B.Sc. Computer Science

Unit III

Class in Java (जावा में क्लास)

Java एक **Object-Oriented Programming (OOP)** language है, जिसका आधार **Class** और **Object** है।

एक **Class** को आप एक **Blueprint** या **Template** की तरह समझ सकते हैं। जैसे घर बनाने से पहले उसका नक्शा (Map) तैयार किया जाता है, वैसे ही Object बनाने से पहले उसकी Class बनाई जाती है। यह बताती है कि उस Object के पास क्या-क्या जानकारी (Variables) होगी और वह क्या-क्या काम (Methods) करेगा।

---

1. Defining a Class (क्लास को डिफाइन करना)

Class को define करने के लिए class keyword का इस्तेमाल होता है।

Syntax:

Java

```
class ClassName {
    // Variables (Data Members)
    // Methods (Behavior)
}
```

---

2. Adding Variables (Data Members)

Variables क्लास की **Properties** या **State** को दर्शाते हैं। इन्हें **Fields** या **Member Variables** भी कहा जाता है।

```
class Car {
    String model; // Variable to store car name
    int speed;    // Variable to store car speed
}
```

3. Adding Methods (Member Functions)

Methods क्लास के **Behaviour** को दर्शाते हैं। यह बताते हैं कि Object क्या काम करेगा।

```
class Car {
    String model;
    int speed;

    // Method to display car details
    void displayDetails() {
        System.out.println("Model: " + model);
        System.out.println("Speed: " + speed + " km/h");
    }
}
```

---

4. Creating and Accessing Class Object

Class सिर्फ एक नक्शा है, यह memory में जगह नहीं लेती। जब हम new keyword का उपयोग करके **Object** बनाते हैं, तब memory allocate होती है। Object के ज़रिए variables और methods को access करने के लिए **Dot (.) operator** का इस्तेमाल किया जाता है।

Code Example:

```
class Student {
    // Class Members: Variables
    int rollNo;
    String name;

    // Class Members: Method
    void study() {
        System.out.println(name + " is studying...");
    }

    void showData() {
        System.out.println("Roll No: " + rollNo);
        System.out.println("Name: " + name);
    }
}

public class Main {
    public static void main(String[] args) {
        // Accessing Class: Creating an Object
        Student s1 = new Student();

        // Accessing Variables using dot operator
        s1.rollNo = 101;
        s1.name = "Rahul";

        // Accessing Methods using dot operator
```

```
        s1.showData();
        s1.study();
    }
}
```

Key Points to Remember (महत्वपूर्ण बातें)

1. **Template vs Reality:** Class सिर्फ एक design है, जबकि Object उस design की real entity है जो memory में रहती है।
2. **Accessing:** Class के अंदर के members (variables/methods) को सीधे use नहीं किया जा सकता, उन्हें Object के नाम के साथ . लगाकर access करना पड़ता है (जैसे: s1.name).
3. **Multiple Objects:** आप एक ही Class से हज़ारों अलग-अलग Objects बना सकते हैं (जैसे s1, s2, s3), और हर Object का अपना अलग data होगा।
4. **Naming Convention:** Java में Class का नाम हमेशा **Capital letter** से शुरू होना चाहिए (जैसे Student, Car).

Constructor in Java (जावा में कंस्ट्रक्टर)

Constructor एक special method होता है जिसका इस्तेमाल **Objects को initialize** करने के लिए किया जाता है। जब आप new keyword के साथ किसी class का object बनाते हैं, तो Constructor अपने आप (automatically) call हो जाता है।

इसका मुख्य काम object के variables को शुरुआती values (initial values) देना होता है।

Key Rules for Creating a Constructor (नियम)

1. **Same Name:** Constructor का नाम वही होना चाहिए जो उसकी **Class** का नाम है।
2. **No Return Type:** इसमें कोई return type नहीं होता (यहाँ तक कि void भी नहीं)।
3. **Automatic Call:** यह केवल object creation के समय ही call होता है।

Types of Constructors

Java में मुख्य रूप से दो तरह के constructors होते हैं:

1. Default Constructor (No-Argument)

अगर आप अपनी class में कोई constructor नहीं बनाते, तो Java compiler अपने आप एक खाली constructor बना देता है। यह variables को default values (जैसे 0, null) देता है।

2. Parameterized Constructor

यह constructor तब इस्तेमाल होता है जब हम object बनाते समय ही अपनी पसंद की values पास करना चाहते हैं।

