WEEK -1

DESIGNS AND PRINCIPLES:

1.SingletonPatternExample

**Logger.java:**

public class Logger {

private static Logger *instance*;

private Logger() {

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

}

public static Logger getInstance() {

if (*instance* == null) {

*instance* = new Logger();

}

return *instance*;

}

public void log(String message) {

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

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

Logger logger1 = Logger.*getInstance*();

Logger logger2 = Logger.*getInstance*();

logger1.log("First message");

logger2.log("Second message");

if (logger1 == logger2) {

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

} else {

System.***out***.println("Different logger instances found.");

}

}

}

2.Factory Method

**Document.java:**

public interface Document {

void open();

}

**DocumentFactory.java:**

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**ExcelDocument.java:**

public class ExcelDocument implements Document{

*@Override*

public void open() {

System.***out***.println("Opening Excel Document...");

}

}

**ExcelDocumentFactory.java:**

public class ExcelDocumentFactory extends DocumentFactory {

*@Override*

public Document createDocument() {

return new ExcelDocument();

}

}

**Main.java:**

public class Main {

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();

}

}

**PdfDocument.java:**

public class PdfDocument implements Document{

*@Override*

public void open() {

System.***out***.println("Opening Pdf Document.....");

}

}

**PdfDocumentFactory.java:**

public class PdfDocumentFactory extends DocumentFactory {

*@Override*

public Document createDocument() {

return new PdfDocument();

}

}

**WordDocument.java:**

public class WordDocument implements Document {

*@Override*

public void open() {

System.***out***.println("Opening word document..");

}

}

**WordDocumentFactory.java:**

public class WordDocumentFactory extends DocumentFactory{

*@Override*

public Document createDocument() {

return new WordDocument();

}

}

3. Financial\_Forecasting:

**FinancialForecast.java:**

public class FinancialForecast {

public static double futureValueRecursive(double principal, double rate, int years) {

if (years == 0) {

return principal;

}

return *futureValueRecursive*(principal, rate, years - 1) \* (1 + rate);

}

public static double futureValueMemoized(double principal, double rate, int years, double[] memo) {

if (years == 0) {

return principal;

}

if (memo[years] != 0) {

return memo[years];

}

memo[years] = *futureValueMemoized*(principal, rate, years - 1, memo) \* (1 + rate);

return memo[years];

}

public static void main(String[] args) {

double principal = 10000;

double rate = 0.05;

int years = 5;

double result = *futureValueRecursive*(principal, rate, years);

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

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

double optimizedResult = *futureValueMemoized*(principal, rate, years, memo);

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

}

}

4. E\_commerce\_Platform\_search\_Function

**Product.java:**

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;

}

public void display() {

System.***out***.println("Product ID: " + productId + ", Name: " + productName + ", Category: " + category);

}

}

**ECommerceSearch.java:**

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 low = 0;

int high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

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

return products[mid];

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

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

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

new Product(101, "Shampoo", "Personal Care"),

new Product(104, "Monitor", "Electronics"),

new Product(102, "Notebook", "Stationery")

};

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

int searchId = 104;

System.***out***.println("Using Linear Search:");

Product result1 = *linearSearch*(products, searchId);

if (result1 != null) result1.display();

else System.***out***.println("Product not found!");

System.***out***.println("\nUsing Binary Search:");

Product result2 = *binarySearch*(products, searchId);

if (result2 != null) result2.display();

else System.***out***.println("Product not found!");

}

}