**WEEK- 1**

**Design Patterns and Principles:**

**Exercise 1:** Implementation of Singleton Pattern.

**Code:** Singleton Pattern using Logger

**Logger.java**

**package** Pattern;

**public** **class** Logger {

**private** **static** Logger *instance*;

**private** Logger() {

System.***out***.println("Logger instance is being created.");

}

**public** **static** Logger getInstance() {

**if** (*instance* == **null**) {

*instance* = **new** Logger();

}

**return** *instance*;

}

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

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

}

}

**Main.java**

**package** Pattern;

**public** **class** Main {

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

Logger logger1 = Logger.*getInstance*();

logger1.log("The first log message");

Logger logger2 = Logger.*getInstance*();

logger2.log("The second log message");

**if** (logger1 == logger2) {

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

} **else** {

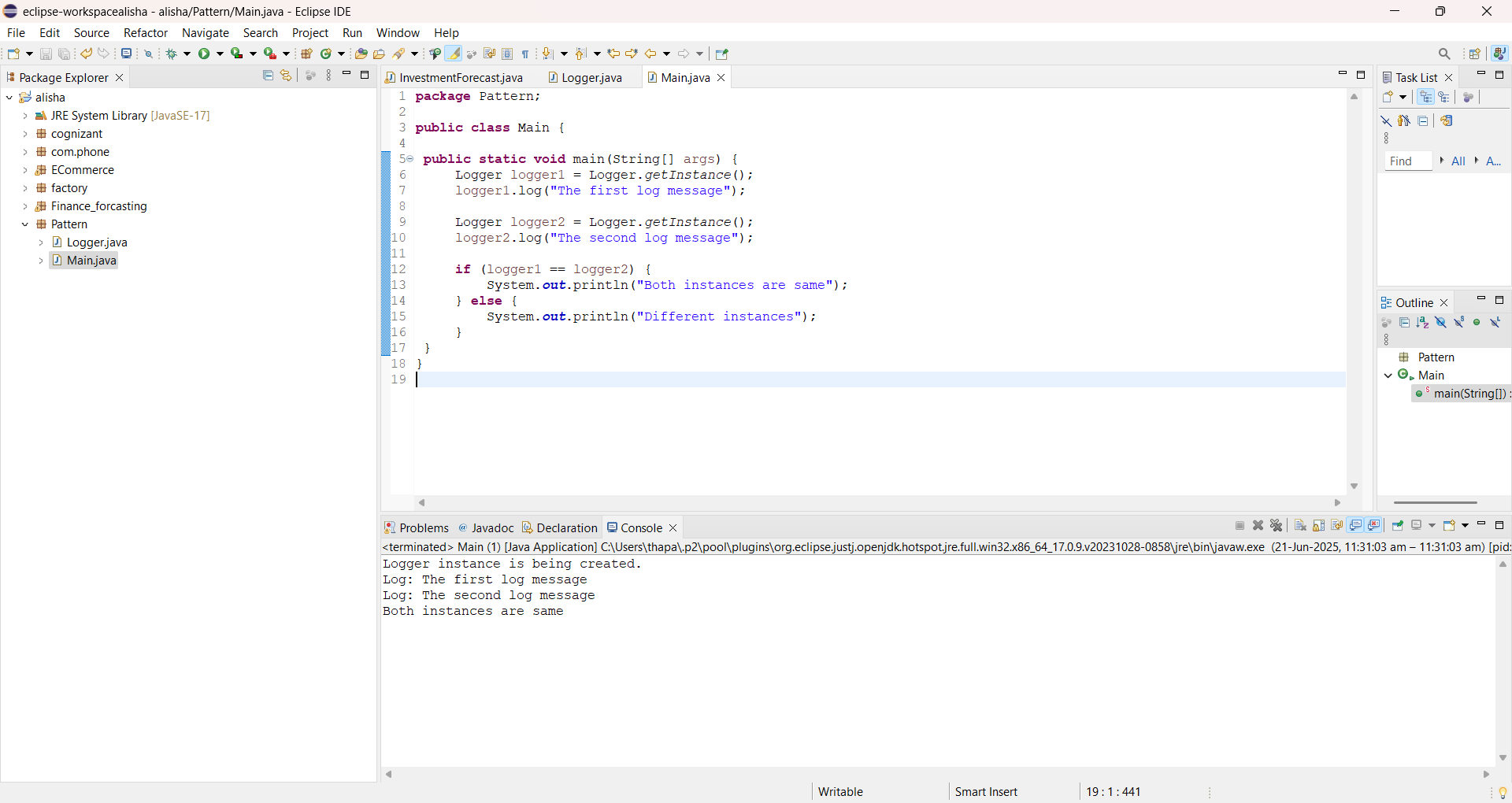
System.***out***.println("Different instances");

