**Module 1 - Design Patterns and Principles**

**Exercise 1**

***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();Module

        }

        return instance;

    }

    public void log(*String* message) {

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

    }

}

***TestLogger.java***

public class *TestLogger* {

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

*Logger* logger1 = *Logger*.getInstance();

*Logger* logger2 = *Logger*.getInstance();

        logger1.log("First log message.");

        logger2.log("Second log message.");

*System*.out.println("Are logger1 and logger2 the same instance? " + (logger1 == logger2));

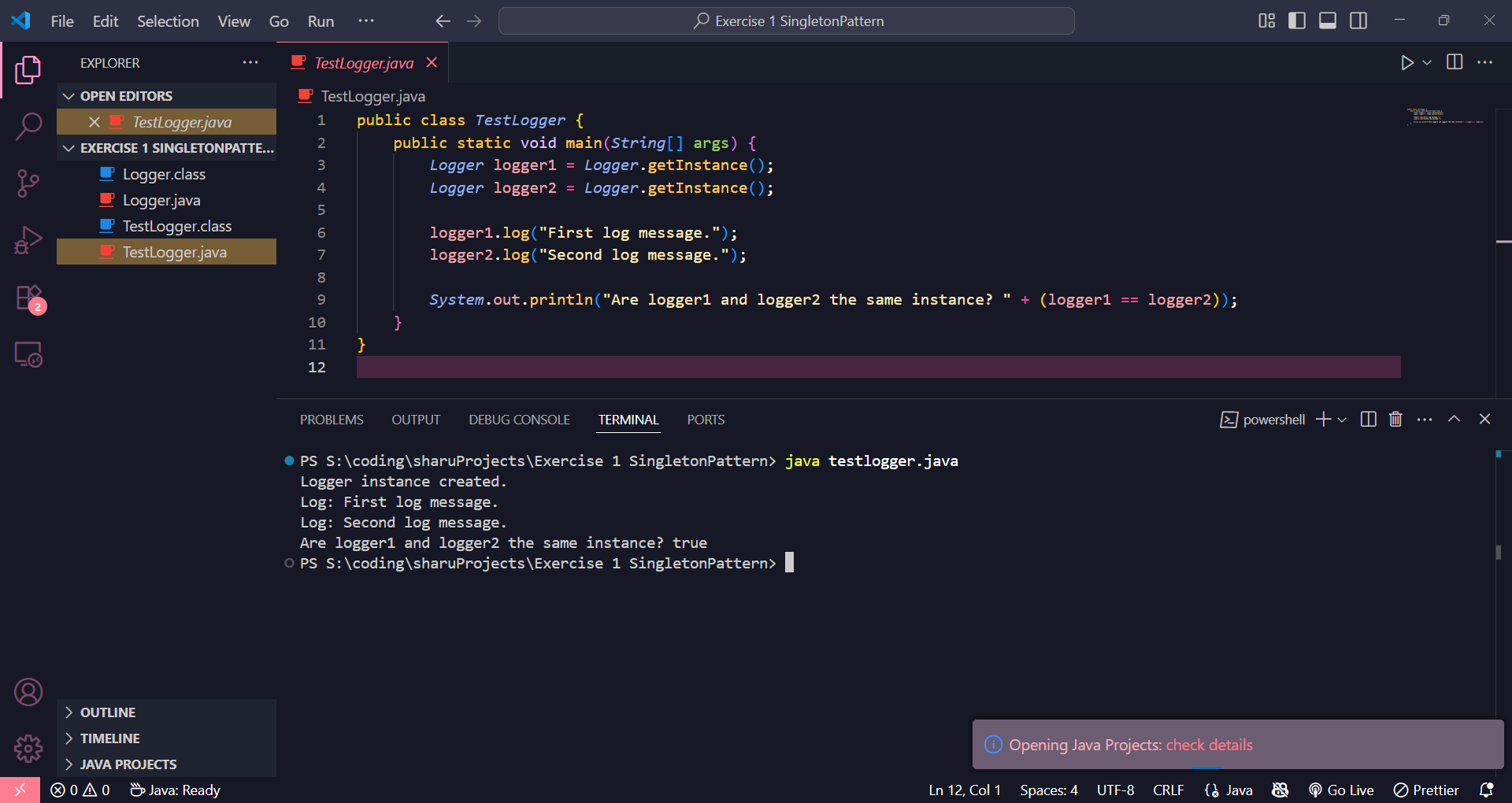
    }

}

**For Compile javac \***

**To Run java *TestLogger.java***

**Result**

******

**Exercise 2**

**Document.java**

public interface Document {

void open();

}

**WordDocument.java**

public class WordDocument implements Document {

@Override

public void open() {

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

}

}

**PdfDocument.java**

public class PdfDocument implements Document {

@Override

public void open() {

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

}

}

**ExcelDocument.java**

public class ExcelDocument implements Document {

@Override

public void open() {

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

}

}

**DocumentFactory.java**

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**WordDocumentFactory.java**

public class WordDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

**PdfDocumentFactory.java**

public class PdfDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

**ExcelDocumentFactory.java**

public class ExcelDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

**TestFactoryMethod.java**

public class TestFactoryMethod {

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();

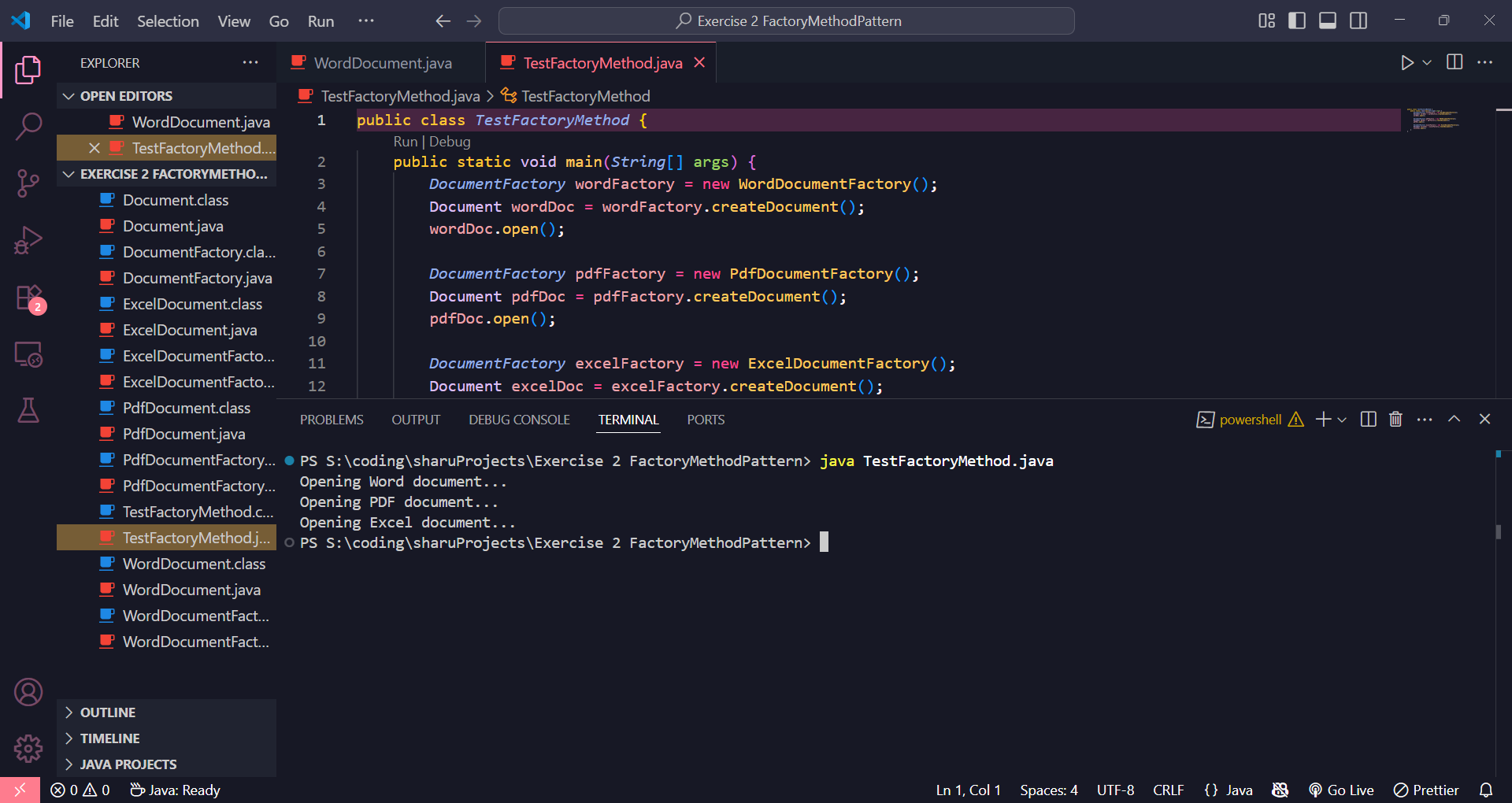
}

}

**For Compile javac \***

**To Run java *TestFactoryMethod.java***

**Result**

****

**Exercise 3**

**Computer.java**

public class Computer {

// Attributes

private String cpu;

private String ram;

private String storage;

// Private constructor

private Computer(Builder builder) {

this.cpu = builder.cpu;

this.ram = builder.ram;

this.storage = builder.storage;

}

// toString method to print Computer details

public String toString() {

return "CPU=" + cpu + ", RAM=" + ram + ", Storage=" + storage;

}

// Static Builder Class

public static class Builder {

private String cpu;

private String ram;

private String storage;

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 Computer build() {

return new Computer(this);

}

}

}

**TestBuilder.java**

public class TestBuilder {

public static void main(String[] args) {

Computer comp1 = new Computer.Builder()

.setCPU("Intel i5")

.setRAM("8GB")

.setStorage("512GB SSD")

.build();

Computer comp2 = new Computer.Builder()

.setCPU("Ryzen 7")

.setRAM("16GB")

.setStorage("1TB HDD")

