CodInClub

Powered By BridgeLabz

Best Practices for Data Structure - LinkedList

1. **Head & Tail Management**: Always maintain the `head` (and `tail` in doubly and circular lists) to avoid traversing the entire list when accessing the first or last elements.
2. **Null Checks**: Before performing operations like deletion or traversal, check if the list is empty to prevent errors.
3. **Efficient Insertion/Deletion**: Insert at the beginning or end for O(1) time complexity. For operations in the middle, ensure proper pointer updates to maintain list integrity.
4. **Memory Management**: Properly nullify pointers (`next`, `prev`) when deleting nodes to prevent memory leaks, especially in languages without garbage collection.
5. **Boundary Handling**: Carefully handle edge cases like inserting/deleting at the head, tail, or middle of the list, ensuring correct pointer updates.
6. **Avoid Infinite Loops** (Circular Lists): Implement conditions to stop traversal after one complete cycle to avoid infinite loops.
7. **Modular Code**: Break operations into small, reusable functions for better readability and maintainability.
8. **Keep Code Simple**: Focus on clarity over complexity. Avoid unnecessary traversals and complex logic unless required for your use case.

1. Singly Linked List: Student Record Management

Problem Statement: Create a program to manage student records using a singly linked list. Each node will store information about a student, including their `Roll Number`, `Name`, `Age`, and `Grade`. Implement the following operations:

1. Add a new student record at the beginning, end, or at a specific position.
2. Delete a student record by `Roll Number`.
3. Search for a student record by `Roll Number`.
4. Display all student records.
5. Update a student's grade based on their `Roll Number`.

Hint:

- Use a singly linked list where each node contains student information and a pointer to the next node.
- The head of the list will represent the first student, and the last node’s next pointer will be `null`.
- Update the `next` pointers when inserting or deleting nodes.

```java
import java.util.Scanner;

class Student {
    int rollNo;
    String name;
    int age;
    char grade;
    Student next;

    Student(int rollNo, String name, int age, char grade) {
        this.rollNo = rollNo;
        this.name = name;
        this.age = age;
        this.grade = grade;
        this.next = null;
    }
}

public class StudentLinkedList {
    static Student head = null;

    public static void addAtBeginning(Student newStudent) {
        newStudent.next = head;
        head = newStudent;
    }

    public static void addAtEnd(Student newStudent) {
        if (head == null) {
            head = newStudent;
            return;
        }
        Student temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = newStudent;
    }

    public static void addAtPosition(Student newStudent, int position) {
```

```
        if (position <= 1 || head == null) {
            addAtBeginning(newStudent);
            return;
        }
        Student temp = head;
        for (int i = 1; i < position - 1 && temp.next != null; i++) {
            temp = temp.next;
        }
        newStudent.next = temp.next;
        temp.next = newStudent;
    }

    public static void deleteByRollNo(int rollNo) {
        if (head == null) return;
        if (head.rollNo == rollNo) {
            head = head.next;
            return;
        }
        Student temp = head;
        while (temp.next != null && temp.next.rollNo != rollNo) {
            temp = temp.next;
        }
        if (temp.next != null) {
            temp.next = temp.next.next;
        }
    }

    public static void searchByRollNo(int rollNo) {
        Student temp = head;
        while (temp != null) {
            if (temp.rollNo == rollNo) {
                System.out.println("Found: " + temp.name + ", Age: " +
temp.age + ", Grade: " + temp.grade);
                return;
            }
            temp = temp.next;
        }
        System.out.println("Student with Roll No " + rollNo + " not
found.");
    }
```

```
    public static void updateGrade(int rollNo, char newGrade) {
        Student temp = head;
        while (temp != null) {
            if (temp.rollNo == rollNo) {
                temp.grade = newGrade;
                System.out.println("Grade updated.");
                return;
            }
            temp = temp.next;
        }
        System.out.println("Roll number not found.");
    }

    public static void displayAll() {
        Student temp = head;
        while (temp != null) {
            System.out.println("Roll No: " + temp.rollNo + ", Name: " +
temp.name + ", Age: " + temp.age + ", Grade: " + temp.grade);
            temp = temp.next;
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int choice;
        do {
            System.out.println("\n1.Add at Beginning\n2.Add at End\n3.Add
at Position\n4.Delete by Roll No\n5.Search by Roll No\n6.Update
Grade\n7.Display All\n0.Exit");
            choice = sc.nextInt();
            switch (choice) {
                case 1:
                case 2:
                case 3:
                    System.out.print("Enter Roll No, Name, Age, Grade: ");
                    int roll = sc.nextInt();
                    String name = sc.next();
                    int age = sc.nextInt();
                    char grade = sc.next().charAt(0);
                    Student newStudent = new Student(roll, name, age,
grade);
```

```
                    if (choice == 1) addAtBeginning(newStudent);
                    else if (choice == 2) addAtEnd(newStudent);
                    else {
                        System.out.print("Enter position: ");
                        int pos = sc.nextInt();
                        addAtPosition(newStudent, pos);
                    }
                    break;
                case 4:
                    System.out.print("Enter Roll No to delete: ");
                    deleteByRollNo(sc.nextInt());
                    break;
                case 5:
                    System.out.print("Enter Roll No to search: ");
                    searchByRollNo(sc.nextInt());
                    break;
                case 6:
                    System.out.print("Enter Roll No and new Grade: ");
                    int r = sc.nextInt();
                    char g = sc.next().charAt(0);
                    updateGrade(r, g);
                    break;
                case 7:
                    displayAll();
                    break;
            }
        } while (choice != 0);
        sc.close();
    }
}
```

---

2. Doubly Linked List: Movie Management System

Problem Statement: Implement a movie management system using a doubly linked list. Each node will represent a movie and contain `Movie Title`, `Director`, `Year of Release`, and `Rating`. Implement the following functionalities:

1. Add a movie record at the beginning, end, or at a specific position.
2. Remove a movie record by `Movie Title`.
3. Search for a movie record by `Director` or `Rating`.
4. Display all movie records in both forward and reverse order.
5. Update a movie's `Rating` based on the `Movie Title`.

