**WEEK 1 – ASSIGNMENTS**

**Exercise 1: E-commerce Platform Search Function**

**import java.util.Arrays;**

**import java.util.Comparator;**

**public class ECommerceSearch {**

**static class Product {**

**int productId;**

**String productName;**

**String category;**

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

**this.productId = productId;**

**this.productName = productName;**

**this.category = category;**

**}**

**@Override**

**public String toString() {**

**return "[" + productId + ", " + productName + ", " + category + "]";**

**}**

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

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

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

**return product;**

**}**

**}**

**return null;**

**}**

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

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

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

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

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

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

**return products[mid];**

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

**low = mid + 1;**

**} else {**

**high = mid - 1;**

**}**

**}**

**return null;**

**}**

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

**Product[] products = {**

**new Product(101, "Shoes", "Footwear"),**

**new Product(102, "Laptop", "Electronics"),**

**new Product(103, "Shirt", "Clothing"),**

**new Product(104, "Headphones", "Electronics"),**

**new Product(105, "Phone", "Electronics")**

**};**

**String searchTerm = "Phone";**

**Product result1 = linearSearch(products, searchTerm);**

**System.out.println("Linear Search Result: " + (result1 != null ? result1 : "Not Found"));**

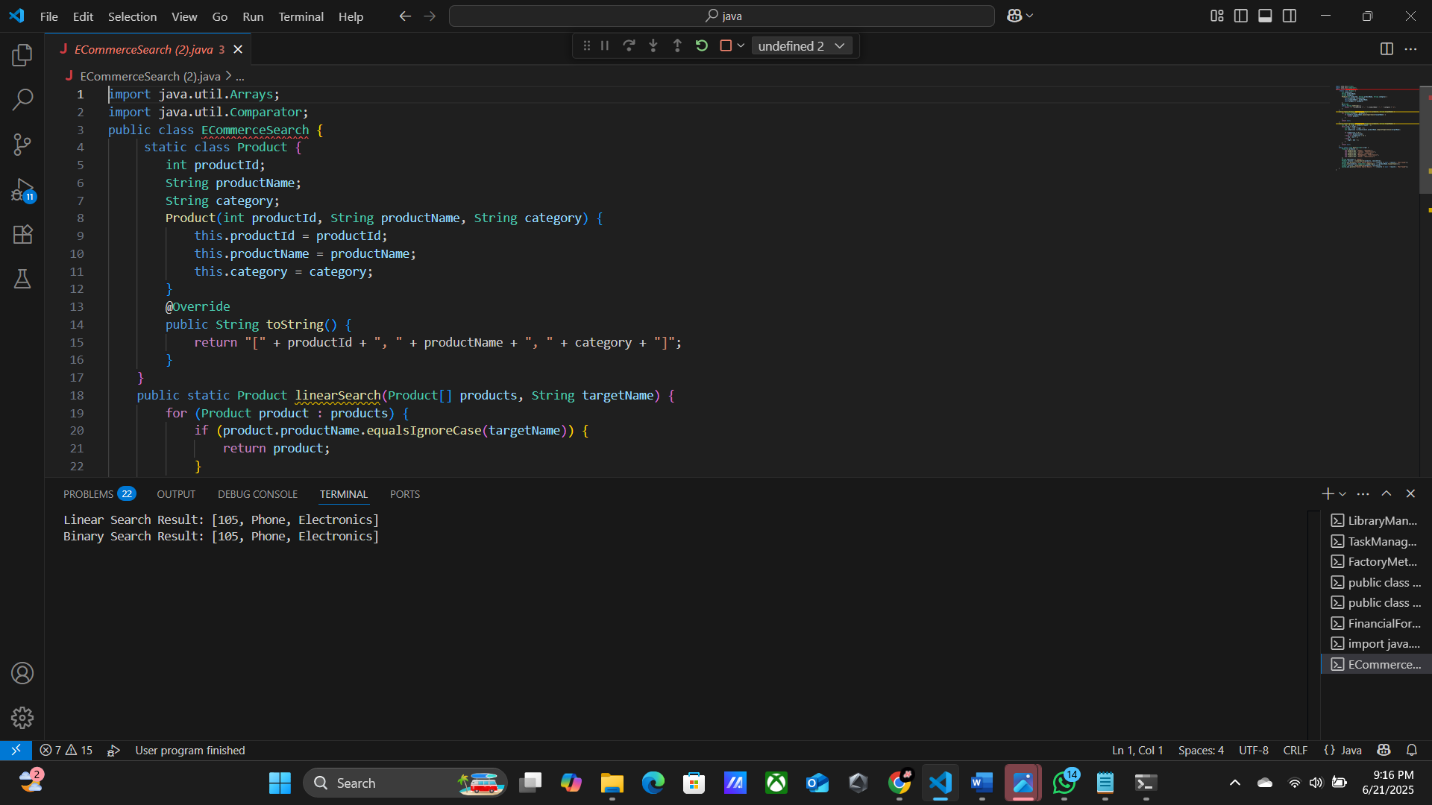
**Arrays.sort(products, Comparator.comparing(p -> p.productName.toLowerCase()));**

