📄 OOP (Object Oriented Programming) Note

📁 Index (সূচিপত্র)

- Collections Overview (List, Set, Map)
- Best OOP Practices

7. 📚 Bonus: Real-world OOP Class Examples in Java
    - BankAccount
    - Library Management
    - Employee Management
    - Brand vs Regular Product
    - □ Student Management System
    - □ Vehicle System (Inheritance)
    - □ E-commerce Order System
    - □ Shape Drawing (Polymorphism)
    - □ Printable Interface Example

🔰 পরিচিতি

OOP (Object-Oriented Programming) হল একটি প্রোগ্রামিং কৌশল, যেখানে বাস্তব জীবনের বস্তু (Object) ও তাদের আচরণ (Behavior) কে কোডে রূপান্তর করা হয়।

□ OOP Topics for Beginners

1. Variables and Data Types (ভ্যারিয়েবল ও ডাটা টাইপ)

সংজ্ঞা: ভ্যারিয়েবল হল তথ্য সংরক্ষণের জায়গা। ডাটা টাইপ বলে দেয় কোন ধরনের তথ্য রাখা যাবে।

Java উদাহরণ:

```
int age = 25;
String name = "Rahim";
double salary = 30500.75;
```

---

2. Control Structures (if, switch, loops)

সংজ্ঞা: প্রোগ্রামে নিয়ন্ত্রণ আনতে if, switch, loop ব্যবহৃত হয়।

Java উদাহরণ:

```
if(age > 18) {
    System.out.println("Adult");
}

for(int i = 1; i <= 5; i++) {
    System.out.println("Number: " + i);
}
```

---

3. Functions/Methods (ফাংশন ও মেথড)

সংজ্ঞা: কোডের একটি নির্দিষ্ট অংশকে বারবার ব্যবহারের জন্য ফাংশন বা মেথড তৈরি করা হয়।

Java উদাহরণ:

```
public void greet(String name) {
    System.out.println("Hello, " + name);
}
```

---

4. Classes and Objects (ক্লাস ও অবজেক্ট)

সংজ্ঞা: ক্লাস হল নীলনকশা (blueprint), আর অবজেক্ট হল তার ব্যবহার।

Java উদাহরণ:

```
class Person {
    String name;
    int age;
}

public class Main {
    public static void main(String[] args) {
        Person p = new Person();
        p.name = "Karim";
        p.age = 30;
        System.out.println(p.name + " is " + p.age + " years old.");
    }
}
```

🔑 OOP Core Concepts (4 Pillars)

1. Encapsulation (তথ্য আড়াল করা)

সংজ্ঞা: তথ্য ও মেথড একসাথে ক্লাসে রেখে বাইরের থেকে গোপন রাখা।

Java উদাহরণ:

```
class Account {
    private double balance;

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

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

---

2. Inheritance (উত্তরাধিকার)

সংজ্ঞা: একটি ক্লাস অন্য ক্লাসের বৈশিষ্ট্য গ্রহণ করে।

Java উদাহরণ:

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

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

---

3. Polymorphism (বহুরূপিতা)

সংজ্ঞা: একই মেথড বিভিন্নভাবে কাজ করতে পারে।

Java উদাহরণ:

```
class Shape {
    void draw() {
        System.out.println("Drawing shape");
    }
}

class Circle extends Shape {
    void draw() {
        System.out.println("Drawing circle");
    }
}

class Square extends Shape {
    void draw() {
        System.out.println("Drawing square");
    }
}
```

---

4. Abstraction (মূল অংশ প্রকাশ, জটিলতা লুকানো)

সংজ্ঞা: শুধু দরকারি তথ্য প্রকাশ করে, অপ্রয়োজনীয় অংশ লুকানো।

Java উদাহরণ:

```
abstract class Vehicle {
    abstract void start();
}

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

Interface vs Abstract Class

সংজ্ঞা:

- Interface: শুধু method declaration থাকে।
- Abstract Class: কিছু method define করা যায়, কিছু declare থাকে।

Java উদাহরণ:

```
interface Printable {
    void print();
}

class Document implements Printable {
    public void print() {
        System.out.println("Printing document...");
    }
}
```

🔧 Additional Key OOP Topics

Constructors and Destructors

```
class Student {
    String name;

    Student(String n) {
        name = n;
    }
}
```

---

this Keyword

```
class Employee {
    String name;

    Employee(String name) {
        this.name = name;
    }
}
```

Access Modifiers (public, private, protected)

```
public class Car {
    private String model;
    protected int year;
}
```

Static vs Instance Members

```
class MathUtil {
    static int count = 0; // shared by all
    int id; // separate for each object
}
```

Method Overloading & Overriding

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

class A {
    void show() { System.out.println("A"); }
}

class B extends A {
    void show() { System.out.println("B"); } // overriding
}
```

Getters and Setters

```
class Book {
    private String title;

    public void setTitle(String t) { title = t; }
    public String getTitle() { return title; }
}
```

Composition vs Inheritance

```
class Engine {
    void start() { System.out.println("Engine started"); }
}

class Car {
    Engine engine = new Engine(); // composition
}
```

Exception Handling

