**Exercise 1: Implementing the Singleton Pattern**

**Scenario:**

You need to ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging

Code:

Logger.java

public 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);

}

}

LoggerTest.java

public class LoggerTest {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

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

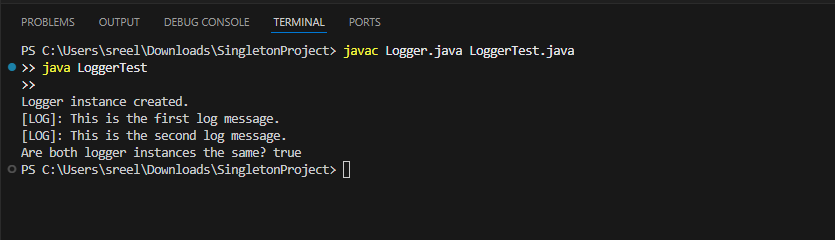
logger2.log("This is the second log message.");

System.out.println("Are both logger instances the same? " + (logger1 == logger2));

}

}

Output:



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

**Scenario:**

You are developing a document management system that needs to create different types of documents (e.g., Word, PDF, Excel). Use the Factory Method Pattern to achieve this.

**Code:**

**Document.java**

public interface Document {

void open();

}

**WordDocument.java**

public class WordDocument implements Document {

public void open() {

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

}

}

**PdfDocument.java**

**public class PdfDocument implements Document {**

**public void open() {**

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

**}**

**}**

**ExcelDocument.java**

public class ExcelDocument implements Document {

public void open() {

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

}

}

### **DocumentFactory.java**

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**WordFactory.java**

**public class WordFactory extends DocumentFactory {**

**public Document createDocument() {**

**return new WordDocument();**

**}**

**}**

**PdfFactory.java**

**public class PdfFactory extends DocumentFactory {**

**public Document createDocument() {**

**return new PdfDocument();**

**}**

**}**

**ExcelFactory.java**

public class ExcelFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

**DocumentFactoryTest.java**

**public class DocumentFactoryTest {**

**public DocumentFactoryTest() {**

**}**

**public static void main(String[] var0) {**

**WordFactory var1 = new WordFactory();**

**Document var2 = var1.createDocument();**

**var2.open();**

**PdfFactory var3 = new PdfFactory();**

**Document var4 = var3.createDocument();**

**var4.open();**

**ExcelFactory var5 = new ExcelFactory();**

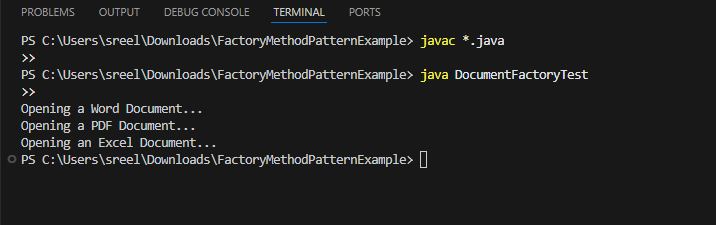
**Document var6 = var5.createDocument();**

**var6.open();**

**}**

**}**

**Output:**

****

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

**Scenario:**

**You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.**

**Code:**

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 String toString() {

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

}

}

SearchEngine.java

import java.util.Arrays;

import java.util.Comparator;

public class SearchEngine {

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, 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 sortProductsById(Product[] products) {

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

}

}

**SearchTest.java**

**public class SearchTest {**

**public static void main(String[] args) {**

**Product[] products = {**

**new Product(104, "Mouse", "Accessories"),**

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

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

**new Product(103, "Keyboard", "Accessories")**

**};**

**System.out.println(" Linear Search for ID 102:");**

**Product result1 = SearchEngine.linearSearch(products, 102);**

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

**SearchEngine.sortProductsById(products); // Binary search requires sorted array**

**System.out.println("\n Binary Search for ID 103:");**

**Product result2 = SearchEngine.binarySearch(products, 103);**

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

**System.out.println("\n Binary Search for ID 999:");**

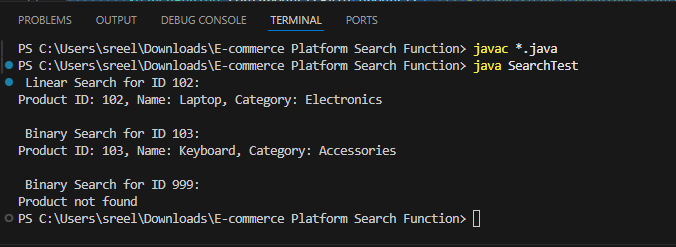
**Product result3 = SearchEngine.binarySearch(products, 999);**