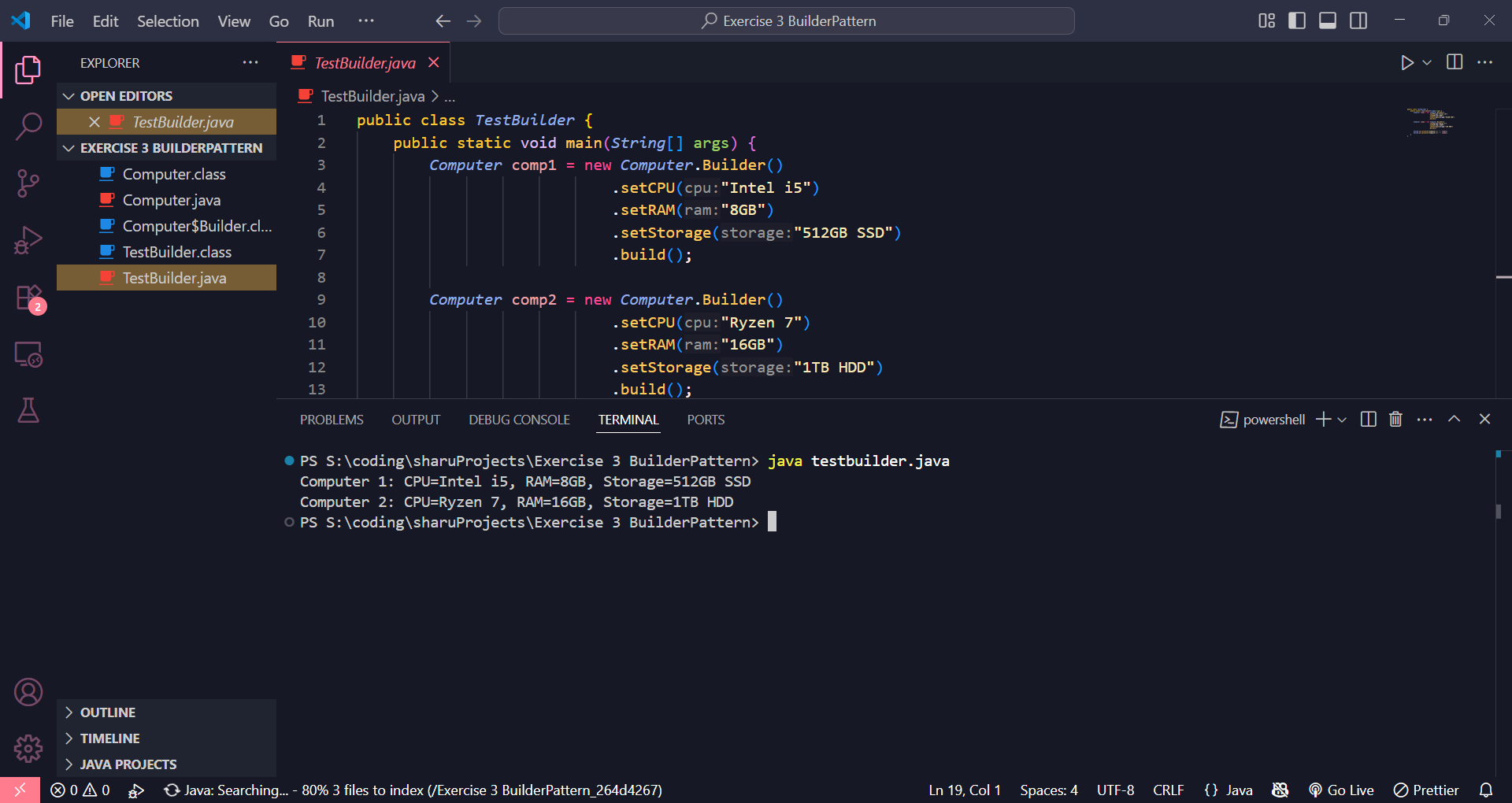
.build();

System.out.println("Computer 1: " + comp1);

System.out.println("Computer 2: " + comp2);

}

}



**Exercise 4**

**1. PaymentProcessor.java**

java

public interface PaymentProcessor {

void processPayment(double amount);

}

**2. PayPalGateway.java**

java

public class PayPalGateway {

public void makePayPalPayment(double amount) {

System.out.println("Paid " + amount + " using PayPal.");

}

}

**3. StripeGateway.java**

java

public class StripeGateway {

public void makeStripeCharge(double amount) {

System.out.println("Charged " + amount + " using Stripe.");

}

}

**4. PayPalAdapter.java**

java

public class PayPalAdapter implements PaymentProcessor {

private PayPalGateway paypal = new PayPalGateway();

@Override

public void processPayment(double amount) {

paypal.makePayPalPayment(amount);

}

}

**5. StripeAdapter.java**

java

public class StripeAdapter implements PaymentProcessor {

private StripeGateway stripe = new StripeGateway();

@Override

public void processPayment(double amount) {

stripe.makeStripeCharge(amount);

}

}

**6. TestAdapter.java**

java

public class TestAdapter {

public static void main(String[] args) {

PaymentProcessor paypalProcessor = new PayPalAdapter();

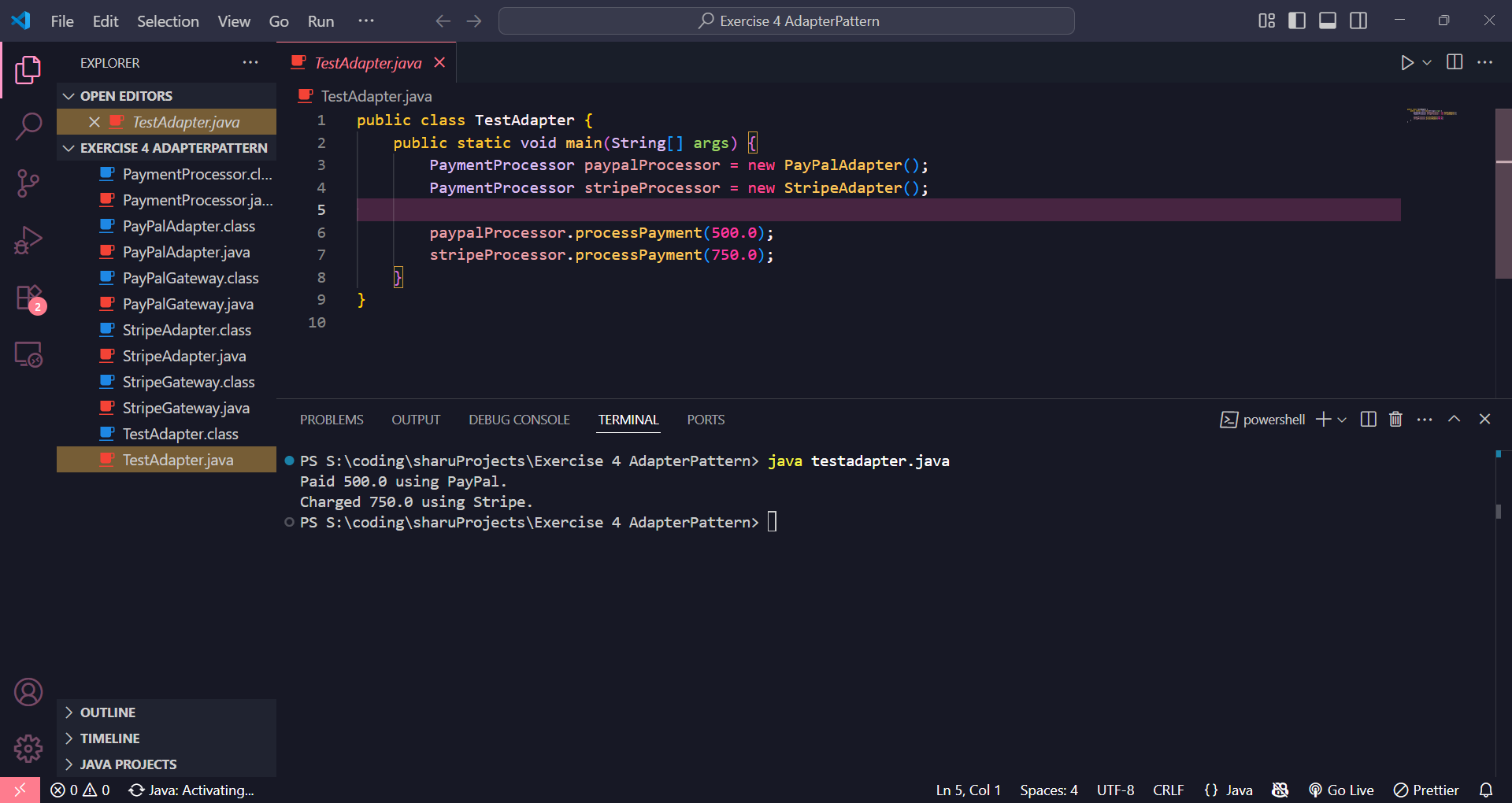
PaymentProcessor stripeProcessor = new StripeAdapter();

paypalProcessor.processPayment(500.0);

stripeProcessor.processPayment(750.0);

}

}



**Exercise 5**

**1. Notifier.java**

java

public interface Notifier {

void send(String message);

}

**2. EmailNotifier.java**

java

public class EmailNotifier implements Notifier {

@Override

public void send(String message) {

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

}

}

**3. NotifierDecorator.java**

java

public abstract class NotifierDecorator implements Notifier {

protected Notifier notifier;

public NotifierDecorator(Notifier notifier) {

this.notifier = notifier;

}

@Override

public void send(String message) {

notifier.send(message); // delegate to base notifier

}

}

**4. SMSNotifierDecorator.java**

java

public class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

@Override

public void send(String message) {

super.send(message); // Send base notification

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

}

}

**5. SlackNotifierDecorator.java**

java

public class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

@Override

public void send(String message) {

super.send(message); // Send previous channels

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

}

}

**6. TestDecorator.java**

java

public class TestDecorator {

public static void main(String[] args) {

// Base: only email

Notifier basicNotifier = new EmailNotifier();

// Email + SMS

Notifier smsNotifier = new SMSNotifierDecorator(basicNotifier);

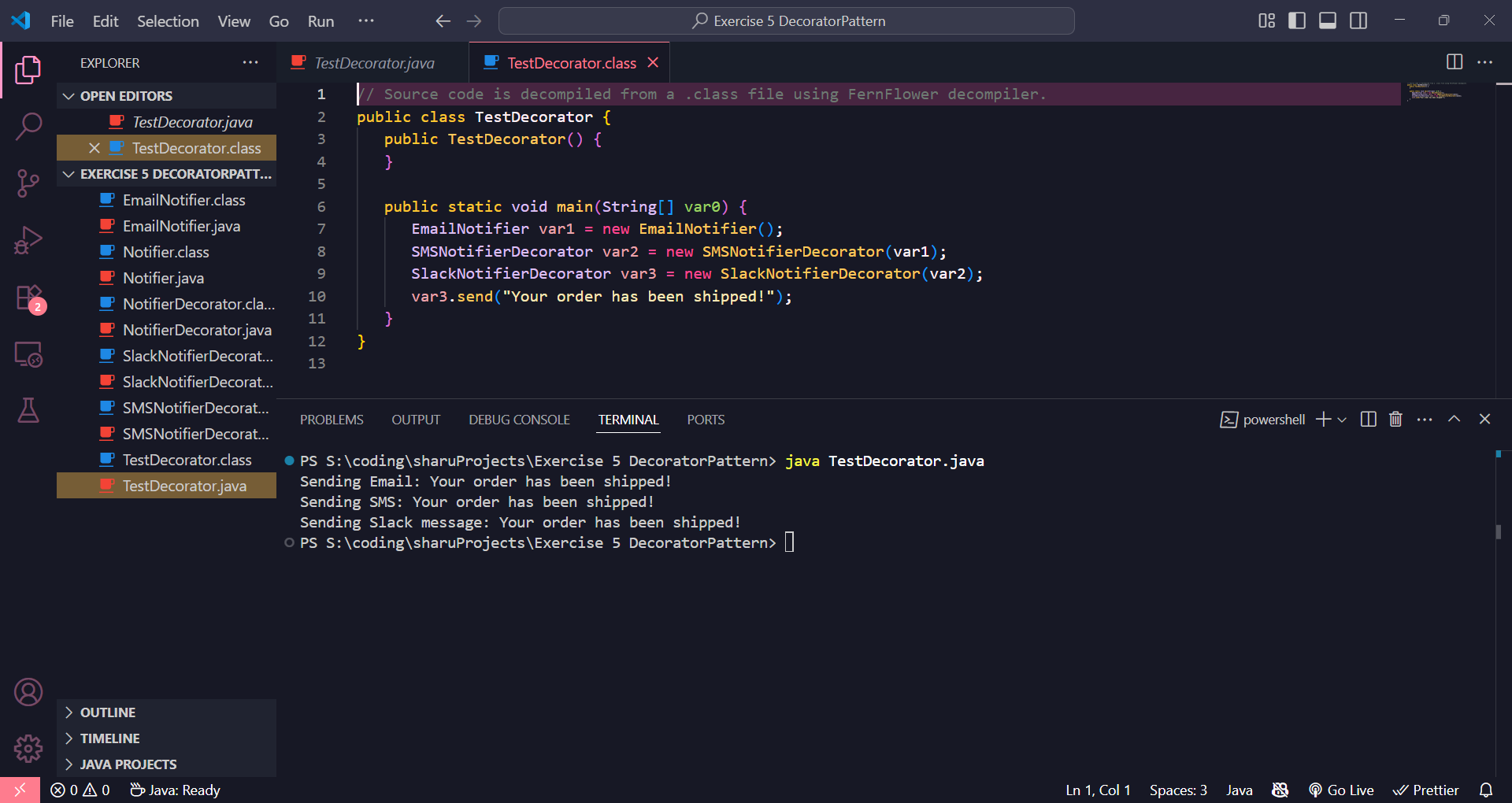
// Email + SMS + Slack

Notifier fullNotifier = new SlackNotifierDecorator(smsNotifier);

fullNotifier.send("Your order has been shipped!");

