**Atchaya V - 6418561**

# **Week 1**

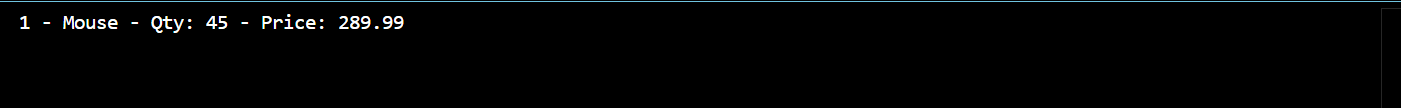
**Data Structures and Algorithms**

**Exercise 1: Inventory Management System**

**PROGRAM:**

1. import java.util.\*;
2. class Product {
3. int productId;
4. String productName;
5. int quantity;
6. double price;
7. public Product(int productId, String productName, int quantity, double price) {
8. this.productId = productId;
9. this.productName = productName;
10. this.quantity = quantity;
11. this.price = price;
12. }
13. }
14. class Inventory {
15. Map<Integer, Product> inventory = new HashMap<>();
16. void addProduct(Product product) {
17. inventory.put(product.productId, product);
18. }
19. void updateProduct(int id, int quantity, double price) {
20. Product p = inventory.get(id);
21. if (p != null) {
22. p.quantity = quantity;
23. p.price = price;
24. }
25. }
26. void deleteProduct(int id) {
27. inventory.remove(id);
28. }
29. void displayInventory() {
30. for (Product p : inventory.values()) {
31. System.out.println(p.productId + " - " + p.productName + " - Qty: " + p.quantity + " - Price: " + p.price);
32. }
33. }
34. }
35. public class InventoryManagement {
36. public static void main(String[] args) {
37. Inventory inv = new Inventory();
38. inv.addProduct(new Product(1, "Mouse", 50, 299.99));
39. inv.addProduct(new Product(2, "Keyboard", 30, 499.50));
40. inv.updateProduct(1, 45, 289.99);
41. inv.deleteProduct(2);
42. inv.displayInventory();
43. }
44. }

