**Exercise 1: Inventory Management System**

**Scenario:**

You are developing an inventory management system for a warehouse. Efficient data storage and retrieval are crucial.

**Solution**

**Product.java**

public class Product {

String productId;

String productName;

int quantity;

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 toString() {

return "ID: " + productId + ", Name: " + productName + ", Quantity: " + quantity + ", Price: " + price;

}

}

**Inventory.java**

import java.util.HashMap;

public class Inventory {

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

public void addProduct(Product product) {

products.put(product.productId, product);

System.out.println("Added: " + product);

}

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

Product p = products.get(productId);

if (p != null) {

p.quantity = quantity;

p.price = price;

System.out.println("Updated: " + p);

} else {

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

}

}

public void deleteProduct(String productId) {

Product removed = products.remove(productId);

if (removed != null) {

System.out.println("Deleted: " + removed);

} else {

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

}

}

public void listAll() {

System.out.println("---- Inventory ----");

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

System.out.println(p);

}

}

}

**InventoryApp.java**

public class InventoryApp {

public static void main(String[] args) {

Inventory inventory = new Inventory();

Product p1 = new Product("P001", "Laptop", 10, 1500.0);

Product p2 = new Product("P002", "Mouse", 50, 25.0);

Product p3 = new Product("P003", "Keyboard", 30, 45.0);

inventory.addProduct(p1);

inventory.addProduct(p2);

inventory.addProduct(p3);

inventory.listAll();

inventory.updateProduct("P002", 60, 27.0);

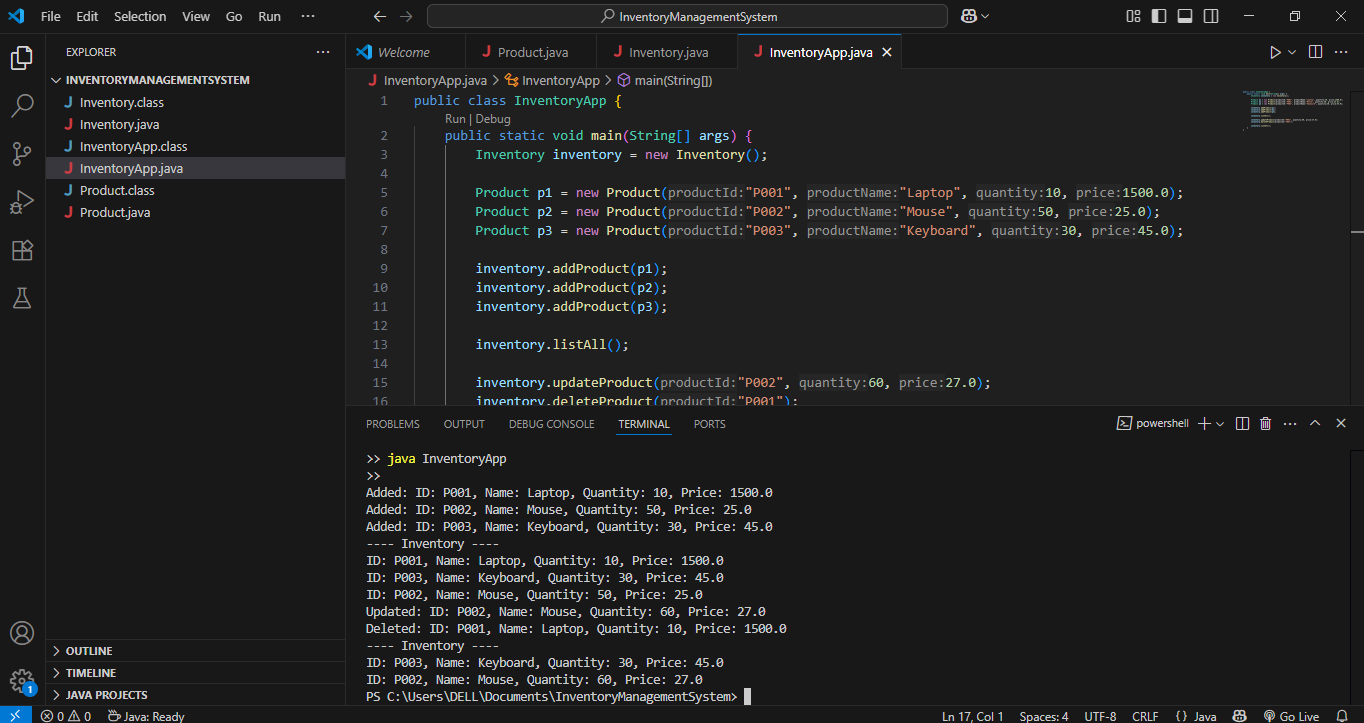
inventory.deleteProduct("P001");

inventory.listAll();

}

}

**Output**



**Exercise 2: E-commerce Platform Search Function**

**Scenario:**

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

**Solution**

**Product.java**

public class Product {

String productId;

String productName;

String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

public String toString() {

return "ID: " + productId + ", Name: " + productName + ", Category: " + category;

}

}

**SearchFunctions.java**

import java.util.Arrays;

import java.util.Comparator;

public class SearchFunctions {

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

for (Product p : products) {

if (p.productName.equalsIgnoreCase(targetName)) {

return p;

}

}

return null;

}

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

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

int left = 0;

int right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

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

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

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

else right = mid - 1;

}

return null;

}

}

**SearchTest.java**

public class SearchTest {

public static void main(String[] args) {

Product[] products = {

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

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

new Product("P003", "Smartphone", "Electronics"),

new Product("P004", "Watch", "Accessories"),

new Product("P005", "Bag", "Fashion")

};

System.out.println("Searching for 'Shoes' using Linear Search:");

Product result1 = SearchFunctions.linearSearch(products, "Shoes");

System.out.println(result1 != null ? result1 : "Not Found");

System.out.println("\nSearching for 'Watch' using Binary Search:");

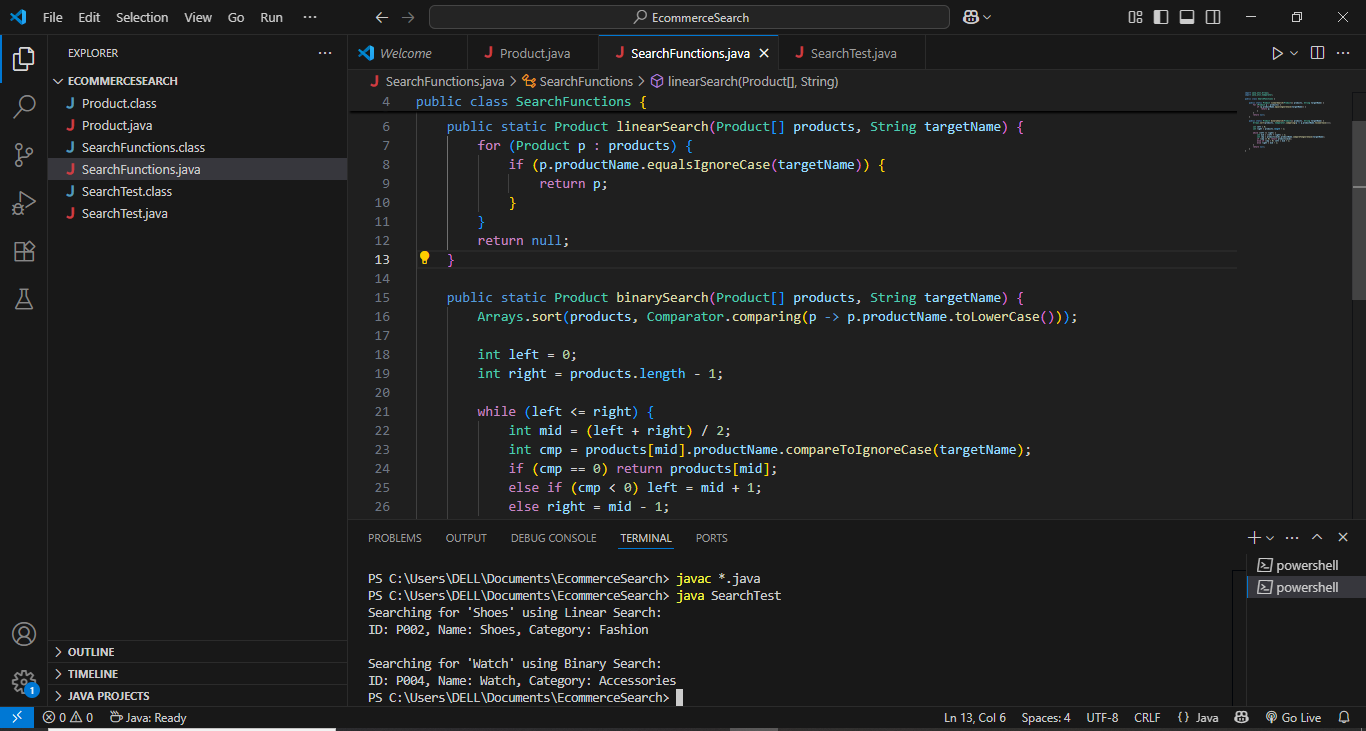
Product result2 = SearchFunctions.binarySearch(products, "Watch");

System.out.println(result2 != null ? result2 : "Not Found");

}

}

**Output**



**Exercise 3: Sorting Customer Orders**

**Scenario:**

You are tasked with sorting customer orders by their total price on an e-commerce platform. This helps in prioritizing high-value orders.

Solution

**Order.java**

public class Order {

String orderId;

String customerName;

double totalPrice;

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

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

public String toString() {

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

}

}

**OrderSorter.java**

public class OrderSorter {

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);

}

}

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 printOrders(Order[] orders) {

for (Order o : orders) {

System.out.println(o);

}

}

}

**SortTest.java**

public class SortTest {

public static void main(String[] args) {

Order[] orders = {

new Order("O101", "Alice", 250.75),

new Order("O102", "Bob", 520.00),

new Order("O103", "Charlie", 150.30),

new Order("O104", "Diana", 399.99)

};

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

OrderSorter.printOrders(orders);

// Bubble Sort

Order[] bubbleSorted = orders.clone();

OrderSorter.bubbleSort(bubbleSorted);

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

OrderSorter.printOrders(bubbleSorted);

// Quick Sort

Order[] quickSorted = orders.clone();

OrderSorter.quickSort(quickSorted, 0, quickSorted.length - 1);

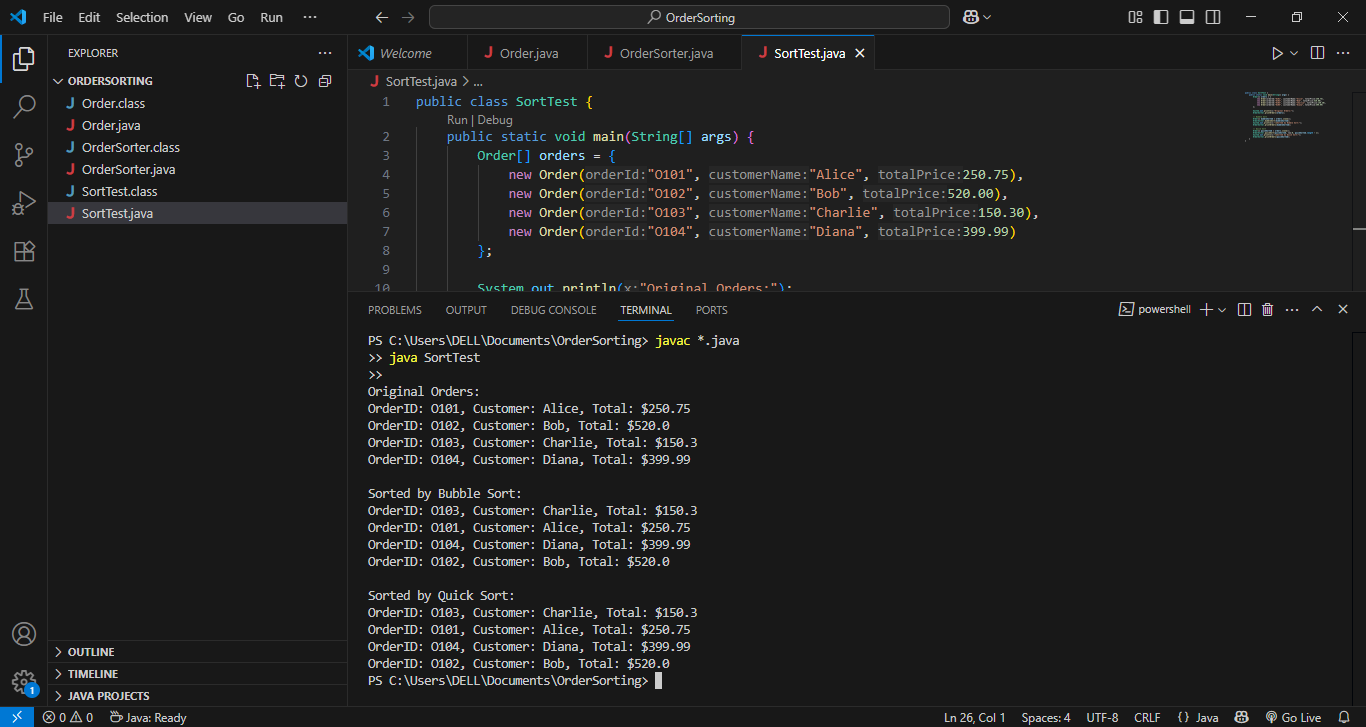
System.out.println("\nSorted by Quick Sort:");

OrderSorter.printOrders(quickSorted);

}

}

**Output**



**Exercise 4: Employee Management System**

**Scenario:**

You are developing an employee management system for a company. Efficiently managing employee records is crucial.

**Solution**

**Employee.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;

}

public String toString() {

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

}

}

**EmployeeManager.java**

public 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;

} else {

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

}

}

// Search Employee by ID

public Employee searchEmployee(int id) {

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

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

return employees[i];

}

}

return null;

}

// Traverse

public void listEmployees() {

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

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

}

}

// Delete Employee by ID

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;

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

return;

}

}

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

}

}

**EmployeeTest.java**

public class EmployeeTest {

public static void main(String[] args) {

EmployeeManager manager = new EmployeeManager(5);

manager.addEmployee(new Employee(101, "Alice", "Manager", 80000));

manager.addEmployee(new Employee(102, "Bob", "Developer", 60000));

manager.addEmployee(new Employee(103, "Charlie", "Tester", 50000));

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

manager.listEmployees();

System.out.println("\nSearching for Employee with ID 102:");

Employee found = manager.searchEmployee(102);

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

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

manager.deleteEmployee(101);

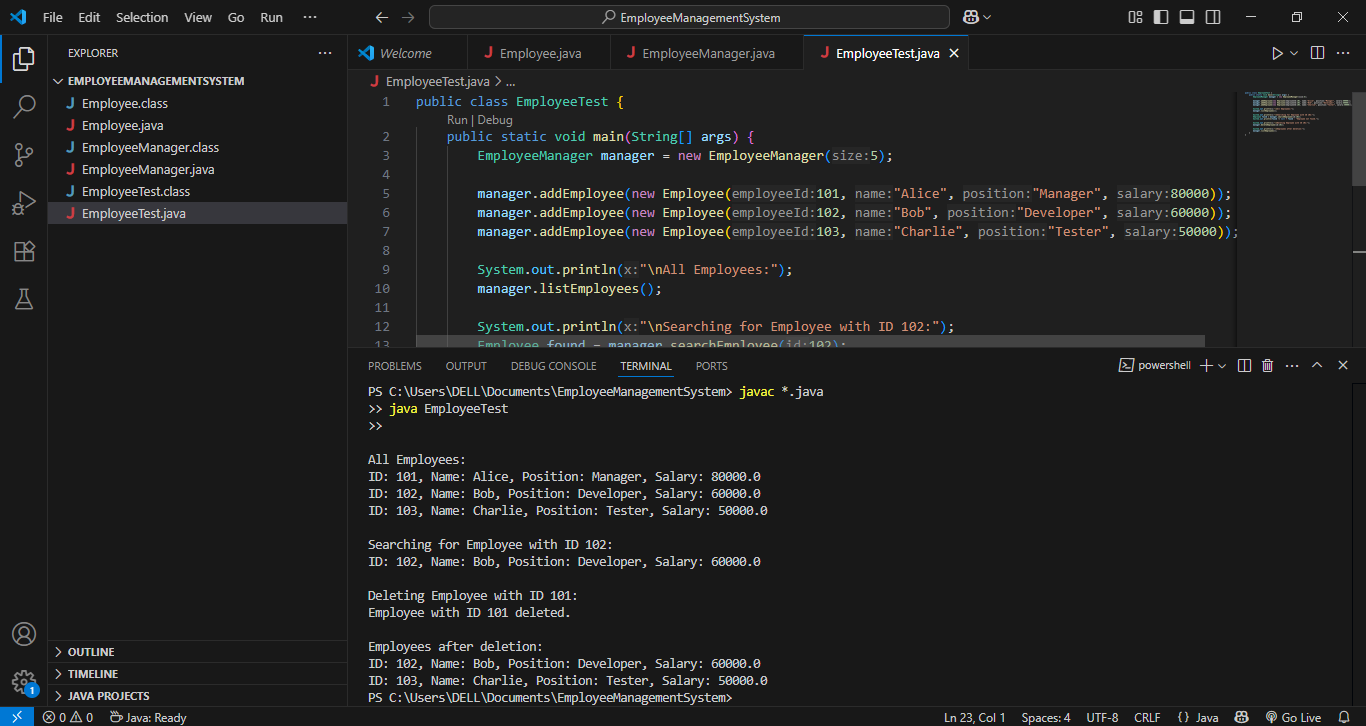
System.out.println("\nEmployees after deletion:");

manager.listEmployees();

}

}

**Output**



**Exercise 5: Task Management System**

**Scenario:**

You are developing a task management system where tasks need to be added, deleted, and traversed efficiently.

Solution

**Task.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;

}

public String toString() {

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

}

}

**TaskNode.java**

public class TaskNode {

Task task;

TaskNode next;

public TaskNode(Task task) {

this.task = task;

this.next = null;

}

}

**TaskLinkedList.java**

public class TaskLinkedList {

private TaskNode head;

// Add task to the end

public void addTask(Task task) {

TaskNode newNode = new TaskNode(task);

if (head == null) {

head = newNode;

} else {

TaskNode current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

}

}

// Search task by ID

public Task searchTask(int taskId) {

TaskNode current = head;

while (current != null) {

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

return current.task;

}

current = current.next;

}

return null;

}

// Traverse tasks

public void listTasks() {

TaskNode current = head;

while (current != null) {

System.out.println(current.task);

current = current.next;

}

}

// Delete task by ID

public void deleteTask(int taskId) {

if (head == null) return;

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

head = head.next;

System.out.println("Task with ID " + taskId + " deleted.");

return;

}

TaskNode current = head;

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

current = current.next;

}

if (current.next != null) {

current.next = current.next.next;

System.out.println("Task with ID " + taskId + " deleted.");

} else {

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

}

}

}

**TaskTest.java**

public class TaskTest {

public static void main(String[] args) {

TaskLinkedList taskList = new TaskLinkedList();

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

taskList.addTask(new Task(2, "Build API", "In Progress"));

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

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

taskList.listTasks();

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

Task found = taskList.searchTask(2);

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

System.out.println("\nDelete Task ID 1:");

taskList.deleteTask(1);

