**Digital Nurture 4.0 – Week 1**

**1.Design principles & Patterns:**

**Exercise 1: Implementing the Singleton Pattern**:

**Program:**

I create a project and package named **SingletonPatternExample** and a seperate test class to verify that only one instance of Logger is created and used across the application.

**Note:** Kindly Ignore the package name.

**test.java:**

package SingletonPatternExample;

class Logger {

private static Logger instance = new Logger();

private Logger() {

System.out.println("Logger was Created!!");

}

public static Logger getInstance() {

return instance;

}

public void log(String message) {

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

}

}

public class test {

public static void main(String[] args) {

Logger l1 = Logger.getInstance();

Logger l2 = Logger.getInstance();

l1.log("This is the First message");

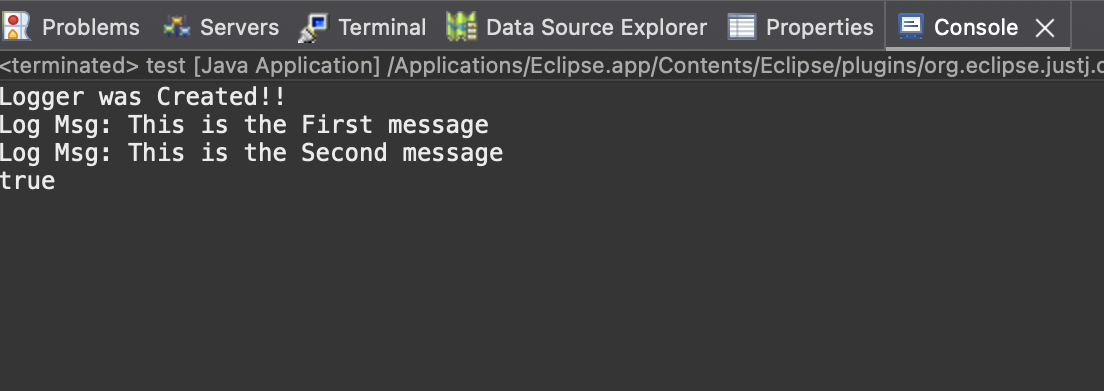
l2.log("This is the Second message");

System.out.println(l1==l2);

}

}

**Output:**



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

**Program:**

Here,I create a project and package named **FactoryMethodPatternExample**.Then I create an interface,Document classes,Concrete classes and Factory Classes with methods in the common class called Main.

Finally In test class creation of different document types using the factory method was verified.

**Note:** Kindly Ignore the package name.

**test.java:**

package FactoryMethodPatternExample;

class Main {

public interface Documents {

void open();

}

public static class WordDocument implements Documents {

public void open() {

System.out.println("Word Document Opened!");

}

}

public static class PdfDocument implements Documents {

public void open() {

System.out.println("Pdf Document Opened!");

}

}

public static class ExcelDocument implements Documents {

public void open() {

System.out.println("Excel Document Opened!");

}

}

public static abstract class DocumentFactory {

public abstract Documents createDocument();

}

public static class WordFactory extends DocumentFactory {

public Documents createDocument() {

return new WordDocument();

}

}

public static class PdfFactory extends DocumentFactory {

public Documents createDocument() {

return new PdfDocument();

}

}

public static class ExcelFactory extends DocumentFactory {

public Documents createDocument() {

return new ExcelDocument();

}

}

}

public class test {

public static void main(String[] args) {

Main.DocumentFactory wordFactory = new Main.WordFactory();

Main.Documents word = wordFactory.createDocument();

word.open();

Main.DocumentFactory pdfFactory = new Main.PdfFactory();

Main.Documents pdf = pdfFactory.createDocument();

pdf.open();

Main.DocumentFactory excelFactory = new Main.ExcelFactory();