Hint:

- Use a doubly linked list where each node has two pointers: one pointing to the next node and the other to the previous node.
- Maintain pointers to both the head and tail for easier insertion and deletion at both ends.
- For reverse display, start from the tail and traverse backward using the `prev` pointers.

```
import java.util.Scanner;

class Movie {
    String title;
    String director;
    int year;
    double rating;
    Movie prev;
    Movie next;

    public Movie(String title, String director, int year, double rating) {
        this.title = title;
        this.director = director;
        this.year = year;
        this.rating = rating;
    }
}

public class MovieManagementSystem {
    static Movie head = null;
    static Movie tail = null;

    public static void addMovieAtEnd(String title, String director, int
year, double rating) {
        Movie newMovie = new Movie(title, director, year, rating);
        if (head == null) {
```

```
            head = tail = newMovie;
        } else {
            tail.next = newMovie;
            newMovie.prev = tail;
            tail = newMovie;
        }
    }

    public static void addMovieAtBeginning(String title, String director,
int year, double rating) {
        Movie newMovie = new Movie(title, director, year, rating);
        if (head == null) {
            head = tail = newMovie;
        } else {
            newMovie.next = head;
            head.prev = newMovie;
            head = newMovie;
        }
    }

    public static void addMovieAtPosition(String title, String director,
int year, double rating, int pos) {
        if (pos == 1) {
            addMovieAtBeginning(title, director, year, rating);
            return;
        }

        Movie newMovie = new Movie(title, director, year, rating);
        Movie temp = head;
        int count = 1;
        while (temp != null && count < pos - 1) {
            temp = temp.next;
            count++;
        }

        if (temp == null || temp.next == null) {
            addMovieAtEnd(title, director, year, rating);
        } else {
            newMovie.next = temp.next;
            newMovie.prev = temp;
            temp.next.prev = newMovie;
```

```
            temp.next = newMovie;
        }
    }

    public static void removeMovieByTitle(String title) {
        Movie temp = head;
        while (temp != null) {
            if (temp.title.equalsIgnoreCase(title)) {
                if (temp == head) {
                    head = head.next;
                    if (head != null) head.prev = null;
                } else if (temp == tail) {
                    tail = tail.prev;
                    if (tail != null) tail.next = null;
                } else {
                    temp.prev.next = temp.next;
                    temp.next.prev = temp.prev;
                }
                System.out.println("Movie removed.");
                return;
            }
            temp = temp.next;
        }
        System.out.println("Movie not found.");
    }

    public static void searchByDirector(String director) {
        Movie temp = head;
        while (temp != null) {
            if (temp.director.equalsIgnoreCase(director)) {
                System.out.println(temp.title + " (" + temp.year + ") -
Rating: " + temp.rating);
            }
            temp = temp.next;
        }
    }

    public static void searchByRating(double rating) {
        Movie temp = head;
        while (temp != null) {
            if (temp.rating >= rating) {
```

```
                System.out.println(temp.title + " (" + temp.year + ") -
Directed by: " + temp.director);
            }
            temp = temp.next;
        }
    }

    public static void displayForward() {
        Movie temp = head;
        while (temp != null) {
            System.out.println(temp.title + " | " + temp.director + " | " +
temp.year + " | " + temp.rating);
            temp = temp.next;
        }
    }

    public static void displayReverse() {
        Movie temp = tail;
        while (temp != null) {
            System.out.println(temp.title + " | " + temp.director + " | " +
temp.year + " | " + temp.rating);
            temp = temp.prev;
        }
    }

    public static void updateRating(String title, double newRating) {
        Movie temp = head;
        while (temp != null) {
            if (temp.title.equalsIgnoreCase(title)) {
                temp.rating = newRating;
                System.out.println("Rating updated.");
                return;
            }
            temp = temp.next;
        }
        System.out.println("Movie not found.");
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int choice;
```

```
        do {
            System.out.println("\n--- Movie Management System ---");
            System.out.println("1. Add movie at beginning");
            System.out.println("2. Add movie at end");
            System.out.println("3. Add movie at specific position");
            System.out.println("4. Remove movie by title");
            System.out.println("5. Search movie by director");
            System.out.println("6. Search movie by rating");
            System.out.println("7. Display all movies (Forward)");
            System.out.println("8. Display all movies (Reverse)");
            System.out.println("9. Update movie rating");
            System.out.println("0. Exit");
            System.out.print("Enter choice: ");
            choice = sc.nextInt();
            sc.nextLine(); // consume newline

            String title, director;
            int year, pos;
            double rating;

            switch (choice) {
                case 1:
                    System.out.print("Enter title: ");
                    title = sc.nextLine();
                    System.out.print("Enter director: ");
                    director = sc.nextLine();
                    System.out.print("Enter year: ");
                    year = sc.nextInt();
                    System.out.print("Enter rating: ");
                    rating = sc.nextDouble();
                    addMovieAtBeginning(title, director, year, rating);
                    break;

                case 2:
                    System.out.print("Enter title: ");
                    title = sc.nextLine();
                    System.out.print("Enter director: ");
                    director = sc.nextLine();
                    System.out.print("Enter year: ");
                    year = sc.nextInt();
                    System.out.print("Enter rating: ");
```

```
                rating = sc.nextDouble();
                addMovieAtEnd(title, director, year, rating);
                break;

            case 3:
                System.out.print("Enter title: ");
                title = sc.nextLine();
                System.out.print("Enter director: ");
                director = sc.nextLine();
                System.out.print("Enter year: ");
                year = sc.nextInt();
                System.out.print("Enter rating: ");
                rating = sc.nextDouble();
                System.out.print("Enter position: ");
                pos = sc.nextInt();
                addMovieAtPosition(title, director, year, rating, pos);
                break;

            case 4:
                System.out.print("Enter title to remove: ");
                title = sc.nextLine();
                removeMovieByTitle(title);
                break;

            case 5:
                System.out.print("Enter director name: ");
                director = sc.nextLine();
                searchByDirector(director);
                break;

            case 6:
                System.out.print("Enter minimum rating: ");
                rating = sc.nextDouble();
                searchByRating(rating);
                break;

            case 7:
                displayForward();
                break;

            case 8:
```

```
                    displayReverse();
                    break;

                case 9:
                    System.out.print("Enter movie title to update: ");
                    title = sc.nextLine();
                    System.out.print("Enter new rating: ");
                    rating = sc.nextDouble();
                    updateRating(title, rating);
                    break;

                case 0:
                    System.out.println("Exiting system...");
                    break;

                default:
                    System.out.println("Invalid choice.");
            }
        } while (choice != 0);
        sc.close();
    }
}
```

