Object-Oriented Programming in Java

1. Introduction to OOP

Object-Oriented Programming (OOP) is a programming approach that organizes software design around **objects rather than functions**. It helps in developing programs that are easy to understand, maintain, and reuse.

In OOP, real-world entities like students, cars, or bank accounts are treated as objects.

2. Object

An **object** is a real-world entity that has:

- **Attributes (data/properties)** → describe the object
- **Methods (functions/behavior)** → define what the object can do

Example: Bank Account

- Attributes: Account number, name, balance
- Methods: Deposit(), Withdraw(), CheckBalance()

Thus, an object represents both **data and behavior**.

3. Class

A **class** is a blueprint or template used to create objects.
It defines the structure and behavior that objects will have.

Example:

```
class Student {
    int rollNo;
    String name;

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

Here, `Student` is a class, and objects of this class will have roll number and name.

4. Data Abstraction

Abstraction means showing only the necessary details and hiding the internal implementation.

Real-life Example:
When you use a mobile phone, you press buttons to perform actions, but you don't know the internal circuitry.

Benefits:

- Reduces complexity
- Improves security
- Makes system easy to use

5. Encapsulation

Encapsulation is the process of combining data and methods into a single unit (class).

It also restricts direct access to data using access modifiers.

Example:

```
class Account {
    private double balance;

    public void deposit(double amount) {
        balance += amount;
    }

    public double getBalance() {
        return balance;
    }
}
```

Advantages:

- Data security
- Controlled access
- Easy maintenance

6. Inheritance

Inheritance allows one class to **acquire properties and methods of another class**.

- **Parent class (Base class)**
- **Child class (Derived class)**

Example:

```
class Animal {
    void eat() {
        System.out.println("Eating...");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("Barking...");
    }
}
```

Types of Inheritance:

1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance
4. Hybrid Inheritance

(*Note: Java does not support multiple inheritance using classes*)

7. Polymorphism

Polymorphism means **one name, many forms**.

It allows a method to perform different tasks based on input.

Types:

(a) Compile-Time Polymorphism (Method Overloading)

Same method name, different parameters.

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

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

(b) Runtime Polymorphism (Method Overriding)

Child class provides its own implementation of parent method.

```
class Animal {
    void sound() {
        System.out.println("Animal sound");
    }
}

class Dog extends Animal {
    void sound() {
        System.out.println("Bark");
    }
}
```

8. Binding

Binding means linking a method call with its definition.

Types:

- **Static Binding (Early Binding)**
  - Done at compile time
  - Faster execution
- **Dynamic Binding (Late Binding)**
  - Done at runtime
  - More flexible

9. Message Passing

Objects communicate with each other by sending messages.

Example:
A user sends a request to a bank account object to check balance.

10. Reusability

Reusability means using existing code again without rewriting it.

It is achieved mainly through **inheritance**.

11. Procedural vs Object-Oriented Programming

Feature	Procedural Programming	Object-Oriented Programming
Focus	Functions	Objects
Approach	Top-down	Bottom-up
Security	Low	High
Reusability	Limited	High

Feature	Procedural Programming	Object-Oriented Programming
Complexity handling	Difficult	Easy

12. Java Development Environment (JDK Tools)

Java provides tools for development:

- **Javac** – Compiles Java code into bytecode
- **Java** – Runs Java programs
- **Javadoc** – Generates documentation
- **Jdb** – Debugging tool
- **Applet Viewer** – Runs applets

13. Variables in Java

A **variable** is a named memory location used to store data.

Types of Variables:

1. **Instance Variable** – Defined inside class, outside methods
2. **Class Variable (Static)** – Shared among all objects
3. **Local Variable** – Declared inside methods
4. **Parameters** – Passed to methods

14. Data Types in Java

(a) Primitive Data Types:

- int, float, double, char, boolean, byte, short, long

(b) Non-Primitive:

- Class, Array, Interface

15. Operators in Java

- ++ → Increment
- -- → Decrement

Prefix: ++a (increase first)
Postfix: a++ (use first, then increase)

16. Arrays

An **array** stores multiple values of the same type.

Example:

```
int[] A = new int[5];
A[0] = 10;
A[1] = 20;
```

Features:

- Fixed size
- Indexed (starts from 0)

17. Class and Object Creation

```
class Rectangle {
    int length;
    int width;
}
```

Creating Object:

```
Rectangle r = new Rectangle();
r.length = 10;
r.width = 5;
```

18. Access Modifiers

Modifier	Access Level
public	Everywhere
private	Within class only
protected	Same package + subclasses
default	Same package

19. Abstract Class and Interface

Abstract Class:

- Can have abstract and non-abstract methods
- Cannot be instantiated

Interface:

- Contains only abstract methods (mostly)

- Supports multiple inheritance

20. Exception Handling

An **exception** is an error that occurs during program execution.

Keywords:

- **try** → risky code
- **catch** → handles error
- **throw** → manually throw exception
- **throws** → declare exception
- **finally** → always executes

Example:

```
try {
    int a = 10 / 0;
} catch (Exception e) {
    System.out.println("Error occurred");
}
```

21. Applications of OOP

OOP is widely used in:

- Real-time systems
- Simulation and modeling
- Database systems
- AI and expert systems
- Web and desktop applications