**EXERCISE 1: IMPLEMENTING THE SINGLETON PATTERN**

**PROGRAM**:

package com.singletonexample;

public class SingletonPatternExample {

public static void main(String[] args) {

System.out.println("--- Testing Singleton Logger ---");

Logger logger1 = Logger.getInstance();

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

Logger logger2 = Logger.getInstance();

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

Logger logger3 = Logger.getInstance();

logger3.log("This is the third log message.");

System.out.println("\n--- Verification ---");

if (logger1 == logger2 && logger2 == logger3) {

System.out.println("All logger instances are the same. Singleton pattern is working correctly!");

} else {

System.out.println("Error: Multiple logger instances were created.");

}

System.out.println("Hashcode of logger1: " + logger1.hashCode());

System.out.println("Hashcode of logger2: " + logger2.hashCode());

System.out.println("Hashcode of logger3: " + logger3.hashCode());

}

}

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;

}

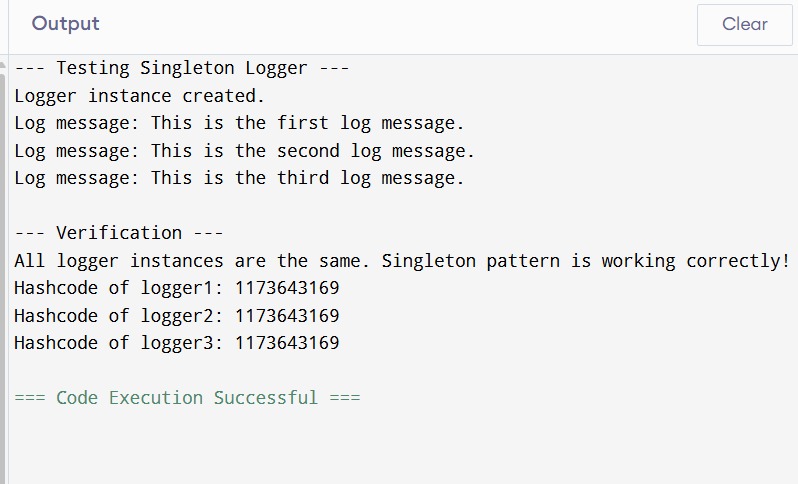
public void log(String message) {

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

}

}

OUTPUT:



**EXERCISE 2: IMPLEMENTING THE FACTORY METHOD PATTERN**

**PROGRAM:**

interface IDocument {

void open();

}

class WordDocument implements IDocument {

public void open() {

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

}

}

class PdfDocument implements IDocument {

public void open() {

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

}

}

class ExcelDocument implements IDocument {

public void open() {

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

}

}

abstract class DocumentFactory {

public abstract IDocument createDocument();

}

class WordDocumentFactory extends DocumentFactory {

public IDocument createDocument() {

return new WordDocument();

}

}

class PdfDocumentFactory extends DocumentFactory {

public IDocument createDocument() {

return new PdfDocument();

}

}

class ExcelDocumentFactory extends DocumentFactory {

public IDocument createDocument() {

return new ExcelDocument();

}

}

public class Main {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

IDocument wordDoc = wordFactory.createDocument();

wordDoc.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

IDocument pdfDoc = pdfFactory.createDocument();

pdfDoc.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

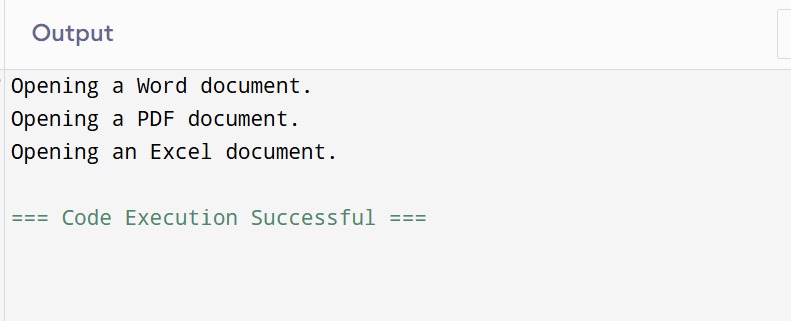
IDocument excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

