**Exercise 5: Task Management System**

**Task Management System.java**

**package** mypackage;

**import** java.util.Scanner;

// Task Node for Singly Linked List

**class** SinglyTask {

**int** taskId;

String taskName;

String status;

SinglyTask next;

**public** SinglyTask(**int** taskId, String taskName, String status) {

**this**.taskId = taskId;

**this**.taskName = taskName;

**this**.status = status;

**this**.next = **null**;

}

}

// Task Node for Doubly Linked List

**class** DoublyTask {

**int** taskId;

String taskName;

String status;

DoublyTask next, prev;

**public** DoublyTask(**int** taskId, String taskName, String status) {

**this**.taskId = taskId;

**this**.taskName = taskName;

**this**.status = status;

**this**.next = **null**;

**this**.prev = **null**;

}

}

// Task Node for Circular Linked List

**class** CircularTask {

**int** taskId;

String taskName;

String status;

CircularTask next;

**public** CircularTask(**int** taskId, String taskName, String status) {

**this**.taskId = taskId;

**this**.taskName = taskName;

**this**.status = status;

**this**.next = **null**;

}

}

**public** **class** TaskManagementSystem {

**static** Scanner *sc* = **new** Scanner(System.***in***);

**static** SinglyTask *singlyHead* = **null**;

**static** DoublyTask *doublyHead* = **null**;

**static** CircularTask *circularHead* = **null**;

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

**while** (**true**) {

System.***out***.println("\nSelect Linked List Type:");

System.***out***.println("1. Singly Linked List");

System.***out***.println("2. Doubly Linked List");

System.***out***.println("3. Circular Linked List");

System.***out***.println("4. Exit");

System.***out***.print("Enter your choice: ");

**int** type = *sc*.nextInt();

**if** (type == 4) **break**;

**while** (**true**) {

System.***out***.println("\n1. Add Task");

System.***out***.println("2. Search Task");

System.***out***.println("3. Display Tasks");

System.***out***.println("4. Delete Task");

System.***out***.println("5. Change Linked List Type");

System.***out***.print("Enter your choice: ");

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

**if** (choice == 5) **break**;

**switch** (type) {

**case** 1: *handleSingly*(choice); **break**;

**case** 2: *handleDoubly*(choice); **break**;

**case** 3: *handleCircular*(choice); **break**;

}

}

}

}

// Singly Linked List Operations

**public** **static** **void** handleSingly(**int** choice) {

**switch** (choice) {

**case** 1:

System.***out***.print("Enter Task ID: ");

**int** id1 = *sc*.nextInt(); *sc*.nextLine();

System.***out***.print("Enter Task Name: ");

String name1 = *sc*.nextLine();

System.***out***.print("Enter Status: ");

String status1 = *sc*.nextLine();

SinglyTask newNode = **new** SinglyTask(id1, name1, status1);

**if** (*singlyHead* == **null**) *singlyHead* = newNode;

**else** {

SinglyTask curr = *singlyHead*;

**while** (curr.next != **null**) curr = curr.next;

curr.next = newNode;

}

System.***out***.println("Task added!");

**break**;

**case** 2:

System.***out***.print("Enter Task ID to search: ");

**int** searchId1 = *sc*.nextInt();

SinglyTask curr1 = *singlyHead*;

**while** (curr1 != **null**) {

**if** (curr1.taskId == searchId1) {

System.***out***.println("Task found: " + curr1.taskId + " " + curr1.taskName + " " + curr1.status);

**return**;

}

curr1 = curr1.next;

}

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

**break**;

**case** 3:

SinglyTask temp1 = *singlyHead*;

**if** (temp1 == **null**) {

System.***out***.println("No tasks.");

**return**;

}

**while** (temp1 != **null**) {

System.***out***.println(temp1.taskId + " " + temp1.taskName + " " + temp1.status);

temp1 = temp1.next;

}

**break**;

**case** 4:

System.***out***.print("Enter Task ID to delete: ");

**int** delId1 = *sc*.nextInt();

**if** (*singlyHead* == **null**) {

System.***out***.println("No tasks.");

**return**;

}

**if** (*singlyHead*.taskId == delId1) {

*singlyHead* = *singlyHead*.next;

System.***out***.println("Task deleted!");

**return**;

}

SinglyTask prev1 = *singlyHead*;

SinglyTask del1 = *singlyHead*.next;

**while** (del1 != **null**) {

**if** (del1.taskId == delId1) {

prev1.next = del1.next;

System.***out***.println("Task deleted!");

**return**;

}

prev1 = del1;

del1 = del1.next;

}

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

**break**;

}

}

// Doubly Linked List Operations

**public** **static** **void** handleDoubly(**int** choice) {

**switch** (choice) {

**case** 1:

System.***out***.print("Enter Task ID: ");

**int** id2 = *sc*.nextInt(); *sc*.nextLine();

System.***out***.print("Enter Task Name: ");

String name2 = *sc*.nextLine();

System.***out***.print("Enter Status: ");

String status2 = *sc*.nextLine();

DoublyTask newNode = **new** DoublyTask(id2, name2, status2);

**if** (*doublyHead* == **null**) *doublyHead* = newNode;

**else** {

DoublyTask curr = *doublyHead*;

**while** (curr.next != **null**) curr = curr.next;

curr.next = newNode;

newNode.prev = curr;

}

System.***out***.println("Task added!");

**break**;

**case** 2:

System.***out***.print("Enter Task ID to search: ");

**int** searchId2 = *sc*.nextInt();

DoublyTask curr2 = *doublyHead*;

**while** (curr2 != **null**) {

**if** (curr2.taskId == searchId2) {

System.***out***.println("Task found: " + curr2.taskId + " " + curr2.taskName + " " + curr2.status);

**return**;

}

curr2 = curr2.next;

}

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

**break**;

**case** 3:

DoublyTask temp2 = *doublyHead*;

**if** (temp2 == **null**) {

System.***out***.println("No tasks.");

**return**;

}

**while** (temp2 != **null**) {

System.***out***.println(temp2.taskId + " " + temp2.taskName + " " + temp2.status);

temp2 = temp2.next;

}

**break**;

**case** 4:

System.***out***.print("Enter Task ID to delete: ");

**int** delId2 = *sc*.nextInt();

**if** (*doublyHead* == **null**) {

System.***out***.println("No tasks.");

**return**;

}

**if** (*doublyHead*.taskId == delId2) {

*doublyHead* = *doublyHead*.next;

**if** (*doublyHead* != **null**) *doublyHead*.prev = **null**;

System.***out***.println("Task deleted!");

**return**;

}

DoublyTask currDel = *doublyHead*.next;

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

**if** (currDel.taskId == delId2) {

currDel.prev.next = currDel.next;

**if** (currDel.next != **null**)

currDel.next.prev = currDel.prev;

System.***out***.println("Task deleted!");

**return**;

}

currDel = currDel.next;

}

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

**break**;

}

}

// Circular Linked List Operations

**public** **static** **void** handleCircular(**int** choice) {

**switch** (choice) {

**case** 1:

System.***out***.print("Enter Task ID: ");

**int** id3 = *sc*.nextInt(); *sc*.nextLine();

System.***out***.print("Enter Task Name: ");

String name3 = *sc*.nextLine();

System.***out***.print("Enter Status: ");

String status3 = *sc*.nextLine();

CircularTask newNode = **new** CircularTask(id3, name3, status3);

**if** (*circularHead* == **null**) {

*circularHead* = newNode;

*circularHead*.next = *circularHead*; // make circular

} **else** {

CircularTask temp = *circularHead*;

**while** (temp.next != *circularHead*) temp = temp.next;

temp.next = newNode;

newNode.next = *circularHead*;

}

System.***out***.println("Task added!");

**break**;

**case** 2:

System.***out***.print("Enter Task ID to search: ");

**int** searchId3 = *sc*.nextInt();

**if** (*circularHead* == **null**) {

System.***out***.println("No tasks.");

**return**;

}

CircularTask tempSearch = *circularHead*;

**do** {

**if** (tempSearch.taskId == searchId3) {

System.***out***.println("Task found: " + tempSearch.taskId + " " + tempSearch.taskName + " " + tempSearch.status);

**return**;

}

tempSearch = tempSearch.next;

} **while** (tempSearch != *circularHead*);

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

**break**;

**case** 3:

**if** (*circularHead* == **null**) {

System.***out***.println("No tasks.");

**return**;

}

CircularTask temp3 = *circularHead*;

**do** {

System.***out***.println(temp3.taskId + " " + temp3.taskName + " " + temp3.status);

temp3 = temp3.next;

} **while** (temp3 != *circularHead*);

**break**;

**case** 4:

System.***out***.print("Enter Task ID to delete: ");

**int** delId3 = *sc*.nextInt();

**if** (*circularHead* == **null**) {

System.***out***.println("No tasks.");

**return**;

}

// Only one node case

**if** (*circularHead*.taskId == delId3 && *circularHead*.next == *circularHead*) {

*circularHead* = **null**;

System.***out***.println("Task deleted!");

**return**;

}

CircularTask prev = *circularHead*;

CircularTask curr = *circularHead*.next;

**boolean** found = **false**;

**do** {

**if** (curr.taskId == delId3) {

prev.next = curr.next;

**if** (curr == *circularHead*) *circularHead* = curr.next;

found = **true**;

**break**;

}

prev = curr;

curr = curr.next;

} **while** (prev.next != *circularHead*);

**if** (found)

System.***out***.println("Task deleted!");

**else**

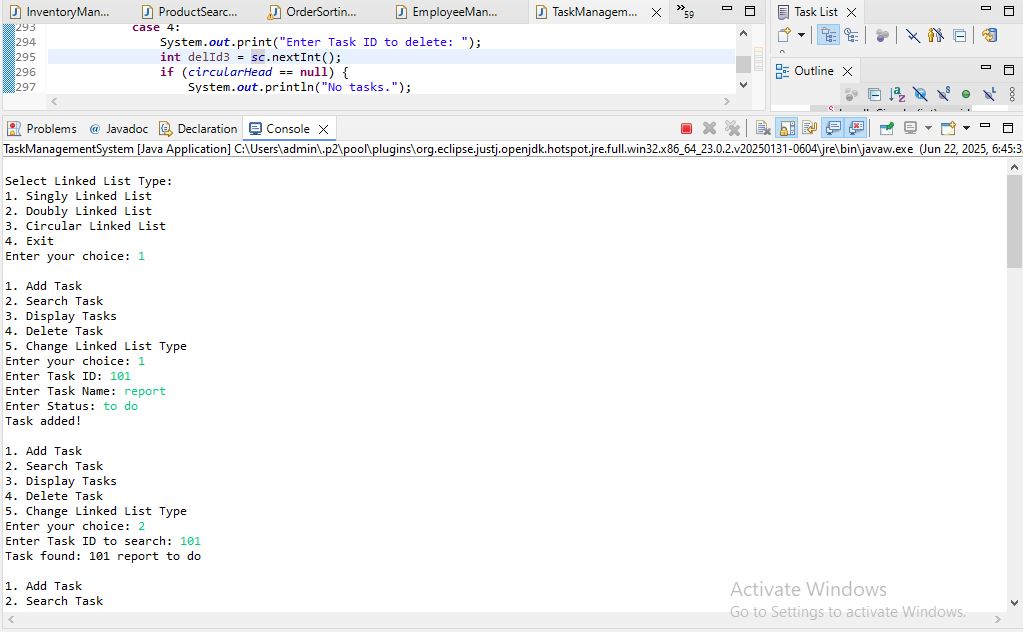
System.***out***.println("Task not found!");

