Candidate Name:- SUCHITHRA M S

Superset ID:-6430920

WEEK – 1 HANDS ON EXERCISE (JAVA FSE DEEPSKILLING)

(DESIGN PATTERN AND PRINCIPLES & ALGORITHM DATA STRUCTURES)

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.

Steps:

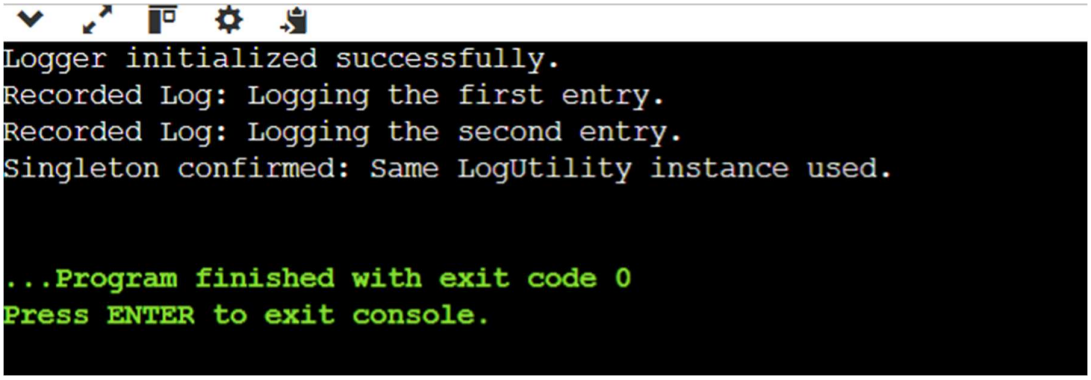
1. Create a new Java project named SingletonPatternExample.
2. Define a Singleton class.
3. Implement the Singleton pattern.
4. Test the Singleton implementation.

Code for the above question:

```
public class LoggerTest {
    public static void main(String[] args) {
        LogUtility firstLog = LogUtility.getInstance();
        firstLog.addLog("Logging the first entry.");

        LogUtility secondLog = LogUtility.getInstance();
        secondLog.addLog("Logging the second entry.");

        if (firstLog == secondLog) {
            System.out.println("Singleton confirmed: Same LogUtility instance used.");
        } else {
            System.out.println("Error: Different LogUtility instances exist.");
        }
    }
```

```
}

class LogUtility {
    private static LogUtility uniqueInstance;

    private LogUtility() {
        System.out.println("Logger initialized successfully.");
    }

    public static LogUtility getInstance() {
        if (uniqueInstance == null) {
            uniqueInstance = new LogUtility();
        }
        return uniqueInstance;
    }

    public void addLog(String message) {
        System.out.println("Recorded Log: " + message);
    }
}
```

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.

Steps:

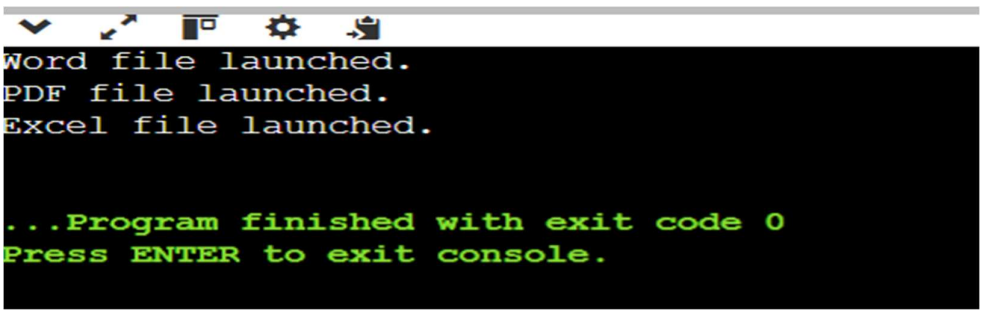
1. Create a new Java project named FactoryMethodPatternExample.
2. Define document classes and factory method.
3. Test the Factory Method implementation.