**Product result2 = binarySearch(products, searchTerm);**

**System.out.println("Binary Search Result: " + (result2 != null ? result2 : "Not Found"));**

**}**

**}**

****

**Exercise 2: Financial Forecasting**

**public class FinancialForecast {**

**public static double futureValueRecursive(double initialAmount, double growthRate, int years) {**

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

**return initialAmount;**

**}**

**return futureValueRecursive(initialAmount, growthRate, years - 1) \* (1 + growthRate);**

**}**

**public static double futureValueIterative(double initialAmount, double growthRate, int years) {**

**double result = initialAmount;**

**for (int i = 1; i <= years; i++) {**

**result \*= (1 + growthRate);**

**}**

**return result;**

**}**

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

**double initialAmount = 10000;**

**double annualGrowthRate = 0.08;**

**int years = 5;**

**double fvRecursive = futureValueRecursive(initialAmount, annualGrowthRate, years);**

**double fvIterative = futureValueIterative(initialAmount, annualGrowthRate, years);**

**System.out.printf("Future Value (Recursive): ?%.2f%n", fvRecursive);**

**System.out.printf("Future Value (Iterative): ?%.2f%n", fvIterative);**

**}**

**}**

**public class FinancialForecast {**

**public static double futureValueRecursive(double initialAmount, double growthRate, int years) {**

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

**return initialAmount;**

**}**

**return futureValueRecursive(initialAmount, growthRate, years - 1) \* (1 + growthRate);**

**}**

**public static double futureValueIterative(double initialAmount, double growthRate, int years) {**

**double result = initialAmount;**

**for (int i = 1; i <= years; i++) {**

**result \*= (1 + growthRate);**

**}**

**return result;**

**}**

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

**double initialAmount = 10000;**

**double annualGrowthRate = 0.08;**

**int years = 5;**

**double fvRecursive = futureValueRecursive(initialAmount, annualGrowthRate, years);**

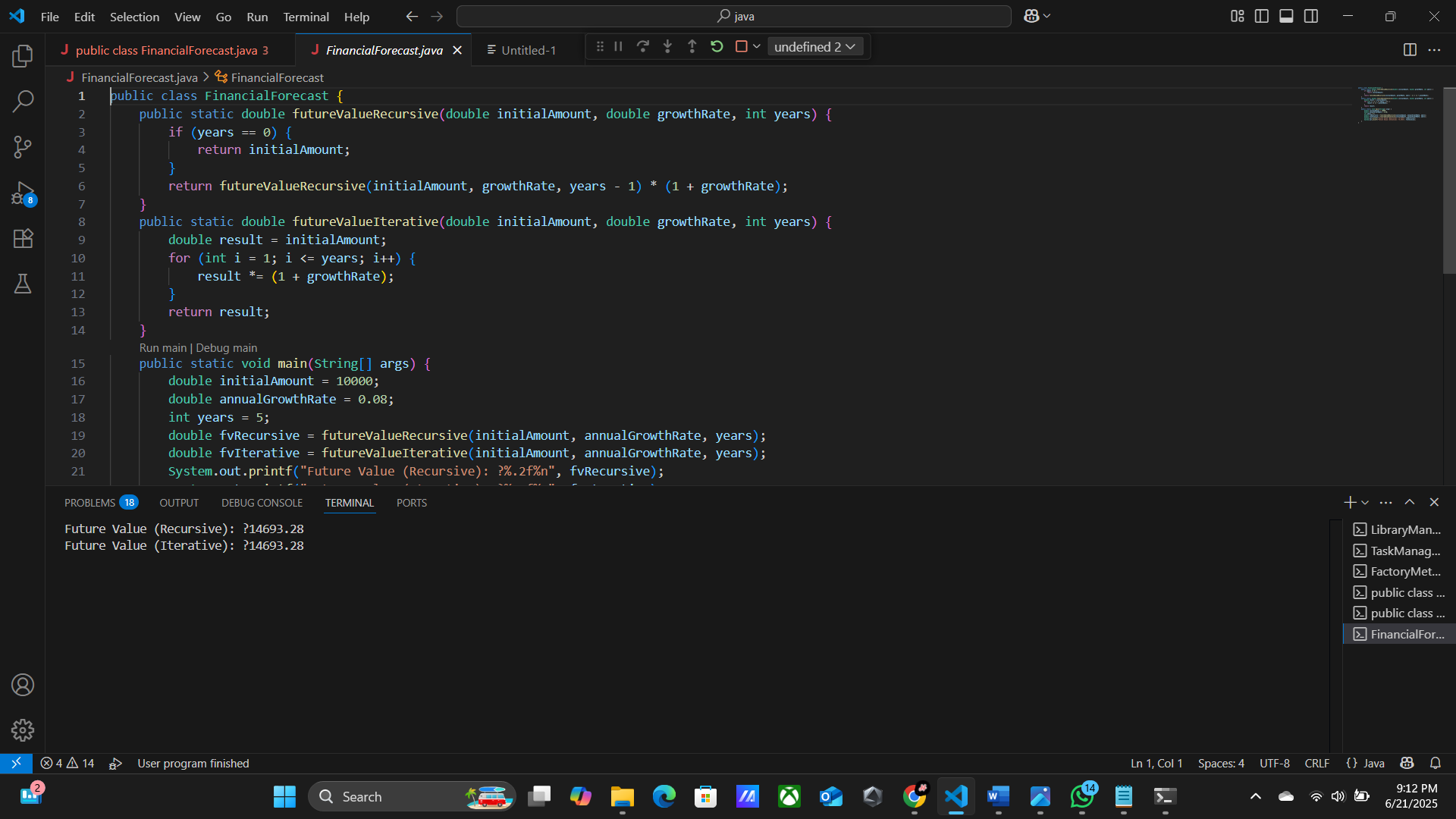
**double fvIterative = futureValueIterative(initialAmount, annualGrowthRate, years);**

**System.out.printf("Future Value (Recursive): ?%.2f%n", fvRecursive);**

**System.out.printf("Future Value (Iterative): ?%.2f%n", fvIterative);**

**}**

**}**

****

**Exercise 3: Implementing the Singleton Pattern**

**public class SingletonPatternExample {**

**static class Logger {**

**private static Logger instance;**

**private Logger() {**

**System.out.println("Logger initialized.");**

**}**

**public static Logger getInstance() {**

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

**instance = new Logger();**

**}**

**return instance;**

**}**

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

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

**}**

**}**

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

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

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

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

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

**if (logger1 == logger2) {**

**System.out.println("Only one instance of Logger exists.");**

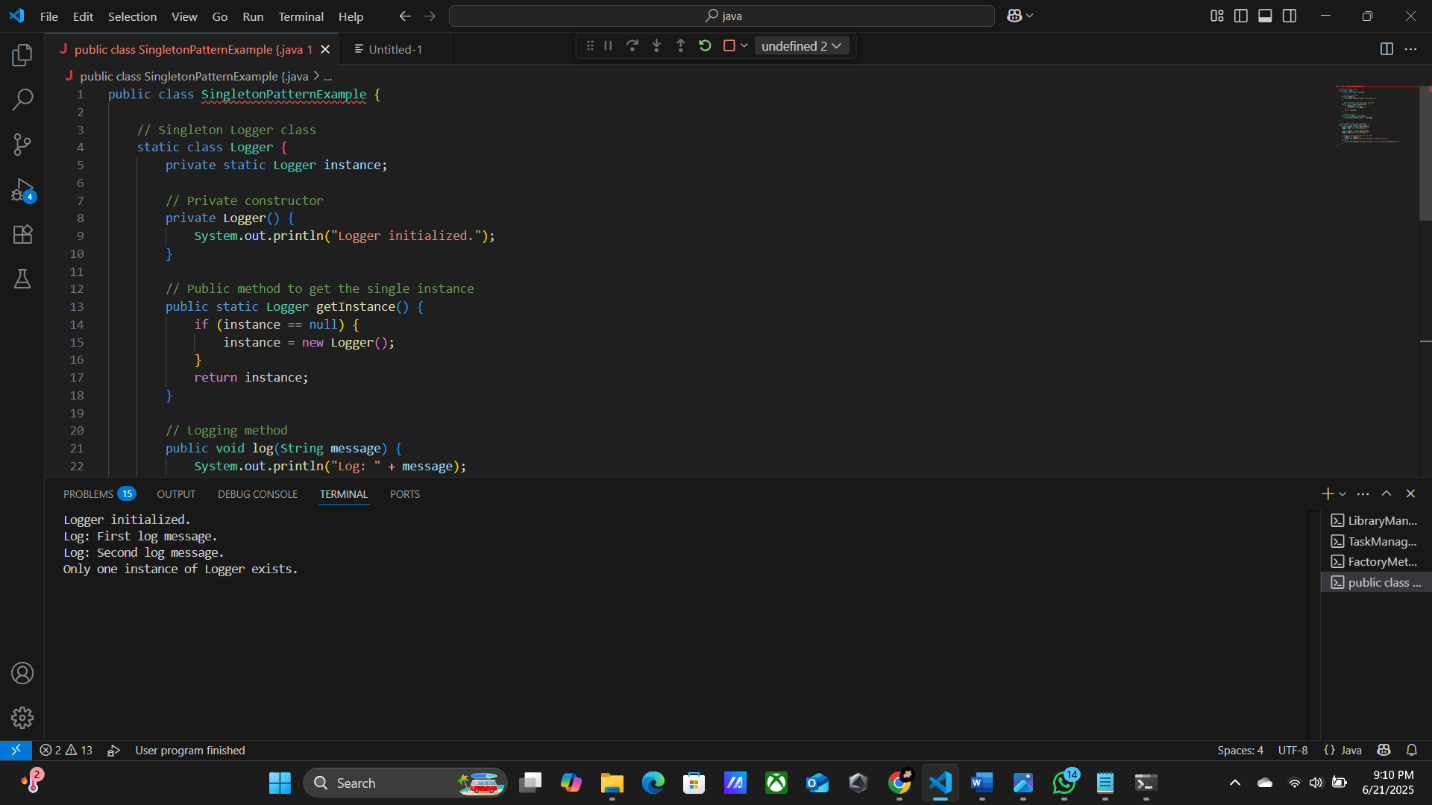
**} else {**

**System.out.println("Multiple instances exist (incorrect implementation).");**

**}**

**}**

**}**

****

**Exercise 4: Implementing the Factory Method Pattern**

**public class FactoryMethodPatternExample {**

**interface Document {**

**void open();**

**}**

**static class WordDocument implements Document {**

**public void open() {**

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

**}**

**}**

**static class PdfDocument implements Document {**

**public void open() {**

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

**}**

**}**

**static class ExcelDocument implements Document {**

**public void open() {**

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

**}**

**}**

**abstract static class DocumentFactory {**

**public abstract Document createDocument();**

**}**

**static class WordDocumentFactory extends DocumentFactory {**

**public Document createDocument() {**

**return new WordDocument();**

**}**

**}**

**static class PdfDocumentFactory extends DocumentFactory {**

**public Document createDocument() {**

**return new PdfDocument();**

**}**

**}**

**static class ExcelDocumentFactory extends DocumentFactory {**

**public Document createDocument() {**

**return new ExcelDocument();**

**}**

**}**

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

**DocumentFactory wordFactory = new WordDocumentFactory();**

**Document wordDoc = wordFactory.createDocument();**

**wordDoc.open();**

**DocumentFactory pdfFactory = new PdfDocumentFactory();**

**Document pdfDoc = pdfFactory.createDocument();**

**pdfDoc.open();**

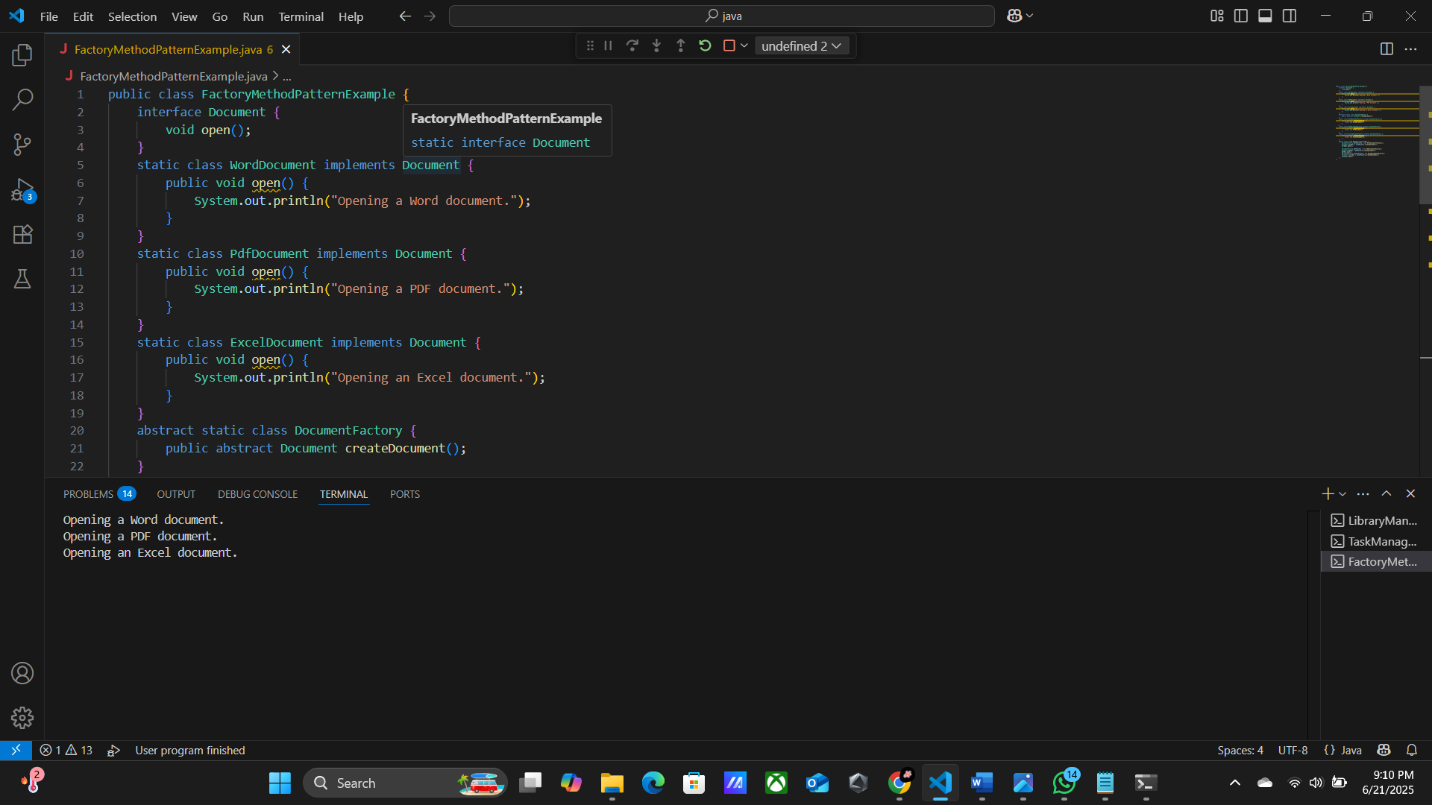
**DocumentFactory excelFactory = new ExcelDocumentFactory();**

**Document excelDoc = excelFactory.createDocument();**

**excelDoc.open();**

**}**

**}**

****

**Exercise 5: Task Management System**

**public class TaskManagementSystem {**

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

**TaskLinkedList taskList = new TaskLinkedList();**

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

**taskList.addTask(new Task(2, "Write Backend", "In Progress"));**

**taskList.addTask(new Task(3, "Testing", "Pending"));**

**taskList.traverseTasks();**

**int searchId = 2;**

**Task found = taskList.searchTask(searchId);**

**System.out.println("\nSearch Result for Task ID " + searchId + ": " + (found != null ? found : "Task not found"));**

**int deleteId = 1;**

**boolean deleted = taskList.deleteTask(deleteId);**

**System.out.println("\nTask Deletion (ID " + deleteId + "): " + (deleted ? "Success" : "Task not found"));**

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

**taskList.traverseTasks();**

**}**

**}**

**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 + " | " + taskName + " | " + status;**

**}**

**}**

**class Node {**

**Task task;**

**Node next;**

**public Node(Task task) {**

**this.task = task;**

**this.next = null;**

**}**

**}**

**class TaskLinkedList {**

**private Node head;**

**public void addTask(Task task) {**

**Node newNode = new Node(task);**

**if (head == null) {**

**head = newNode;**

**} else {**

**Node current = head;**

**while (current.next != null) {**

**current = current.next;**

**}**

**current.next = newNode;**

**}**

**}**

**public Task searchTask(int taskId) {**

**Node current = head;**

**while (current != null) {**

**if (current.task.taskId == taskId) {**

**return current.task;**

**}**

**current = current.next;**

**}**

**return null;**

**}**

**public boolean deleteTask(int taskId) {**

**if (head == null) return false;**

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

**head = head.next;**

**return true;**

**}**

**Node current = head;**

**while (current.next != null) {**

**if (current.next.task.taskId == taskId) {**

**current.next = current.next.next;**

**return true;**

**}**

**current = current.next;**

**}**

**return false;**

**}**

**public void traverseTasks() {**

**if (head == null) {**

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

**return;**

**}**

**Node current = head;**

**System.out.println("Task List:");**

**while (current != null) {**

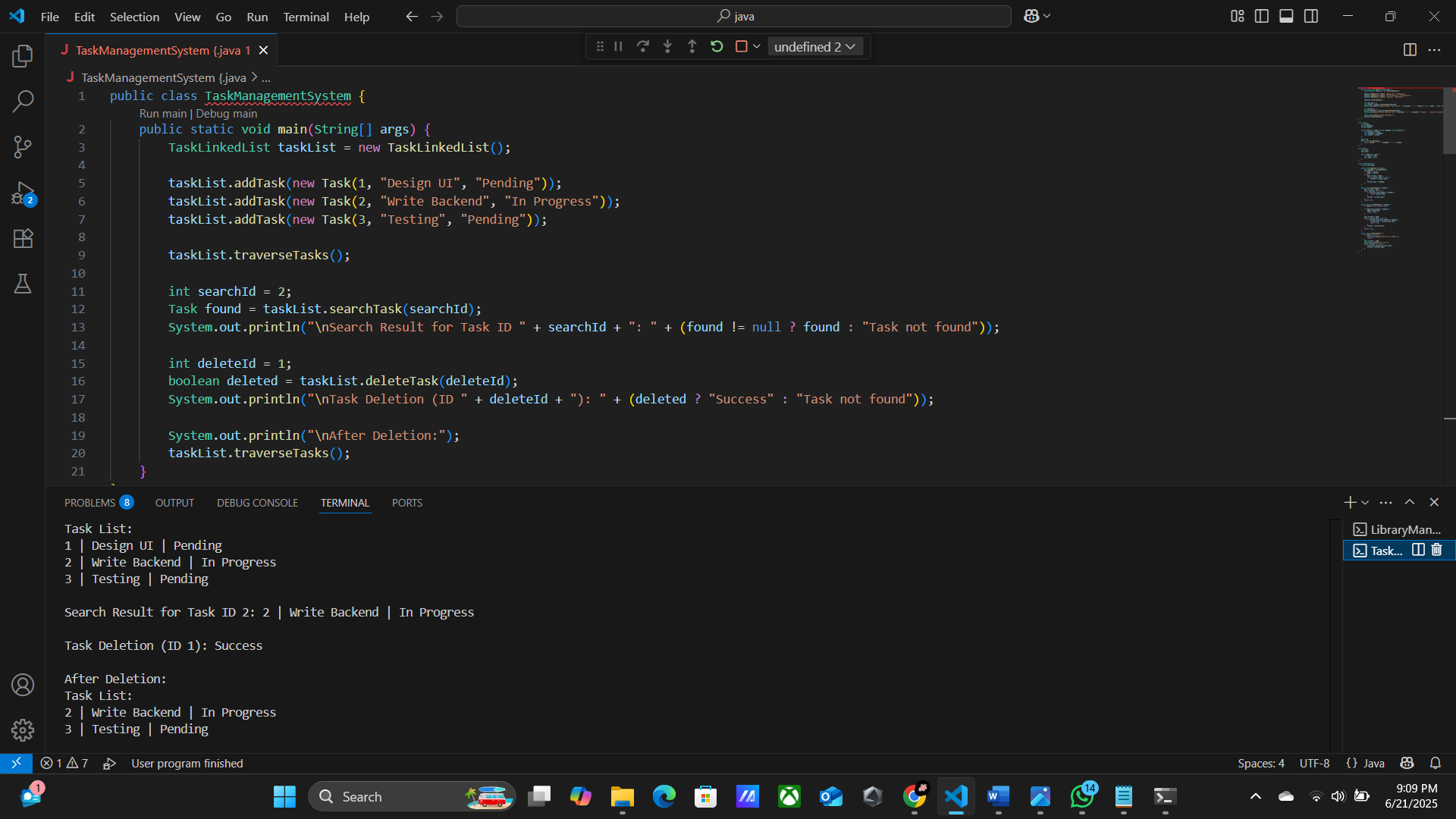
**System.out.println(current.task);**

**current = current.next;**

**}**

**}**

**}**

****

**Exercise 6: Library Management System**

**import java.util.ArrayList;**

**import java.util.Collections;**

**import java.util.Comparator;**

**class Book {**

**private int bookId;**

**private String title;**

**private String author;**

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

**this.bookId = bookId;**

**this.title = title;**

**this.author = author;**

**}**

**public int getBookId() {**

**return bookId;**

**}**

**public String getTitle() {**

**return title;**

**}**

**public String getAuthor() {**

**return author;**

**}**

**@Override**

**public String toString() {**

**return "ID: " + bookId + ", Title: '" + title + "', Author: " + author;**

**}**

**}**

**public class LibraryManagementSystem {**

**private ArrayList<Book> books;**

**public LibraryManagementSystem() {**

**books = new ArrayList<>();**

**}**

**public void addBook(Book book) {**

**books.add(book);**

**}**

**public Book linearSearchByTitle(String title) {**

**for (Book book : books) {**

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

**return book;**

**}**

**}**

**return null;**

**public Book binarySearchByTitle(String title) {**

**Collections.sort(books, Comparator.comparing(Book::getTitle, String.CASE\_INSENSITIVE\_ORDER));**

**int left = 0;**

**int right = books.size() - 1;**

**while (left <= right) {**

**int mid = left + (right - left) / 2;**

**int comparison = books.get(mid).getTitle().compareToIgnoreCase(title);**

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

**return books.get(mid);**

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

**left = mid + 1;**

**} else {**

**right = mid - 1;**

**}**

**}**

**return null;**

**}**

**public void sortBooksByTitle() {**

**Collections.sort(books, Comparator.comparing(Book::getTitle, String.CASE\_INSENSITIVE\_ORDER));**

**}**

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

**LibraryManagementSystem library = new LibraryManagementSystem();**

**library.addBook(new Book(1, "The Great Gatsby", "F. Scott Fitzgerald"));**

**library.addBook(new Book(2, "To Kill a Mockingbird", "Harper Lee"));**

**library.addBook(new Book(3, "1984", "George Orwell"));**

**library.addBook(new Book(4, "Pride and Prejudice", "Jane Austen"));**

**library.addBook(new Book(5, "The Hobbit", "J.R.R. Tolkien"));**

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

**Book book1 = library.linearSearchByTitle("1984");**

**System.out.println(book1 != null ? "Found: " + book1 : "Book not found");**

**Book book2 = library.linearSearchByTitle("Unknown Book");**

**System.out.println(book2 != null ? "Found: " + book2 : "Book not found");**

**System.out.println("\nBinary Search Results:");**

**library.sortBooksByTitle();**

**Book book3 = library.binarySearchByTitle("The Great Gatsby");**

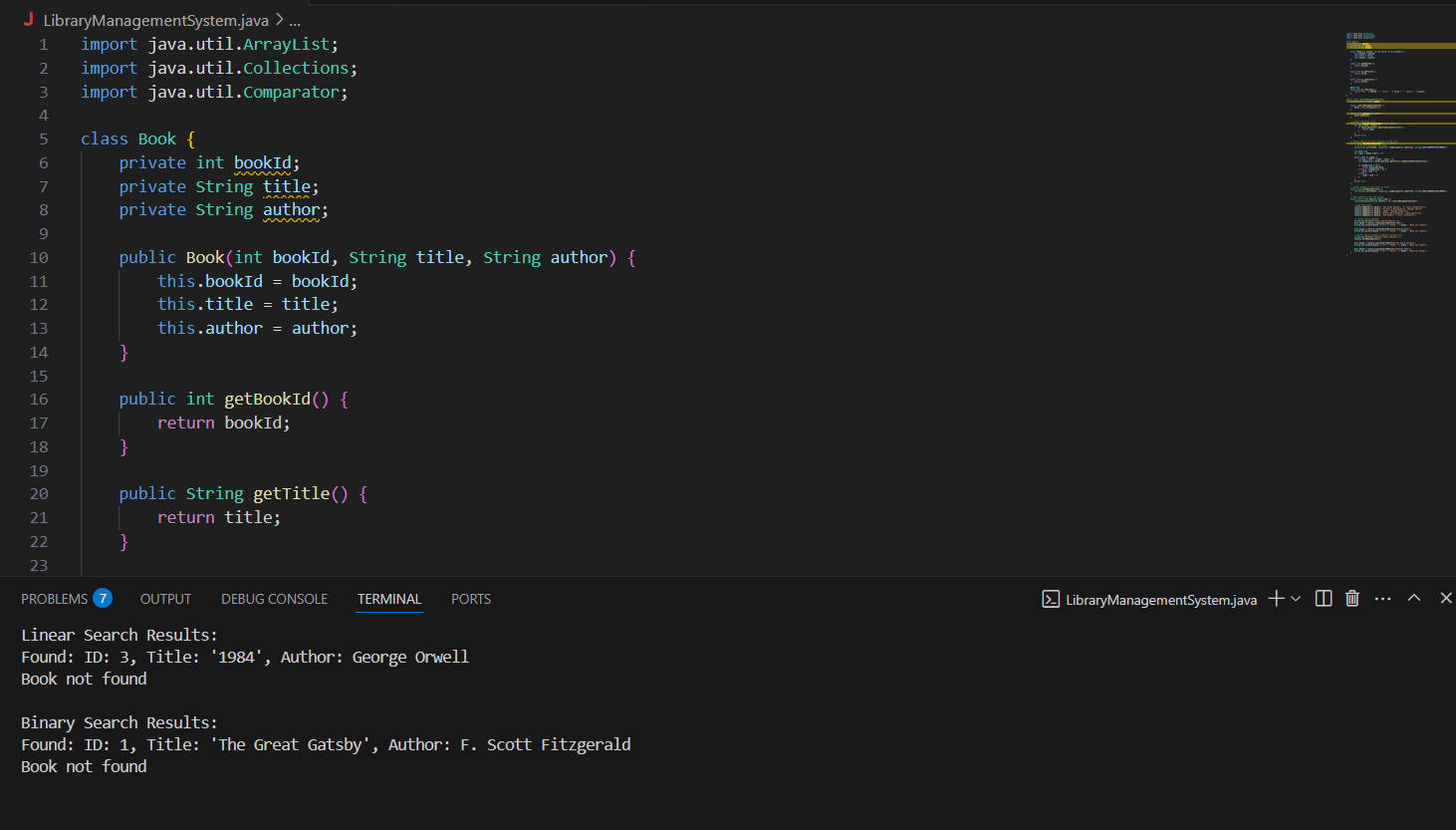
**System.out.println(book3 != null ? "Found: " + book3 : "Book not found");**

**Book book4 = library.binarySearchByTitle("Unknown Book");**

**System.out.println(book4 != null ? "Found: " + book4 : "Book not found");**

**}**

**}**

****

**Exercise 7: Employee Management System**

**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;**

**}**

**}**

**public class EmployeeManagementSystem {**

**private Employee[] employees;**

**private int count;**

**public EmployeeManagementSystem(int capacity) {**

**employees = new Employee[capacity];**

**count = 0;**

**}**

**public void addEmployee(Employee emp) {**

**if (count < employees.length) {**

**employees[count++] = emp;**

**} else {**

**System.out.println("Employee list is full.");**

**}**

**}**

**public Employee searchEmployee(int id) {**

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

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

**return employees[i];**

**}**

**}**

**return null;**

**}**

**public void displayAllEmployees() {**

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

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

**}**

**}**

**public void deleteEmployee(int id) {**

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

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

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

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

**}**

**employees[--count] = null; // remove last reference**

**System.out.println("Employee with ID " + id + " deleted.");**

**return;**

**}**

**}**

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

**}**

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

**EmployeeManagementSystem ems = new EmployeeManagementSystem(5);**

**ems.addEmployee(new Employee(101, "Alice", "Developer", 60000));**

**ems.addEmployee(new Employee(102, "Bob", "Manager", 80000));**

**ems.addEmployee(new Employee(103, "Charlie", "Analyst", 55000));**

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

**ems.displayAllEmployees();**

**System.out.println("\nSearch for ID 102:");**

**Employee found = ems.searchEmployee(102);**

**System.out.println(found != null ? found : "Not found");**

**System.out.println("\nDeleting ID 101:");**

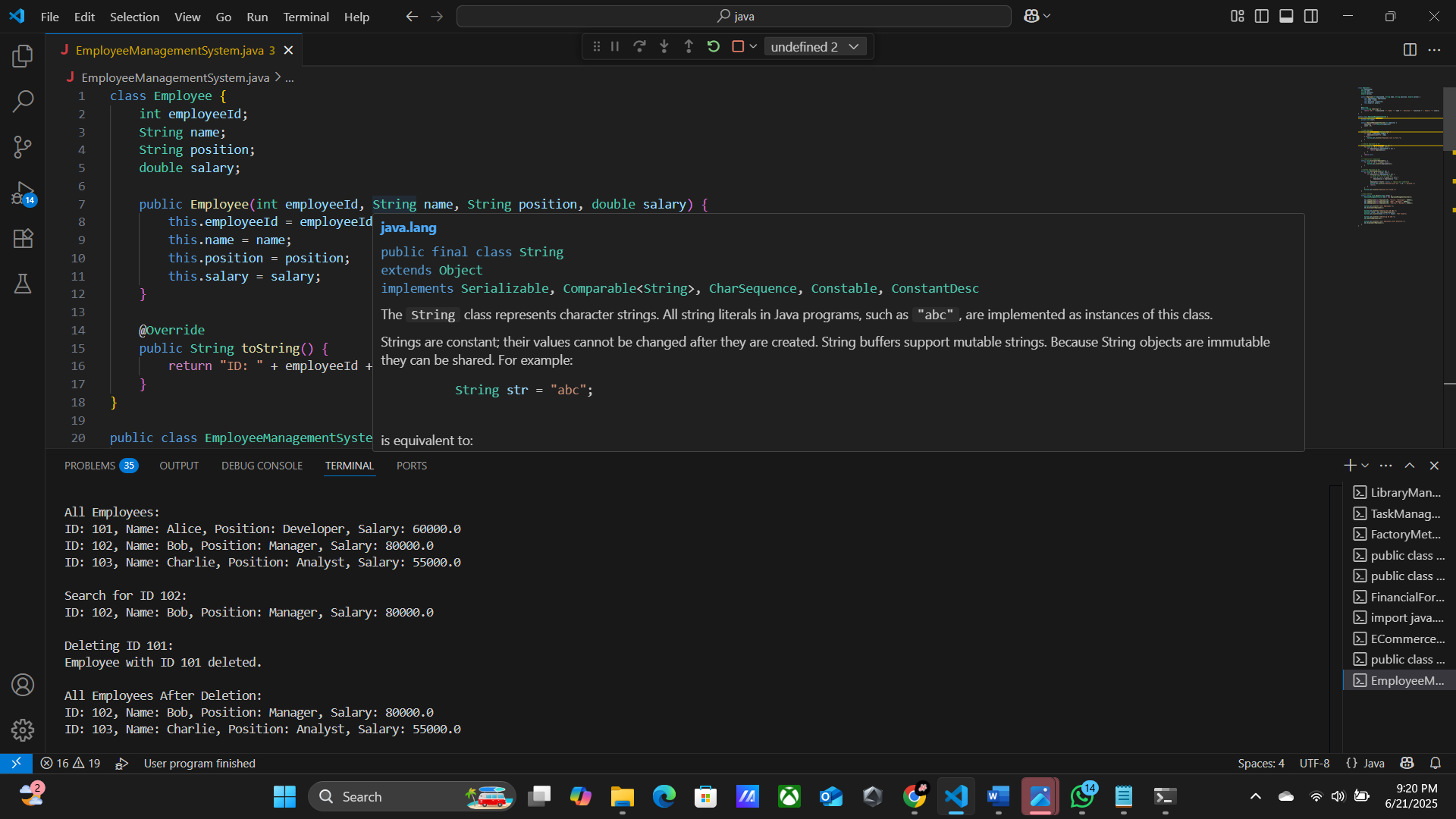
**ems.deleteEmployee(101);**

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

**ems.displayAllEmployees();**

**}**

**}**

****

**Exercise 8: Sorting Customer Orders**

**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: $" + totalPrice;**

**}**

**}**

**public class OrderSorting {**

**public static void bubbleSort(Order[] orders) {**

**int n = orders.length;**

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

**boolean swapped = false;**

**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;**

**swapped = true;**

**}**

**}**

**if (!swapped) break;**

**}**

**}**

**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++;**

**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 displayOrders(Order[] orders) {**

**for (Order order : orders) {**

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

**}**

**}**

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

**Order[] orders = {**

**new Order(101, "Alice", 250.75),**

**new Order(102, "Bob", 99.99),**

**new Order(103, "Charlie", 350.20),**

**new Order(104, "David", 150.00),**

**new Order(105, "Eve", 120.45)**

**};**

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

**displayOrders(orders);**

**bubbleSort(orders);**

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

**displayOrders(orders);**

**orders = new Order[]{**

**new Order(101, "Alice", 250.75),**

**new Order(102, "Bob", 99.99),**

**new Order(103, "Charlie", 350.20),**

**new Order(104, "David", 150.00),**

**new Order(105, "Eve", 120.45)**

**};**

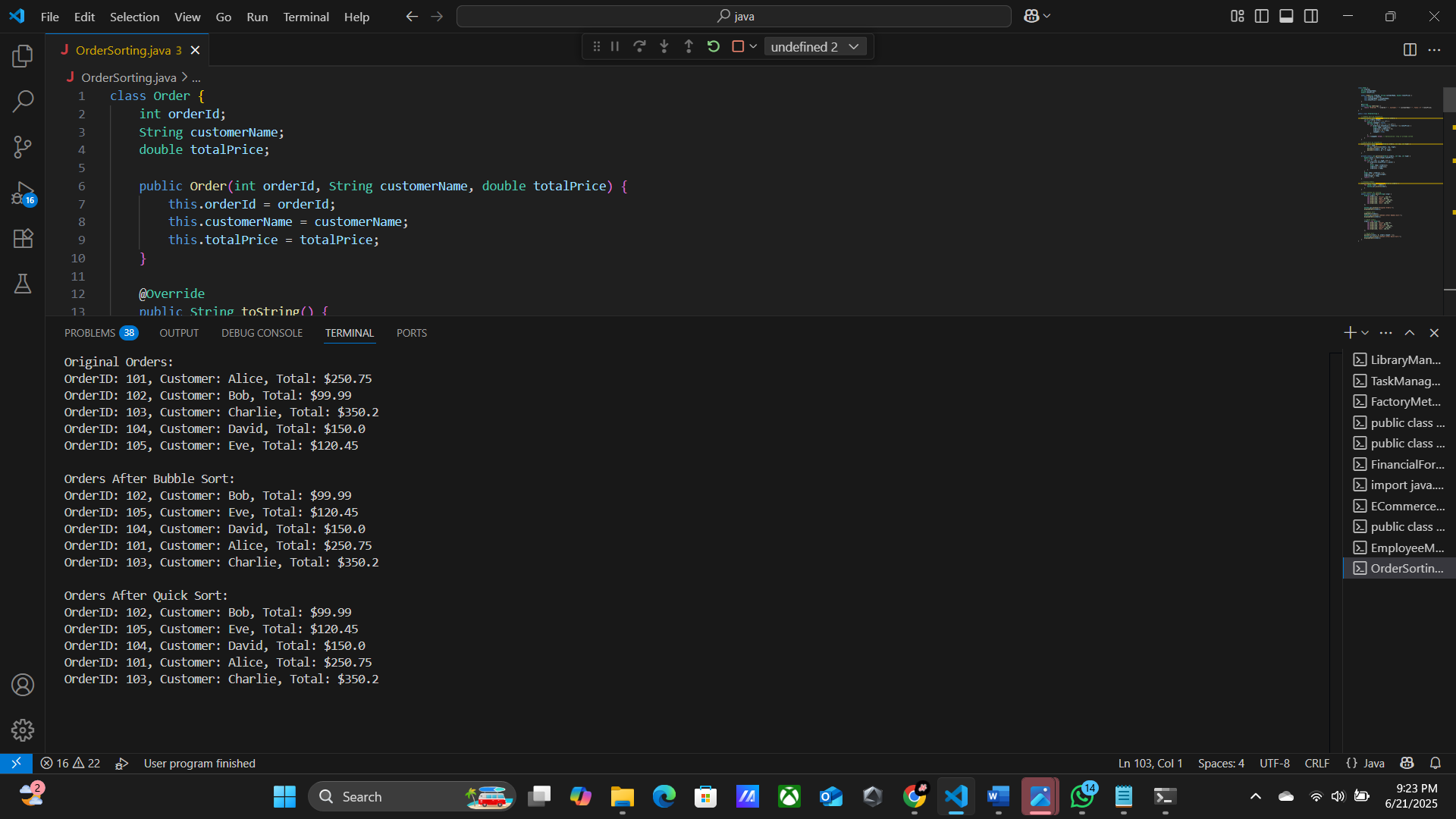
**quickSort(orders, 0, orders.length - 1);**

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

**displayOrders(orders);**

**}**

**}**

****

**Exercise 9: Implementing the Builder Pattern**

**public class BuilderPatternExample {**

**public static class Computer {**

**private final String cpu;**

**private final String ram;**

**private final String storage;**

**private final String gpu;**

**private final String motherboard;**

**private final String powerSupply;**

**private Computer(Builder builder) {**

**this.cpu = builder.cpu;**

**this.ram = builder.ram;**

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

**this.gpu = builder.gpu;**

**this.motherboard = builder.motherboard;**

**this.powerSupply = builder.powerSupply;**

**}**

**public static class Builder {**

**private final String cpu;**

**private final String ram;**

**private String storage;**

**private String gpu;**

**private String motherboard;**

**private String powerSupply;**

**public Builder(String cpu, String ram) {**

**this.cpu = cpu;**

**this.ram = ram;**

**}**

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

**this.storage = storage;**

**return this;**

**}**

**public Builder setGPU(String gpu) {**

**this.gpu = gpu;**

**return this;**

**}**

**public Builder setMotherboard(String motherboard) {**

**this.motherboard = motherboard;**

**return this;**

**}**

**public Builder setPowerSupply(String powerSupply) {**

**this.powerSupply = powerSupply;**

**return this;**

**}**

**public Computer build() {**

**return new Computer(this);**

**}**

**}**

**@Override**

**public String toString() {**

**return "Computer Configuration:\n" +**

**"CPU: " + cpu + "\n" +**

**"RAM: " + ram + "\n" +**

**"Storage: " + (storage != null ? storage : "Not Included") + "\n" +**

**"GPU: " + (gpu != null ? gpu : "Not Included") + "\n" +**

**"Motherboard: " + (motherboard != null ? motherboard : "Not Included") + "\n" +**

**"Power Supply: " + (powerSupply != null ? powerSupply : "Not Included");**

**}**

**}**

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

**Computer basicComputer = new Computer.Builder("Intel i3", "4GB").build();**

**Computer gamingComputer = new Computer.Builder("Intel i9", "32GB")**

**.setGPU("NVIDIA RTX 4090")**

**.setStorage("2TB SSD")**

**.setMotherboard("ASUS ROG STRIX")**

**.setPowerSupply("850W Gold")**

**.build();**

**Computer officeComputer = new Computer.Builder("AMD Ryzen 5", "16GB")**

**.setStorage("512GB SSD")**

**.setMotherboard("Gigabyte B550M")**

**.build();**

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

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

**System.out.println("\n=== Gaming Computer ===");**

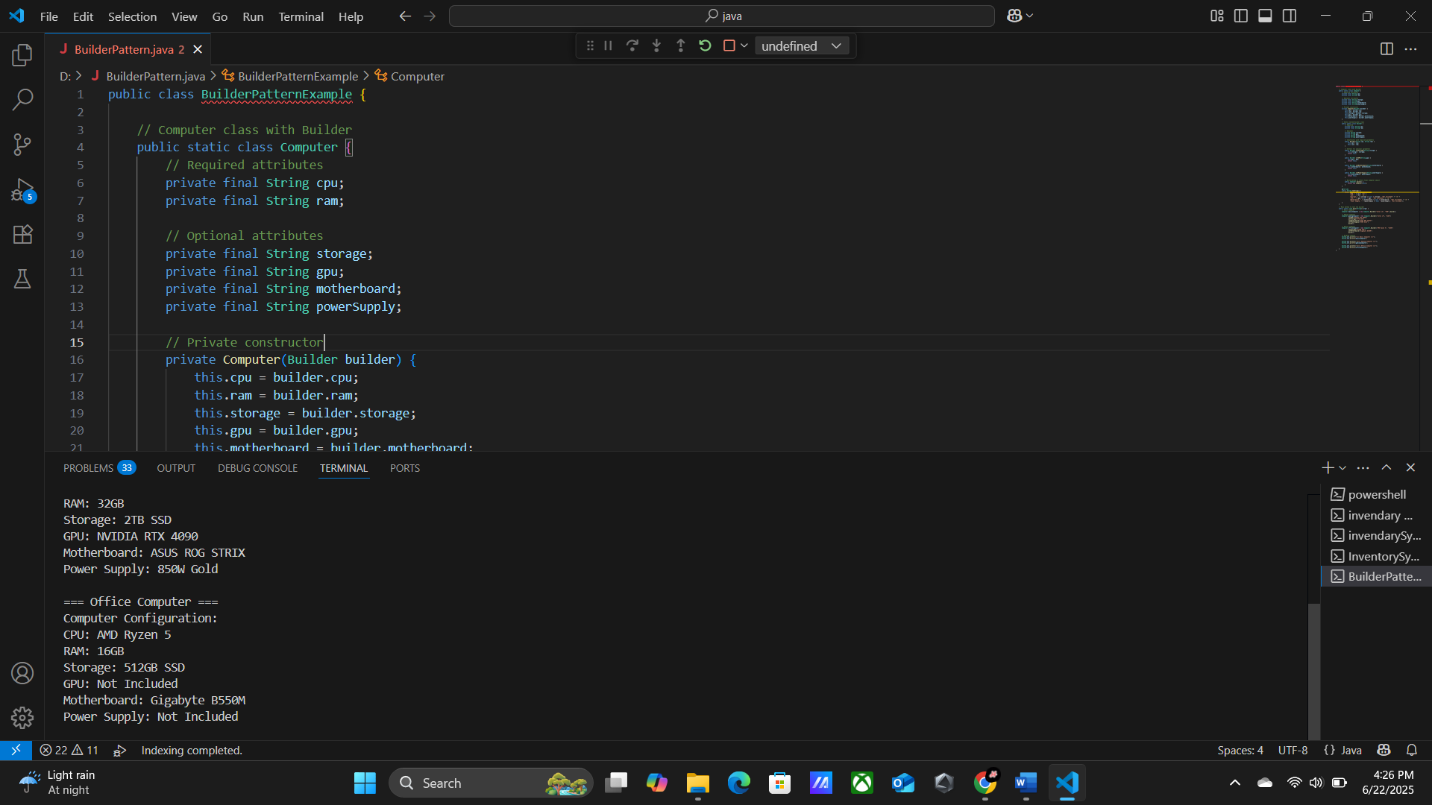
**System.out.println(gamingComputer);**

**System.out.println("\n=== Office Computer ===");**

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

**}**

**}**

****