PAGE NO.
DATE

Differentiate b/w POP and OOP

Procedural Oriented (e.g., C)	Objects Oriented (eg. Java)
Focauses on Functions/Algorithms (How to do it).	Focauses on Data/Objects (What is involved).
Program is divided into small functions.	Program is divided into Objects and Classes.
Less Secure. Data moves freely b/w functions.	More secure. Data is hidden using. Encapsulation
Follows Top-Down approach (Main → Sub-Functions)	Follows Bottom-up approach (Build objects → Integrate them).
Less reusable code	High reusability due to inheritance and Polymorphism.

Define class and Objects with an example

1. Class:

- A class is a user-defined blueprint or prototype from which objects are created.
- It represents the set of properties (fields) and methods that are common to all objects of one type.
- It does not occupy memory space until an objects is instantiated.

- Syntax (Java)

```
class Car {
    String color;
    void drive() {
        // code.
    }
}
```

2. Object:

- An object is a basic unit of object-oriented Programming and represents real-life entities.
- It is an instance of a class.
- When an object is created, memory is allocated in the heap.
- It has three characteristics:
  - State: (Represented by variables/attributes, e.g. Red Color)
  - Behavior: (Represented by methods, e.g., driving, braking)
  - Identity: (Unique address in memory)

* Code

```
// Defination of the class (The blueprint)
class Student {
        int id;
        String name;

        void study() {
            System.out.println(name + " is studying for exams.");
        }
}

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

        s1.id = 101;
        s1.name = "Rahul";

        s1.study();

    }
}
```

* Output

Rahul is studying for exams.

Student s1: Creates a refrence variable in stack memory.
new student(): Creates the actual objects in Heap memory.
=; Links the refrence to the objects.

The 4 Pillars of OOPs // Main features of OOPs

1. Encapsulation (Packing Data)

- Encapsulation is defined as the wrapping up of data (variables) and code (methods) acting on the data into a single unit (class).

- It provides Data Hiding. The internal data of an object cannot be accessed directly from outside the class.

- It is achieved by declaring the variables of a class as ~~pri~~ private and providing public setter and getter methods to modify and view the value.

* Code

```
class account {
    private double balance; // Data hidden (Private)

    public void deposit (double amount) {
        if (amount > 0) {
            balance += amount
        }
    }
}
```

PAGE NO.
DATE

...ures of OOPs

...wrapping up
...s) acting on
...class).

...data of an
...from outside

...variables of a
...iding public
...odify and view

...hidden (Private)

...t) {

PAGE NO.
DATE

2. Abstraction

- Data abstraction refers to the property by which only the essential details are displayed to the user, while the background implementation is hidden.
- It reduces programming complexity and effort.
- In Java, abstraction is achieved using Abstraction classes and Interfaces.

* Practical code:

```
abstract class Vehicle {
        abstract void start(); // Sirf idea diya, implementation nhi
}

class Car extends Vehicle {
        void start() {
            System.out.println("starts with Key");
    }
}
```

3. Inheritance (Reusability)

- Inheritance is the mechanism by which one class acquires the properties (fields) and behaviors (methods) of another class.

- Be It promotes code Reusability and establishes a parent-child relationship. (Parent)
- Supper Class: The class whose features are inherited.
- Sub class: The class that inherits the other class (child)

PAGE NO.
DATE

* Code

```
class Animal { // Parent
    void eat() { system.out.println("Eating...."); }
}
class Dog extends Animal {
    void bark() { system.out.println("Barking..");
    }
}
```

4. Polymorphism (One name, Many forms)

- Polymorphism is the ability of a variable, function or object to take on multiple forms.
- It allows us to perform a single action in different ways.
- Multiple methods with the same name but different arguments. (Method overloading)
- Providing a specific implementation of a method that is already provided by its parent ~~child~~ class. (Method overriding)

* Code

```
~~class Calculate~~
class Calculator {
    int add(int a, int b) { return a+b; }
    double add(double a, double b) { return a+b; }
}
```

PAGE NO.
DATE

Java Basic

1. Anatomy of a Java Program

```
1  public class Firstprogram {
2          public static void main(String[] args) {
3              System.out.println("Hello, Engineer!");
4          }
5      }
```

Class Firstprogram: All are in a Class, File name Firstprogram.java.

public: JVM (Java virtual Machine), access control.

static: For make object and call main method, so use static

void: nothing return

String[] args: Ye command-line arguments lene

System: built in class

out: System class's static object

println(): for print method

2. Data types —

A. Primitive Data Types

Data Type	Size	Description	Example
byte	1	Very small number (-128 to 127)	byte age = 20;
short	2	Small number	short salary = 10,000;

int	4	Standard integers (Default)	int p = 100000;
log	8	Very large integers	long dist = 1234567891;
float	4	Decimal number	float pi = 3.14f;
double	8	Long decimal number	double price = 99.99;
char	2	Single character	char grade = 'A';
boolean	1	True / False	boolean isPass = True;

B. Non-primitive :- Stores objects -

String, Arrays, Classes

3. Type Conversion & Casting

(i) Widening casting (Implicit/Automatic)

Converting a smaller type to a large type size.

byte > short > char > int > long > float > double

```
int m = 9;
double md = m
```

(ii) Narrowing Casting (Explicit/Manual):

- Converting a larger type to a smaller size type.
- Requires parenthesis (targetType).
- Warning : Data loss possible (decimal part removed)

```
double myd = 9.78d;
int myi = (int) myd;
```

1. ** Constructors

A constru
to init
on objec

- Rule -
  - Must
  - Must
- Types -

1. Default

2. Paramet

* Code

class

Defaut

Paramet
-rized
Constru
for.

