Karthika Vijay - 6377005

Exercise 1: Inventory Management System

Scenario:

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

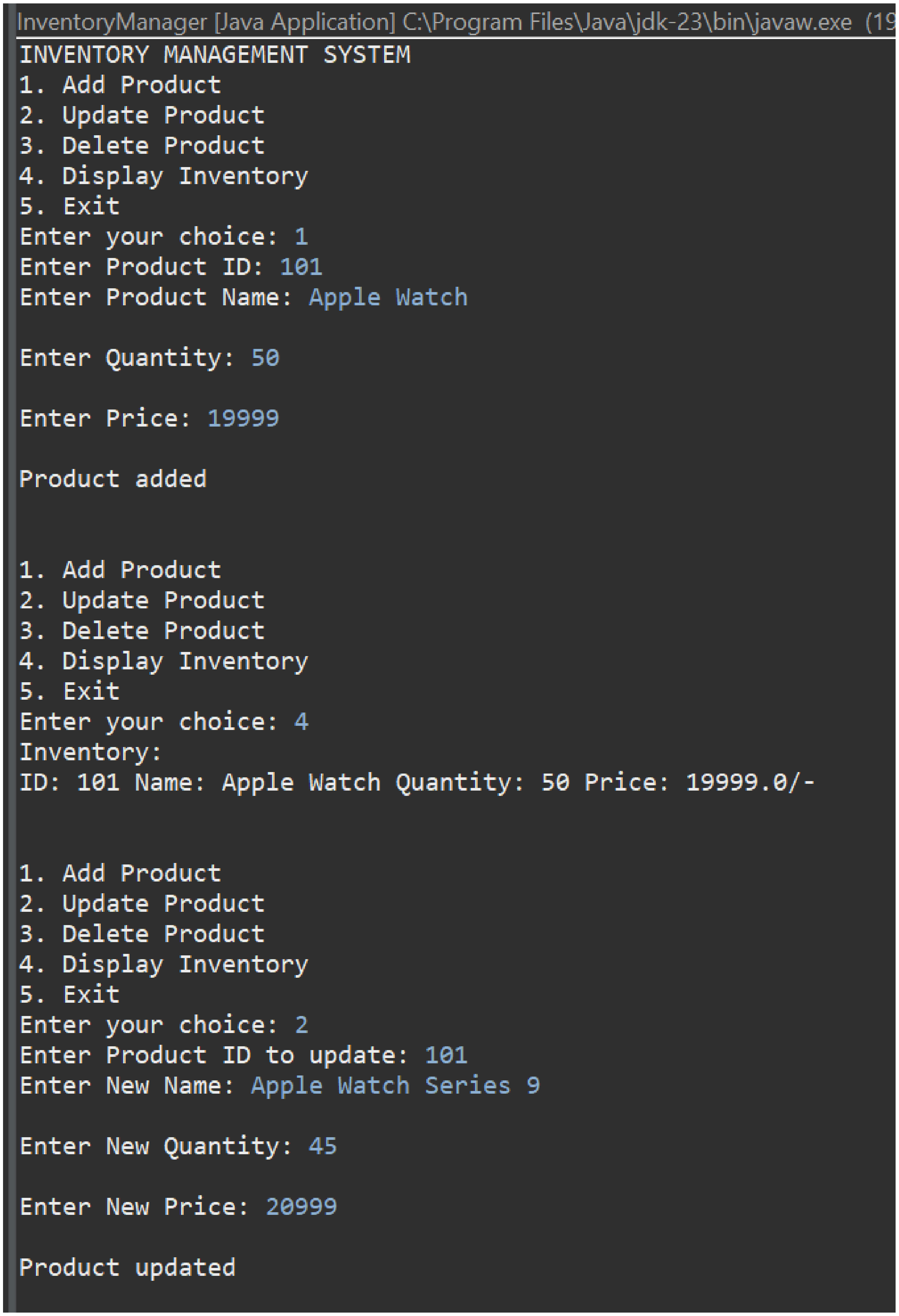
Steps:

1. **Understand the Problem:**
    - Explain why data structures and algorithms are essential in handling large inventories.
      Data structures and algorithms help store, access, and manage large inventories efficiently. With structures like HashMaps, you can retrieve products in constant time. Algorithms help in sorting, searching, and updating records quickly. They ensure fast performance, even as data grows. This makes the system scalable and reliable.
    - Discuss the types of data structures suitable for this problem.

      For inventory management, HashMap is ideal for fast lookups using product IDs. ArrayList can store products in order but is slower for searches. TreeMap keeps data sorted by keys, useful for ordered reports. LinkedList is good for frequent insertions/deletions. The choice depends on required operations.
2. **Setup:**
    - Create a new project for the inventory management system.
3. **Implementation:**
    - Define a class Product with attributes like **productId**, **productName**, **quantity**, and **price**.
    - Choose an appropriate data structure to store the products (e.g., ArrayList, HashMap).
    - Implement methods to add, update, and delete products from the inventory.

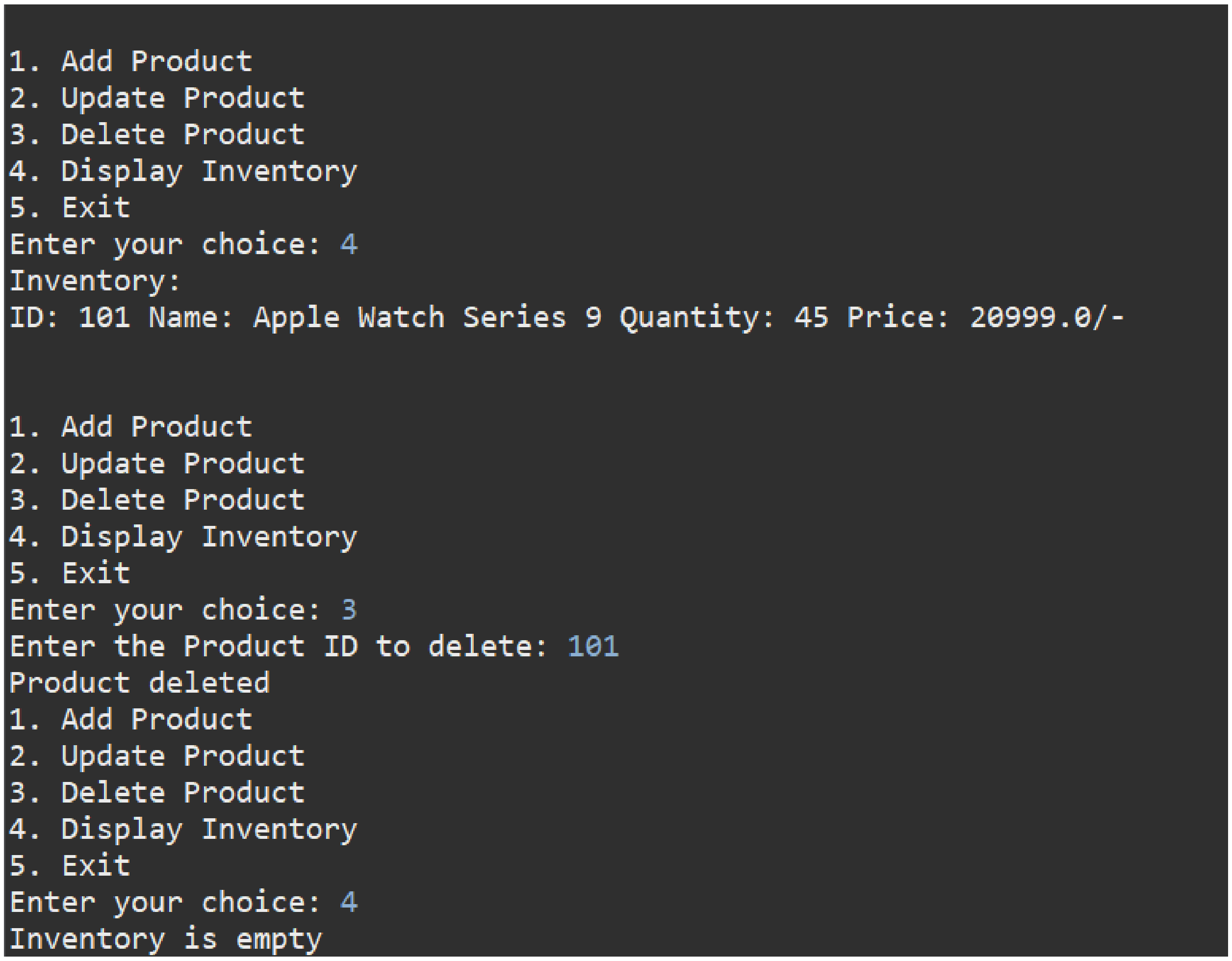
```java
package myproject;
import java.util.*;

public class InventoryManager {
      public static HashMap<Integer, Product> inventory = new HashMap<>();
      public static Scanner sc = new Scanner(System.in);

      public static void main(String[] args) {
            System.out.println("INVENTORY MANAGEMENT SYSTEM");
            while(true) {
                  System.out.println("1. Add Product\n2. Update Product\n3. Delete Product\n4. Display Inventory\n5. Exit");
                  System.out.print("Enter your choice: ");
                  int choice = sc.nextInt();

                  switch(choice) {
```

```java
            case 1:
                add();
                break;

            case 2:
                update();
                break;

            case 3:
                delete();
                break;

            case 4:
                display();
                break;

            case 5:
                System.out.println("Exiting Interface...");
                return;

            default:
                System.out.println("Invalid choice");
            }
        }
    }

    public static void add() {

        System.out.print("Enter Product ID: ");
        int id = sc.nextInt();
        sc.nextLine();

        if(inventory.containsKey(id)) {
            System.out.println("Product ID already exists");
            return;
        }

        System.out.print("Enter Product Name: ");
        String name = sc.nextLine();
        System.out.println();
        System.out.print("Enter Quantity: ");
        int quantity = sc.nextInt();
        System.out.println();
        System.out.print("Enter Price: ");
        double price = sc.nextDouble();
        System.out.println();

        Product p = new Product(id, name, quantity, price);
        inventory.put(id, p);
        System.out.println("Product added");

        System.out.println();
        System.out.println();
    }

    public static void update() {
        System.out.print("Enter Product ID to update: ");
        int id  = sc.nextInt();
        sc.nextLine();

        if(!inventory.containsKey(id)) {
            System.out.println("Product not found");
            return;
        }

        Product p = inventory.get(id);

        System.out.print("Enter New Name: ");
        p.name = sc.nextLine();
        System.out.println();
        System.out.print("Enter New Quantity: ");
        p.quantity = sc.nextInt();
        System.out.println();
        System.out.print("Enter New Price: ");
        p.price = sc.nextDouble();
        System.out.println();

        System.out.println("Product updated");
        System.out.println();
        System.out.println();
    }

    public static void delete() {
        System.out.print("Enter the Product ID to delete: ");
            int id = sc.nextInt();

            if(inventory.remove(id) != null) {
                System.out.println("Product deleted");
            }else {
                System.out.println("Product not found");
            }
        }

        public static void display() {
            if(inventory.isEmpty()) {
                System.out.println("Inventory is empty");
            }else {
                System.out.println("Inventory: ");
                for(Product p : inventory.values()) {
                    System.out.println("ID: " + p.id + " Name: " + p.name + " Quantity: " + p.quantity + " Price: " + p.price + "/-");
                }
                System.out.println();
                System.out.println();
            }
        }
    }

    class Product{
        int id;
        String name;
        int quantity;
        double price;

        Product(int id, String name, int quantity, double price){
            this.id = id;
            this.name = name;
            this.quantity = quantity;
            this.price = price;
        }
    }
```



InventoryManager [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe (19

```
INVENTORY MANAGEMENT SYSTEM
1. Add Product
2. Update Product
3. Delete Product
4. Display Inventory
5. Exit
Enter your choice: 1
Enter Product ID: 101
Enter Product Name: Apple Watch

Enter Quantity: 50

Enter Price: 19999

Product added


1. Add Product
2. Update Product
3. Delete Product
4. Display Inventory
5. Exit
Enter your choice: 4
Inventory:
ID: 101 Name: Apple Watch Quantity: 50 Price: 19999.0/-


1. Add Product
2. Update Product
3. Delete Product
4. Display Inventory
5. Exit
Enter your choice: 2
Enter Product ID to update: 101
Enter New Name: Apple Watch Series 9

Enter New Quantity: 45

Enter New Price: 20999

Product updated
```


```
1. Add Product
2. Update Product
3. Delete Product
4. Display Inventory
5. Exit
Enter your choice: 4
Inventory:
ID: 101 Name: Apple Watch Series 9 Quantity: 45 Price: 20999.0/-


1. Add Product
2. Update Product
3. Delete Product
4. Display Inventory
5. Exit
Enter your choice: 3
Enter the Product ID to delete: 101
Product deleted
1. Add Product
2. Update Product
3. Delete Product
4. Display Inventory
5. Exit
Enter your choice: 4
Inventory is empty
```

4. **Analysis:**

   - Analyze the time complexity of each operation (add, update, delete) in your chosen data structure.
     Add: O(1) average – Inserts product using its ID as the key.
     Update: O(1) average – Accesses and modifies product by ID.
     Delete: O(1) average – Removes product by ID directly.
     Worst case (rare): O(n) due to hash collisions.
     HashMap is highly efficient for large datasets.

   - Discuss how you can optimize these operations.
     Use HashMap for O(1) add, update, and delete by product ID.
     Validate inputs to avoid duplicates or invalid data.
     Use batch processing for bulk updates.
     Cache frequently accessed products.
     Use proper hash functions to reduce collisions.

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.
     Big O notation describes the worst-case time or space complexity of an algorithm as the input size grows. It helps you understand how efficiently an algorithm performs and scales. By comparing complexities like O(1), O(n), or O(log n), you can choose the most efficient algorithm. It ignores constants and focuses on growth rate.

   - Describe the best, average, and worst-case scenarios for search operations.
     Best Case: The element is found in the first comparison – O(1).
     Average Case: The element is found somewhere in the middle – typically O(n/2) ≈ O(n) in linear search.
     Worst Case: The element is not present or is at the last position – O(n) in linear search, O(log n) in binary search.

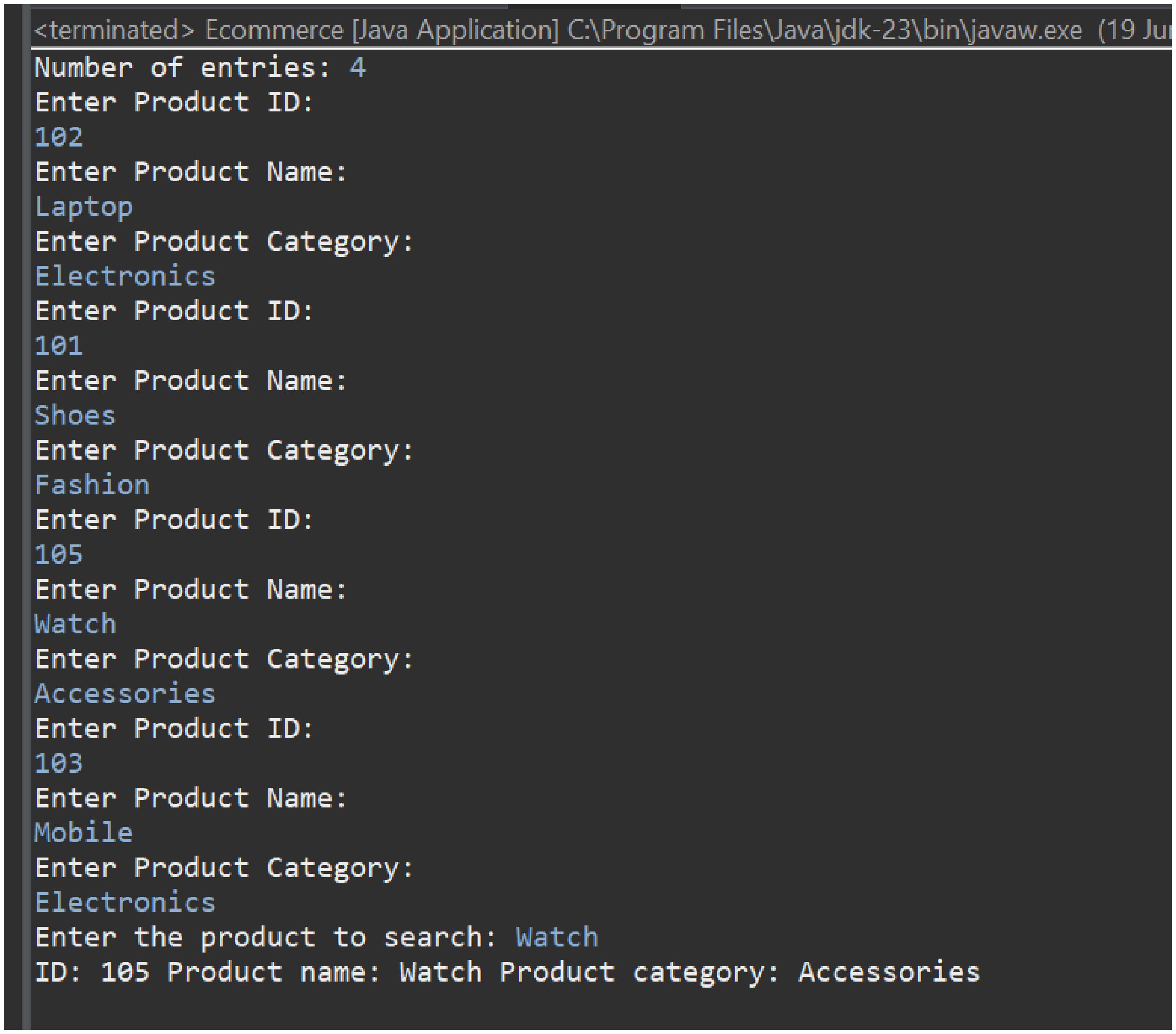
2. **Setup:**
   - Create a class **Product** with attributes for searching, such as **productId, productName**, and **category**.

3. **Implementation:**
   - Implement linear search and binary search algorithms.
   - Store products in an array for linear search and a sorted array for binary search.

LINEAR SEARCH:

```java
package myproject;
import java.util.*;

public class Ecommerce {
        public static Scanner sc = new Scanner(System.in);
        public static void main(String[] args) {
                System.out.print("Number of entries: ");
                int N = sc.nextInt();
                sc.nextLine();

                Item[] item = new Item[N];

                for(int i = 0; i < N; i ++) {
                        System.out.println("Enter Product ID: ");
```

```java
            int id = sc.nextInt();
            sc.nextLine();
            System.out.println("Enter Product Name: ");
            String name = sc.nextLine();
            System.out.println("Enter Product Category: ");
            String category = sc.nextLine();

            item[i] = new Item(id, name, category);
        }

        System.out.print("Enter the product to search: ");
        String name = sc.nextLine();

        Item result = linearSearch(item, name);
        if(result == null) {
            System.out.print("Product not found");
        }else {
            System.out.print("ID: " + result.id + " Product name: " + result.name + " Product category: " + result.category);
        }
    }

    public static Item linearSearch(Item[] item, String name) {
        for(Item it : item) {
            if(it.name.equalsIgnoreCase(name)) {
                return it;
            }
        }
        return null;
    }
}

class Item{
    int id;
    String name;
    String category;

    Item(int id, String name, String category){
        this.id = id;
        this.name = name;
        this.category = category;
    }
}
```

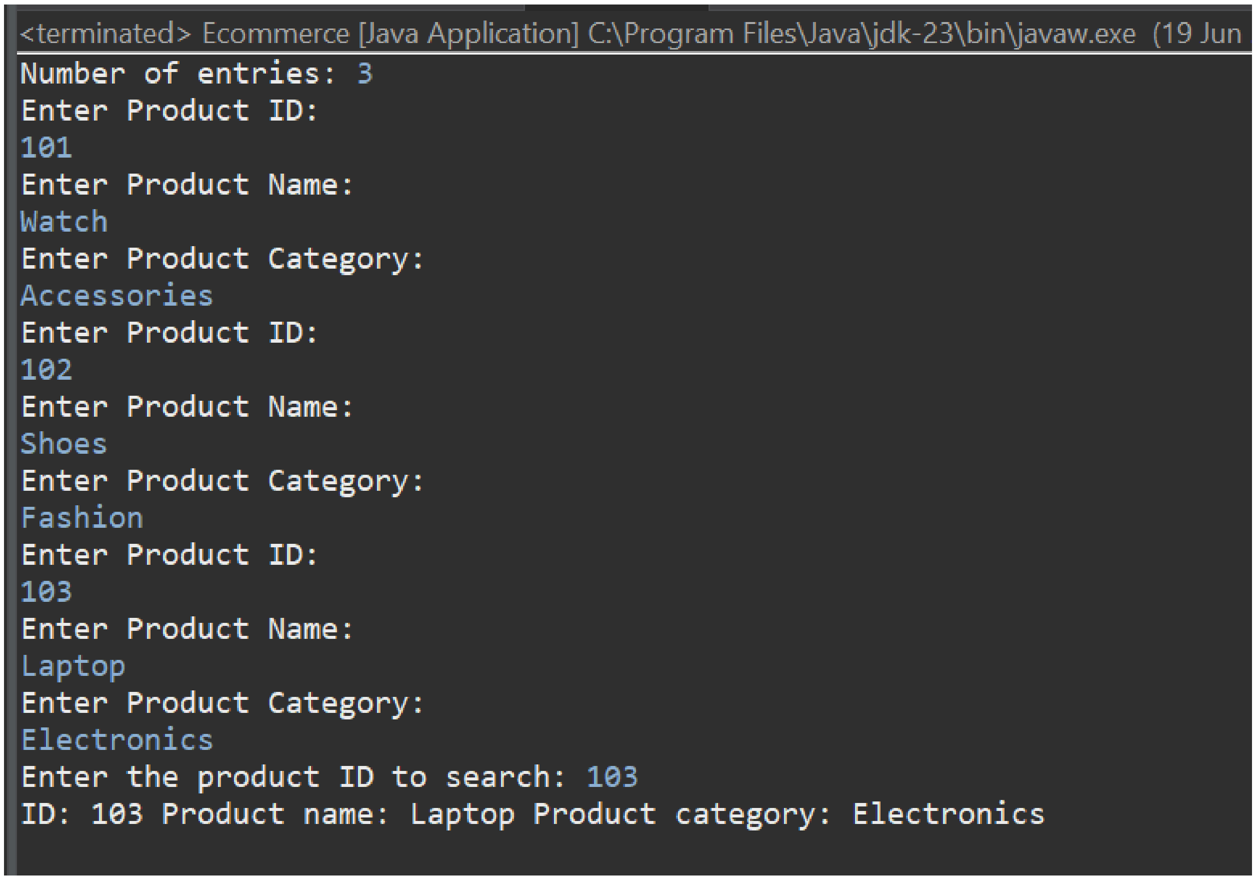


```
<terminated> Ecommerce [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe  (19 Ju
Number of entries: 4
Enter Product ID:
102
Enter Product Name:
Laptop
Enter Product Category:
Electronics
Enter Product ID:
101
Enter Product Name:
Shoes
Enter Product Category:
Fashion
Enter Product ID:
105
Enter Product Name:
Watch
Enter Product Category:
Accessories
Enter Product ID:
103
Enter Product Name:
Mobile
Enter Product Category:
Electronics
Enter the product to search: Watch
ID: 105 Product name: Watch Product category: Accessories
```

BINARY SEARCH:

```java

package myproject;
import java.util.*;

public class Ecommerce {
	public static Scanner sc = new Scanner(System.in);
	public static void main(String[] args) {
		System.out.print("Number of entries: ");
		int N = sc.nextInt();
		sc.nextLine();

		Item[] item = new Item[N];

		for(int i = 0; i < N; i ++) {
			System.out.println("Enter Product ID: ");
			int id = sc.nextInt();
			sc.nextLine();
			System.out.println("Enter Product Name: ");
```

```java
            String name = sc.nextLine();
            System.out.println("Enter Product Category: ");
            String category = sc.nextLine();

            item[i] = new Item(id, name, category);
        }

        System.out.print("Enter the product ID to search: ");
        int target = sc.nextInt();

        Item result = binarySearch(item, target);
        if(result == null) {
            System.out.print("Product not found");
        }else {
            System.out.print("ID: " + result.id + " Product name: " + result.name + " Product category: " + result.category);
        }
    }

    public static Item binarySearch(Item[] item, int target) {
        int left = 0, right = item.length - 1;

        while(left <= right) {
            int mid = left + (right - left) / 2;
            if(item[mid].id == target) {
                return item[mid];
            }else if(item[mid].id < target) {
                left = mid + 1;
            }else {
                right = mid - 1;
            }
        }

        return null;
    }
}

class Item{
    int id;
    String name;
    String category;

    Item(int id, String name, String category){
        this.id = id;
        this.name = name;
        this.category = category;
    }
}
```



```
<terminated> Ecommerce [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe  (19 Jun
Number of entries: 3
Enter Product ID:
101
Enter Product Name:
Watch
Enter Product Category:
Accessories
Enter Product ID:
102
Enter Product Name:
Shoes
Enter Product Category:
Fashion
Enter Product ID:
103
Enter Product Name:
Laptop
Enter Product Category:
Electronics
Enter the product ID to search: 103
ID: 103 Product name: Laptop Product category: Electronics
```

4. **Analysis:**
   - Compare the time complexity of linear and binary search algorithms.
     Linear Search: Time complexity is O(n) in the worst case, as it checks each element one by one.
     Binary Search: Time complexity is O(log n), as it divides the array in half each time.

   - Discuss which algorithm is more suitable for your platform and why.
     Binary search is more suitable if the product list is sorted, as it offers faster search time (O(log n)). If the data is unsorted or frequently changing, linear search is better due to its simplicity and no need for sorting. For small datasets, both work fine, but for large e-commerce platforms, binary search is more efficient after sorting.

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).

     Bubble Sort: Repeatedly swaps adjacent elements if they're in the wrong order. Time: $O(n^2)$. Simple but slow.

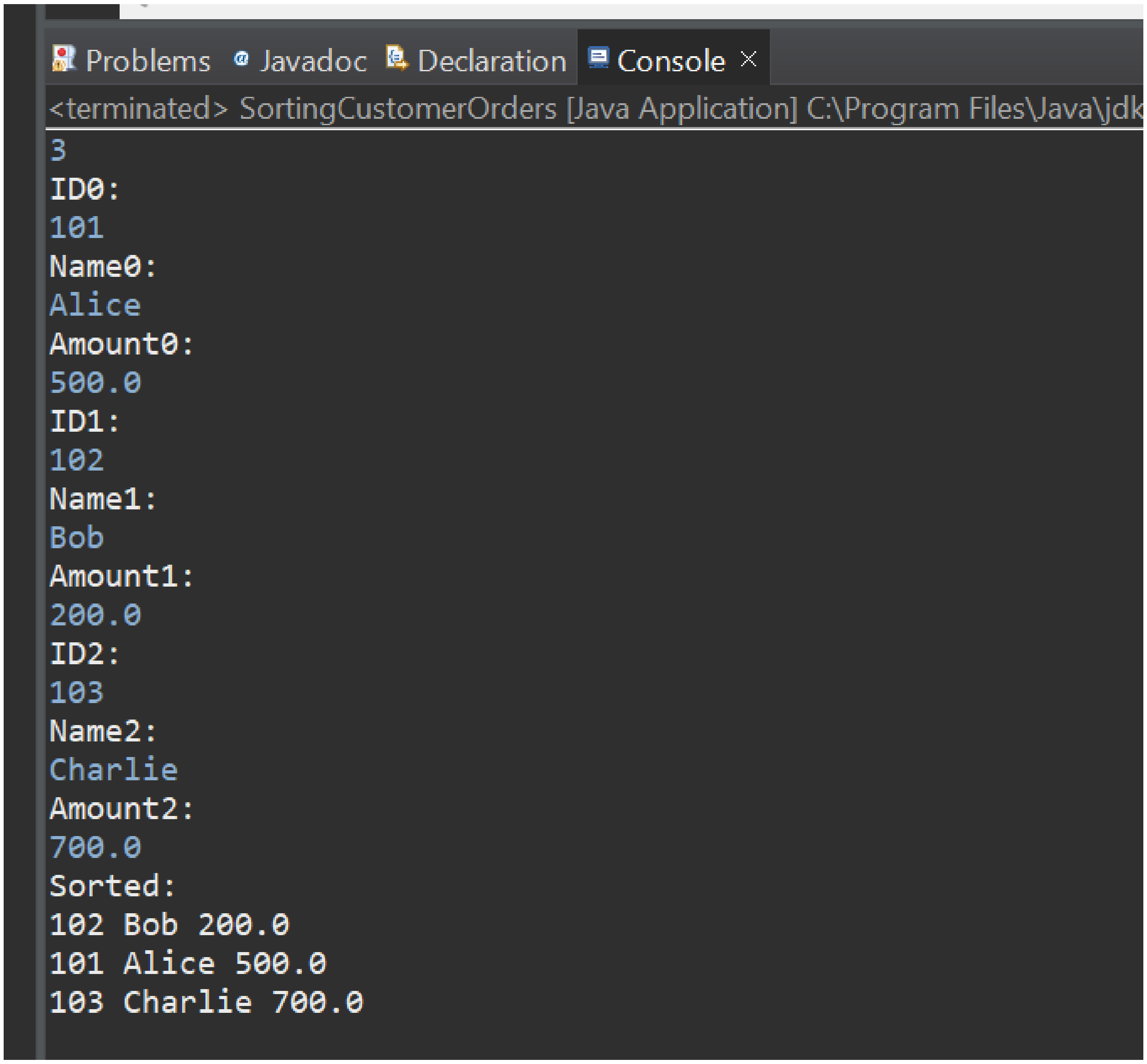
     Insertion Sort: Builds the sorted array one item at a time. Good for small or nearly sorted data. Time: $O(n^2)$.

     Quick Sort: Divides array using a pivot and sorts partitions recursively. Time: $O(n \log n)$ average, $O(n^2)$ worst.

     Merge Sort: Divides array into halves, sorts and merges them. Time: $O(n \log n)$ always. Uses extra space.
2. **Setup:**
   - Create a class **Order** with attributes like **orderId**, **customerName**, and **totalPrice**.
3. **Implementation:**
   - Implement **Bubble Sort** to sort orders by **totalPrice**.

```java
package myproject;
import java.util.*;

class Order{
        int id;
        String name;
        double amount;

        public Order(int id, String name, double amount) {
                this.id = id;
                this.name = name;
                this.amount = amount;
        }
}

public class SortingCustomerOrders {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                int N = sc.nextInt();

                Order[] orders = new Order[N];

                for(int i = 0;i < N; i ++) {
                        System.out.println("ID" + i +": ");
                        int id = sc.nextInt();

                        System.out.println("Name" + i + ": ");
                        String name = sc.next();

                        System.out.println("Amount" + i + ": ");
                        double amount = sc.nextDouble();
```

```

            orders[i] = new Order(id, name, amount);
        }

        bubblesort(N, orders);

        System.out.println("Sorted: ");
        for(Order order : orders) {
            System.out.println(order.id + " " + order.name + " " + order.amount);
        }
    }

    public static void bubblesort(int N, Order[] orders) {
        boolean swap;

        for(int i = 0; i < N - 1; i ++) {
            swap = false;
            for(int j = 0; j < N - 1; j ++) {
                if(orders[j].amount > orders[j + 1].amount) {
                    Order temp = orders[j];
                    orders[j] = orders[j + 1];
                    orders[j + 1] = temp;
                    swap = true;
                }
            }

            if(!swap) {
                break;
            }
        }
    }
}
```

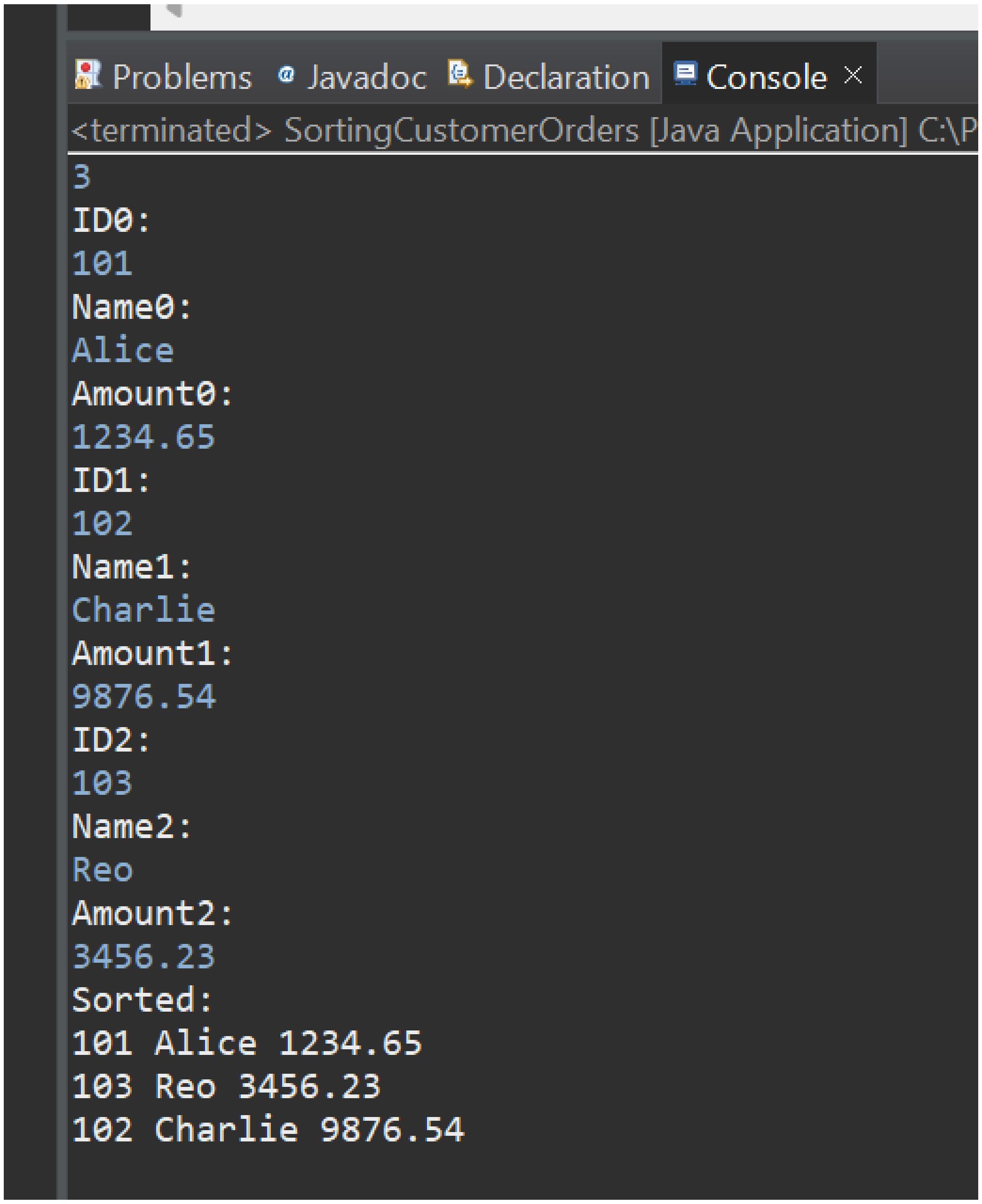


```
<terminated> SortingCustomerOrders [Java Application] C:\Program Files\Java\jdk
3
ID0:
101
Name0:
Alice
Amount0:
500.0
ID1:
102
Name1:
Bob
Amount1:
200.0
ID2:
103
Name2:
Charlie
Amount2:
700.0
Sorted:
102 Bob 200.0
101 Alice 500.0
103 Charlie 700.0
```

- Implement **Quick Sort** to sort orders by **totalPrice**.

```java
package myproject;
import java.util.*;

class Order{
	int id;
	String name;
	double amount;

	public Order(int id, String name, double amount) {
		this.id = id;
		this.name = name;
		this.amount = amount;
	}
```

```java
}

public class SortingCustomerOrders {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int N = sc.nextInt();

        Order[] orders = new Order[N];

        for(int i = 0;i < N; i ++) {
            System.out.println("ID" + i +": ");
            int id = sc.nextInt();

            System.out.println("Name" + i + ": ");
            String name = sc.next();

            System.out.println("Amount" + i + ": ");
            double amount = sc.nextDouble();

            orders[i] = new Order(id, name, amount);
        }

        quicksort(orders, 0, N - 1);

        System.out.println("Sorted: ");
        for(Order order : orders) {
            System.out.println(order.id + " " + order.name + " " + order.amount);
        }
    }

    public static void quicksort(Order[] orders, int low, int high) {
        if(low < high) {
            int pivot = partition(orders, low, high);

            quicksort(orders, low, pivot - 1);
            quicksort(orders, pivot + 1, high);
        }
    }

    public static int partition(Order[] orders, int low, int high) {
        double pivot = orders[high].amount;
        int i = low - 1;

        for(int j = low; j < high; j ++) {
            if(orders[j].amount < pivot) {
                i ++;
                swap(orders, i, j);
            }
        }

        swap(orders, i + 1, high);
        return i + 1;
    }

    public static void swap(Order[] orders, int i, int j) {
          Order temp = orders[i];
          orders[i] = orders[j];
          orders[j] = temp;

    }
}
```



```
<terminated> SortingCustomerOrders [Java Application] C:\P
3
ID0:
101
Name0:
Alice
Amount0:
1234.65
ID1:
102
Name1:
Charlie
Amount1:
9876.54
ID2:
103
Name2:
Reo
Amount2:
3456.23
Sorted:
101 Alice 1234.65
103 Reo 3456.23
102 Charlie 9876.54
```

4. **Analysis:**

   - Compare the performance (time complexity) of Bubble Sort and Quick Sort.
     Bubble Sort has a time complexity of $O(n^2)$ in the average and worst cases.

Quick Sort has O(n log n) on average and O(n²) in the worst case (rare with good pivots).

- Discuss why Quick Sort is generally preferred over Bubble Sort.
  Quick Sort is generally preferred because it is much faster (O(n log n) average) compared to Bubble Sort's O(n²). It works efficiently on large datasets and uses a divide-and-conquer approach.

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.

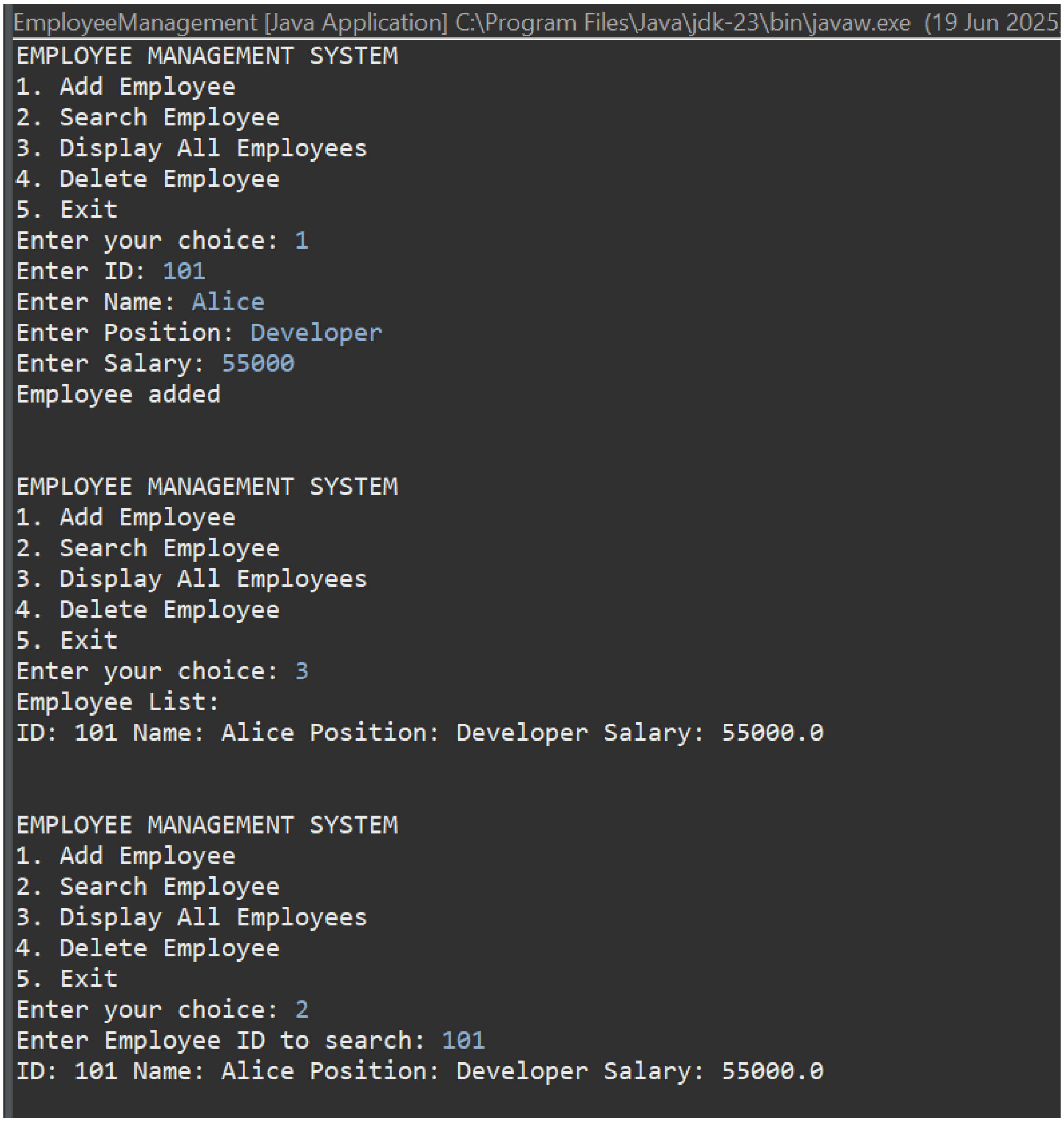
     Arrays are stored in contiguous memory locations, meaning all elements are placed next to each other. This allows fast access using indexing (e.g., arr[2]), as the address is calculated directly.
     Advantages include:

     1. *Fast element access (O(1) time).*
     2. *Easy iteration and cache-friendly.*
     3. *Memory efficiency when size is fixed.*
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.

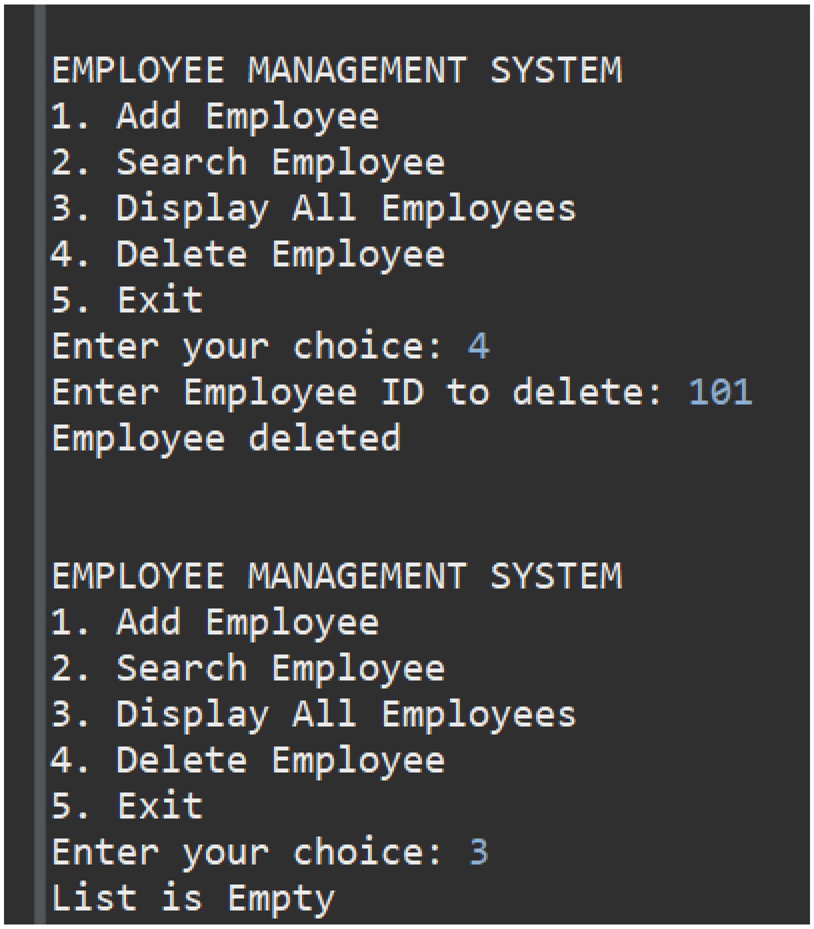
```
package myproject;
import java.util.*;

public class EmployeeManagement {
	static final int MAX = 10;
	static Employee[] emp = new Employee[MAX];
	static int count = 0;
	public static Scanner sc = new Scanner(System.in);
	public static void main(String[] args) {
		while(true) {
			System.out.println("EMPLOYEE MANAGEMENT SYSTEM");
```

```java
                System.out.println("1. Add Employee\n2. Search Employee\n3. Display All Employees\n4. Delete Employee\n5. Exit");
                System.out.print("Enter your choice: ");

                int choice = sc.nextInt();
                switch(choice) {
                case 1:
                    add();
                    break;

                case 2:
                    search();
                    break;

                case 3:
                    display();
                    break;

                case 4:
                    delete();
                    break;

                case 5:
                    System.out.println("Exitinf Interface...");
                    break;

                default:
                    System.out.println("Invalid choice");
                }
            }
        }

        public static void add() {
            if(count >= MAX) {
                System.out.println("List is Full");
                return;
            }

            System.out.print("Enter ID: ");
            int id = sc.nextInt();
            sc.nextLine();
            System.out.print("Enter Name: ");
            String name = sc.nextLine();
            System.out.print("Enter Position: ");
            String position = sc.nextLine();
            System.out.print("Enter Salary: ");
            double salary = sc.nextDouble();

            emp[count ++] = new Employee(id, name, position, salary);
            System.out.println("Employee added");
            System.out.println();
            System.out.println();
        }

        public static void search() {
```

```java
        System.out.print("Enter Employee ID to search: ");
        int id = sc.nextInt();

        for(int i = 0; i < count; i ++) {
            if(emp[i].id == id) {
                System.out.println("ID: " + emp[i].id + " Name: " + emp[i].name + " Position: " + emp[i].position + " Salary: " + emp[i].salary);
                System.out.println();
                System.out.println();
                return;
            }
        }

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

    public static void display() {
        if(count == 0) {
            System.out.println("List is Empty");
            return;
        }

        System.out.println("Employee List: ");
        for(int i = 0; i < count; i ++) {
            System.out.println("ID: " + emp[i].id + " Name: " + emp[i].name + " Position: " + emp[i].position + " Salary: " + emp[i].salary);
        }

        System.out.println();
        System.out.println();
    }

    public static void delete() {
        System.out.print("Enter Employee ID to delete: ");
            int id = sc.nextInt();

            for(int i = 0; i < count; i ++) {
                if(emp[i].id == id) {
                    for(int j = i; j < count - 1;  j ++) {
                        emp[j] = emp[j + 1];
                    }

                    emp[-- count] = null;
                    System.out.println("Employee deleted");
                    System.out.println();
                    System.out.println();
                    return;
                }
            }

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

    class Employee{
        int id;
        String name;
        String position;
        double salary;

        Employee(int id, String name, String position, double salary){
            this.id = id;
            this.name = name;
            this.position = position;
            this.salary = salary;
        }
    }
```



```
EmployeeManagement [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe  (19 Jun 2025
EMPLOYEE MANAGEMENT SYSTEM
1. Add Employee
2. Search Employee
3. Display All Employees
4. Delete Employee
5. Exit
Enter your choice: 1
Enter ID: 101
Enter Name: Alice
Enter Position: Developer
Enter Salary: 55000
Employee added


EMPLOYEE MANAGEMENT SYSTEM
1. Add Employee
2. Search Employee
3. Display All Employees
4. Delete Employee
5. Exit
Enter your choice: 3
Employee List:
ID: 101 Name: Alice Position: Developer Salary: 55000.0


EMPLOYEE MANAGEMENT SYSTEM
1. Add Employee
2. Search Employee
3. Display All Employees
4. Delete Employee
5. Exit
Enter your choice: 2
Enter Employee ID to search: 101
ID: 101 Name: Alice Position: Developer Salary: 55000.0
```



```
EMPLOYEE MANAGEMENT SYSTEM
1. Add Employee
2. Search Employee
3. Display All Employees
4. Delete Employee
5. Exit
Enter your choice: 4
Enter Employee ID to delete: 101
Employee deleted


EMPLOYEE MANAGEMENT SYSTEM
1. Add Employee
2. Search Employee
3. Display All Employees
4. Delete Employee
5. Exit
Enter your choice: 3
List is Empty
```

4. **Analysis:**

    - Analyze the time complexity of each operation (add, search, traverse, delete).
      Add: O(1) if adding at the end; O(n) if shifting is needed.
      Search: O(n) (linear search through array).
      Traverse: O(n) (visit each element once).
      Delete: O(n) due to shifting elements after deletion.

    - Discuss the limitations of arrays and when to use them.
      Limitations of arrays:
        1. *Fixed size — can't grow or shrink dynamically.*
        2. *Insertion/deletion is slow due to shifting.*
        3. *Wastes memory if size is overestimated.*

      Use arrays when the size is known and fast indexed access is required.

Exercise 5: Task Management System

Scenario:

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

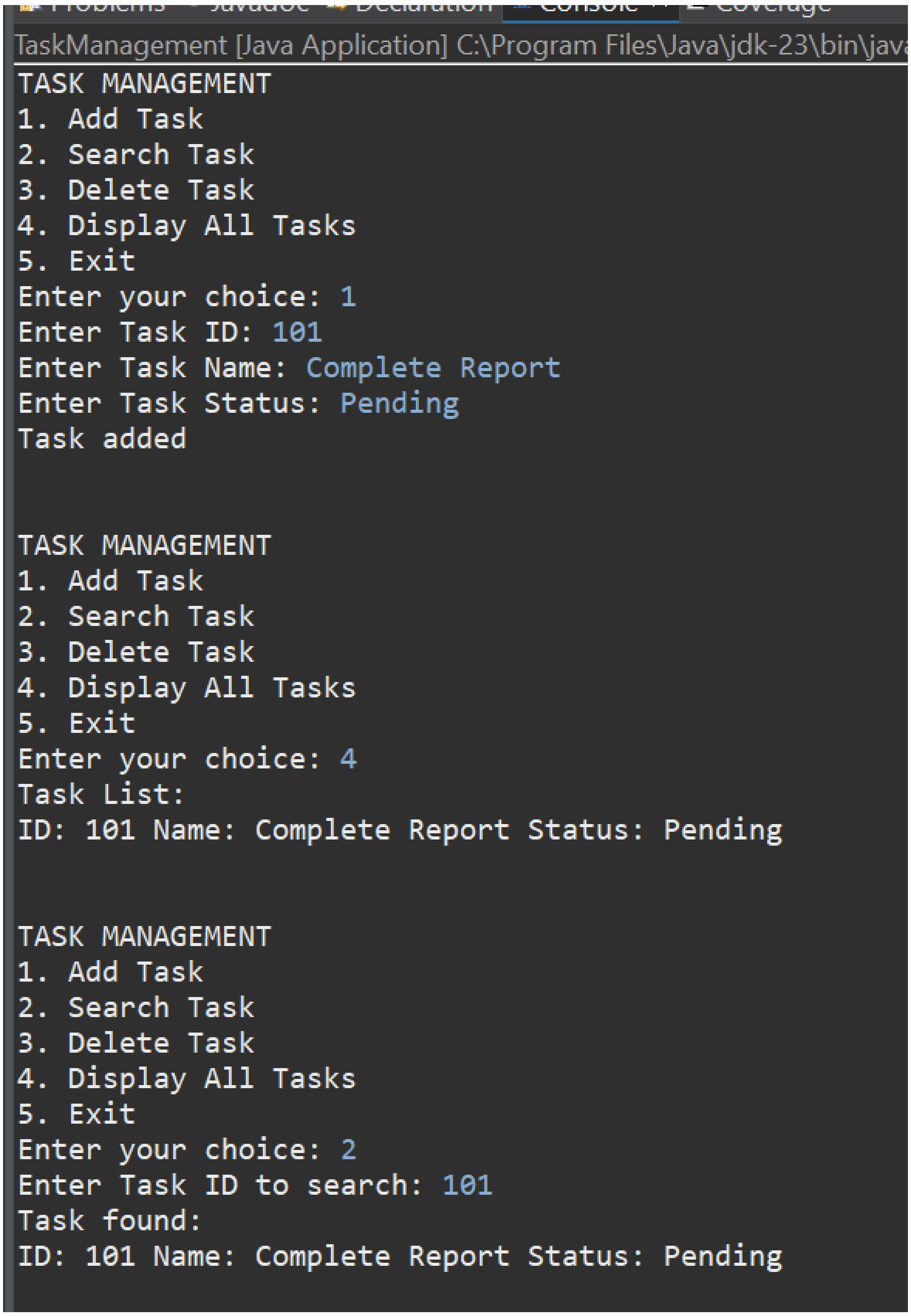
Steps:

1. **Understand Linked Lists:**
   - Explain the different types of linked lists (Singly Linked List, Doubly Linked List).

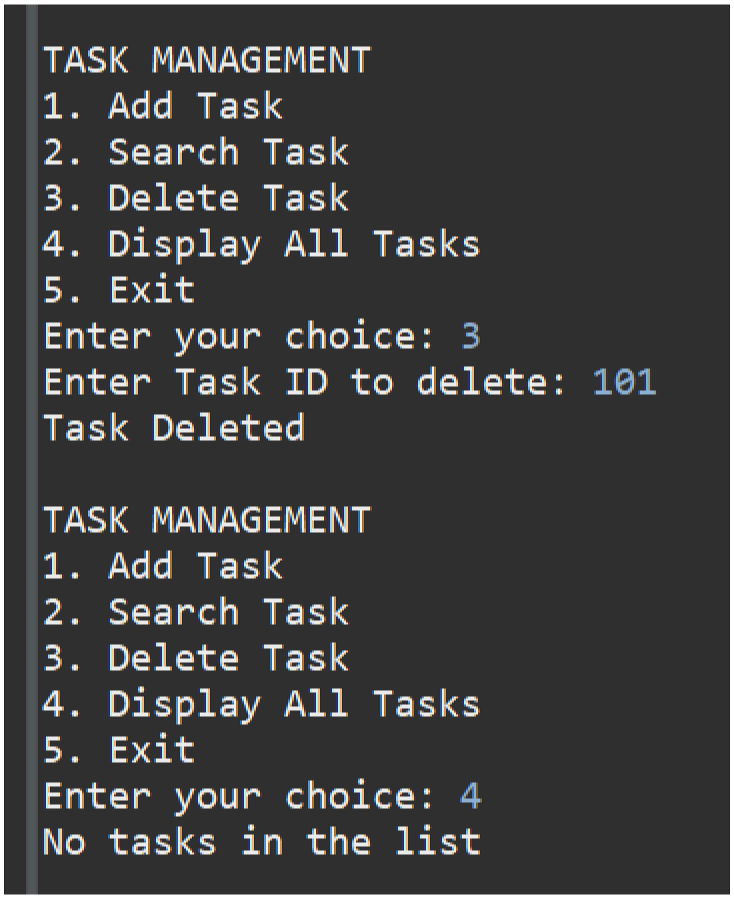
     Singly Linked List has nodes with data and a pointer to the next node only. It's one-directional and simple.

     Doubly Linked List has nodes with data, a pointer to the next node, and a pointer to the previous node. It allows traversal in both directions but uses more memory.
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.

```java
package myproject;
import java.util.*;

public class TaskManagement {
    public static Task head = null;
    public static Scanner sc = new Scanner(System.in);

    public static void main(String args) {
        TaskManagement m = new TaskManagement();
        while(true) {
            System.out.println("TASK MANAGEMENT");
            System.out.println("1. Add Task\n2. Search Task\n3. Delete Task\n4. Display All Tasks\n5. Exit");
            System.out.print("Enter your choice: ");

            int choice = sc.nextInt();
            sc.nextLine();

            switch(choice) {
            case 1:
                System.out.print("Enter Task ID: ");
                int id = sc.nextInt();
                sc.nextLine();
                System.out.print("Enter Task Name: ");
                String name = sc.nextLine();
                System.out.print("Enter Task Status: ");
                String status = sc.nextLine();
                add(id, name, status);
                break;

            case 2:
                System.out.print("Enter Task ID to search: ");
                int search = sc.nextInt();
                search(search);
                break;

            case 3:
                System.out.print("Enter Task ID to delete: ");
                int del = sc.nextInt();
                delete(del);
                break;

            case 4:
                display();
                break;

            case 5:
                System.out.println("Exitinf Interface...");
                break;

            default:
                System.out.println("Invalid choice");
            }
        }
    }

    public static void add(int id, String name, String status) {
        Task task = new Task(id, name, status);

        if(head == null) {
            head = task;
        }else {
            Task temp = head;
            while(temp.next != null) {
                temp = temp.next;
            }

            temp.next = task;
        }

        System.out.println("Task added");
        System.out.println();
        System.out.println();
    }


    public static void search(int id) {
        Task temp = head;
        while(temp != null) {
            if(temp.id == id) {
                System.out.println("Task found:\nID: " +  temp.id + "Name: " + temp.name + " Status: " + temp.status);
                System.out.println();
                System.out.println();
                return;
            }

            temp = temp.next;
        }

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

    public static void delete(int id) {
        if(head == null) {
            System.out.println("No tasks to delete");
            return;
        }

            if(head.id == id) {
                head = head.next;
                System.out.print("Task Deleted");
                System.out.println();
                System.out.println();
                return;
            }

            Task temp = head;
            while(temp.next != null && temp.next.id != id) {
                temp = temp.next;
            }


            if(temp.next == null) {
                System.out.println("Task not found");
                System.out.println();
                System.out.println();
            }else {
                temp.next = temp.next.next;
                System.out.println("Task Deleted");
                System.out.println();
                System.out.println();
            }
        }

        public static void display() {
            if(head == null) {
                System.out.println("No tasks in the list");
                System.out.println();
                System.out.println();
                return;
            }

            Task temp = head;
            System.out.println("Task List:");
            while(temp != null) {
                System.out.println("ID: " + temp.id + " Name: " + temp.name + " Status: " + temp.status);
                temp = temp.next;
            }
        }
    }

    class Task{
          int id;
          String name;
          String status;
          Task next;

          Task(int id, String name, String status){
                this.id = id;
                this.name = name;
                this.status = status;
                this.next = null;
          }
    }
```



```
TaskManagement [Java Application] C:\Program Files\Java\jdk-23\bin\java
TASK MANAGEMENT
1. Add Task
2. Search Task
3. Delete Task
4. Display All Tasks
5. Exit
Enter your choice: 1
Enter Task ID: 101
Enter Task Name: Complete Report
Enter Task Status: Pending
Task added


TASK MANAGEMENT
1. Add Task
2. Search Task
3. Delete Task
4. Display All Tasks
5. Exit
Enter your choice: 4
Task List:
ID: 101 Name: Complete Report Status: Pending


TASK MANAGEMENT
1. Add Task
2. Search Task
3. Delete Task
4. Display All Tasks
5. Exit
Enter your choice: 2
Enter Task ID to search: 101
Task found:
ID: 101 Name: Complete Report Status: Pending
```



```
TASK MANAGEMENT
1. Add Task
2. Search Task
3. Delete Task
4. Display All Tasks
5. Exit
Enter your choice: 3
Enter Task ID to delete: 101
Task Deleted

TASK MANAGEMENT
1. Add Task
2. Search Task
3. Delete Task
4. Display All Tasks
5. Exit
Enter your choice: 4
No tasks in the list
```

4. **Analysis:**
   - Analyze the time complexity of each operation.
     Insertion at beginning (Singly Linked List / Doubly Linked List): O(1)
     Insertion at end: O(n) for Singly Linked List, O(1) for Doubly Linked List.
     Deletion: O(n) for Singly Linked List, O(1) for Doubly Linked List.
     Traversal/Search: O(n) for both.
     Doubly Linked List is faster for deletion if backward traversal is needed.

   - Discuss the advantages of linked lists over arrays for dynamic data.
     Linked lists are dynamic in size, so they don't require predefined space like arrays.
     Insertion and deletion are faster (O(1) at beginning), especially in the middle.
     They avoid memory wastage due to resizing.
     Efficient for applications where frequent insert/delete is needed.

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.

      Linear Search checks each element one by one until the target is found or the list ends. It works on unsorted data and has a time complexity of O(n).

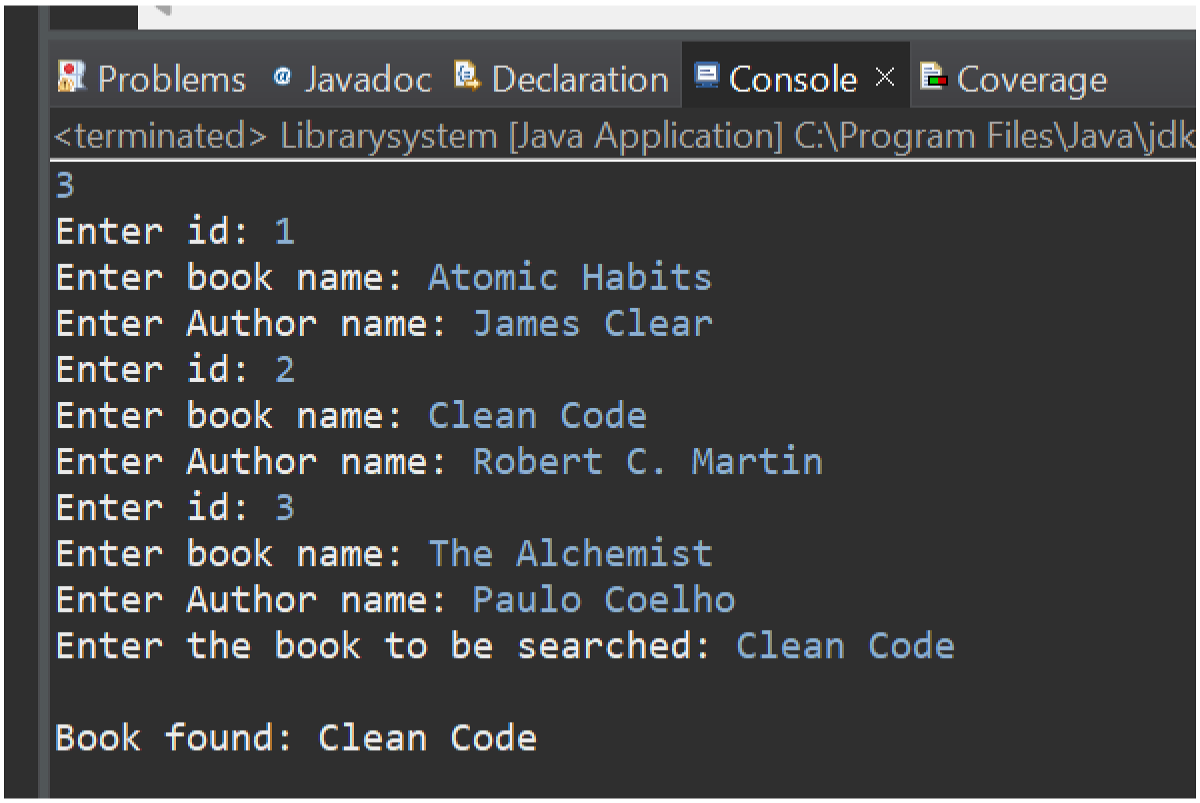
      Binary Search repeatedly divides a sorted list in half to find the target. It is faster with a time complexity of O(log n) but only works on sorted data.

2. **Setup:**
    - Create a class **Book** with attributes like **bookId**, **title**, and **author**.