Code for the above question:

```
public class DocumentTester {
    public static void main(String[] args) {
        Creator wordGen = new WordFactory();
        Creator pdfGen = new PdfFactory();
        Creator excelGen = new ExcelFactory();

        Document word = wordGen.create();
        Document pdf = pdfGen.create();
        Document excel = excelGen.create();

        word.open();
        pdf.open();
        excel.open();
    }
}

interface Document {
    void open();
}

class WordFile implements Document {
    public void open() {
        System.out.println("Word file launched.");
    }
}

class PdfFile implements Document {
    public void open() {
        System.out.println("PDF file launched.");
    }
}
```

```
class ExcelFile implements Document {
    public void open() {
        System.out.println("Excel file launched.");
    }
}

abstract class Creator {
    public abstract Document create();
}

class WordFactory extends Creator {
    public Document create() {
        return new WordFile();
    }
}

class PdfFactory extends Creator {
    public Document create() {
        return new PdfFile();
    }
}

class ExcelFactory extends Creator {
    public Document create() {
        return new ExcelFile();
    }
}
```

Output:



Exercise 7: Financial Forecasting

Scenario:

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

Steps:

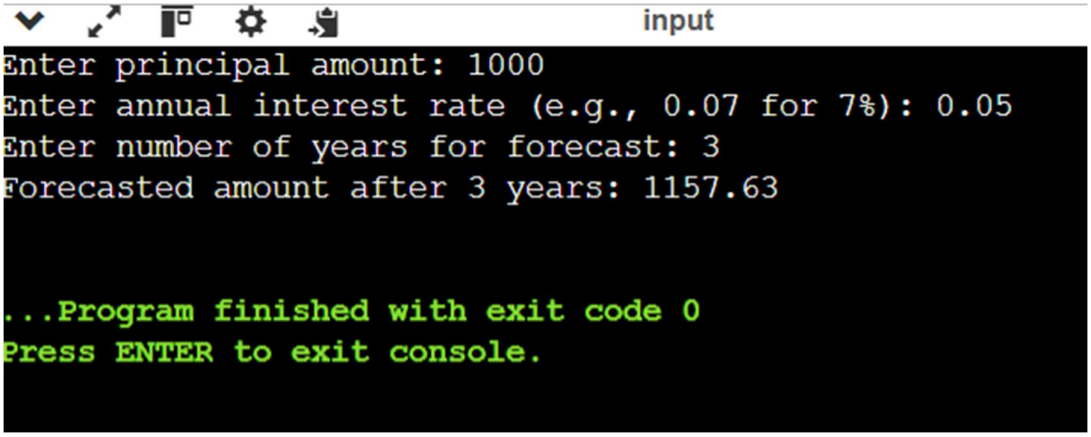
1. Understand recursion.
2. Implement a recursive algorithm for forecasting.
3. Analyze complexity and optimize.

Code for above question:

```
import java.util.Scanner;


public class FinanceForecast {
   public static double forecastValue(double principal, double rate, int years) {
      if (years == 0) {
         return principal;
      }
      return forecastValue(principal, rate, years - 1) * (1 + rate);
   }

   public static void main(String[] args) {
      Scanner input = new Scanner(System.in);
      System.out.print("Enter principal amount: ");
      double principal = input.nextDouble();

      System.out.print("Enter annual interest rate (e.g., 0.07 for 7%): ");
      double rate = input.nextDouble();

      System.out.print("Enter number of years for forecast: ");
      int years = input.nextInt();

      double projected = forecastValue(principal, rate, years);
      System.out.printf("Forecasted amount after %d years: %.2f\n", years, projected);
```

```
    }
}
```

Output:



Exercise 8: 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.

Steps:

1. Implement linear and binary search.
2. Compare performance.

Code for above question:

```
import java.util.Arrays;

class Product implements Comparable<Product> {
    int productId;
    String productName;
    String category;

    public Product(int productId, String productName, String category) {
        this.productId = productId;
        this.productName = productName;
        this.category = category;
    }
```

```
    public String toString() {
        return productId + " | " + productName + " (" + category + ")";
    }

    public int compareTo(Product other) {
        return Integer.compare(this.productId, other.productId);
    }
}

public class SearchDemo {
    public static void main(String[] args) {
        Product[] catalog = {
            new Product(103, "Laptop", "Electronics"),
            new Product(101, "Shirt", "Apparel"),
            new Product(105, "Headphones", "Electronics"),
            new Product(102, "Book", "Education"),
            new Product(104, "Shoes", "Footwear")
        };

        System.out.println("Linear Search:");
        Product foundLinear = findLinear(catalog, 102);
        System.out.println(foundLinear != null ? "Found: " + foundLinear : "Not found.");

        Arrays.sort(catalog);
        System.out.println("Binary Search:");
        Product foundBinary = findBinary(catalog, 104);
        System.out.println(foundBinary != null ? "Found: " + foundBinary : "Not found.");
    }

    public static Product findLinear(Product[] list, int id) {
        for (Product p : list) {
            if (p.productId == id) {
                return p;
            }
        }
        return null;
    }
```

```
    public static Product findBinary(Product[] list, int id) {
        int low = 0, high = list.length - 1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (list[mid].productId == id) {
                return list[mid];
            } else if (list[mid].productId < id) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return null;
    }
}
```

Output: