same reference ID  
    }  
  
    public static void main(String args[]){  
        A5 obj=new A5();  
        System.out.println(obj); //prints the reference ID  
  
        obj.m();  
    }  
}
```