

## Exercise 1: Inventory Management System

### Scenario:

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

### Steps:

#### 1. Understand the Problem:

- Explain why data structures and algorithms are essential in handling large inventories.
- Discuss the types of data structures suitable for this problem.

Data Structure used- hashmap

#### 2. Setup:

- Create a new project for the inventory management system.

#### 3. Implementation:

##### Product.java

```
package module2.Algorithms_Data_Structures.InventoryManagementSystem;
```

```
public class Product {  
    private int productId;  
    private String productName;  
    private int quantity;  
    private double price;  
  
    // Constructor  
    public Product(int productId, String productName, int quantity, double price) {  
        this.productId = productId;  
        this.productName = productName;  
        this.quantity = quantity;  
        this.price = price;  
    }  
    // Getters and Setters
```

```
public int getProductId() {  
    return productId;  
}
```

```
public void setProductId(int productId) {  
    this.productId = productId;  
}
```

```
public String getProductName() {  
    return productName;  
}
```

```
public void setProductName(String productName) {  
    this.productName = productName;  
}
```

```
public int getQuantity() {  
    return quantity;  
}
```

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

```
public double getPrice() {  
    return price;  
}
```

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

```
// toString() for display
```

```
@Override
```

```

        public String toString() {
            return "Product [productId=" + productId + ", productName=" +
productName + ", quantity=" + quantity
            + ", price=" + price + "]\n";
        }
    }
}

```

### **InventoryService.java**

```

package module2.Algorithms_Data_Structures.InventoryManagementSystem;

import java.util.HashMap;

public class InventoryService {
    private HashMap<Integer, Product> inventory;

    public InventoryService() {
        inventory = new HashMap<>();
    }

    public void addProduct(Product product) {
        inventory.put(product.getProductId(), product);
    }

    public void updateProduct(int productId, int newQty, double newPrice) {
        Product p = inventory.get(productId);
        if (p != null) {
            p.setQuantity(newQty);
            p.setPrice(newPrice);
        }
    }

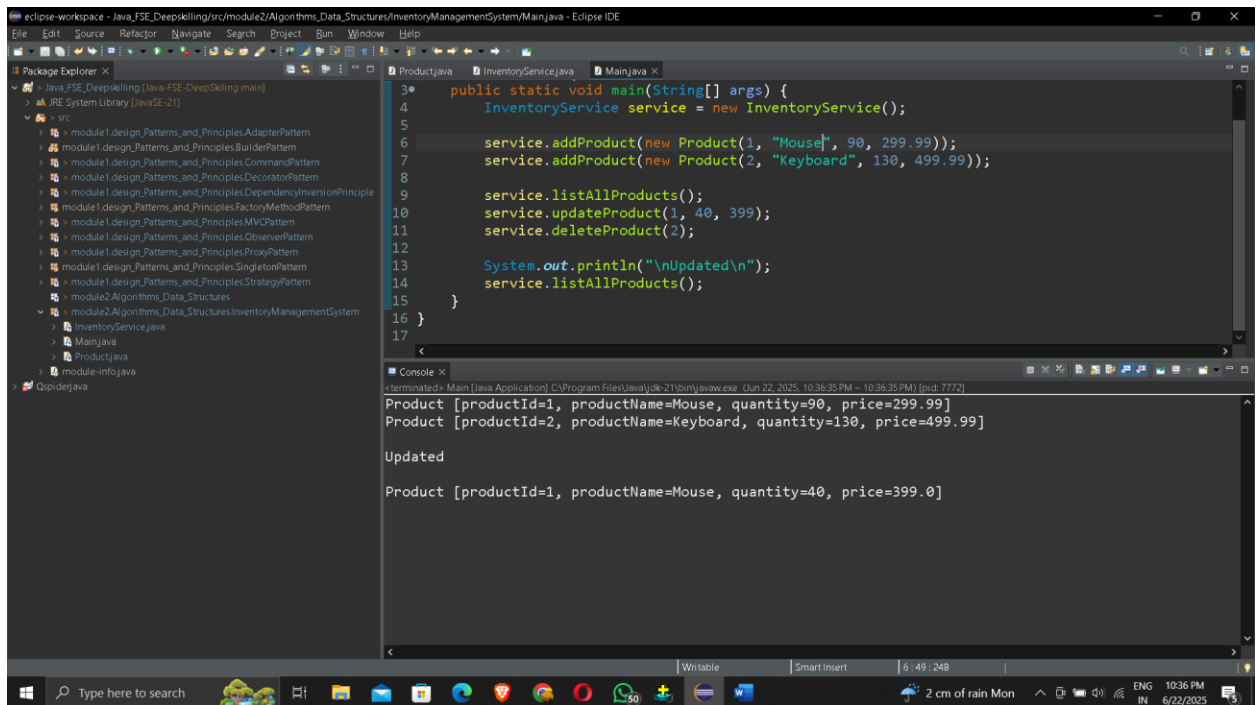
    public void deleteProduct(int productId) {
        inventory.remove(productId);
    }
}

