**Exercise 1: Implementing the Singleton Pattern**

**Scenario:**

You need to ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

**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);

    }

}

**Main.java**

public class Main {

    public static void main(String[] args) {

        Logger logger1 = Logger.getInstance();

        logger1.log("First message");

        Logger logger2 = Logger.getInstance();

        logger2.log("Second message");

        if (logger1 == logger2) {

            System.out.println("Singleton confirmed: Only one instance exists.");

        } else {

            System.out.println("Singleton failed: Multiple instances found.");

        }

    }

}

**OUTPUT:**

****

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

**Scenario:**

You are developing a document management system that needs to create different types of documents (e.g., Word, PDF, Excel). Use the Factory Method Pattern to achieve this.

**FactoryMethodPattern.java**

interface Document {

    void open();

}

class WordDocument implements Document {

    public void open() {

        System.out.println("Opening Word Document...");

    }

}

class PdfDocument implements Document {

    public void open() {

        System.out.println("Opening PDF Document...");

}

}

class ExcelDocument implements Document {

    public void open() {

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

    }

}

abstract class DocumentFactory {

    public abstract Document createDocument();

}

class WordDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new WordDocument();

    }

}

class PdfDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new PdfDocument();

    }

}

class ExcelDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new ExcelDocument();

    }

}

public class FactoryMethodPatternExample {

    public static void main(String[] args) {

        DocumentFactory wordFactory = new WordDocumentFactory();

        Document wordDoc = wordFactory.createDocument();

        wordDoc.open();

        DocumentFactory pdfFactory = new PdfDocumentFactory();

        Document pdfDoc = pdfFactory.createDocument();

        pdfDoc.open();

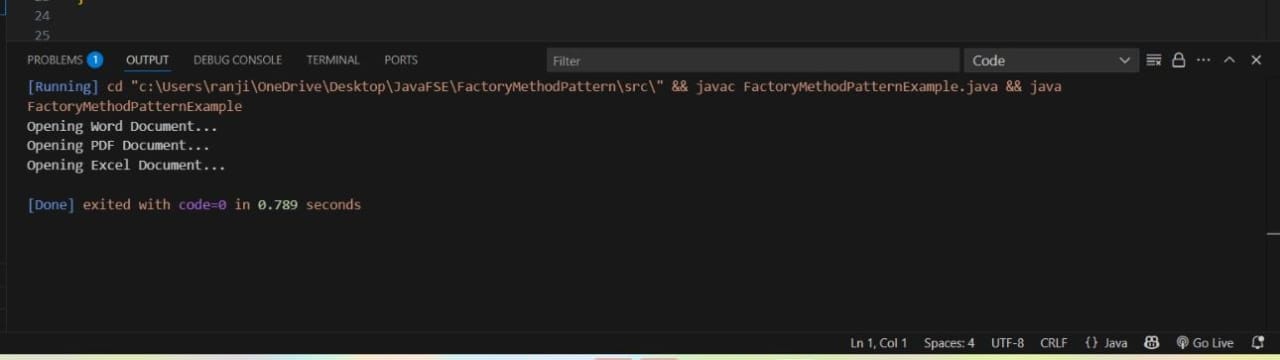
        DocumentFactory excelFactory = new ExcelDocumentFactory();

        Document excelDoc = excelFactory.createDocument();

        excelDoc.open();

    }

}

****

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

**Scenario:**

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance

**EcommerceSearch.java**

import java.util.Arrays;

import java.util.Comparator;

class Product {

    int productId;

    String productName;

    String category;

    Product(int id, String name, String category) {

        this.productId = id;

        this.productName = name;

        this.category = category;

    }

    @Override

    public String toString() {

        return "Product ID: " + productId + ", Name: " + productName + ", Category: " + category;

    }

}

public class EcommerceSearch {

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

        for (Product product : products) {

            if (product.productName.equalsIgnoreCase(targetName)) {

                return product;

            }

        }

        return null;

    }

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

        int left = 0;

        int right = products.length - 1;

        while (left <= right) {

            int mid = (left + right) / 2;

            int comparison = products[mid].productName.compareToIgnoreCase(targetName);

            if (comparison == 0) {

                return products[mid];

            } else if (comparison < 0) {

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        return null;

    }

    public static void main(String[] args) {

        Product[] products = {

            new Product(101, "Laptop", "Electronics"),

            new Product(102, "Mobile", "Electronics"),

            new Product(103, "Shoes", "Footwear"),

            new Product(104, "Watch", "Accessories"),

            new Product(105, "Headphones", "Electronics")

        };

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

        Product result1 = linearSearch(products, "Watch");

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

        System.out.println("\n---- Binary Search ----");

        Arrays.sort(products, Comparator.comparing(p -> p.productName));

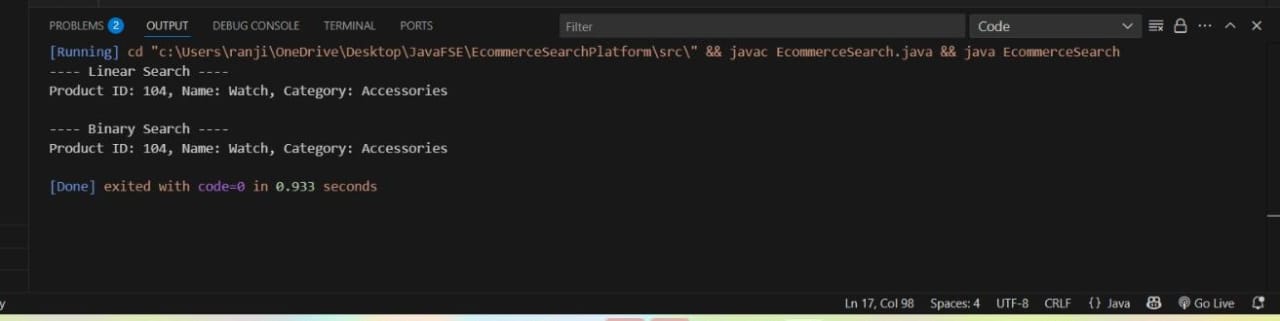
        Product result2 = binarySearch(products, "Watch");

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

    }

}

**OUTPUT**



**Exercise 7: Financial Forecasting**

**Scenario:**

You are developing a financial forecasting tool that predicts future values based on past data.

**FinancialForcasting.java**

public class FinancialForecast {

    public static double forecast(double currentValue, double growthRate, int years) {

        if (years == 0)

            return currentValue;

        return forecast(currentValue \* (1 + growthRate), growthRate, years - 1);

    }

    public static void main(String[] args) {

        double currentValue = 10000;

        double growthRate = 0.08;

        int years = 5;

        double futureValue = forecast(currentValue, growthRate, years);

        System.out.printf("Predicted value after %d years: ₹%.2f\n", years, futureValue);

    }

}

**OUTPUT**