**OUTPUT:**

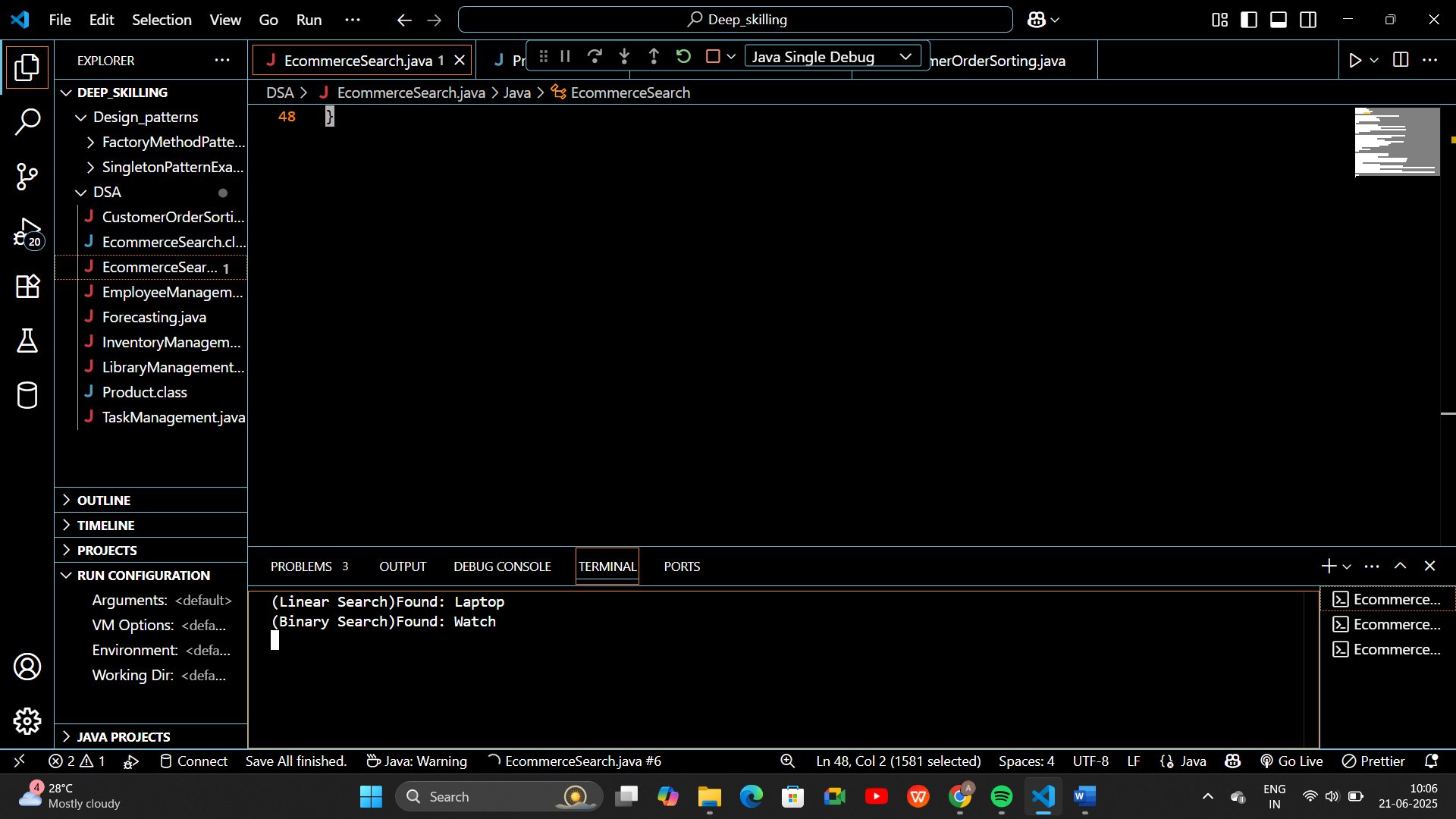
****

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

**PROGRAM:**

1. class Product {
2. int productId;
3. String productName;
4. String category;
5. public Product(int id, String name, String category) {
6. this.productId = id;
7. this.productName = name;
8. this.category = category;
9. }
10. }
11. public class EcommerceSearch {
12. static Product linearSearch(Product[] products, String name) {
13. for (Product p : products)
14. if (p.productName.toLowerCase().equals(name)) return p;
15. return null;
16. }
17. static Product binarySearch(Product[] products, String name) {
18. int low = 0, high = products.length - 1;
19. while (low <= high) {
20. int mid = (low + high) / 2;
21. int cmp = products[mid].productName.compareTo(name);
22. if (cmp == 0) return products[mid];
23. else if (cmp < 0) low = mid + 1;
24. else high = mid - 1;
25. }
26. return null;
27. }
28. public static void main(String[] args) {
29. //Scanner sc=new Scanner(System.in);
30. Product[] products = {
31. new Product(1, "Laptop", "Electronics"),
32. new Product(2, "Phone", "Electronics"),
33. new Product(3, "Watch", "Accessories")
34. };
35. //  String name1=sc.next();
36. Product found1 = linearSearch(products, "laptop");
37. System.out.println(found1 != null ? "(Linear Search)Found: " + found1.productName : "Not Found");
38. //  String name2=sc.next();
39. Product found2 = binarySearch(products, "Watch");
40. System.out.println(found2 != null ? "(Binary Search)Found: " + found2.productName : "Not Found");
41. }
42. }