Code Example: Working with Constructors

```
class Employee {
    int id;
    String name;

    // 1. Default Constructor
    Employee() {
        System.out.println("Default Constructor called!");
```

```
        id = 0;
        name = "Unknown";
    }

    // 2. Parameterized Constructor
    Employee(int i, String n) {
        System.out.println("Parameterized Constructor called!");
        id = i;
        name = n;
    }

    void display() {
        System.out.println("ID: " + id + ", Name: " + name);
    }
}

public class Main {
    public static void main(String[] args) {
        // Calling Default Constructor
        Employee emp1 = new Employee();
        emp1.display();

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

        // Calling Parameterized Constructor
        Employee emp2 = new Employee(101, "Rahul");
        emp2.display();
    }
}
```

Output

```
Default Constructor called!
ID: 0, Name: Unknown
-------------------------
Parameterized Constructor called!
ID: 101, Name: Rahul
```

Constructor Overloading

जैसे Methods को overload किया जा सकता है, वैसे ही एक class में एक से ज़्यादा constructors हो सकते हैं, बस उनके parameters अलग-अलग होने चाहिए। इसे **Constructor Overloading** कहते हैं।

Difference: Method vs Constructor

Feature	Constructor	Method
Purpose	Object को initialize करना।	Object के behavior/logic को लिखना।
Name	Class के नाम जैसा ही होना चाहिए।	कोई भी नाम हो सकता है।
Return Type	कोई return type नहीं होता।	Return type (या void) होना ज़रूरी है।

Invocation Object बनते ही अपने आप call होता है। इसे manually call करना पड़ता है।

Key Points to Remember (महत्वपूर्ण बातें)

1. **The this Keyword:** अगर constructor के parameters और class variables का नाम same है, तो this.name = name; का इस्तेमाल किया जाता है।
2. **Explicit Call:** अगर आपने कम से कम एक Parameterized constructor बना दिया, तो Java अपना default constructor देना बंद कर देता है। फिर आपको खुद ही default constructor लिखना पड़ेगा।
3. **Inheritance:** Constructors को static, abstract या final नहीं बनाया जा सकता।

Array (एरे)

Array एक समान प्रकार के डेटा (similar data types) का एक समूह होता है। यह एक ही variable के नाम के तहत कई values को store करने के लिए इस्तेमाल किया जाता है।

सरल शब्दों में: अगर आपको 50 छात्रों के Roll Numbers store करने हैं, तो आपको 50 अलग-अलग variables (r1, r2, r3...) बनाने की ज़रूरत नहीं है। आप सिर्फ एक **Array** बना सकते हैं।

Key Features of Arrays (मुख्य विशेषताएं)

1. **Fixed Size:** एक बार Array बनाने के बाद उसका size बदला नहीं जा सकता।
2. **Index-based:** Array की पहली value हमेशा **index 0** पर store होती है, दूसरी **index 1** पर, और इसी तरह।
3. **Contiguous Memory:** Array के सभी elements memory में एक साथ (लगातार) store होते हैं।

---

1. Declaring and Creating an Array

Java में Array को new keyword के ज़रिए memory allocate की जाती है।

Syntax: dataType[] arrayName = new dataType[size];

Example: int[] numbers = new int[5]; // Stores 5 integers

---

2. Initializing an Array

आप Array को declare करते समय भी values दे सकते हैं:

Java
```
int[] marks = {90, 85, 70, 95, 80};
```

Code Example: Working with Array

This program stores names in an array and prints them using a loop.

```java
public class ArrayExample {
    public static void main(String[] args) {
        // Declaration and instantiation
        String[] students = new String[3];

        // Adding values (Initialization)
        students[0] = "Rahul";
        students[1] = "Priya";
        students[2] = "Amit";

        // Accessing values using a loop
        System.out.println("List of Students:");
        for (int i = 0; i < students.length; i++) {
            // Using .length property to get array size
            System.out.println("Index " + i + ": " + students[i]);
        }
    }
}
```

Output

```
List of Students:
Index 0: Rahul
Index 1: Priya
Index 2: Amit
```

Types of Arrays in Java

1. **Single-Dimensional Array:** एक सीधी list की तरह (ऊपर दिया गया example)।
2. **Multi-Dimensional Array:** इसे "Array of Arrays" भी कहते हैं। यह data को rows और columns (Table format) में store करता है।

Example of 2D Array:

Java
```
int[][] matrix = {
  {1, 2},
  {3, 4}
};
// matrix[0][0] is 1, matrix[1][1] is 4
```

Key Points to Remember (महत्वपूर्ण बातें)

1. **length property:** Array का size पता करने के लिए arrayName.length का उपयोग किया जाता है।