1. # Constructors (objects initializers)

A constructor is a special type of method used to initialize an objects. It is called implicity when an object is created.

- Rule -
  - Must have the same name as the class.
  - Must not have a return type (not even void).

- Types -

1. Default Constructor: No arguments. ~~Init~~ Initiolizes default values (0, null).

2. Parameterized Constructor: Takes arguments to initialize specific values.

* Code

```
class Student {
    int id;
    string name;

    Student() {                                   // Default
        System.out.println("Student object Created.");
        id = 0;
        name = "UnKnown";
    }
    Student(int i, string n) {                    // Parameterized Constructor
        id = i;
        name = n;
    }
```

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

public class Main {
    public static void main(String[] args) {
        Student s1 = new student();
        Student s2 = new Student(101, "Rahul");

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

2. The this keyword (current object Refrence)

'this' keyword is a refrence variable that refers to the current class object.

To resolve the naming conflict b/w instance variables (class fields) and parameters (local variables).

* Practical code

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

    Employee(int id, string name) {
        this.id = id;
        this.name = name;
    }
}
```

3. Method Overloading (compile-time Polymorphism)

If a class has multiple methods having the same name but different in parameters (number or type of arguments), it is known as Method Overloading.

Increases the readability of the program (User doesn't need to remember different name like addTwo(), addThree()..).

You cannot overload by just changing the return type.

* code

```
class Calculator {
    int add(int a, int b) {
        return a+b;
    }

    int add(int a, int b, int c){
        return a+b+c;
    }

    double add(double a, double b) {
        return a+b;
    }
}
```

4. Static keyword (Briefly)

- Static Variable : static string college = "MDU";
- Static Method : static void changecollege() {...}

Exploring String class

In Java a string is an object that represents a sequence of characters. It is not a primitive data type but a predefined class in the java.lang package.

- Key Features:

1. Object-Based: Strings are objects, stored in the Heap memory.

2. Immutability: Once a string object is created, its data cannot be changed. If we try to alter it, a new object is created.

3. String Constant Pool (SCP): A special memory area within the Heap to store unique string literals to save memory.

- Ways to create String:

1. By Literal: String s = "MDU Rohtak"; (Memory-efficient, checks SCP first).

2. By new Keyword: String s = new String("MDU Rohtak"); (Forces creation of a new object in Heap, even if it exists in SCP)

PAGE NO.
DATE

- Important string Methods

Method	Description	Example
length()	Returns the number of characters.	"Hi".length() → 2
charAt(int i)	Returns char at specific index.	"Java".charAt(0) → 'J'
concat(string s)	Joins two strings.	"A".concat("B") → "AB"
equals(Object o)	Compares content of strings.	s1.equals(s2) → true/false
substring(int i)	Returns part of the string.	"Hello".substring(2) → "llo"

Arrays

An Array in Java is an object which contains elements of a similar data type. Ideally, it is a collection of homogeneous data types stored in contiguous memory locations.

- In Java, arrays are treated as objects.
- Memory is allocated dynamically on the Heap.
- Java checks for ArrayIndexOutOfBounds Expection (C/C++ garbage value de deta hai, Java error deta hai agar limit cross karo).

- Types of Arrays:

1. Single Dimensional Array: A simple list of variables.
   - Syntax: `int[] arr = new int[5];`

2. Multi-Dimensional Array: Arrays within an array (Matrix Format).
   - Syntax: `int[][] arr = new int[3][3];`

Garbage collection (GC)

Garbage collection (GC) in Java is the process of automatic memory management. It finds and deletes the objects that are no longer reachable or needed by the program to free up heap memory resources.

- How it works:

1. Unreachable objects: An objects is considered garbage if it doesn't have any reference pointing to it
2. Daemon Thread : The Garbage collection runs as a background thread.
3. System.gc() : Although GC is automatic, we can request the JVM to run it using this method.
4. finalize() : Just before destroying an object the Garbage Collector calls the finalize() method of that object to perform cleanup activities (like closing files).

- Advantages:

- Increases memory efficiency.
- Programmer doesn't need to manually manages memory
- Prevents memory leaks.

s the process
. It finds and
no longer
program to free

considered garbage
have any reference

collection runs as a

automatic, we can
M to run it

e destroying an object
Collector calls the
object to perform
ing files).

manually manages

· Code

```
public class GarbageCollectionDemo {

    public void finalize() {
        System.out.println("Garbage Collected: Deleted")

    public static void main(String[] args) {
      GarbageCollectionDemo obj1 = new Garb-----();
       "           "        "      " 2  =  "     ·  ·  ·  ·  ·();

      obj1 = null;
      " 2 = "

      system.gc();

      system.out.println("Request ------");
    }
}
```

Gemini

Java OOPs

PRO

1. Theory Notes (Exam Point of View)

Operators: Java provides a rich set of operators to manipulate variables.

1. **Arithmetic:** `+`, `-`, `*`, `/`, `%` (Modulo - Remainder).
2. **Relational:** `==`, `!=`, `>`, `<`, `>=`, `<=`.
3. **Logical:** `&&` (AND), `||` (OR), `!` (NOT).
4. **Assignment:** `=`, `+=`, `-=`.
5. **Ternary Operator:** `(condition) ? true_value : false_value`. (Shortest if-else).

Control Flow Statements:

- **Decision Making:** `if`, `if-else`, `switch`.
- **Looping:** `for`, `while`, `do-while`, `for-each`.
- **Jump:** `break`, `continue`, `return`.

Ask Gemini 3

Tools

Pro

Gemini can make mistakes, so double-check it