3. **Implementation:**
    - Implement linear search to find books by title.

```java
package myproject;
import java.util.*;

public class Librarysystem {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int N = sc.nextInt();
        Book[] books = new Book[N];

        for(int i = 0; i < N; i ++) {
            System.out.print("Enter id: ");
            int id = sc.nextInt();
            sc.nextLine();

            System.out.print("Enter book name: ");
            String title = sc.nextLine();

            System.out.print("Enter Author name: ");
            String author = sc.nextLine();

            books[i] = new Book(id, title, author);
        }
        System.out.print("Enter the book to be searched: ");
        String target = sc.nextLine();
        System.out.println();

        Book result = linearsearch(books, target);

        if(result == null) {
            System.out.print("Book not found");
        }else{
            System.out.print("Book found: " + result.title);
        }

    }
```

```

	public static Book linearsearch(Book[] books, String target) {
		for(Book b : books) {
			if(b.title.equalsIgnoreCase(target)) {
				return b;
			}
		}

		return null;
	}
}

class Book{
	int bookid;
	String title;
	String author;

	Book(int bookid, String title, String author){
		this.bookid = bookid;
		this.title = title;
		this.author = author;
	}
}
```

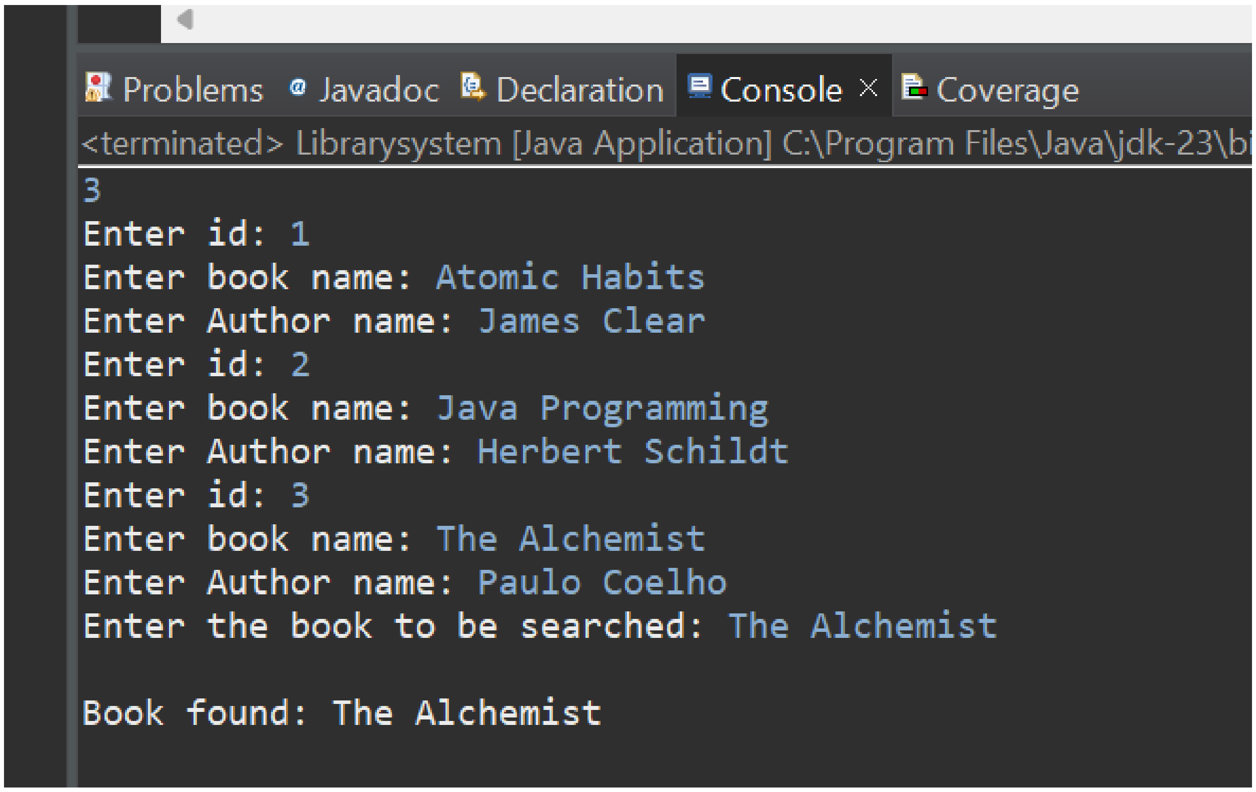


```
<terminated> Librarysystem [Java Application] C:\Program Files\Java\jdk
3
Enter id: 1
Enter book name: Atomic Habits
Enter Author name: James Clear
Enter id: 2
Enter book name: Clean Code
Enter Author name: Robert C. Martin
Enter id: 3
Enter book name: The Alchemist
Enter Author name: Paulo Coelho
Enter the book to be searched: Clean Code

Book found: Clean Code
```

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

```
package myproject;
```

```java
import java.util.*;

public class Librarysystem {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int N = sc.nextInt();
        Book[] books = new Book[N];

        for(int i = 0; i < N; i ++) {
            System.out.print("Enter id: ");
            int id = sc.nextInt();
            sc.nextLine();

            System.out.print("Enter book name: ");
            String title = sc.nextLine();

            System.out.print("Enter Author name: ");
            String author = sc.nextLine();

            books[i] = new Book(id, title, author);
        }
        System.out.print("Enter the book to be searched: ");
        String target = sc.nextLine();
        System.out.println();

        Book result = binarysearch(books, target);

        if(result == null) {
            System.out.print("Book not found");
        }else{
            System.out.print("Book found: " + result.title);
        }

    }

    public static Book binarysearch(Book[] books, String target) {
        int low = 0;
        int high = books.length - 1;

        while(low <= high) {
            int mid = (low + high) / 1;
            int search = books[mid].title.compareToIgnoreCase(target);

            if(search == 0) {
                return books[mid];
            }else if(search < 0) {
                low = mid + 1;
            }else {
                high = mid - 1;
            }
        }

        return null;
    }
}

class Book{
	int bookid;
	String title;
	String author;

	Book(int bookid, String title, String author){
		this.bookid = bookid;
		this.title = title;
		this.author = author;
	}
}
```



```
<terminated> Librarysystem [Java Application] C:\Program Files\Java\jdk-23\bi
3
Enter id: 1
Enter book name: Atomic Habits
Enter Author name: James Clear
Enter id: 2
Enter book name: Java Programming
Enter Author name: Herbert Schildt
Enter id: 3
Enter book name: The Alchemist
Enter Author name: Paulo Coelho
Enter the book to be searched: The Alchemist

Book found: The Alchemist
```

4. **Analysis:**

   - Compare the time complexity of linear and binary search.
     Linear Search has a time complexity of O(n) in the worst case, as it may need to check every element.
     Binary Search has a better time complexity of O(log n) because it halves the search space with each step, but it requires the data to be sorted.

   - Discuss when to use each algorithm based on the data set size and order.
     Use linear search when the dataset is small or unsorted, as it doesn't require any pre-processing.

Use binary search when the dataset is large and sorted, as it is much faster and more efficient in such cases.
If sorting is expensive and only one search is needed, linear search may be preferred.

Exercise 7: Financial Forecasting

Scenario:

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

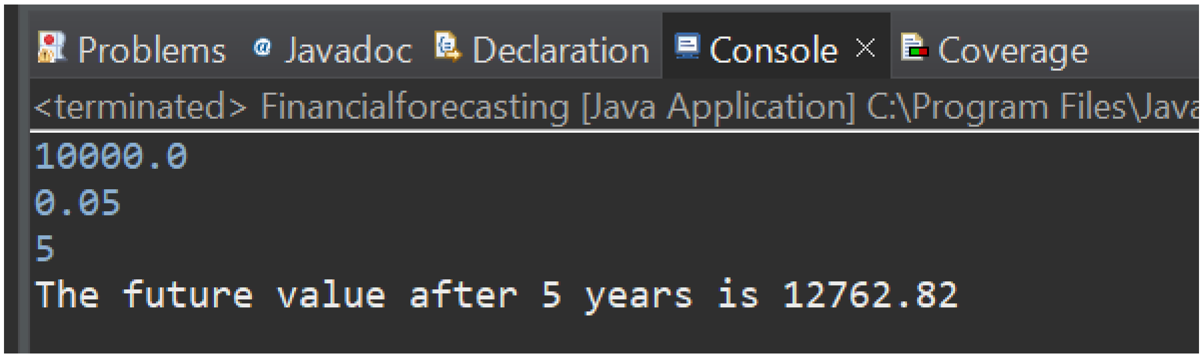
Steps:

1. **Understand Recursive Algorithms:**
   - Explain the concept of recursion and how it can simplify certain problems.

     Recursion is a programming technique where a function calls itself to solve smaller instances of the same problem. It simplifies problems like factorial, Fibonacci, and tree traversal by breaking them into simpler subproblems. Each recursive call reduces the problem size until a base case is reached. This makes the code more readable and easier to write for problems with repetitive patterns.
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.

```java
package myproject;
import java.util.*;

public class Financialforecasting {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		double cv = sc.nextDouble();
		double gr = sc.nextDouble();
		int years = sc.nextInt();

		double fv = futurevalue(cv, gr, years);
		System.out.printf("The future value after %d years is %.2f",
years, fv);
		System.out.println();
	}

	public static double futurevalue(double cv, double gr, int years) {
		if(years == 0) {
			return cv;
		}

		return (gr + 1) * futurevalue(cv, gr, years - 1);
```

```
        }
}
```



```
<terminated> Financialforecasting [Java Application] C:\Program Files\Java
10000.0
0.05
5
The future value after 5 years is 12762.82
```

4. **Analysis:**

   - Discuss the time complexity of your recursive algorithm.
     The time complexity of the recursive futurevalue function is O(n), where n is the number of years. This is because the function makes one recursive call per year until it reaches the base case. There are no repeated calculations, so it's linear in time. However, it uses additional stack space due to recursion.

   - Explain how to optimize the recursive solution to avoid excessive computation.
     To optimize a recursive solution, you can use memoization to store results of subproblems and avoid recalculating them. This reduces time complexity by reusing previously computed values. Alternatively, convert recursion to an iterative approach, which is often more efficient and avoids stack overflow. For example, using Math.pow() in the future value problem is a direct and faster method.