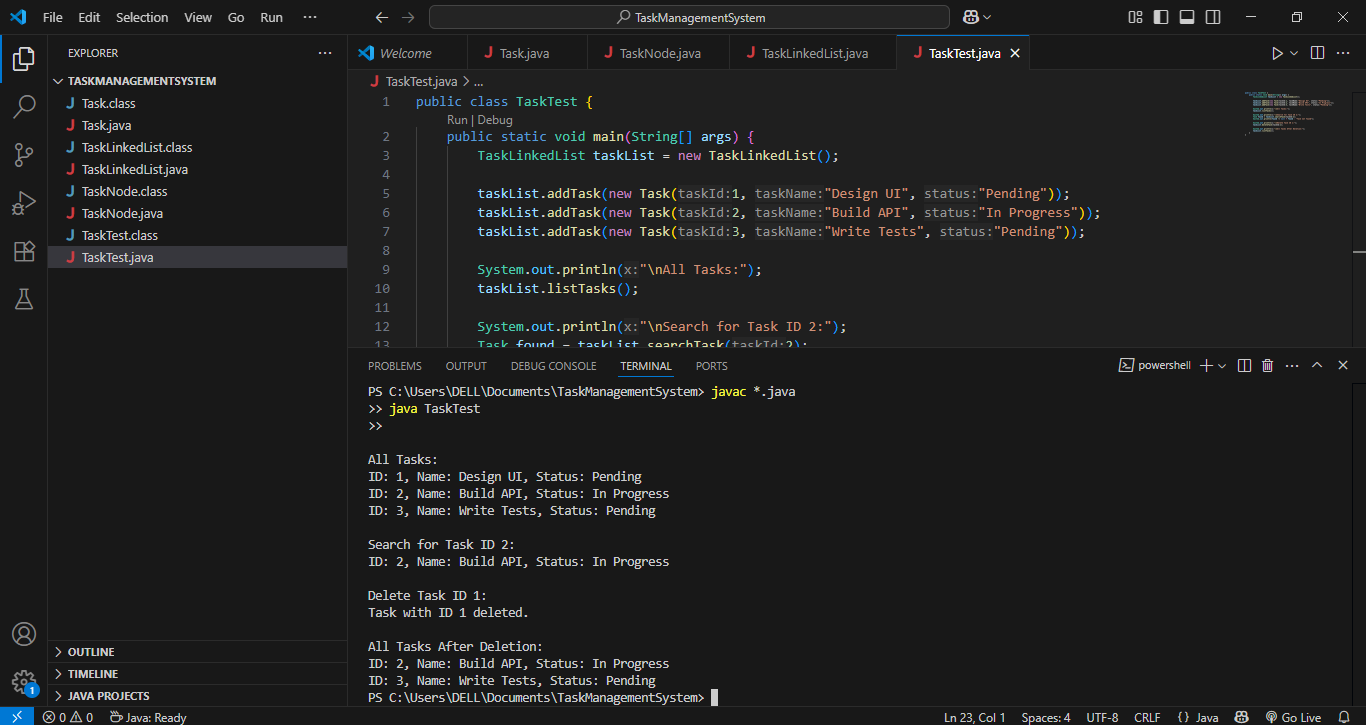
System.out.println("\nAll Tasks After Deletion:");

taskList.listTasks();

}

}

**Output**



**Exercise 6: Library Management System**

**Scenario:**

You are developing a library management system where users can search for books by title or author.

Solution

**Book.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;

}

public String toString() {

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

}

}

**Library.java**

import java.util.Arrays;

import java.util.Comparator;

public class Library {

Book[] books;

int count;

public Library(int size) {

books = new Book[size];

count = 0;

}

public void addBook(Book book) {

if (count < books.length) {

books[count++] = book;

}

}

// Linear search

public Book linearSearchByTitle(String title) {

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

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

return books[i];

}

}

return null;

}

// Binary search (requires sorted array)

public Book binarySearchByTitle(String title) {

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

int left = 0, right = count - 1;

while (left <= right) {

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

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

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

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

else right = mid - 1;

}

return null;

}

public void printAllBooks() {

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

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

}

}

}

**LibraryTest.java**

public class LibraryTest {

public static void main(String[] args) {

Library library = new Library(5);

library.addBook(new Book(1, "The Hobbit", "J.R.R. Tolkien"));

library.addBook(new Book(2, "1984", "George Orwell"));

library.addBook(new Book(3, "To Kill a Mockingbird", "Harper Lee"));

library.addBook(new Book(4, "The Catcher in the Rye", "J.D. Salinger"));

library.addBook(new Book(5, "Pride and Prejudice", "Jane Austen"));

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

library.printAllBooks();

System.out.println("\n🔍 Linear Search for '1984':");

Book found1 = library.linearSearchByTitle("1984");