```

```
public Product getProduct(int productId) {  
    return inventory.get(productId);  
}  
  
public void listAllProducts() {  
    for (Product p : inventory.values()) {  
        System.out.println(p);  
    }  
}  
}
```

### **Main.java**

```
package module2.Algorithms_Data_Structures.InventoryManagementSystem;  
  
public class Main {  
    public static void main(String[] args) {  
        InventoryService service = new InventoryService();  
  
        service.addProduct(new Product(1, "Mouse", 90, 299.99));  
        service.addProduct(new Product(2, "Keyboard", 130, 499.99));  
  
        service.listAllProducts();  
        service.updateProduct(1, 40, 399);  
        service.deleteProduct(2);  
  
        System.out.println("\nUpdated\n");  
        service.listAllProducts();  
    }  
}
```



## 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.

### Steps:

#### 1. Understand Asymptotic Notation:

- Explain Big O notation and how it helps in analyzing algorithms.
- Describe the best, average, and worst-case scenarios for search operations.

#### 2. Setup:

- Create a class **Product** with attributes for searching, such as **productId**, **productName**, and **category**.

#### 3. Implementation:

##### Product.java

```
package module2.Algorithms_Data_Structures.Ecommerce_Search;
```

```
public class Product {  
    private int productId;
```

```
private String productName;

private String category;

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

    this.productId = productId;

    this.productName = productName;

    this.category = category;

}
```

```
public int getProductId() { return productId; }

public String getProductName() { return productName; }

public String getCategory() { return category; }
```

```
@Override

public String toString() {

    return productId + " - " + productName + " [" + category + "];

}

}
```

### **SearchOperation.java**

```
package module2.Algorithms_Data_Structures.Ecommerce_Search;
```

```
import java.util.Arrays;

import java.util.Comparator;
```

```
public class SearchingOperation {

    // Linear Search

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

        for (Product p : products) {
```

```

        if (p.getProductName().equalsIgnoreCase(targetName)) {
            return p;
        }
    }
    return null;
}

// Binary Search (requires sorted array)
public Product binarySearch(Product[] products, String targetName) {
    Arrays.sort(products, Comparator.comparing(Product::getProductName));

    int left = 0, right = products.length - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        int cmp = products[mid].getProductName().compareToIgnoreCase(targetName);

        if (cmp == 0) return products[mid];
        else if (cmp < 0) left = mid + 1;
        else right = mid - 1;
    }
    return null;
}
}

```

### **Main.java**

```

package module2.Algorithms_Data_Structures.Ecommerce_Search;

public class Main {
    public static void main(String[] args) {
        Product[] products = {

```

```
        new Product(101, "Laptop", "Electronics"),
        new Product(102, "Shoes", "Footwear"),
        new Product(103, "Phone", "Electronics"),
        new Product(104, "Book", "Stationery")
    };

    SearchingOperation search = new SearchingOperation();

    String target = "Phone";

    Product linearResult = search.linearSearch(products, target);

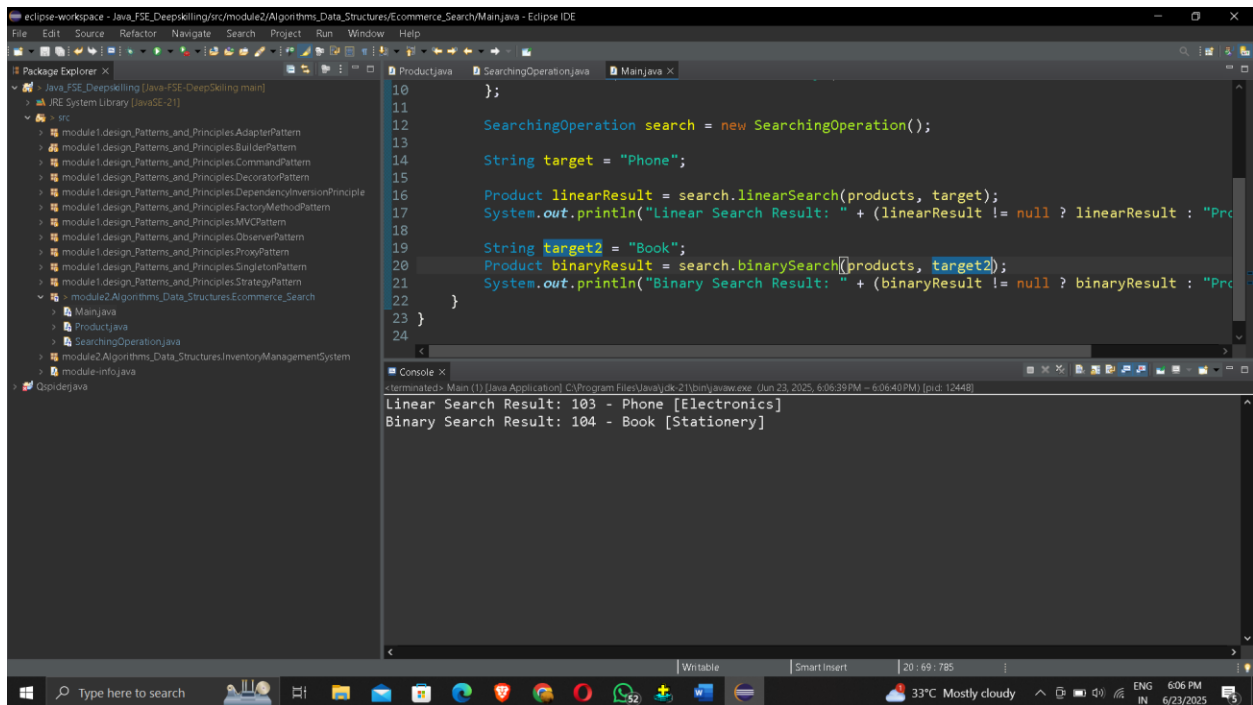
    System.out.println("Linear Search Result: " + (linearResult != null ? linearResult : "Product
Not Found"));

    String target2 = "Book";

    Product binaryResult = search.binarySearch(products, target2);

    System.out.println("Binary Search Result: " + (binaryResult != null ? binaryResult :
"Product Not Found"));
    }
}
```





## 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.

### Steps:

#### 1. Understand Sorting Algorithms:

- Explain different sorting algorithms (Bubble Sort, Insertion Sort, Quick Sort, Merge Sort).

#### 2. Setup:

- Create a class **Order** with attributes like **orderId**, **customerName**, and **totalPrice**.

#### 3. Implementation:

##### Order.java

```
package module2.Algorithms_Data_Structures.Sorting_Order;
```

```
public class Order {  
    private int orderId;  
    private String customerName;
```

```
private double totalPrice;
```

```
public Order(int orderId, String customerName, double totalPrice) {  
    this.orderId = orderId;  
    this.customerName = customerName;  
    this.totalPrice = totalPrice;  
}
```

```
public int getOrderId() { return orderId; }  
public String getCustomerName() { return customerName; }  
public double getTotalPrice() { return totalPrice; }
```

```
@Override
```

```
public String toString() {  
    return orderId + " - " + customerName + " : ₹" + totalPrice;  
}  
}
```

### **SortOrder.java**

```
package module2.Algorithms_Data_Structures.Sorting_Order;
```

```
public class SortOrder {
```

```
// Bubble Sort
```

```
public 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].getTotalPrice() > orders[j + 1].getTotalPrice()) {  
                Order temp = orders[j];  
                orders[j] = orders[j + 1];  
                orders[j + 1] = temp;  
            }  
        }  
    }  
}
```

```

        orders[j + 1] = temp;
    }
}
}
}

// Quick Sort
public 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 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;
}

```

```

// Utility to print array
public void printOrders(Order[] orders) {
    for (Order o : orders) {
        System.out.println(o);
    }
}
}

Main.java

package module2.Algorithms_Data_Structures.Sorting_Order;

public class Main {
    public static void main(String[] args) {
        Order[] orders1 = {
            new Order(301, "Ravi", 750.50),
            new Order(302, "Sneha", 1200.00),
            new Order(303, "Ankit", 450.25),
            new Order(304, "Meena", 999.99),
            new Order(305, "Kiran", 1100.00)
        };

        SortOrder sorter = new SortOrder();

        //
        // System.out.println("Original Orders:");
        // sorter.printOrders(orders1);

        // Bubble Sort
        sorter.bubbleSort(orders1);
        System.out.println("\nBubble Sorted Orders by Price:");
        sorter.printOrders(orders1);

        // Reset array for quick sort

```

```

Order []orders2 = new Order[] {
    new Order(201, "Alice", 499.99),
    new Order(202, "Bob", 1399.50),
    new Order(203, "Charlie", 299.99),
    new Order(204, "Diana", 999.99)
};

sorter.quickSort(orders2, 0, orders2.length - 1);

System.out.println("\nQuick Sorted Orders by Price:");

sorter.printOrders(orders2);
}
}

```

```

// Main.java
9      new Order(304, "Meena", 999.99),
10     new Order(305, "Kiran", 1100.00)
11     };
12
13     SortOrder sorter = new SortOrder();
14
15     // System.out.println("Original Orders:");
16     // sorter.printOrders(orders1);
17
18     // Bubble Sort
19     sorter.bubbleSort(orders1);
20     System.out.println("\nBubble Sorted Orders by Price:");
21     sorter.printOrders(orders1);
22
23     // Reset array for quick sort

```

```

// SortOrder.java
1  public class SortOrder {
2
3      private Order[] orders;
4
5      public SortOrder() {
6          orders = new Order[10];
7      }
8
9      public void bubbleSort() {
10         for (int i = 0; i < orders.length - 1; i++) {
11             for (int j = 0; j < orders.length - i - 1; j++) {
12                 if (orders[j].price > orders[j + 1].price) {
13                     Order temp = orders[j];
14                     orders[j] = orders[j + 1];
15                     orders[j + 1] = temp;
16                 }
17             }
18         }
19     }
20
21     public void quickSort(int low, int high) {
22         if (low < high) {
23             int pivotIndex = partition(low, high);
24             quickSort(low, pivotIndex - 1);
25             quickSort(pivotIndex + 1, high);
26         }
27     }
28
29     private int partition(int low, int high) {
30         int pivot = orders[high].price;
31         int i = low - 1;
32         for (int j = low; j < high; j++) {
33             if (orders[j].price < pivot) {
34                 i++;
35                 Order temp = orders[i];
36                 orders[i] = orders[j];
37                 orders[j] = temp;
38             }
39         }
40         Order temp = orders[i + 1];
41         orders[i + 1] = orders[high];
42         orders[high] = temp;
43         return i + 1;
44     }
45
46     public void printOrders() {
47         for (Order order : orders) {
48             System.out.println(order);
49         }
50     }
51 }

```

```

Bubble Sorted Orders by Price:
303 - Ankit : ₹450.25
301 - Ravi : ₹750.5
304 - Meena : ₹999.99
305 - Kiran : ₹1100.0
302 - Sneha : ₹1200.0

Quick Sorted Orders by Price:
203 - Charlie : ₹299.99
201 - Alice : ₹499.99
204 - Diana : ₹999.99
202 - Bob : ₹1399.5

```

## Exercise 4: Employee Management System

### Scenario:

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

## Steps:

### 1. Understand Array Representation:

- Explain how arrays are represented in memory and their advantages.

### 2. Setup:

- Create a class Employee with attributes like **employeeId**, **name**, **position**, and **salary**.

### 3. Implementation:

- Use an array to store employee records.
- Implement methods to **add**, **search**, **traverse**, and **delete** employees in the array.

## Employee.java

```
package module2.Algorithms_Data_Structures.EmployeeManagement;
```

```
public class Employee {
    private int employeeId;
    private String name;
    private String position;
    private 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 int getEmployeeId() { return employeeId; }
    public String getName() { return name; }
    public String getPosition() { return position; }
    public double getSalary() { return salary; }

    @Override
    public String toString() {
        return employeeId + " - " + name + " [" + position + "] ₹" + salary;
    }
}
```

## EmployeeService.java

```
package module2.Algorithms_Data_Structures.EmployeeManagement;
```

```
public class EmployeeService {
    private Employee[] employees;
    private int size;
```

```

public EmployeeService(int capacity) {
    employees = new Employee[capacity];
    size = 0;
}

// Add Employee
public void addEmployee(Employee emp) {
    if (size < employees.length) {
        employees[size++] = emp;
    } else {
        System.out.println("Employee array is full!");
    }
}

// Search Employee by ID
public Employee searchEmployee(int empId) {
    for (int i = 0; i < size; i++) {
        if (employees[i].getEmployeeId() == empId) {
            return employees[i];
        }
    }
    return null;
}

// Traverse All Employees
public void listEmployees() {
    for (int i = 0; i < size; i++) {
        System.out.println(employees[i]);
    }
}

// Delete Employee by ID
public boolean deleteEmployee(int empId) {
    for (int i = 0; i < size; i++) {
        if (employees[i].getEmployeeId() == empId) {
            for (int j = i; j < size - 1; j++) {
                employees[j] = employees[j + 1];
            }
            employees[--size] = null;
            return true;
        }
    }
    return false;
}
}

Main.java
package module2.Algorithms_Data_Structures.EmployeeManagement;

```

```

public class Main {
    public static void main(String[] args) {
        EmployeeService service = new EmployeeService(10);

        service.addEmployee(new Employee(101, "Ravi ", "Developer", 65000));
        service.addEmployee(new Employee(102, "Sneha ", "Designer", 58000));
        service.addEmployee(new Employee(103, "Mehta", "Manager", 85000));
        service.addEmployee(new Employee(104, "Meena", "QA Engineer", 55000));

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

        System.out.println("\nSearching for Employee ID 2:");
        Employee found = service.searchEmployee(102);
        System.out.println(found != null ? found : "Employee not found");

        System.out.println("\nDeleting Employee ID 2:");
        boolean deleted = service.deleteEmployee(102);
        System.out.println(deleted ? "Deleted successfully" : "Employee not found");

        System.out.println("\nUpdated Employee List:");
        service.listEmployees();
    }
}

```

The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Displays the project structure, including the 'src' folder and various design patterns like AdapterPattern, BuilderPattern, CommandPattern, etc.
- Editor:** Shows the 'Main.java' file with the code from the previous block. The code is color-coded and includes line numbers from 13 to 27.
- Console:** Displays the output of the program, showing the list of employees, the search result for Employee ID 2 (Sneha), the deletion of Employee ID 2, and the updated employee list.

The console output is as follows:

```

terminated> Main (3) [Java Application] C:\Program Files\Java\jdk-21\bin\java.exe (Jun 23, 2025, 7:28:49 PM - 7:28:50 PM) [pid: 15184]
101 - Ravi [Developer] ₹65000.0
102 - Sneha [Designer] ₹58000.0
103 - Mehta [Manager] ₹85000.0
104 - Meena [QA Engineer] ₹55000.0

Searching for Employee ID 2:
102 - Sneha [Designer] ₹58000.0

Deleting Employee ID 2:
Deleted successfully

Updated Employee List:
101 - Ravi [Developer] ₹65000.0
103 - Mehta [Manager] ₹85000.0

```



## Exercise 5: Task Management System

### Scenario:

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

### Steps:

#### 1. Understand Linked Lists:

- Explain the different types of linked lists (Singly Linked List, Doubly Linked List).

#### 2. Setup:

- Create a class **Task** with attributes like **taskId**, **taskName**, and **status**.

#### 3. Implementation:

- Implement a singly linked list to manage tasks.
- Implement methods to **add**, **search**, **traverse**, and **delete** tasks in the linked list.

### Task.java

```
package module2.Algorithms_Data_Structures.TaskManagement;
```

```
public class Task {  
    private int taskId;  
    private String taskName;  
    private String status;  
  
    public Task(int taskId, String taskName, String status) {  
        this.taskId = taskId;  
        this.taskName = taskName;  
        this.status = status;  
    }  
  
    public int getTaskId() { return taskId; }  
    public String getTaskName() { return taskName; }  
    public String getStatus() { return status; }  
  
    @Override  
    public String toString() {  
        return taskId + " - " + taskName + " [" + status + "];"  
    }  
}
```

### TaskLL.java

```
package module2.Algorithms_Data_Structures.TaskManagement;
```

```

public class TaskLL {
    class Node {
        Task task;
        Node next;

        Node(Task task) {
            this.task = task;
            this.next = null;
        }
    }

    private Node head;

    // Add task at end
    public void addTask(Task task) {
        Node newNode = new Node(task);
        if (head == null) {
            head = newNode;
        } else {
            Node temp = head;
            while (temp.next != null)
                temp = temp.next;
            temp.next = newNode;
        }
    }

    // Traverse and display tasks
    public void traverseTasks() {
        Node temp = head;
        if (temp == null) {
            System.out.println("Task list is empty.");
            return;
        }
        while (temp != null) {
            System.out.println(temp.task);
            temp = temp.next;
        }
    }

    // Search by taskId
    public Task searchTask(int id) {
        Node temp = head;
        while (temp != null) {
            if (temp.task.getTaskId() == id)
                return temp.task;
            temp = temp.next;
        }
    }
}

```

```

    }
    return null;
}
// Delete by taskId
public boolean deleteTask(int id) {
    if (head == null) return false;

    if (head.task.getTaskId() == id) {
        head = head.next;
        return true;
    }
    Node prev = head;
    Node curr = head.next;

    while (curr != null) {
        if (curr.task.getTaskId() == id) {
            prev.next = curr.next;
            return true;
        }
        prev = curr;
        curr = curr.next;
    }

    return false;
}
}

```

### **Main.java**

```

package module2.Algorithms_Data_Structures.TaskManagement;

public class Main {
    public static void main(String[] args) {
        TaskLL taskList = new TaskLL();

        taskList.addTask(new Task(1, "Design Pattern", "Completed"));
        taskList.addTask(new Task(2, "Implement Data Structures", "In Progress"));
        taskList.addTask(new Task(3, "Spring", "Pending"));

        System.out.println("All Tasks:");
        taskList.traverseTasks();

        System.out.println("\nSearching for Task ID 2:");
        Task found = taskList.searchTask(2);
        System.out.println(found != null ? found : "Task not found");
    }
}

```

```

        System.out.println("\nDeleting Task ID 2:");
        boolean deleted = taskList.deleteTask(2);
        System.out.println(deleted ? "Deleted successfully" : "Task not found");

        System.out.println("\nUpdated Task List:");
        taskList.traverseTasks();
    }
}

```

The screenshot shows the Eclipse IDE with a project named 'Java\_FSE\_DeepSkillring'. The package explorer on the left shows a hierarchy of packages and classes. The main editor displays the code for 'Main.java', which includes methods for searching, deleting, and updating tasks. The console window at the bottom shows the output of the program, including the task list before and after operations.

```

13      System.out.println("\nSearching for Task ID 2:");
14      Task found = taskList.searchTask(2);
15      System.out.println(found != null ? found : "Task not found");
16
17
18      System.out.println("\nDeleting Task ID 2:");
19      boolean deleted = taskList.deleteTask(2);
20      System.out.println(deleted ? "Deleted successfully" : "Task not found");
21
22      System.out.println("\nUpdated Task List:");
23      taskList.traverseTasks();
24  }
25  }
26
27

```

Console Output:

```

All Tasks:
1 - Design Pattern [Completed]
2 - Implement Data Structures [In Progress]
3 - Spring [Pending]

Searching for Task ID 2:
2 - Implement Data Structures [In Progress]

Deleting Task ID 2:
Deleted successfully

Updated Task List:
1 - Design Pattern [Completed]
3 - Spring [Pending]

```

## Exercise 6: Library Management System

### Scenario:

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

### Steps:

1. **Understand Search Algorithms:**
  - Explain linear search and binary search algorithms.
2. **Setup:**
  - Create a class **Book** with attributes like **bookId**, **title**, and **author**.
3. **Implementation:**

- Implement linear search to find books by title.
- Implement binary search to find books by title (assuming the list is sorted).