}

}



**Exercise 6**

**1. Image.java (Interface)**

java

public interface Image {

void display();

}

**2. RealImage.java (Heavy class)**

java

public class RealImage implements Image {

private String filename;

public RealImage(String filename) {

this.filename = filename;

loadFromServer();

}

private void loadFromServer() {

System.out.println("Loading image from server: " + filename);

}

@Override

public void display() {

System.out.println("Displaying: " + filename);

}

}

**3. ProxyImage.java (Proxy class)**

java

public class ProxyImage implements Image {

private RealImage realImage;

private String filename;

public ProxyImage(String filename) {

this.filename = filename;

}

@Override

public void display() {

if (realImage == null) {

realImage = new RealImage(filename); // Lazy load

}

realImage.display();

}

}

**4. TestProxy.java (Main class)**

java

public class TestProxy {

public static void main(String[] args) {

Image image1 = new ProxyImage("pic1.jpg");

Image image2 = new ProxyImage("pic2.jpg");

System.out.println("Image1 first call:");

image1.display(); // Loads + displays

System.out.println("\nImage1 second call:");

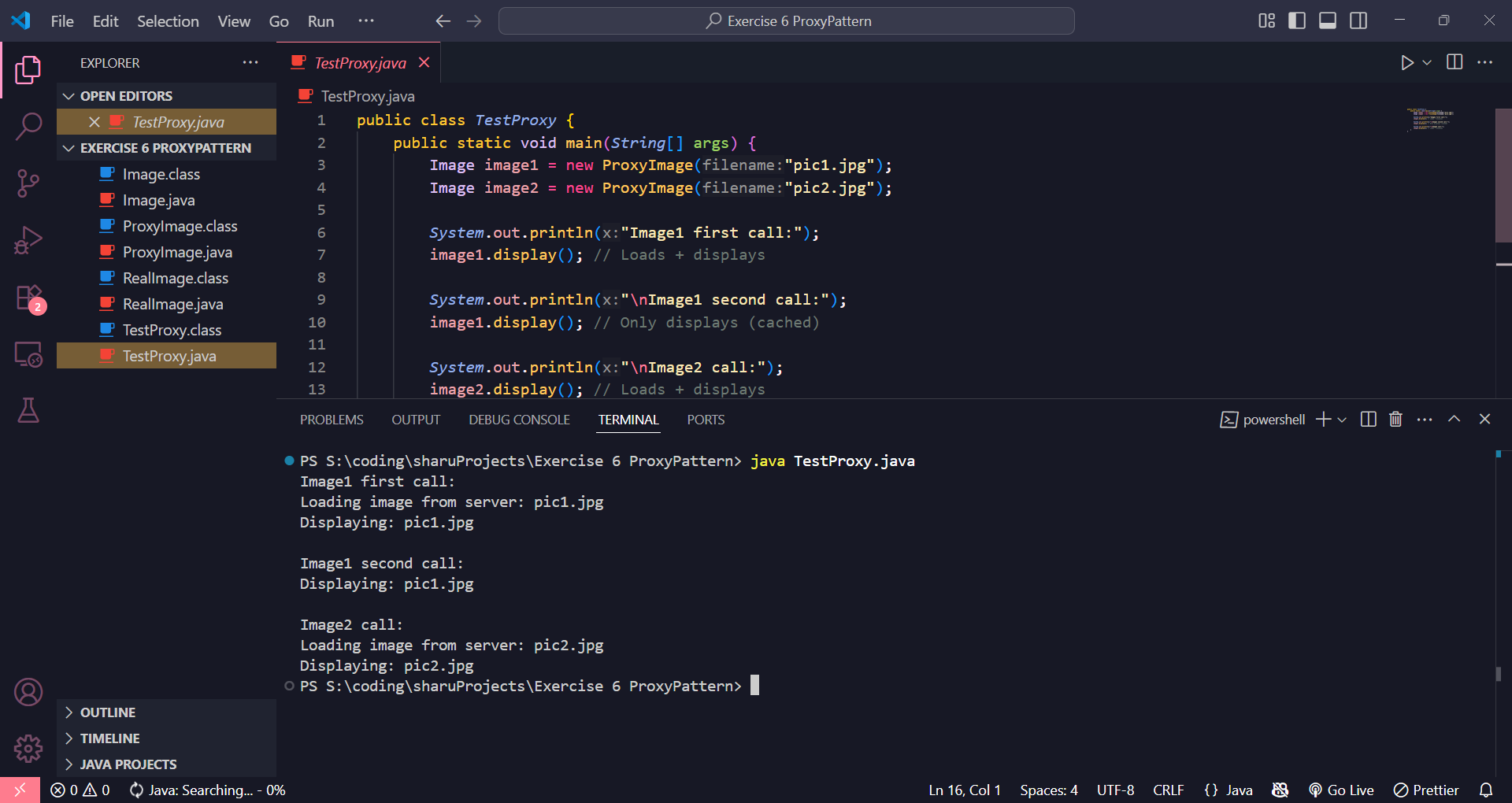
image1.display(); // Only displays (cached)

System.out.println("\nImage2 call:");

image2.display(); // Loads + displays

}

}



**Exercise 7**

**1. Stock.java (Subject interface)**

java

public interface Stock {

void registerObserver(Observer o);

void removeObserver(Observer o);

void notifyObservers();

}

**2. Observer.java (Observer interface)**

java

public interface Observer {

void update(String stockName, double price);

}

**3. StockMarket.java (Concrete subject)**

java

import java.util.\*;

public class StockMarket implements Stock {

private List<Observer> observers = new ArrayList<>();

private String stockName;

private double stockPrice;

public void setStock(String name, double price) {

this.stockName = name;

this.stockPrice = price;

notifyObservers();

}

@Override

public void registerObserver(Observer o) {

observers.add(o);

}

@Override

public void removeObserver(Observer o) {

observers.remove(o);

}

@Override

public void notifyObservers() {

for (Observer o : observers) {

o.update(stockName, stockPrice);

}

}

}

**4. MobileApp.java (Concrete observer)**

java

public class MobileApp implements Observer {

private String name;

public MobileApp(String name) {

this.name = name;

}

@Override

public void update(String stockName, double price) {

System.out.println("MobileApp [" + name + "]: " + stockName + " price updated to " + price);

}

}

**5. WebApp.java (Concrete observer)**

java

public class WebApp implements Observer {

private String name;

public WebApp(String name) {

this.name = name;

}

@Override

public void update(String stockName, double price) {

System.out.println("WebApp [" + name + "]: " + stockName + " price updated to " + price);

}

}

**6. TestObserver.java (Main class)**

java

public class TestObserver {

public static void main(String[] args) {

StockMarket stockMarket = new StockMarket();

Observer mobile1 = new MobileApp("JD's Mobile");

Observer web1 = new WebApp("JD's Web");

stockMarket.registerObserver(mobile1);

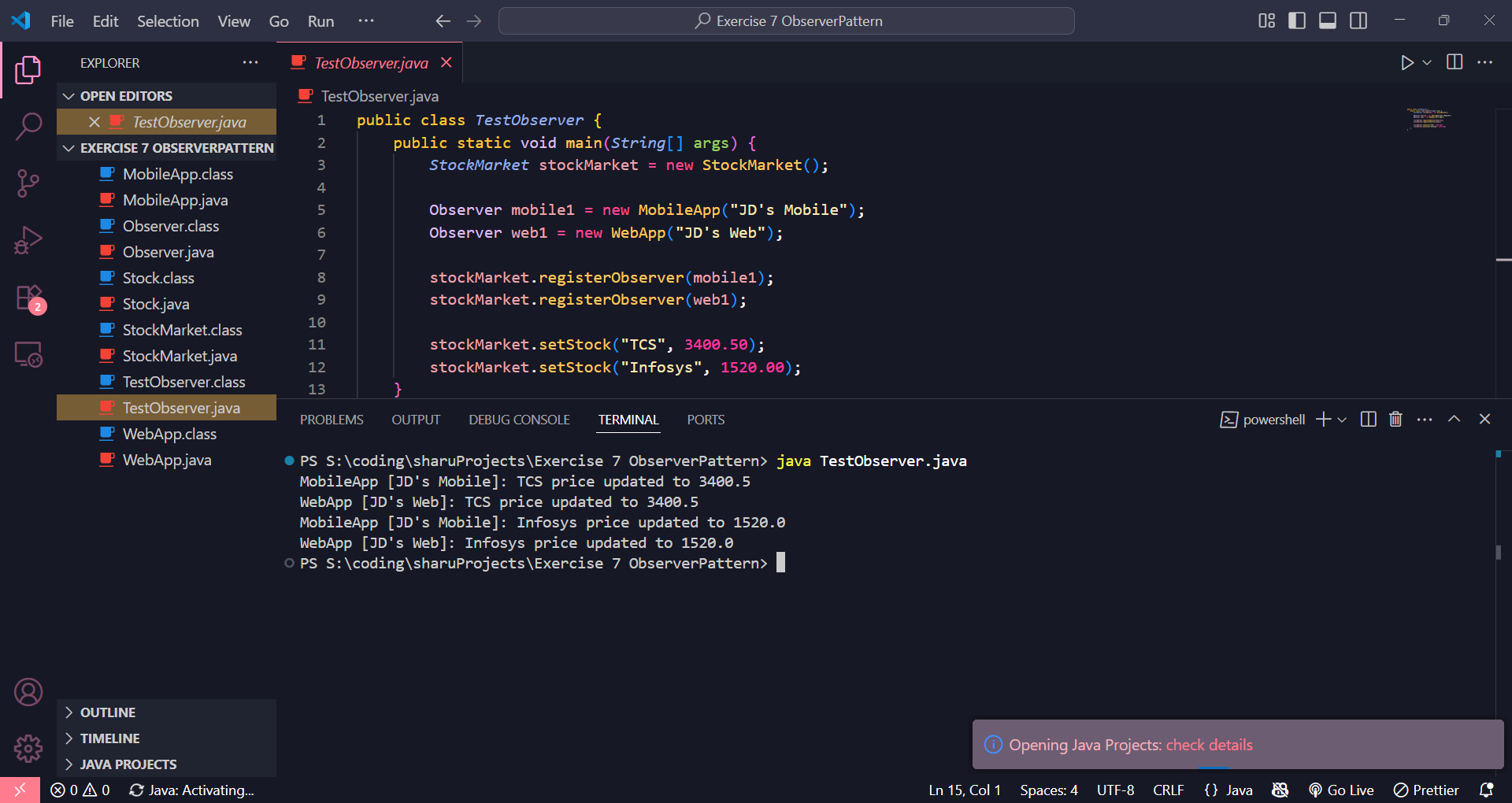
stockMarket.registerObserver(web1);

stockMarket.setStock("TCS", 3400.50);

stockMarket.setStock("Infosys", 1520.00);

}

}



**Exercise 8**

**1. PaymentStrategy.java (Interface)**

java

public interface PaymentStrategy {

void pay(double amount);

}

**2. CreditCardPayment.java (Concrete strategy)**

java

public class CreditCardPayment implements PaymentStrategy {

private String cardNumber;

public CreditCardPayment(String cardNumber) {

this.cardNumber = cardNumber;

}

@Override

public void pay(double amount) {

System.out.println("Paid ₹" + amount + " using Credit Card: " + cardNumber);

}

}

**3. PayPalPayment.java (Concrete strategy)**

java

public class PayPalPayment implements PaymentStrategy {

private String email;

public PayPalPayment(String email) {

this.email = email;

}

@Override

public void pay(double amount) {

System.out.println("Paid ₹" + amount + " using PayPal: " + email);

}

}

**4. PaymentContext.java (Context class)**

java

public class PaymentContext {

private PaymentStrategy paymentStrategy;

public void setPaymentStrategy(PaymentStrategy strategy) {

this.paymentStrategy = strategy;

}

public void pay(double amount) {

if (paymentStrategy != null) {

paymentStrategy.pay(amount);

} else {

System.out.println("Payment method not set!");

}

}

}

**5. TestStrategy.java (Main test class)**

java

public class TestStrategy {

public static void main(String[] args) {

PaymentContext context = new PaymentContext();

// Pay with credit card

context.setPaymentStrategy(new CreditCardPayment("1234-5678-9012-3456"));

context.pay(1000.00);

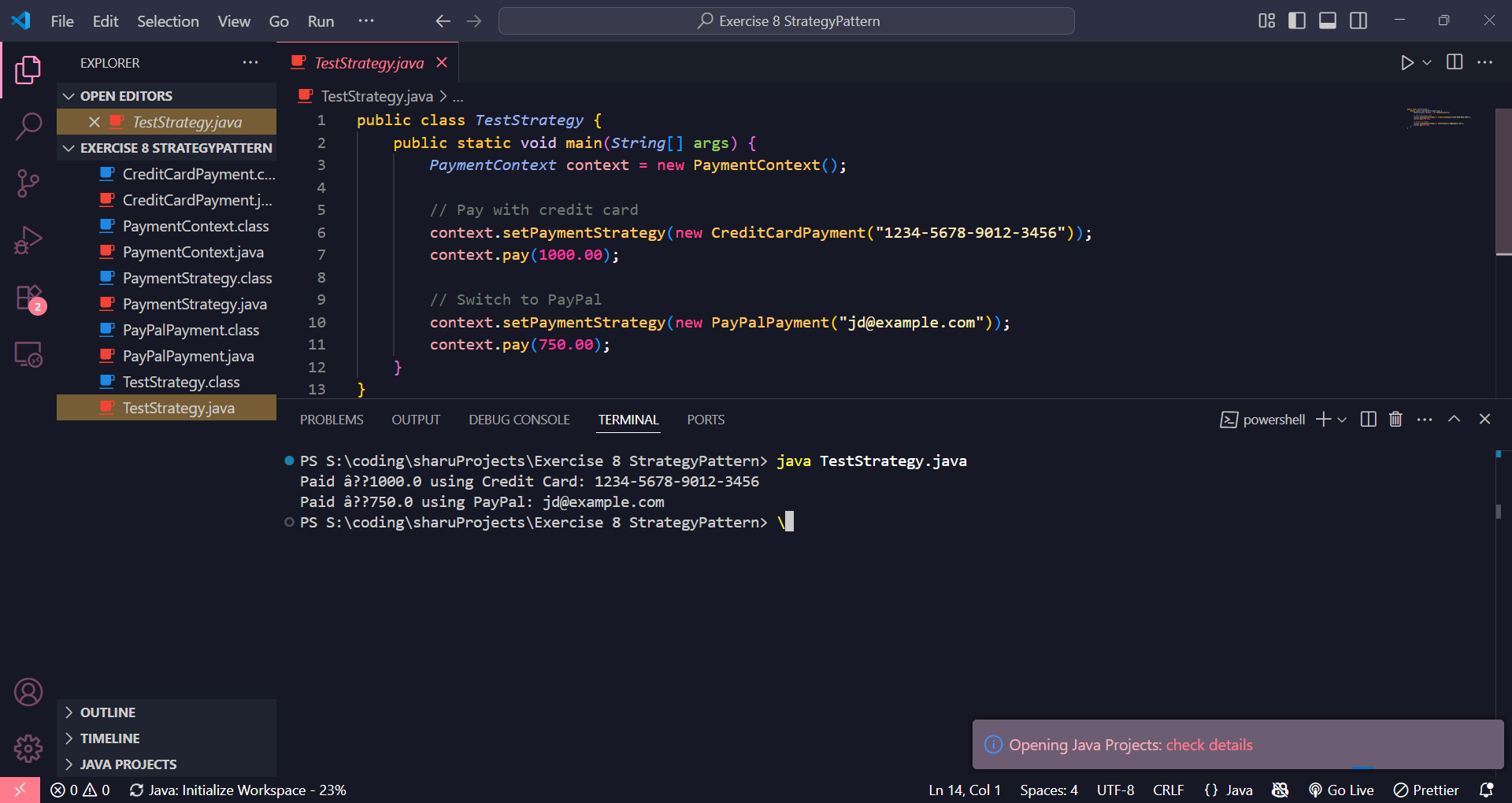
// Switch to PayPal

context.setPaymentStrategy(new PayPalPayment("jd@example.com"));

context.pay(750.00);

}

}



**Exercise 9**

**1. Command.java (Command interface)**

java

public interface Command {

void execute();

}

**2. Light.java (Receiver class)**

java

public class Light {

public void turnOn() {

System.out.println("Light is ON");

}

public void turnOff() {

System.out.println("Light is OFF");

}

}

**3. LightOnCommand.java (Concrete command)**

java

public class LightOnCommand implements Command {

private Light light;

public LightOnCommand(Light light) {

this.light = light;

}

@Override

public void execute() {

light.turnOn();

}

}

**4. LightOffCommand.java (Concrete command)**

java

public class LightOffCommand implements Command {

private Light light;

public LightOffCommand(Light light) {

this.light = light;

}

@Override

public void execute() {

light.turnOff();

}

}

**5. RemoteControl.java (Invoker)**

java

public class RemoteControl {

private Command command;

public void setCommand(Command command) {

this.command = command;

}

public void pressButton() {

if (command != null) {

command.execute();

} else {

System.out.println("No command set!");

}

}

}

**6. TestCommand.java (Main test class)**

java

public class TestCommand {

public static void main(String[] args) {

Light livingRoomLight = new Light();

Command lightOn = new LightOnCommand(livingRoomLight);

Command lightOff = new LightOffCommand(livingRoomLight);

RemoteControl remote = new RemoteControl();

remote.setCommand(lightOn);

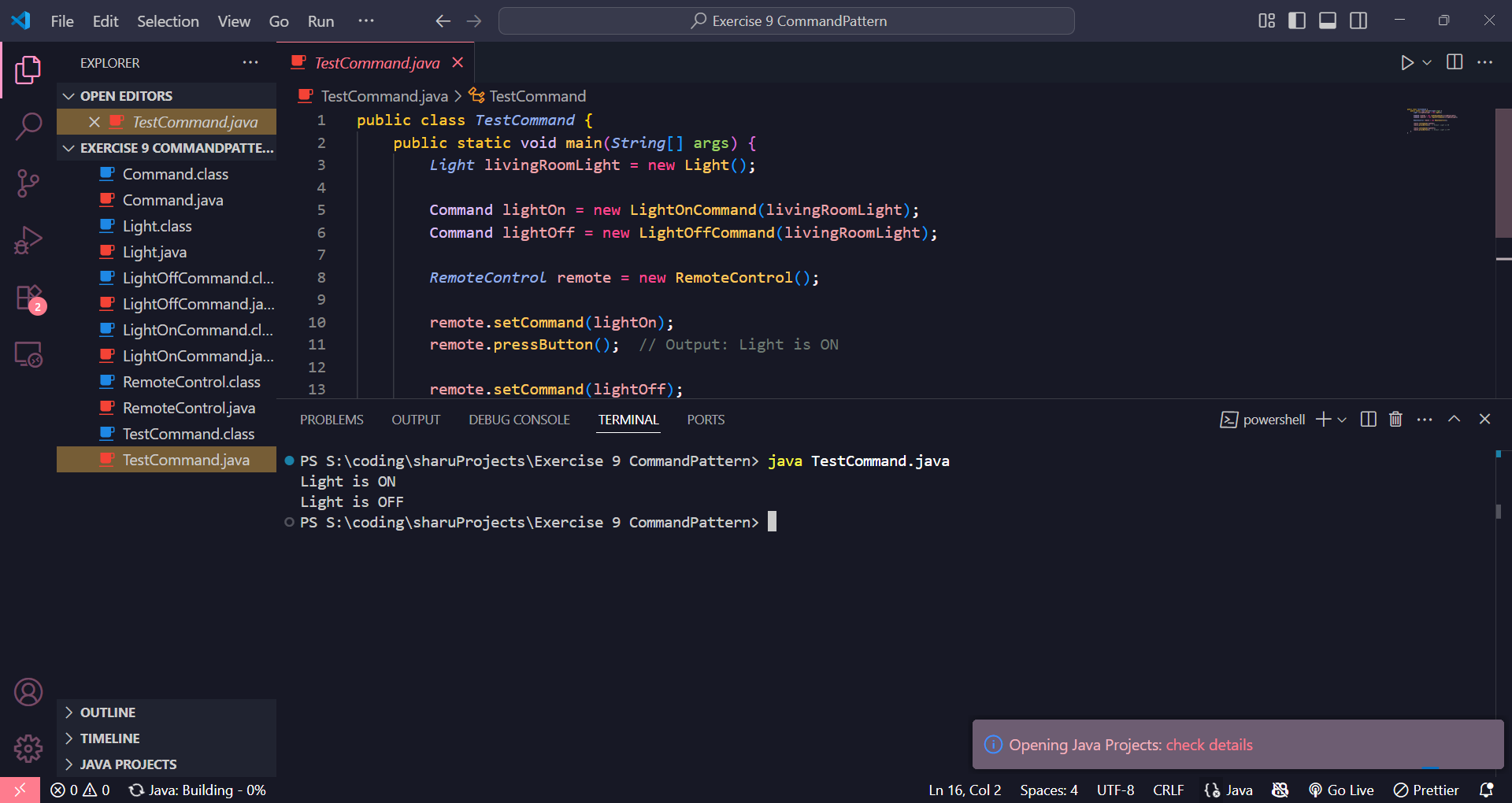
remote.pressButton(); // Output: Light is ON

remote.setCommand(lightOff);

remote.pressButton(); // Output: Light is OFF

}

}



**Exercise 10**

**1. Student.java (Model)**

java

public class Student {

private String name;

private String id;

private String grade;

public Student(String name, String id, String grade) {

this.name = name;

this.id = id;

this.grade = grade;

}

// Getters

public String getName() { return name; }

public String getId() { return id; }

public String getGrade() { return grade; }

// Setters

public void setName(String name) { this.name = name; }

public void setId(String id) { this.id = id; }

public void setGrade(String grade) { this.grade = grade; }

}

**2. StudentView.java (View)**

java

public class StudentView {

public void displayStudentDetails(String name, String id, String grade) {

System.out.println("=== Student Details ===");

System.out.println("Name : " + name);

System.out.println("ID : " + id);

System.out.println("Grade : " + grade);

}

}

**3. StudentController.java (Controller)**

java

public class StudentController {

private Student student;

private StudentView view;

public StudentController(Student student, StudentView view) {

this.student = student;

this.view = view;

}

public void setStudentName(String name) {

student.setName(name);

}

public void setStudentGrade(String grade) {

student.setGrade(grade);

}

public void updateView() {

view.displayStudentDetails(student.getName(), student.getId(), student.getGrade());

}

}

**4. MVCTest.java (Main)**

java

public class MVCTest {

public static void main(String[] args) {

// Step 1: Create the model

Student student = new Student("Sharu", "23CS101", "A");

// Step 2: Create the view

StudentView view = new StudentView();

// Step 3: Create the controller

StudentController controller = new StudentController(student, view);

// Step 4: Initial display

controller.updateView();

// Step 5: Update and display again

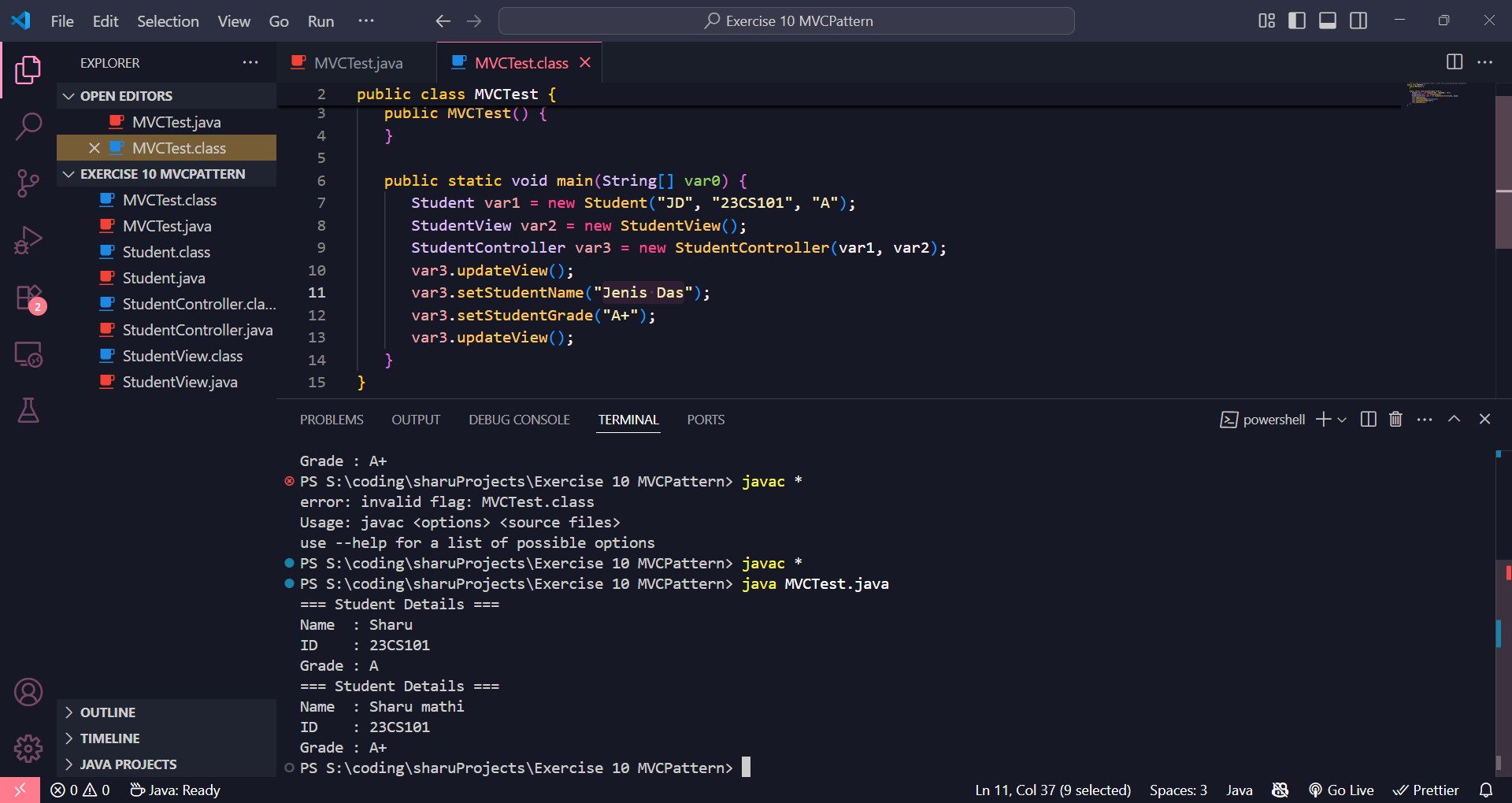
controller.setStudentName("Sharu Mathi");

controller.setStudentGrade("A+");

controller.updateView();

}

}



**Exercise 11**

**1. CustomerRepository.java (Interface)**

java

public interface CustomerRepository {

String findCustomerById(int id);

}

**2. CustomerRepositoryImpl.java (Concrete implementation)**

java

public class CustomerRepositoryImpl implements CustomerRepository {

@Override

public String findCustomerById(int id) {

// For simplicity, returning dummy customer info

if (id == 1) {

return "Customer #1: Sharu, sharu@example.com";

} else if (id == 2) {

return "Customer #2: Alice, alice@example.com";

} else {

return "Customer not found.";

}

}

}

**3. CustomerService.java (Service class with injected dependency)**

java

public class CustomerService {

private CustomerRepository repository;

// Constructor Injection

public CustomerService(CustomerRepository repository) {

this.repository = repository;

}

public void printCustomer(int id) {

String customer = repository.findCustomerById(id);

System.out.println(customer);

}

}

**4. TestDependencyInjection.java (Main class)**

java

public class TestDependencyInjection {

public static void main(String[] args) {

// Create the repository implementation

CustomerRepository repo = new CustomerRepositoryImpl();

// Inject repository into service via constructor

CustomerService service = new CustomerService(repo);

// Use service

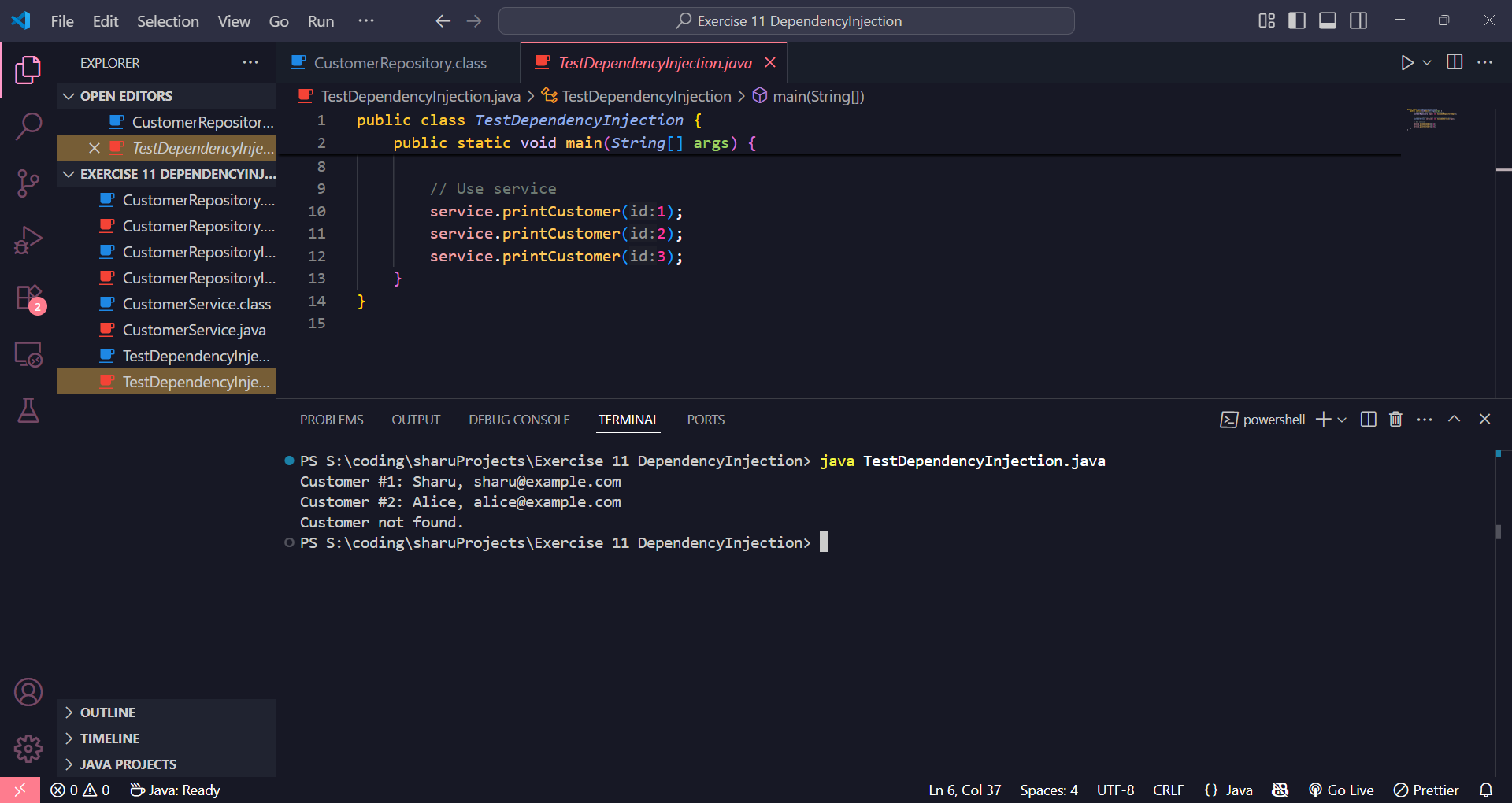
service.printCustomer(1);

service.printCustomer(2);

service.printCustomer(3);

}

}



Module 2 - Data Structures and Algorithms

**Exercise 1: Inventory Management System**

The task is to create an inventory system for a warehouse. We need fast operations to handle large data efficiently.  
Using a **HashMap** ensures quick lookup by productId.

**Product.java**

java

public class Product {

int productId;

String productName;

int quantity;

double price;

public Product(int productId, String productName, int quantity, double price) {

this.productId = productId;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

}

**Main.java**

java

import java.util.HashMap;

public class Main {

static HashMap<Integer, Product> inventory = new HashMap<>();

public static void addProduct(Product p) {

inventory.put(p.productId, p);

}

public static void updateProduct(int id, int quantity, double price) {

Product p = inventory.get(id);

if (p != null) {

p.quantity = quantity;

p.price = price;

}

}

public static void deleteProduct(int id) {

inventory.remove(id);

}

public static void main(String[] args) {

addProduct(new Product(1, "Keyboard", 20, 1200));

addProduct(new Product(2, "Monitor", 15, 7000));

updateProduct(2, 18, 6800);

deleteProduct(1);

for (Product p : inventory.values()) {

System.out.println(p.productId + " " + p.productName + " " + p.quantity + " " + p.price);

}

}

}

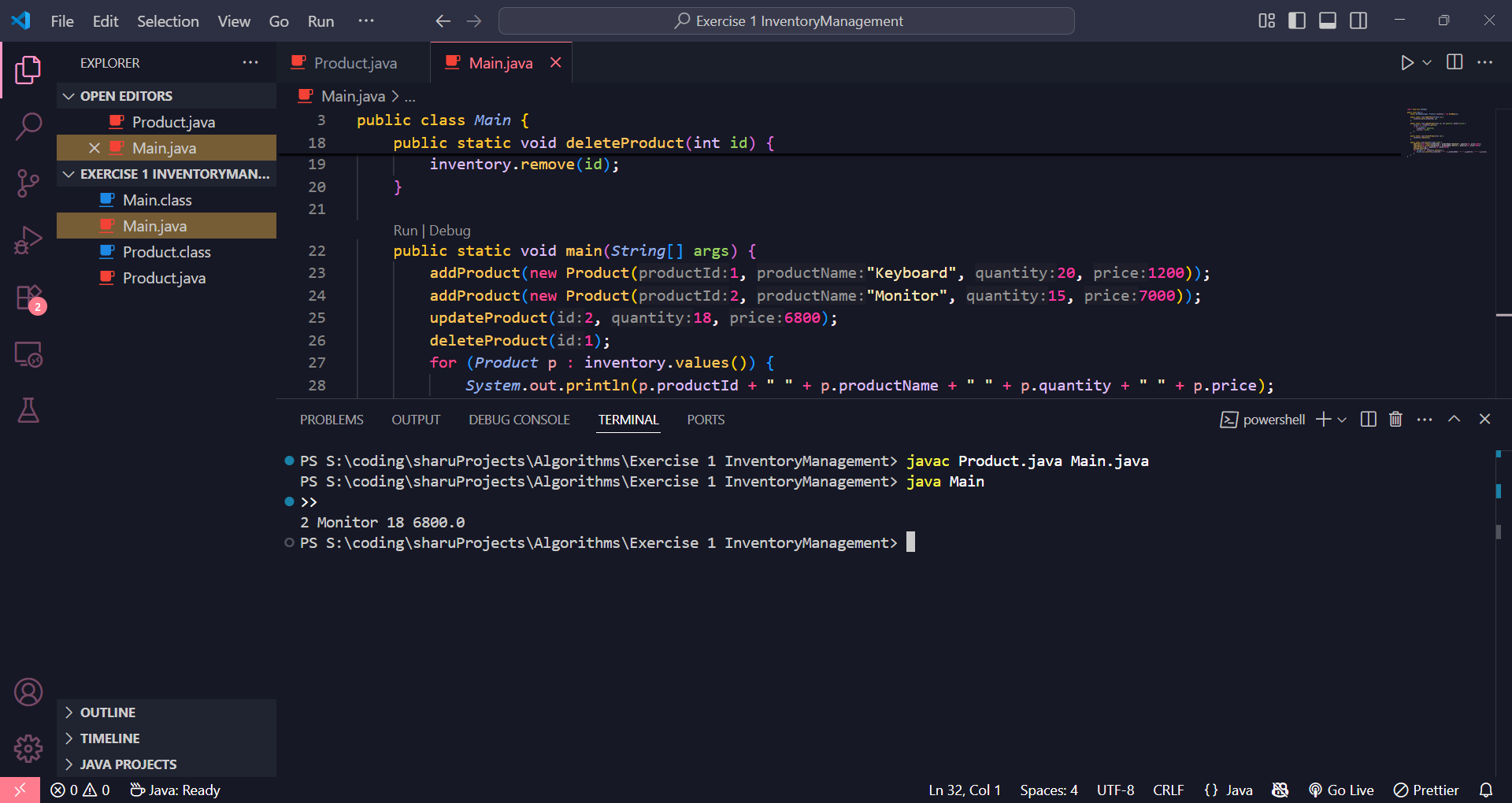
**To Compile**

javac Product.java Main.java

**To Run**

java Main

**Result**

****

**Exercise 2 EcommerceSearch**

We’re building a search feature for an e-commerce platform.  
Two types of searches are used: Linear Search for unsorted data and Binary Search for sorted data.  
We compare both to understand performance differences using Big O notation.

**Product.java**

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;

}

}

**Main.java**

java

import java.util.Arrays;

import java.util.Comparator;

public class Main {

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

for (int i = 0; i < products.length; i++) {

if (products[i].productName.equalsIgnoreCase(name)) {

return i;

}

}

return -1;

}

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

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

while (low <= high) {

int mid = (low + high) / 2;

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

if (cmp == 0) return mid;

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

else high = mid - 1;

}

return -1;

}

public static void main(String[] args) {

Product[] products = {

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

new Product(2, "Chair", "Furniture"),

new Product(3, "Phone", "Electronics")

};

int index1 = linearSearch(products, "Chair");

System.out.println("Linear Search Index: " + index1);

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

int index2 = binarySearch(products, "Chair");

System.out.println("Binary Search Index: " + index2);

}

}

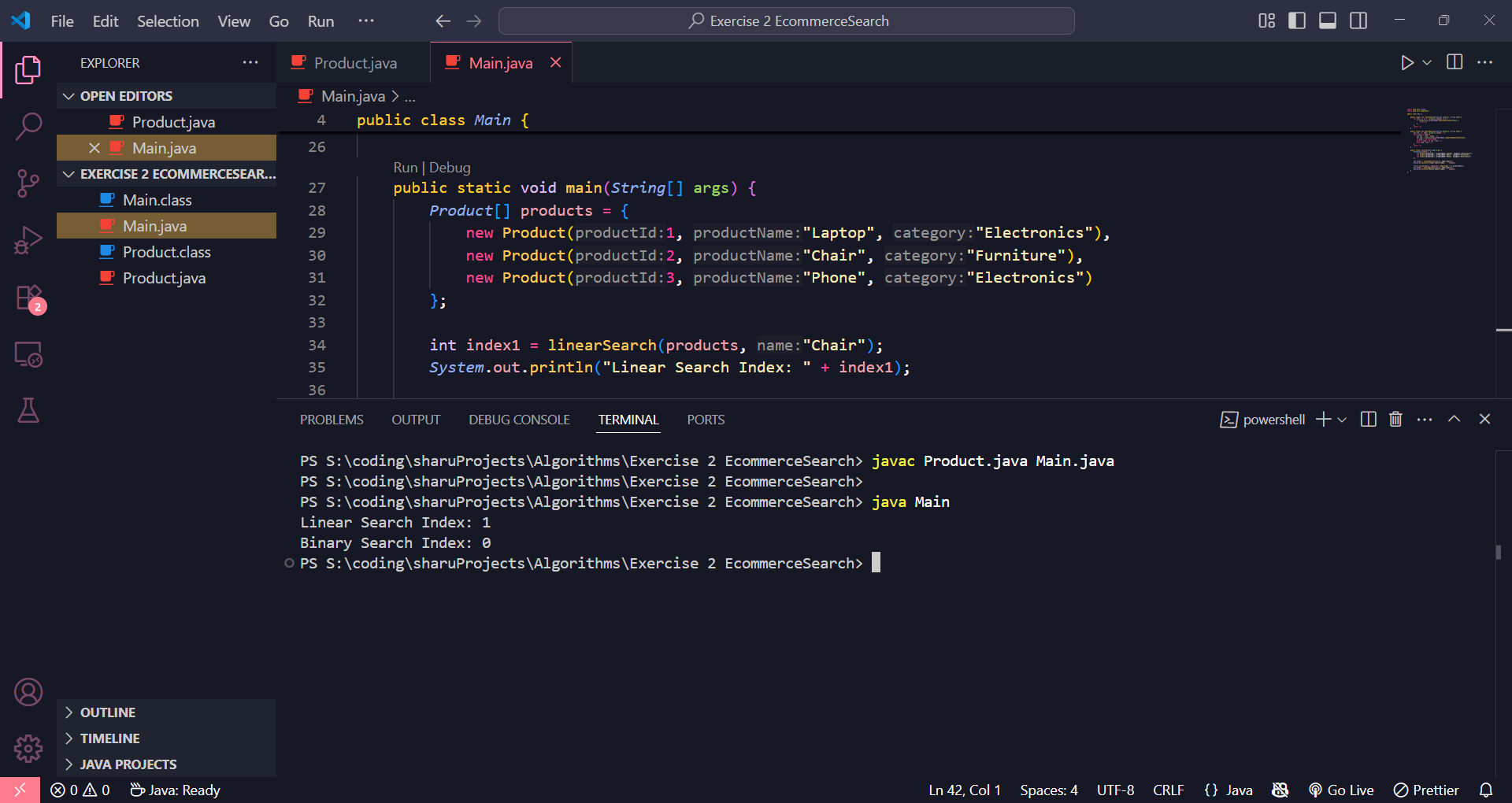
**To Compile**

javac Product.java Main.java

**To Run**

java Main

**Result**



**Exercise 3: Sorting Customer Orders**

**Problem Overview + Solution File Structure**

We need to sort customer orders based on totalPrice to prioritize high-value ones.  
We’ll use two sorting algorithms: **Bubble Sort** and **Quick Sort**, and compare their performance.  
**Bubble Sort** is simple but slow; **Quick Sort** is faster and more efficient for large datasets.

**Order.java**

java

public class Order {

int orderId;

String customerName;

double totalPrice;

public Order(int orderId, String customerName, double totalPrice) {

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

}

**Main.java**

java

public class Main {

public static void bubbleSort(Order[] orders) {

int n = orders.length;

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (orders[j].totalPrice > orders[j + 1].totalPrice) {

Order temp = orders[j];

orders[j] = orders[j + 1];

orders[j + 1] = temp;

}

}

}

}

public static void quickSort(Order[] orders, int low, int high) {

if (low < high) {

int pi = partition(orders, low, high);

quickSort(orders, low, pi - 1);

quickSort(orders, pi + 1, high);

}

}

public static int partition(Order[] orders, int low, int high) {

double pivot = orders[high].totalPrice;

int i = low - 1;

for (int j = low; j < high; j++) {

if (orders[j].totalPrice < pivot) {

i++;

Order temp = orders[i];

orders[i] = orders[j];

orders[j] = temp;

}

}

Order temp = orders[i + 1];

orders[i + 1] = orders[high];

orders[high] = temp;

return i + 1;

}

public static void printOrders(Order[] orders) {

for (Order o : orders) {

System.out.println(o.orderId + " " + o.customerName + " " + o.totalPrice);

}

}

public static void main(String[] args) {

Order[] orders = {

new Order(101, "Alice", 750.5),

new Order(102, "Bob", 1200.0),

new Order(103, "Charlie", 450.0)

};

System.out.println("Before Sorting:");

printOrders(orders);

bubbleSort(orders);

System.out.println("After Bubble Sort:");

printOrders(orders);

orders = new Order[]{

new Order(101, "Alice", 750.5),

new Order(102, "Bob", 1200.0),

new Order(103, "Charlie", 450.0)

};

quickSort(orders, 0, orders.length - 1);

System.out.println("After Quick Sort:");

printOrders(orders);

}

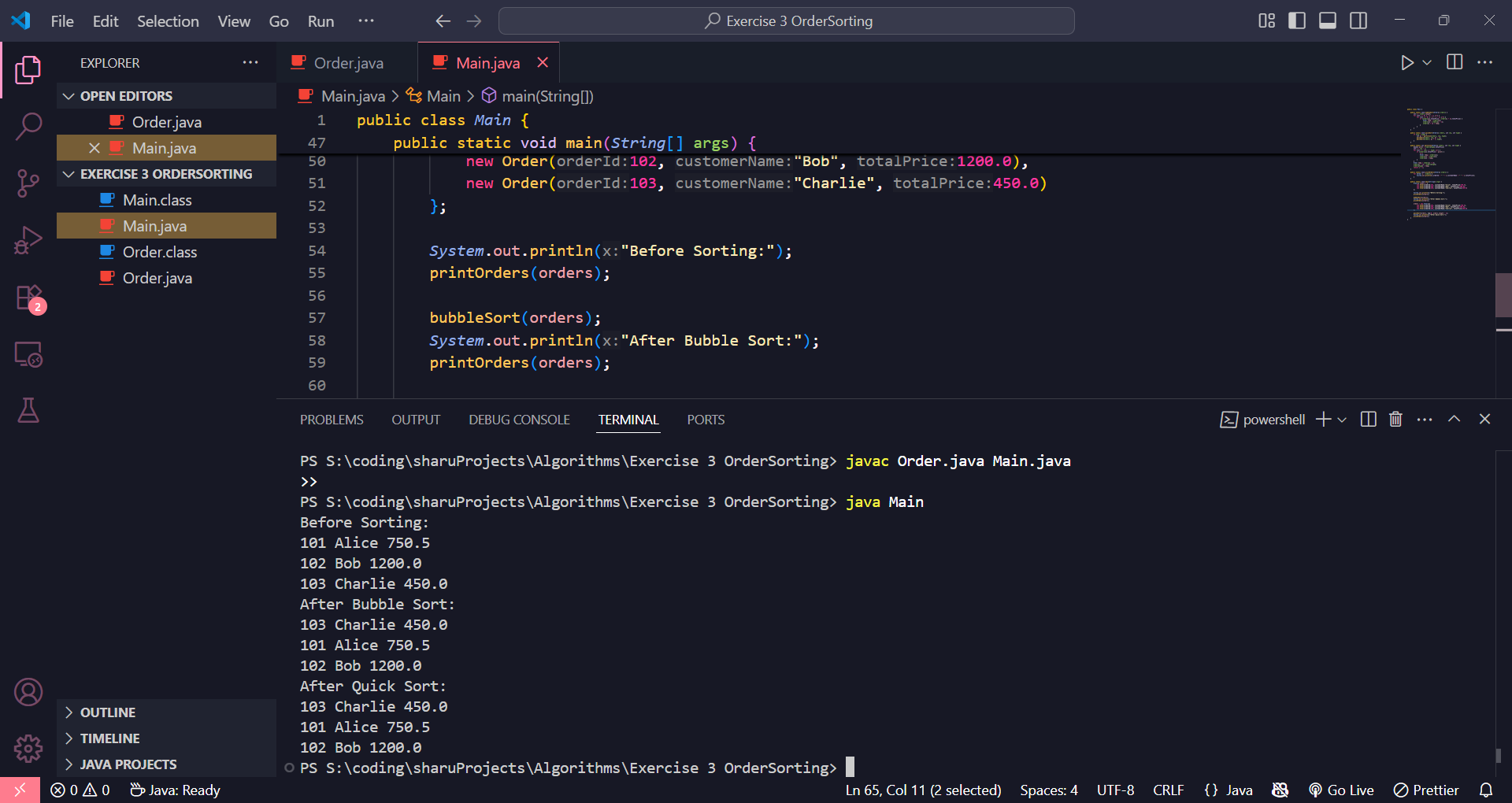
}

**To Compile**

javac Order.java Main.java

**To Run**

java Main

****

**Exercise 4: Employee Management System**

**Problem Overview**

This system manages employee records using a simple array.  
You will store employee data and implement basic operations like add, search, traverse, and delete.  
Arrays offer constant-time access but have fixed sizes and inefficient insert/delete operations.

**Employee.java**

java

public class Employee {

int employeeId;

String name;

String position;

double salary;

public Employee(int employeeId, String name, String position, double salary) {

this.employeeId = employeeId;

this.name = name;

this.position = position;

this.salary = salary;

}

}

**Main.java**

java

public class Main {

static Employee[] employees = new Employee[10];

static int count = 0;

public static void addEmployee(Employee e) {

if (count < employees.length) {

employees[count++] = e;

}

}

public static int searchEmployee(int id) {

for (int i = 0; i < count; i++) {

if (employees[i].employeeId == id) {

return i;

}

}

return -1;

}

public static void deleteEmployee(int id) {

int index = searchEmployee(id);

if (index != -1) {

for (int i = index; i < count - 1; i++) {

employees[i] = employees[i + 1];

}

employees[--count] = null;

}

}

public static void traverseEmployees() {

for (int i = 0; i < count; i++) {

Employee e = employees[i];

System.out.println(e.employeeId + " " + e.name + " " + e.position + " " + e.salary);

}

}

public static void main(String[] args) {

addEmployee(new Employee(1, "Alice", "Manager", 80000));

addEmployee(new Employee(2, "Bob", "Engineer", 60000));

addEmployee(new Employee(3, "Charlie", "Analyst", 50000));

System.out.println("All Employees:");

traverseEmployees();

deleteEmployee(2);

System.out.println("After Deletion:");

traverseEmployees();

}

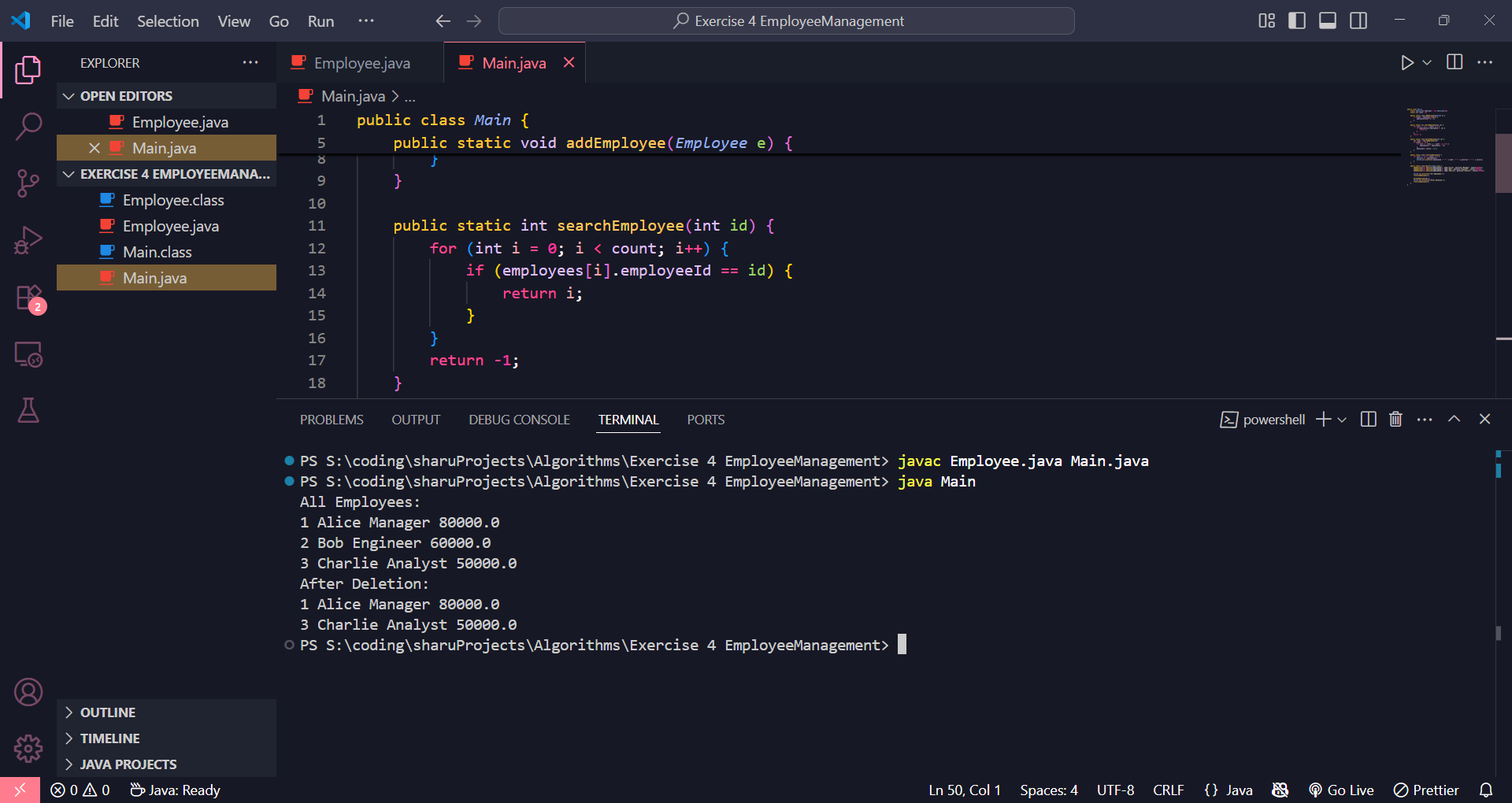
}

**To Compile**

javac Employee.java Main.java

**To Run**

java Main

**Result**

**Exercise 5: Task Management System**

**Problem Overview**

This system manages tasks using a Singly Linked List.  
Linked lists allow efficient dynamic memory usage and easy insert/delete without shifting elements.  
Useful when tasks change frequently and memory flexibility is important.

**Task.java**

java

public class Task {

int taskId;

String taskName;

String status;

public Task(int taskId, String taskName, String status) {

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

}

}

**Node.java**

java

public class Node {

Task task;

Node next;

public Node(Task task) {

this.task = task;

this.next = null;

}

}

**Main.java**

java

public class Main {

static Node head = null;

public static void addTask(Task task) {

Node newNode = new Node(task);

if (head == null) {

head = newNode;

} else {

Node temp = head;

while (temp.next != null) {

temp = temp.next;

}

temp.next = newNode;

}

}

public static Node searchTask(int id) {

Node temp = head;

while (temp != null) {

if (temp.task.taskId == id) {

return temp;

}

temp = temp.next;

}

return null;

}

public static void deleteTask(int id) {

if (head == null) return;

if (head.task.taskId == id) {

head = head.next;

return;

}

Node temp = head;

while (temp.next != null) {

if (temp.next.task.taskId == id) {

temp.next = temp.next.next;

return;

}

temp = temp.next;

}

}

public static void traverseTasks() {

Node temp = head;

while (temp != null) {

Task t = temp.task;

System.out.println(t.taskId + " " + t.taskName + " " + t.status);

temp = temp.next;

}

}

public static void main(String[] args) {

addTask(new Task(1, "Design UI", "Pending"));

addTask(new Task(2, "Implement Backend", "In Progress"));

addTask(new Task(3, "Write Tests", "Pending"));

System.out.println("All Tasks:");

traverseTasks();

deleteTask(2);

System.out.println("After Deletion:");

traverseTasks();

}

}

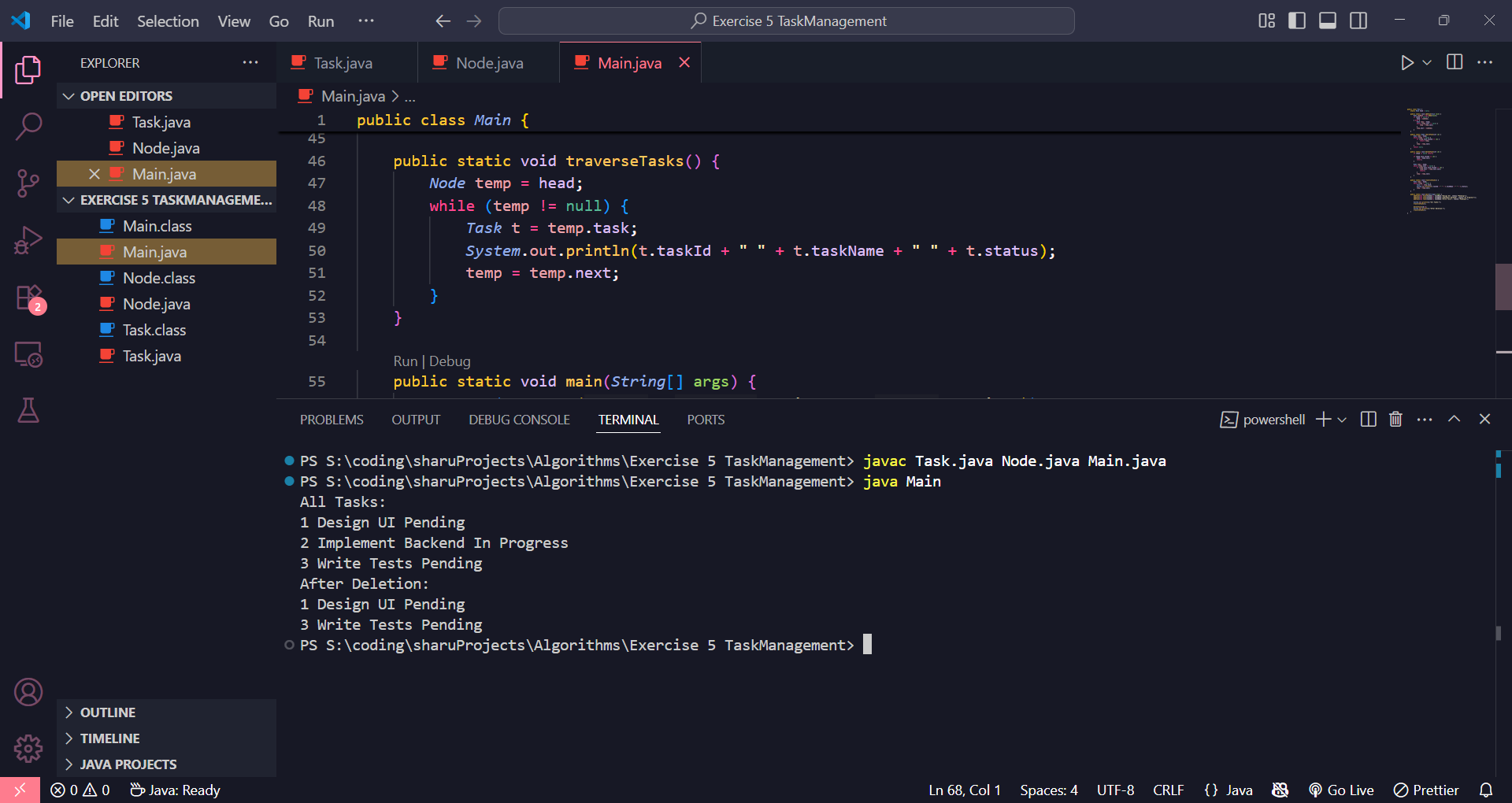
**To Compile**

javac Task.java Node.java Main.java

**To Run**

java Main

**Result**

****

**Exercise 6: Library Management System**

**Problem Overview**

This system allows users to search books by title using both **linear search** and **binary search**.  
Linear search works on unsorted lists, while binary search requires a sorted list and is more efficient for large datasets.

**Book.java**

java

public class Book {

int bookId;

String title;

String author;

public Book(int bookId, String title, String author) {

this.bookId = bookId;

this.title = title;

this.author = author;

}

}

**Main.java**

java

import java.util.Arrays;

import java.util.Comparator;

public class Main {

public static int linearSearch(Book[] books, String title) {

for (int i = 0; i < books.length; i++) {

if (books[i].title.equalsIgnoreCase(title)) {

return i;

}

}

return -1;

}

public static int binarySearch(Book[] books, String title) {

int low = 0, high = books.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

int cmp = books[mid].title.compareToIgnoreCase(title);

if (cmp == 0) return mid;

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

else high = mid - 1;

}

return -1;

}

public static void main(String[] args) {

Book[] books = {

new Book(101, "Java Programming", "Author A"),

new Book(102, "Data Structures", "Author B"),

new Book(103, "Algorithms", "Author C")

};

int indexLinear = linearSearch(books, "Data Structures");

System.out.println("Linear Search Index: " + indexLinear);

Arrays.sort(books, Comparator.comparing(b -> b.title));

int indexBinary = binarySearch(books, "Data Structures");

System.out.println("Binary Search Index: " + indexBinary);

}

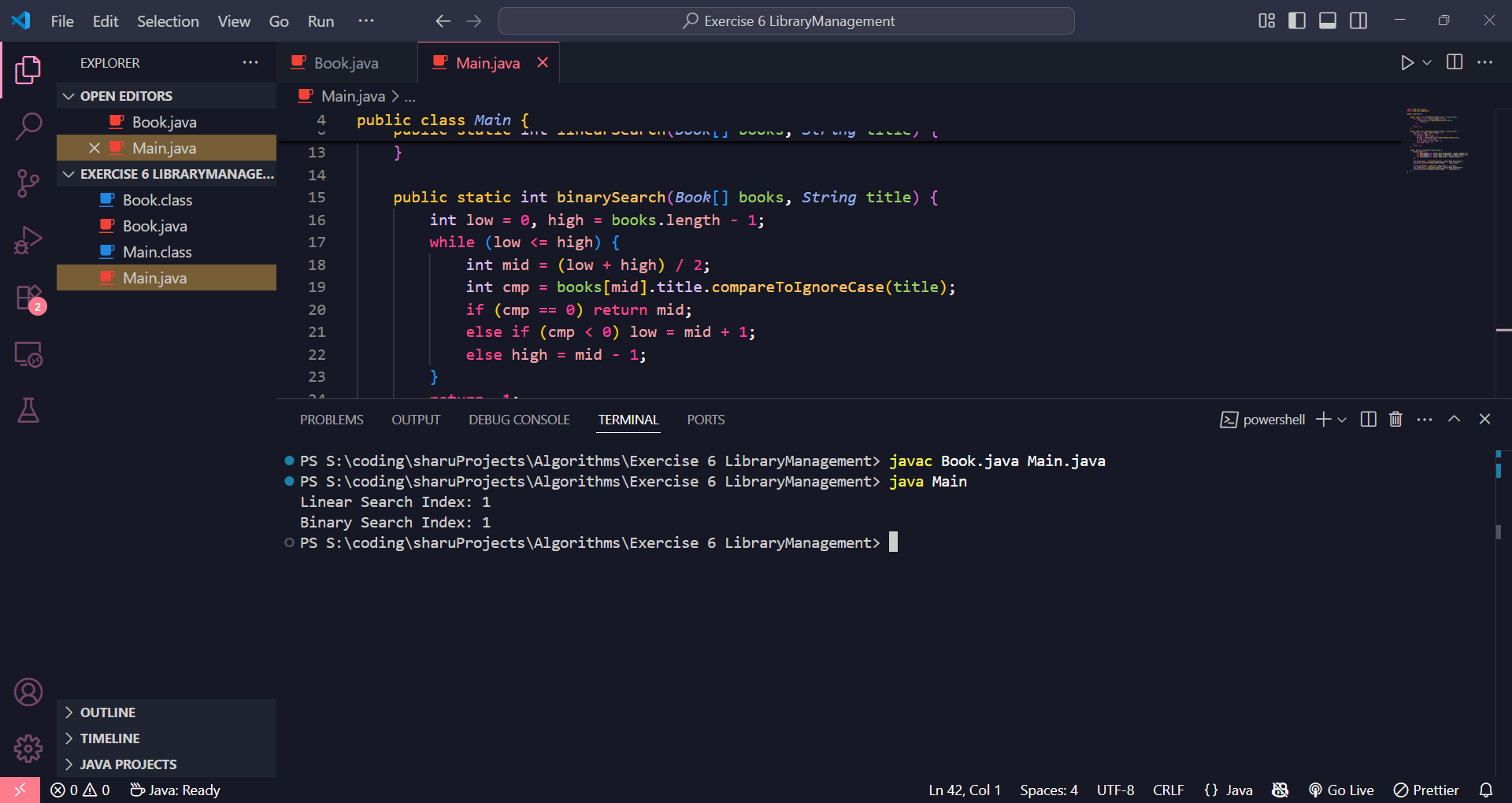
}

**To Compile**

javac Book.java Main.java

**To Run**

java Main

****

**Exercise 7: Financial Forecasting**

**Problem Overview**

This exercise involves implementing a recursive method to predict future financial values based on past growth rates.  
Recursion simplifies the calculation by breaking the problem into smaller subproblems.

**Main.java**

java

public class Main {

public static double futureValue(double presentValue, double growthRate, int years) {

if (years == 0) {

return presentValue;

}

return futureValue(presentValue, growthRate, years - 1) \* (1 + growthRate);

}

public static void main(String[] args) {

double presentValue = 1000;

double growthRate = 0.05;

int years = 5;

double predictedValue = futureValue(presentValue, growthRate, years);

System.out.println("Future Value after " + years + " years: " + predictedValue);

}

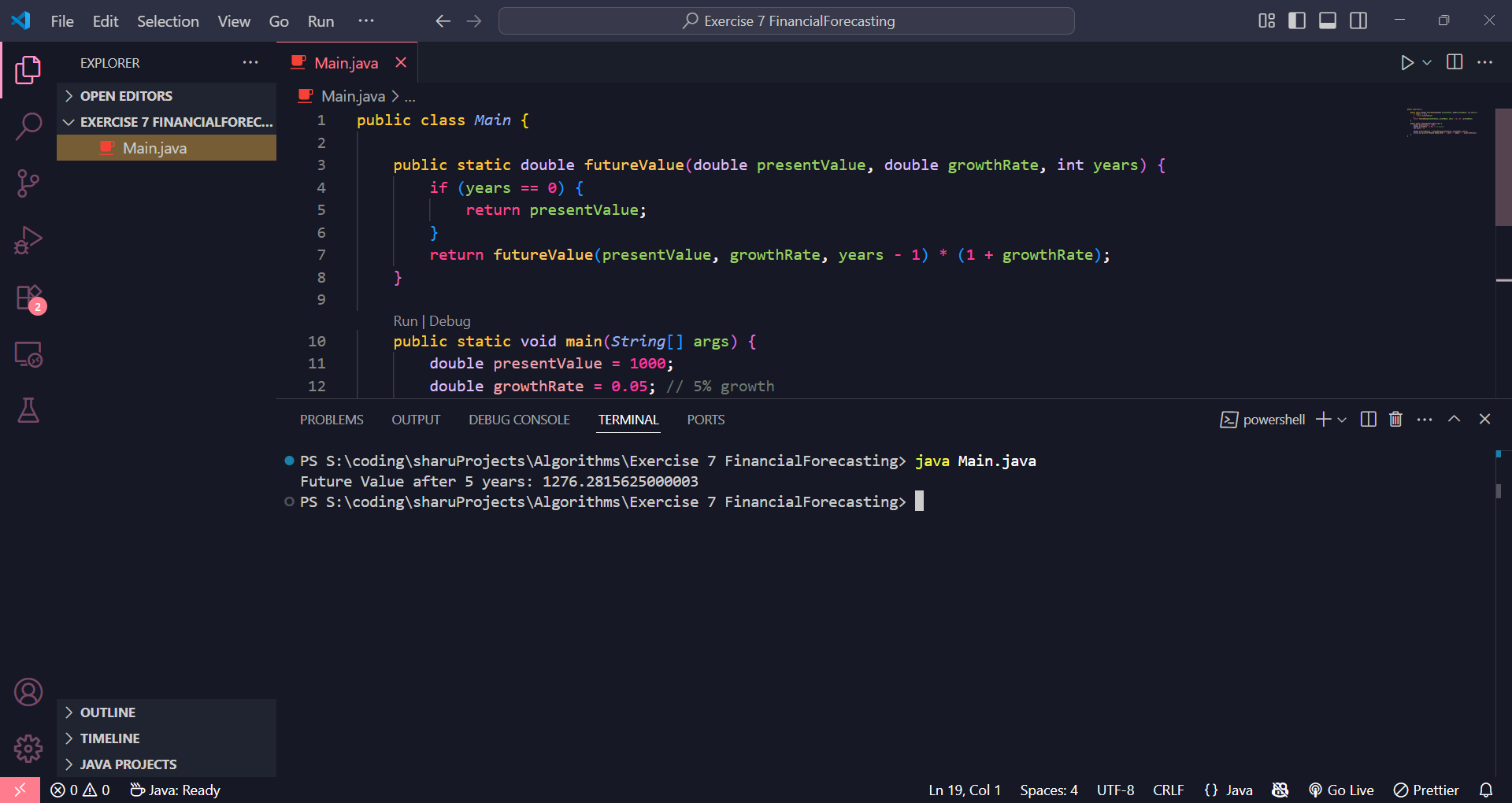
}

**To Compile**

javac Main.java

**To Run**

java Main

****