**OUTPUT:**

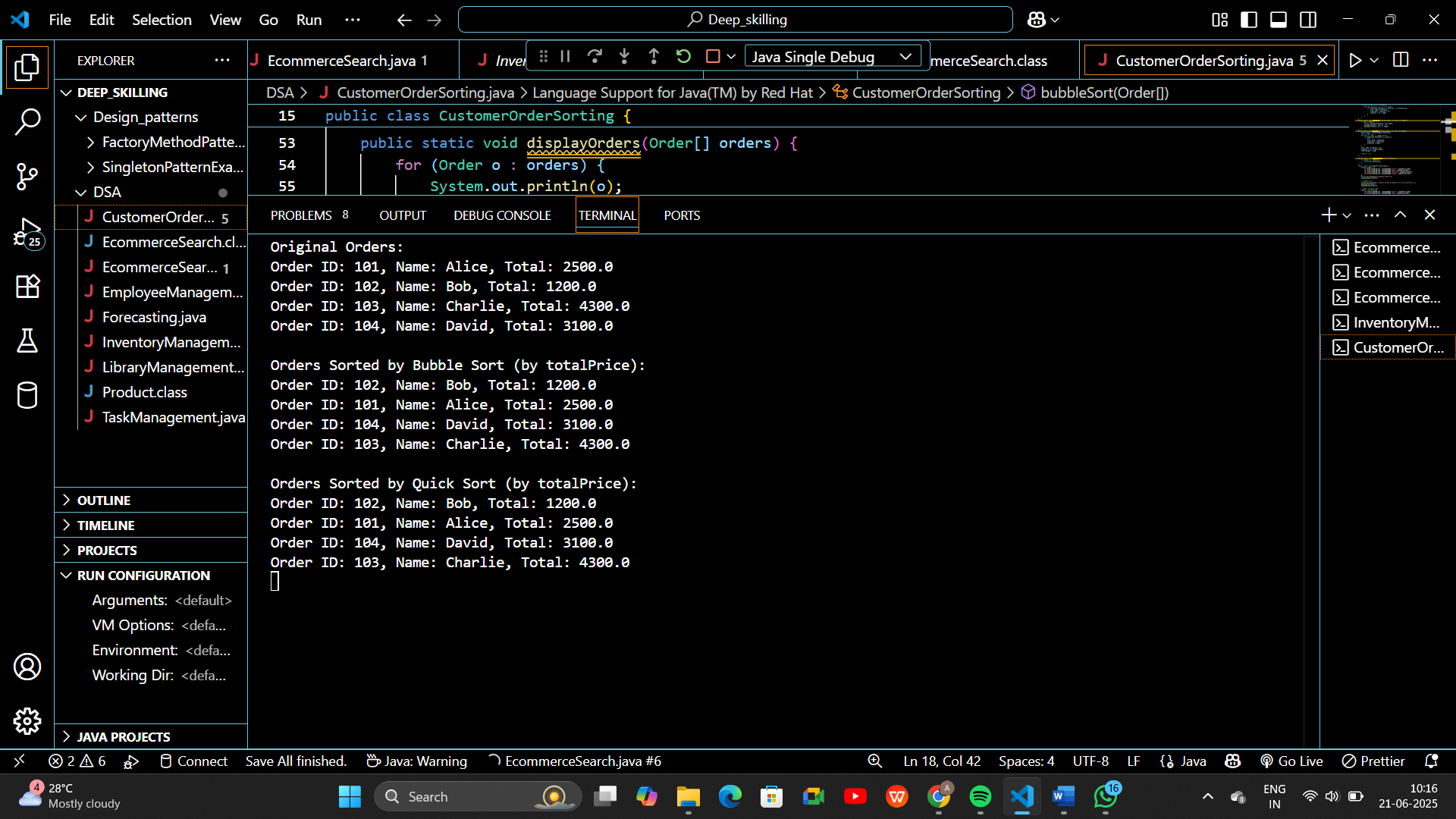


**Exercise 3: Sorting Customer Orders**

**PROGRAM:**

1. class Order {
2. int orderId;
3. String customerName;
4. double totalPrice;
5. public Order(int orderId, String customerName, double totalPrice) {
6. this.orderId = orderId;
7. this.customerName = customerName;
8. this.totalPrice = totalPrice;
9. }
10. public String toString() {
11. return "Order ID: " + orderId + ", Name: " + customerName + ", Total: " + totalPrice;
12. }
13. }
14. public class CustomerOrderSorting {
15. public static void bubbleSort(Order[] orders) {
16. int n = orders.length;
17. for (int i = 0; i < n - 1; i++) {
18. for (int j = 0; j < n - 1 - i; j++) {
19. if (orders[j].totalPrice > orders[j + 1].totalPrice) {
20. Order temp = orders[j];
21. orders[j] = orders[j + 1];
22. orders[j + 1] = temp;
23. }
24. }
25. }
26. }
27. public static void quickSort(Order[] orders, int low, int high) {
28. if (low < high) {
29. int pi = partition(orders, low, high);
30. quickSort(orders, low, pi - 1);
31. quickSort(orders, pi + 1, high);
32. }
33. }
34. public static int partition(Order[] orders, int low, int high) {
35. double pivot = orders[high].totalPrice;
36. int i = low - 1;
37. for (int j = low; j < high; j++) {
38. if (orders[j].totalPrice <= pivot) {
39. i++;
40. Order temp = orders[i];
41. orders[i] = orders[j];
42. orders[j] = temp;
43. }
44. }
45. Order temp = orders[i + 1];
46. orders[i + 1] = orders[high];
47. orders[high] = temp;
48. return i + 1;
49. }
50. public static void displayOrders(Order[] orders) {
51. for (Order o : orders) {
52. System.out.println(o);
53. }
54. }
55. public static void main(String[] args) {
56. Order[] orders = {
57. new Order(101, "Alice", 2500),
58. new Order(102, "Bob", 1200),
59. new Order(103, "Charlie", 4300),
60. new Order(104, "David", 3100)
61. };
62. System.out.println("Original Orders:");
63. displayOrders(orders);
64. // Bubble Sort
65. System.out.println("\nOrders Sorted by Bubble Sort (by totalPrice):");
66. bubbleSort(orders);
67. displayOrders(orders);
68. // Reset original order
69. orders = new Order[]{
70. new Order(101, "Alice", 2500),
71. new Order(102, "Bob", 1200),
72. new Order(103, "Charlie", 4300),
73. new Order(104, "David", 3100)
74. };
75. // Quick Sort
76. System.out.println("\nOrders Sorted by Quick Sort (by totalPrice):");
77. quickSort(orders, 0, orders.length - 1);
78. displayOrders(orders);
79. }
80. }

**OUTPUT:**

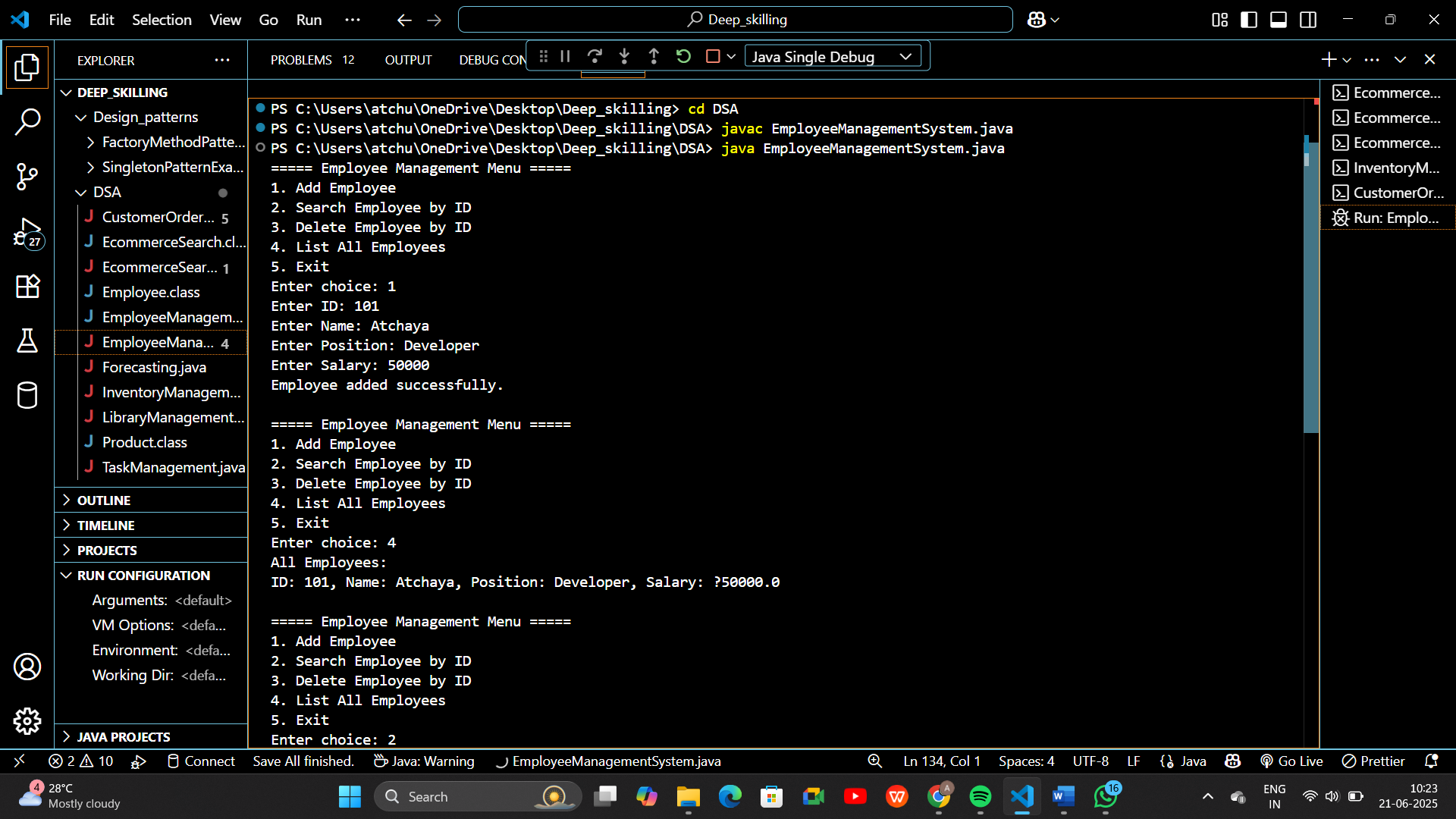


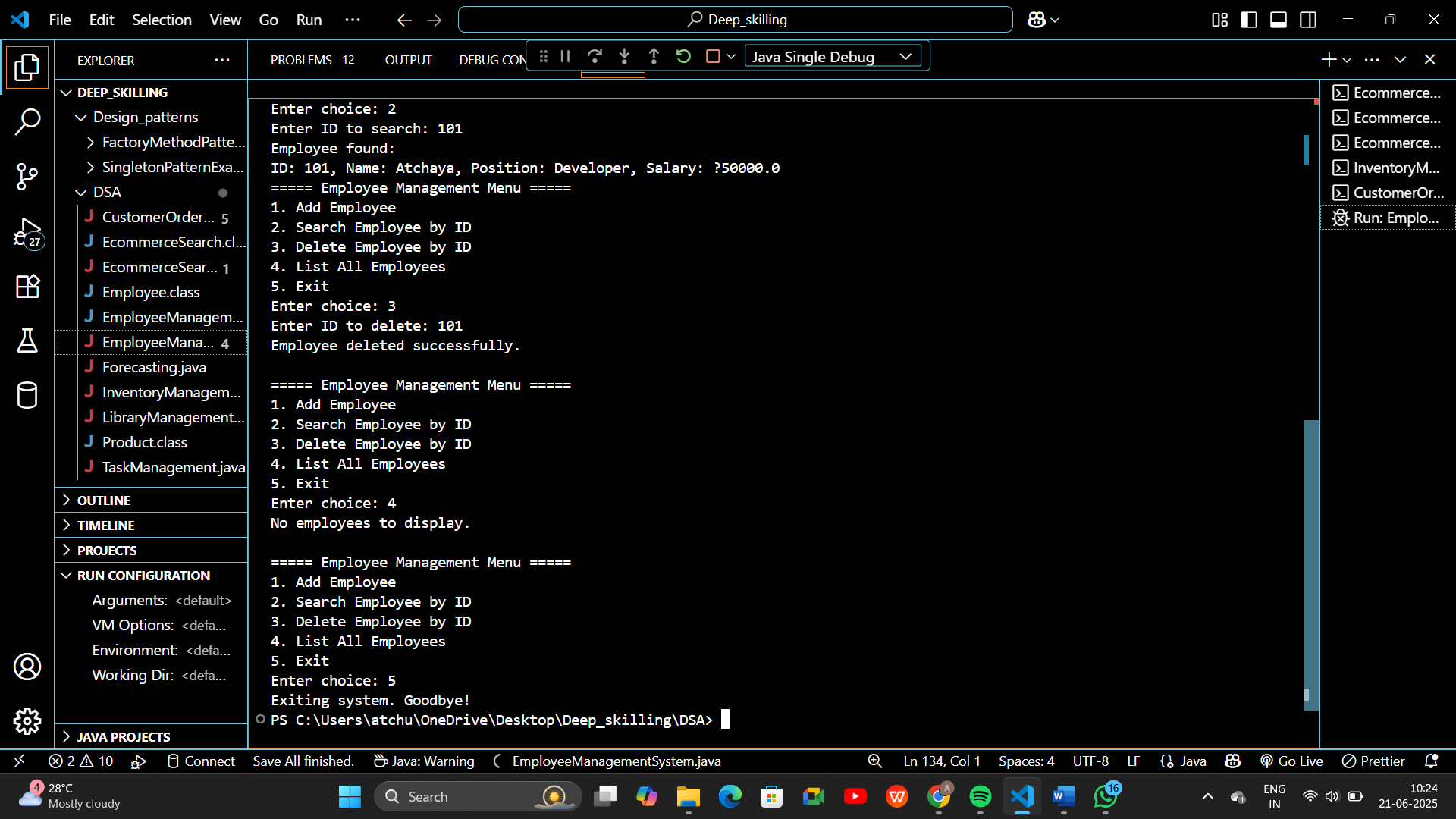
**Exercise 4: Employee Management System**

**PROGRAM:**

1. import java.util.Scanner;
2. class Employee {
3. int employeeId;
4. String name;
5. String position;
6. double salary;
7. public Employee(int employeeId, String name, String position, double salary) {
8. this.employeeId = employeeId;
9. this.name = name;
10. this.position = position;
11. this.salary = salary;
12. }
13. public void display() {
14. System.out.println("ID: " + employeeId + ", Name: " + name + ", Position: " + position + ", Salary: ₹" + salary);
15. }
16. }
17. public class EmployeeManagementSystem {
18. private Employee[] employees;
19. private int count;
20. public EmployeeManagementSystem(int size) {
21. employees = new Employee[size];
22. count = 0;
23. }
24. public void addEmployee(Employee emp) {
25. if (count < employees.length) {
26. employees[count++] = emp;
27. System.out.println("Employee added successfully.\n");
28. } else {
29. System.out.println("Error: Employee list is full.\n");
30. }
31. }
32. public void searchEmployee(int id) {
33. for (int i = 0; i < count; i++) {
34. if (employees[i].employeeId == id) {
35. System.out.println("Employee found:");
36. employees[i].display();
37. return;
38. }
39. }
40. System.out.println("Employee with ID " + id + " not found.\n");
41. }
42. public void deleteEmployee(int id) {
43. for (int i = 0; i < count; i++) {
44. if (employees[i].employeeId == id) {
45. for (int j = i; j < count - 1; j++) {
46. employees[j] = employees[j + 1];
47. }
48. employees[--count] = null;
49. System.out.println("Employee deleted successfully.\n");
50. return;
51. }
52. }
53. System.out.println("Employee with ID " + id + " not found. Cannot delete.\n");
54. }
55. public void listEmployees() {
56. if (count == 0) {
57. System.out.println("No employees to display.\n");
58. return;
59. }
60. System.out.println("All Employees:");
61. for (int i = 0; i < count; i++) {
62. employees[i].display();
63. }
64. System.out.println();
65. }
66. public static void main(String[] args) {
67. EmployeeManagementSystem system = new EmployeeManagementSystem(10);
68. Scanner sc = new Scanner(System.in);
69. int choice;
70. do {
71. System.out.println("===== Employee Management Menu =====");
72. System.out.println("1. Add Employee");
73. System.out.println("2. Search Employee by ID");
74. System.out.println("3. Delete Employee by ID");
75. System.out.println("4. List All Employees");
76. System.out.println("5. Exit");
77. System.out.print("Enter choice: ");
78. choice = sc.nextInt();
79. sc.nextLine();  // consume newline
80. switch (choice) {
81. case 1:
82. System.out.print("Enter ID: ");
83. int id = sc.nextInt();
84. sc.nextLine();  // consume newline
85. System.out.print("Enter Name: ");
86. String name = sc.nextLine();
87. System.out.print("Enter Position: ");
88. String position = sc.nextLine();
89. System.out.print("Enter Salary: ");
90. double salary = sc.nextDouble();
91. system.addEmployee(new Employee(id, name, position, salary));
92. break;
93. case 2:
94. System.out.print("Enter ID to search: ");
95. int searchId = sc.nextInt();
96. system.searchEmployee(searchId);
97. break;
98. case 3:
99. System.out.print("Enter ID to delete: ");
100. int deleteId = sc.nextInt();
101. system.deleteEmployee(deleteId);
102. break;
103. case 4:
104. system.listEmployees();
105. break;
106. case 5:
107. System.out.println("Exiting system. Goodbye!");
108. break;
109. default:
110. System.out.println("Invalid choice. Try again.\n");
111. }
112. } while (choice != 5);
113. sc.close();
114. }
115. }

**OUTPUT:**

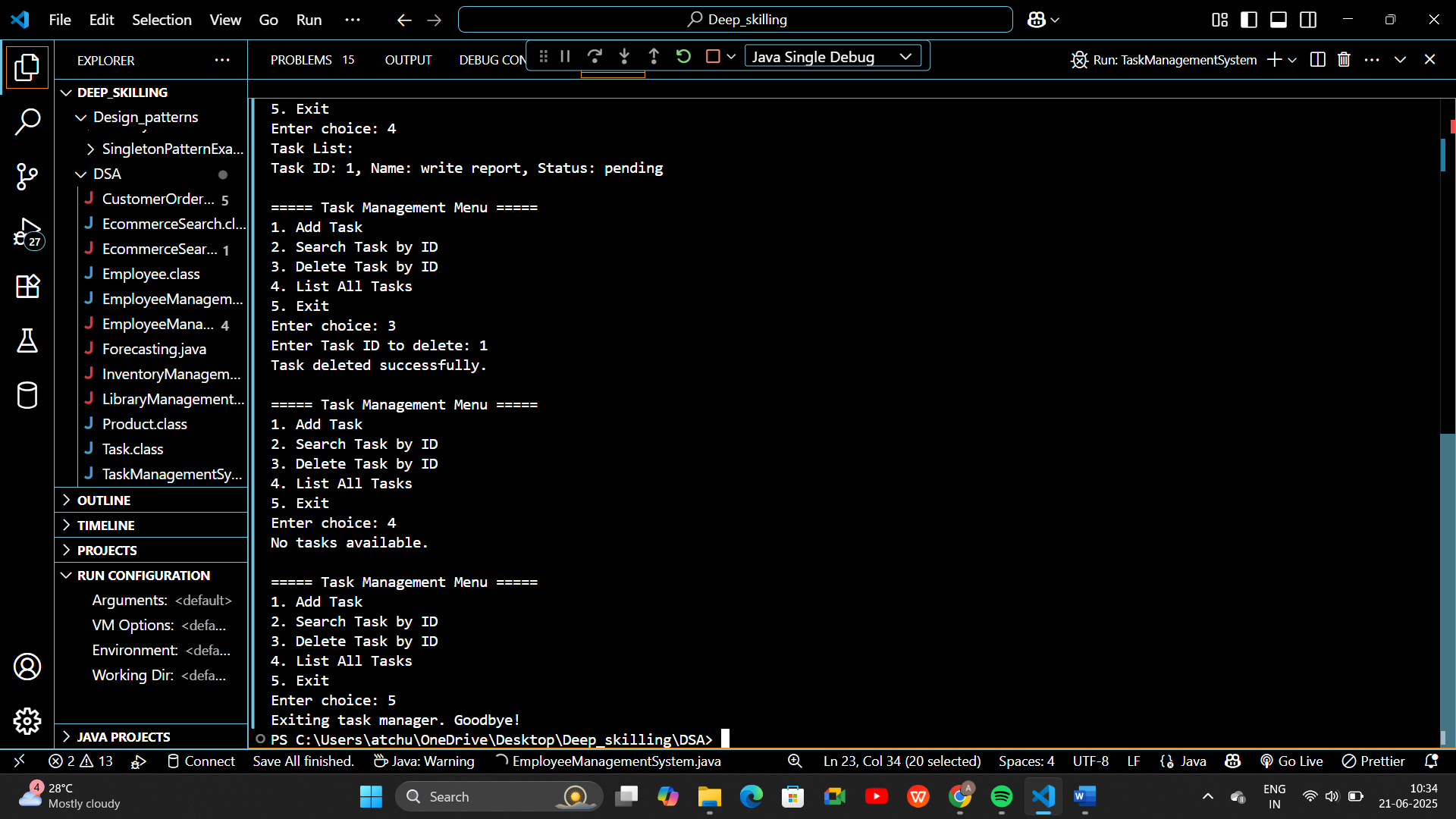
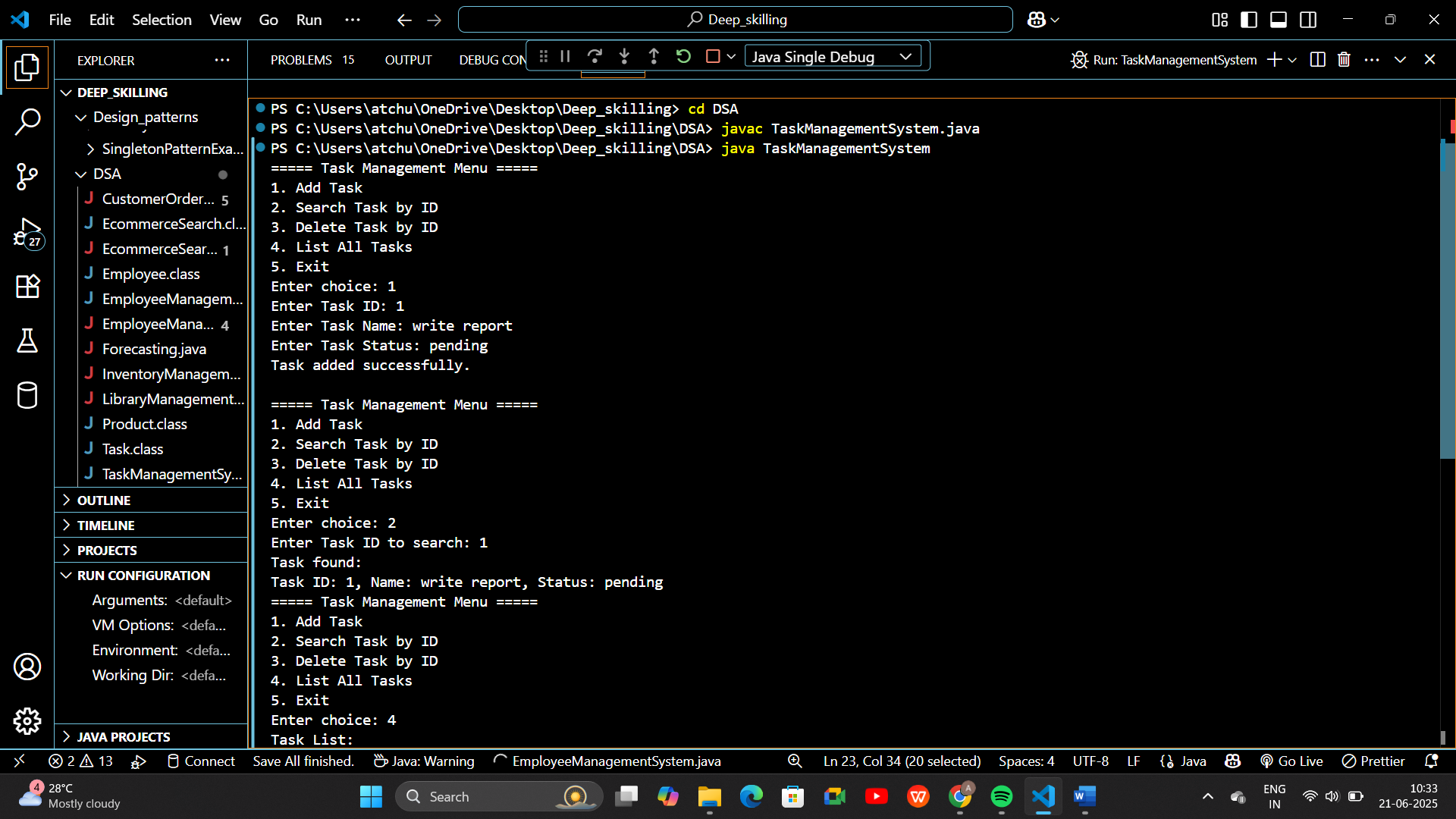
****

****

**Exercise 5: Task Management System**

**PROGRAM:**

1. import java.util.Scanner;
2. class Task {
3. int taskId;
4. String taskName;
5. String status;
6. Task next;
7. public Task(int taskId, String taskName, String status) {
8. this.taskId = taskId;
9. this.taskName = taskName;
10. this.status = status;
11. this.next = null;
12. }
13. public void display() {
14. System.out.println("Task ID: " + taskId + ", Name: " + taskName + ", Status: " + status);
15. }
16. }
18. public class TaskManagementSystem {
19. private Task head; // head of the linked list
20. // Add a new task at the end
21. public void addTask(int id, String name, String status) {
22. Task newTask = new Task(id, name, status);
23. if (head == null) {
24. head = newTask;
25. } else {
26. Task temp = head;
27. while (temp.next != null) {
28. temp = temp.next;
29. }
30. temp.next = newTask;
31. }
32. System.out.println("Task added successfully.\n");
33. }
34. public void searchTask(int id) {
35. Task temp = head;
36. while (temp != null) {
37. if (temp.taskId == id) {
38. System.out.println("Task found:");
39. temp.display();
40. return;
41. }
42. temp = temp.next;
43. }
44. System.out.println("Task with ID " + id + " not found.\n");
45. }
46. public void listTasks() {
47. if (head == null) {
48. System.out.println("No tasks available.\n");
49. return;
50. }
51. System.out.println("Task List:");
52. Task temp = head;
53. while (temp != null) {
54. temp.display();
55. temp = temp.next;
56. }
57. System.out.println();
58. }
59. public void deleteTask(int id) {
60. if (head == null) {
61. System.out.println("Task list is empty.\n");
62. return;
63. }
65. if (head.taskId == id) {
66. head = head.next;
67. System.out.println("Task deleted successfully.\n");
68. return;
69. }
70. Task prev = null;
71. Task current = head;
72. while (current != null && current.taskId != id) {
73. prev = current;
74. current = current.next;
75. }
76. if (current == null) {
77. System.out.println("Task with ID " + id + " not found.\n");
78. } else {
79. prev.next = current.next;
80. System.out.println("Task deleted successfully.\n");
81. }
82. }
84. public static void main(String[] args) {
85. TaskManagementSystem system = new TaskManagementSystem();
86. Scanner sc = new Scanner(System.in);
87. int choice;
88. do {
89. System.out.println("===== Task Management Menu =====");
90. System.out.println("1. Add Task");
91. System.out.println("2. Search Task by ID");
92. System.out.println("3. Delete Task by ID");
93. System.out.println("4. List All Tasks");
94. System.out.println("5. Exit");
95. System.out.print("Enter choice: ");
96. choice = sc.nextInt();
97. sc.nextLine();
98. switch (choice) {
99. case 1:
100. System.out.print("Enter Task ID: ");
101. int id = sc.nextInt();
102. sc.nextLine();
103. System.out.print("Enter Task Name: ");
104. String name = sc.nextLine();
105. System.out.print("Enter Task Status: ");
106. String status = sc.nextLine();
107. system.addTask(id, name, status);
108. break;
109. case 2:
110. System.out.print("Enter Task ID to search: ");
111. int searchId = sc.nextInt();
112. system.searchTask(searchId);
113. break;
114. case 3:
115. System.out.print("Enter Task ID to delete: ");
116. int deleteId = sc.nextInt();
117. system.deleteTask(deleteId);
118. break;
119. case 4:
120. system.listTasks();
121. break;
122. case 5:
123. System.out.println("Exiting task manager. Goodbye!");
124. break;
125. default:
126. System.out.println("Invalid choice. Try again.\n");
127. }
128. } while (choice != 5);
129. sc.close();
130. }
131. }