### **Book.java**

```
package module2.Algorithms_Data_Structures.LibraryManagement;
```

```
public 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 bookId + " - " + title + " by " + author;  
    }  
}
```

### **Library.java**

```
package module2.Algorithms_Data_Structures.LibraryManagement;
```

```
import java.util.Arrays;  
import java.util.Comparator;  
  
public class Library {  
  
    // Linear Search by Title  
    public Book linearSearch(Book[] books, String targetTitle) {  
        for (Book b : books) {  
            if (b.getTitle().equalsIgnoreCase(targetTitle)) {  
                return b;  
            }  
        }  
        return null;  
    }  
}
```

```

// Binary Search by Title (Assumes sorted list)
public Book binarySearch(Book[] books, String targetTitle) {
    Arrays.sort(books, Comparator.comparing(Book::getTitle)); // Ensure sorted

    int left = 0, right = books.length - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        int cmp = books[mid].getTitle().compareToIgnoreCase(targetTitle);
        if (cmp == 0) return books[mid];
        else if (cmp < 0) left = mid + 1;
        else right = mid - 1;
    }
    return null;
}

//Print all books
public void listBooks(Book[] books) {
    for (Book b : books) {
        System.out.println(b);
    }
}
}

```

### **Main.java**

```

package module2.Algorithms_Data_Structures.LibraryManagement;

public class Main {
    public static void main(String[] args) {
        Book[] books = {
            new Book(1, "The Alchemist", "Paulo Coelho"),
            new Book(2, "To Kill a Mockingbird", "Harper Lee"),
            new Book(3, "1984", "George Orwell"),
            new Book(4, "The Great Gatsby", "F. Scott Fitzgerald"),
            new Book(5, "Moby Dick", "Herman Melville")
        };

        Library service = new Library();

        System.out.println("All Books:");
        service.listBooks(books);

        String searchTitle = "1984";
    }
}

```

```

Book linearResult = service.linearSearch(books, searchTitle);
System.out.println("\nLinear Search Result: " + (linearResult != null ? linearResult : "Not Found"));

searchTitle = "The Alchemist";
Book binaryResult = service.binarySearch(books, searchTitle);
System.out.println("Binary Search Result: " + (binaryResult != null ? binaryResult : "Not Found"));
}
}

```

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left lists the project structure, including a package named 'module2.Algorithms\_Data\_Structures.LibraryManagement'. The code editor in the center displays the following Java code:

```

13 Library service = new Library();
14
15 System.out.println("All Books:");
16 service.listBooks(books);
17
18 String searchTitle = "1984";
19
20 Book linearResult = service.linearSearch(books, searchTitle);
21 System.out.println("\nLinear Search Result: " + (linearResult != null ? linearResult : "Not Found"));
22
23 searchTitle = "The Alchemist";
24 Book binaryResult = service.binarySearch(books, searchTitle);
25 System.out.println("Binary Search Result: " + (binaryResult != null ? binaryResult : "Not Found"));
26 }
27 }

```

The Console window at the bottom shows the output of the program:

```

All Books:
1 - The Alchemist by Paulo Coelho
2 - To Kill a Mockingbird by Harper Lee
3 - 1984 by George Orwell
4 - The Great Gatsby by F. Scott Fitzgerald
5 - Moby Dick by Herman Melville

Linear Search Result: 3 - 1984 by George Orwell
Binary Search Result: 1 - The Alchemist by Paulo Coelho

```

## Exercise 7: Financial Forecasting

### Scenario:

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

### Steps:

#### 1. Understand Recursive Algorithms:

- Explain the concept of recursion and how it can simplify certain problems.

#### 2. Setup:

- Create a method to calculate the future value using a recursive approach.

### 3. Implementation:

- Implement a recursive algorithm to predict future values based on past growth rates.

#### **Forecast.java**

```
package module2.Algorithms_Data_Structures.FinancialForecast;

public class Forecast {

    // Recursive function to calculate future value
    public double forecastValue(double initialValue, double growthRate, int n) {
        if (n == 0) return initialValue;
        return forecastValue(initialValue, growthRate, n - 1) * (1 + growthRate);
    }

    // Optimized: Tail recursion using helper (optional)
    public double forecastValueTail(double initialValue, double growthRate, int n) {
        return forecastHelper(initialValue, growthRate, n);
    }
    private double forecastHelper(double value, double rate, int n) {
        if (n == 0) return value;
        return forecastHelper(value * (1 + rate), rate, n - 1);
    }
}
```

#### **Main.java**

```
package module2.Algorithms_Data_Structures.FinancialForecast;

public class Main {

    public static void main(String[] args) {
        Forecast service = new Forecast();

        double initial = 10000.0;
```



```
double growthRate = 0.10; // 10% annual growth
```

```
int years = 5;
```

```
double result = service.forecastValue(initial, growthRate, years);
```

