**WEEK-1: Engineering Concepts**

**Designs and Patterns**

**Exercise 1: Implementing the Singleton Pattern**

**1.Logger.java:**

public class Logger{

    private static Logger instance;

    private Logger(){

        System.out.println("Logger initialized.");

    }

    public static Logger getInstance(){

        if (instance == null) {

            instance = new Logger();

        }

        return instance;

    }

    public void log(String message) {

        System.out.println("[LOG]: " + message);

    } }

**2.TestLogger.java:**

public class TestLogger{

    public static void main(String[] args){

        Logger logger1 = Logger.getInstance();

        logger1.log("First");

        Logger logger2 = Logger.getInstance();

        logger2.log("Second");

        if (logger1 == logger2){

            System.out.println("Both instances are same, Singleton");

        }else{

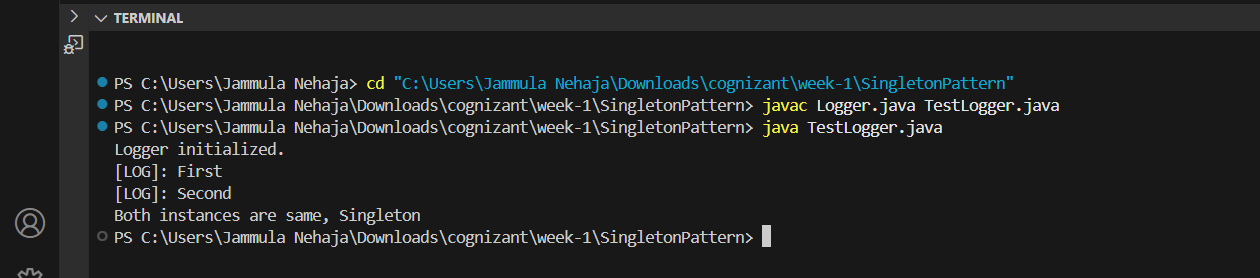
            System.out.println("Different");

        }

    }

}

Output:



**Exercise 2: Implementing the Factory Method Pattern**

1.Document.java:

public interface Document{

    void open();

}

2.WordDoc.java:

public class WordDoc implements Document {

    public void open(){

        System.out.println("Opening word");

    }

}

3.Pdf.java:

public class Pdf implements Document{

    public void open(){

        System.out.println("Opening pdf");

    }

}

4.Excel.java:

public class Excel implements Document {

    public void open(){

        System.out.println("Opening Excel");

    }

}

5.DocFact.java:

public abstract class DocFact{

    public abstract Document createDoc();

}

6.WordFact.java:

public class WordFact extends DocFact{

    public Document createDoc(){

        return new WordDoc();

    }

}

7.pdfFact.java:

public class pdfFact extends DocFact {

    public Document createDoc() {

        return new Pdf();

    }

}

8.ExcelFact.java:

public class pdfFact extends DocFact {

    public Document createDoc() {

        return new Pdf();

    }

}

9.TestFact.java:

public class TestFact {

    public static void main(String[] args){

        DocFact wfact=new WordFact();

        Document wrd=wfact.createDoc();

        wrd.open();

        DocFact pfact=new pdfFact();

        Document pdf=pfact.createDoc();

        pdf.open();

        DocFact exc=new ExcelFact();

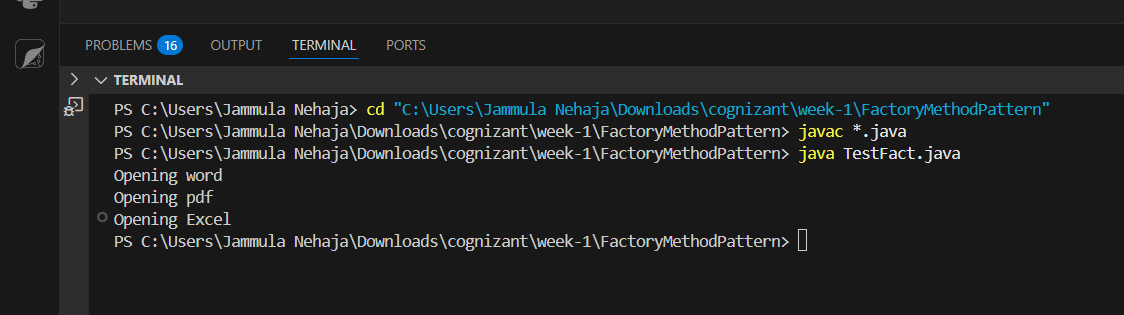
        Document excl=exc.createDoc();

        excl.open();

