OOPS using JAva

Adesh Lokhande

1. Abstraction
   Abstraction ka matlab hai unnecessary details ko hide karke sirf essential features ko dikhana. Matlab, user ko sirf important cheezein dikhti hain aur implementation ka complex part hide rehta hai. Jaise car ka example lo: driver ko sirf steering, brake, aur accelerator use karne hote hain, engine ke internal workings ke baare mein sochna nahi padta.

2. Inheritance
   Inheritance ka matlab hai ek class (child/subclass) dusri class (parent/superclass) ke properties aur methods ko reuse kar sakti hai. Isse code duplicate nahi hota aur maintain karna easy ho jata hai. Jaise, `Vehicle` class se `Car` aur `Bike` classes common features inherit kar sakti hain.

3. Encapsulation
   Encapsulation ka matlab hai data aur methods ko ek single unit (class) me bundle karna aur data ko direct access se protect karna. Isme usually private variables hote hain aur unhe access karne ke liye public getter/setter methods use hote hain. Ye security aur data integrity ke liye important hai.

4. Classes, Subclasses and Superclasses
   Class ek blueprint hai jisme objects ke liye properties (variables) aur behaviors (methods) define hote hain. Subclass ek class hai jo kisi dusri class (superclass) se inherit karti hai, matlab parent class ke features reuse karti hai. Superclass wo class hoti hai jisse features inherit kiye jaate hain. Jaise `Animal` superclass se `Dog` subclass inherit kar sakta hai.

5. Polymorphism and Overloading
   Polymorphism ka matlab hai ek interface ya method ka multiple forms me kaam karna. Overloading ek type ka polymorphism hai jisme same method name ke saath different parameters use hote hain. Jaise `add(int a, int b)` aur `add(int a, int b, int c)` same method name hai, lekin different arguments ke saath kaam karte hain.

6. Message Communication
   Message communication ka matlab hai objects ek dusre se data ya request bhejte hain methods ke through. Ye OOP me objects ke beech interaction ka main way hai. Jaise, `Car` object `Engine` object ko `start()` method call karke message bhejta hai engine start karne ke liye.

7. Procedure-Oriented vs. Object-Oriented Programming concept
   Procedure-Oriented Programming (POP) me focus functions/procedures par hota hai, aur data alag store hota hai. Object-Oriented Programming (OOP) me data aur functions dono objects me encapsulate hote hain, aur real-world modeling easy hoti hai. OOP me code reusable, maintainable aur secure hota hai compared to POP.

8. Basics of Java
   Java ek high-level, object-oriented programming language hai jo platform-independent hai (Write Once, Run Anywhere). Java me code classes aur objects ke through likha jata hai. Isme strong type checking, automatic memory management (garbage collection), aur built-in libraries hoti hain jo programming easy aur secure banati hain.

9. Background/History of Java
   Java ko 1991 me James Gosling aur Sun Microsystems team ne develop kiya tha. Pehle iska naam Oak tha, lekin later Java rakh diya gaya. Java ka main goal platform-independent aur secure programming language create karna tha jo internet aur network applications ke liye suitable ho.

10. Java and the Internet
    Java internet ke liye perfect hai kyunki ye platform-independent hai aur secure applications banata hai. Java applets aur web applications ko easily run kar sakta hai browsers me without kisi extra setup ke. Iski network libraries se client-server communication bhi easy hoti hai.

11. Advantages of Java
    Java ke kuch advantages hain:
- **Platform-independent:** Write Once, Run Anywhere.
- **Object-Oriented:** Code reusable aur maintainable.
- **Secure:** Memory leaks aur unauthorized access se safe.
- **Robust:** Error handling aur strong type checking.
- **Multithreading Support:** Multiple tasks ek saath perform kar sakta hai.
- **Rich API:** Ready-made libraries programming easy banate hain.