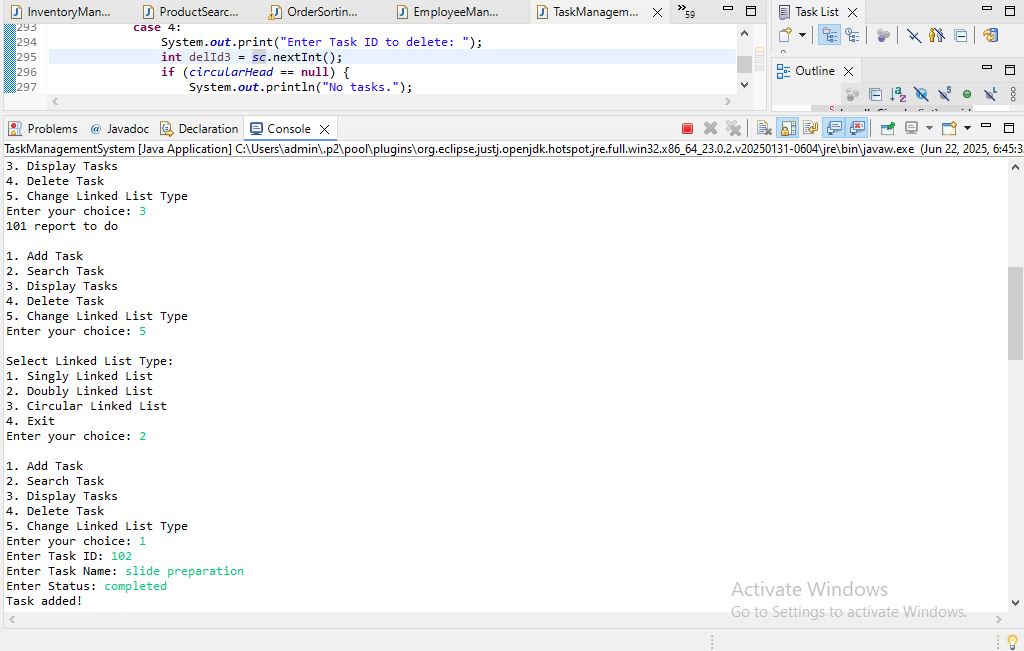
**break**;

