# Week 1 -Desing Patterns

**Exercise 1: Implementing the Singleton Pattern**

**Logger.java**

**package** singleton;

**public** **class** Logger

{

**private** **static** Logger *loggerInstance*;

**private** Logger()

{

System.***out***.println("I am from private Logger Constructor");

}

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

{

**if**(*loggerInstance*==**null**)

{

*loggerInstance*=**new** Logger();

}

**return** *loggerInstance*;

}

**public** **void** log(String text)

{

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

}

}

**Main.Java**

package singleton;

public class Main {

public static void main(String[] args) {

Logger log1=Logger.*getInstance*();

Logger log2=Logger.*getInstance*();

log1.log("First Log Message");

log2.log("Second Log Message");

//Testing of singleton pattern implementation

// Check if both references point to the same object

if (log1 == log2) {

System.*out*.println("Both logger1 and logger2 are the same instance.");

} else {

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

}

}

}

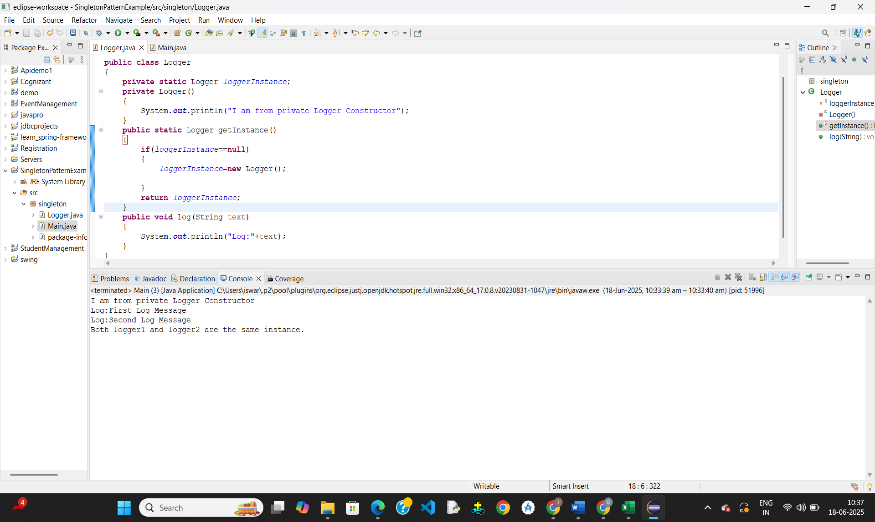
**Output:**

I am from private Logger Constructor

Log:First Log Message

Log:Second Log Message

Both logger1 and logger2 are the same instance.



**Exercise 3: Implementing the Builder Pattern**

**Computer.java**

**package** builderPattern;

**public** **class** Computer {

**private** String CPU;

**private** String RAM;

**private** String storage;

**private** String graphicsCard;

// Private constructor accepts Builder object

**private** Computer(Builder builder) {

**this**.CPU = builder.CPU;

**this**.RAM = builder.RAM;

**this**.storage = builder.storage;

**this**.graphicsCard = builder.graphicsCard;

}

// Print method to show configuration

**public** **void** showSpecs() {

System.***out***.println("CPU: " + CPU);

System.***out***.println("RAM: " + RAM);

System.***out***.println("Storage: " + storage);

System.***out***.println("Graphics Card: " + graphicsCard);

}

// Static Nested Builder Class

**public** **static** **class** Builder {

**private** String CPU;

**private** String RAM;

**private** String storage;

**private** String graphicsCard;

**public** Builder setCPU(String CPU) {

**this**.CPU = CPU;

**return** **this**;

}

**public** Builder setRAM(String RAM) {

**this**.RAM = RAM;

**return** **this**;

}

**public** Builder setStorage(String storage) {

**this**.storage = storage;

**return** **this**;

}

**public** Builder setGraphicsCard(String graphicsCard) {

**this**.graphicsCard = graphicsCard;

**return** **this**;

}

**public** Computer build() {

**return** **new** Computer(**this**);

}

}

}

**Main.java**

**package** builderPattern;

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

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

// Basic computer

Computer basicComputer = **new** Computer.Builder()