    }

}

Output:



**Data Structures and Algorithms**

**Exercise 2: E-commerce Platform Search Function**

1.EcommerceSearch.java:

import java.util.Arrays;

class Product {

    int pId;

    String pName;

    String categ;

    public Product(int pId, String pName, String categ) {

        this.pId = pId;

        this.pName = pName;

        this.categ = categ;

    }

    public String toString() {

        return pId + " - " + pName + " - " + categ;

    }

}

class Search {

    public static Product linearSearch(Product[] products, String name) {

        for (Product p : products) {

            if (p.pName.equalsIgnoreCase(name)) {

                return p;

            }

        }

        return null;

    }

    public static Product binarySearch(Product[] products, String name) {

        int low = 0, high = products.length - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

            int cmp = products[mid].pName.compareToIgnoreCase(name);

            if (cmp == 0) return products[mid];

            else if (cmp < 0) low = mid + 1;

            else high = mid - 1;

        }

        return null;

    }

}

public class EcommerceSearch {

    public static void main(String[] args) {

        Product[] products = {

            new Product(1, "Laptop", "Electronics"),

            new Product(2, "Phone", "Electronics"),

            new Product(3, "Shoes", "Fashion"),

        };

        System.out.println("Linear Search:");

        Product result1 = Search.linearSearch(products, "Shoes");

        System.out.println(result1 != null ? result1 : "not found");

        Arrays.sort(products, (a, b) -> a.pName.compareToIgnoreCase(b.pName));

        System.out.println("Binary Search:");

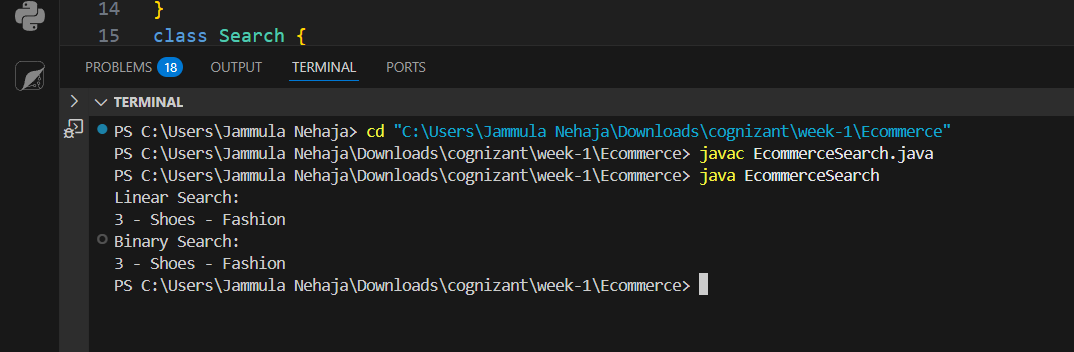
        Product result2 = Search.binarySearch(products, "Shoes");

        System.out.println(result2 != null ? result2 : "not found");

    }

}

Output:



**Exercise 7: Financial Forecasting**

public class FinancialForecast {

    public static double calculateFutureValue(double presentValue, double growthRate, int years) {

        if (years == 0) {

            return presentValue;

        }

        return calculateFutureValue(presentValue, growthRate, years - 1) \* (1 + growthRate);

    }

    public static double calculateFutureValueMemo(double presentValue, double growthRate, int years, double[] memo) {

        if (years == 0) return presentValue;

        if (memo[years] != 0) return memo[years];

        memo[years] = calculateFutureValueMemo(presentValue, growthRate, years - 1, memo) \* (1 + growthRate);

        return memo[years];

    }

    public static void main(String[] args) {

        double presentValue = 10000;

        double growthRate = 0.1;

        int years = 5;

        System.out.println("Recursive Forecast (simple):");

        double futureValue = calculateFutureValue(presentValue, growthRate, years);

        System.out.printf("Future Val after %d yrs = %.2f\n", years, futureValue);

        System.out.println("\nRec Forecast(memoization):");

        double[] memo = new double[years + 1];

        double optimizedValue = calculateFutureValueMemo(presentValue, growthRate, years, memo);

        System.out.printf("Future Val after %d yrs = %.2f\n", years, optimizedValue);

    }

}

Output:

