//BuilderPattern

package BuilderPattern;

class Computer {

    // Required parameters

    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;

    }

    // ✅ Static Builder class

    public static class Builder {

        private String CPU;

        private String RAM;

        private String Storage;

        public Builder(String CPU, String RAM, String Storage) {

            this.CPU = CPU;

            this.RAM = RAM;

            this.Storage = Storage;

        }

        // ✅ build() method

        public Computer build() {

            return new Computer(this);

        }

    }

    @Override

    public String toString() {

        return "Computer [CPU=" + CPU + ", RAM=" + RAM + ", Storage=" + Storage +

                "]";

    }

}

public class Main {

    public static void main(String[] args) {

        // Configuration 1: Gaming PC

        Computer gamingPC = new Computer.Builder("Intel i9", "32GB", "1TB SSD")

                .build();

        // Configuration 2: Office PC

        Computer officePC = new Computer.Builder("Intel i5", "8GB", "512GB SSD")

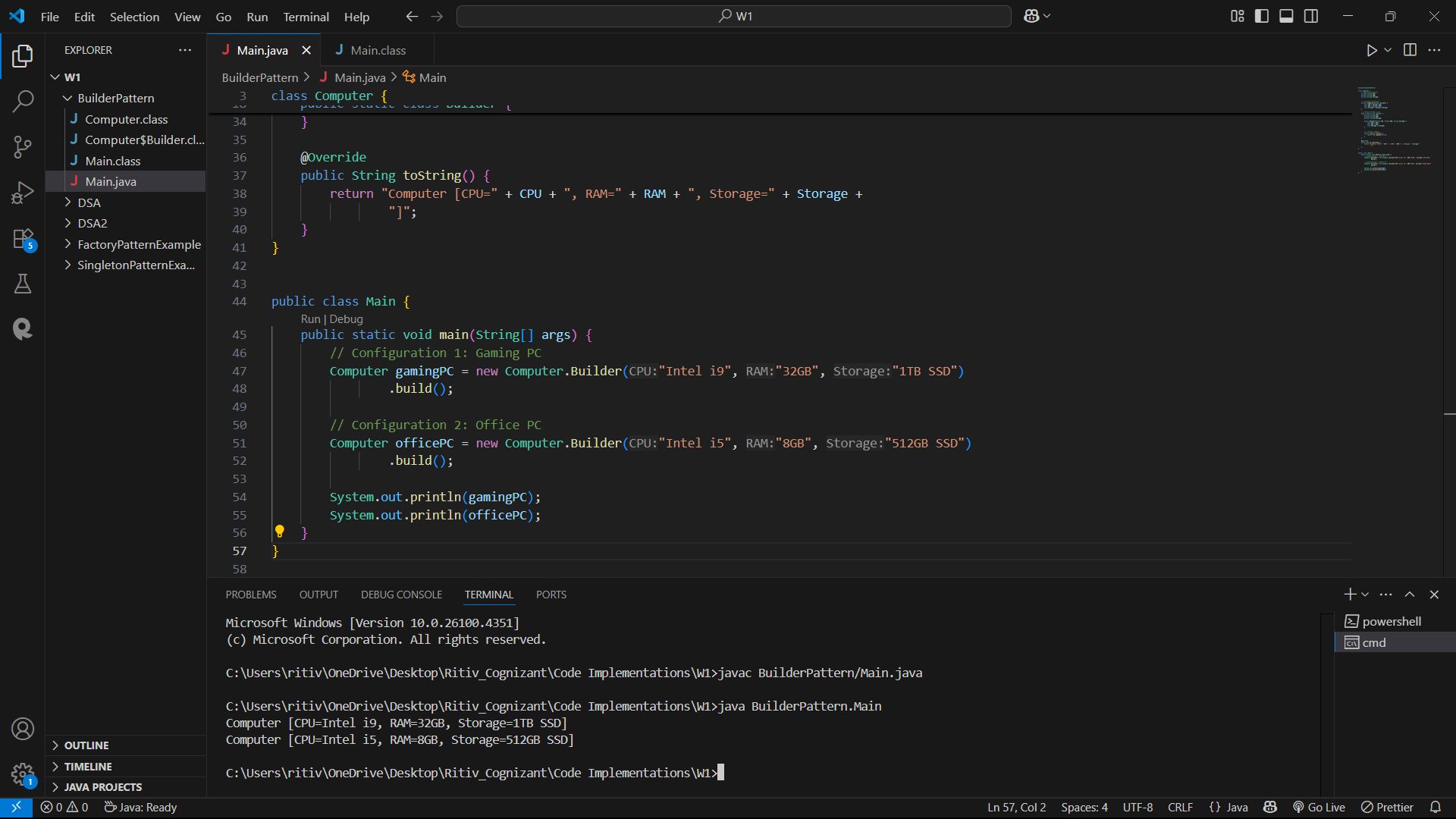
                .build();

        System.out.println(gamingPC);

        System.out.println(officePC);

    }

}



//Adapter Pattern

package AdapterPattern;

// PaymentProcessor.java

interface PaymentProcessor {

    void processPayment(double amount);

}

// PayPal.java

class PayPal {

    public void sendMoney(double amount) {

        System.out.println("Processing payment of Rs." + amount + " through PayPal.");

    }

}

// Stripe.java

class Stripe {

    public void makePayment(double amount) {

        System.out.println("Processing payment of Rs." + amount + " through Stripe.");

    }

}

// Razorpay.java

class Razorpay {

    public void transfer(double amount) {

        System.out.println("Processing payment of Rs." + amount + " through Razorpay.");

    }

}

// PayPalAdapter.java

class PayPalAdapter implements PaymentProcessor {

    private PayPal paypal;

    public PayPalAdapter(PayPal paypal) {

        this.paypal = paypal;

    }

    public void processPayment(double amount) {

        paypal.sendMoney(amount);

    }

}

// StripeAdapter.java

class StripeAdapter implements PaymentProcessor {

    private Stripe stripe;

    public StripeAdapter(Stripe stripe) {

        this.stripe = stripe;

    }

    public void processPayment(double amount) {

        stripe.makePayment(amount);

    }

}

// RazorpayAdapter.java

class RazorpayAdapter implements PaymentProcessor {

    private Razorpay razorpay;

    public RazorpayAdapter(Razorpay razorpay) {

        this.razorpay = razorpay;

    }

    public void processPayment(double amount) {

        razorpay.transfer(amount);

    }

}

// Main.java

public class Main {

    public static void main(String[] args) {

        PaymentProcessor paypal = new PayPalAdapter(new PayPal());

        PaymentProcessor stripe = new StripeAdapter(new Stripe());

        PaymentProcessor razorpay = new RazorpayAdapter(new Razorpay());

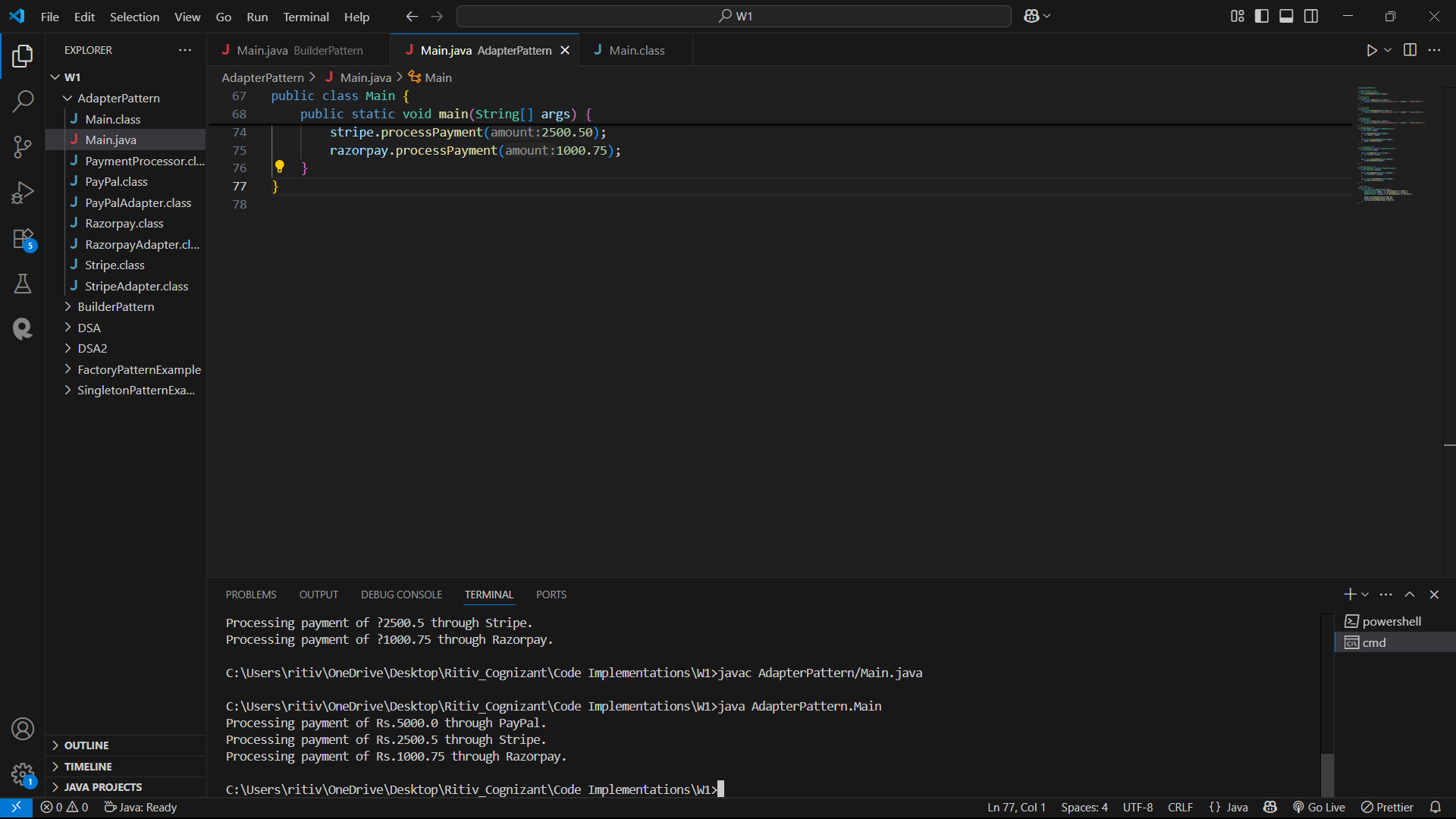
        paypal.processPayment(5000.00);

        stripe.processPayment(2500.50);

        razorpay.processPayment(1000.75);

    }

}



//Decorator Pattern

package DecoratorPattern;

interface Notifier {

    void send(String message);

}

class EmailNotifier implements Notifier {

    public void send(String message) {

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

    }

}

abstract class NotifierDecorator implements Notifier {

    protected Notifier wrappedNotifier;

    public NotifierDecorator(Notifier notifier) {

        this.wrappedNotifier = notifier;

    }

    public void send(String message) {

        wrappedNotifier.send(message);

    }

}

class SMSNotifierDecorator extends NotifierDecorator {

    public SMSNotifierDecorator(Notifier notifier) {

        super(notifier);

    }

    public void send(String message) {

        super.send(message);

        sendSMS(message);

    }

    private void sendSMS(String message) {

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

    }

}

class SlackNotifierDecorator extends NotifierDecorator {

    public SlackNotifierDecorator(Notifier notifier) {

        super(notifier);

    }

    public void send(String message) {

        super.send(message);

        sendSlack(message);

    }

    private void sendSlack(String message) {

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

    }

}

public class Main {

    public static void main(String[] args) {

        // Basic notifier

        Notifier baseNotifier = new EmailNotifier();

        // Add SMS notification

        Notifier smsNotifier = new SMSNotifierDecorator(baseNotifier);

        // Add Slack on top of SMS and Email

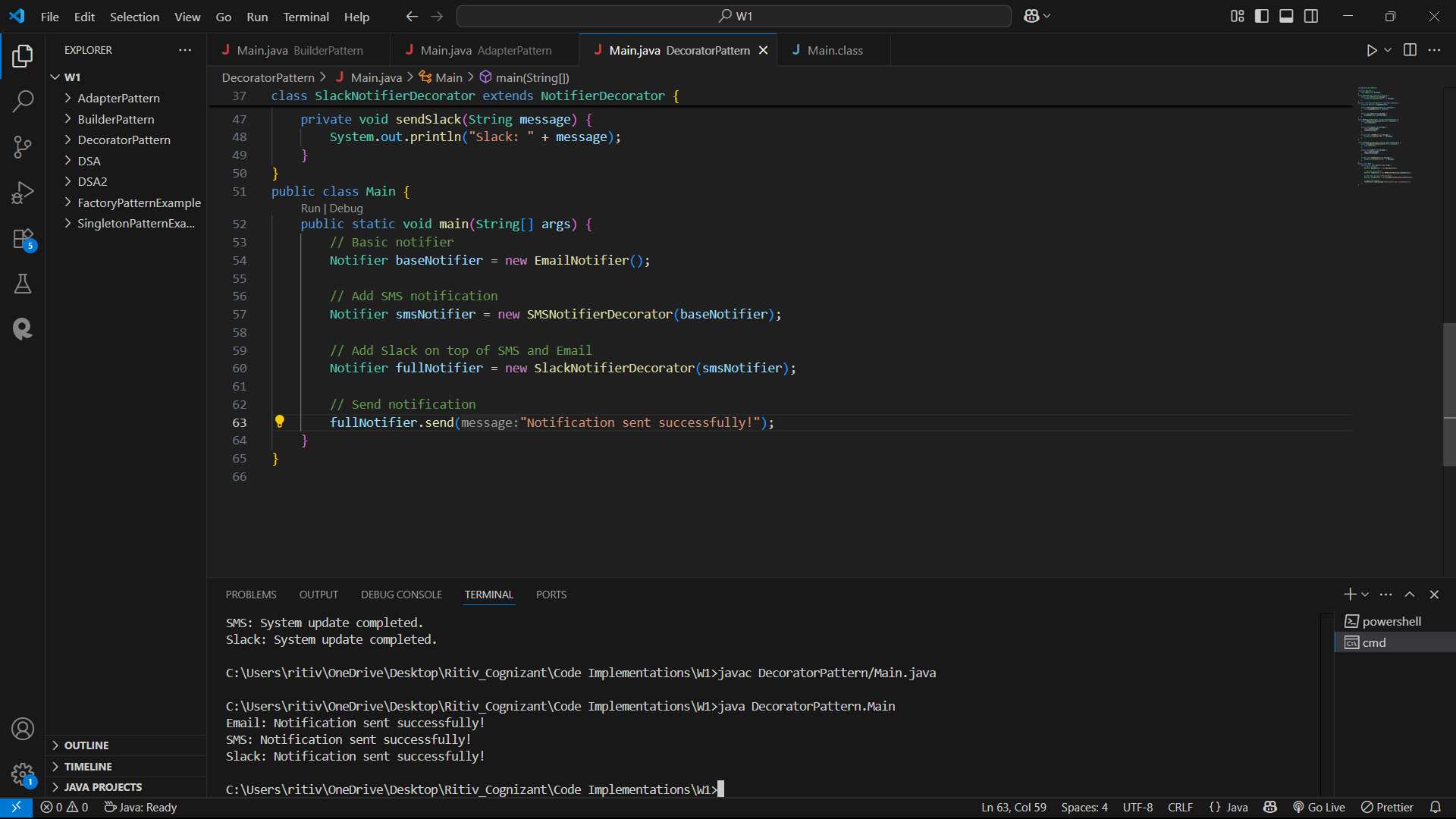
        Notifier fullNotifier = new SlackNotifierDecorator(smsNotifier);

        // Send notification

        fullNotifier.send("Notification sent successfully!");

    }

}



//Proxy Pattern

package ProxyPattern;

interface Image {

    void display();

}

class RealImage implements Image {

    private String filename;

    public RealImage(String filename) {

        this.filename = filename;

        loadFromRemoteServer();

    }

    private void loadFromRemoteServer() {

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

    }

    public void display() {

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

    }

}

class ProxyImage implements Image {

    private RealImage realImage;

    private String filename;

    public ProxyImage(String filename) {

        this.filename = filename;

    }

    public void display() {

        if (realImage == null) {

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

        } else {

            System.out.println("Using cached image: " + filename);

        }

        realImage.display();

    }

}

public class Main {

    public static void main(String[] args) {

        Image img1 = new ProxyImage("Parrot.png");

        Image img2 = new ProxyImage("Squirrel.png");

        // First display - triggers load

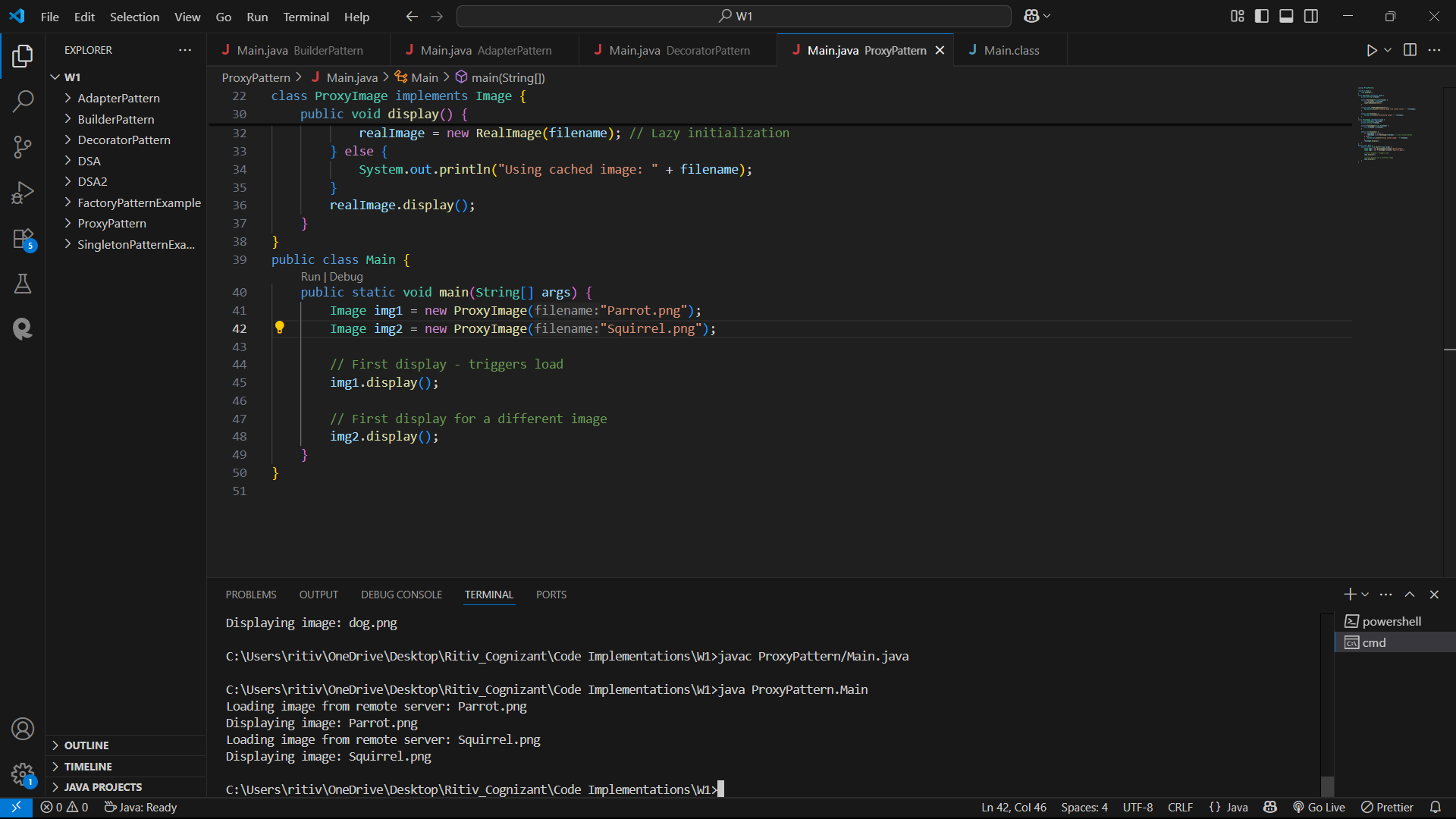
        img1.display();

        // First display for a different image

        img2.display();

    }

}



//Observer Pattern

package ObserverPattern;

import java.util.ArrayList;

import java.util.List;

interface Stock {

    void register(Observer o);

    void deregister(Observer o);

    void notifyObservers();

}

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();

    }

    public void register(Observer o) {

        observers.add(o);

    }

    public void deregister(Observer o) {

        observers.remove(o);

    }

    public void notifyObservers() {

        for (Observer o : observers) {

            o.update(stockName, stockPrice);

        }

    }

}

interface Observer {

    void update(String stockName, double stockPrice);

}

class MobileApp implements Observer {

    private String name;

    public MobileApp(String name) {

        this.name = name;

    }

    public void update(String stockName, double stockPrice) {

        System.out.println("MobileApp [" + name + "] - Stock: " + stockName + " Price: Rs." + stockPrice);

    }

}

class WebApp implements Observer {

    private String name;

    public WebApp(String name) {

        this.name = name;

    }

    public void update(String stockName, double stockPrice) {

        System.out.println("WebApp [" + name + "] - Stock: " + stockName + " Price: Rs." + stockPrice);

    }

}

public class Main {

    public static void main(String[] args) {

        StockMarket market = new StockMarket();

        Observer mobileUser = new MobileApp("Ritiv");

        Observer webUser = new WebApp("Best-in-vest");

        market.register(mobileUser);

        market.register(webUser);

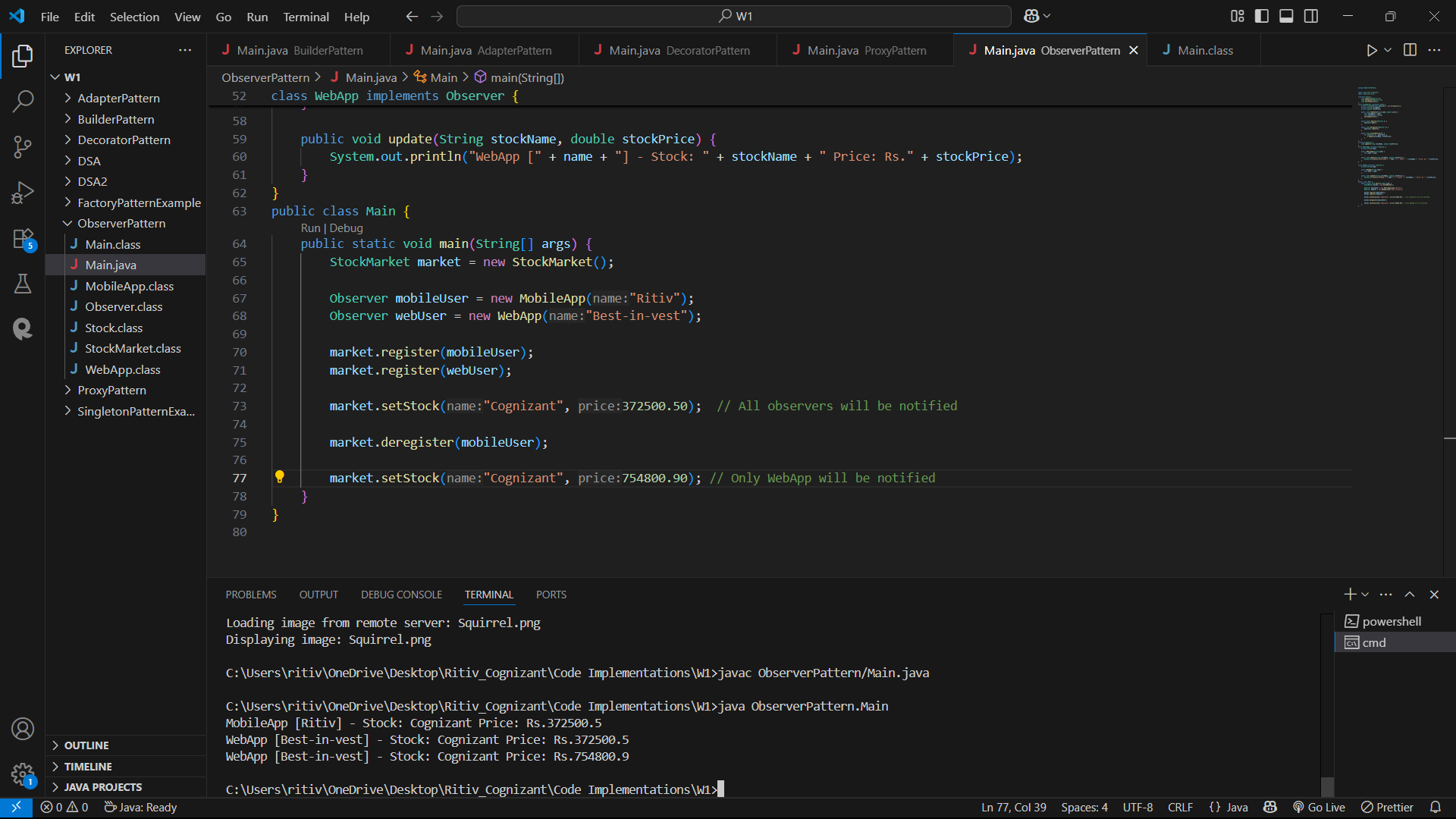
        market.setStock("Cognizant", 372500.50);  // All observers will be notified

        market.deregister(mobileUser);

        market.setStock("Cognizant", 754800.90); // Only WebApp will be notified

    }

}



//Strategy Pattern

package StrategyPattern;

interface PaymentStrategy {

    void pay(double amount);

}

class CreditCardPayment implements PaymentStrategy {

    private String cardNumber;

    public CreditCardPayment(String cardNumber) {

        this.cardNumber = cardNumber;

    }

    public void pay(double amount) {

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

    }

}

class PayPalPayment implements PaymentStrategy {

    private String email;

    public PayPalPayment(String email) {

        this.email = email;

    }

    public void pay(double amount) {

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

    }

}

class PaymentContext {

    private PaymentStrategy strategy;

    public void setStrategy(PaymentStrategy strategy) {

        this.strategy = strategy;

    }

    public void payAmount(double amount) {

        if (strategy != null) {

            strategy.pay(amount);

        } else {

            System.out.println("No payment strategy selected.");

        }

    }

}

public class Main {

    public static void main(String[] args) {

        PaymentContext context = new PaymentContext();

        context.setStrategy(new CreditCardPayment("5669-5678-8769-3456"));

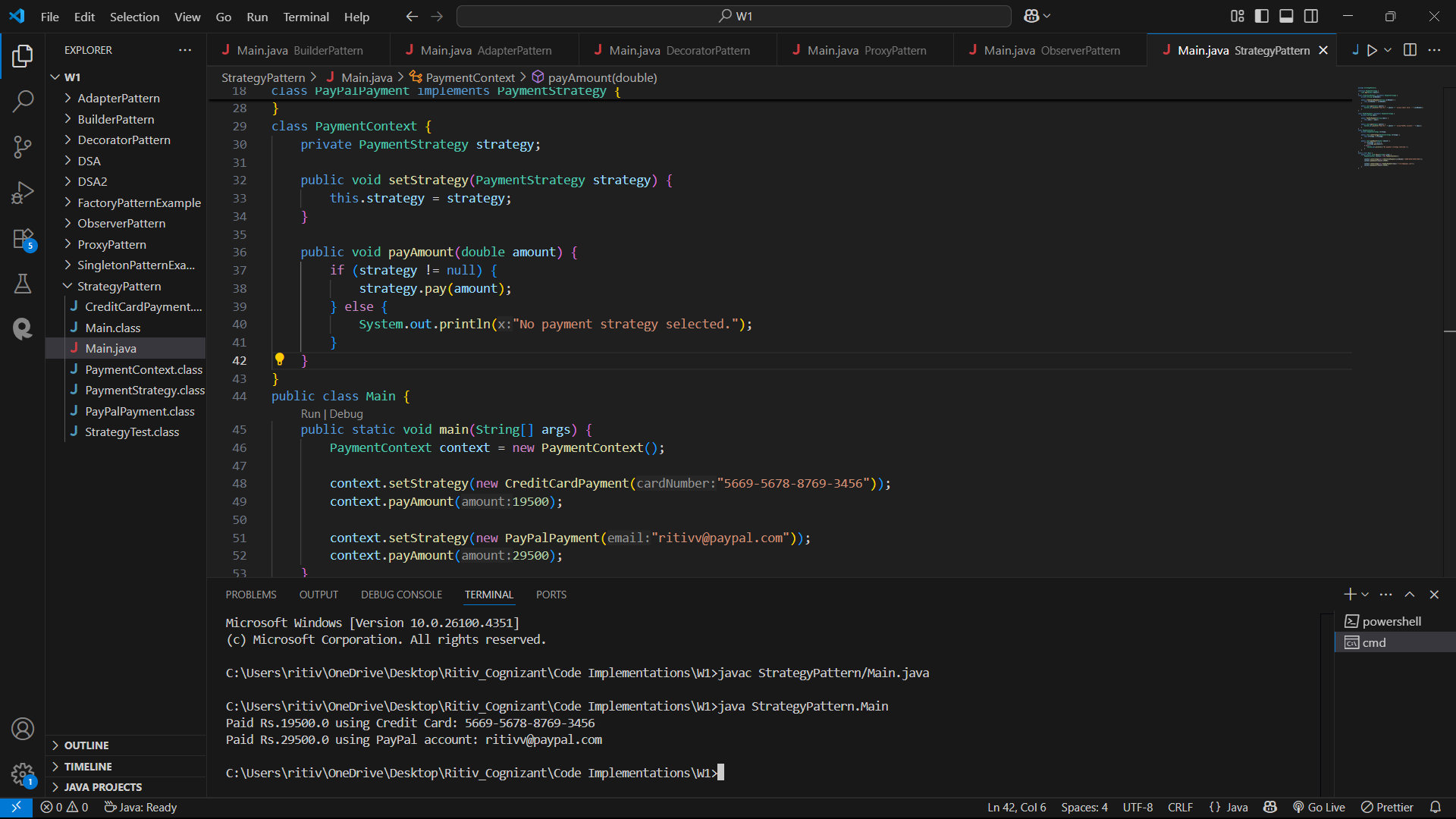
        context.payAmount(19500);

        context.setStrategy(new PayPalPayment("ritivv@paypal.com"));

        context.payAmount(29500);

    }

}



//Command Pattern

package CommandPattern;

interface Command {

    void execute();

}

class Light {

    public void turnOn() {

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

    }

    public void turnOff() {

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

    }

}

class LightOnCommand implements Command {

    private Light light;

    public LightOnCommand(Light light) {

        this.light = light;

    }

    public void execute() {

        light.turnOn();

    }

}

class LightOffCommand implements Command {

    private Light light;

    public LightOffCommand(Light light) {

        this.light = light;

    }

    public void execute() {

        light.turnOff();

    }

}

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.");

        }

    }

}public class Main {

    public static void main(String[] args) {

        Light light = new Light();

        Command lightOn = new LightOnCommand(light);

        Command lightOff = new LightOffCommand(light);

        RemoteControl remote = new RemoteControl();

        remote.setCommand(lightOn);

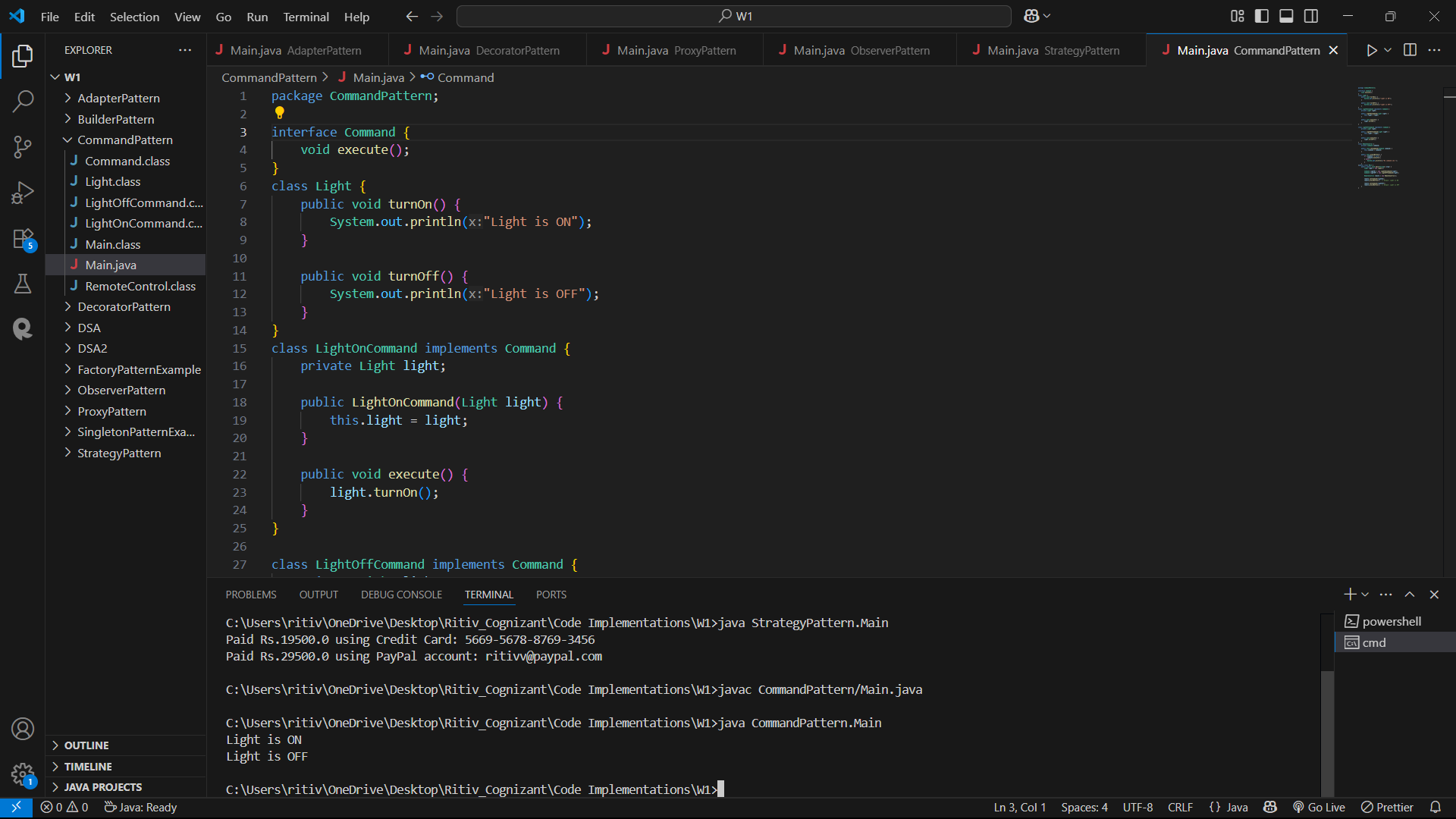
        remote.pressButton();  // Output: Light is ON

        remote.setCommand(lightOff);

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

    }

}



//MVC Pattern

package MVCPattern;

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 and Setters

    public String getName() { return name; }

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

    public String getId() { return id; }

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

    public String getGrade() { return grade; }

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

}

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);

    }

}

class StudentController {

    private Student model;

    private StudentView view;

    public StudentController(Student model, StudentView view) {

        this.model = model;

        this.view = view;

    }

    // Controller methods to update model

    public void setStudentName(String name) {

        model.setName(name);

    }

    public void setStudentId(String id) {

        model.setId(id);

    }

    public void setStudentGrade(String grade) {

        model.setGrade(grade);

    }

    // Controller methods to get model data

    public String getStudentName() { return model.getName(); }

    public String getStudentId() { return model.getId(); }

    public String getStudentGrade() { return model.getGrade(); }

    // Update view

    public void updateView() {

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

    }

}

public class Main {

    public static void main(String[] args) {

        // Step 1: Create model

        Student student = new Student("Ritiv", "X169", "A+");

        // Step 2: Create view

        StudentView view = new StudentView();

        // Step 3: Create controller

        StudentController controller = new StudentController(student, view);

        // Step 4: Display initial details

        controller.updateView();

        // Step 5: Update model via controller

        controller.setStudentName("Ritiv Mohapatra");

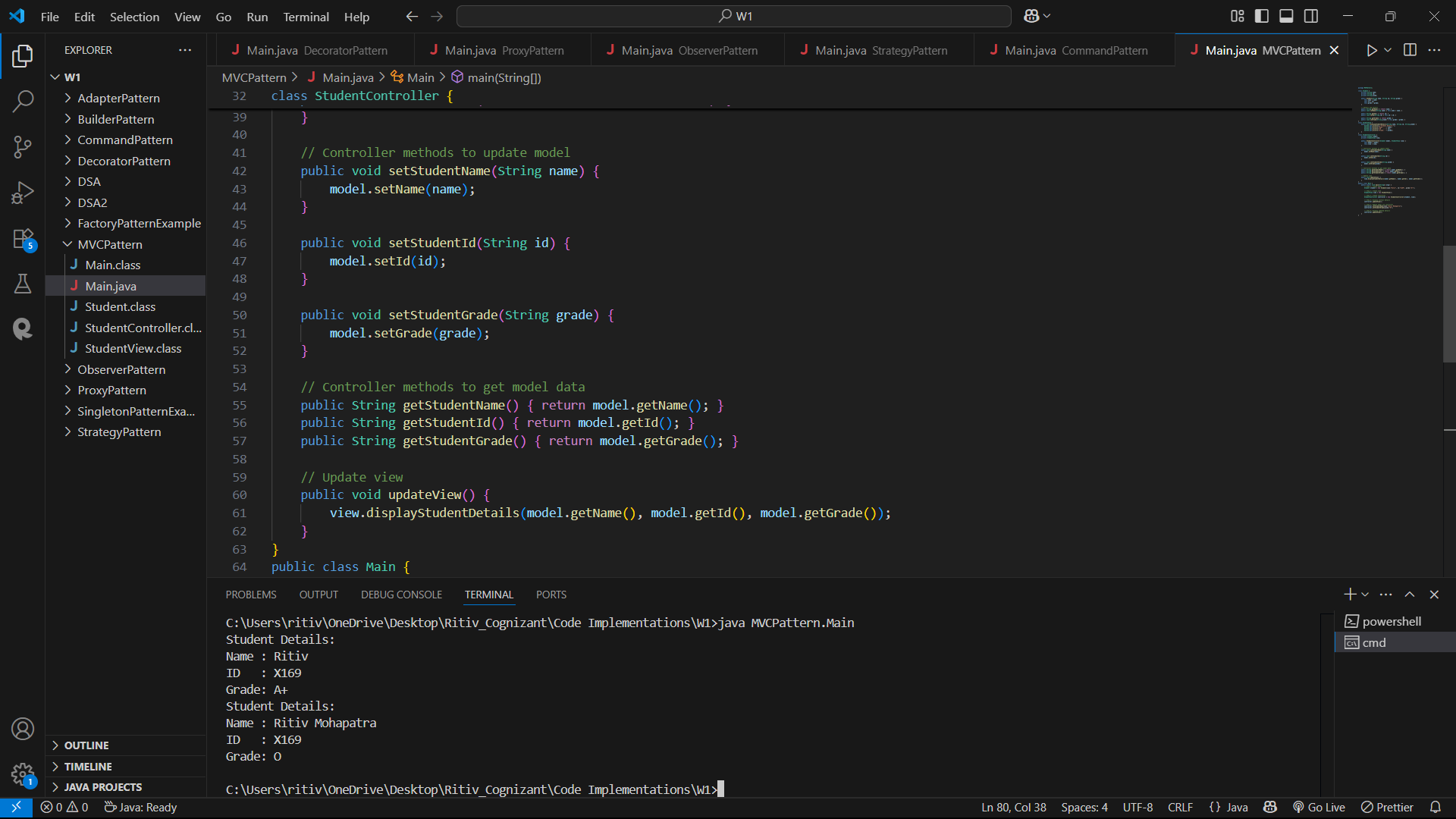
        controller.setStudentGrade("O");

        // Step 6: Display updated details

        controller.updateView();

    }

}



//Dependency Injection

package DependencyInjection;

import java.util.HashMap;

import java.util.Map;

interface CustomerRepository {

    Customer findCustomerById(String id);

}

class Customer {

    private String id;

    private String name;

    public Customer(String id, String name) {

        this.id = id;

        this.name = name;

    }

    // Getters

    public String getId() { return id; }

    public String getName() { return name; }

}

class CustomerRepositoryImpl implements CustomerRepository {

    private Map<String, Customer> database = new HashMap<>();

    public CustomerRepositoryImpl() {

        // Sample data

        database.put("C001", new Customer("C001", "Ritiv"));

        database.put("C002", new Customer("C002", "Chaitanya"));

    }

    public Customer findCustomerById(String id) {

        return database.get(id);

    }

}

class CustomerService {

    private CustomerRepository customerRepository;

    // Constructor injection

    public CustomerService(CustomerRepository customerRepository) {

        this.customerRepository = customerRepository;

    }

    public void displayCustomer(String id) {

        Customer customer = customerRepository.findCustomerById(id);

        if (customer != null) {

            System.out.println("Customer Found:");

            System.out.println("ID   : " + customer.getId());

            System.out.println("Name : " + customer.getName());

        } else {

            System.out.println("Customer not found for ID: " + id);

        }

    }

}

public class Main {

    public static void main(String[] args) {

        // Create the repository

        CustomerRepository repository = new CustomerRepositoryImpl();

        // Inject repository into service

        CustomerService service = new CustomerService(repository);

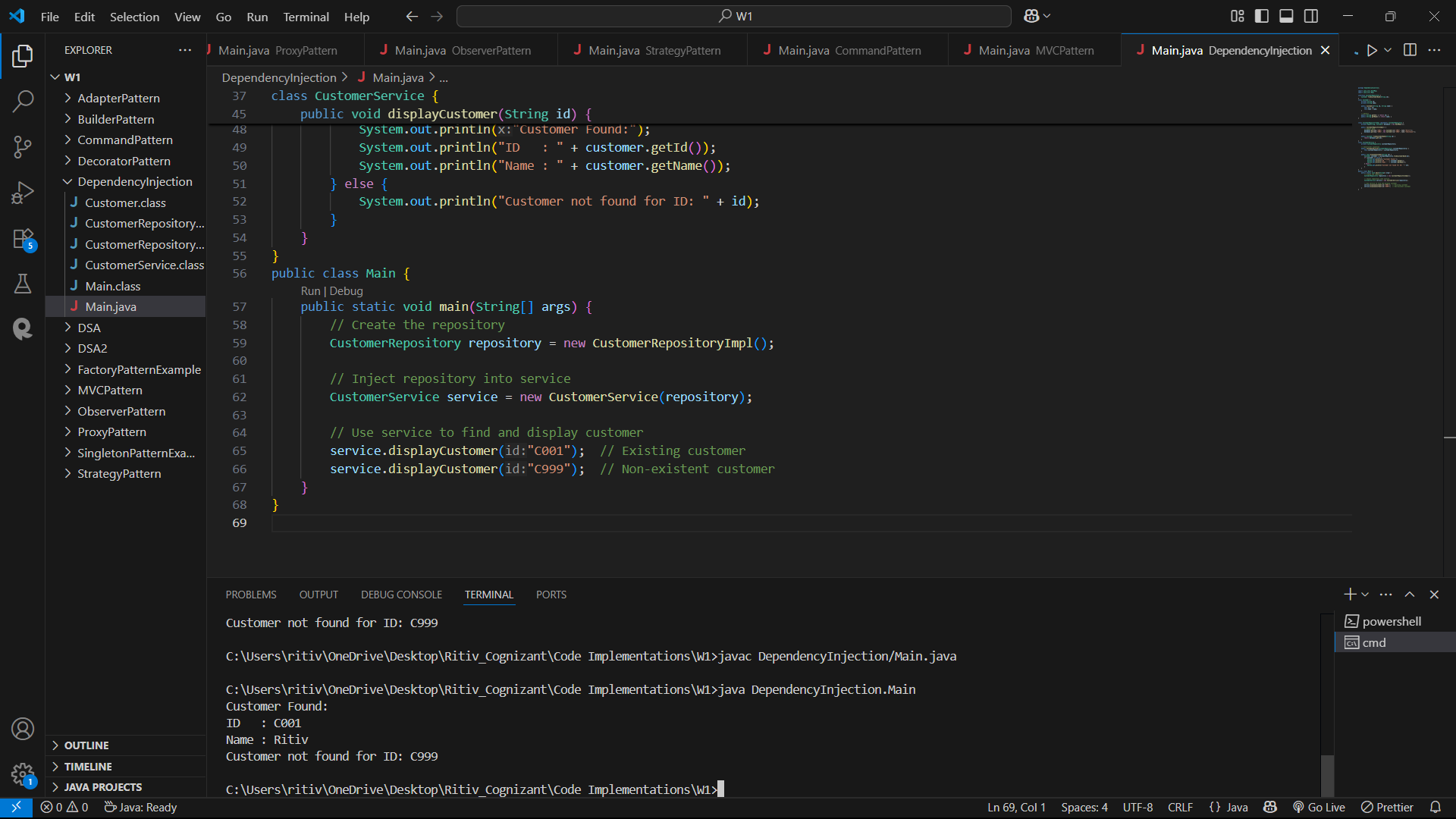
        // Use service to find and display customer

        service.displayCustomer("C001");  // Existing customer

        service.displayCustomer("C999");  // Non-existent customer

    }

}



//Data Structure and Algorithms//

//Inventory Management System

package InventoryManagement;

import java.util.HashMap;

import java.util.Map;

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;

    }

    @Override

    public String toString() {

        return "ProductID: " + productId + ", Name: " + productName +

               ", Quantity: " + quantity + ", Price: $" + price;

    }

}

class Inventory {

    private Map<Integer, Product> products;

    public Inventory() {

        products = new HashMap<>();

    }

    // Add a product

    public void addProduct(Product product) {

        if (products.containsKey(product.productId)) {

            System.out.println("Product ID already exists.");

        } else {

            products.put(product.productId, product);

            System.out.println("Product added.");

        }

    }

    // Update an existing product

    public void updateProduct(int productId, String name, int quantity, double price) {

        if (products.containsKey(productId)) {

            Product product = products.get(productId);

            product.productName = name;

            product.quantity = quantity;

            product.price = price;

            System.out.println("Product updated.");

        } else {

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

        }

    }

    // Delete a product

    public void deleteProduct(int productId) {

        if (products.remove(productId) != null) {

            System.out.println("Product removed.");

        } else {

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

        }

    }

    // Display all products

    public void displayProducts() {

        if (products.isEmpty()) {

            System.out.println("Inventory is empty.");

        } else {

            for (Product product : products.values()) {

                System.out.println(product);

            }

        }

    }

}

public class Main {

    public static void main(String[] args) {

        Inventory inventory = new Inventory();

        // Sample operations

        inventory.addProduct(new Product(1, "Laptop", 10, 992.99));

        inventory.addProduct(new Product(2, "Mouse", 50, 17.99));

        inventory.displayProducts();

        inventory.updateProduct(1, "Gaming Laptop", 8, 1299.99);

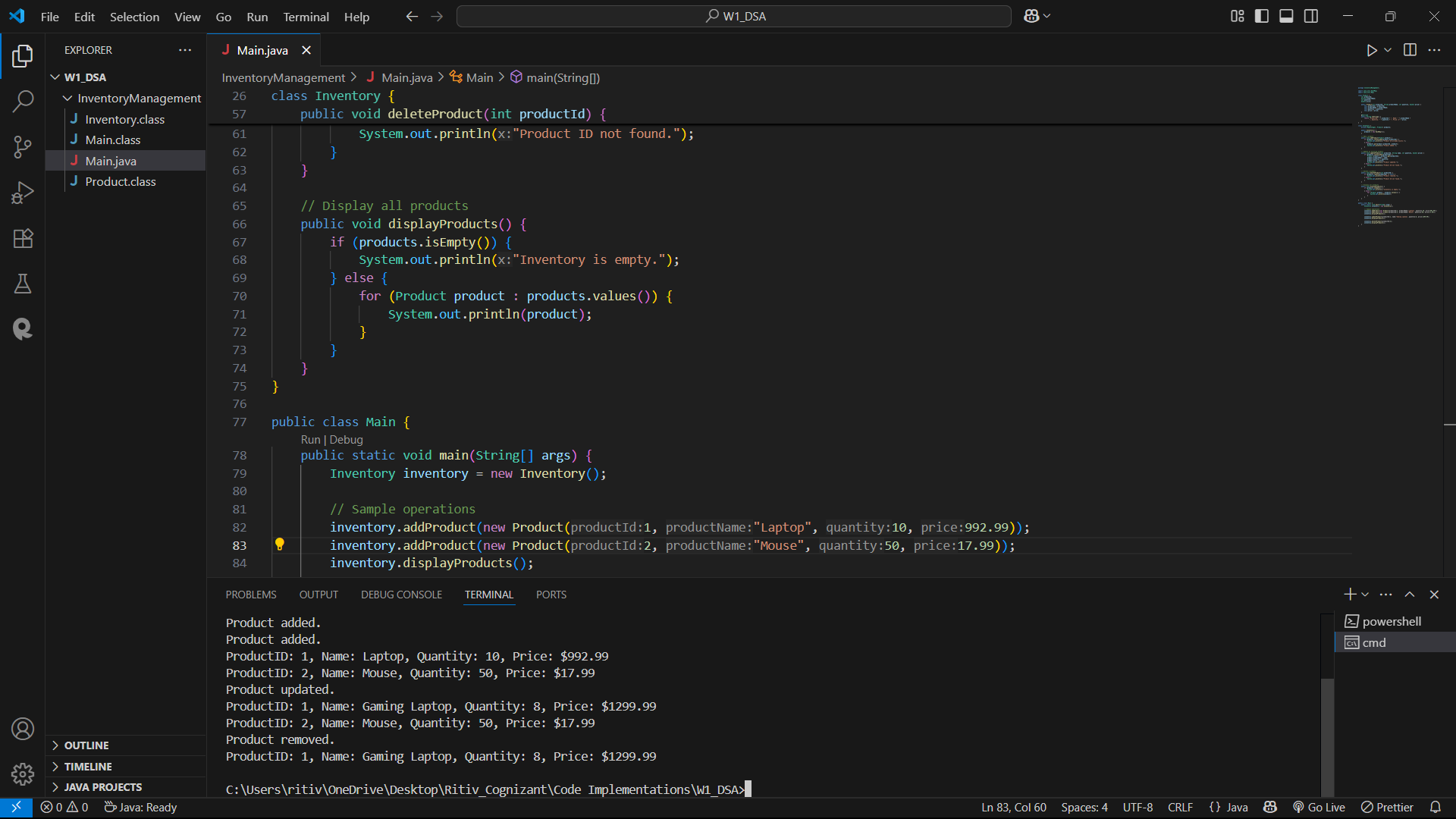
        inventory.displayProducts();

        inventory.deleteProduct(2);

        inventory.displayProducts();

    }

}



//Sorting Customer Orders

package SortingCustomerOrders;

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;

    }

    @Override

    public String toString() {

        return "OrderID: " + orderId + ", Customer: " + customerName + ", Total Price: $" + totalPrice;

    }

}

public class OrderManager {

    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) {

                    // Swap

                    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);

        }

    }

    private 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++;

                // Swap

                Order temp = orders[i];

                orders[i] = orders[j];

                orders[j] = temp;

            }

        }

        // Swap pivot

        Order temp = orders[i + 1];

        orders[i + 1] = orders[high];

        orders[high] = temp;

        return i + 1;

    }

    public static void main(String[] args) {

        Order[] orders = {

            new Order(101, "Alice", 450.0),

            new Order(102, "Bob", 120.5),

            new Order(103, "Charlie", 230.0),

            new Order(104, "David", 320.75)

        };

        System.out.println("Original Orders:");

        printOrders(orders);

        // Bubble Sort

        bubbleSort(orders);

        System.out.println("\nAfter Bubble Sort:");

        printOrders(orders);

        // Re-initialize for Quick Sort

        orders = new Order[] {

            new Order(101, "Alice", 450.0),

            new Order(102, "Bob", 120.5),

            new Order(103, "Charlie", 230.0),

            new Order(104, "David", 320.75)

        };

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

        System.out.println("\nAfter Quick Sort:");

        printOrders(orders);

    }

    public static void printOrders(Order[] orders) {

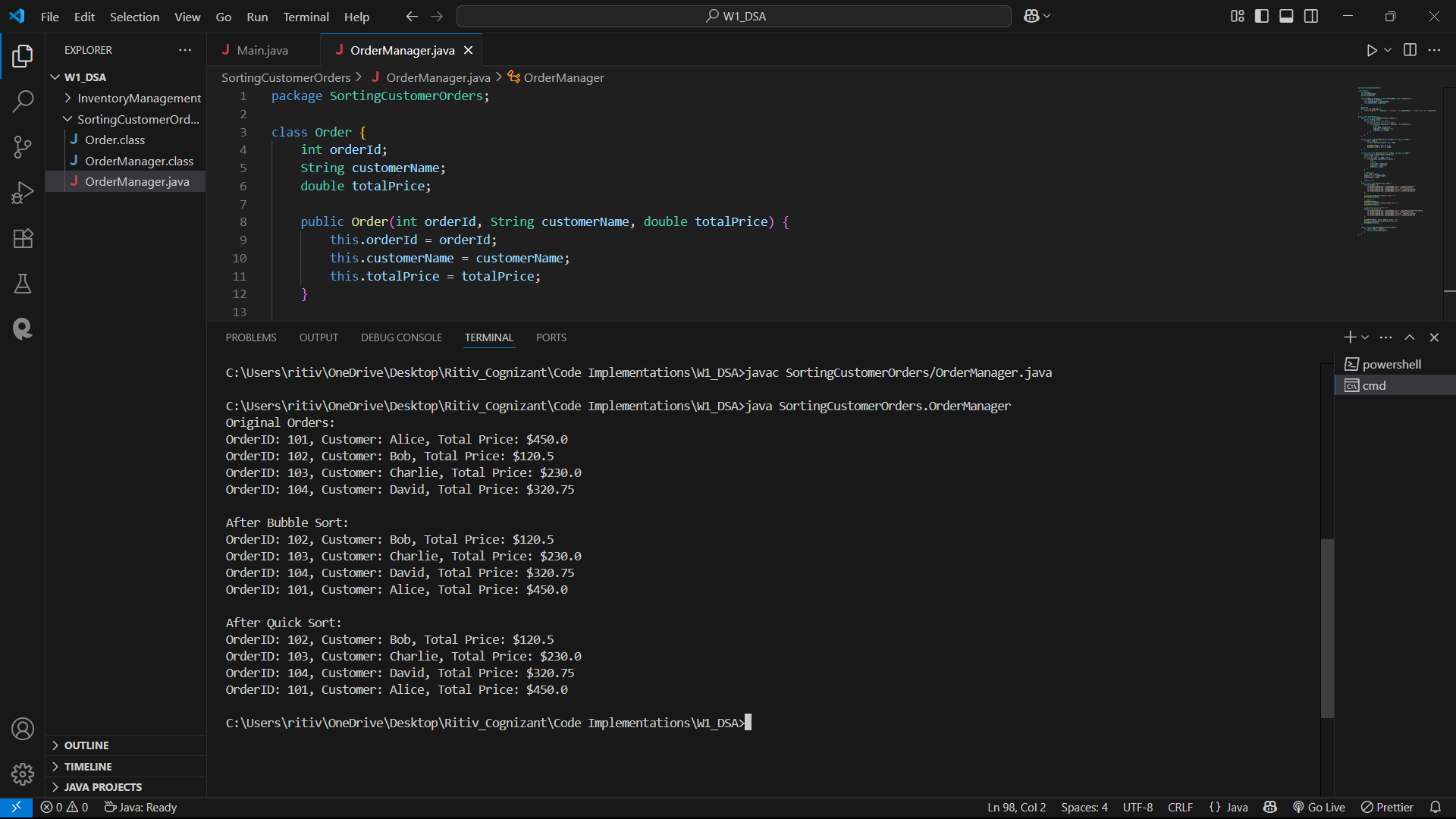
        for (Order order : orders) {

            System.out.println(order);

        }

    }

}



//Employee Management System

package EmployeeManagement;

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;

    }

    @Override

    public String toString() {

        return "ID: " + employeeId + ", Name: " + name + ", Position: " + position + ", Salary: $" + salary;

    }

}

class EmployeeManager {

    private Employee[] employees;

    private int count;

    public EmployeeManager(int size) {

        employees = new Employee[size];

        count = 0;

    }

    // Add employee

    public void addEmployee(Employee e) {

        if (count < employees.length) {

            employees[count++] = e;

            System.out.println("Employee added.");

        } else {

            System.out.println("Employee array is full.");

        }

    }

    // Search employee by ID

    public Employee searchEmployee(int employeeId) {

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

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

                return employees[i];

            }

        }

        return null;

    }

    // Traverse (print all employees)

    public void traverseEmployees() {

        if (count == 0) {

            System.out.println("No employees in the system.");

            return;

        }

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

            System.out.println(employees[i]);

        }

    }

    // Delete employee by ID

    public void deleteEmployee(int employeeId) {

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

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

                // Shift remaining employees left

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

                    employees[j] = employees[j + 1];

                }

                employees[--count] = null; // Clear last spot

                System.out.println("Employee deleted.");

                return;

            }

        }

        System.out.println("Employee not found.");

    }

}

public class Main {

    public static void main(String[] args) {

        EmployeeManager manager = new EmployeeManager(5);

        manager.addEmployee(new Employee(1, "Alice", "Manager", 70000));

        manager.addEmployee(new Employee(2, "Bob", "Developer", 50000));

        manager.addEmployee(new Employee(3, "Charlie", "Designer", 45000));

        System.out.println("\nAll Employees:");

        manager.traverseEmployees();

        System.out.println("\nSearching for employee with ID 2:");

        Employee found = manager.searchEmployee(2);

        System.out.println(found != null ? found : "Employee not found.");

        System.out.println("\nDeleting employee with ID 2...");

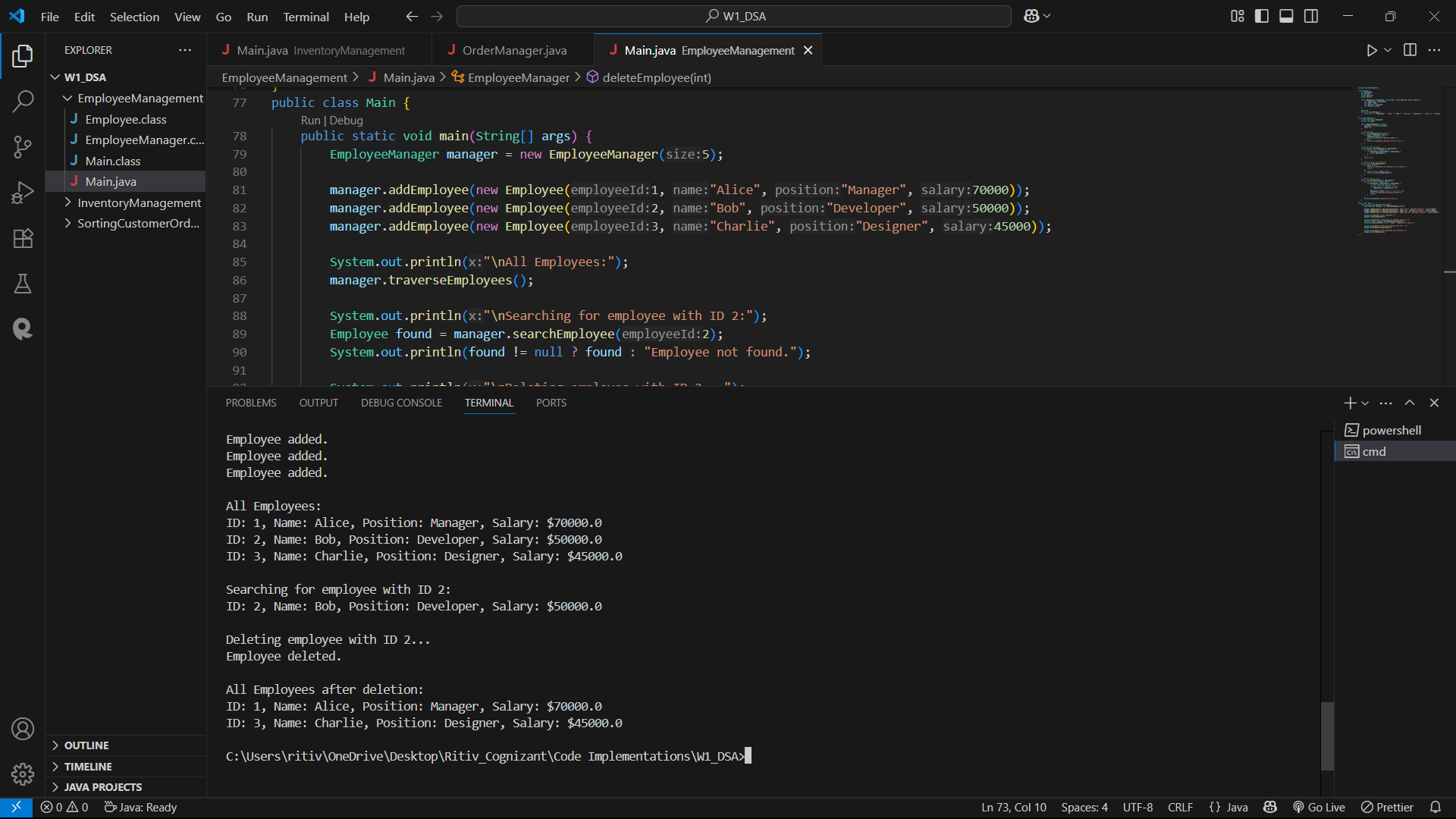
        manager.deleteEmployee(2);