.setCPU("Intel i3")

.setRAM("8GB")

.build();

// Gaming computer

Computer gamingComputer = **new** Computer.Builder()

.setCPU("Intel i9")

.setRAM("32GB")

.setStorage("1TB SSD")

.setGraphicsCard("NVIDIA RTX 4080")

.build();

// Print specs

System.***out***.println("Basic Computer:");

basicComputer.showSpecs();

System.***out***.println("\nGaming Computer:");

gamingComputer.showSpecs();

}

}

**Output:**

Basic Computer:

CPU: Intel i3

RAM: 8GB

Storage: null

Graphics Card: null

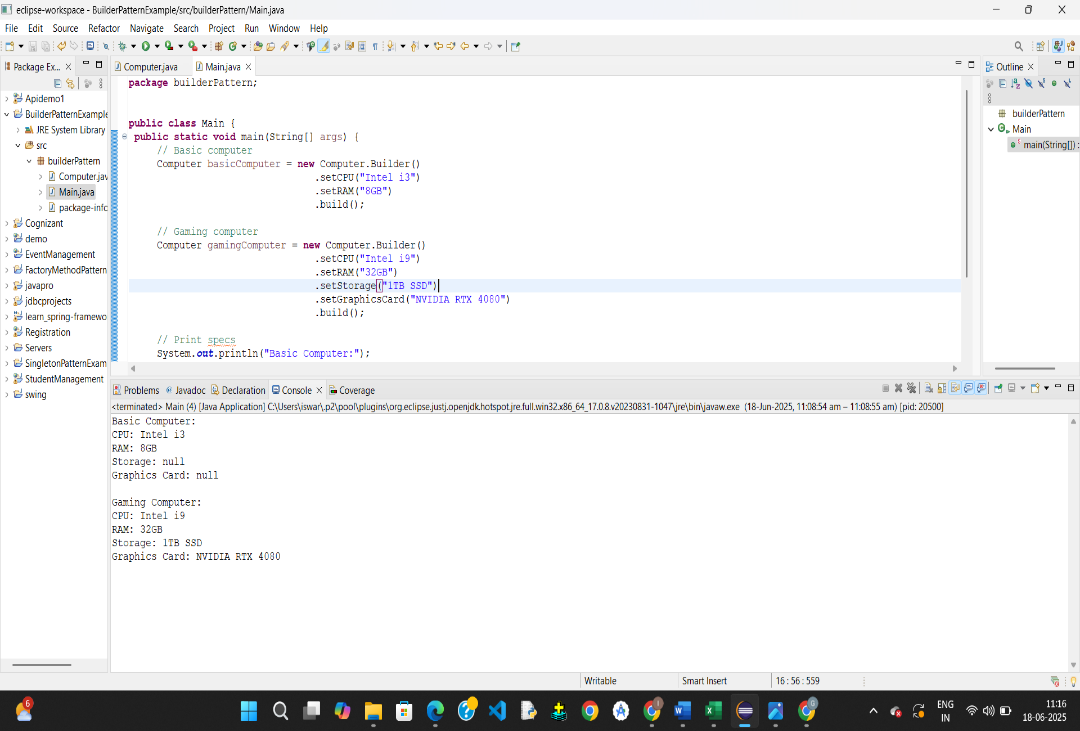
Gaming Computer:

CPU: Intel i9

RAM: 32GB

Storage: 1TB SSD

Graphics Card: NVIDIA RTX 4080



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

**package** factoryMethodPattern;

**public** **interface** Documnet

{

**public** **void** open();

}

**WordDocument.java**

**package** factoryMethodPattern;

**public** **class** WordDocument **implements** Documnet

{

**public** **void** open()

{

System.***out***.println("opening word document");

}

}

**PdfDocument.java**

**package** factoryMethodPattern;

**public** **class** PdfDocument **implements** Documnet

{

**public** **void** open()

{

System.***out***.println("opening pdf document");