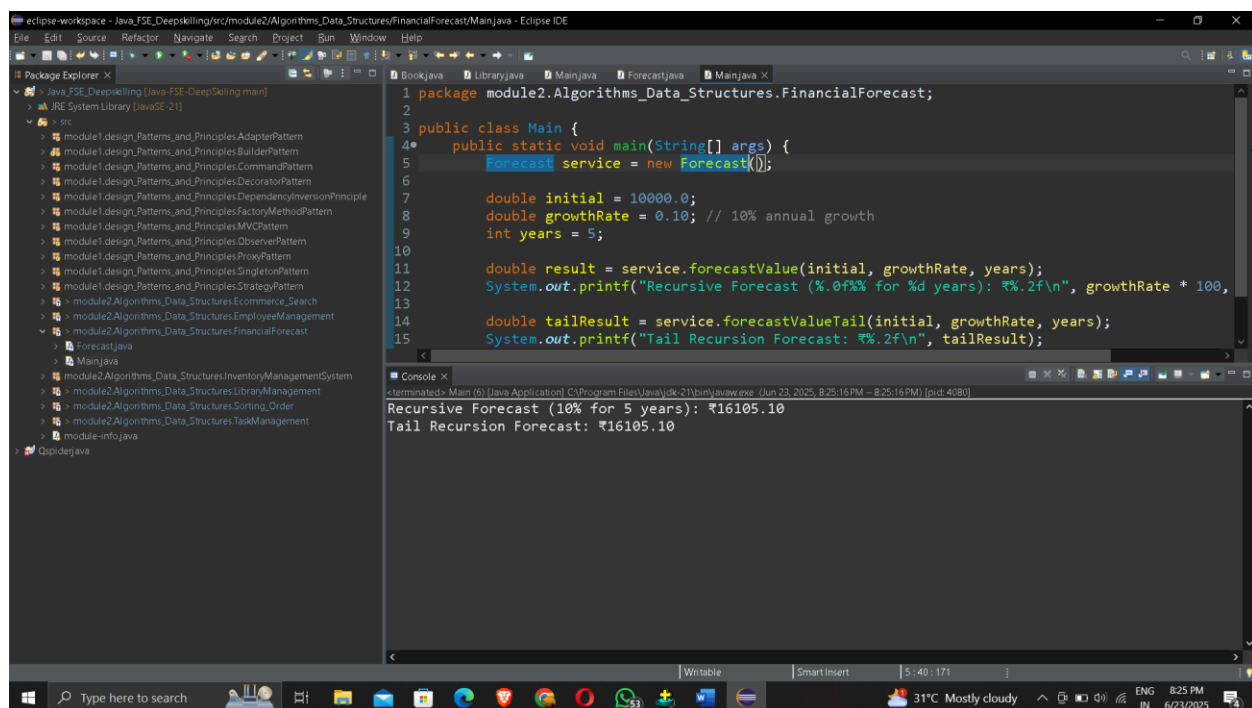
```
System.out.printf("Recursive Forecast (%.0f%% for %d years): ₹%.2f\n", growthRate * 100, years, result);
```

```
double tailResult = service.forecastValueTail(initial, growthRate, years);
```

```
System.out.printf("Tail Recursion Forecast: ₹%.2f\n", tailResult);
```

```
}
```

```
}
```



```
1 package module2.Algorithms_Data_Structures.FinancialForecast;
2
3 public class Main {
4     public static void main(String[] args) {
5         Forecast service = new Forecast();
6
7         double initial = 10000.0;
8         double growthRate = 0.10; // 10% annual growth
9         int years = 5;
10
11         double result = service.forecastValue(initial, growthRate, years);
12         System.out.printf("Recursive Forecast (%.0f%% for %d years): ₹%.2f\n", growthRate * 100,
13
14         double tailResult = service.forecastValueTail(initial, growthRate, years);
15         System.out.printf("Tail Recursion Forecast: ₹%.2f\n", tailResult);
16     }
17 }
```

```
<terminated> Main (6) [Java Application] C:\Program Files\Java\jdk-21\bin\java.exe -Dun 23, 2025, 8:25:16 PM - 8:25:16 PM [pid: 4080]
Recursive Forecast (10% for 5 years): ₹16105.10
Tail Recursion Forecast: ₹16105.10
```