---

3. Circular Linked List: Task Scheduler

Problem Statement: Create a task scheduler using a circular linked list. Each node in the list represents a task with `Task ID`, `Task Name`, `Priority`, and `Due Date`. Implement the following functionalities:

1. Add a task at the beginning, end, or at a specific position in the circular list.
2. Remove a task by `Task ID`.
3. View the current task and move to the next task in the circular list.
4. Display all tasks in the list starting from the head node.
5. Search for a task by `Priority`.

Hint:

- Use a circular linked list where the last node's `next` pointer points back to the first node, creating a circular structure.
- Ensure that the list loops when traversed from the head node, so tasks can be revisited in a circular manner.
- When deleting or adding tasks, maintain the circular nature by updating the appropriate `next` pointers.

```
import java.util.Scanner;

class Task {
    int taskId;
    String taskName;
    int priority;
    String dueDate;
    Task next;

    public Task(int taskId, String taskName, int priority,
String dueDate) {
        this.taskId = taskId;
        this.taskName = taskName;
        this.priority = priority;
        this.dueDate = dueDate;
    }
}

public class TaskScheduler {
    static Task head = null;
    static Task tail = null;
    static Task current = null;

    public static void addTaskAtEnd(int id, String name, int
priority, String date) {
        Task newTask = new Task(id, name, priority, date);
        if (head == null) {
            head = tail = newTask;
            newTask.next = head;
```

```
        } else {
            tail.next = newTask;
            newTask.next = head;
            tail = newTask;
        }
    }

    public static void addTaskAtBeginning(int id, String name,
int priority, String date) {
        Task newTask = new Task(id, name, priority, date);
        if (head == null) {
            head = tail = newTask;
            newTask.next = head;
        } else {
            newTask.next = head;
            head = newTask;
            tail.next = head;
        }
    }

    public static void addTaskAtPosition(int id, String name,
int priority, String date, int pos) {
        if (pos == 1) {
            addTaskAtBeginning(id, name, priority, date);
            return;
        }

        Task newTask = new Task(id, name, priority, date);
        Task temp = head;
        int count = 1;

        while (count < pos - 1 && temp.next != head) {
            temp = temp.next;
            count++;
        }
```

```
        newTask.next = temp.next;
        temp.next = newTask;

        if (temp == tail) {
            tail = newTask;
        }
    }

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

        Task temp = head;
        Task prev = tail;

        do {
            if (temp.taskId == id) {
                if (temp == head) {
                    head = head.next;
                    tail.next = head;
                } else if (temp == tail) {
                    tail = prev;
                    tail.next = head;
                } else {
                    prev.next = temp.next;
                }

                if (current == temp) {
                    current = current.next;
                }

                System.out.println("Task removed.");
```

```
                return;
            }

            prev = temp;
            temp = temp.next;
        } while (temp != head);

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

    public static void viewCurrentTask() {
        if (current == null) {
            if (head == null) {
                System.out.println("No tasks scheduled.");
                return;
            }
            current = head;
        }

        System.out.println("Current Task: ID=" + current.taskId
+ ", Name=" + current.taskName + ", Priority=" +
current.priority + ", Due Date=" + current.dueDate);
        current = current.next;
    }

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

        Task temp = head;
        do {
            System.out.println("ID: " + temp.taskId + ", Name: "
+ temp.taskName + ", Priority: " + temp.priority + ", Due Date:
```

```
" + temp.dueDate);
            temp = temp.next;
        } while (temp != head);
    }

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

        Task temp = head;
        boolean found = false;

        do {
            if (temp.priority == p) {
                System.out.println("ID: " + temp.taskId + ",
Name: " + temp.taskName + ", Due Date: " + temp.dueDate);
                found = true;
            }
            temp = temp.next;
        } while (temp != head);

        if (!found) {
            System.out.println("No tasks found with priority " +
p);
        }
    }

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

        do {
            System.out.println("\n--- Task Scheduler ---");
```

```
            System.out.println("1. Add task at beginning");
            System.out.println("2. Add task at end");
            System.out.println("3. Add task at specific
position");
            System.out.println("4. Remove task by ID");
            System.out.println("5. View current task and move to
next");
            System.out.println("6. Display all tasks");
            System.out.println("7. Search tasks by priority");
            System.out.println("0. Exit");
            System.out.print("Enter your choice: ");
            choice = sc.nextInt();
            sc.nextLine(); // consume newline

            int id, priority, pos;
            String name, dueDate;

            switch (choice) {
                case 1:
                    System.out.print("Enter Task ID: ");
                    id = sc.nextInt();
                    sc.nextLine();
                    System.out.print("Enter Task Name: ");
                    name = sc.nextLine();
                    System.out.print("Enter Priority: ");
                    priority = sc.nextInt();
                    sc.nextLine();
                    System.out.print("Enter Due Date: ");
                    dueDate = sc.nextLine();
                    addTaskAtBeginning(id, name, priority,
dueDate);
                    break;

                case 2:
                    System.out.print("Enter Task ID: ");
```

```
                    id = sc.nextInt();
                    sc.nextLine();
                    System.out.print("Enter Task Name: ");
                    name = sc.nextLine();
                    System.out.print("Enter Priority: ");
                    priority = sc.nextInt();
                    sc.nextLine();
                    System.out.print("Enter Due Date: ");
                    dueDate = sc.nextLine();
                    addTaskAtEnd(id, name, priority, dueDate);
                    break;

                case 3:
                    System.out.print("Enter Task ID: ");
                    id = sc.nextInt();
                    sc.nextLine();
                    System.out.print("Enter Task Name: ");
                    name = sc.nextLine();
                    System.out.print("Enter Priority: ");
                    priority = sc.nextInt();
                    sc.nextLine();
                    System.out.print("Enter Due Date: ");
                    dueDate = sc.nextLine();
                    System.out.print("Enter Position: ");
                    pos = sc.nextInt();
                    addTaskAtPosition(id, name, priority,
dueDate, pos);
                    break;

                case 4:
                    System.out.print("Enter Task ID to remove:
");
                    id = sc.nextInt();
                    removeTaskById(id);
                    break;
```

```
                case 5:
                    viewCurrentTask();
                    break;

                case 6:
                    displayAllTasks();
                    break;

                case 7:
                    System.out.print("Enter priority to search:
");
                    priority = sc.nextInt();
                    searchByPriority(priority);
                    break;

                case 0:
                    System.out.println("Exiting Task
Scheduler.");
                    break;

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

        } while (choice != 0);

        sc.close();
    }
}
```