12. Java Virtual Machine & Byte Code
    Java code pehle **bytecode** me convert hota hai, jo platform-independent hota hai. Ye bytecode **Java Virtual Machine (JVM)** ke through run hota hai. JVM har platform ke liye specific hota hai, isliye same Java program alag-alag systems par run ho sakta hai bina change kiye.

13. Java Environment Setup
    Java run karne ke liye environment setup karna padta hai. Isme **JDK (Java Development Kit)** install karna hota hai, jo compiler aur tools provide karta hai. Fir **PATH variable** set karke system ko JDK location batani hoti hai. IDE (like Eclipse ya IntelliJ) use karke coding aur running easy ho jati hai.

14. Java Program Structure
    Java program me basic structure hota hai:
- **Class Declaration:** `` `class` `` keyword se class define hoti hai.
- **Main Method:** `` `public static void main(String[] args)` `` program execution start karta hai.
- **Statements/Instructions:** Code jo program ka logic define karta hai.

- **Comments:** Code explain karne ke liye, compiler ignore karta hai.
  Ye structure standard hota hai har Java program me.

15. Integers
    Integers Java me numbers ke liye use hote hain jo decimal point ke bina hote hain. Java me integer ke liye **byte, short, int, aur long** data types hote hain, jo alag-alag size aur range ke liye use hote hain. Ye numbers mathematical calculations aur counting me kaam aate hain.

16. Floating Point type
    Floating point types decimal numbers ko store karte hain. Java me **float** aur **double** types hote hain. `float` less precision aur chhoti memory ke liye, `double` high precision aur large numbers ke liye use hota hai. Ye real numbers aur calculations me kaam aate hain.

17. Characters
    Characters single letters, digits, ya symbols ko represent karte hain. Java me **char** data type use hota hai, jo 16-bit Unicode character store karta hai. Ye text processing aur strings ke parts ke liye kaam aata hai.

18. Booleans
    Boolean data type sirf **true** ya **false** values store karta hai. Java me **boolean** keyword use hota hai. Ye decision-making aur conditional statements (like if-else) me kaam aata hai.

19. User Defined Data Type
    User-defined data types wo types hote hain jo programmer khud create karta hai, jaise **class, enum, aur interface**. Ye custom data aur behavior define karne ke liye use hote hain, jisse complex problems easily solve ki ja sakti hain aur code reusable banta hai.

20. Identifiers & Literals
    Identifiers names hote hain jo variables, methods, aur classes ko denote karte hain. Java me letters, digits, `_` aur `$` use kar sakte hain, lekin number se start nahi kar sakte.
    Literals fixed values hoti hain jo code me directly use hoti hain, jaise `10, 3.14, 'A', true`.

21. Declarations of constants & variables
    Variables wo memory locations hain jisme data store hota hai aur value change ho sakti hai. Java me declare karte hain: `int age;`
    Constants wo values hoti hain jo change nahi hoti. Java me `final` keyword se declare karte hain: `final int DAYS_IN_WEEK = 7;`

22. Type Conversion and Casting
    Type Conversion automatic data type change hai, jaise `int` se `double` me value convert ho jati hai (widening).

Type Casting manually data type change karna hai, jaise `(int) 3.14` se `double` ko `int` me convert karte hain (narrowing). Ye calculations aur compatibility ke liye important hai.

23. Scope of variables & default values of variables declared
Scope of variables ka matlab hai variable ka program me kahaan-kahaan access possible hai. Local variables sirf method ke andar access hote hain, instance variables class ke andar, aur static variables class level par globally accessible hote hain.
Default values wo values hain jo automatically assign hoti hain agar variable explicitly initialize na ho:

- int → 0
- float/double → 0.0
- char → '\u0000'
- boolean → false
- Object references → null

24. Wrapper classes
Wrapper classes Java me primitive data types ko objects me convert karne ke liye use hote hain. Jaise `int` → `Integer`, `char` → `Character`, `boolean` → `Boolean`. Ye classes methods provide karte hain jaise parsing, comparison, aur conversion between types.