}

}

}

**Output:**

****

**Exercise 2:** Implementing Factory Method Pattern

**Code:** Factory Method Pattern

To explain the Factory Method Design Pattern, I chose a real-life relatable example: different mobile operating systems like Android, iOS, and Windows.

**Structure of My Code:**

**1. Interface: OS**

package com.phone;

public interface OS {

void spec();

}

This is the common interface that all operating system classes implement. It contains one method: spec(), which each OS class will define differently based on its features.

**2. Concrete Implementations of OS Interface**

* **Andriod.java**

public class Andriod implements OS {

public void spec() {

System.out.println("Most Powerful OS");

}

}

* **IOS.java**

public class IOS implements OS {

public void spec() {

System.out.println("Most Secure OS");

}

}

* **Windows.java**

public class Windows implements OS {

public void spec() {

System.out.println("I'm about to die...");

}

}

Each of these classes represents a specific OS and provides its own message when the spec() method is called.

**3. Factory Class: OperatingSystemFactory**

public class OperatingSystemFactory {

public OS getInstance(String str) {

if (str.equals("Open"))

return new Andriod();

else if (str.equals("Closed"))

return new IOS();

else

return new Windows();

}

}

This is the Factory class that contains logic to decide which OS object to return based on the input string str.

* If we pass "Open" → it returns an instance of Andriod.
* If we pass "Closed" → it returns an instance of IOS.
* For any other string → it returns an instance of Windows.

This means that the client doesn't need to know or change the creation logic, it only needs to pass the type.

**4. Main Class: Factory**

package factory;

import com.phone.\*;

public class Factory {

public static void main(String args[]) {

OperatingSystemFactory osf = new OperatingSystemFactory();

OS obj = osf.getInstance("Open");

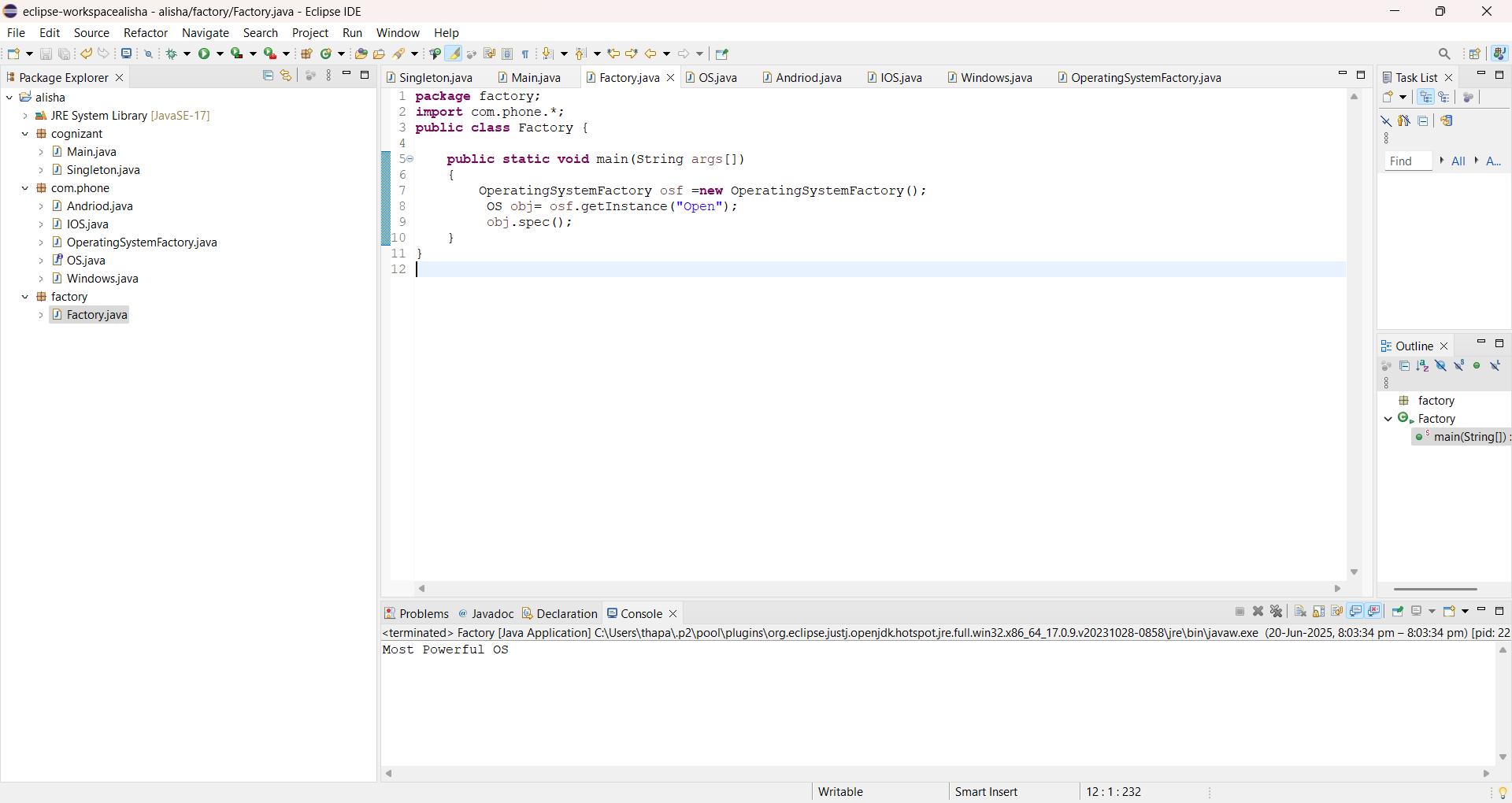
obj.spec();

