# **WEEK 1**

### **DESIGN PATTERNS AND PRINCIPLES:**

### SINGLETON PATTERN:

### SOURCE CODE:

**Logger.java**

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

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

**private** Logger() {}

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

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

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

}

**return** *instance*;

}

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

System.***out***.println(message);

}

}

**TestLogger.java**

public class TestLogger {

public static void main(String[] args) {

Logger logger1 = Logger.*getInstance*();

logger1.log("Logging from logger1.");

Logger logger2 = Logger.*getInstance*();

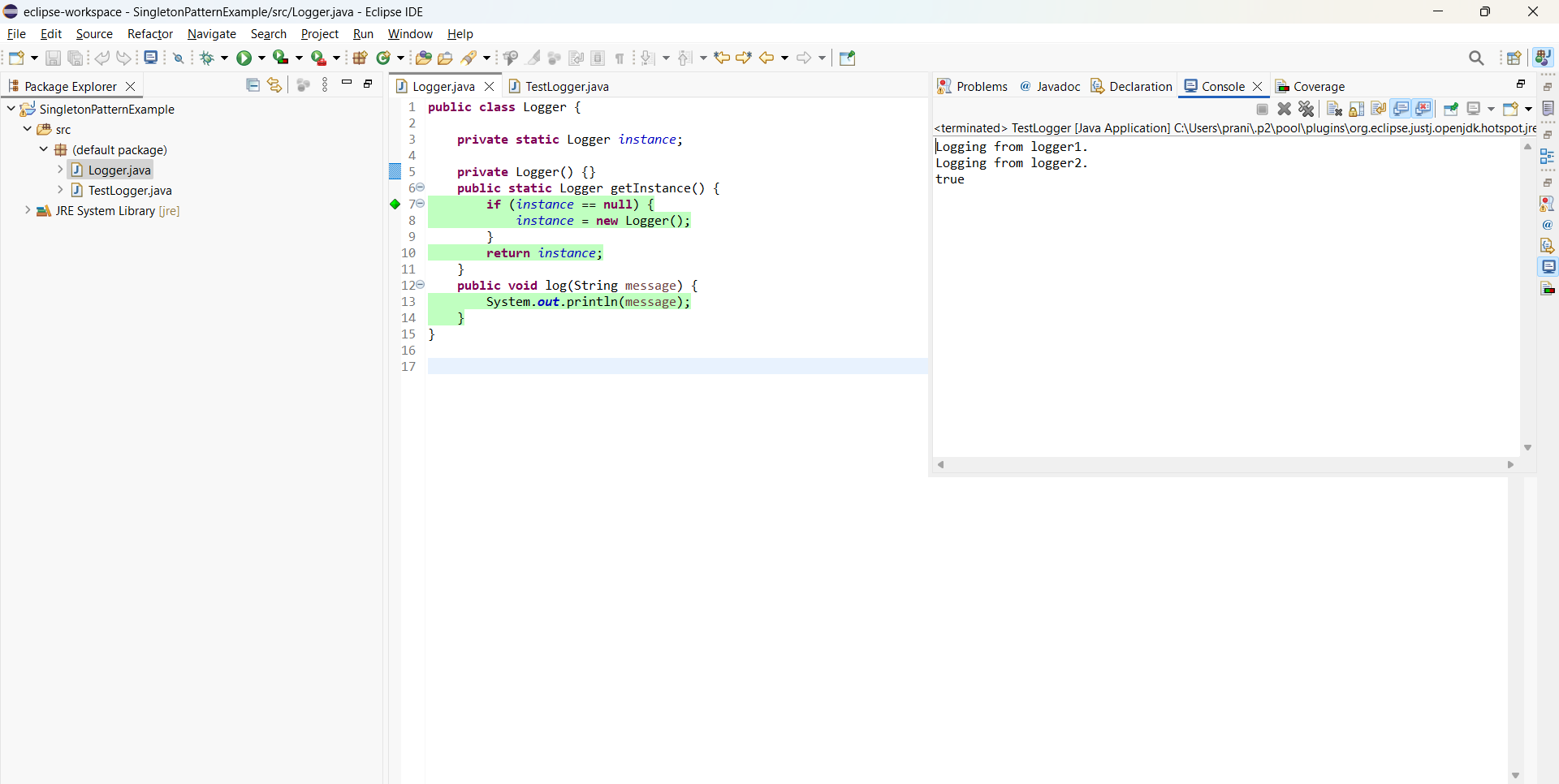
logger2.log("Logging from logger2.");

System.*out*.println(logger1 == logger2);

}

}

**OUTPUT:**

****

### FACTORY METHOD PATTERN

### SOURCE CODE

**FactoryMethodPattern.java:**

**public** **class** FactoryMethodPattern{

**interface** Document {

**void** open();

}

**static** **class** WordDocument **implements** Document {

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

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

}

}

**static** **class** PdfDocument **implements** Document {

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

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

}

}

**static** **class** ExcelDocument **implements** Document{

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

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

}

}

**abstract** **static** **class** DocumentFactory {

**public** **abstract** Document createDocument();

}

**static** **class** WordDocumentFactory **extends** DocumentFactory {

**public** Document createDocument() {

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

}

}

**static** **class** PdfDocumentFactory **extends** DocumentFactory {

**public** Document createDocument() {

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

}

}

**static** **class** ExcelDocumentFactory **extends** DocumentFactory {

**public** Document createDocument() {

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

}

}

**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();

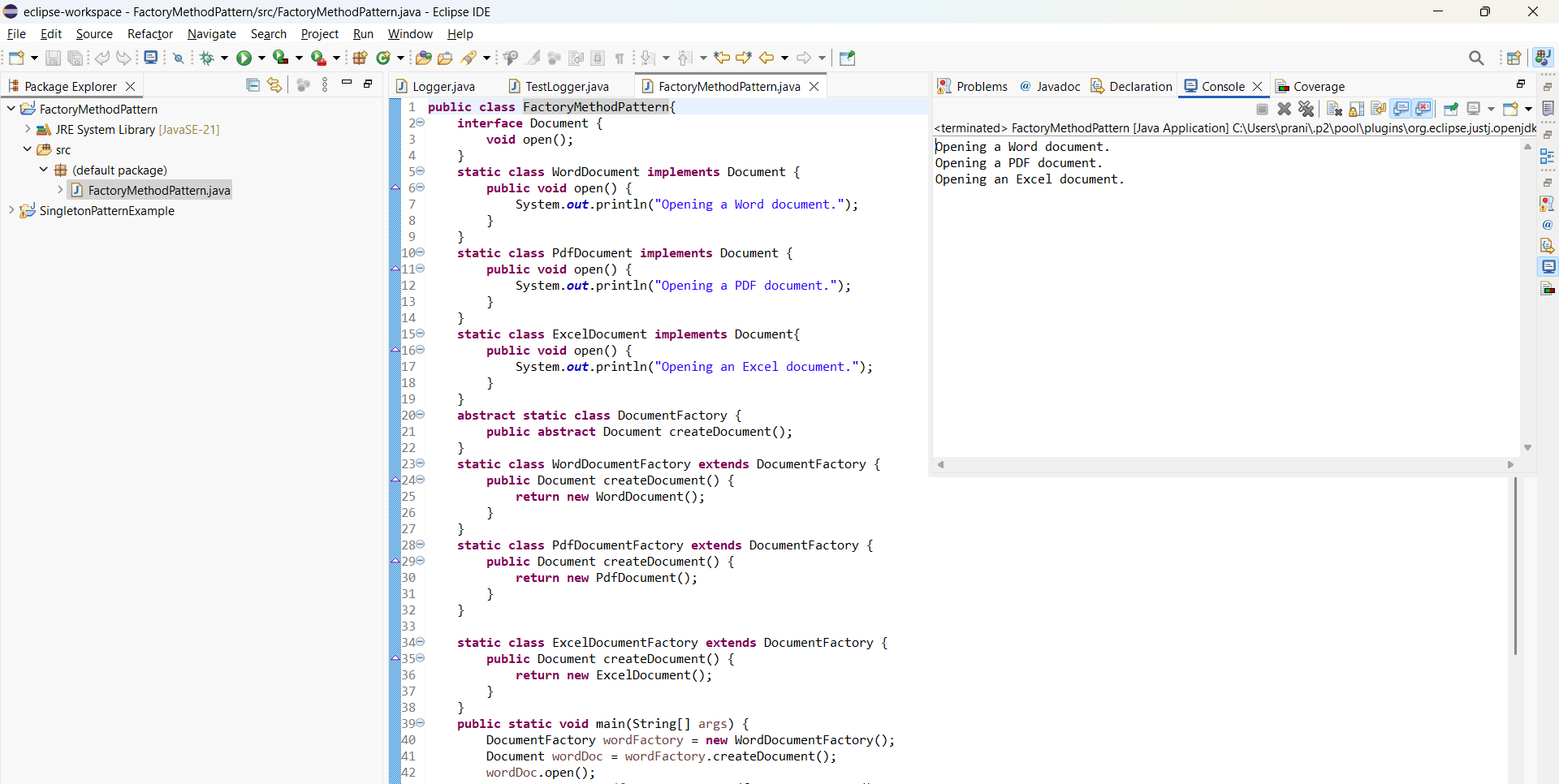
Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

**OUTPUT:**

****

## **DATA STRUCTURES AND ALGORITHMS**

### SOURCE CODE

**EcommerceSearch.java:**

**import** java.util.Arrays;

**public** **class** EcommerceSearch{

**static** **class** Product {

**int** productId;

String productName;

String category;

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

**this**.productId = productId;

**this**.productName = productName;

**this**.category = category;

}

**public** String toString() {

**return** "ProductID: " + productId + ", Name: " + productName + ", Category: " + category;

}

}

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

**for** (Product product : products) {

**if** (product.productName.equalsIgnoreCase(productName)) {

**return** product;

}

}

**return** **null**;

}

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

**int** low = 0, high = products.length - 1;

**while** (low <= high) {

**int** mid = (low + high) / 2;

**int** comparison = products[mid].productName.compareToIgnoreCase(productName);

**if** (comparison == 0) {

**return** products[mid];

} **else** **if** (comparison < 0) {

low = mid + 1;

} **else** {

high = mid - 1;

}

}

**return** **null**;

}

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

Product[] products = {

**new** Product(1, "Laptop", "Electronics"),

**new** Product(2, "Phone", "Electronics"),

**new** Product(3, "Shoes", "Fashion"),

**new** Product(4, "Watch", "Accessories"),

**new** Product(5, "Table", "Furniture")

};

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

Product foundProduct = *linearSearch*(products, "Phone");

**if** (foundProduct != **null**) {

System.***out***.println("Product found: " + foundProduct);

} **else** {

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

}

Arrays.*sort*(products, (p1, p2) -> p1.productName.compareToIgnoreCase(p2.productName));

System.***out***.println("\nUsing Binary Search:");

foundProduct = *binarySearch*(products, "Watch");

**if** (foundProduct != **null**) {

System.***out***.println("Product found: " + foundProduct);

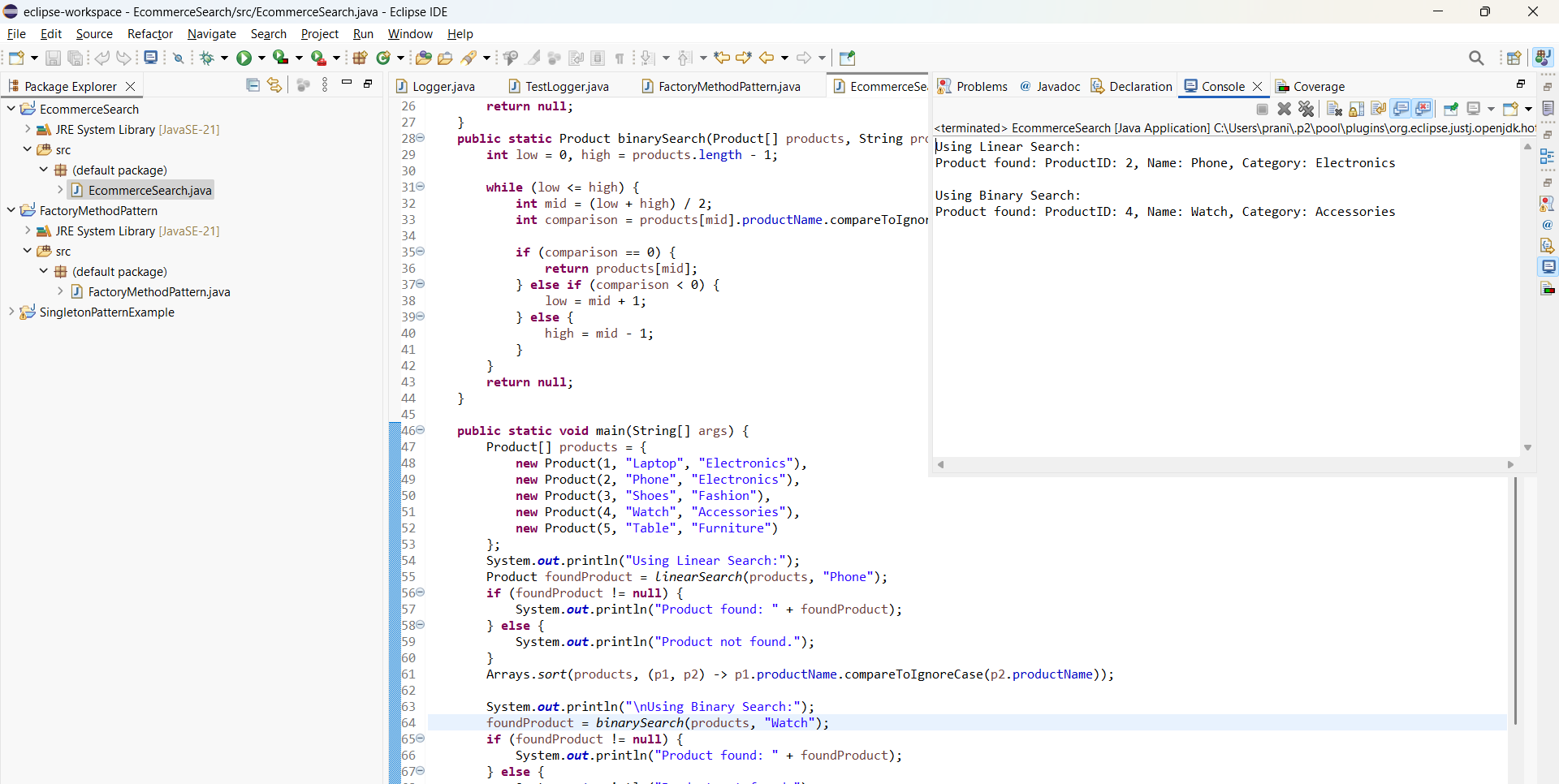
} **else** {

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

}

}

}

**OUTPUT:**

### FINANCIAL FORECASTING

#### SOURCE CODE:

**public** **class** FinancialForecasting {

**public** **static** **double** predictFutureValue(**double** currentValue, **double** growthRate, **int** years) {

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

**return** currentValue;

}

**return** *predictFutureValue*(currentValue \* (1 + growthRate / 100), growthRate, years - 1);

}

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

**double** currentValue = 800;

**double** growthRate = 10;

**int** years = 2;

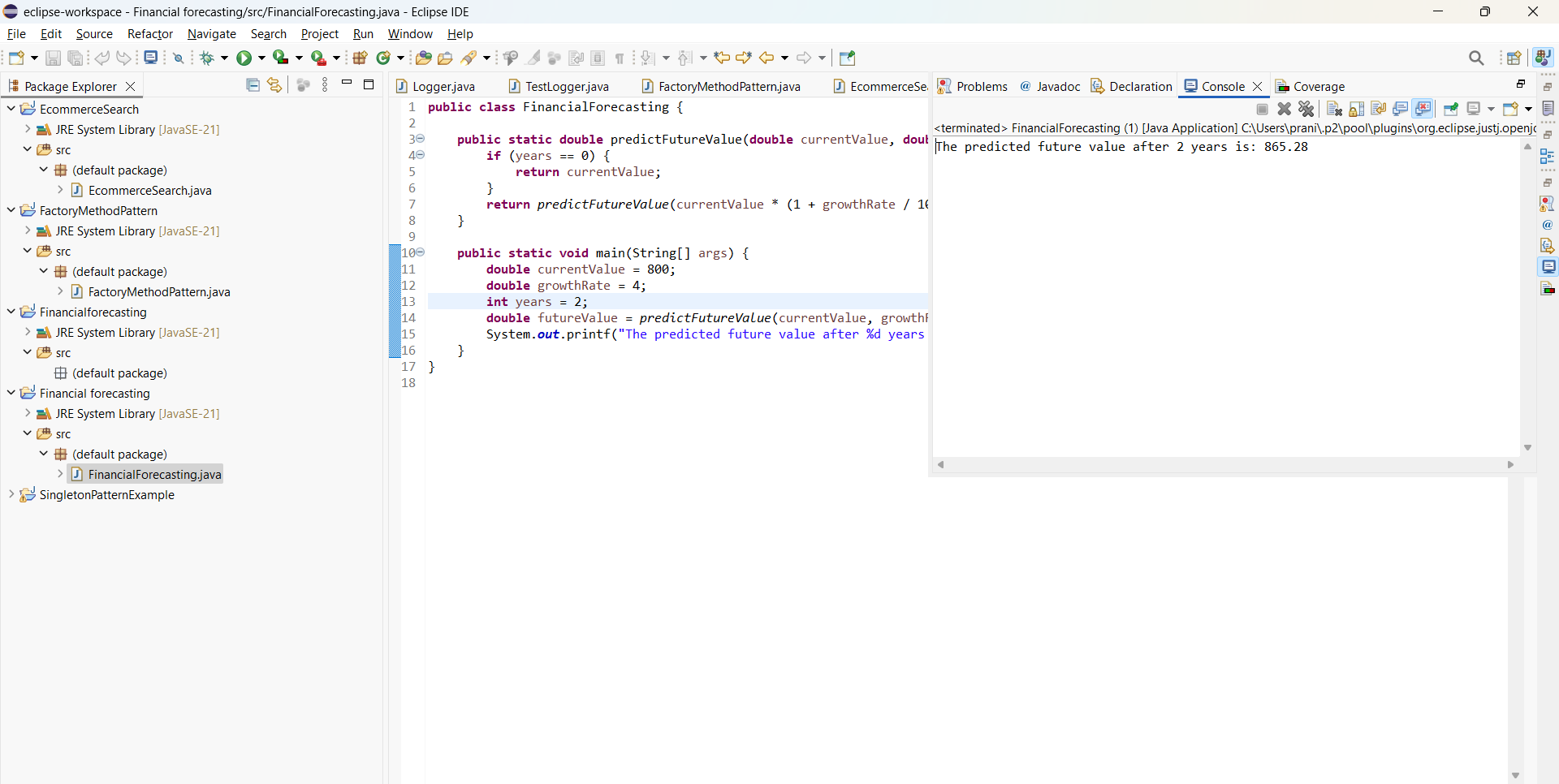
**double** futureValue = *predictFutureValue*(currentValue, growthRate, years);

System.***out***.printf("The predicted future value after %d years is: %.2f%n", years, futureValue);

}

}

**OUTPUT:**

****

## **ADDITIONAL MODULES:**

### **Adapter Pattern:**

**SOURCE CODE:**

**public** **class** AdapterPattern{

**interface** PaymentProcessor {

**void** processPayment(**double** amount);

}

**static** **class** PhonePe {

**void** transferAmount(**double** amount) {

System.***out***.println("Payment of ₹" + amount + " processed using PhonePe.");

}

}

**static** **class** GooglePay {

**void** payAmount(**double** amount) {

System.***out***.println("Payment of ₹" + amount + " processed using GooglePay.");

}

}

**static** **class** PhonePeAdapter **implements** PaymentProcessor {

**private** **final** PhonePe phonePe;

PhonePeAdapter(PhonePe phonePe) {

**this**.phonePe = phonePe;

}

**public** **void** processPayment(**double** amount) {

phonePe.transferAmount(amount);

}

}

**static** **class** GooglePayAdapter **implements** PaymentProcessor {

**private** **final** GooglePay googlePay;

GooglePayAdapter(GooglePay googlePay) {

**this**.googlePay = googlePay;

}

**public** **void** processPayment(**double** amount) {

googlePay.payAmount(amount);

}

}

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

PaymentProcessor phonePeProcessor = **new** PhonePeAdapter(**new** PhonePe());

phonePeProcessor.processPayment(1500.0);

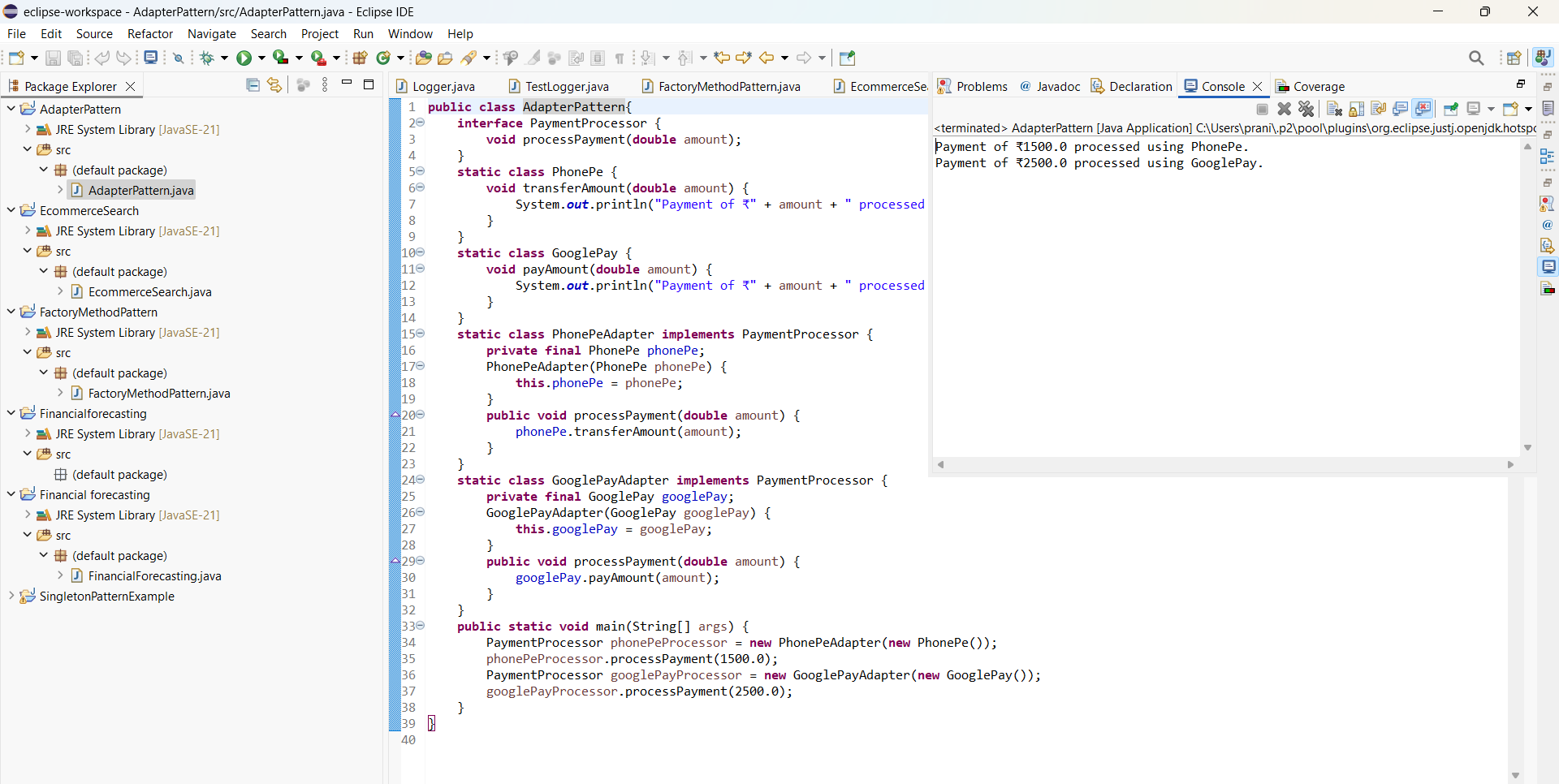
PaymentProcessor googlePayProcessor = **new** GooglePayAdapter(**new** GooglePay());

googlePayProcessor.processPayment(2500.0);

}

}

**OUTPUT:**

****