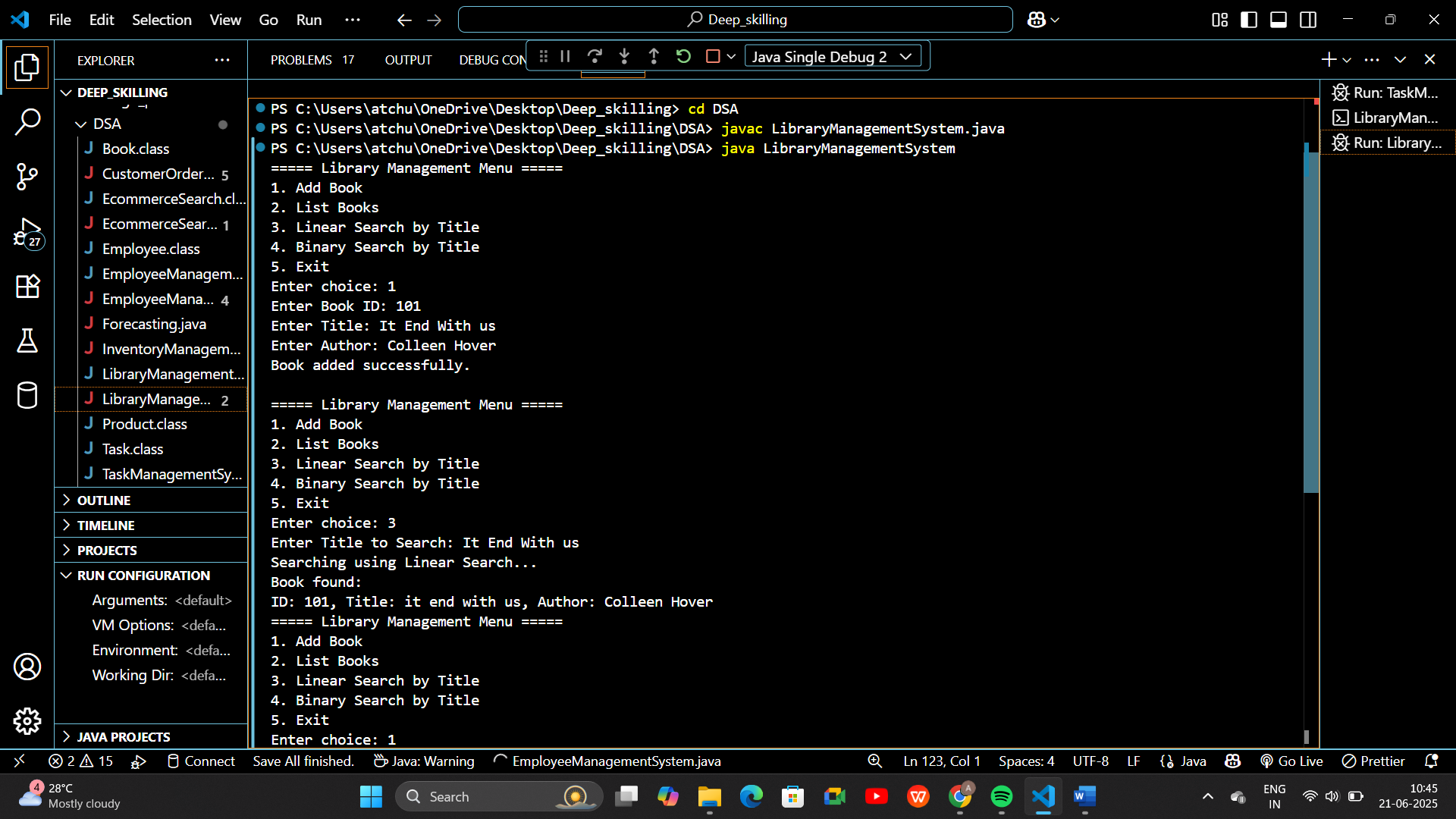
**OUTPUT:**

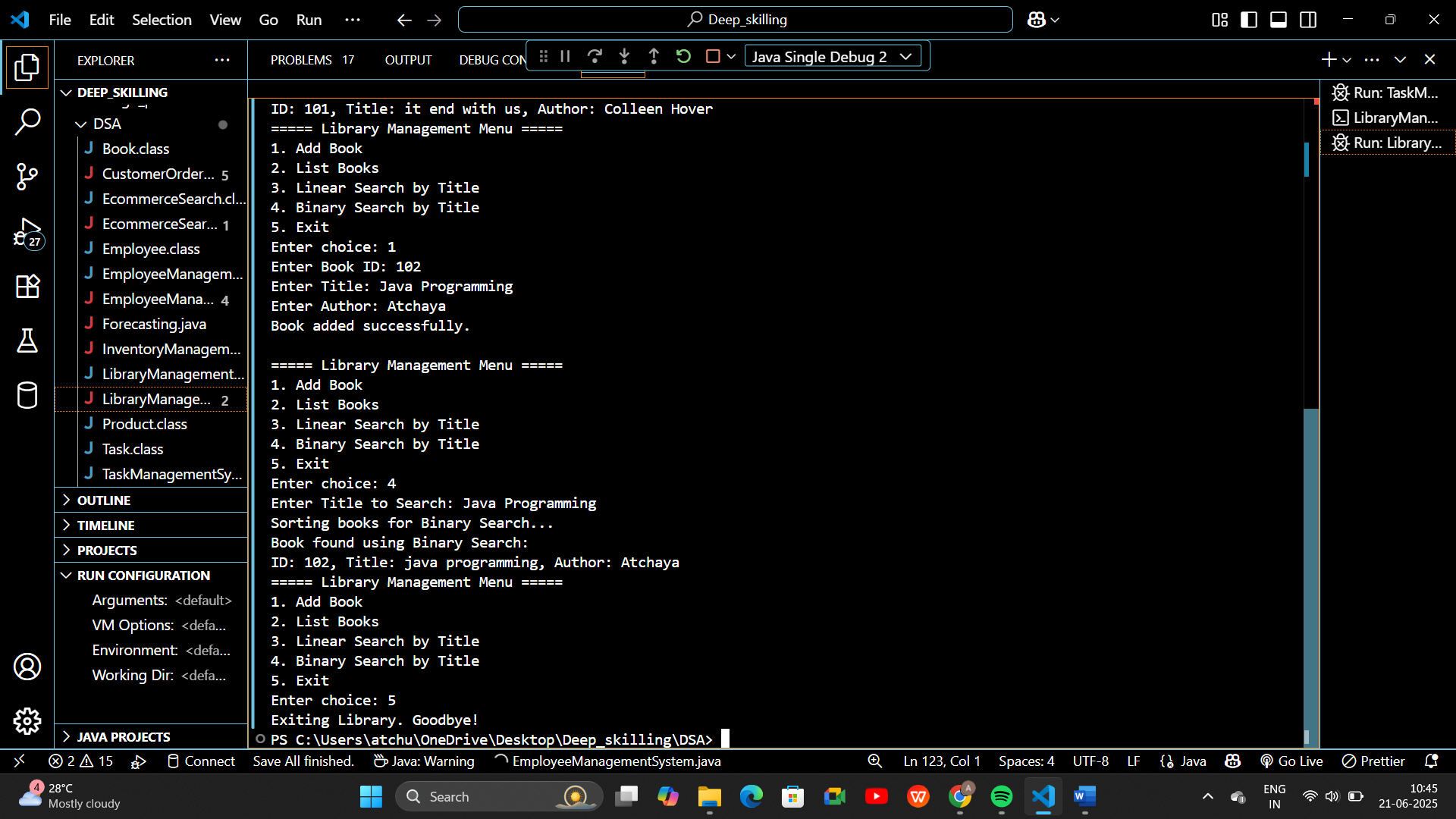
**Exercise 6: Library Management System**