        System.out.println("\nAll Employees after deletion:");

        manager.traverseEmployees();

    }

}



//Task Management System

package TaskManagement;

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;

    }

    @Override

    public String toString() {

        return "TaskID: " + taskId + ", Name: " + taskName + ", Status: " + status;

    }

}

class Node {

    Task task;

    Node next;

    public Node(Task task) {

        this.task = task;

        this.next = null;

    }

}

class TaskLinkedList {

    private Node head;

    public TaskLinkedList() {

        head = null;

    }

    // Add task to end of list

    public void addTask(Task task) {

        Node newNode = new Node(task);

        if (head == null) {

            head = newNode;

        } else {

            Node curr = head;

            while (curr.next != null) {

                curr = curr.next;

            }

            curr.next = newNode;

        }

        System.out.println("Task added.");

    }

    // Search task by ID

    public Task searchTask(int taskId) {

        Node curr = head;

        while (curr != null) {

            if (curr.task.taskId == taskId) {

                return curr.task;

            }

            curr = curr.next;

        }

        return null;

    }

    // Traverse and display all tasks

    public void traverseTasks() {

        if (head == null) {

            System.out.println("No tasks available.");

            return;

        }

        Node curr = head;

        while (curr != null) {

            System.out.println(curr.task);

            curr = curr.next;

        }

    }

    // Delete task by ID

    public void deleteTask(int taskId) {

        if (head == null) {

            System.out.println("Task list is empty.");

            return;

        }

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

            head = head.next;

            System.out.println("Task deleted.");

            return;

        }

        Node curr = head;

        while (curr.next != null && curr.next.task.taskId != taskId) {

            curr = curr.next;

        }

        if (curr.next == null) {

            System.out.println("Task not found.");

        } else {

            curr.next = curr.next.next;

            System.out.println("Task deleted.");

        }

    }

}

public class Main {

    public static void main(String[] args) {

        TaskLinkedList taskList = new TaskLinkedList();

        taskList.addTask(new Task(1, "Design database", "Pending"));

        taskList.addTask(new Task(2, "Implement login", "Pending"));

        taskList.addTask(new Task(3, "Test APIs", "Completed"));

        System.out.println("\nAll Tasks:");

        taskList.traverseTasks();

        System.out.println("\nSearching for Task ID 2:");

        Task t = taskList.searchTask(2);

        System.out.println(t != null ? t : "Task not found.");

        System.out.println("\nDeleting Task ID 2...");

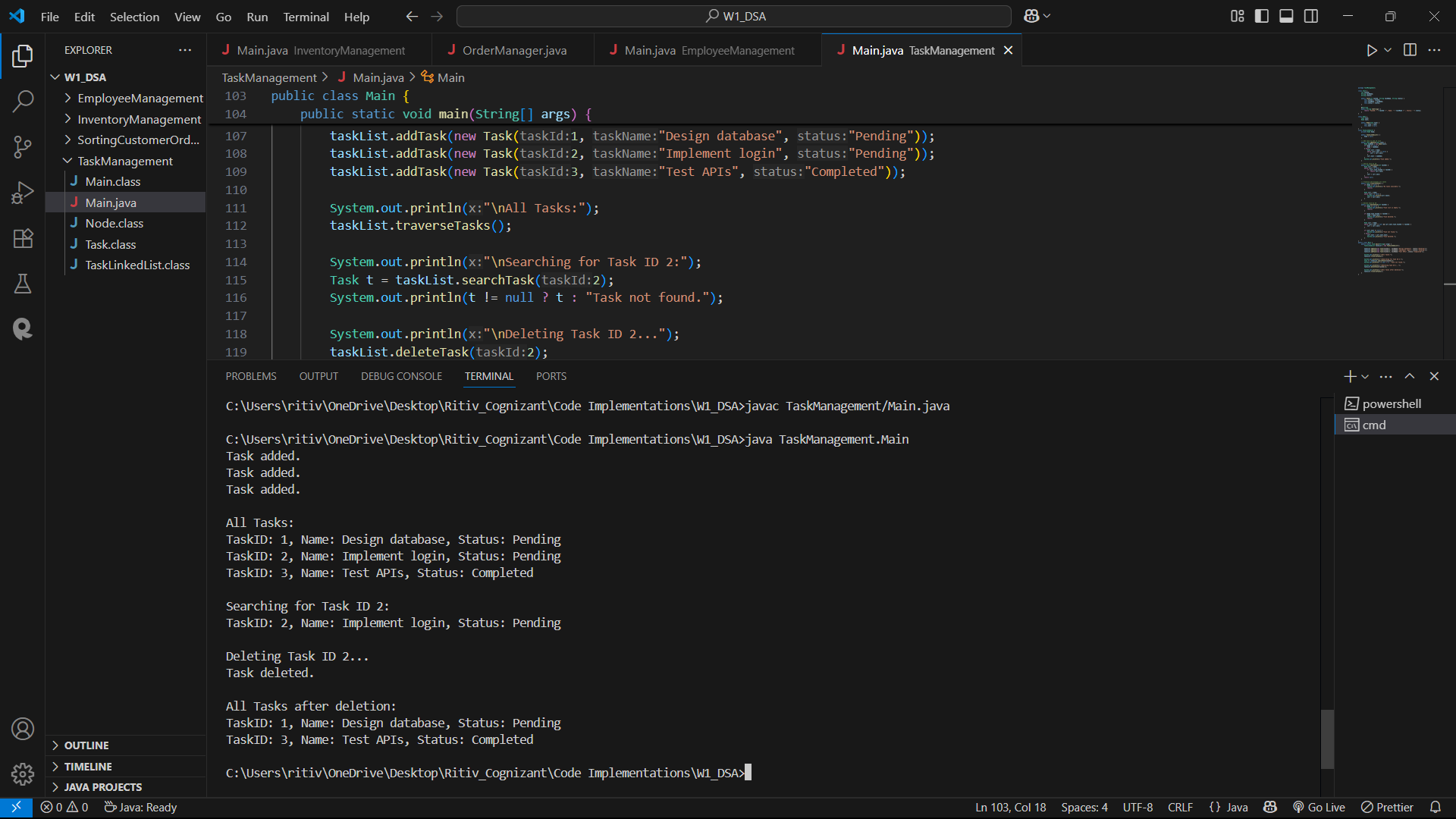
        taskList.deleteTask(2);

        System.out.println("\nAll Tasks after deletion:");

        taskList.traverseTasks();

    }

}



//Library Management System

package BookManagement;

import java.util.Arrays;

import java.util.Comparator;

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;

    }

    @Override

    public String toString() {

        return "BookID: " + bookId + ", Title: \"" + title + "\", Author: " + author;

    }

}

class BookManager {

    private Book[] books;

    private int count;

    public BookManager(int size) {

        books = new Book[size];

        count = 0;

    }

    // Add a book

    public void addBook(Book book) {

        if (count < books.length) {

            books[count++] = book;

        } else {

            System.out.println("Book list is full.");

        }

    }

    // Linear search by title

    public void linearSearch(String title) {

        boolean found = false;

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

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

                System.out.println("Found: " + books[i]);

                found = true;

            }

        }

        if (!found) {

            System.out.println("Book not found with title: " + title);

        }

    }

    // Sort books by title (for binary search)

    public void sortBooksByTitle() {

        Arrays.sort(books, 0, count, Comparator.comparing(book -> book.title.toLowerCase()));

    }

    // Binary search by title

    public void binarySearch(String title) {

        int low = 0;

        int high = count - 1;

        title = title.toLowerCase();

        while (low <= high) {

            int mid = (low + high) / 2;

            String midTitle = books[mid].title.toLowerCase();

            if (midTitle.equals(title)) {

                System.out.println("Found: " + books[mid]);

                return;

            } else if (midTitle.compareTo(title) < 0) {

                low = mid + 1;

            } else {

                high = mid - 1;

            }

        }

        System.out.println("Book not found with title: " + title);

    }

    // Display all books

    public void displayBooks() {

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

            System.out.println(books[i]);

        }

    }

}

public class Main {

    public static void main(String[] args) {

        BookManager manager = new BookManager(10);

        manager.addBook(new Book(101, "The Alchemist", "Paulo Coelho"));

        manager.addBook(new Book(102, "Clean Code", "Robert C. Martin"));

        manager.addBook(new Book(103, "Thinking in Java", "Bruce Eckel"));

        manager.addBook(new Book(104, "Design Patterns", "Erich Gamma"));

        System.out.println("\nAll Books:");

        manager.displayBooks();

        System.out.println("\nLinear Search for 'Clean Code':");

        manager.linearSearch("Clean Code");

        System.out.println("\nBinary Search for 'Design Patterns':");

        manager.sortBooksByTitle(); // Important before binary search

        manager.binarySearch("Design Patterns");

        System.out.println("\nBinary Search for 'Unknown Book':");

        manager.binarySearch("Unknown Book");

    }

}