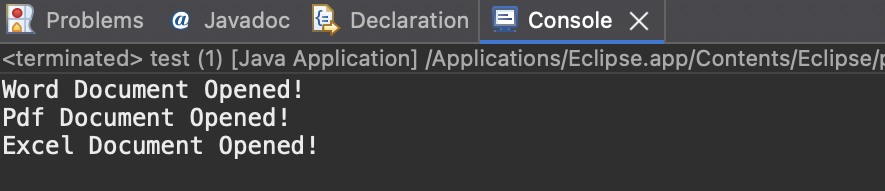
Main.Documents excel = excelFactory.createDocument();

excel.open();

}

}

**Output:**



**2.Data structures and Algorithms:**

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

**Program:**

Here, I created a main class named ProductSearch for build and test the program.Inside the main class I create a static class named product with productId,productName and category as they mentioned in the setup.

**ProductSearch.java:**

import java.util.\*;

public class ProductSearch {

static 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 static Product linearSearch(Product[] products, String target) {

for (Product n : products) {

if (n.productName.equalsIgnoreCase(target)) {

return n;

}

}

return null;

}

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

int low = 0, high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

int comp = products[mid].productName.compareToIgnoreCase(target);

if (comp == 0) {

return products[mid];

} else if (comp < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

new Product(1, "Laptop", "Electronics"),

new Product(2, "Shirt", "Clothing"),

new Product(3, "Bike", "Vehicle"),

new Product(4, "Fans", "TV & Appliances"),

new Product(5, "Watch", "Accessories")};

String Target1 = "Watch";

Product Result1 = linearSearch(products, Target1);

System.out.println("By Linear Search = " + (Result1 != null ? Result1 : "Product not available"));

String Target2 = "Shirt";

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

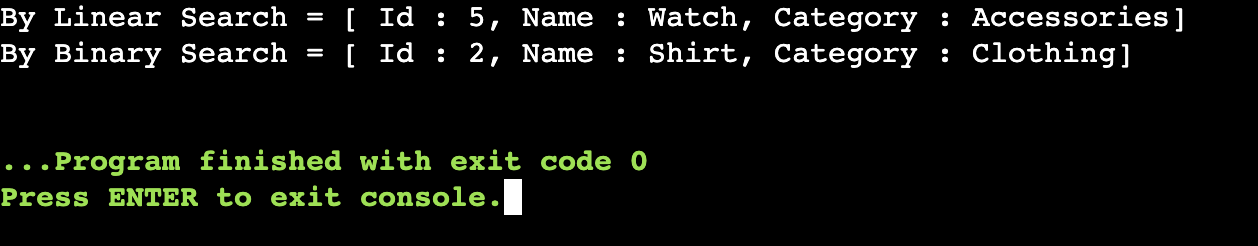
Product Result2 = binarySearch(products, Target2);

System.out.println("By Binary Search = " + (Result2 != null ? Result2 : "Product not available"));

}

}

**Output:**



**Exercise 7: Financial Forecasting:**

**Program:**

Here, I create a main class named FinancialForecastValue.It will calculate the Future Prediction Value by Investment,Annual Growth Rate, and Number of Years.

The formula is FPV(n) = FPV(n-1)\*(1+r)

FPV(n) = Future Prediction Value of n years

r = Growth Rate

**FinancialForcastValue.java:**

import java.util.Scanner;

public class FinancialForecastValue {

public static double predictValue(double initial,int yrs,double rate) {

if (yrs == 0) {

return initial;

}

return predictValue(initial,yrs - 1,rate) \* (1+rate);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter your initial investment amount : ");

double initialVal = scanner.nextDouble();

System.out.print("Enter your annual growth rate (Range 0 to 1 for 0% to 100%) : ");

double rate = scanner.nextDouble();

System.out.print("Enter number of years : ");

int yrs = scanner.nextInt();

double finalResult = predictValue(initialVal,yrs,rate);

System.out.printf("The predicted Future value after %d years is %.2f%n", yrs, finalResult);

}

}

**Output:**