```
try {
    int x = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Divide by zero error");
}
```

□ SOLID Principles

1. **RP**: এক ক্লাসের একটিই দায়িত্ব থাকবে
2. **OCP**: ক্লাস পরিবর্তন না করে নতুন বৈশিষ্ট্য যোগ করা যাবে
3. **LSP**: প্যারেন্ট ক্লাসের জায়গায় চাইল্ড ক্লাস ব্যবহারযোগ্য
4. **ISP**: বড় ইন্টারফেস না করে ছোট ছোট ইন্টারফেস
5. **DIP**: High-level মডিউল যেন Low-level-এ নির্ভর না করে

সংক্ষেপে SOLID:

- **S**: Single Responsibility Principle
- **O**: Open/Closed Principle
- **L**: Liskov Substitution Principle
- **I**: Interface Segregation Principle
- **D**: Dependency Inversion Principle

🔢 Advanced Topics (সংক্ষেপে)

- **Enums**: স্থায়ী মানের সেট (e.g. enum Days {MON, TUE})
- **final**: পরিবর্তন করা যায় না
- **Immutable Class**: একবার তৈরি হলে পরিবর্তন অযোগ্য
- **Casting**: এক টাইপকে অন্য টাইপে রূপান্তর
- **Collections**: List, Set, Map – ডাটা সংরক্ষণের কাঠামো

📚 Bonus: Real-world Java Class Example (সংক্ষেপে)

□ Student Management System

```
class Student {
    String name;
    int id;
}
```

```
class StudentManager {
    List<Student> students = new ArrayList<>();

    void addStudent(Student s) {
        students.add(s);
    }
}
```

□ Shape Drawing (Polymorphism)

```
abstract class Shape {
    abstract void draw();
}

class Circle extends Shape {
    void draw() {
        System.out.println("Drawing Circle");
    }
}
```

🕮 Bonus: Real-world OOP Class Examples in Java

1⃣ BankAccount Example

```
class BankAccount {
    private String accountHolder;
    private double balance;

    public BankAccount(String name, double initialBalance) {
        accountHolder = name;
        balance = initialBalance;
    }

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

    public void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
        } else {
            System.out.println("Insufficient funds");
        }
    }

    public void display() {
        System.out.println("Account: " + accountHolder + ", Balance: " +
balance);
    }
}
```

2️⃣ Library Management Example

```
class Book {
    private String title;
    private boolean isAvailable = true;

    public Book(String title) {
        this.title = title;
    }

    public void borrow() {
        if (isAvailable) {
            isAvailable = false;
            System.out.println(title + " borrowed.");
        } else {
            System.out.println(title + " is not available.");
        }
    }

    public void returnBook() {
        isAvailable = true;
        System.out.println(title + " returned.");
    }
}
```

3️⃣ Employee Management System

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

    public Employee(String name, int id) {
        this.name = name;
        this.id = id;
    }

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

class Manager extends Employee {
    private int teamSize;

    public Manager(String name, int id, int teamSize) {
        super(name, id);
        this.teamSize = teamSize;
    }

    public void showDetails() {
        super.showDetails();
        System.out.println("Team Size: " + teamSize);
    }
}
```

4 Brand vs Regular Product

```
class Product {
    protected String name;
    protected double price;

    public Product(String name, double price) {
        this.name = name;
        this.price = price;
    }

    public void showInfo() {
        System.out.println("Product: " + name + ", Price: " + price);
    }
}

class BrandProduct extends Product {
    private String brandName;

    public BrandProduct(String name, double price, String brandName) {
        super(name, price);
        this.brandName = brandName;
    }

    @Override
    public void showInfo() {
        System.out.println("Brand Product: " + brandName + " - " + name +
", Price: " + price);
    }
}
```

5 Student Management System

```
class Student {
    private String name;
    private int roll;
    private double gpa;

    public Student(String name, int roll, double gpa) {
        this.name = name;
        this.roll = roll;
        this.gpa = gpa;
    }

    public void showInfo() {
        System.out.println("Roll: " + roll + ", Name: " + name + ", GPA: "
+ gpa);
    }
}
```

6 Vehicle System (Inheritance Example)

```
class Vehicle {
    protected String model;
```

```
    protected int year;

    public Vehicle(String model, int year) {
        this.model = model;
        this.year = year;
    }

    public void showDetails() {
        System.out.println("Model: " + model + ", Year: " + year);
    }
}

class Car extends Vehicle {
    private int doors;

    public Car(String model, int year, int doors) {
        super(model, year);
        this.doors = doors;
    }

    @Override
    public void showDetails() {
        super.showDetails();
        System.out.println("Doors: " + doors);
    }
}
```

7 E-commerce Order System

```
class Order {
    private int orderId;
    private String customerName;

    public Order(int orderId, String customerName) {
        this.orderId = orderId;
        this.customerName = customerName;
    }

    public void printOrder() {
        System.out.println("Order ID: " + orderId + ", Customer: " +
customerName);
    }
}
```

8 Polymorphism Example: Shape Drawing

```
abstract class Shape {
    abstract void draw();
}

class Circle extends Shape {
    void draw() {
        System.out.println("Drawing Circle...");
    }
}

class Rectangle extends Shape {
    void draw() {
        System.out.println("Drawing Rectangle...");
```

```
    }
}
```

9 Interface Implementation Example: Printable

```
interface Printable {
    void print();
}

class Document implements Printable {
    public void print() {
        System.out.println("Printing Document...");
    }
}

class ImageFile implements Printable {
    public void print() {
        System.out.println("Printing Image...");
    }
}
```