**PROGRAM:**

1. import java.util.Arrays;
2. import java.util.Scanner;
3. class Book implements Comparable<Book> {
4. int bookId;
5. String title;
6. String author;
7. public Book(int bookId, String title, String author) {
8. this.bookId = bookId;
9. this.title = title.toLowerCase();
10. this.author = author;
11. }
12. public void display() {
13. System.out.println("ID: " + bookId + ", Title: " + title + ", Author: " + author);
14. }
16. @Override
17. public int compareTo(Book other) {
18. return this.title.compareTo(other.title);
19. }
20. }
21. public class LibraryManagementSystem {
22. static Book[] books = new Book[10];
23. static int count = 0;
25. public static void addBook(int id, String title, String author) {
26. if (count < books.length) {
27. books[count++] = new Book(id, title, author);
28. System.out.println("Book added successfully.\n");
29. } else {
30. System.out.println("Library is full.\n");
31. }
32. }
33. // Linear Search: O(n)
34. public static void linearSearch(String title) {
35. System.out.println("Searching using Linear Search...");
36. for (int i = 0; i < count; i++) {
37. if (books[i].title.equalsIgnoreCase(title)) {
38. System.out.println("Book found:");
39. books[i].display();
40. return;
41. }
42. }
43. System.out.println("Book not found.\n");
44. }
45. // Binary Search: O(log n)
46. public static void binarySearch(String title) {
47. System.out.println("Sorting books for Binary Search...");
48. Arrays.sort(books, 0, count); // sort first
49. int low = 0, high = count - 1;
50. title = title.toLowerCase();
51. while (low <= high) {
52. int mid = (low + high) / 2;
53. int cmp = books[mid].title.compareTo(title);
54. if (cmp == 0) {
55. System.out.println("Book found using Binary Search:");
56. books[mid].display();
57. return;
58. } else if (cmp < 0) {
59. low = mid + 1;
60. } else {
61. high = mid - 1;
62. }
63. }
64. System.out.println("Book not found using Binary Search.\n");
65. }
67. public static void listBooks() {
68. if (count == 0) {
69. System.out.println("Library is empty.\n");
70. return;
71. }
72. System.out.println("Books in Library:");
73. for (int i = 0; i < count; i++) {
74. books[i].display();
75. }
76. System.out.println();
77. }
79. public static void main(String[] args) {
80. Scanner sc = new Scanner(System.in);
81. int choice;
82. do {
83. System.out.println("===== Library Management Menu =====");
84. System.out.println("1. Add Book");
85. System.out.println("2. List Books");
86. System.out.println("3. Linear Search by Title");
87. System.out.println("4. Binary Search by Title");
88. System.out.println("5. Exit");
89. System.out.print("Enter choice: ");
90. choice = sc.nextInt();
91. sc.nextLine();
92. switch (choice) {
93. case 1:
94. System.out.print("Enter Book ID: ");
95. int id = sc.nextInt();
96. sc.nextLine();
97. System.out.print("Enter Title: ");
98. String title = sc.nextLine();
99. System.out.print("Enter Author: ");
100. String author = sc.nextLine();
101. addBook(id, title, author);
102. break;
103. case 2:
104. listBooks();
105. break;
106. case 3:
107. System.out.print("Enter Title to Search: ");
108. String searchLinear = sc.nextLine();
109. linearSearch(searchLinear);
110. break;
111. case 4:
112. System.out.print("Enter Title to Search: ");
113. String searchBinary = sc.nextLine();
114. binarySearch(searchBinary);
115. break;
116. case 5:
117. System.out.println("Exiting Library. Goodbye!");
118. break;
119. default:
120. System.out.println("Invalid choice. Try again.\n");
121. }
122. } while (choice != 5);
123. sc.close();
124. }
125. }

**OUTPUT:**





**Exercise 7: Financial Forecasting**

**PROGRAM:**

1. import java.util.HashMap;
2. import java.util.Scanner;
3. public class FinancialForecasting {
4. // Memoization map to cache computed results for optimization
5. static HashMap<Integer, Double> memo = new HashMap<>();
6. /\*\*
7. \* Recursive method to calculate future value.
8. \* Formula: futureValue(year) = futureValue(year-1) \* (1 + rate)
9. \*/
10. public static double predictValue(int year, double initialValue, double rate) {
11. if (year == 0) {
12. return initialValue;
13. }
14. // Check if result is already computed
15. if (memo.containsKey(year)) {
16. return memo.get(year);
17. }
18. // Recursive step
19. double result = predictValue(year - 1, initialValue, rate) \* (1 + rate);
20. memo.put(year, result); // Store result for reuse
21. return result;
22. }
23. // Driver program
24. public static void main(String[] args) {
25. Scanner sc = new Scanner(System.in);
26. System.out.println("=== Financial Forecasting Tool ===");
27. System.out.print("Enter initial value (e.g., profit or investment): ");
28. double initialValue = sc.nextDouble();
29. System.out.print("Enter annual growth rate (e.g., 0.10 for 10%): ");
30. double rate = sc.nextDouble();
31. System.out.print("Enter number of years to forecast: ");
32. int years = sc.nextInt();
33. System.out.println("\nForecasted values over " + years + " years:");
34. for (int i = 0; i <= years; i++) {
35. double value = predictValue(i, initialValue, rate);
36. System.out.printf("Year %d: %.2f\n", i, value);
37. }
38. sc.close();
39. }
40. }

**Output:**

