Name:-Ritesh Kumar Sahoo

GitHub Link:- <https://github.com/Riteshksahoo/CognizantDotNet>CognizantDotNet

**-----------------------------------------------------------------------------------------**

**Exercise 1: Implementing the Singleton Pattern**

**Logger.java:-**

public class Logger {

private static Logger singleInstance;

private Logger() {

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

}

public static Logger getInstance() {

if (singleInstance == null) {

singleInstance = new Logger();

}

return singleInstance;

}

public void log(String message) {

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

}

}

**Main.java :-**

public class Main {

public static void main(String[] args) {

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

System.out.println("Wait....");

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

logger1.log("Starting the application....");

logger2.log("Continuing the logging....");

if (logger1 == logger2) {

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

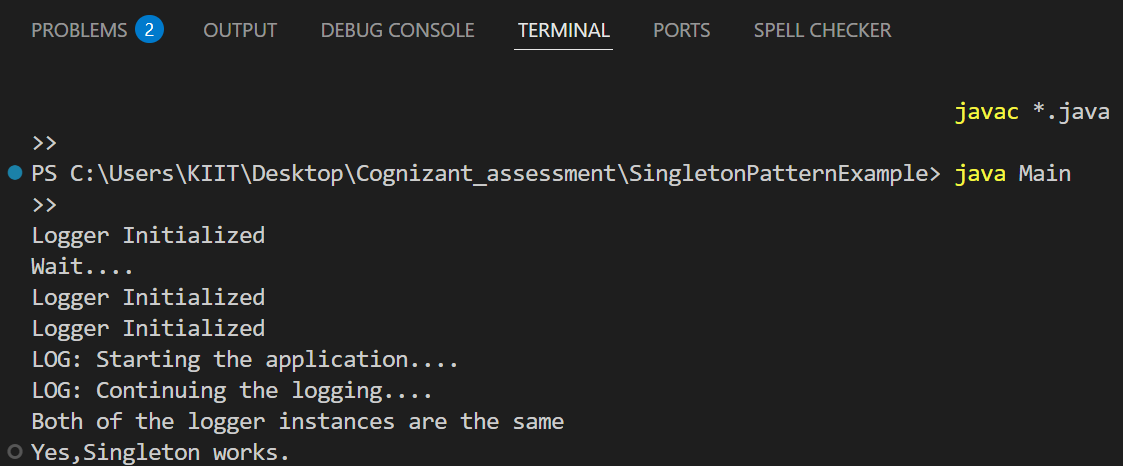
System.err.println("Yes,Singleton works.");

} else {

System.out.println("Logger of the instances are different (Singleton failed).");

}

}

}  
**OUTPUT:-**

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

**Document.java:-**

package FactoryMethodPatternExample;

public interface Document {

void open();

}

**DocumentFactory.java**

package FactoryMethodPatternExample;

public abstract class DocumentFactory {

public abstract Document createDocument();

}  
**ExcelDocument.java**

package FactoryMethodPatternExample;

public class ExcelDocument implements Document {

@Override

public void open() {

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

}

}  
**ExcelFactory.java**package FactoryMethodPatternExample;

public class ExcelFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

**Main.java**package FactoryMethodPatternExample;

public class Main {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordFactory();

Document wordDoc = wordFactory.createDocument();

wordDoc.open();

DocumentFactory pdfFactory = new PdfFactory();

Document pdfDoc = pdfFactory.createDocument();

pdfDoc.open();

DocumentFactory excelFactory = new ExcelFactory();

Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}  
**PdfDocument.java**package FactoryMethodPatternExample;

public class PdfDocument implements Document {

@Override

public void open() {

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

}

}

**PdfFactory.java**package FactoryMethodPatternExample;

public class PdfFactory extends DocumentFactory {

@Override

public Document createDocument()

{

return new PdfDocument();

}

}  
**WordDocument.java**package FactoryMethodPatternExample;

public class WordDocument implements Document {

@Override

public void open() {

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

}

}  
**WordFactory.java**package FactoryMethodPatternExample;

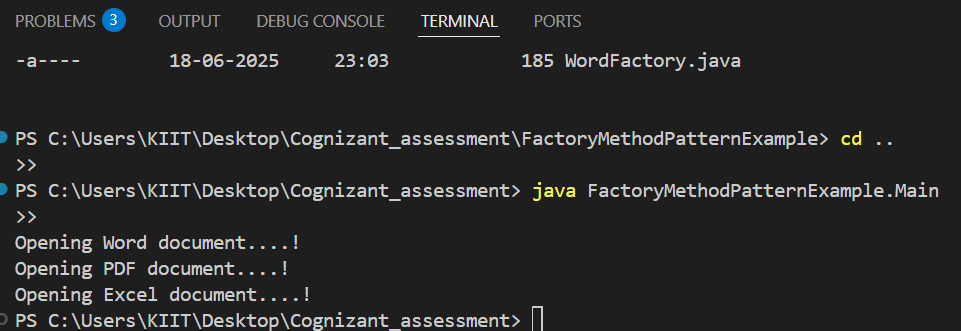
public class WordFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}  
**OUTUT**