2. **Default Values:** अगर आप array बनाकर उसे values नहीं देते, तो Java उसे default values दे देता है (जैसे int के लिए 0, Object के लिए null)।
3. **Exception:** अगर आप array के size से बाहर की index access करने की कोशिश करेंगे (जैसे size 5 है और आप index 10 मांग रहे हैं), तो Java **ArrayIndexOutOfBoundsException** देगा।

String in Java (जावा में स्ट्रिंग)

Java में **String** characters का एक sequence होता है। C या C++ के विपरीत, Java में String कोई primitive data type नहीं है, बल्कि यह एक **Object** है जो java.lang.String class से आता है।

Key Features of Strings (मुख्य विशेषताएं)

- **Immutable:** String एक बार बन जाए तो उसे बदला नहीं जा सकता। अगर आप उसे modify करने की कोशिश करेंगे, तो Java memory में एक नया String object बना देगा।
- **Index-based:** Array की तरह String के characters भी **index 0** से शुरू होते हैं।
- **String Pool:** Java memory बचाने के लिए **String Constant Pool (SCP)** का इस्तेमाल करता है जहाँ duplicate strings को दोबारा नहीं बनाया जाता।

---

1. Creating a String (स्ट्रिंग बनाने के तरीके)

Java में String बनाने के दो मुख्य तरीके हैं:

- By String Literal: यह सबसे आसान तरीका है। इसमें Java पहले pool में चेक करता है, अगर string वहां है तो नया object नहीं बनाता।

  String s = "Hello";

- By new Keyword: यह तरीका हमेशा Heap memory में एक नया object बनाता है, चाहे वह pool में पहले से मौजूद हो या नहीं।

  String s = new String("Hello");

---

2. Useful String Methods (ज़रूरी मेथड्स)

Java की String class हमें data manipulate करने के लिए कई built-in methods देती है:

Method	Description	Example (String s = "Java")
length()	String की लंबाई बताता है।	s.length() -> 4
charAt(i)	किसी index का character देता है।	s.charAt(0) -> 'J'

toUpperCase()	Capital letters में बदलता है।	s.toUpperCase() -> "JAVA"
equals()	Content compare करता है।	s.equals("Java") -> true
substring(i, j)	String का एक हिस्सा काटता है।	s.substring(0, 2) -> "Ja"

Code Example: Working with String

This program demonstrates string creation and various common string operations.

```
public class StringExample {
    public static void main(String[] args) {
        // Creating strings using literals
        String firstName = "Rahul";
        String lastName = "Sharma";

        // Concatenation (Joining strings)
        String fullName = firstName + " " + lastName;
        System.out.println("Full Name: " + fullName);

        // Finding length of string
        System.out.println("Length of Name: " + fullName.length());

        // Comparing two strings
        String s1 = "JAVA";
        String s2 = "java";

        System.out.println("Is s1 same as s2? " + s1.equals(s2));
      System.out.println("Is s1 same as s2 (ignore case)? " + s1.equalsIgnoreCase(s2));

        // Getting a specific character
        char firstChar = firstName.charAt(0);
        System.out.println("First character of Rahul: " + firstChar);
    }
}
```

Output

```
Full Name: Rahul Sharma
Length of Name: 12
Is s1 same as s2? false
Is s1 same as s2 (ignore case)? true
First character of Rahul: R
```

Key Points to Remember (महत्वपूर्ण बातें)

1. **length() vs length:** Array का size पता करने के लिए .length (बिना ब्रैकेट) यूज़ होता है, लेकिन String के लिए .length() method का यूज़ होता है।
2. **== vs .equals():** == memory address check करता है, जबकि .equals() String के अंदर लिखे हुए शब्दों (content) को चेक करता है। हमेशा content चेक करने के लिए .equals() यूज़ करें।
3. **Null String:** अगर आप String s; लिखते हैं और उसे initialize नहीं करते, तो उसकी default value null होती है।

Inheritance in Java (इनहेरिटेंस)

Inheritance OOPs का एक concept है जिसमें एक class (Child) दूसरी class (Parent) की properties और methods को हासिल (acquire) कर लेती है। इसका सबसे बड़ा फायदा **Code Reusability** है।

इसे **Parent-Child relationship** या **IS-A relationship** भी कहा जाता है।

Key Terms (मुख्य शब्द)

- **Super Class (Parent):** वह class जिसके features inherit किए जाते हैं।
- **Sub Class (Child):** वह class जो features को inherit करती है।
- **`extends` keyword:** यह keyword inheritance लागू करने के लिए इस्तेमाल होता है।

Types of Inheritance in Java (इनहेरिटेंस के प्रकार)

Types of Inheritance in Java

Java में मुख्य रूप से ये types होते हैं:

1. **Single Inheritance:** एक Parent और एक Child।
2. **Multilevel Inheritance:** एक Parent, उसका Child, और फिर उस Child का भी एक Child (जैसे: दादाजी -> पिता -> बेटा)।
3. **Hierarchical Inheritance:** एक Parent और उसके कई सारे Children।

Note: Java में **Multiple Inheritance** (दो Parents की एक संतान) classes के ज़रिए support नहीं होती क्योंकि इससे confusion (Ambiguity) पैदा होता है। इसे 'Interfaces' के ज़रिए किया जाता है।

1. Single Inheritance

जब एक child class सिर्फ एक parent class से inherit होती है।

Example: Class B extends Class A.

```
// Parent Class
class Account {
    double balance = 5000.0;

    void displayBalance() {
        System.out.println("Current Balance: " + balance);
```

```
    }
}

// Child Class inheriting Account
class SavingsAccount extends Account {
    double interestRate = 4.5;

    void showInterest() {
        System.out.println("Interest Rate: " + interestRate + "%");
    }
}

public class Main {
    public static void main(String[] args) {
        SavingsAccount sa = new SavingsAccount();
        sa.displayBalance(); // Inherited from Account
        sa.showInterest();   // Own method
    }
}
```

```
Output

Current Balance: 5000.0
Interest Rate: 4.5%
```

2. Multilevel Inheritance

जब एक inheritance की chain बनती है। (Grandparent -> Parent -> Child)।

- **Example:** Class C extends Class B, and Class B extends Class A.

```
class Vehicle {
    void start() {
        System.out.println("Vehicle is starting...");
    }
}

class Car extends Vehicle {
    void drive() {
        System.out.println("Car is driving on 4 wheels.");
    }
}

class ElectricCar extends Car {
    void charge() {
        System.out.println("Electric car is charging battery...");
    }
}

public class MultilevelDemo {
    public static void main(String[] args) {
        ElectricCar tesla = new ElectricCar();
```

```
Output

Vehicle is starting...
Car is driving on 4 wheels.
Electric car is charging battery...
```

```
        tesla.start();  // From Grandparent (Vehicle)
        tesla.drive();  // From Parent (Car)
        tesla.charge(); // From itself (ElectricCar)
    }
}
```

3. Hierarchical Inheritance

जब एक parent class की एक से ज़्यादा child classes होती हैं।

Example: Class B and Class C both extend Class A.

```
class Employee {
    void work() {
        System.out.println("Employee is working.");
    }
}

class Developer extends Employee {
    void writeCode() {
        System.out.println("Developer is writing Java code.");
    }
}

class Manager extends Employee {
    void handleTeam() {
        System.out.println("Manager is managing the team.");
    }
}

public class HierarchicalDemo {
    public static void main(String[] args) {
        Developer d = new Developer();
        Manager m = new Manager();

        d.work();      // Inherited from Employee
        d.writeCode(); // Own method

        m.work();       // Inherited from Employee
        m.handleTeam(); // Own method
    }
}
```

```
Output

Employee is working.
Developer is writing Java code.
Employee is working.
Manager is managing the team.
```

Abstract Class

एक ऐसी क्लास होती है जिसे abstract कीवर्ड के साथ घोषित किया जाता है। इसका मुख्य उद्देश्य एक "ब्लूप्रिंट" (खाका) तैयार करना है ताकि अन्य क्लासेज उसे इनहेरिट (Inherit) कर सकें।

सरल शब्दों में कहें तो, Abstract Class हमें यह बताती है कि **"क्या करना है"**, लेकिन **"कैसे करना है"** (Implementation) यह उस क्लास पर निर्भर करता है जो इसे इनहेरिट करती है।

Abstract Class की मुख्य विशेषताएं

1. **Object नहीं बना सकते:** आप Abstract class का ऑब्जेक्ट (new कीवर्ड का उपयोग करके) नहीं बना सकते।
2. **Methods:** इसमें **Abstract Methods** (बिना बॉडी वाले) और **Non-abstract Methods** (बॉडी वाले) दोनों हो सकते हैं।
3. **Inheritance:** इसे इस्तेमाल करने के लिए किसी दूसरी क्लास से extends करना ज़रूरी है।
4. **Constructors:** इसमें कंस्ट्रक्टर और स्टैटिक मेथड्स भी हो सकते हैं।

Abstract Method क्या है?

