**Exercise 1: Implementing the Singleton Pattern**

package com.example.singleton;

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);

}

}

package com.example.singleton;

public class Main {

public static void main(String[] args) {

Logger logger1 = Logger.*getInstance*();

Logger logger2 = Logger.*getInstance*();

logger1.log("First log message.");

logger2.log("Second log message.");

if (logger1 == logger2) {

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

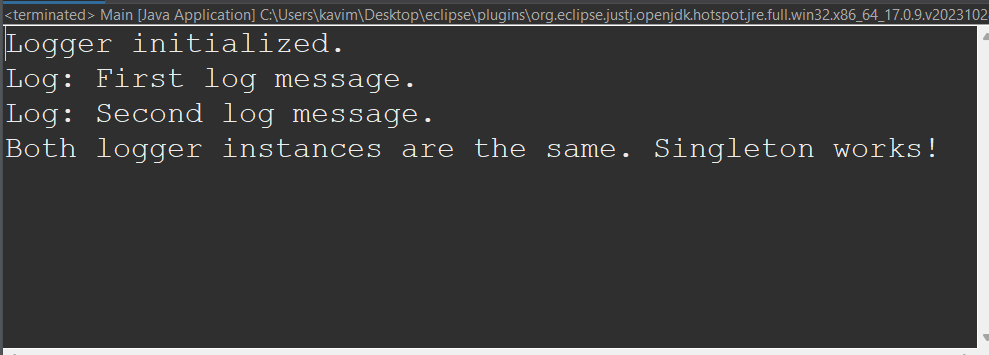
} else {

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

}

}

}



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

import java.util.\*;

interface Document {

void open();

}

class WordDocument implements Document {

public void open() {

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

}

}

class PdfDocument implements Document {

public void open() {

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

}

}

class ExcelDocument implements Document {

public void open() {

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

}

}

abstract class DocumentFactory {

public abstract Document createDocument();

}

class WordFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

class PdfFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

class ExcelFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

public class Main {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordFactory();

Document word = wordFactory.createDocument();

word.open();

DocumentFactory pdfFactory = new PdfFactory();

Document pdf = pdfFactory.createDocument();

pdf.open();

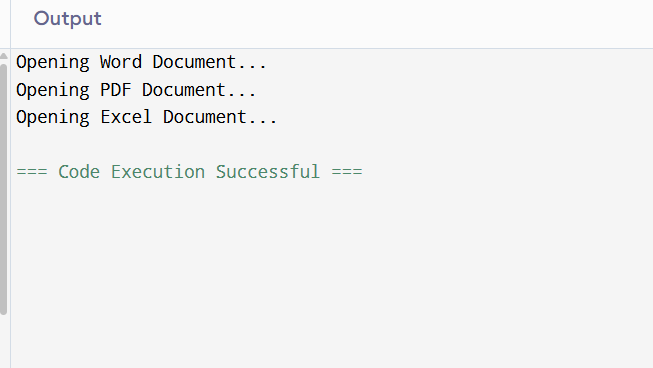
DocumentFactory excelFactory = new ExcelFactory();

Document excel = excelFactory.createDocument();

excel.open();

}

}



**Exercise 3: E-commerce Platform Search Function**

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;

}

public String toString() {

return "Product ID: " + productId + ", Name: " + productName + ", Category: " + category;

}

}

public class Main {

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

for (Product p : products) {

if (p.productId == id) {

return p;

}

}

return null;

}

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

int left = 0, right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

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

return products[mid];

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

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

new Product(103, "Shoes", "Footwear"),

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

new Product(105, "Watch", "Accessories"),

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

new Product(104, "Bag", "Luggage")

};

System.out.println("Linear Search for Product ID 105:");

Product result1 = linearSearch(products, 105);

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

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

System.out.println("\nBinary Search for Product ID 105:");

Product result2 = binarySearch(products, 105);

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

}

}



**Exercise 4: Financial Forecasting**

public class Main {

public static double forecast(double currentValue, double growthRate, int years) {

if (years == 0) {

return currentValue;

}

return forecast(currentValue \* (1 + growthRate), growthRate, years - 1);

}

public static void main(String[] args) {

double initialValue = 10000;

double growthRate = 0.10;

int years = 5;

double futureValue = forecast(initialValue, growthRate, years);

System.out.printf("Predicted value after %d years: Rs %.2f\n", years, futureValue);

}

}