**DATA STRUCTURE AND ALGORITHM  
  
Exercise 2: E-commerce Platform Search Function  
  
EcommerceSearch.java:-**package EcommerceSearch;

import java.util.Arrays;

import java.util.Comparator;

public class ECommerceSearch {

public static Product linearSearch(Product[] products, int targetId) {

for (Product product : products) {

if (product.productId == targetId) {

return product;

}

}

return null;

}

public static Product binarySearch(Product[] products, int targetId) {

int left = 0;

int right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

if (products[mid].productId == targetId) {

return products[mid];

} else if (products[mid].productId < targetId) {

left = mid + 1;

}

else {

right = mid - 1;

}

}

return null;

}

public static void sortProductsById(Product[] products) {

Arrays.sort(products, Comparator.comparingInt(p -> p.productId));

}

}

**Product.java:-**package EcommerceSearch;

public 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;

}

}

**Main.java:-**

package EcommerceSearch;

public class Main {

public static void main(String[] args) {

Product[] products = {

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

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

new Product(105, "Shoes", "Fashion"),

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

new Product(104, "T-shirt", "Fashion")

};

int targetId = 102;

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

Product result1 = ECommerceSearch.linearSearch(products, targetId);

printResult(result1);

ECommerceSearch.sortProductsById(products);

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

Product result2 = ECommerceSearch.binarySearch(products, targetId);

printResult(result2);

}

private static void printResult(Product p) {

if (p != null) {

System.out.println("Found Product: " + p.productName +

" (ID: " + p.productId + ", Category: " + p.category + ")");

}

else {

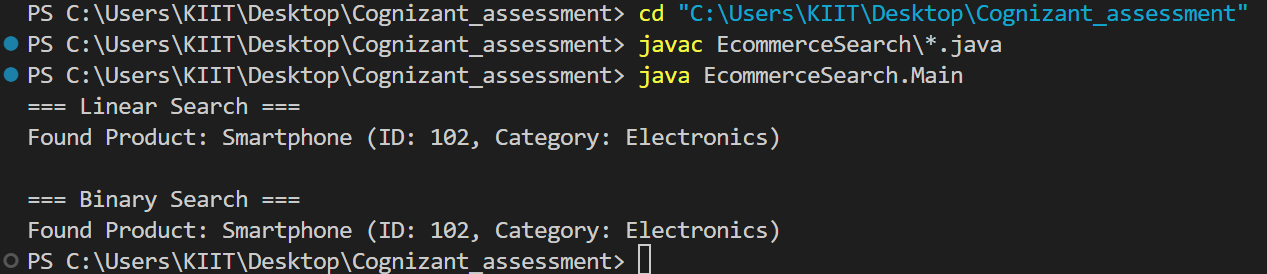
System.out.println("Product not found.");

}

}

}

**OUTPUT:-**



**Exercise 7: Financial Forecasting**  
  
 **FinancialForecasting.java:-**

import java.util.Scanner;

public class FinancialForecasting {

public static double calculateFutureValue(double initialValue, double rate, int years) {

if (years == 0) {

return initialValue;

}

return calculateFutureValue(initialValue, rate, years - 1) \* (1 + rate);

}

public static double calculateFutureValueMemo(double initialValue, double rate, int years, double[] memo) {

if (years == 0) {

return initialValue;

}

if (memo[years] != 0) {

return memo[years];

}

memo[years] = calculateFutureValueMemo(initialValue, rate, years - 1, memo) \* (1 + rate);

return memo[years];

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter Initial Investment Value: ");

double initialValue = sc.nextDouble();

System.out.print("Enter Annual Growth Rate (in %): ");

double ratePercent = sc.nextDouble();

System.out.print("Enter Number of Years: ");

int years = sc.nextInt();

double rate = ratePercent / 100.0;

double futureValue = calculateFutureValue(initialValue, rate, years);

System.out.printf("Future Value (Recursive): %.2f\n", futureValue);

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

double futureValueMemo = calculateFutureValueMemo(initialValue, rate, years, memo);

System.out.printf("Future Value (Memoized): %.2f\n", futureValueMemo);

sc.close();

}

}  
**OUTPUT:-**