WEEK-1

DESIGN PATTERNS AND PRINCIPLES

1. SINGLETON PATTERN:

CODE

public class Logger {

    private static Logger instance;

    private Logger() {

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

    }

    public static Logger getInstance() {

        if (instance == null) {

            instance = new Logger();

        }

        return instance;

    }

    public void log(String message) {

        System.out.println("Log: " + message);

    }

}

TEST CLASS:

public class LoggerTest {

   public LoggerTest() {

   }

   public static void main(String[] var0) {

      Logger var1 = Logger.getInstance();

      Logger var2 = Logger.getInstance();

      var2.log("Hello from logger2");

      if (var1 == var2) {

         System.out.println("Both loggers are the same instance.");

      } else {

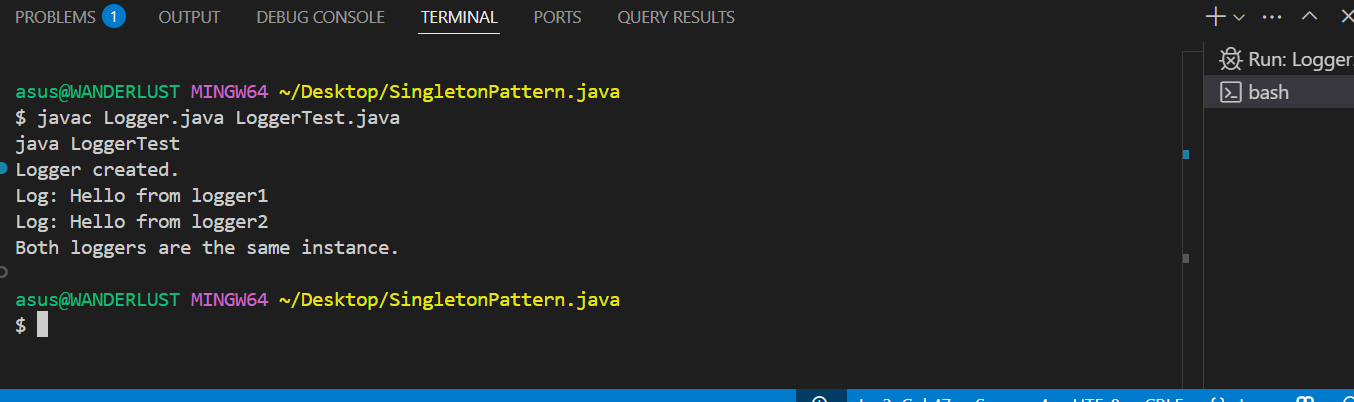
         System.out.println("Different instances detected.");

      }

   }

}

OUTPUT:



1. FACTORY METHOD:

CODE

// Interface

interface Document {

    void open();

}

// Concrete Document Classes

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 Factory

abstract class DocumentFactory {

    public abstract Document createDocument();

}

// Concrete Factories

class WordFactory extends DocumentFactory {

    public Document createDocument() {

        return new WordDocument();

    }

}

class PdfFactory extends DocumentFactory {

    public Document createDocument() {

        return new PdfDocument();

    }

}

class ExcelFactory extends DocumentFactory {

    public Document createDocument() {

        return new ExcelDocument();

    }

}

// Main Class

public class FactoryPatternDemo {

    public static void main(String[] args) {

        DocumentFactory factory;

        factory = new WordFactory();

        Document word = factory.createDocument();

        word.open();

        factory = new PdfFactory();

        Document pdf = factory.createDocument();

        pdf.open();

        factory = new ExcelFactory();

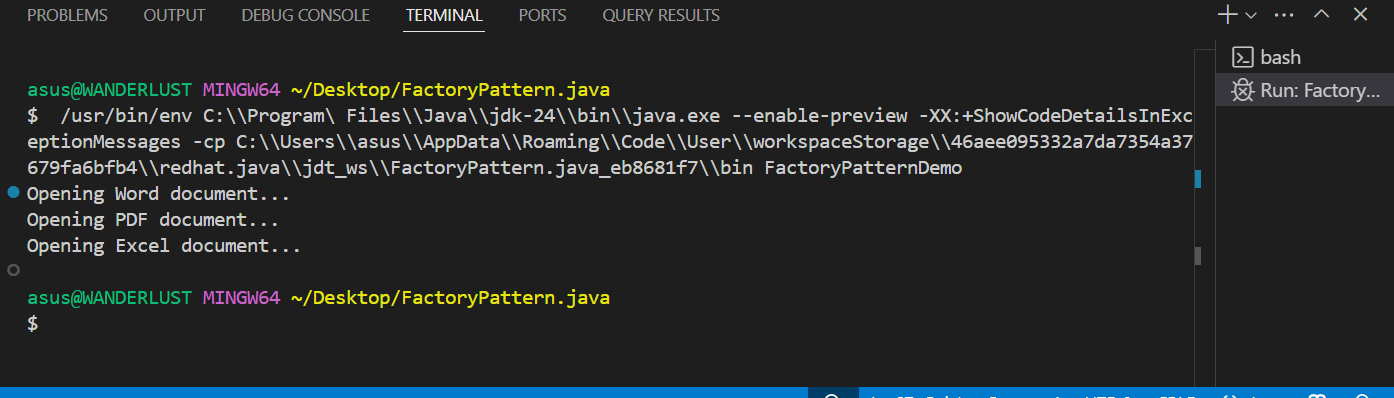
        Document excel = factory.createDocument();

        excel.open();

    }

}

OUTPUT:



DATASTRUCTURES AND ALGORITHM

1. E-commerce platform search function

CODE:

import java.util.Arrays;

import java.util.Comparator;

class Product {

    int productId;

    String productName;

    String category;

    public Product(int productId, String productName, String category) {

        this.productId = productId;

        this.productName = productName;

        this.category = category;

    }

    @Override

    public String toString() {

        return "Product ID: " + productId +

               ", Name: " + productName +

               ", Category: " + category;

    }

}

public class SearchFunction {

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

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

            else if (cmp < 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, "Shoes", "Fashion"),

            new Product(103, "Mobile", "Electronics"),

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

        };

        Product resultLinear = linearSearch(products, "Shoes");

        if (resultLinear != null) {

            System.out.println("Linear Search Found: " + resultLinear);

        } else {

            System.out.println("Product not found using Linear Search");

        }

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

        Product resultBinary = binarySearch(products, "Shoes");

        if (resultBinary != null) {

            System.out.println("Binary Search Found: " + resultBinary);

        } else {

            System.out.println("Product not found using Binary Search");

        }

    }

}

OUTPUT:



1. FINANCIAL FORECASTING

CODE:

public class FinancialForecast {

    // Recursive method to forecast future value

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

        if (years == 0) {

            return currentValue;

        } else {

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

        }

    }

    public static void main(String[] args) {

        double startingValue = 10000.0; // Initial amount

        double annualGrowthRate = 0.08; // 8% growth per year

        int numberOfYears = 5;

        double predictedValue = forecastValue(startingValue, annualGrowthRate, numberOfYears);

        System.out.printf("Forecasted Value after %d years: %.2f\n", numberOfYears, predictedValue);

    }

}

OUTPUT:

