**Q. Implementing the Singleton Pattern:**

**Code:**

public class Main {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("This is the first message.");

Logger logger2 = Logger.getInstance();

logger2.log("This is the second message.");

if (logger1 == logger2) {

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

} else {

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

}

}

}

class Logger {

private static Logger instance;

private Logger() {

System.out.println("Logger instance created.");

} public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

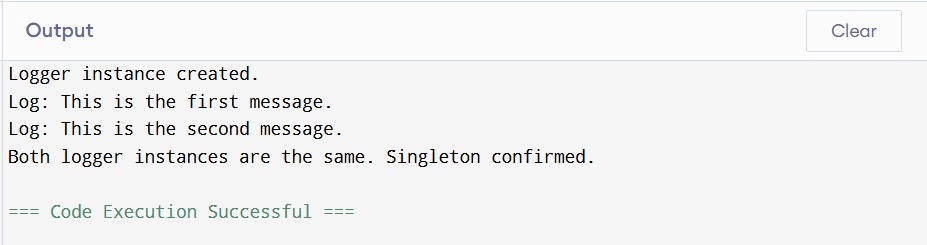
public void log(String message) {

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

}

}

**Output Screenshot:**



**Q. Implementing the Factory Method Pattern:**

**Code:**

public class Main {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

Document wordDoc = wordFactory.createDocument();

wordDoc.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdfDoc = pdfFactory.createDocument();

pdfDoc.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

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 WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

**Output Screenshot:**  


**Q. E-commerce Platform Search Function :**

**Code:**

import java.util.Arrays;

import java.util.Comparator;

public class Main {

public static void main(String[] args) {

Product[] products = {

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

new Product(203, "Shoes", "Footwear"),

new Product(150, "Phone", "Electronics"),

new Product(309, "T-shirt", "Apparel"),

new Product(180, "Watch", "Accessories")

};

int linearResult = linearSearch(products, 150);

System.out.println("Linear Search Result Index: " + linearResult);

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

int binaryResult = binarySearch(products, 150);

System.out.println("Binary Search Result Index: " + binaryResult);

}

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

for (int i = 0; i < products.length; i++) {

if (products[i].productId == productId) {

return i;

}

}

return -1;

}

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

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

while (left <= right) {

int mid = (left + right) / 2;

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

return mid;

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

left = mid + 1;

} else {

right = mid - 1;

}

}

return -1;

}

}

class Product {

int productId;

String productName;

String category;

public Product(int productId, String productName, String category) {

this.productId = productId;

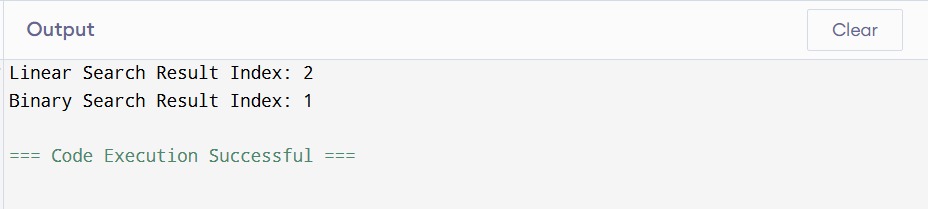
this.productName = productName;

this.category = category;

}

}

**Output Screenshot:**



**Q. Financial Forecasting**

**Code:**

public class Main {

public static void main(String[] args) {

double initialAmount = 10000;

double growthRate = 0.10;

int years = 5;

double futureValue = calculateFutureValue(initialAmount, growthRate, years);

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

}

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

if (years == 0) {

return initial;

}

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

}

}

**Output Screenshot:**