एक ऐसा मेथड जिसका केवल नाम और सिग्नेचर होता है, लेकिन उसकी कोई कोडिंग (Body) नहीं होती। इसके आगे abstract कीवर्ड लगा होता है।

एक सरल उदाहरण (Example)

मान लीजिए हमारे पास एक Animal क्लास है। अब "Animal" का अपना कोई निश्चित साउंड नहीं होता, लेकिन "Dog" या "Cat" का होता है।

```
// Abstract Class
abstract class Animal {
    // Abstract method (इसकी कोई बॉडी नहीं है)
    public abstract void animalSound();

    // Regular method (इसकी बॉडी है)
    public void sleep() {
        System.out.println("Zzz...");
    }
}

// Subclass (Inherit करने वाली क्लास)
class Dog extends Animal {
    public void animalSound() {
        // यहाँ हम बता रहे हैं कि आवाज़ 'कैसे' होगी
```

Output

```
The dog says: bow wow
Zzz...
```

```
        System.out.println("The dog says: bow wow");
    }
}

class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog(); // Dog का ऑब्जेक्ट बनाया
        myDog.animalSound();
        myDog.sleep();
    }
}
```

Abstract Class का उपयोग कब करें?

- जब आप चाहते हैं कि कुछ कॉमन फीचर्स सभी सब-क्लासेस में हों (जैसे sleep())।
- जब आप चाहते हैं कि हर सब-क्लास किसी विशेष काम को अपने तरीके से करे (जैसे animalSound())।
- **Abstraction** को लागू करने के लिए, जिससे आप केवल जरूरी जानकारी यूजर को दिखाते हैं और इंटरनल डिटेल्स छुपा लेते हैं।

Abstract Class vs Interface

फीचर	Abstract Class	Interface
Methods	Abstract और Non-abstract दोनों हो सकते हैं।	इसमें ज्यादातर Abstract मेथड्स होते हैं।
Variables	Final, Non-final, Static सब हो सकते हैं।	केवल Static और Final होते हैं।
Inheritance	एक क्लास केवल एक ही Abstract class को extends कर सकती है।	एक क्लास कई Interfaces को implements कर सकती है।

What is an Interface? (सरल भाषा में)

कल्पना कीजिए कि एक **"Remote Control"** एक इंटरफेस है। रिमोट पर बटन होते हैं जैसे powerOn(), volumeUp(), और changeChannel()।

रिमोट यह नहीं जानता कि टीवी अंदर से कैसे चालू होता है, वह बस एक **नियम (Rule)** बनाता है कि "जो भी डिवाइस इस रिमोट से जुड़ना चाहती है, उसे ये बटन दबाने पर काम करना ही होगा।"

जब **Samsung** या **Sony** अपना टीवी बनाते हैं, तो वे इस रिमोट के नियमों को **लागू (Implement)** करते हैं और बताते हैं कि उनके टीवी में ये बटन क्या काम करेंगे।

Key Features of Interface (मुख्य विशेषताएं)

- **Rules Only:** इसमें सिर्फ काम की लिस्ट होती है (Methods), काम कैसे होगा यह नहीं लिखा होता।
- **Blueprint:** यह क्लास के लिए एक नक्शा है कि उसे कौन-कौन से फीचर्स देने ही पड़ेंगे।
- **Multiple Inheritance:** एक क्लास एक साथ कई इंटरफेस के नियमों को मान सकती है।

---

1. Syntax (लिखने का तरीका)

इंटरफेस बनाने के लिए interface कीवर्ड का इस्तेमाल होता है और उसे क्लास में जोड़ने के लिए implements का।

Java
```
interface Remote {
   void powerOn(); // No body, only the rule
}
```

---

2. Why we use it? (इसका उपयोग क्यों?)

Java में एक क्लास दो अलग-अलग क्लासेस से जुड़ नहीं सकती, लेकिन एक क्लास कई इंटरफेस से जुड़ सकती है। इसे **Multiple Inheritance** कहते हैं।

Full English Code Example: The Remote System

This program shows how different companies (Samsung and Sony) follow the same interface rules.

```java
// The Interface (The Rule Book)
interface Remote {
    void powerOn();
    void changeChannel();
}

// Samsung follows the rules
class SamsungTV implements Remote {
    public void powerOn() {
        System.out.println("Samsung TV is turning ON with logo.");
    }
    public void changeChannel() {
        System.out.println("Samsung TV channel changed.");
    }
}

// Sony follows the same rules but in its own way
class SonyTV implements Remote {
    public void powerOn() {
        System.out.println("Sony TV is turning ON with high-quality sound.");
    }
```

```
    public void changeChannel() {
        System.out.println("Sony TV channel changed using smart menu.");
    }
}

public class Main {
    public static void main(String[] args) {
        // We can use the Remote reference for any TV
        Remote myTV = new SamsungTV();
        myTV.powerOn();

        myTV = new SonyTV();
        myTV.powerOn();
    }
}
```

```
Output
Samsung TV is turning ON with logo.
Sony TV is turning ON with high-quality sound.
```