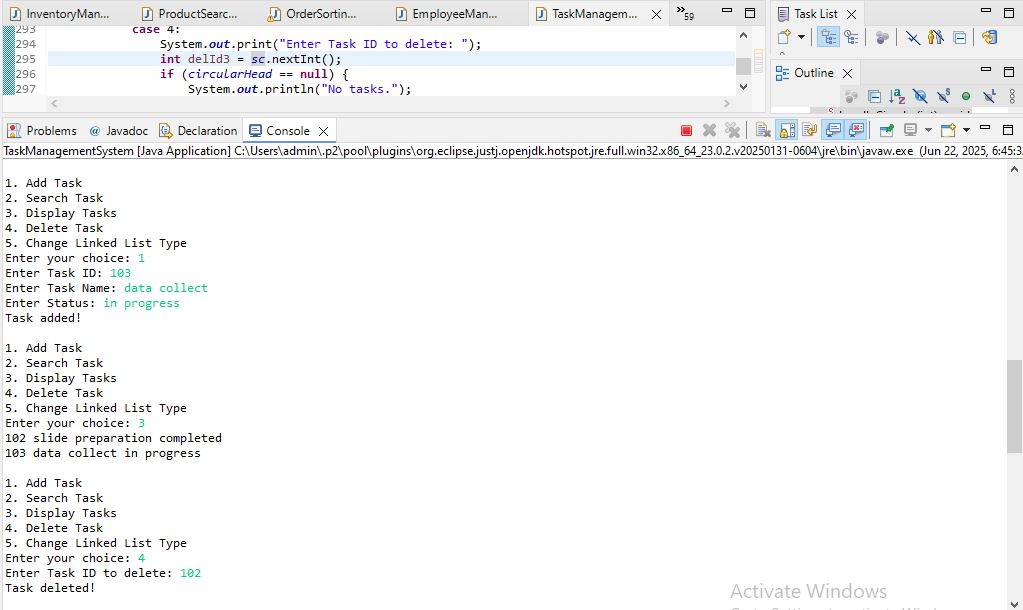
}

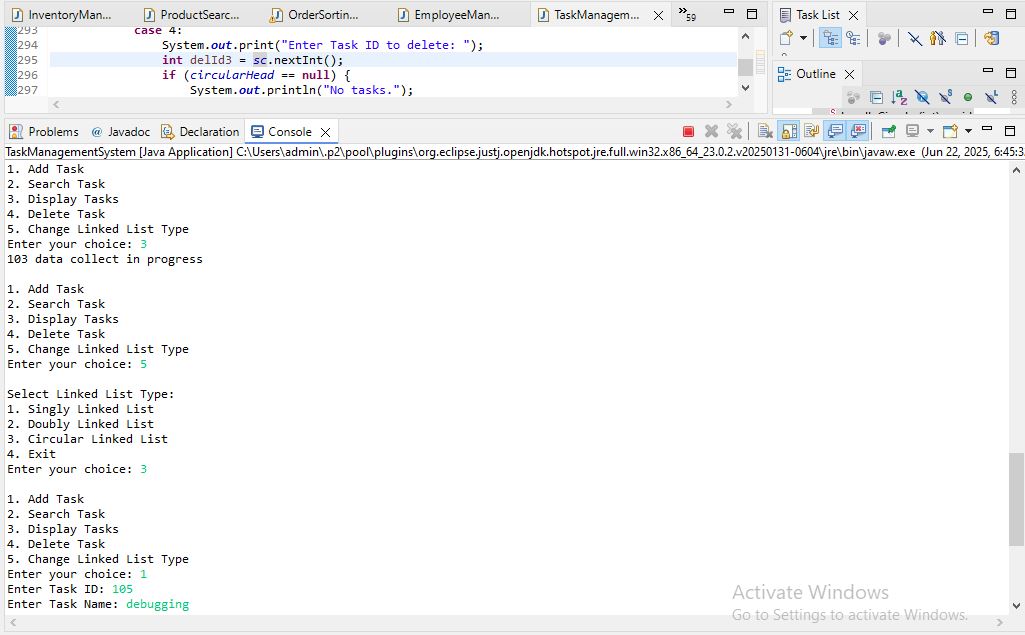
}

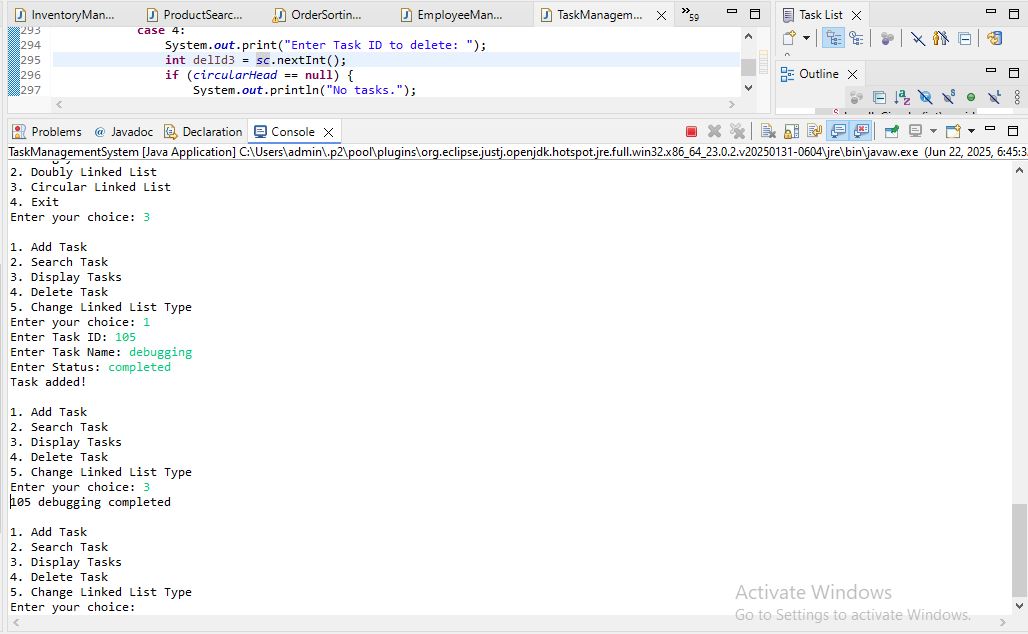
}











We are developing a **Task Management System** where the user can choose to store tasks using:

* **Singly Linked List**
* **Doubly Linked List**
* **Circular Linked List**

This system allows users to:

* Add tasks
* Search tasks
* Display all tasks
* Delete tasks

## **Linked List Concepts**

### A. **Singly Linked List**

* Every node has:
  + Data: taskId, taskName, status
  + Pointer to the next node
* The last node points to null.

#### **Advantages:**

* Dynamic memory allocation
* Easy insertion and deletion at beginning

#### **Disadvantages:**

* Cannot traverse backwards
* Deleting specific node requires traversal

### B. **Doubly Linked List**

* Every node has:
  + Data: taskId, taskName, status
  + Pointer to the next node (next)
  + Pointer to the previous node (prev)

#### **Advantages:**

* Can traverse forward and backward
* Deletion is easier since we have reference to previous node

#### **Disadvantages:**

* Extra memory needed for prev pointer
* Slightly more complex insertion/deletion

### C. **Circular Linked List**

* Similar to singly linked list but:
  + Last node’s next points back to head.
* No node has null pointer.

#### **Advantages:**

* Continuous traversal possible
* Useful for round-robin scheduling, buffering systems

#### **Disadvantages:**

* Special care needed in deletion & insertion
* Termination condition must be carefully handled (since no null)

## **Setup**

We created 3 classes for node representation:

| **Linked List Type** | **Class Name** |
| --- | --- |
| Singly Linked List | SinglyTask |
| Doubly Linked List | DoublyTask |