4. Singly Linked List: Inventory Management System

CodInClub

Powered By BridgeLabz

Problem Statement: Design an inventory management system using a singly linked list where each node stores information about an item such as `Item Name`, `Item ID`, `Quantity`, and `Price`. Implement the following functionalities:

1. Add an item at the beginning, end, or at a specific position.
2. Remove an item based on `Item ID`.
3. Update the quantity of an item by `Item ID`.
4. Search for an item based on `Item ID` or `Item Name`.
5. Calculate and display the total value of inventory (Sum of `Price * Quantity` for each item).
6. Sort the inventory based on `Item Name` or `Price` in ascending or descending order.

Hint:

- Use a singly linked list where each node represents an item in the inventory.
- Implement sorting using an appropriate algorithm (e.g., merge sort) on the linked list.
- For total value calculation, traverse through the list and sum up `Quantity * Price` for each item.

```
import java.util.Scanner;

class Item {
    String itemName;
    int itemId;
    int quantity;
    double price;
    Item next;

    public Item(String itemName, int itemId, int quantity, double price) {
        this.itemName = itemName;
        this.itemId = itemId;
        this.quantity = quantity;
        this.price = price;
    }
}

public class InventoryManager {
    static Item head = null;

    public static void addItemAtBeginning(String name, int id, int qty,
```

```
double price) {
        Item newItem = new Item(name, id, qty, price);
        newItem.next = head;
        head = newItem;
    }

    public static void addItemAtEnd(String name, int id, int qty, double
price) {
        Item newItem = new Item(name, id, qty, price);
        if (head == null) {
            head = newItem;
            return;
        }

        Item temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }

        temp.next = newItem;
    }

    public static void addItemAtPosition(String name, int id, int qty,
double price, int pos) {
        if (pos == 1) {
            addItemAtBeginning(name, id, qty, price);
            return;
        }

        Item newItem = new Item(name, id, qty, price);
        Item temp = head;
        int count = 1;

        while (temp != null && count < pos - 1) {
            temp = temp.next;
            count++;
        }

        if (temp == null) {
            System.out.println("Position out of range.");
            return;
```

```
        }

        newItem.next = temp.next;
        temp.next = newItem;
    }

    public static void removeItemById(int id) {
        if (head == null) return;

        if (head.itemId == id) {
            head = head.next;
            return;
        }

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

        if (temp.next != null) {
            temp.next = temp.next.next;
            System.out.println("Item removed.");
        } else {
            System.out.println("Item not found.");
        }
    }

    public static void updateQuantityById(int id, int newQty) {
        Item temp = head;
        while (temp != null) {
            if (temp.itemId == id) {
                temp.quantity = newQty;
                System.out.println("Quantity updated.");
                return;
            }
            temp = temp.next;
        }
        System.out.println("Item not found.");
    }

    public static void searchByIdOrName(int id, String name) {
```

```
        Item temp = head;
        boolean found = false;

        while (temp != null) {
            if (temp.itemId == id || temp.itemName.equalsIgnoreCase(name))
{
                System.out.println("ID: " + temp.itemId + ", Name: " +
temp.itemName + ", Qty: " + temp.quantity + ", Price: " + temp.price);
                found = true;
            }
            temp = temp.next;
        }

        if (!found) System.out.println("No matching item found.");
    }

    public static void calculateTotalInventoryValue() {
        Item temp = head;
        double total = 0;

        while (temp != null) {
            total += temp.quantity * temp.price;
            temp = temp.next;
        }

        System.out.println("Total Inventory Value: ₹" + total);
    }

    public static void sortByNameOrPrice(boolean byName, boolean ascending)
{
        if (head == null || head.next == null) return;

        Item sorted = null;

        while (head != null) {
            Item curr = head;
            head = head.next;

            if (sorted == null || compare(curr, sorted, byName, ascending))
{
                curr.next = sorted;
```

```
                sorted = curr;
            } else {
                Item temp = sorted;
                while (temp.next != null && !compare(curr, temp.next,
byName, ascending)) {
                    temp = temp.next;
                }
                curr.next = temp.next;
                temp.next = curr;
            }
        }

        head = sorted;
        System.out.println("Inventory sorted.");
    }

    private static boolean compare(Item a, Item b, boolean byName, boolean
ascending) {
        if (byName) {
            return ascending ? a.itemName.compareToIgnoreCase(b.itemName) <
0 : a.itemName.compareToIgnoreCase(b.itemName) > 0;
        } else {
            return ascending ? a.price < b.price : a.price > b.price;
        }
    }

    public static void displayInventory() {
        if (head == null) {
            System.out.println("Inventory is empty.");
            return;
        }

        Item temp = head;
        while (temp != null) {
            System.out.println("ID: " + temp.itemId + ", Name: " +
temp.itemName + ", Qty: " + temp.quantity + ", Price: " + temp.price);
            temp = temp.next;
        }
    }

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

```
        Scanner sc = new Scanner(System.in);
        int choice, id, qty, pos;
        double price;
        String name;

        do {
            System.out.println("\n--- Inventory Management ---");
            System.out.println("1. Add Item at Beginning");
            System.out.println("2. Add Item at End");
            System.out.println("3. Add Item at Position");
            System.out.println("4. Remove Item by ID");
            System.out.println("5. Update Quantity by ID");
            System.out.println("6. Search Item by ID or Name");
            System.out.println("7. Calculate Total Inventory Value");
            System.out.println("8. Sort by Name Asc/Desc");
            System.out.println("9. Sort by Price Asc/Desc");
            System.out.println("10. Display Inventory");
            System.out.println("0. Exit");
            System.out.print("Enter choice: ");
            choice = sc.nextInt();
            sc.nextLine();

            switch (choice) {
                case 1:
                case 2:
                case 3:
                    System.out.print("Enter Item Name: ");
                    name = sc.nextLine();
                    System.out.print("Enter Item ID: ");
                    id = sc.nextInt();
                    System.out.print("Enter Quantity: ");
                    qty = sc.nextInt();
                    System.out.print("Enter Price: ");
                    price = sc.nextDouble();

                    if (choice == 1) addItemAtBeginning(name, id, qty,
price);
                    else if (choice == 2) addItemAtEnd(name, id, qty,
price);
                    else {
                        System.out.print("Enter Position: ");
```

```
                    pos = sc.nextInt();
                    addItemAtPosition(name, id, qty, price, pos);
                }
                break;

            case 4:
                System.out.print("Enter ID to remove: ");
                id = sc.nextInt();
                removeItemById(id);
                break;

            case 5:
                System.out.print("Enter ID: ");
                id = sc.nextInt();
                System.out.print("Enter New Quantity: ");
                qty = sc.nextInt();
                updateQuantityById(id, qty);
                break;

            case 6:
                System.out.print("Enter ID (enter -1 if unknown): ");
                id = sc.nextInt();
                sc.nextLine();
                System.out.print("Enter Name (enter blank if unknown):
");
                name = sc.nextLine();
                searchByIdOrName(id, name);
                break;

            case 7:
                calculateTotalInventoryValue();
                break;

            case 8:
                System.out.print("Sort Ascending? (true/false): ");
                boolean asc1 = sc.nextBoolean();
                sortByNameOrPrice(true, asc1);
                break;

            case 9:
                System.out.print("Sort Ascending? (true/false): ");
```

```
                    boolean asc2 = sc.nextBoolean();
                    sortByNameOrPrice(false, asc2);
                    break;

                case 10:
                    displayInventory();
                    break;

                case 0:
                    System.out.println("Exiting...");
                    break;

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

        } while (choice != 0);

        sc.close();
    }
}
```

---

5. Doubly Linked List: Library Management System

Problem Statement: Design a library management system using a doubly linked list. Each node represents a book and contains the following attributes: `Book Title`, `Author`, `Genre`, `Book ID`, and `Availability Status`. Implement the following functionalities:

1. Add a new book at the beginning, end, or at a specific position.
2. Remove a book by `Book ID`.
3. Search for a book by `Book Title` or `Author`.
4. Update a book's `Availability Status`.
5. Display all books in forward and reverse order.
6. Count the total number of books in the library.

Hint:

- Use a doubly linked list with two pointers (`next` and `prev`) in each node to facilitate traversal in both directions.
- Ensure that when removing a book, both the `next` and `prev` pointers are correctly updated.
- Displaying in reverse order will require traversal from the last node using `prev` pointers.