25. Comment Syntax
Comments code me explanations likhne ke liye use hote hain aur compiler ignore karta hai. Java me 3 types ke comments hain:

- **Single-line comment:** `// Ye comment hai`
- **Multi-line comment:** `/* Ye comment multiple lines me ho sakta hai */`
- **Documentation comment:** `/** Ye JavaDoc ke liye hota hai */`

26. Garbage Collection
Garbage Collection Java me automatic memory management ka process hai. Ye unused objects ko detect karke memory free kar deta hai, jisse memory leak aur performance issues avoid hote hain. Programmer ko manually memory free karne ki zarurat nahi hoti.

27. Types of Arrays
Arrays multiple values ek single variable me store karne ke liye use hote hain. Java me arrays ke types:

- **Single-dimensional array:** Linear list of elements, jaise `int[] arr = {1,2,3};`
- **Multi-dimensional array:** Elements ka grid ya table, jaise `int[][] matrix = {{1,2},{3,4}};`
Ye data ko organized aur easily accessible banate hain.

28. Creation, Concatenation and Conversion of a String

- **Creation:** Strings Java me `" "` ya `new String()` se create karte hain, jaise `String name = "Adesh";`
- **Concatenation:** Strings ko join karna `+` operator ya `concat()` method se, jaise `"Hello " + "World"`

- **Conversion:** Primitive types ko string me convert karne ke liye `String.valueOf()` use hota hai, aur string ko numeric type me convert karne ke liye `Integer.parseInt()` ya `Double.parseDouble()` use karte hain.

29. Decision & Control Statements
    Decision statements program me conditions ke basis par different actions perform karte hain. Jaise **if, if-else, switch**.
    Control statements loop ya flow control karte hain: **for, while, do-while** loops aur **break, continue, return** statements use hote hain. Ye program ko flexible aur dynamic banate hain.

30. Different Operators
    Java me operators symbols hote hain jo values par operations perform karte hain. Types:

- **Arithmetic:** `+ - * / %`
- **Relational:** `== != > < >= <=`
- **Logical:** `&& || !`
- **Assignment:** `= += -= *= /=`
- **Unary:** `++ -- + - !`

    Ye calculations, comparisons, aur decision-making me kaam aate hain.

31. Defining classes, fields and methods
    Class ek blueprint hai objects create karne ke liye.

- **Fields:** Variables jo class ke properties store karte hain.
- **Methods:** Functions jo class ke behaviors define karte hain.

    Example:

```
class Car {
    String color; // field
    void drive() { // method
        System.out.println("Car is driving");
    }
}
```

32. Creating objects
    Objects class ke real instances hote hain. Java me **new** keyword se object create karte hain.
    Example:

```
Car myCar = new Car();
myCar.color = "Red";
myCar.drive();
```

Yahan `myCar` ek object hai `Car` class ka, jiske through fields aur methods access kar sakte hain.

33. Accessing rules

Accessing rules define karte hain kaunse class members (fields/methods) accessible hain. Java me access modifiers:

- **private:** Sirf same class me accessible
- **default (no modifier):** Same package me accessible
- **protected:** Same package + subclasses me accessible
- **public:** Anywhere accessible

Ye data security aur encapsulation ke liye important hain.

34. This keyword

`this` keyword current object ko refer karta hai. Ye fields aur methods ko differentiate karne ke liye use hota hai, especially jab local variable aur field ka naam same ho.

Example:

```
class Car {
    String color;
    Car(String color) {
        this.color = color; // field ko local variable se differentiate karte hain
    }
}
```

35. Static keyword

`static` keyword class level members create karta hai jo objects se independent hote hain. Static fields aur methods directly class ke through access hote hain, object banaye bina.

Example:

```
class Car {
    static int wheels = 4; // static field
    static void display() { // static method
        System.out.println("Wheels: " + wheels);
    }
}
Car.display();
```

36. Method overloading

Method overloading ek class me same method name ke saath different parameters use karna hai. Ye compile-time polymorphism ka example hai.