}

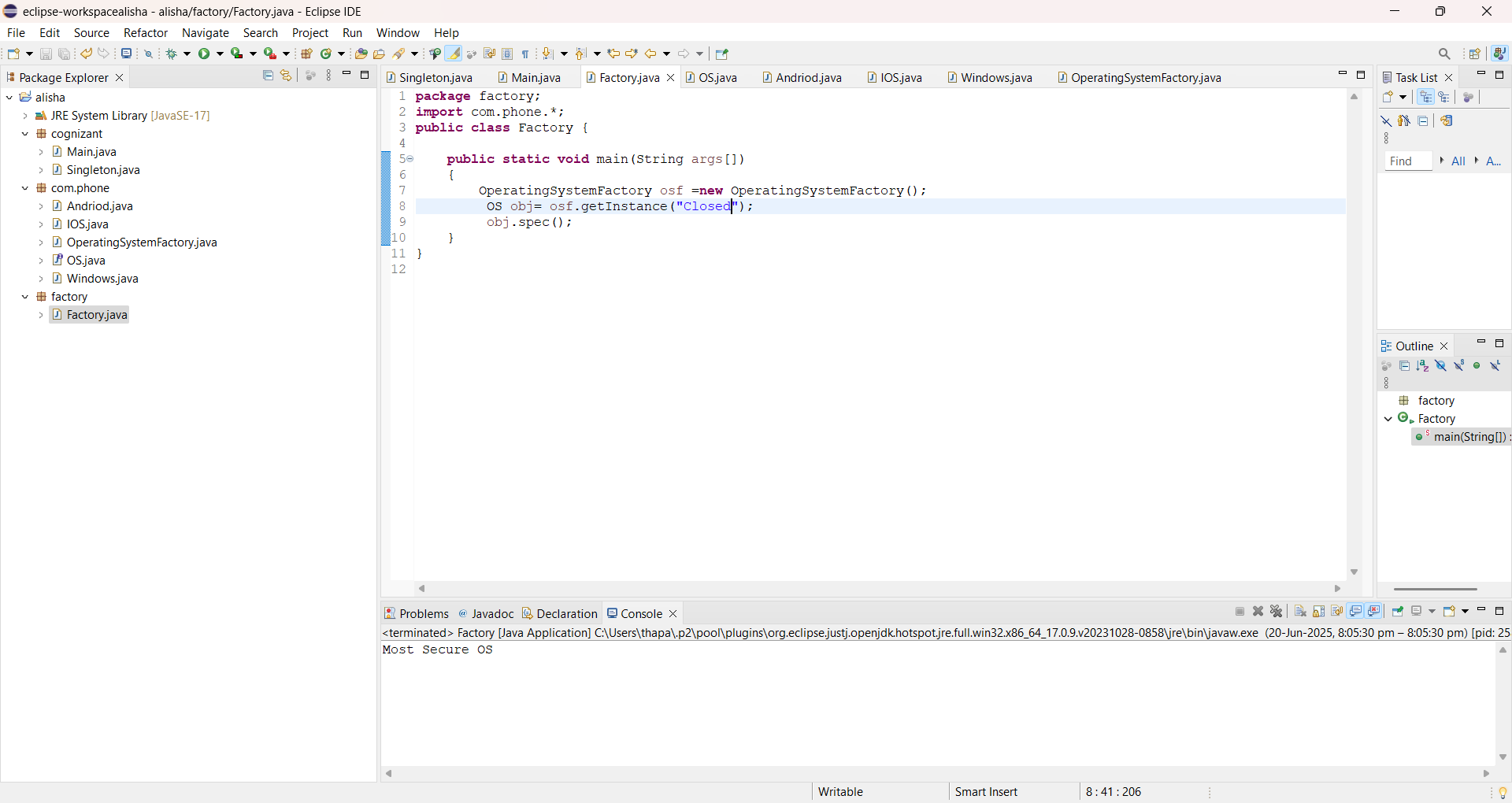
}

* Here, the main method does not directly create Android/iOS/Windows objects.
* Instead, it uses the