}

}

**ExcelDocument.java**

**package** factoryMethodPattern;

**public** **class** ExcelDocument **implements** Documnet

{

**public** **void** open()

{

System.***out***.println("opening excel document");

}

}

**Factory class ->Factory.java**

**package** factoryMethodPattern;

**public** **class** Factory {

**public** **static** Documnet createDocument(String type) {

**switch** (type.toLowerCase()) {

**case** "word":

**return** **new** WordDocument();

**case** "pdf":

**return** **new** PdfDocument();

**case** "excel":

**return** **new** ExcelDocument();

**default**:

**throw** **new** IllegalArgumentException("Unknown document type: " + type);

}

}

}

**package** factoryMethodPattern;

**Main.java**

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

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

Documnet doc1 = Factory.*createDocument*("word");

doc1.open();

Documnet doc2 = Factory.*createDocument*("pdf");

doc2.open();

Documnet doc3 = Factory.*createDocument*("excel");

doc3.open();

}

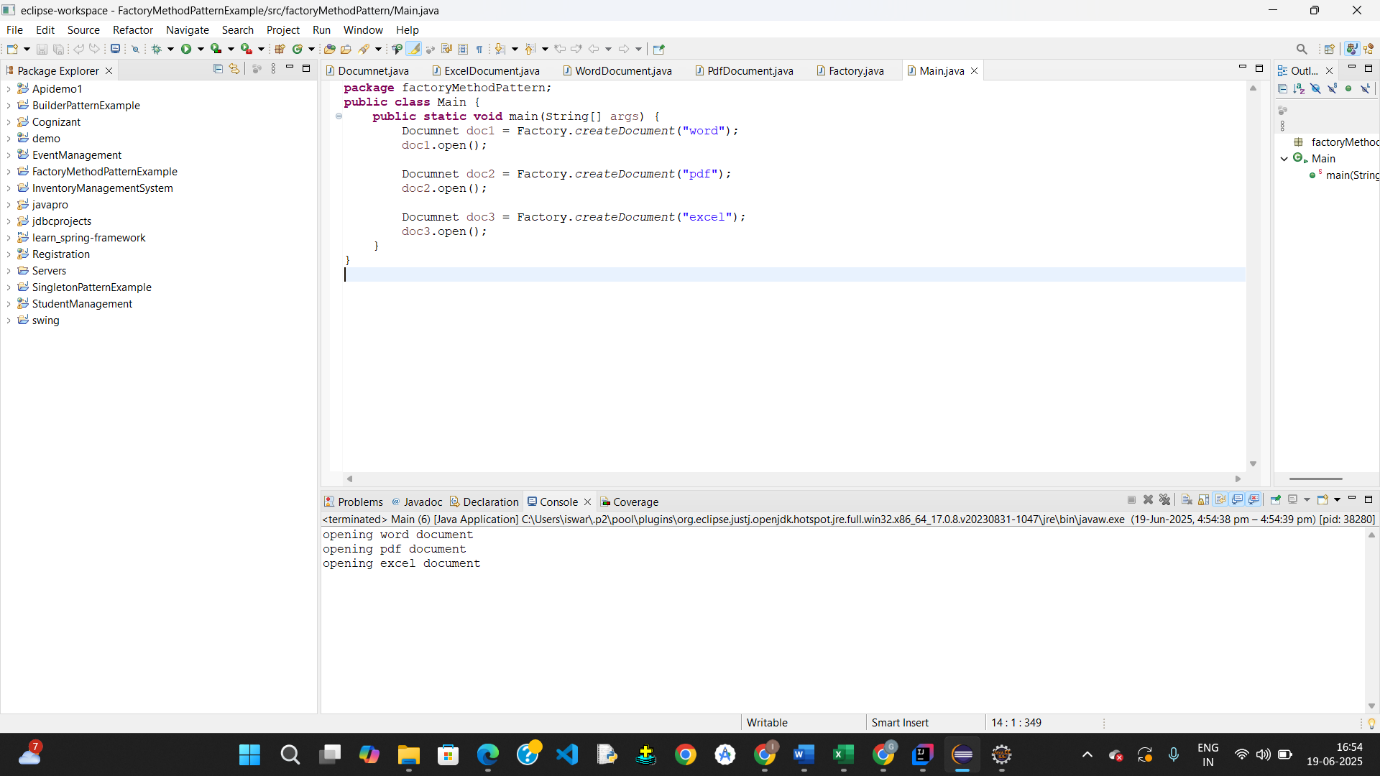
}

**Output:**

opening word document

opening pdf document

opening excel document



# Week-1

# Data Structures and Algorithms

**Exercise 1: Inventory Management System**

**Product.java**

**package** inventorySystem;

**public** **class** Product

{

**int** productId;

String productName;

**int** productQuantity;

**int** price;

**public** Product(**int** productId,String productName,**int** productQuantity,

**int** price)

{

**this**.productId=productId;

**this**.productName=productName;

**this**.productQuantity=productQuantity;

**this**.price=price;

}

**public** String toString()

{

**return** "Product Id: "+productId+" Product Name: "+productName+

" Product Quantity: "+productQuantity+" Price: "+price;

}

}

**ProductServices.java**

package inventorySystem;

import java.util.\*;

public class ProductService

{

HashMap<Integer,Product> products =new HashMap<>();

public void addproduct(Product prod)

{

products.put(prod.productId,prod);

}

public void updateproduct(Product prod)

{

products.put(prod.productId,prod);

}

public void deleteproduct(int prodid)

{

products.remove(prodid);

}

public void displayproducts()

{

for(Product prod:products.values())

{

System.*out*.println(prod);

}

}

}

**Main.java**

package inventorySystem;

public class Main {

public static void main(String[] args)

{

ProductService ps=new ProductService();

ps.addproduct(new Product(1,"phone",23,40000));

ps.addproduct(new Product(2,"laptop",44,70000));

//ps.deleteproduct(1);

//ps.updateproduct(new Product(2,"lapo",46,987987));

ps.displayproducts();