```
import java.util.*;

class Book {
    String title;
    String author;
    String genre;
    int bookId;
    boolean isAvailable;
    Book next, prev;

    public Book(String title, String author, String genre, int bookId,
boolean isAvailable) {
        this.title = title;
        this.author = author;
        this.genre = genre;
        this.bookId = bookId;
        this.isAvailable = isAvailable;
        this.next = null;
        this.prev = null;
    }
}

class LibraryManagementSystem {
    Book head = null, tail = null;

    public void addBook(String title, String author, String genre, int
bookId, boolean isAvailable, int position) {
        Book newBook = new Book(title, author, genre, bookId, isAvailable);
        if (head == null) {
            head = tail = newBook;
        } else if (position == 1) {
            newBook.next = head;
            head.prev = newBook;
            head = newBook;
        } else {
```

```
            Book temp = head;
            int count = 1;
            while (temp.next != null && count < position - 1) {
                temp = temp.next;
                count++;
            }
            newBook.next = temp.next;
            newBook.prev = temp;
            if (temp.next != null) {
                temp.next.prev = newBook;
            } else {
                tail = newBook;
            }
            temp.next = newBook;
        }
        System.out.println("Book added successfully.");
    }

    public void removeBook(int bookId) {
        if (head == null) {
            System.out.println("Library is empty.");
            return;
        }
        Book temp = head;
        while (temp != null) {
            if (temp.bookId == bookId) {
                if (temp == head) {
                    head = temp.next;
                    if (head != null) head.prev = null;
                } else if (temp == tail) {
                    tail = temp.prev;
                    if (tail != null) tail.next = null;
                } else {
                    temp.prev.next = temp.next;
                    temp.next.prev = temp.prev;
                }
                System.out.println("Book removed successfully.");
                return;
            }
            temp = temp.next;
        }
```

```
        System.out.println("Book ID not found.");
    }

    public void searchBook(String keyword) {
        Book temp = head;
        boolean found = false;
        while (temp != null) {
            if (temp.title.equalsIgnoreCase(keyword) ||
temp.author.equalsIgnoreCase(keyword)) {
                System.out.println("Found Book - ID: " + temp.bookId + ",
Title: " + temp.title + ", Author: " + temp.author + ", Genre: " +
temp.genre + ", Available: " + temp.isAvailable);
                found = true;
            }
            temp = temp.next;
        }
        if (!found) {
            System.out.println("No book found with the given keyword.");
        }
    }

    public void updateAvailability(int bookId, boolean status) {
        Book temp = head;
        while (temp != null) {
            if (temp.bookId == bookId) {
                temp.isAvailable = status;
                System.out.println("Book availability updated.");
                return;
            }
            temp = temp.next;
        }
        System.out.println("Book ID not found.");
    }

    public void displayForward() {
        Book temp = head;
        while (temp != null) {
            System.out.println("ID: " + temp.bookId + ", Title: " +
temp.title + ", Author: " + temp.author + ", Genre: " + temp.genre + ",
Available: " + temp.isAvailable);
            temp = temp.next;
```

```
        }
    }

    public void displayBackward() {
        Book temp = tail;
        while (temp != null) {
            System.out.println("ID: " + temp.bookId + ", Title: " +
temp.title + ", Author: " + temp.author + ", Genre: " + temp.genre + ",
Available: " + temp.isAvailable);
            temp = temp.prev;
        }
    }

    public void countBooks() {
        int count = 0;
        Book temp = head;
        while (temp != null) {
            count++;
            temp = temp.next;
        }
        System.out.println("Total number of books: " + count);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        LibraryManagementSystem lib = new LibraryManagementSystem();
        while (true) {
            System.out.println("\n1. Add Book\n2. Remove Book\n3. Search
Book\n4. Update Availability\n5. Display Forward\n6. Display Backward\n7.
Count Books\n8. Exit");
            System.out.print("Choose option: ");
            int choice = sc.nextInt();
            switch (choice) {
                case 1:
                    sc.nextLine();
                    System.out.print("Title: ");
                    String title = sc.nextLine();
                    System.out.print("Author: ");
                    String author = sc.nextLine();
                    System.out.print("Genre: ");
                    String genre = sc.nextLine();
```

```
                    System.out.print("Book ID: ");
                    int bookId = sc.nextInt();
                    System.out.print("Available (true/false): ");
                    boolean available = sc.nextBoolean();
                    System.out.print("Position to insert (1 for beginning):
");
                    int pos = sc.nextInt();
                    lib.addBook(title, author, genre, bookId, available,
pos);
                    break;
                case 2:
                    System.out.print("Enter Book ID to remove: ");
                    int removeId = sc.nextInt();
                    lib.removeBook(removeId);
                    break;
                case 3:
                    sc.nextLine();
                    System.out.print("Enter title or author to search: ");
                    String keyword = sc.nextLine();
                    lib.searchBook(keyword);
                    break;
                case 4:
                    System.out.print("Enter Book ID to update: ");
                    int updateId = sc.nextInt();
                    System.out.print("Enter new availability (true/false):
");
                    boolean status = sc.nextBoolean();
                    lib.updateAvailability(updateId, status);
                    break;
                case 5:
                    lib.displayForward();
                    break;
                case 6:
                    lib.displayBackward();
                    break;
                case 7:
                    lib.countBooks();
                    break;
                case 8:
                    System.out.println("Exiting...");
                    return;
```

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

---

6. Circular Linked List: Round Robin Scheduling Algorithm

Problem Statement: Implement a round-robin CPU scheduling algorithm using a circular linked list. Each node will represent a process and contain `Process ID`, `Burst Time`, and `Priority`. Implement the following functionalities:

1. Add a new process at the end of the circular list.
2. Remove a process by `Process ID` after its execution.
3. Simulate the scheduling of processes in a round-robin manner with a fixed time quantum.
4. Display the list of processes in the circular queue after each round.
5. Calculate and display the average waiting time and turn-around time for all processes.

Hint:

- Use a circular linked list to represent a queue of processes.
- Each process executes for a fixed time quantum, and then control moves to the next process in the circular list.
- Maintain the current node as the process being executed, and after each round, update the list to simulate execution.