OperatingSystemFactory class to get an instance based on the type.

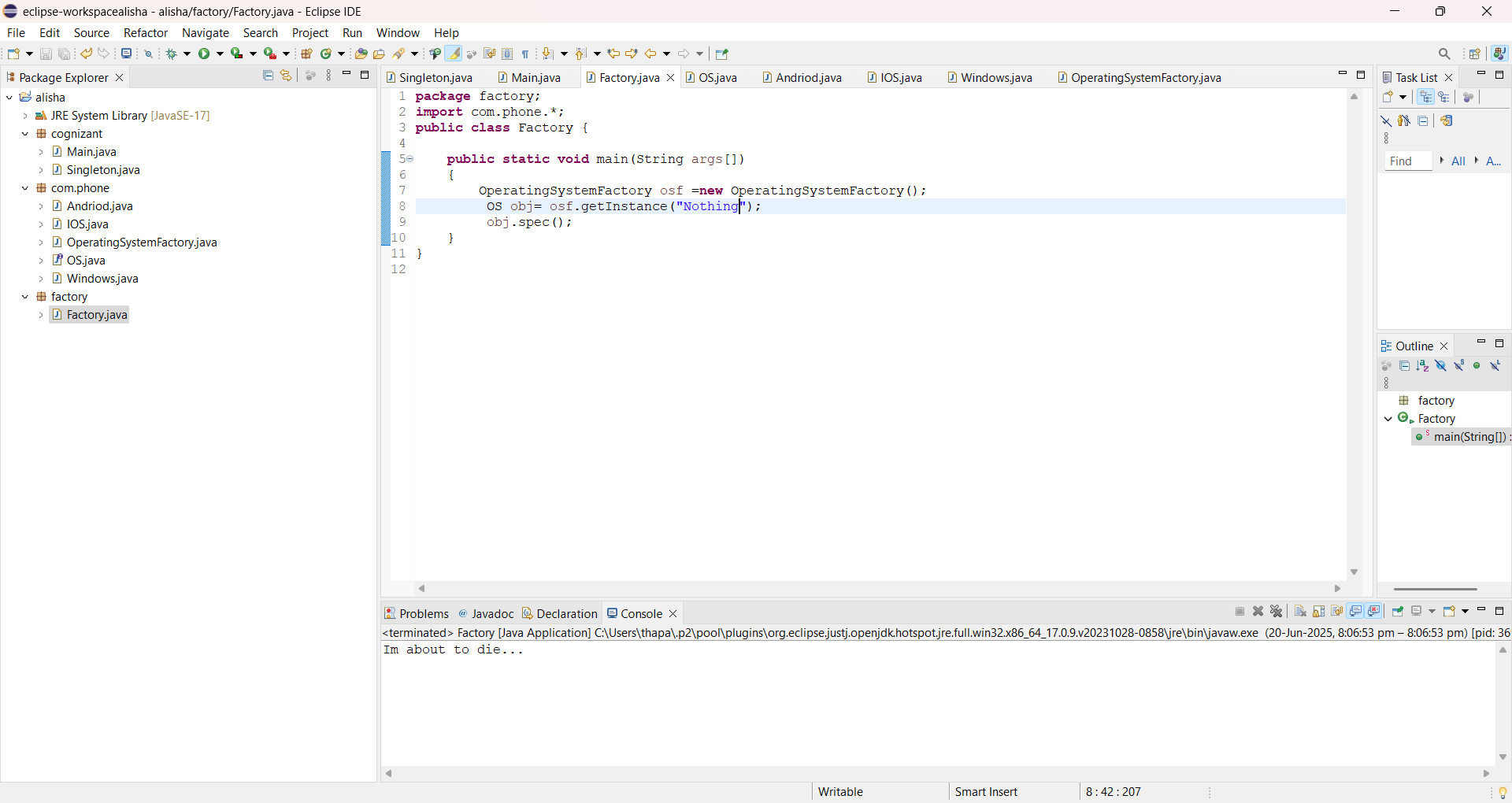
**Output:** We passed "Open", so the output will be



**Output:** we passed "Closed", so the output will be



**Output:** If we passed "Unknown" or any other string, it would print

****

**Data Structures and Algorithms:**

**Exercise 2:** E-commerce Platform Searching Function

**Code:**

**Main.java:**

package ECommerce;

import java.util.\*;

public class EcommerceSearch {

public static Product linearSearch(Product[] pd, String key) {

for (Product p : pd) {

if (p.proname.contains(key)) {

return p;

}

}

return null;

}

public static Product binarySearch(Product[] pd, String key) {

int l = 0, h= pd.length - 1;

while (l <= h) {

int mid = (l + h) / 2;

int cmp = pd[mid].proname.compareToIgnoreCase(key);

if (cmp == 0) {

return pd[mid];

} else if (cmp < 0) {

l= mid + 1;

} else {

h = mid - 1;

}

}

return null; }

public static void main(String[] args) {

Product[] p = {

new Product(101, "Shoes", "Fashion", 2000),

new Product(102, "Shirt", "Fashion", 1299),

new Product(103, "Kurtas", "Traditional", 999),

new Product(104, "Shoe Rack", "Furniture", 1799),

new Product(105, "Smartphone", "Electronics", 14999)

};

Scanner sc = new Scanner(System.*in*);

System.*out*.print("Enter product name to search: ");

String key = sc.nextLine().toLowerCase();

Product lsearch = *linearSearch*(p, key);

if (lsearch != null) {

System.*out*.println("Linear Search Result: " + lsearch);

} else {

System.*out*.println("Product not found in Linear Search");

}

Arrays.*sort*(p, Comparator.*comparing*(prod -> prod.proname));

Product Bsearch = *binarySearch*(p, key);

if (Bsearch != null) {

System.*out*.println("Binary Search Result: " + Bsearch);

} else {

System.*out*.println("Product not found in Binary Search");

}

}

}

**Product.java:**

**package** ECommerce;

**public** **class** Product {

**int** proid;

String proname;

String category;

**int** price;

**public** Product(**int** proid, String proname, String category, **int** price) {

**this**.proid = proid;

**this**.proname = proname.toLowerCase();

**this**.category = category;

**this**.price = price;

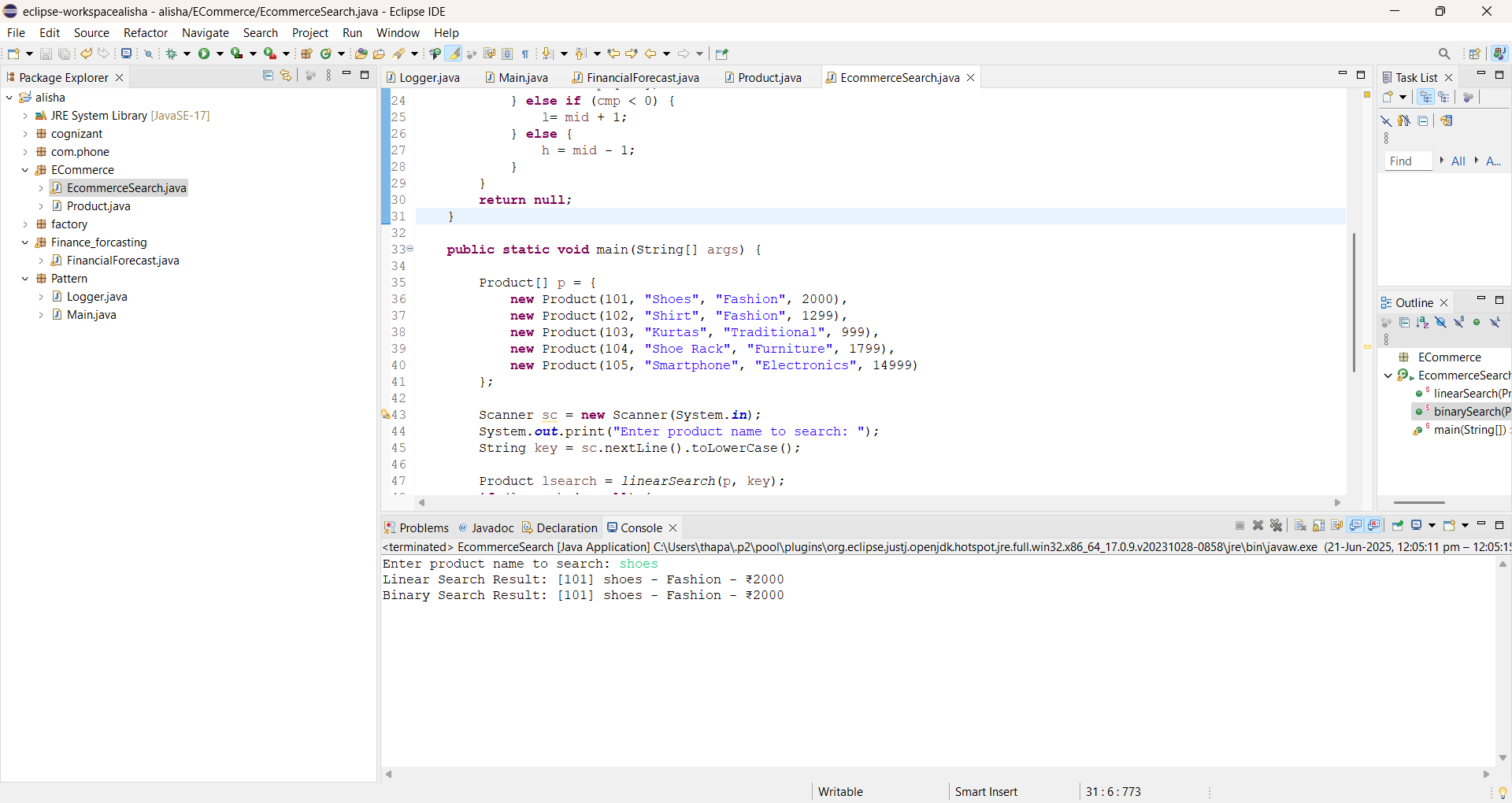
}

**public** String toString() {

**return** "[" + proid + "] " + proname + " - " + category + " - ₹" + price;

}

}

**Output:**

**Exercise 7:** Financial Forecasting

**Code:**

**package** Finance\_forcasting;

**import** java.util.Scanner;

**public** **class** FinancialForecast {

**public** **static** **int** predict(**int** amt, **int** rate, **int** yrs) {

**if** (yrs == 0) {

**return** amt;

}

**int** increase = amt + (amt \* rate / 100);

**return** *predict*(increase, rate, yrs - 1);

}

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

Scanner sc = **new** Scanner(System.***in***);

System.***out***.print("Enter initial investment amount: ");

**int** start = sc.nextInt();

System.***out***.print("Enter annual growth rate (in %): ");

**int** rate = sc.nextInt();

System.***out***.print("Enter number of years: ");

**int** yrs = sc.nextInt();

**int** result = *predict*(start, rate, yrs);

System.***out***.println("Predicted value after " + yrs + " years: ₹" + result);

}

}

**Output:**

