OOP(Java)

Short Notes based on ***Exit Exam***

Zemzem labs

1. Introduction to OOP

What is OOP?

A programming paradigm using **objects** to model real-world entities.

Why OOP?

- Code reusability
- Scalability
- Maintainability
- Better modeling of real systems

Four Pillars of OOP

1. Encapsulation
2. Abstraction
3. Inheritance
4. Polymorphism

2. Classes & Objects

Class

A blueprint or template for creating objects.

Object

An instance of a class containing:

- Data (fields)
- Methods (functions)

Example

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

   void study() {
      System.out.println(name + " is studying.");
```

```
    }
}
```

Object Creation

```
Student s = new Student();
s.name = "John";
s.study();
```

3. Constructors

Special method used to initialize objects.

Types

- **Default Constructor** (provided by Java)
- **No-arg Constructor**
- **Parameterized Constructor**

Example

```
class Student {
    Student(String name) {
        System.out.println("Hello " + name);
    }
}
```

4. Encapsulation

Wrapping **data + methods** inside a class using private fields + public getters/setters.

Example

```
class Bank {
    private double balance;

    public void setBalance(double b) { balance = b; }
    public double getBalance() { return balance; }
}
```

Benefits:

- Data hiding
- Better control
- Increased security

5. Abstraction

Hiding internal complexity and showing only necessary details.

Achieved by

- Abstract classes
- Interfaces

Abstract Class Example

```
abstract class Animal {
   abstract void sound();
}
```

Interface Example

```
interface Vehicle {
   void start();
}
```

6. Inheritance

Mechanism where one class acquires features of another.

Types in Java

- **Single Inheritance**
- **Multilevel Inheritance**
- **Hierarchical Inheritance**

Java does **not** support **multiple inheritance** using classes (only through interfaces).

Example

```
class Animal {}
class Dog extends Animal {}
```

Benefits:

- Code reusability
- Extensibility

7. Polymorphism

"Many forms" — same method behaves differently.

Two Types

1. **Compile-time Polymorphism** (Method Overloading)
2. **Runtime Polymorphism** (Method Overriding)

Method Overloading

Same method name, different parameter list.

```
void add(int a, int b) {}

void add(double a, double b) {}
```

Method Overriding

Defined in subclass with same method signature.

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

8. this Keyword

Refers to current object.

Uses:

- Distinguish instance variables from parameters
- Call another constructor
- Return current object

this.name = name;

9. super Keyword

Refers to parent class.

Uses:

- Access parent constructor
- Access parent variables
- Access parent methods

super.sound();

10. Final Keyword

Java's restriction keyword.

- **final variable:** cannot change
- **final method:** cannot override
- **final class:** cannot extend (e.g., String, Math)

11. Static Keyword

Belongs to class, not object.

- Static variables
- Static methods
- Static blocks

static int count = 0;

12. Packages

Used to organize classes.

- java.lang
- java.util
- java.io

Create a package

package mypack;

Import

import java.util.Scanner;

13. Interfaces (Advanced)

Java 8+ allows:

- Default methods
- Static methods

```
interface A {
   default void show() {}
}
```

A class can implement **multiple interfaces**.

14. Abstract vs Interface (Comparison)

Feature	Abstract Class	Interface
Methods	Abstract + concrete	Abstract + default + static
Variables	Non-final allowed	Must be public static final
Inheritance	Single	Multiple allowed
Use Case	Partial abstraction	Full abstraction

15. Exception Handling

Keywords

- try
- catch
- finally
- throw
- throws

Example

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

16. Object Class Methods

Every class in Java inherits from Object.

Important methods:

- toString()
- equals()
- hashCode()
- clone()
- getClass()

17. Inner Classes

Types:

- Member inner class
- Static inner class
- Local inner class
- Anonymous class

Useful for encapsulation & event handling.

18. Java OOP Design Principles

- **DRY (Don't Repeat Yourself)**
- **KISS (Keep It Simple)**
- **SOLID Principles**
    - Single Responsibility
    - Open/Closed
    - Liskov Substitution
    - Interface Segregation
    - Dependency Inversion

🌟 Ultra-Condensed Exam Summary

- OOP = Encapsulation + Abstraction + Inheritance + Polymorphism
- Class = template; Object = instance
- Constructor initializes objects
- Overloading = compile-time; Overriding = runtime
- Abstract class vs Interface: partial vs full abstraction
- this refers to current object; super refers to parent
- final keyword restricts modification
- static keyword belongs to class
- Java supports single inheritance (classes) + multiple inheritance (interfaces)
- Exceptions: try, catch, finally