**OUTPUT:**



**EXERCISE 3: E-COMMERCE PLATFORM SEARCH FUNCTION**

**PROGRAM:**

import java.util.\*;

class Product {

int productId;

String productName;

String category;

public Product(int id, String name, String category) {

this.productId = id;

this.productName = name;

this.category = category;

}}

public class Main {

public static Product linearSearch(Product[] products, String name) {

for (Product p : products) {

if (p.productName.equalsIgnoreCase(name)) {

return p;

}

}

return null;

}

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

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

while (left <= right) {

int mid = (left + right) / 2;

int cmp = products[mid].productName.compareToIgnoreCase(name);

if (cmp == 0) return products[mid];

else if (cmp < 0) left = mid + 1;

else right = mid - 1;

}

return null;

}

public static void printProduct(Product p) {

if (p != null) {

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

} else {

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

}

}

public static void main(String[] args) {

Product[] products = {

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

new Product(102, "Shirt", "Apparel"),

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

new Product(104, "Shoes", "Footwear"),

new Product(105, "Book", "Education")

};

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

printProduct(linearSearch(products, "Mobile"));

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

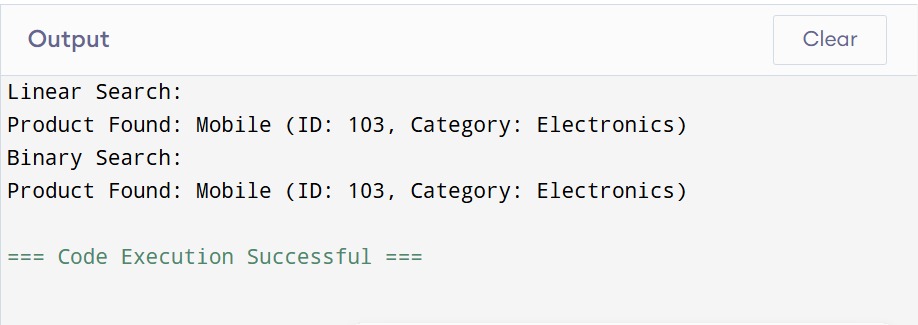
System.out.println("Binary Search:");

printProduct(binarySearch(products, "Mobile"));

    }

}

**Output:**

****

**EXERCISE 4: FINANCIAL FORECASTING**

**PROGRAM:**

public class VeryShortFinancialForecast {

public static double calculateFutureValueRecursive(double presentValue, double annualGrowthRate, int years) {

if (years == 0) {

return presentValue;

}

return calculateFutureValueRecursive(presentValue, annualGrowthRate, years - 1) \* (1 + annualGrowthRate);

}

public static void main(String[] args) {

double initialInvestment = 1000.0;

double growthRate = 0.05;

int forecastYears = 10;

System.out.println("--- Very Short Financial Forecasting ---");

System.out.printf("Initial Investment: %.2f%n", initialInvestment);

System.out.printf("Annual Growth Rate: %.2f%%%n", growthRate \* 100);

System.out.printf("Forecast Years: %d%n", forecastYears);

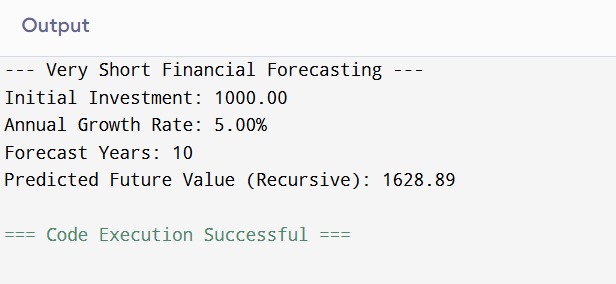
double futureValue = calculateFutureValueRecursive(initialInvestment, growthRate, forecastYears);

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

    }

}

**OUTPUT:**

****