```
import java.util.*;

class Process {
    int pid, burstTime, priority, remainingTime;
    Process next;

    public Process(int pid, int burstTime, int priority) {
        this.pid = pid;
        this.burstTime = burstTime;
```

```
        this.priority = priority;
        this.remainingTime = burstTime;
        this.next = null;
    }
}

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

    static void addProcess(int pid, int burstTime, int priority) {
        Process newNode = new Process(pid, burstTime, priority);
        if (head == null) {
            head = newNode;
            newNode.next = head;
        } else {
            Process temp = head;
            while (temp.next != head) temp = temp.next;
            temp.next = newNode;
            newNode.next = head;
        }
    }

    static void removeProcess(int pid) {
        if (head == null) return;
        Process curr = head, prev = null;
        do {
            if (curr.pid == pid) {
                if (prev == null) {
                    if (curr.next == head) head = null;
                    else {
                        Process tail = head;
                        while (tail.next != head) tail = tail.next;
                        head = head.next;
                        tail.next = head;
                    }
                } else {
                    prev.next = curr.next;
                }
                return;
            }
```

```
            prev = curr;
            curr = curr.next;
        } while (curr != head);
    }

    static void simulate(int quantum) {
        if (head == null) {
            System.out.println("No processes to schedule.");
            return;
        }
        Process curr = head;
        int time = 0, totalWT = 0, totalTAT = 0, n = 0;
        while (true) {
            boolean done = true;
            Process temp = head;
            do {
                if (temp.remainingTime > 0) {
                    done = false;
                    if (temp.remainingTime > quantum) {
                        time += quantum;
                        temp.remainingTime -= quantum;
                    } else {
                        time += temp.remainingTime;
                        totalWT += time - temp.burstTime;
                        totalTAT += time;
                        temp.remainingTime = 0;
                        removeProcess(temp.pid);
                    }
                }
                temp = temp.next;
            } while (temp != head);
            if (done) break;
        }
        System.out.println("Average Waiting Time: " + (totalWT / 3.0));
        System.out.println("Average Turnaround Time: " + (totalTAT / 3.0));
    }

    static void display() {
        if (head == null) {
            System.out.println("Queue is empty.");
            return;
```

```
        }
        Process temp = head;
        do {
            System.out.println("PID: " + temp.pid + ", Burst: " +
temp.burstTime + ", Priority: " + temp.priority);
            temp = temp.next;
        } while (temp != head);
    }

    public static void main(String[] args) {
        System.out.println("Enter number of processes:");
        int n = sc.nextInt();
        for (int i = 1; i <= n; i++) {
            System.out.println("Enter Burst Time and Priority for Process "
+ i + ":");
            addProcess(i, sc.nextInt(), sc.nextInt());
        }
        System.out.println("Initial Queue:");
        display();
        System.out.println("Enter time quantum:");
        int quantum = sc.nextInt();
        simulate(quantum);
    }
}
```

---

7. Singly Linked List: Social Media Friend Connections

Problem Statement: Create a system to manage social media friend connections using a singly linked list. Each node represents a user with `User ID`, `Name`, `Age`, and `List of Friend IDs`. Implement the following operations:

1. Add a friend connection between two users.
2. Remove a friend connection.
3. Find mutual friends between two users.
4. Display all friends of a specific user.
5. Search for a user by `Name` or `User ID`.
6. Count the number of friends for each user.

Hint:

- Use a singly linked list where each node contains a list of friends (which can be another linked list or array of `Friend IDs`).
- For mutual friends, traverse both lists and compare the `Friend IDs`.
- The `List of Friend IDs` for each user can be implemented as a nested linked list or array.

```
import java.util.*;

class User {
    int userId;
    String name;
    int age;
    List<Integer> friendIds;
    User next;

    User(int userId, String name, int age) {
        this.userId = userId;
        this.name = name;
        this.age = age;
        this.friendIds = new ArrayList<>();
        this.next = null;
    }
}

public class SocialMediaManager {
    User head;

    void addUser(int userId, String name, int age) {
        User newUser = new User(userId, name, age);
        if (head == null) head = newUser;
        else {
            User temp = head;
            while (temp.next != null) temp = temp.next;
            temp.next = newUser;
        }
    }

    User findUser(int userId) {
        User temp = head;
```

```
        while (temp != null) {
            if (temp.userId == userId) return temp;
            temp = temp.next;
        }
        return null;
    }

    void addFriend(int uid1, int uid2) {
        User u1 = findUser(uid1), u2 = findUser(uid2);
        if (u1 != null && u2 != null && !u1.friendIds.contains(uid2)) {
            u1.friendIds.add(uid2);
            u2.friendIds.add(uid1);
        }
    }

    void removeFriend(int uid1, int uid2) {
        User u1 = findUser(uid1), u2 = findUser(uid2);
        if (u1 != null && u2 != null) {
            u1.friendIds.remove((Integer) uid2);
            u2.friendIds.remove((Integer) uid1);
        }
    }

    void showFriends(int userId) {
        User user = findUser(userId);
        if (user != null) {
            System.out.println(user.name + "'s Friends: " +
user.friendIds);
        }
    }