Example:

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

Yahan `add` method multiple forms me kaam kar raha hai.

37. Final keyword

`final` keyword ka use values, methods, aur classes ko immutable banane ke liye hota hai.

- **Final variable:** Value change nahi ho sakti
- **Final method:** Override nahi ho sakta
- **Final class:** Inherit nahi kiya ja sakta

Example:

```
final class Car { }  // cannot be subclassed
final int MAX_SPEED = 120;  // value cannot change
```

38. Default constructors

Default constructor wo constructor hai jo programmer define na kare to Java automatically provide karta hai. Ye object create karte waqt fields ko default values assign karta hai.

Example:

```
class Car {
    String color;
}
Car myCar = new Car(); // Default constructor call hua
```

39. Parameterized constructors

Parameterized constructor wo constructor hai jo object create karte waqt values accept karta hai aur fields initialize karta hai.

Example:

```
class Car {
    String color;
    Car(String c) { // parameterized constructor
        color = c;
    }
}
Car myCar = new Car("Red"); // color field set ho gaya
```

40. Copy constructors

Copy constructor ek object ke values ko dusre object me copy karne ke liye use hota hai. Java me explicitly define karna padta hai.

Example:

```
class Car {
    String color;
    Car(Car c) { // copy constructor
        this.color = c.color;
    }
}
```

```
Car car1 = new Car("Red");
Car car2 = new Car(car1); // car1 ke values car2 me copy ho gaye
```

11. Passing object as a parameter
    Java me objects ko methods ya constructors me parameter ke roop me pass kar sakte hain. Ye pass-by-reference ki tarah behave karta hai, matlab method me object ke original data ko access ya modify kar sakte hain.
    Example:

```
class Car {
    String color;
    Car(String c) { color = c; }
    void display(Car c) {
        System.out.println(c.color);
    }
}
Car myCar = new Car("Red");
myCar.display(myCar); // object pass hua
```

12. Constructor overloading
    Constructor overloading ek class me multiple constructors define karna hai with different parameters. Ye object creation me flexibility deta hai.
    Example:

```
class Car {
    String color;
    int speed;
    Car() { color = "White"; speed = 0; }          // default constructor
    Car(String c) { color = c; speed = 0; }       // parameterized constructor
    Car(String c, int s) { color = c; speed = s; } // another parameterized constructor
}
```

13. Inheritance
    Inheritance ka matlab hai ek class (child/subclass) dusri class (parent/superclass) ke properties aur methods ko inherit kar sakti hai. Isse code reuse hota hai aur maintain karna easy ho jata hai.
    Example:

```
class Vehicle { int speed; }
class Car extends Vehicle { void display() { System.out.println(speed); } }
```

14. Types of inheritance: single, multiple, multilevel, hierarchical and hybrid inheritance

- **Single inheritance:** Ek child class ek parent class se inherit karti hai.
- **Multiple inheritance (Java me interface ke through):** Ek class multiple interfaces se inherit kar sakti hai.
- **Multilevel inheritance:** Class B inherit karti hai Class A se, aur Class C inherit karti hai Class B se.

- **Hierarchical inheritance:** Multiple child classes ek parent class se inherit karte hain.
- **Hybrid inheritance:** Multiple inheritance types ka combination (Java me interfaces use karke implement hota hai).

15. Concepts of method overriding

    Method overriding me child class parent class ke method ko same signature ke saath redefine karti hai. Ye run-time polymorphism ka example hai aur dynamic behavior allow karta hai.

    Example:

```
class Vehicle {
    void display() { System.out.println("Vehicle"); }
}
class Car extends Vehicle {
    void display() { System.out.println("Car"); } // method override
}
```

16. Extending class

    Extending class ka matlab hai ek class dusri class ke features inherit kare using `**extends**` keyword. Ye parent class ke properties aur methods child class me available kar deta hai.

    Example:

```
class Vehicle { int speed; }
class Car extends Vehicle { void show() { System.out.println(speed); } }
```

17. Super class

    Super class wo class hai jisse child/subclass inherit karta hai. Ye parent class ke properties aur methods provide karti hai jo child class reuse kar sakti hai. Java me `**super**` keyword se parent class members access kiye ja sakte hain.

    Example:

```
class Vehicle { int speed; }
class Car extends Vehicle {
    void show() { System.out.println(super.speed); } // parent class ka field access
}
```

18. Abstract Class

    Abstract class wo class hai jisme kuch methods ka implementation nahi hota (abstract methods) aur kuch methods implemented ho sakte hain. Ye class directly object create nahi kar sakti, sirf inherit karke use hoti hai.

    Example:

```
abstract class Vehicle {
    abstract void start(); // abstract method
    void stop() { System.out.println("Vehicle stopped"); } // regular method
}
```

```
class Car extends Vehicle {
    void start() { System.out.println("Car started"); } // implement abstract method
}
```

49. Creating package

    Package classes aur interfaces ko group karne ka way hai, jisse code organized aur reusable hota hai. Package create karne ke liye `package` keyword use karte hain.

    Example:

```
package myPackage; // package declare
class Car {
    void display() { System.out.println("Car"); }
}
```

50. Importing package

    Importing package ka matlab hai dusre package ki classes ko use karna. Iske liye `import` keyword use hota hai.

    Example:

```
import myPackage.Car; // specific class import
Car c = new Car();
c.display();
```

51. Access rules for packages

    Package ke access rules decide karte hain kaunse classes aur members accessible hain:

- **Public:** Kisi bhi package se access ho sakta hai.
- **Protected:** Same package aur subclasses me accessible.
- **Default (no modifier):** Sirf same package me accessible.
- **Private:** Sirf same class me accessible.

    Ye encapsulation aur modularity maintain karne ke liye important hai.

52. Class hiding rules in a package

    Package me **class hiding** ka matlab hai kisi class ko sirf package ke andar accessible rakhna aur bahar se hide karna. Iske liye **default (no modifier)** access use karte hain. Public classes har package se accessible hoti hain, lekin non-public classes sirf same package me visible hoti hain.

53. Defining interface

    Interface ek blueprint hai jo methods ke signatures define karta hai bina implementation ke. Classes is interface ko implement karke methods ka logic provide karti hain. Interface multiple inheritance allow karne ka safe way hai Java me.

    Example:

```java
interface Vehicle {
    void start();
    void stop();
}
class Car implements Vehicle {
    public void start() { System.out.println("Car started"); }
    public void stop() { System.out.println("Car stopped"); }
}
```

54. Inheritance on interfaces

    Java me **interfaces** bhi inherit ho sakte hain using `extends` keyword. Ek interface dusre interface ke methods inherit karta hai aur child interface unhe implement kar sakta hai. Ye multiple inheritance achieve karne ka safe tarika hai.

    Example:

```java
interface Vehicle { void start(); }
interface ElectricVehicle extends Vehicle { void charge(); }
class Car implements ElectricVehicle {
    public void start() { System.out.println("Car started"); }
    public void charge() { System.out.println("Car charging"); }
}
```

55. Implementing interface

    Implementing interface ka matlab hai ek class interface ke methods ko define karna. Class `implements` keyword use karke interface ke saare abstract methods ka logic provide karti hai.

    Example:

```java
interface Vehicle { void start(); }
class Car implements Vehicle {
    public void start() { System.out.println("Car started"); }
}
```

56. Multiple inheritance using interface

    Java me **multiple inheritance** directly classes ke through nahi hota, lekin **interfaces** ke through possible hai. Ek class ek se zyada interfaces implement karke multiple parent features le sakti hai.

    Example:

```java
interface Vehicle { void start(); }
interface Electric { void charge(); }
class Car implements Vehicle, Electric {
    public void start() { System.out.println("Car started"); }
    public void charge() { System.out.println("Car charging"); }
}
```

57. Introduction
Exception handling ka matlab hai program me errors (exceptions) ko handle karna taki program crash na ho. Java me try-catch-finally blocks aur `throw`/`throws` keywords use hote hain errors ko manage karne ke liye. Ye robust aur error-free programs banane me help karta hai.