Key Points to Remember (महत्वपूर्ण बातें)

1. **Must Implement:** अगर किसी क्लास ने इंटरफेस लिया है, तो उसे उसके **सारे मेथड्स** बनाने ही पड़ेंगे।
2. **No Object:** आप new Remote() नहीं लिख सकते क्योंकि रिमोट खुद में कोई मशीन नहीं है, वह सिर्फ एक नियम है।
3. **Variables:** इंटरफेस में अगर आप int speed = 10; लिखते हैं, तो वह हमेशा के लिए फिक्स (Final) हो जाता है।

Final Keyword

Java में **final** एक keyword है जिसका इस्तेमाल "प्रतिबंध" (Restriction) लगाने के लिए किया जाता है। सरल शब्दों में, जब आप किसी चीज़ को final घोषित कर देते हैं, तो उसे **बदला नहीं जा सकता**।

आप final keyword का इस्तेमाल तीन जगहों पर कर सकते हैं: **Variables**, **Methods**, और **Classes** पर।

---

1. Final Variable (वैल्यू को फिक्स करना)

अगर आप किसी variable को final बना देते हैं, तो उसकी वैल्यू को दोबारा बदला नहीं जा सकता। यह एक **Constant** (स्थिरांक) बन जाता है।

- **Rule:** एक बार वैल्यू देने के बाद आप उसे री-असाइन नहीं कर सकते।

---

2. Final Method (ओवरराइडिंग रोकना)

अगर आप किसी Parent class के method को final बना देते हैं, तो Child class उस method को **Override** (बदल) नहीं सकती।

- **Rule:** इसका इस्तेमाल तब होता है जब आप चाहते हैं कि कोई खास लॉजिक सभी subclasses में एक जैसा ही रहे।

---

3. Final Class (इनहेरिटेंस रोकना)

अगर आप किसी class को final बना देते हैं, तो उस class को कोई दूसरी class **Inherit (extend)** नहीं कर सकती।

- **Rule:** सुरक्षा के लिए इसका इस्तेमाल किया जाता है ताकि कोई आपकी class का गलत फायदा न उठा सके। Java की String class भी एक final class है।

Full English Code Example: Using Final Keyword

यह प्रोग्राम दिखाता है कि final का इस्तेमाल कैसे करते हैं और उसे बदलने की कोशिश करने पर क्या होता है।

```
// Final Class cannot be extended
final class Security {
    void check() {
        System.out.println("Security check passed.");
    }
}

// class Hack extends Security { } // This will cause a COMPILE ERROR

class Parent {
    // Final Method cannot be overridden
    final void displayID() {
        System.out.println("Your ID is: 101");
    }
}

class Child extends Parent {
    // void displayID() { } // This will cause a COMPILE ERROR
}

public class FinalDemo {
    public static void main(String[] args) {
        // Final Variable
        final int speedLimit = 80;

        // speedLimit = 100; // This will cause a COMPILE ERROR

        System.out.println("Fixed Speed Limit: " + speedLimit);

        Child obj = new Child();
        obj.displayID();
    }
}
```

Output

```
Fixed Speed Limit: 80
Your ID is: 101
```

Comparison Table: Final Usage

Used with	Result (परिणाम)
Variable	वैल्यू Constant बन जाती है (बदली नहीं जा सकती)।
Method	Child class इस method को override नहीं कर सकती।
Class	इस class से नई subclasses नहीं बनाई जा सकतीं।

Key Points to Remember (महत्वपूर्ण बातें)

1. **Error:** अगर आप final चीज़ को बदलने की कोशिश करेंगे, तो Java **Compile-time error** देगा।
2. **Blank Final Variable:** आप एक final variable को बिना वैल्यू के declare कर सकते हैं, लेकिन उसे सिर्फ **Constructor** के अंदर ही वैल्यू दी जा सकती है।
3. **Purpose:** इसका मुख्य उद्देश्य **Security** और **Stability** (स्थिरता) है।

Static Keyword in Java

1. Introduction (परिचय)

Java में static एक Non-access modifier है। इसका मुख्य उपयोग Memory Management के लिए किया जाता है।

जब हम किसी चीज़ (Variable, Method, Block या Inner class) को static घोषित करते हैं, तो वह किसी विशेष ऑब्जेक्ट का हिस्सा न रहकर पूरी Class का हिस्सा बन जाती है।

---

2. Key Features (मुख्य विशेषताएं)

- **Memory Efficiency:** static मेंबर्स को मेमोरी केवल एक बार मिलती है जब क्लास लोड होती है।
- **Direct Access:** इन्हें एक्सेस करने के लिए ऑब्जेक्ट (new) की ज़रूरत नहीं होती। इन्हें सीधे ClassName.name से एक्सेस किया जा सकता है।
- **Shared Resource:** यह सभी ऑब्जेक्ट्स के बीच एक "Common" डेटा की तरह काम करता है।