**System.out.println(result3 != null ? result3 : "Product not found");**

**}**

**}**

**Output:**

****

**Exercise 4: Financial Forecasting**

**Scenario:**

**You are developing a financial forecasting tool that predicts future values based on past data.**

**Code:**

**FinancialForecast.java**

**public class FinancialForecast {**

**// Recursive method to calculate future value**

**public static double calculateFutureValue(double principal, double rate, int years) {**

**if (years == 0) {**

**return principal; // base case**

**}**

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

**}**

**public static void main(String[] args) {**

**double principal = 10000; // initial investment**

**double rate = 0.08; // 8% annual growth**

**int years = 5;**

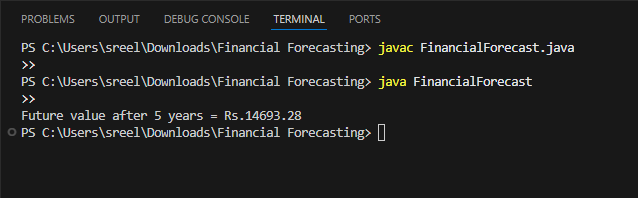
**double futureValue = calculateFutureValue(principal, rate, years);**

**System.out.printf("Future value after %d years = Rs.%.2f%n", years, futureValue);**

**}**

**}**

**Output:**

****

**Exercise 5: Implementing the Decorator Pattern**

**Scenario:**

**You are developing a notification system where notifications can be sent via multiple channels (e.g., Email, SMS). Use the Decorator Pattern to add functionalities dynamically**

**Code:**

**Notifier.java**

**public interface Notifier {**

**void send(String message);**

**}**

**EmailNotifier.java**

**public class EmailNotifier implements Notifier {**

**@Override**

**public void send(String message) {**

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

**}**

**}**

**NotifierDecorator.java**

**public abstract class NotifierDecorator implements Notifier {**

**protected Notifier wrappedNotifier;**

**public NotifierDecorator(Notifier notifier) {**

**this.wrappedNotifier = notifier;**

**}**

**public void send(String message) {**

**wrappedNotifier.send(message); // Delegate to the wrapped notifier**

**}**

**}**

**SMSNotifierDecorator.java**

**public class SMSNotifierDecorator extends NotifierDecorator {**

**public SMSNotifierDecorator(Notifier notifier) {**

**super(notifier);**

**}**

**@Override**

**public void send(String message) {**

**super.send(message); // Send existing notification**

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

**}**

**}**

**SlackNotifierDecorator.java**

**public class SlackNotifierDecorator extends NotifierDecorator {**

**public SlackNotifierDecorator(Notifier notifier) {**

**super(notifier);**

**}**

**@Override**

**public void send(String message) {**

**super.send(message); // Send existing notification**

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

**}**

**}**

**NotificationTest.java**

**public class NotificationTest {**

**public static void main(String[] args) {**

**// Basic notifier**

**Notifier notifier = new EmailNotifier();**

**// Add SMS and Slack notifications using decorators**

**Notifier multiChannelNotifier = new SlackNotifierDecorator(**

**new SMSNotifierDecorator(notifier));**

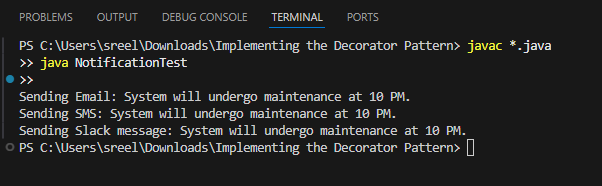
**// Send a message**

**multiChannelNotifier.send("System will undergo maintenance at 10 PM.");**

**}**

**}**

**Output:**

****