58. Built in classes for Exception Handling
Java me **built-in exception classes** predefined hoti hain `java.lang` package me, jo common errors handle karti hain. Examples:

- `ArithmeticException` – division by zero
- `NullPointerException` – null object access
- `ArrayIndexOutOfBoundsException` – invalid array index
- `IOException` – input/output errors

Ye classes program ko safe aur predictable banati hain.

59. Mechanism of Exception Handling in Java
Java me **exception handling** try-catch-finally mechanism se hota hai:

- **try block:** Code jo exception throw kar sakta hai
- **catch block:** Exception handle karne ke liye
- **finally block:** Cleanup code (optional, always execute)

Java me `throw` aur `throws` keywords se exceptions explicitly bhi manage kiye ja sakte hain.

60. Error Handling Exception Classes
Java me **exception classes** do main types ki hoti hain:

- **Checked exceptions:** Compile-time par check hoti hain, jaise `IOException`, `SQLException`. Inhe handle karna mandatory hai.
- **Unchecked exceptions:** Runtime par hoti hain, jaise `ArithmeticException`, `NullPointerException`. Inhe handle karna optional hai.

Ye classes program errors ko manage karne me help karti hain.

61. Creating thread
Java me **thread** create karne ke 2 main ways hain:
62. **Extending Thread class:**

```
class MyThread extends Thread {
    public void run() { System.out.println("Thread running"); }
}
MyThread t = new MyThread();
t.start();
```

2. **Implementing Runnable interface:**

```
class MyRunnable implements Runnable {
    public void run() { System.out.println("Thread running"); }
}
```

```
Thread t = new Thread(new MyRunnable());
t.start();
```

`start()` method thread ko execute karta hai aur `run()` me task define hota hai.

52. Extending Thread class
    Thread create karne ke liye ek class **Thread** ko extend karti hai aur `run()` method override karti hai. Fir `start()` method call karke thread execute hota hai.
    Example:

```
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread is running");
    }
}
MyThread t = new MyThread();
t.start();
```

53. Implementing Runnable interface
    Thread create karne ke liye **Runnable** interface implement karte hain aur `run()` method define karte hain. Fir `Thread` class ka object banakar `start()` call karte hain.
    Example:

```
class MyRunnable implements Runnable {
    public void run() {
        System.out.println("Thread is running");
    }
}
Thread t = new Thread(new MyRunnable());
t.start();
```

54. Life cycle of a thread
    Java me **thread life cycle** me 5 states hoti hain:
55. **New:** Thread object create hota hai.
56. **Runnable:** `start()` call ke baad ready to run.
57. **Running:** CPU thread execute kar raha hai (`run()` method).
58. **Waiting/Blocked:** Thread temporarily wait kar raha hai resource ke liye.
59. **Terminated:** `run()` method complete hone ke baad thread dead ho jata hai.

55. Thread priority & thread synchronization

- **Thread priority:** Har thread ka priority hota hai (1 to 10) jo scheduler ko decide karne me help karta hai kaunsa thread pehle run hoga. `setPriority()` aur `getPriority()` use karte hain.
- **Thread synchronization:** Multiple threads same resource access kar rahe ho to data inconsistency avoid karne ke liye synchronization use hoti hai. `synchronized` keyword se critical section protect hota

hai.

56. Exception handling in threads
    Threads me exceptions handle karne ke liye **try-catch** block `run()` method ke andar use karte hain. Agar exception handle na ho to thread terminate ho sakta hai. Java me `UncaughtExceptionHandler` bhi use karke thread exceptions globally handle kiye ja sakte hain.
    Example:

```
public void run() {
    try {
        int a = 10 / 0;
    } catch (ArithmeticException e) {
        System.out.println("Exception caught: " + e);
    }
}
```