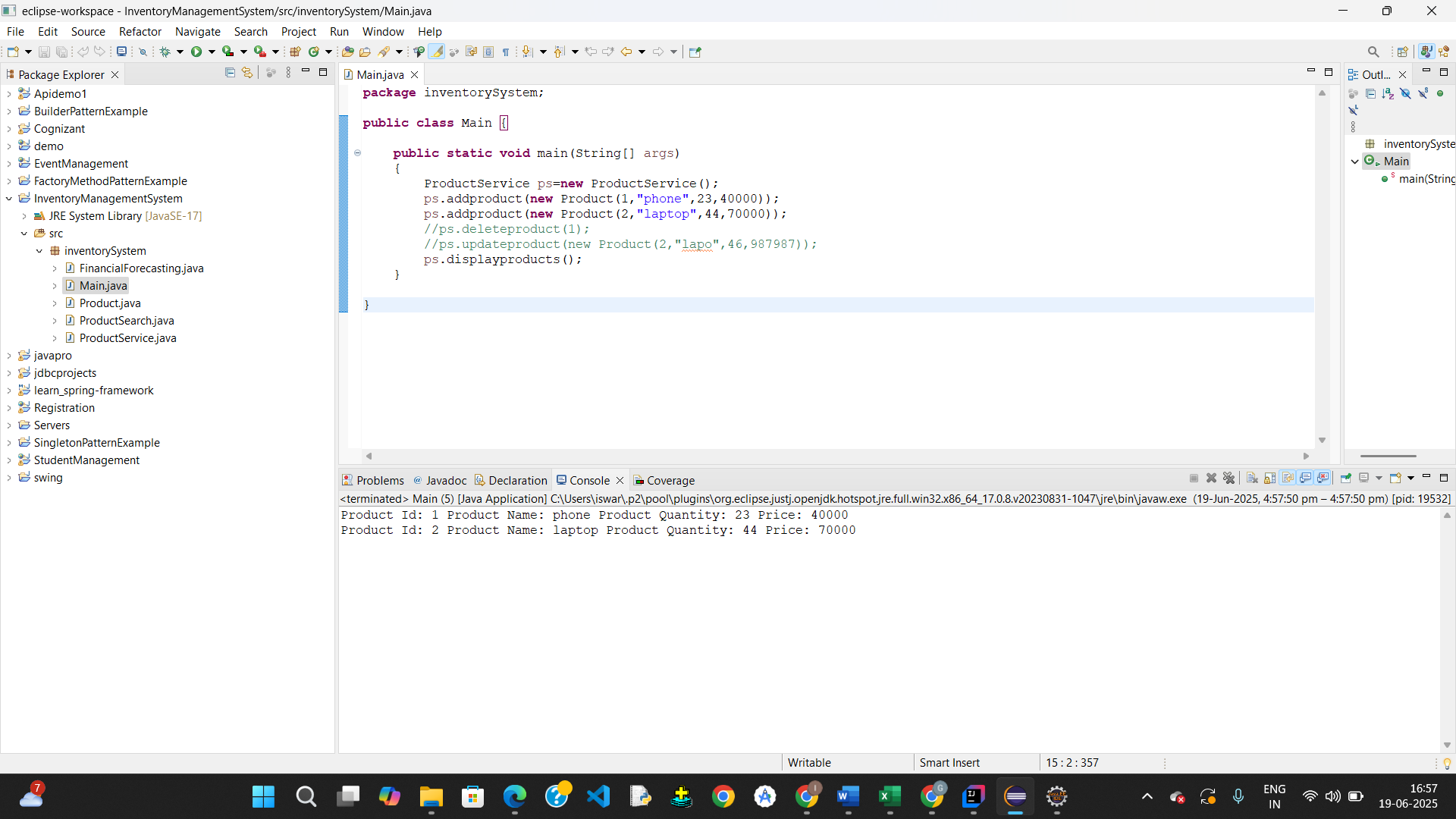
}

}

Output:

Product Id: 1 Product Name: phone Product Quantity: 23 Price: 40000

Product Id: 2 Product Name: lapo Product Quantity: 46 Price: 987987



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

**Product.java**

**package** inventorySystem;

**public** **class** Product

{

**int** productId;

String productName;

**int** productQuantity;

**int** price;

**public** Product(**int** productId,String productName,**int** productQuantity,

**int** price)

{

**this**.productId=productId;

**this**.productName=productName;

**this**.productQuantity=productQuantity;

**this**.price=price;

}

**public** String toString()

{

**return** "Product Id: "+productId+" Product Name: "+productName+

" Product Quantity: "+productQuantity+" Price: "+price;

}

}

**ProductSearch.java**

package inventorySystem;

import java.util.Arrays;

import java.util.Comparator;

public class ProductSearch {

// Linear Search by product name

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

for (Product p : products) {

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

return p;

}

}

return null;

}

// Binary Search by product name (requires sorted array)

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 main(String[] args) {

Product[] products = {

new Product(1, "Laptop",23,30000),

new Product(2, "Shoes",40,70000),

new Product(3, "Camera", 2,10000),

new Product(4, "Book",26,1000),

new Product(5, "Watch", 5,50000)

};

// Linear Search

Product foundLinear = *linearSearch*(products, "Camera");

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

// Binary Search (must sort first)

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

Product foundBinary = *binarySearch*(products, "Camera");

