WEEK – 1

DESIGN PATTERNS AND PRINCIPLES

EXERCISE – 1 Implementing the Singleton Pattern (MANDATORY)

**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();

}

return instance;

}

public void log(String message) {

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

}

}

**Main.java:**

public class TestLogger {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("This is the first log message");

Logger logger2 = Logger.getInstance();

logger2.log("This is the second log message");

if (logger1 == logger2) {

System.out.println("Both logger instances are the same (Singleton works).");

} else {

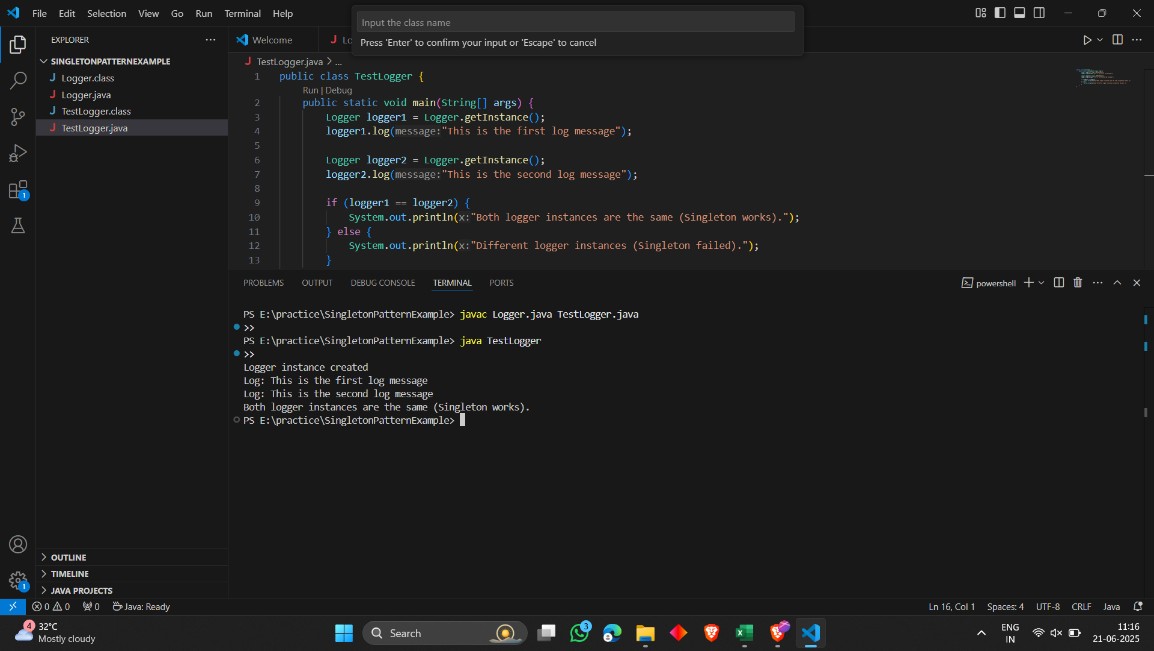
System.out.println("Different logger instances (Singleton failed).");

}

}

}

**Output:**



EXERCISE – 2 Implementing the Factory Method Pattern (MANDATORY)

**Document.java:**

public interface Document {

void open();

}

**WordDocument.java:**

public class WordDocument implements Document {

public void open() {

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

}

}

**PdfDocument.java:**

public class PdfDocument implements Document {

public void open() {

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

}

}

**ExcelDocument.java:**

public class ExcelDocument implements Document {

public void open() {

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

}

}

**DocumentFactory.java:**

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**WordDocumentFactory.java:**

public class WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

**PdfDocumentFactory.java:**

public class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

**ExcelDocumentFactory.java:**

public class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

**FactoryTest.java:**

public class FactoryTest {

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();

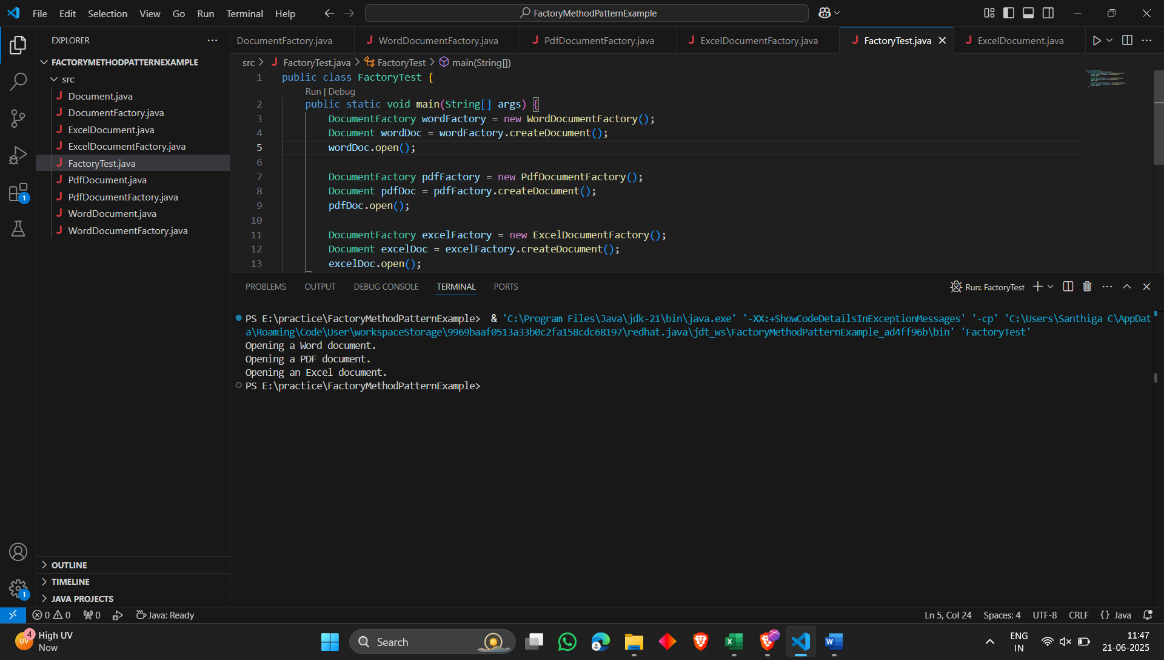
Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

**Output:**



EXERCISE – 3 Implementing the Builder Pattern

**Computer.java:**

public class Computer {

private String cpu;

private String ram;

private String storage;

private boolean hasGraphicsCard;

private boolean hasBluetooth;

private Computer(Builder builder) {

this.cpu = builder.cpu;

this.ram = builder.ram;

this.storage = builder.storage;

this.hasGraphicsCard = builder.hasGraphicsCard;

this.hasBluetooth = builder.hasBluetooth;

}

public static class Builder {

private String cpu;

private String ram;

private String storage;

private boolean hasGraphicsCard;

private boolean hasBluetooth;

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(boolean hasGraphicsCard) {

this.hasGraphicsCard = hasGraphicsCard;

return this;

}

public Builder setBluetooth(boolean hasBluetooth) {

this.hasBluetooth = hasBluetooth;

return this;

}

public Computer build() {

return new Computer(this);

}

}

@Override

public String toString() {

return "Computer [CPU=" + cpu + ", RAM=" + ram + ", Storage=" + storage

+ ", Graphics Card=" + hasGraphicsCard + ", Bluetooth=" + hasBluetooth + "]";

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

Computer basicComputer = new Computer.Builder()

.setCpu("Intel i5")

.setRam("8GB")

.setStorage("256GB SSD")

.build();

Computer gamingComputer = new Computer.Builder()

.setCpu("Intel i9")

.setRam("32GB")

.setStorage("1TB SSD")

.setGraphicsCard(true)

.setBluetooth(true)

.build();

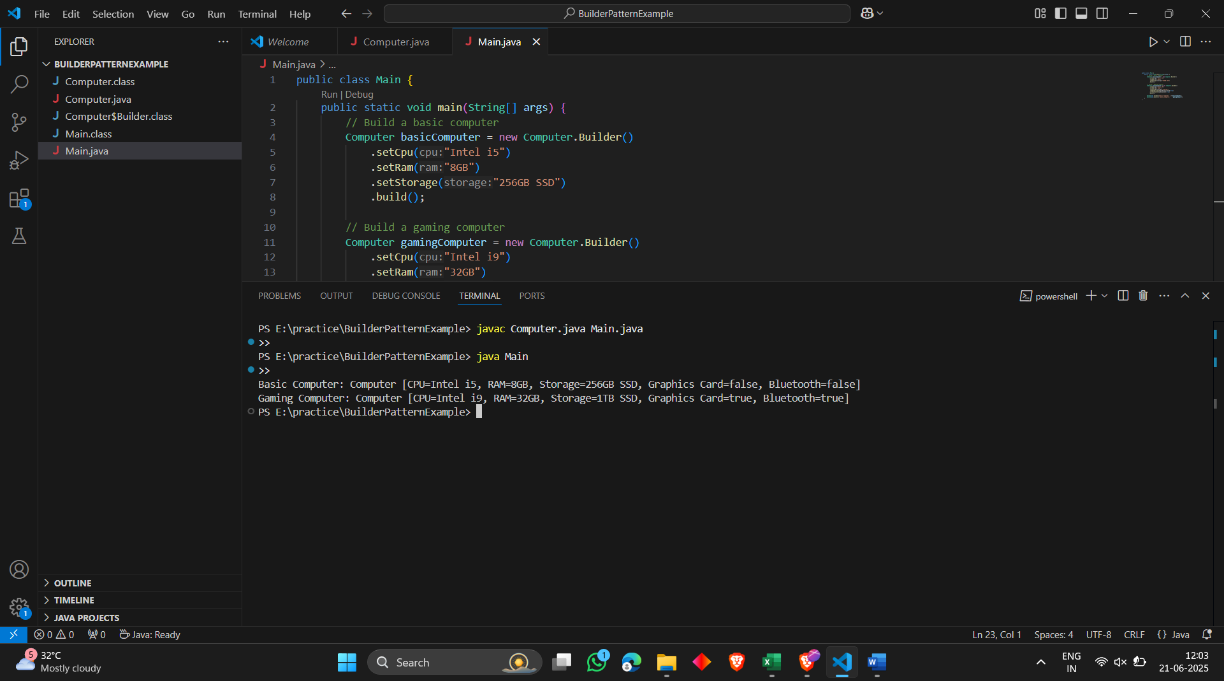
System.out.println("Basic Computer: " + basicComputer);

System.out.println("Gaming Computer: " + gamingComputer);

}

}

**Output:**

**:**

EXERCISE – 4 Implementing the Adapter Pattern

**PaymentProcessor.java:**

public interface PaymentProcessor {

void processPayment(double amount);

}

**PayPalGateway.java:**

public class PayPalGateway {

public void makePayment(double amount) {

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

}

}

**StripeGateway.java:**

public class StripeGateway {

public void sendPayment(double amount) {

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

}

}

**PayPalAdapter.java:**

public class PayPalAdapter implements PaymentProcessor {

private PayPalGateway paypal;

public PayPalAdapter(PayPalGateway paypal) {

this.paypal = paypal;

}

public void processPayment(double amount) {

paypal.makePayment(amount);

}

}

**StripeAdapter.java:**

public class StripeAdapter implements PaymentProcessor {

private StripeGateway stripe;

public StripeAdapter(StripeGateway stripe) {

this.stripe = stripe;

}

public void processPayment(double amount) {

stripe.sendPayment(amount);

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

PaymentProcessor paypalProcessor = new PayPalAdapter(new PayPalGateway());

PaymentProcessor stripeProcessor = new StripeAdapter(new StripeGateway());

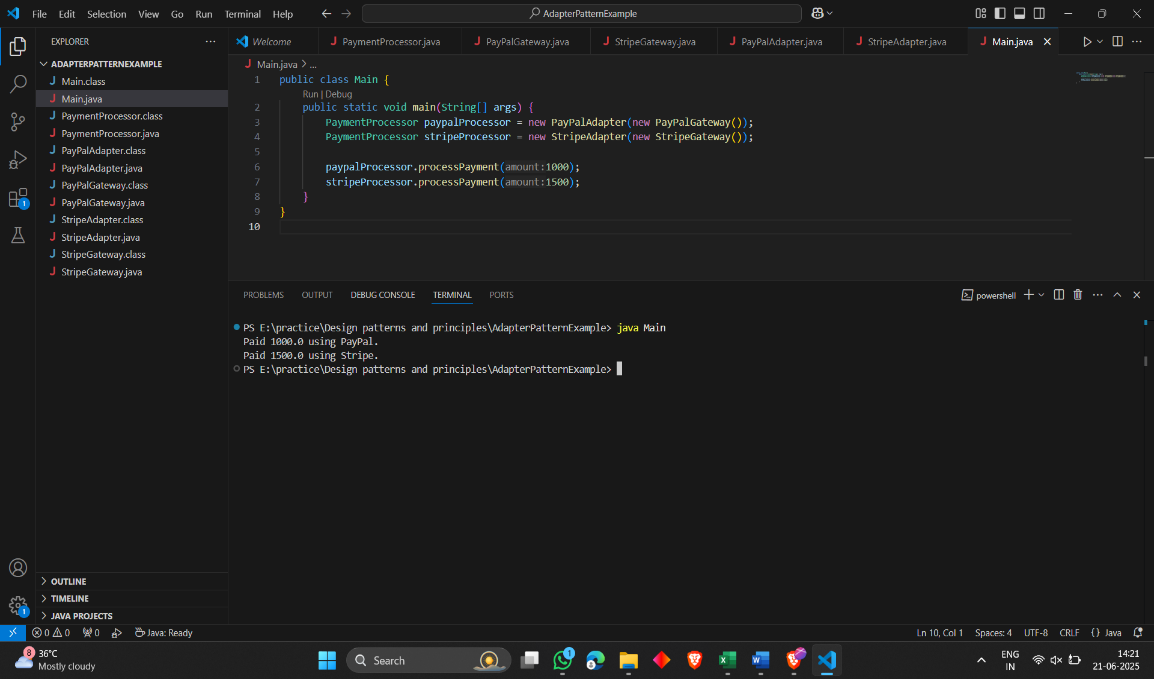
paypalProcessor.processPayment(1000);

stripeProcessor.processPayment(1500);

}

}

**Output:**



EXERCISE - 5 Implementing the Decorator Pattern

**Notifier.java:**

public interface Notifier {

void send(String message);

}

**EmailNotifier.java:**

public class EmailNotifier implements Notifier {

public void send(String message) {

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

}

}

**NotifierDecorator.java:**

public abstract class NotifierDecorator implements Notifier {

protected Notifier wrappedNotifier;

public NotifierDecorator(Notifier notifier) {

this.wrappedNotifier = notifier;

}

public void send(String message) {

wrappedNotifier.send(message);

}

}

**SMSNotifierDecorator.java:**

public class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

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

}

}

**SlackNotifierDecorator.java:**

public class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

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

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

Notifier notifier = new EmailNotifier();

notifier = new SMSNotifierDecorator(notifier);

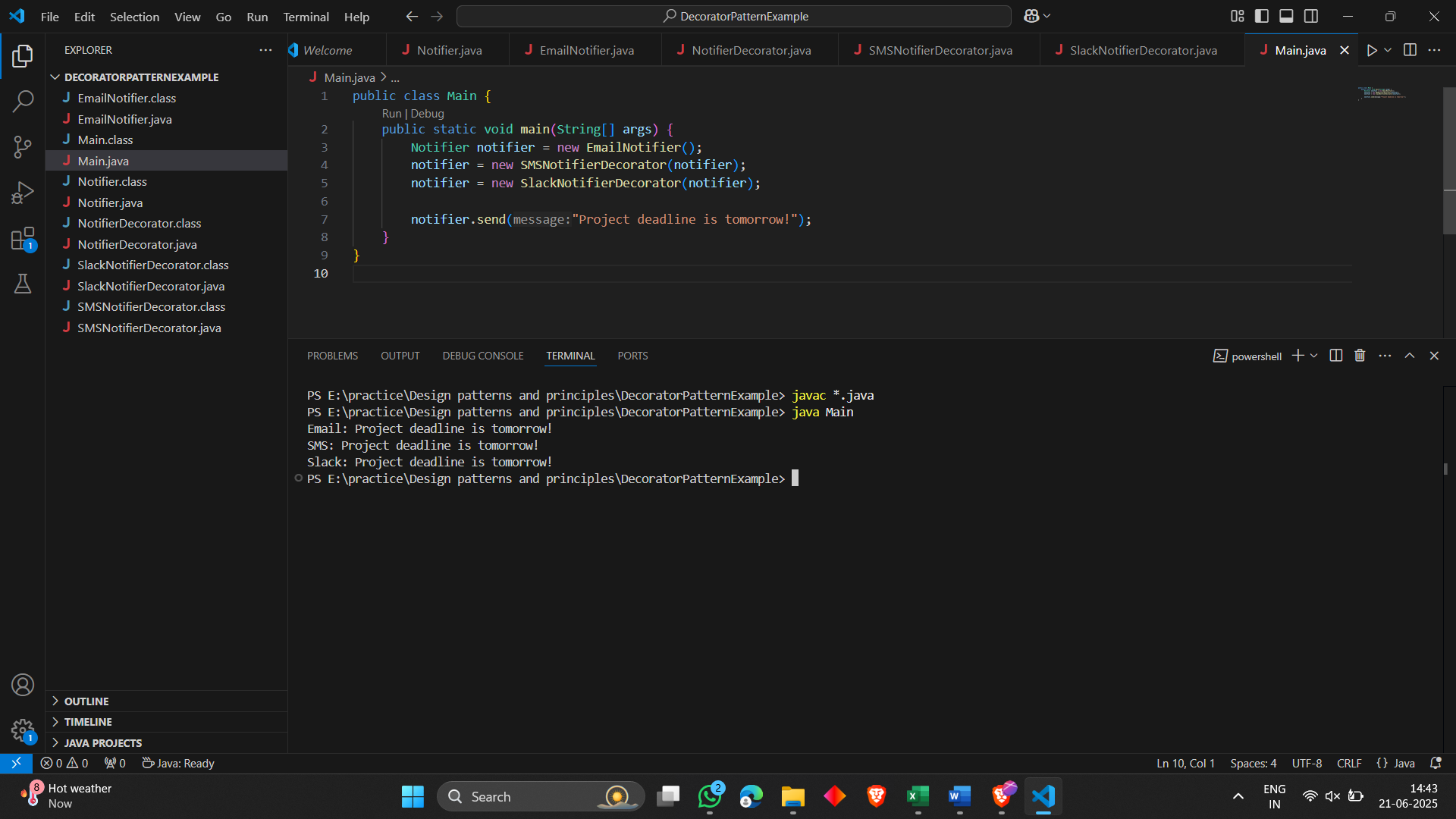
notifier = new SlackNotifierDecorator(notifier);

notifier.send("Project deadline is tomorrow!");

}

}

**Output:**

****

EXERCISE - 6 Implementing the Proxy Pattern

**Image.java:**

public interface Image {

void display();

}

**RealImage.java:**

public 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);

}

@Override

public void display() {

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

}

}

**ProxyImage.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);

}

realImage.display();

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

Image image1 = new ProxyImage("photo1.jpg");

Image image2 = new ProxyImage("photo2.jpg");

image1.display();

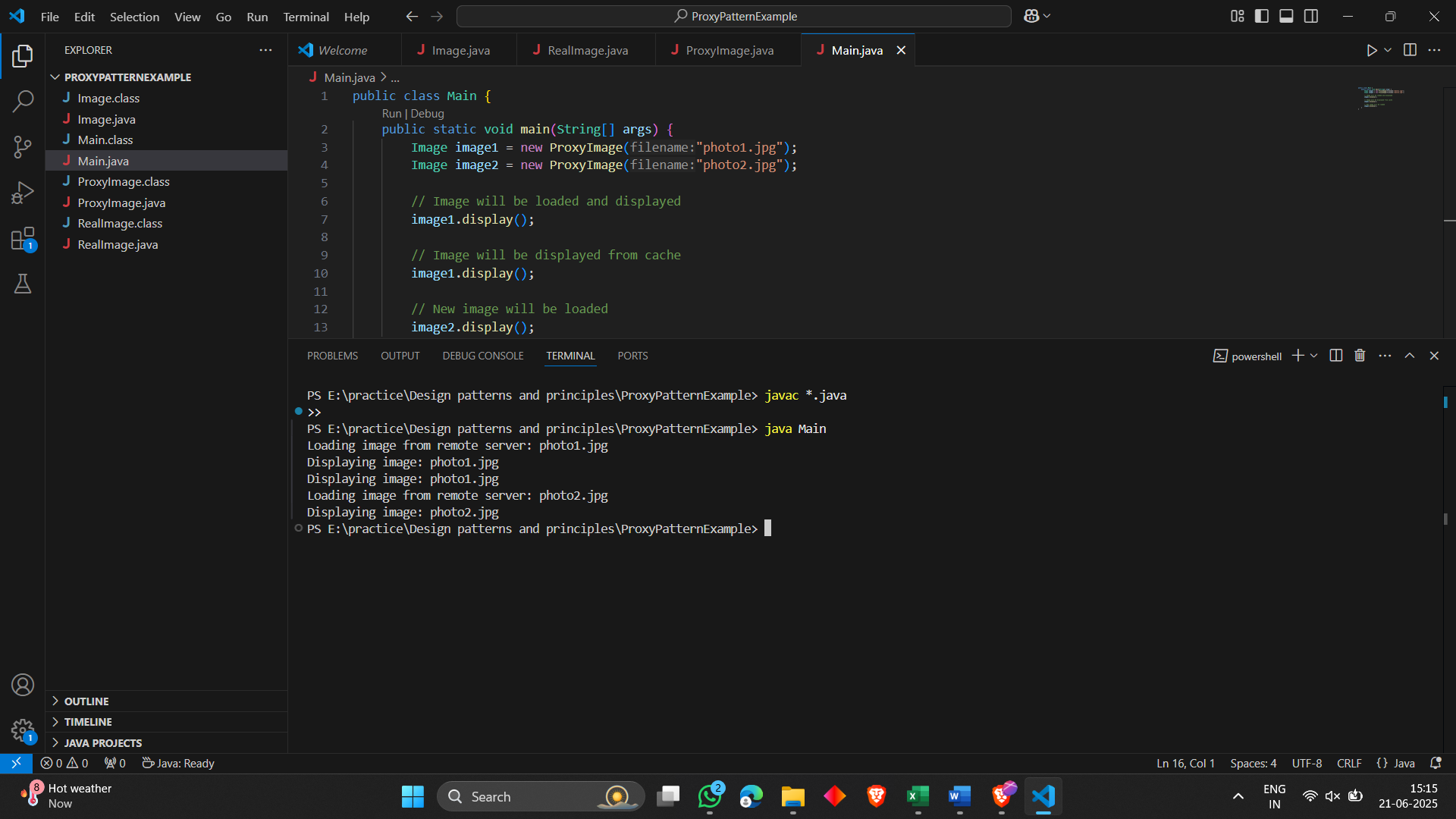
image1.display();

image2.display();

}

}

**Output:**



EXERCISE - 7 Implementing the Observer Pattern

**Observer.java:**

public interface Observer {

void update(String stockName, double price);

}

**Stock.java:**

public interface Stock {

void registerObserver(Observer o);

void removeObserver(Observer o);

void notifyObservers();

}

**StockMarket.java:**

import java.util.\*;

public class StockMarket implements Stock {

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

private String stockName;

private double price;

public void setStock(String stockName, double price) {

this.stockName = stockName;

this.price = 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, price);

}

}

}

**MobileApp.java:**

public class MobileApp implements Observer {

@Override

public void update(String stockName, double price) {

System.out.println("MobileApp: " + stockName + " price updated to ₹" + price);

}

}

**WebApp.java:**

public class WebApp implements Observer {

@Override

public void update(String stockName, double price) {

System.out.println("WebApp: " + stockName + " price updated to ₹" + price);

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

StockMarket market = new StockMarket();

Observer mobile = new MobileApp();

Observer web = new WebApp();

market.registerObserver(mobile);

market.registerObserver(web);

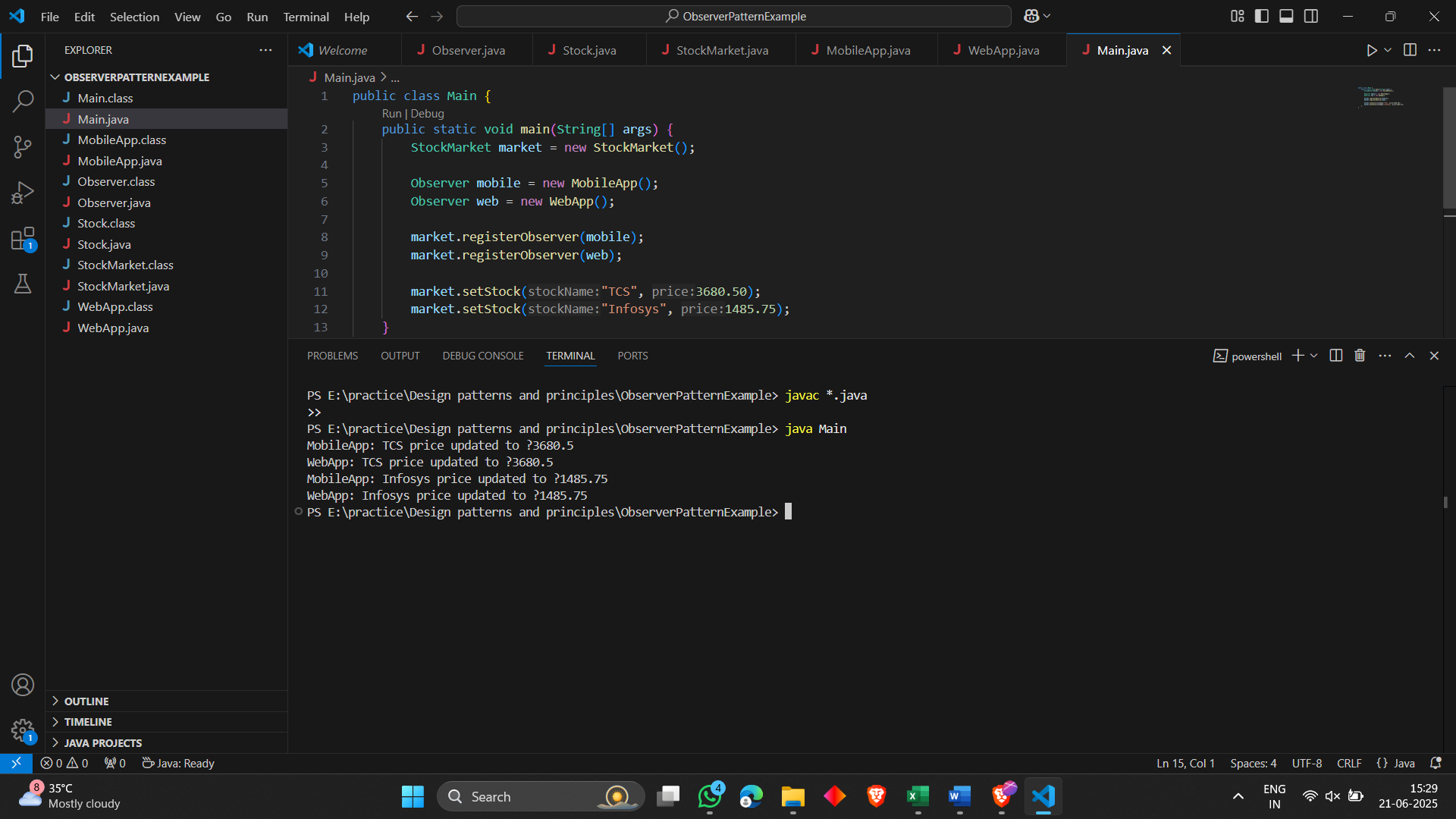
market.setStock("TCS", 3680.50);

market.setStock("Infosys", 1485.75);

}

}

**Output:**

****

EXERCISE-8 Implementing the Strategy Pattern

**PaymentStrategy.java:**

public interface PaymentStrategy {

void pay(int amount);

}

**CreditCardPayment.java:**

public class CreditCardPayment implements PaymentStrategy {

private String cardNumber;

public CreditCardPayment(String cardNumber) {

this.cardNumber = cardNumber;

}

public void pay(int amount) {

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

}

}

**PayPalPayment.java:**

public class PayPalPayment implements PaymentStrategy {

private String email;

public PayPalPayment(String email) {

this.email = email;

}

public void pay(int amount) {

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

}

}

**PaymentContext.java:**

public class PaymentContext {

private PaymentStrategy strategy;

public void setPaymentStrategy(PaymentStrategy strategy) {

this.strategy = strategy;

}

public void payAmount(int amount) {

strategy.pay(amount);

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

PaymentContext context = new PaymentContext();

context.setPaymentStrategy(new CreditCardPayment("1234"));

context.payAmount(1500);

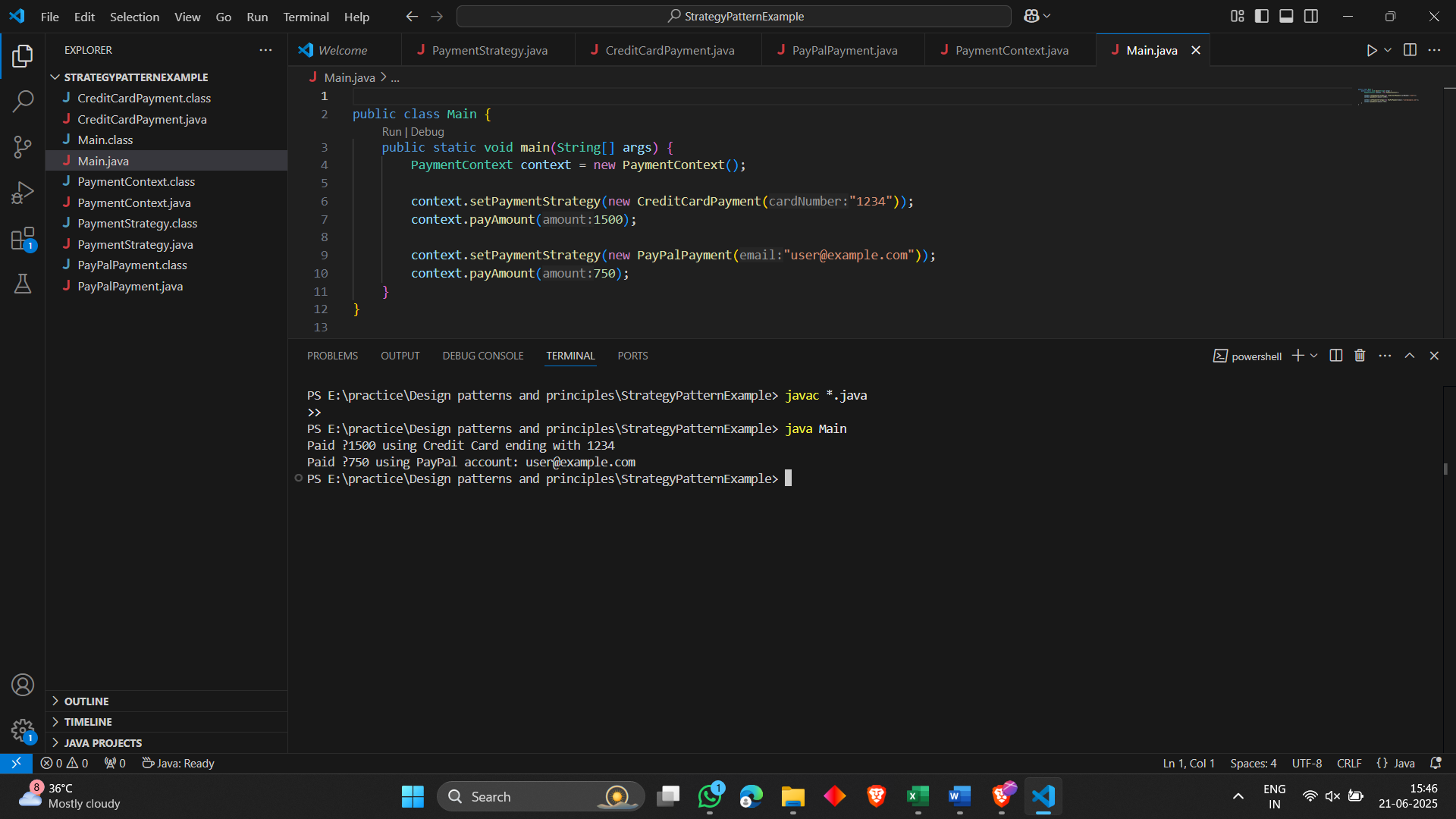
context.setPaymentStrategy(new PayPalPayment("user@example.com"));

context.payAmount(750);

}

}

**Output:**

****

EXERCISE – 9 Implementing the Command Pattern

**Command.java:**

public interface Command {

void execute();

}

**LightOnCommand.java:**

public class LightOnCommand implements Command {

private Light light;

public LightOnCommand(Light light) {

this.light = light;

}

public void execute() {

light.turnOn();

}

}

**LightOffCommand.java:**

public class LightOffCommand implements Command {

private Light light;

public LightOffCommand(Light light) {

this.light = light;

}

public void execute() {

light.turnOff();

}

}

**Light.java:**

public class Light {

public void turnOn() {

System.out.println("The light is ON");

}

public void turnOff() {

System.out.println("The light is OFF");

}

}

**RemoteControl.java:**

public class RemoteControl {

private Command command;

public void setCommand(Command command) {

this.command = command;

}

public void pressButton() {

command.execute();

}

}

**Main.java:**

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();

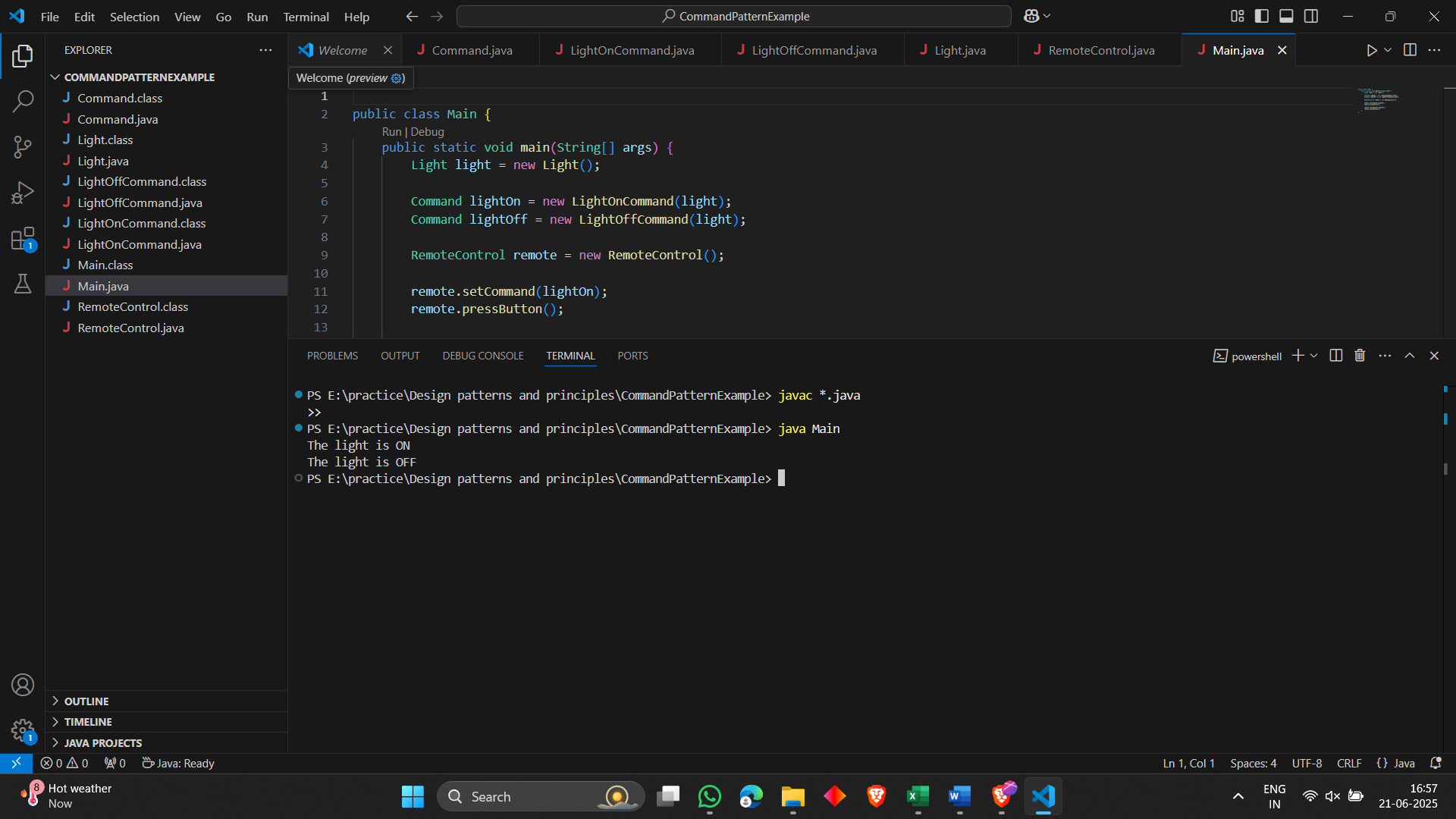
remote.setCommand(lightOff);

remote.pressButton();

}

}

**Output:**



EXERCISE - 10 Implementing the MVC Pattern

**Student.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;

}

public String getName() { return name; }

public String getId() { return id; }

public String getGrade() { return grade; }

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

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

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

}

**StudentView.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);

}

}

**StudentController.java:**

public class StudentController {

private Student model;

private StudentView view;

public StudentController(Student model, StudentView view) {

this.model = model;

this.view = view;

}

public void updateView() {

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

}

public void setStudentName(String name) {

model.setName(name);

}

public void setStudentGrade(String grade) {

model.setGrade(grade);

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

Student student = new Student("Alice", "101", "A");

StudentView view = new StudentView();

StudentController controller = new StudentController(student, view);

controller.updateView();

controller.setStudentName("Bob");

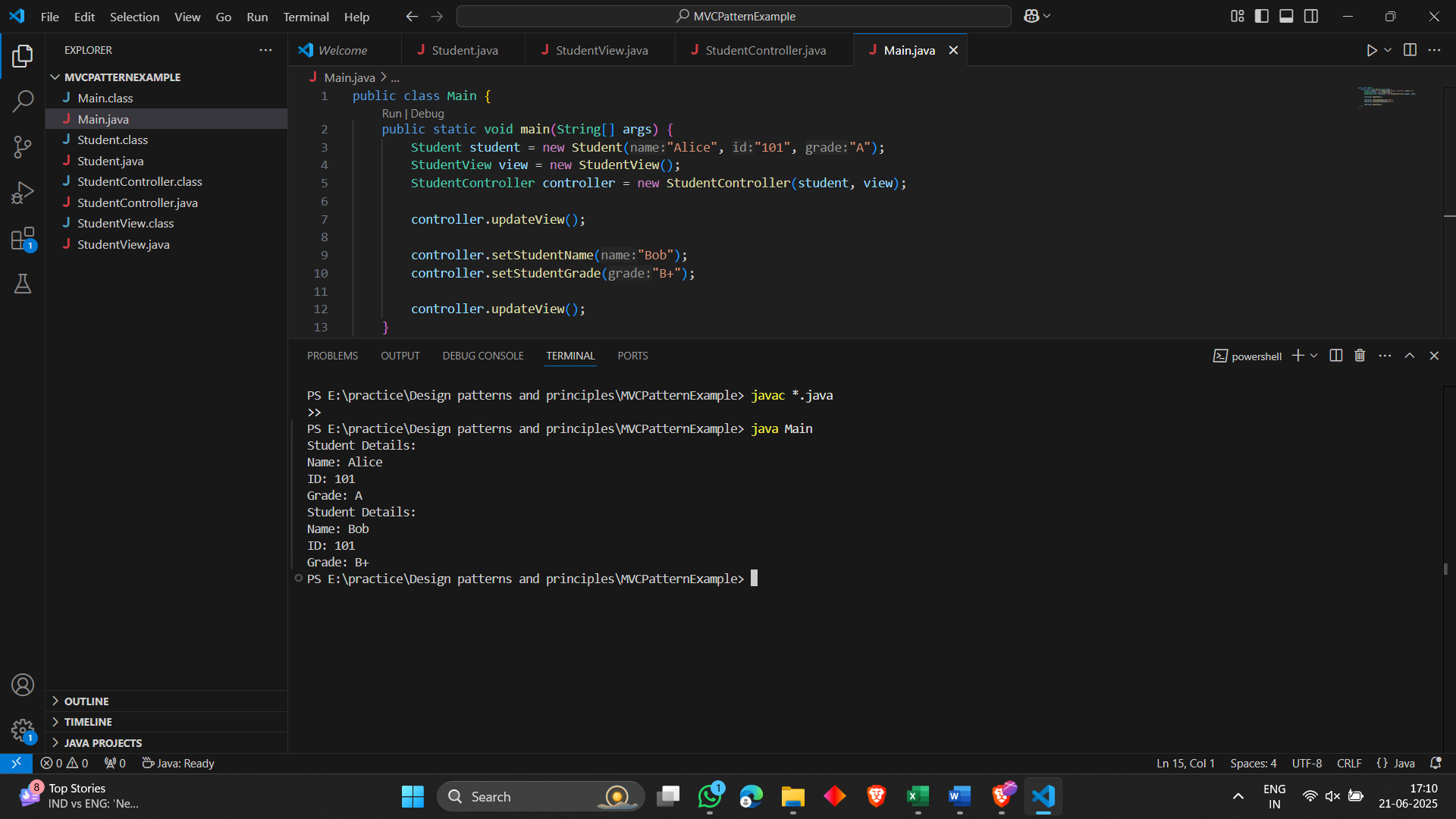
controller.setStudentGrade("B+");

controller.updateView();

}

}

**Output:**

****

EXERCISE - 11 Implementing Dependency Injection

**CustomerRepository.java:**

public interface CustomerRepository {

String findCustomerById(int id);

}

**CustomerRepositoryImpl.java:**

public class CustomerRepositoryImpl implements CustomerRepository {

@Override

public String findCustomerById(int id) {

return "Customer ID: " + id + ", Name: John Doe";

}

}

**CustomerService.java:**

public class CustomerService {

private CustomerRepository customerRepository;

public CustomerService(CustomerRepository customerRepository) {

this.customerRepository = customerRepository;

}

public void displayCustomer(int id) {

String customer = customerRepository.findCustomerById(id);

System.out.println(customer);

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

CustomerRepository repo = new CustomerRepositoryImpl(); // dependency

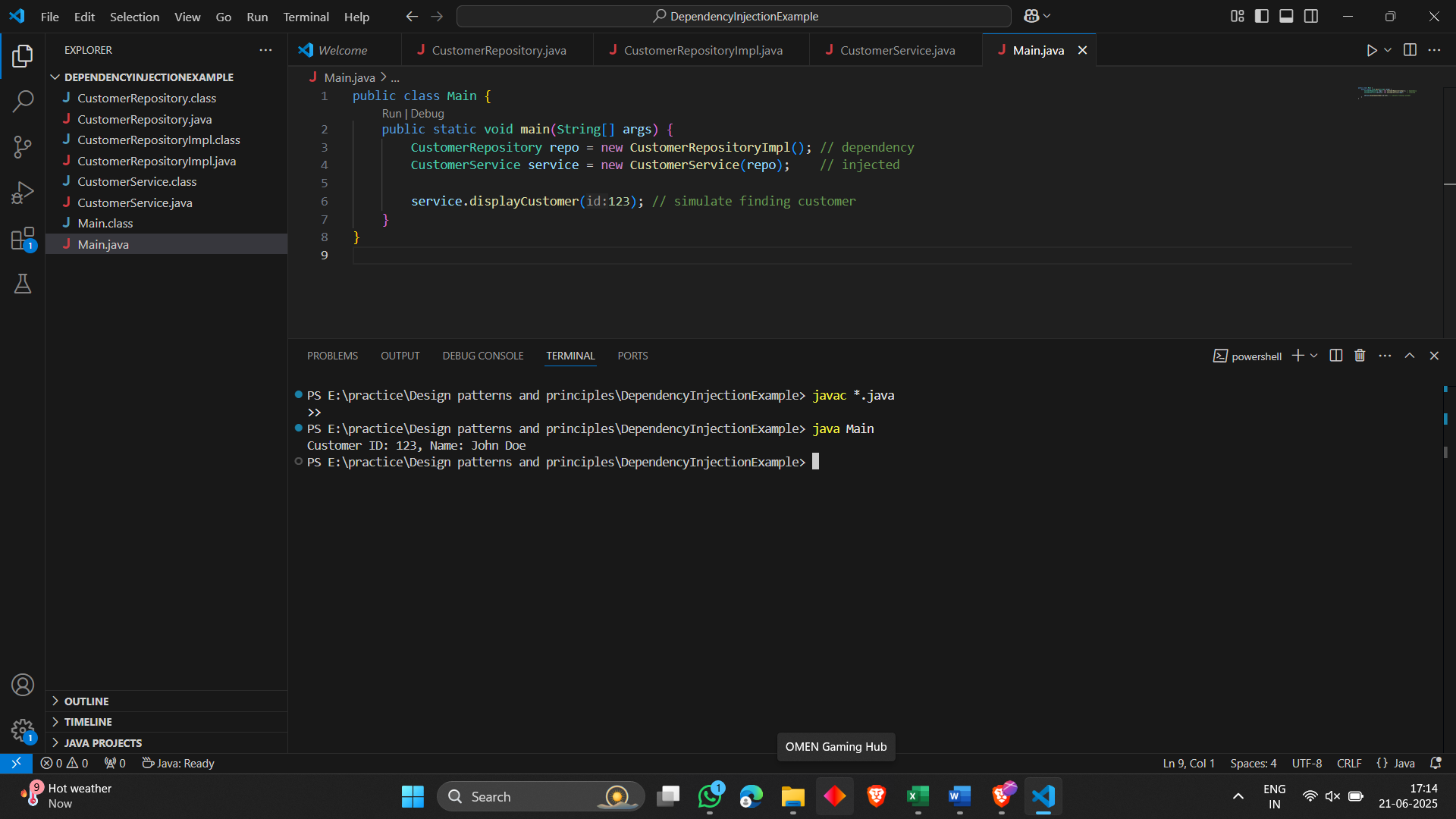
CustomerService service = new CustomerService(repo); // injected

service.displayCustomer(123); // simulate finding customer

}

}

**Output:**

****

ALGORITHMS\_DATA STRUCTURES

EXERCISE – 1 Inventory Management System

**Product.java:**

public class Product {

private String productId;

private String productName;

private int quantity;

private double price;

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

this.productId = productId;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

public String getProductId() { return productId; }

public String getProductName() { return productName; }

public int getQuantity() { return quantity; }

public double getPrice() { return price; }

public void setQuantity(int quantity) { this.quantity = quantity; }

public void setPrice(double price) { this.price = price; }

@Override

public String toString() {

return productId + " | " + productName + " | Qty: " + quantity + " | ₹" + price;

}

}

**InventoryManager.java:**

import java.util.HashMap;

public class InventoryManager {

private HashMap<String, Product> inventory = new HashMap<>();

public void addProduct(Product product) {

inventory.put(product.getProductId(), product);

}

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

Product product = inventory.get(productId);

if (product != null) {

product.setQuantity(quantity);

product.setPrice(price);

}

}

public void deleteProduct(String productId) {

inventory.remove(productId);

}

public void displayInventory() {

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

System.out.println(product);

}

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

InventoryManager manager = new InventoryManager();

Product p1 = new Product("P001", "Laptop", 10, 75000);

Product p2 = new Product("P002", "Mouse", 50, 500);

manager.addProduct(p1);

manager.addProduct(p2);

manager.displayInventory();

manager.updateProduct("P001", 8, 74000);

manager.deleteProduct("P002");

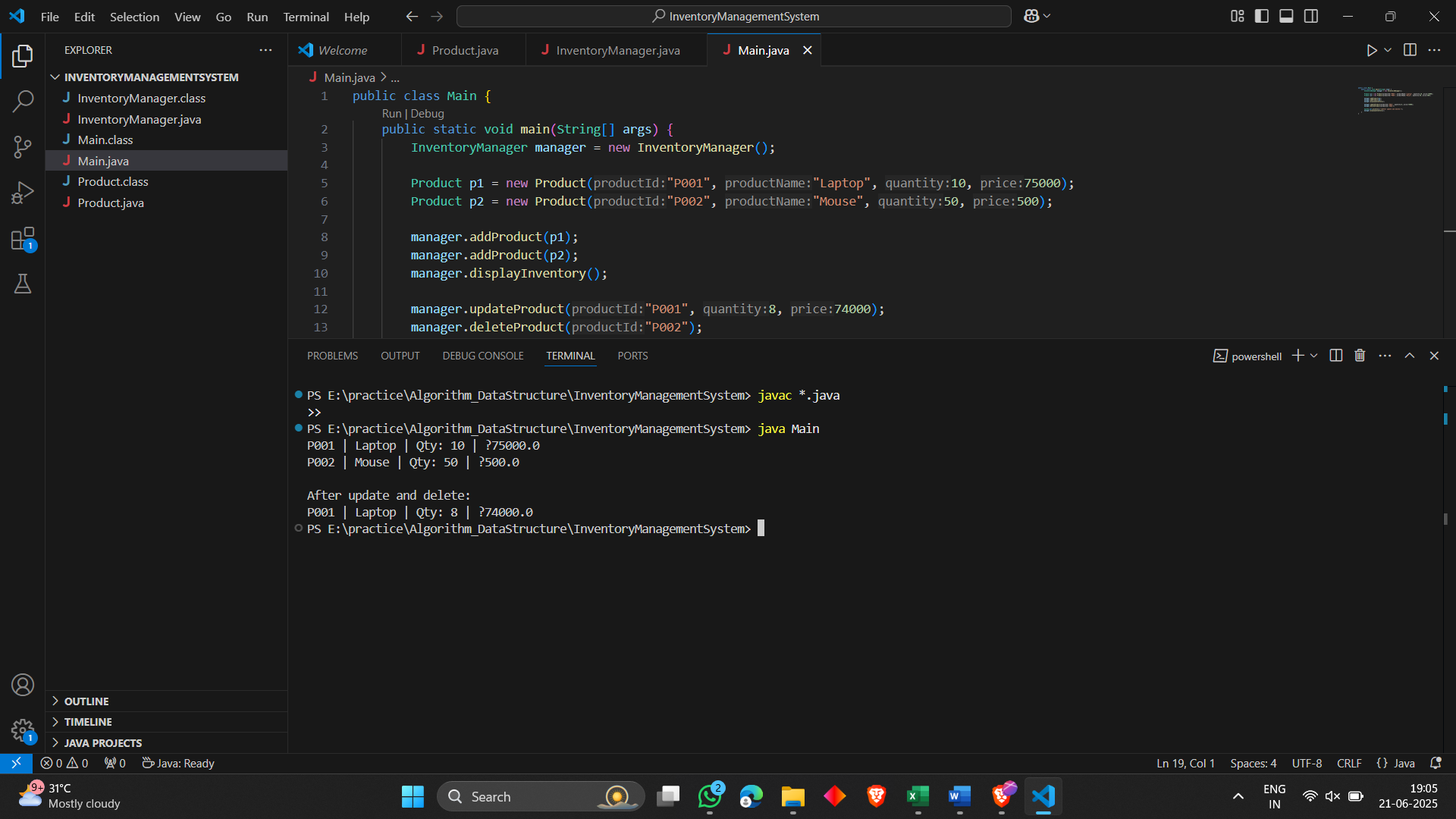
System.out.println("\nAfter update and delete:");

manager.displayInventory();

}

}

**Output:**

****

EXERCISE - 2 E-commerce Platform Search Function (MANDATORY)

**Product.java:**

public class Product {

private String productId;

private String productName;

private String category;

public Product(String productId, String productName, String category) {

this.productId = productId;

this.productName = productName;

this.category = category;

}

public String getProductName() { return productName; }

@Override

public String toString() {

return productId + " | " + productName + " | " + category;

}

}

**SearchEngine.java:**

import java.util.Arrays;

public class SearchEngine {

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

for (Product product : products) {

if (product.getProductName().equalsIgnoreCase(name)) {

return product;

}

}

return null;

}

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

Arrays.sort(products, (a, b) -> a.getProductName().compareToIgnoreCase(b.getProductName()));

int left = 0, right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

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

if (cmp == 0) return products[mid];

else if (cmp < 0) right = mid - 1;

else left = mid + 1;

}

return null;

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

Product[] products = {

new Product("P001", "Laptop", "Electronics"),

new Product("P002", "Shoes", "Apparel"),

new Product("P003", "Watch", "Accessories"),

new Product("P004", "Mouse", "Electronics")

};

Product found1 = SearchEngine.linearSearch(products, "Watch");

System.out.println("Linear Search Result: " + (found1 != null ? found1 : "Not found"));

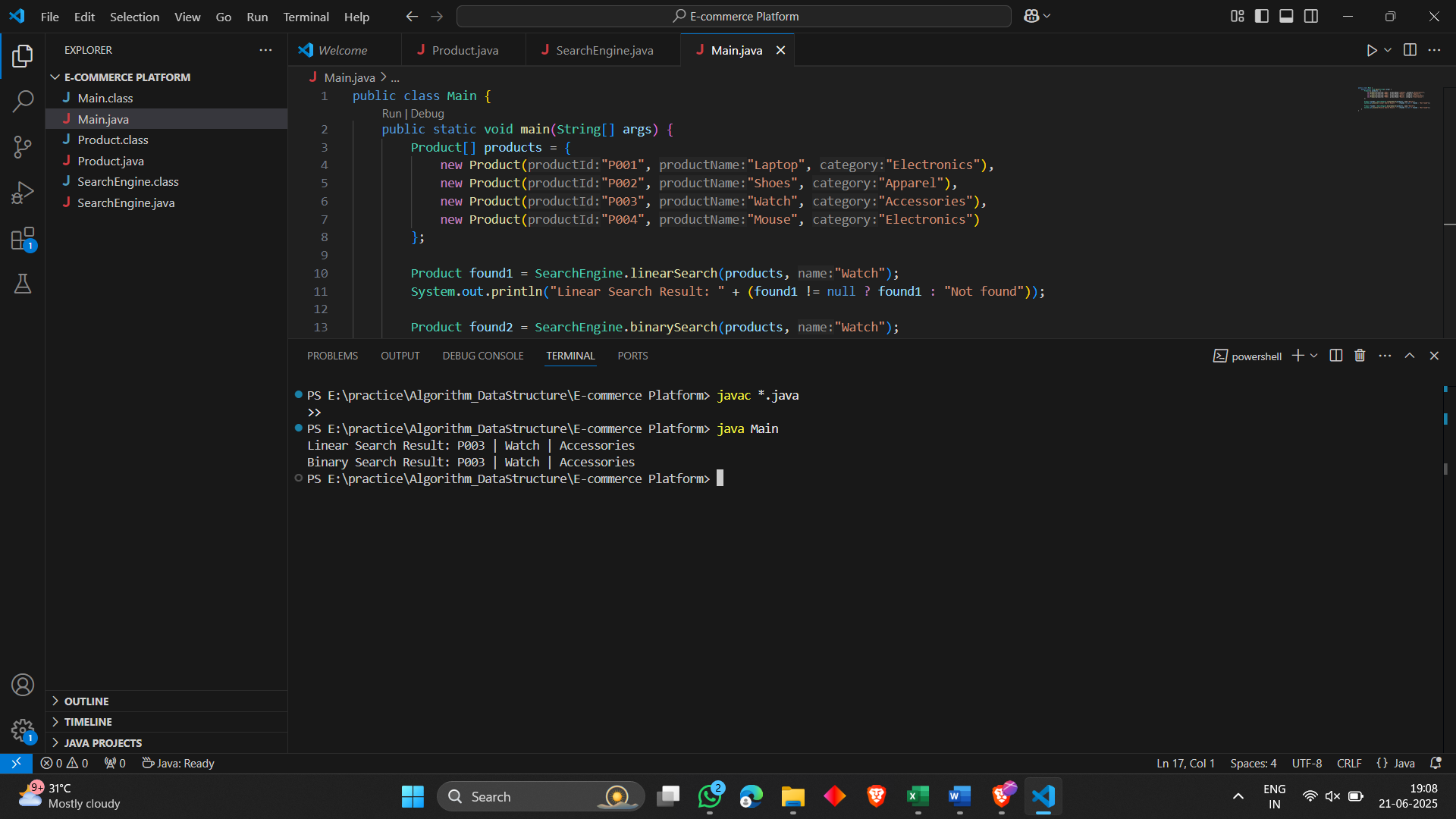
Product found2 = SearchEngine.binarySearch(products, "Watch");

System.out.println("Binary Search Result: " + (found2 != null ? found2 : "Not found"));

}

}

**Output:**



EXERCISE - 3 Sorting Customer Orders

**Order.java:**

public class Order {

private String orderId;

private String customerName;

private double totalPrice;

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

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

public double getTotalPrice() {

return totalPrice;

}

@Override

public String toString() {

return orderId + " | " + customerName + " | ₹" + totalPrice;

}

}

**Sorter.java:**

public class Sorter {

public static void bubbleSort(Order[] orders) {

int n = orders.length;

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

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

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

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].getTotalPrice();

int i = low - 1;

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

if (orders[j].getTotalPrice() <= 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 display(Order[] orders) {

for (Order o : orders) {

System.out.println(o);

}

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

Order[] orders = {

new Order("O101", "Alice", 1200.50),

new Order("O102", "Bob", 800.00),

new Order("O103", "Charlie", 2500.75),

new Order("O104", "Daisy", 500.25)

};

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

Sorter.display(orders);

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

Sorter.bubbleSort(orders);

Sorter.display(orders);

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

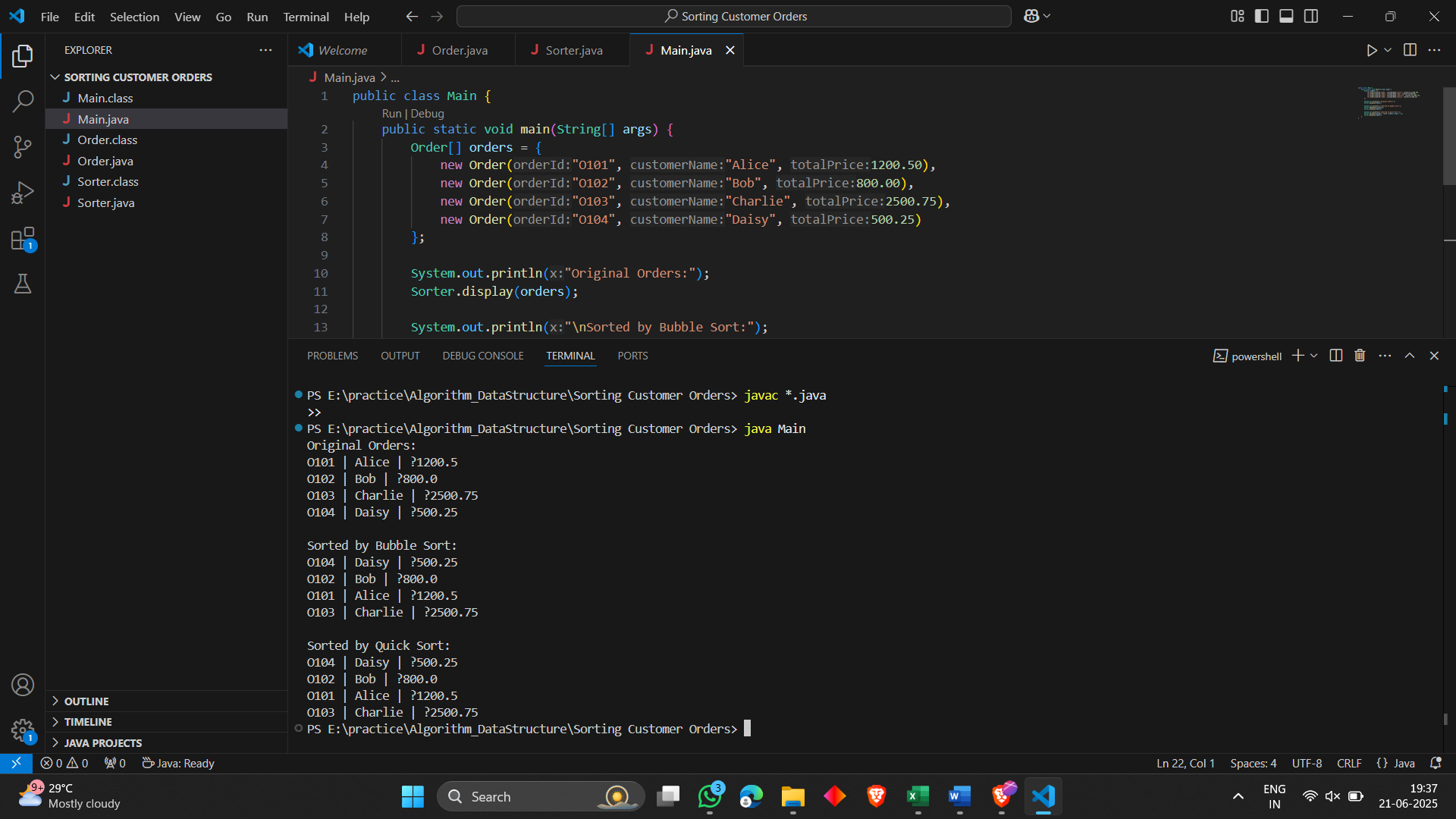
Sorter.quickSort(orders, 0, orders.length - 1);

Sorter.display(orders);

}

}

**Output:**

****

**EXERCISE - 4 Employee Management System**

**Employee.java:**

public class Employee {

private String employeeId;

private String name;

private String position;

private double salary;

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

this.employeeId = employeeId;

this.name = name;

this.position = position;

this.salary = salary;

}

public String getEmployeeId() {

return employeeId;

}

@Override

public String toString() {

return employeeId + " | " + name + " | " + position + " | ₹" + salary;

}

}

**EmployeeManager.java:**

public class EmployeeManager {

private Employee[] employees = new Employee[100];

private int count = 0;

public void addEmployee(Employee e) {

if (count < employees.length) {

employees[count++] = e;

} else {

System.out.println("Array is full");

}

}

public Employee searchEmployee(String id) {

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

if (employees[i].getEmployeeId().equals(id)) {

return employees[i];

}

}

return null;

}

public void deleteEmployee(String id) {

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

if (employees[i].getEmployeeId().equals(id)) {

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

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

}

employees[--count] = null;

return;

}

}

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

}

public void displayEmployees() {

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

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

}

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

EmployeeManager manager = new EmployeeManager();

manager.addEmployee(new Employee("E001", "John", "Developer", 60000));

manager.addEmployee(new Employee("E002", "Alice", "Tester", 50000));

manager.addEmployee(new Employee("E003", "Bob", "Manager", 75000));

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

manager.displayEmployees();

System.out.println("\nSearching E002:");

System.out.println(manager.searchEmployee("E002"));

System.out.println("\nDeleting E001...");

manager.deleteEmployee("E001");

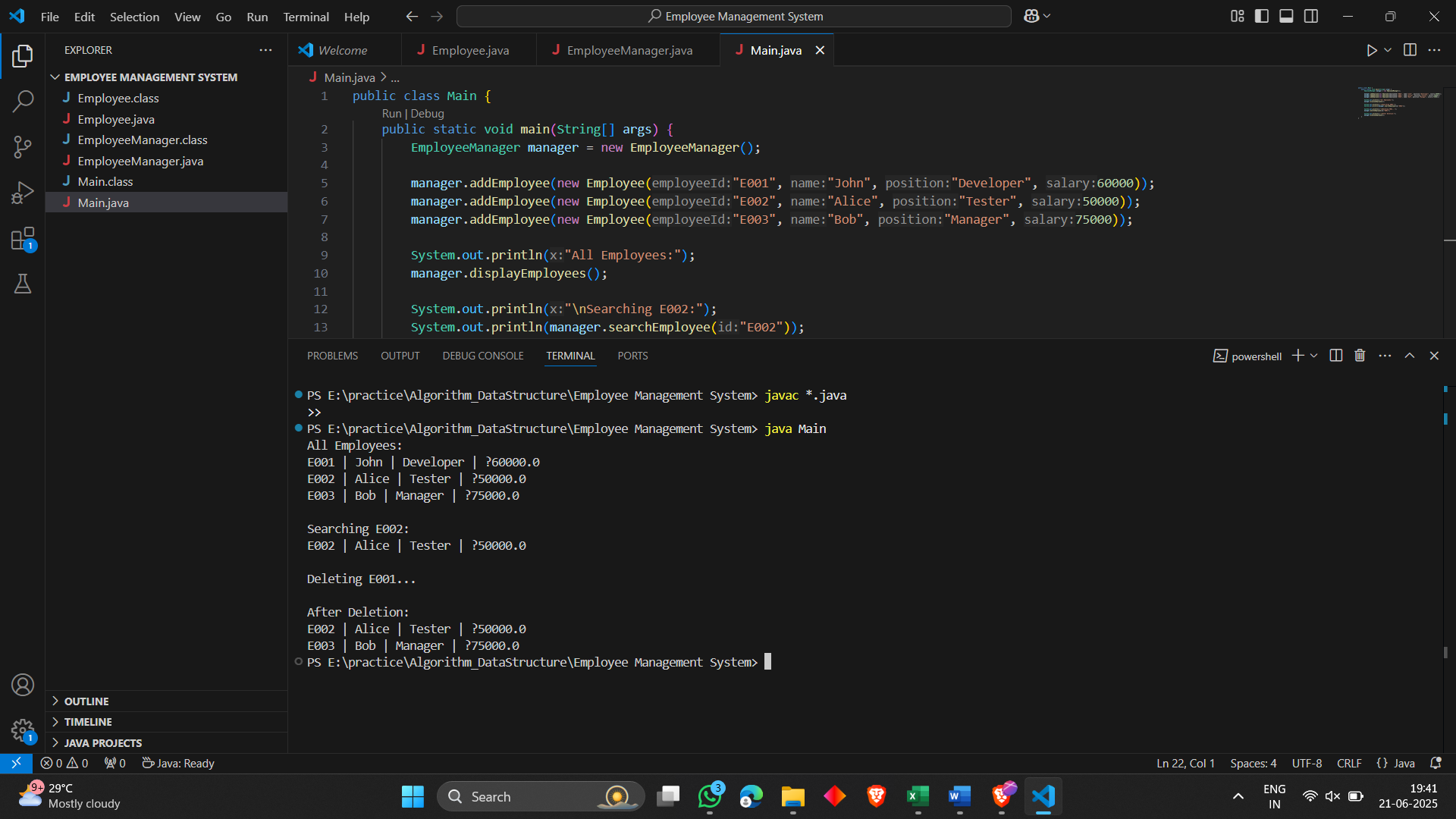
System.out.println("\nAfter Deletion:");

manager.displayEmployees();

}

}

**Output:**

****

EXERCISE - 5 Task Management System

**Task.java:**

public class Task {

String taskId;

String taskName;

String status;

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

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

}

@Override

public String toString() {

return taskId + " | " + taskName + " | " + status;

}

}

**TaskNode.java:**

public class TaskNode {

Task task;

TaskNode next;

public TaskNode(Task task) {

this.task = task;

this.next = null;

}

}

**TaskManager.java:**

public class TaskManager {

private TaskNode head = null;

public void addTask(Task task) {

TaskNode newNode = new TaskNode(task);

newNode.next = head;

head = newNode;

}

public Task searchTask(String taskId) {

TaskNode current = head;

while (current != null) {

if (current.task.taskId.equals(taskId)) {

return current.task;

}

current = current.next;

}

return null;

}

public void deleteTask(String taskId) {

if (head == null) return;

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

head = head.next;

return;

}

TaskNode current = head;

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

current = current.next;

}

if (current.next != null) {

current.next = current.next.next;

}

}

public void displayTasks() {

TaskNode current = head;

while (current != null) {

System.out.println(current.task);

current = current.next;

}

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

TaskManager manager = new TaskManager();

System.out.println("---- Adding Tasks ----");

manager.addTask(new Task("T101", "Design Module", "Pending"));

manager.addTask(new Task("T102", "Implement Login", "In Progress"));

manager.addTask(new Task("T103", "Write Tests", "Completed"));

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

manager.displayTasks();

System.out.println("\n---- Searching for T102 ----");

Task found = manager.searchTask("T102");

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

System.out.println("\n---- Deleting T101 ----");

manager.deleteTask("T101");

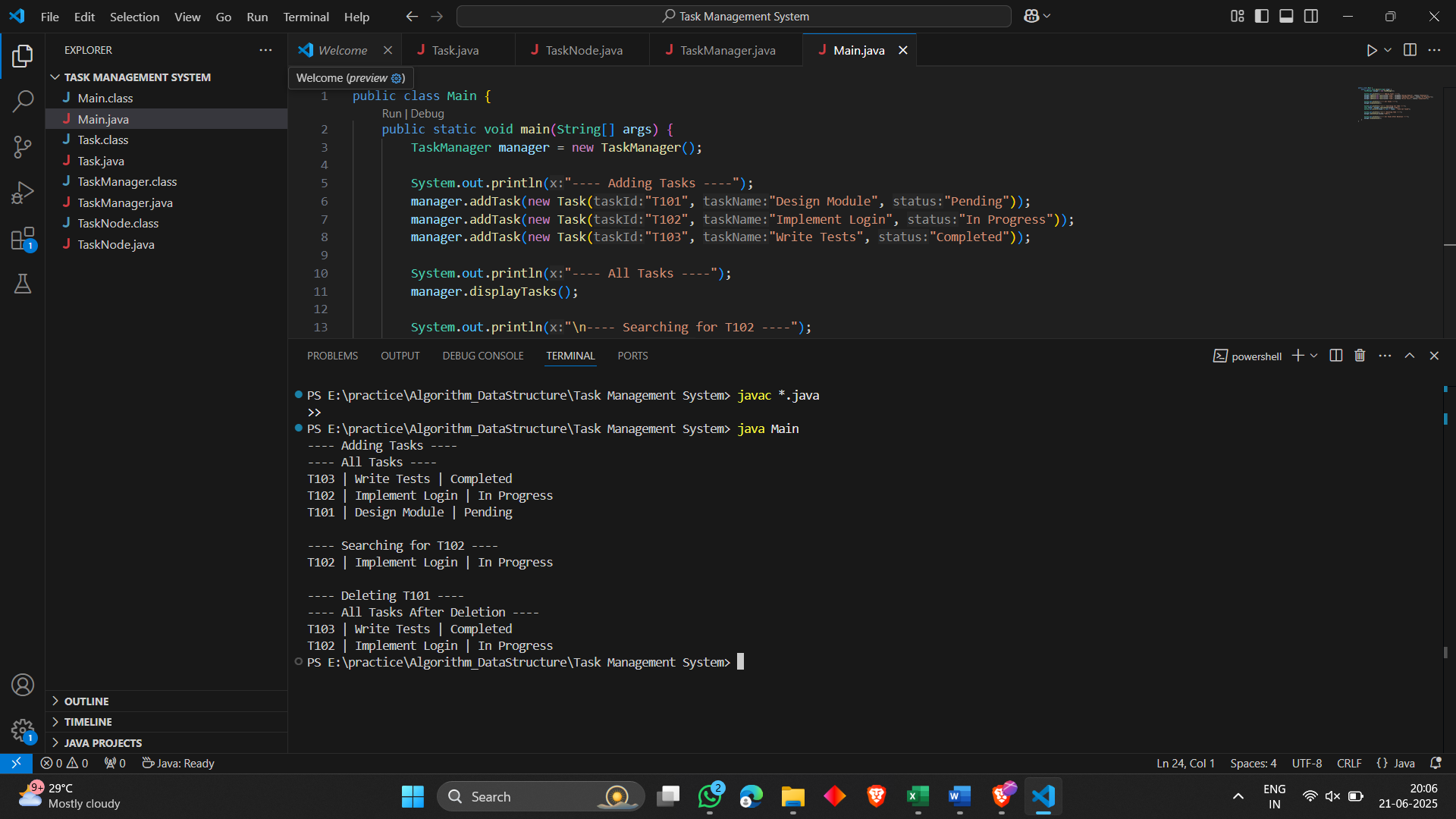
System.out.println("---- All Tasks After Deletion ----");

manager.displayTasks();

