**Engineering concepts**

**Design Patterns and Principles.docx**

**Exercise 1: Implementing the Singleton Pattern**

**Code** :

public class SingletonPattern {

      static class Logger {

        private static Logger instance;

        private Logger() {

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

        }

        public static Logger getInstance() {

            if (instance == null) {

                instance = new Logger();

            }

            return instance;

        }

        public void log(String message) {

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

        }

    }

public static void main(String[] args) {

        Logger logger1 = Logger.getInstance();

        logger1.log("This is the first log message.");

        Logger logger2 = Logger.getInstance();

        logger2.log("This is the second log message.");

        if (logger1 == logger2) {

            System.out.println("Both logger instances are the same (Singleton confirmed).");

        } else {

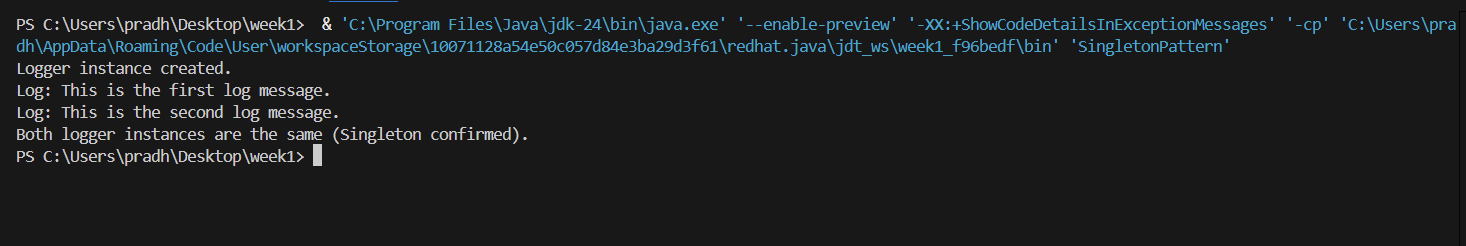
            System.out.println("Different logger instances (Singleton violated).");

        }

    }

}

**Output** :



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

**Code:**

interface Document {

    void open();

}

class WordDocument implements Document {

    @Override

    public void open() {

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

    }

}

class PdfDocument implements Document {

    @Override

    public void open() {

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

    }

}

class ExcelDocument implements Document {

    @Override

    public void open() {

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

    }

}

abstract class DocumentFactory {

    public abstract Document createDocument();

}

class WordDocumentFactory extends DocumentFactory {

    @Override

    public Document createDocument() {

        return new WordDocument();

    }

}

class PdfDocumentFactory extends DocumentFactory {

    @Override

    public Document createDocument() {

        return new PdfDocument();

    }

}

class ExcelDocumentFactory extends DocumentFactory {

    @Override

    public Document createDocument() {

        return new ExcelDocument();

    }

}

public class FactoryMethodPatternExample {

    public static void main(String[] args) {

        DocumentFactory wordFactory = new WordDocumentFactory();

        Document word = wordFactory.createDocument();

        word.open();

        DocumentFactory pdfFactory = new PdfDocumentFactory();

        Document pdf = pdfFactory.createDocument();

        pdf.open();

        DocumentFactory excelFactory = new ExcelDocumentFactory();

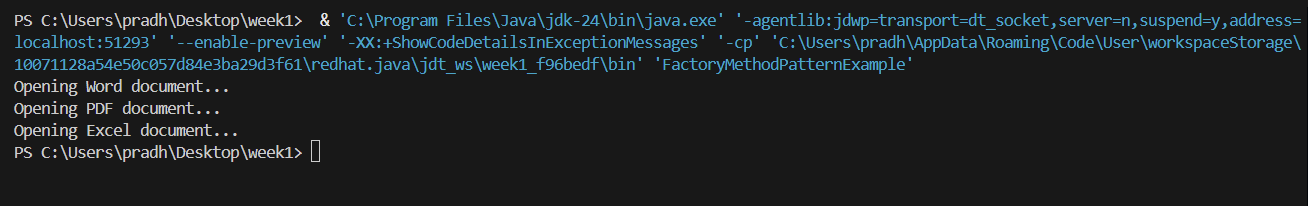
        Document excel = excelFactory.createDocument();

        excel.open();

    }

}

**Output :**



**Data structures and Algorithms**

**Exercise 2 : 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 "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 low = 0, high = products.length - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

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

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

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

            else high = mid - 1;

        }

        return null;

    }

    public static void main(String[] args) {

        Product[] products = {

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

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

            new Product(103, "Shirt", "Clothing"),

            new Product(104, "Shoes", "Footwear"),

            new Product(105, "Book", "Stationery")

        };

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

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

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

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

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

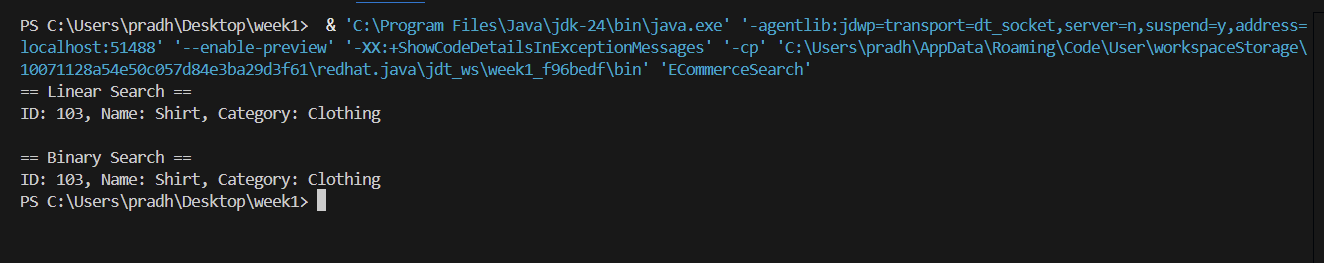
        Product result2 = binarySearch(products, "Shirt");

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

    }

}

**Output :**



**Exercise 7: Financial Forecasting**

**Code :**

public class FinancialForecast {

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

                if (years == 0) {

            return presentValue;

        }

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

    }

    public static void main(String[] args) {

        double presentValue = 1000.0; // Initial investment

        double growthRate = 0.05;     // 5% annual growth

        int years = 5;                // Forecast for 5 years

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

        System.out.printf("Future value after %d years: %.2f\n", years, result);

    }

}

**Output :**