---

3. Types of Static Members (स्टैटिक के प्रकार)

A. Static Variables (Class Variables)

जब कोई वेरिएबल static होता है, तो उसकी पूरी मेमोरी में केवल **एक ही कॉपी** बनती है।

- **उपयोग:** ऐसी वैल्यू के लिए जो सबके लिए समान हो (जैसे: collegeName, companyName, bankInterestRate)।
- **फायदा:** अगर 1000 स्टूडेंट्स के लिए कॉलेज का नाम स्टोर करना है, तो static यूज़ करने पर मेमोरी सिर्फ 1 बार लगेगी, 1000 बार नहीं।

B. Static Methods

ये वे मेथड्स हैं जिन्हें कॉल करने के लिए ऑब्जेक्ट बनाने की आवश्यकता नहीं होती।

- **नियम 1:** Static method केवल static डेटा को ही एक्सेस कर सकता है (वह non-static डेटा को नहीं देख सकता)।
- **नियम 2:** Static method के अंदर this और super कीवर्ड का इस्तेमाल नहीं किया जा सकता।
- **उदाहरण:** public static void main(String[] args) (Java का सबसे प्रसिद्ध स्टैटिक मेथड)।

C. Static Block

इसका उपयोग static variables को इनिशियलाइज़ (Initialize) करने के लिए किया जाता है। यह main मेथड से भी पहले चलता है, जैसे ही क्लास मेमोरी में लोड होती है।

D. Static Nested Class

केवल इनर क्लासेस ही static हो सकती हैं। इन्हें बाहरी क्लास के ऑब्जेक्ट की ज़रूरत नहीं होती।

---

4. Difference: Static vs Non-Static

Feature	Static	Non-Static (Instance)
Binding	Class के साथ जुड़ा होता है।	Object के साथ जुड़ा होता है।
Memory	क्लास लोड होते समय एक ही बार।	हर बार ऑब्जेक्ट बनाने पर अलग-अलग।
Access	ClassName.method()	object.method()
Storage Area	Method Area (Static Pool)	Heap Memory

---

5. Coding Example for

```
class Student {
    int rollNo;        // Non-static (हर स्टूडेंट का अलग होगा)
    String name;       // Non-static
    static String college = "LPU"; // Static (सबका सेम रहेगा)

    // Static Method
    static void changeCollege() {
        college = "IIT";
    }

    // Constructor
    Student(int r, String n) {
        rollNo = r;
        name = n;
    }
```

```
    void display() {
        System.out.println(rollNo + " " + name + " " + college);
    }
}

public class TestStatic {
    public static void main(String[] args) {
        // बिना ऑब्जेक्ट बनाए कॉलेज बदल दिया
        Student.changeCollege();

        Student s1 = new Student(101, "Rahul");
        Student s2 = new Student(102, "Amit");

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

6. Why is main method static? (V.I.P Question)

इंटरव्यू में अक्सर पूछा जाता है कि public static void main क्यों?

जवाब: ताकि JVM (Java Virtual Machine) प्रोग्राम शुरू करते समय बिना ऑब्जेक्ट बनाए सीधे इस मेथड को कॉल कर सके। अगर यह static नहीं होता, तो JVM कन्फ्यूज हो जाता कि किसका ऑब्जेक्ट पहले बनाए।

Method Overloading के नियम

किसी मेथड को ओवरलोड करने के लिए, कम से कम एक चीज़ अलग होनी चाहिए:

1. **पैरामीटर्स की संख्या (Number of parameters):** जैसे एक मेथड में 2 पैरामीटर हों और दूसरे में 3।
2. **पैरामीटर्स का डेटा टाइप (Data type of parameters):** जैसे एक में int हो और दूसरे में double हो।
3. **पैरामीटर्स का क्रम (Sequence of parameters):** जैसे (int, String) और (String, int)।

महत्वपूर्ण नोट: केवल **Return Type** (जैसे void या int) बदलने से Method Overloading नहीं होती। पैरामीटर्स का बदलना ज़रूरी है।

---

2. (Why Overloading?)

सोचिए अगर आपको दो नंबर जोड़ने के लिए addInt(), तीन नंबर के लिए addThree() और डेसीमल नंबर के लिए addDouble() नाम के अलग-अलग मेथड्स बनाने पड़ें। यह बहुत कन्फ्यूजिंग होगा। Overloading की मदद से आप एक ही नाम add() रख सकते हैं, जो अलग-अलग इनपुट के हिसाब से खुद काम करेगा। इससे कोड की **Readability** बढ़ती है।

---

3. (Example)

```
class Calculator {
    // 1. दो इंटीजर जोड़ने के लिए
    void sum(int a, int b) {
        System.out.println("Sum of 2 numbers: " + (a + b));
    }

    // 2. तीन इंटीजर जोड़ने के लिए (संख्या बदली)
    void sum(int a, int b, int c) {
        System.out.println("Sum of 3 numbers: " + (a + b + c));
    }

    // 3. दो डबल वैल्यूज के लिए (डेटा टाइप बदला)
    void sum(double a, double b) {
        System.out.println("Sum in double: " + (a + b));
    }
}

public class Main {
    public static void main(String[] args) {
        Calculator obj = new Calculator();

        obj.sum(10, 20);          // पहला मेथड चलेगा
        obj.sum(10, 20, 30);      // दूसरा मेथड चलेगा
```

```
        obj.sum(10.5, 20.5);       // तीसरा मेथड चलेगा
    }
}
```

---

4. Difference Table for Exam

विशेषता (Feature)	विवरण (Description)
नाम	सभी मेथड्स का नाम समान (Same) होना चाहिए।
सिग्नेचर	पैरामीटर्स (Arguments) अलग होने चाहिए।
समय (Time)	यह कंपाइल-टाइम पर तय होता है (Compile-time Polymorphism)।
क्लास	यह एक ही क्लास के अंदर होता है।

Export to Sheets

क्या "Main" मेथड को ओवरलोड किया जा सकता है?

हाँ! आप public static void main को ओवरलोड कर सकते हैं, लेकिन JVM हमेशा उसी main को चलाएगा जिसका सिग्नेचर String[] args वाला है। बाकी ओवरलोडेड main मेथड्स को आपको खुद कॉल करना पड़ेगा।

Method Overriding in Java

1. Definition (परिभाषा)

जब एक Child class (Sub-class) में वही मेथड बनाया जाता है जो उसके Parent class (Super-class) में पहले से मौजूद है, तो Child class वाला मेथड Parent class के मेथड को "Override" कर देता है (यानी उसकी जगह ले लेता है)।

इसे **Runtime Polymorphism** या **Dynamic Method Dispatch** भी कहा जाता है क्योंकि Java यह फैसला "Runtime" पर लेता है कि कौन सा मेथड चलेगा।

---

2. Overriding के नियम

1. **Name:** मेथड का नाम Parent class के मेथड जैसा ही होना चाहिए।
2. **Parameters:** पैरामीटर्स (Arguments) की संख्या और टाइप बिल्कुल सेम होने चाहिए।
3. **Return Type:** रिटर्न टाइप भी सेम (या सह-परिवर्ती/covariant) होना चाहिए।
4. **Access Modifier:** Child class में एक्सेस लेवल, Parent class से कम नहीं हो सकता। (जैसे अगर Parent में protected है, तो Child में private नहीं हो सकता, सिर्फ protected या public होगा)।
5. **Static Methods:** Static मेथड्स को Override नहीं किया जा सकता (इसे Method Hiding कहते हैं)।

---

3. उदाहरण (Example)

```java
// Parent Class
class Bank {
    int getRateOfInterest() {
        return 0;
    }
}

// Child Class 1
class SBI extends Bank {
    @Override // यह एक annotation है जो बताता है कि हम override कर रहे हैं
    int getRateOfInterest() {
        return 8;
    }
}

// Child Class 2
class AXIS extends Bank {
    @Override
    int getRateOfInterest() {
        return 9;
    }
}

public class TestOverriding {
    public static void main(String args[]) {
        SBI s = new SBI();
        AXIS a = new AXIS();

        System.out.println("SBI Interest: " + s.getRateOfInterest()); // SBI
वाला चलेगा
        System.out.println("AXIS Interest: " + a.getRateOfInterest()); //
AXIS वाला चलेगा
    }
}
```

4. Difference Table: Overloading vs Overriding

Feature	Method Overloading	Method Overriding
नाम (Name)	सेम होना चाहिए।	सेम होना चाहिए।
पैरामीटर (Parameters)	अलग (Different) होने चाहिए।	बिल्कुल सेम (Same) होने चाहिए।
क्लास (Class)	एक ही क्लास के अंदर होता है।	दो अलग-अलग क्लासेज (Inheritance) में होता है।
समय (Time)	Compile-time पर तय होता है।	Runtime पर तय होता है।
उद्देश्य (Goal)	कोड की Readability बढ़ाना।	किसी मेथड को विशेष रूप से लागू करना।