}

}

**Output:**



EXERCISE - 6 Library Management System

**Book.java:**

public class Book {

String bookId;

String title;

String author;

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

this.bookId = bookId;

this.title = title;

this.author = author;

}

public String getTitle() { return title; }

@Override

public String toString() {

return bookId + " | " + title + " | " + author;

}

}

**SearchBook.java:**

import java.util.Arrays;

public class SearchBook {

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

for (Book book : books) {

if (book.getTitle().equalsIgnoreCase(title)) {

return book;

}

}

return null;

}

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

Arrays.sort(books, (a, b) -> a.getTitle().compareToIgnoreCase(b.getTitle()));

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

while (low <= high) {

int mid = (low + high) / 2;

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

if (cmp == 0) return books[mid];

else if (cmp < 0) high = mid - 1;

else low = mid + 1;

}

return null;

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

Book[] books = {

new Book("B001", "Java Programming", "James Gosling"),

new Book("B002", "Data Structures", "Mark Allen"),

new Book("B003", "Algorithms", "Robert Sedgewick")

};

System.out.println("---- Linear Search ----");

Book linearResult = SearchBook.linearSearch(books, "Algorithms");

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

System.out.println("\n---- Binary Search ----");

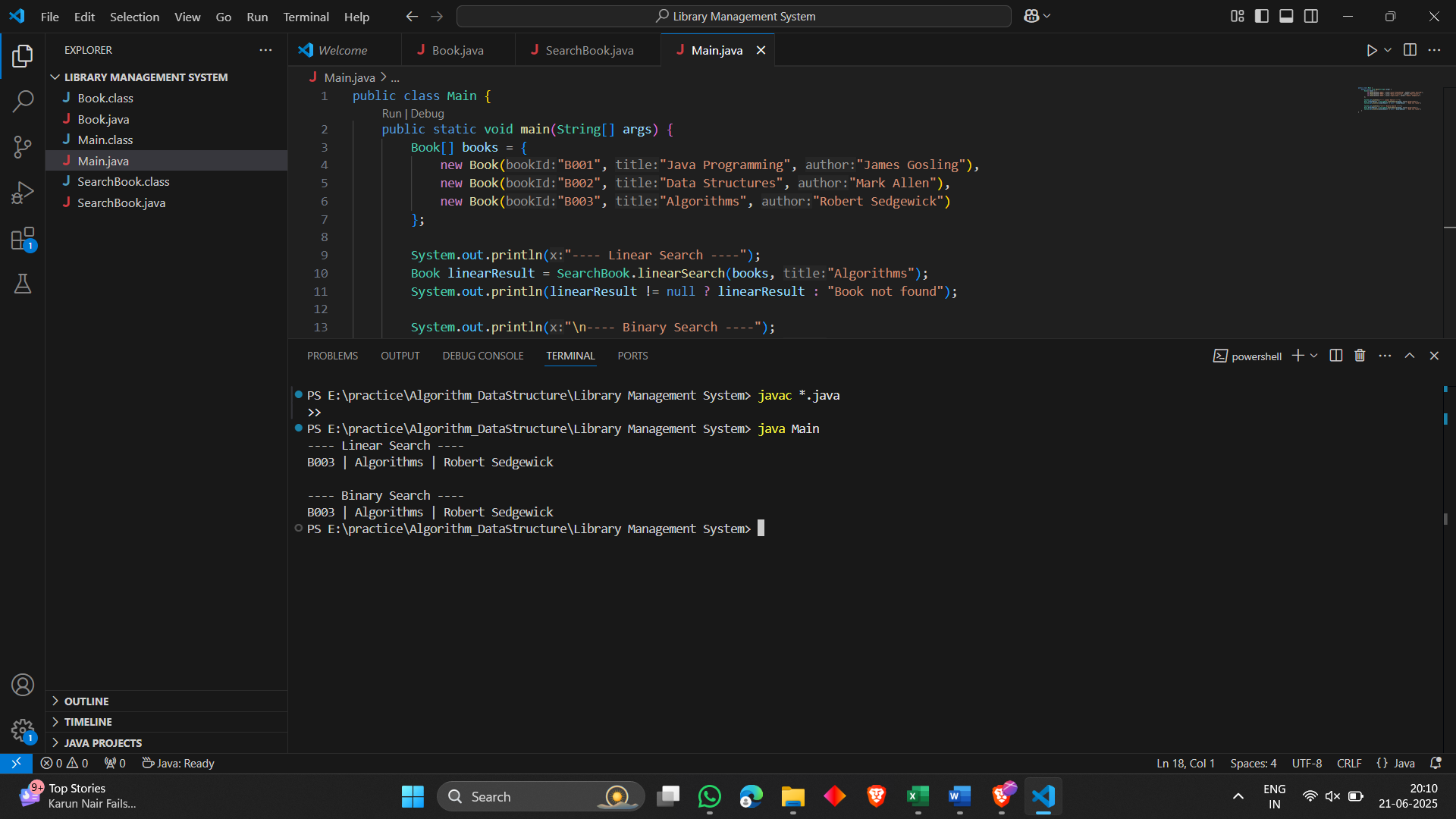
Book binaryResult = SearchBook.binarySearch(books, "Algorithms");

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

}

}

**Output:**

****

EXERCISE - 7 Financial Forecasting (MANDATORY)

**Main.java:**

public class Main {

public static void main(String[] args) {

double initialValue = 10000;

double growthRate = 0.05;

int years = 5;

System.out.println("=== Financial Forecasting ===");

double futureValueRecursive = calculateFutureValueRecursive(initialValue, growthRate, years);

System.out.printf("Future value after %d years (Recursive): %.2f\n", years, futureValueRecursive);

double futureValueIterative = calculateFutureValueIterative(initialValue, growthRate, years);

System.out.printf("Future value after %d years (Iterative): %.2f\n", years, futureValueIterative);

}

public static double calculateFutureValueRecursive(double value, double growthRate, int years) {

if (years == 0) {

return value;

}

return calculateFutureValueRecursive(value, growthRate, years - 1) \* (1 + growthRate);

}

public static double calculateFutureValueIterative(double value, double growthRate, int years) {

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

value \*= (1 + growthRate);

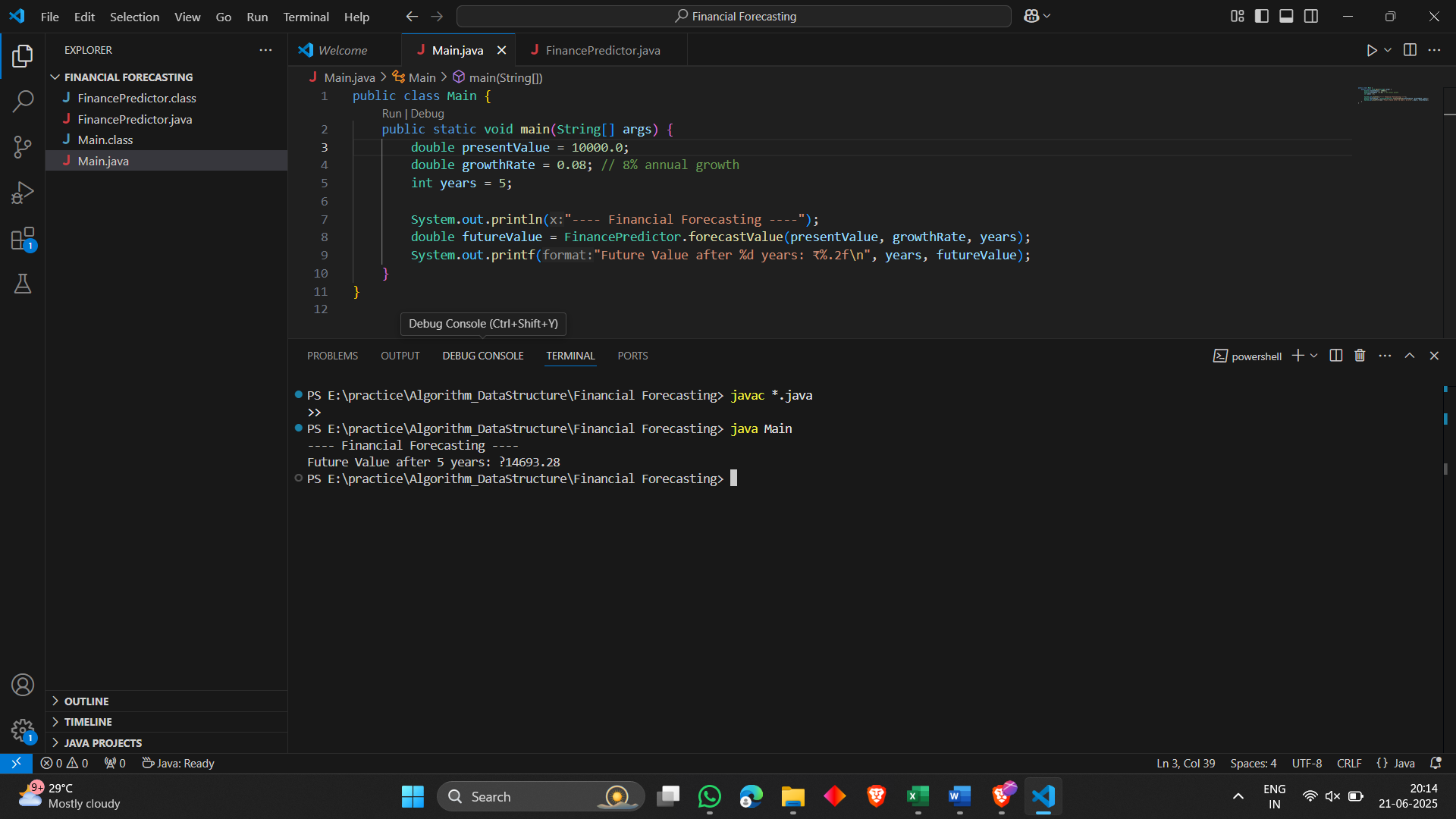
}

return value;

}

}

**Output:**

****