System.out.println(found1 != null ? found1 : "Book not found");

System.out.println("\n🔎 Binary Search for 'The Hobbit':");

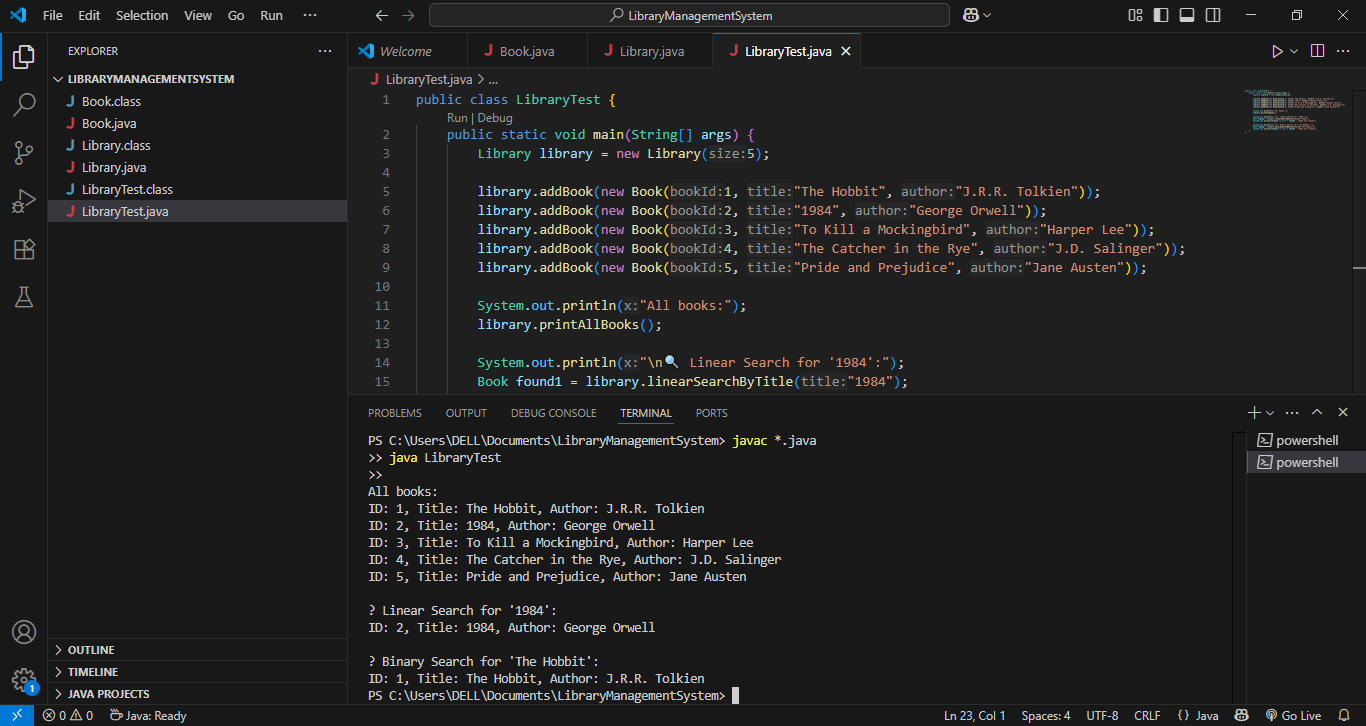
Book found2 = library.binarySearchByTitle("The Hobbit");

System.out.println(found2 != null ? found2 : "Book not found");

}

}

**Output**



**Exercise 7: Financial Forecasting**

**Scenario:**

You are developing a financial forecasting tool that predicts future values based on past dat

Solution

**Forecast.java**

public class Forecast {

// Recursive method to calculate future value

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

if (years == 0) return presentValue;

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

}

}

**ForecastTest.java**

public class ForecastTest {

public static void main(String[] args) {

double presentValue = 1000.0;

double growthRate = 0.10; // 10%

int years = 5;

double futureValue = Forecast.predictFutureValue(presentValue, growthRate, years);

System.out.printf("Predicted future value after %d years: ₹%.2f\n", years, futureValue);

}

}

**Output**