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

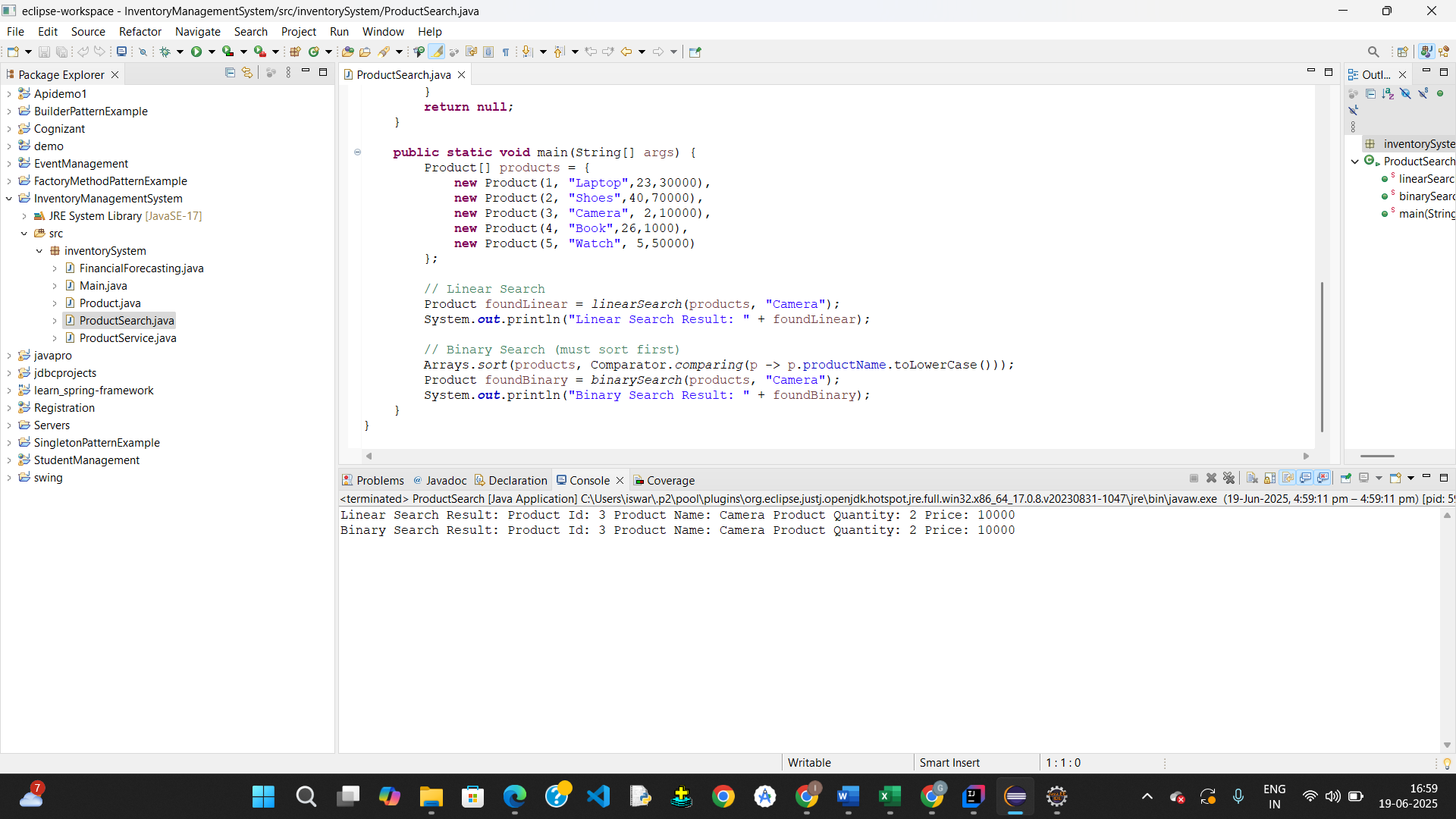
}

}

Output:

Linear Search Result: Product Id: 3 Product Name: Camera Product Quantity: 2 Price: 10000

Binary Search Result: Product Id: 3 Product Name: Camera Product Quantity: 2 Price: 10000



**Exercise 7: Financial Forecasting**

package inventorySystem;

public class FinancialForecasting {

public static double futureValue(int n, double initialValue, double rate) {

if (n == 0) {

return initialValue;

}

return futureValue(n - 1, initialValue, rate) \* (1 + rate);

}

public static void main(String[] args) {

double initial = 10000;

double rate = 0.05;

int years = 5;

double result = futureValue(years, initial, rate);

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

}

}

**Output:**

Future Value after 5 years: 12762.82 Rupees