| Circular Linked List | CircularTask |

Each class has:

* taskId (int)
* taskName (String)
* status (String)

The main class TaskManagementSystem handles all the operations.

## **Implementation**

The program works as follows:

1 User first selects Linked List type  
2Then user performs operations:

* Add Task
* Search Task
* Display Tasks
* Delete Task

### **Operations Implemented for All 3 Types:**

| **Operation** | **Description** |
| --- | --- |
| **Add Task** | Insert new task at end of list |
| **Search Task** | Search for taskId |
| **Traverse Tasks** | Display all tasks |
| **Delete Task** | Delete task by taskId |

## **Time Complexity Analysis**

| **Operation** | **Time Complexity** | **Explanation** |
| --- | --- | --- |
| **Add** | O(n) | Traverse to end |
| **Search** | O(n) | Linear search |
| **Traverse** | O(n) | Visit each node |
| **Delete** | O(n) | Search + remove |

Complexity is same for all 3 types because of linear structure.

## **Why Linked List instead of Array?**

| **Array** | **Linked List** |
| --- | --- |
| Fixed size | Dynamic size |
| Insertion/Deletion costly (shift needed) | Easy insertion/deletion |
| Random access possible (O(1)) | Only sequential access (O(n)) |
| Memory allocation contiguous | Non-contiguous allocation |

* **Linked Lists** are more suitable when:
  + The number of tasks is not fixed.
  + Frequent insertions and deletions are expected.

## **When to use each Linked List?**

| **Linked List Type** | **Use Case** |
| --- | --- |
| **Singly Linked List** | Simple dynamic lists |
| **Doubly Linked List** | When backward traversal is needed |
| **Circular Linked List** | When round-robin style operations or endless cycles are needed |