    void mutualFriends(int uid1, int uid2) {
        User u1 = findUser(uid1), u2 = findUser(uid2);
        if (u1 != null && u2 != null) {
            List<Integer> mutual = new ArrayList<>(u1.friendIds);
            mutual.retainAll(u2.friendIds);
            System.out.println("Mutual Friends: " + mutual);
        }
    }

    void search(String keyword) {
```

```
        User temp = head;
        while (temp != null) {
            if (temp.name.equalsIgnoreCase(keyword) ||
Integer.toString(temp.userId).equals(keyword)) {
                System.out.println("Found: " + temp.name + " (ID: " +
temp.userId + ")");
                return;
            }
            temp = temp.next;
        }
        System.out.println("User not found.");
    }

    void countFriends() {
        User temp = head;
        while (temp != null) {
            System.out.println(temp.name + " has " + temp.friendIds.size()
+ " friends.");
            temp = temp.next;
        }
    }

    public static void main(String[] args) {
        SocialMediaManager sm = new SocialMediaManager();
        sm.addUser(1, "Alice", 20);
        sm.addUser(2, "Bob", 22);
        sm.addUser(3, "Charlie", 23);

        sm.addFriend(1, 2);
        sm.addFriend(1, 3);
        sm.addFriend(2, 3);

        sm.showFriends(1);
        sm.mutualFriends(1, 2);
        sm.search("Charlie");
        sm.countFriends();
        sm.removeFriend(1, 2);
        sm.showFriends(1);
    }
}
```

---

8. Doubly Linked List: Undo/Redo Functionality for Text Editor

Problem Statement: Design an undo/redo functionality for a text editor using a doubly linked list. Each node represents a state of the text content (e.g., after typing a word or performing a command). Implement the following:

1. Add a new text state at the end of the list every time the user types or performs an action.
2. Implement the undo functionality (revert to the previous state).
3. Implement the redo functionality (revert back to the next state after undo).
4. Display the current state of the text.
5. Limit the undo/redo history to a fixed size (e.g., last 10 states).

Hint:

- Use a doubly linked list where each node represents a state of the text.
- The `next` pointer will represent the forward history (redo), and the `prev` pointer will represent the backward history (undo).
- Keep track of the current state and adjust the `next` and `prev` pointers for undo/redo operations.

```
import java.util.*;

class TextState {
    String content;
    TextState prev, next;

    TextState(String content) {
        this.content = content;
    }
}

public class TextEditor {
    private TextState current;
    private int size = 0;
    private final int MAX_SIZE = 10;
```

```
    public void addState(String content) {
        TextState newState = new TextState(content);
        if (current != null) {
            current.next = newState;
            newState.prev = current;
        }
        current = newState;
        size++;
        trimHistory();
    }

    public void undo() {
        if (current != null && current.prev != null) {
            current = current.prev;
            System.out.println("Undo: " + current.content);
        } else {
            System.out.println("No more undo available.");
        }
    }

    public void redo() {
        if (current != null && current.next != null) {
            current = current.next;
            System.out.println("Redo: " + current.content);
        } else {
            System.out.println("No more redo available.");
        }
    }

    public void showCurrent() {
        System.out.println("Current: " + (current != null ? current.content
: "Empty"));
    }

    private void trimHistory() {
        TextState temp = current;
        int count = 1;
        while (temp.prev != null) {
            temp = temp.prev;
            count++;
```

```
            if (count > MAX_SIZE) {
                temp.prev.next = null;
                temp.prev = null;
                break;
            }
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        TextEditor editor = new TextEditor();
        while (true) {
            System.out.println("1. Add Text\n2. Undo\n3. Redo\n4. Show
Current\n5. Exit");
            int choice = sc.nextInt();
            sc.nextLine();
            switch (choice) {
                case 1:
                    System.out.print("Enter text: ");
                    editor.addState(sc.nextLine());
                    break;
                case 2:
                    editor.undo();
                    break;
                case 3:
                    editor.redo();
                    break;
                case 4:
                    editor.showCurrent();
                    break;
                case 5:
                    return;
                default:
                    System.out.println("Invalid option.");
            }
        }
    }
}
```

9. Circular Linked List: Online Ticket Reservation System

Problem Statement: Design an online ticket reservation system using a circular linked list, where each node represents a booked ticket. Each node will store the following information: `Ticket ID`, `Customer Name`, `Movie Name`, `Seat Number`, and `Booking Time`. Implement the following functionalities:

1. Add a new ticket reservation at the end of the circular list.
2. Remove a ticket by `Ticket ID`.
3. Display the current tickets in the list.
4. Search for a ticket by `Customer Name` or `Movie Name`.
5. Calculate the total number of booked tickets.

Hint:

- Use a circular linked list to represent the ticket reservations, with the last node's `next` pointer pointing to the first node.
- When removing a ticket, update the circular pointers accordingly.
- For displaying all tickets, traverse the list starting from the first node, looping back after reaching the last node.

```
import java.util.Scanner;

class Ticket {
    int ticketId;
    String customerName, movieName, seatNumber, bookingTime;
    Ticket next;

    Ticket(int id, String name, String movie, String seat, String time) {
        ticketId = id;
        customerName = name;
        movieName = movie;
        seatNumber = seat;
        bookingTime = time;
    }
}

public class TicketReservationSystem {
    static Ticket head = null;
```

```
    static void addTicket(int id, String name, String movie, String seat,
String time) {
        Ticket newTicket = new Ticket(id, name, movie, seat, time);
        if (head == null) {
            head = newTicket;
            head.next = head;
        } else {
            Ticket temp = head;
            while (temp.next != head)
                temp = temp.next;
            temp.next = newTicket;
            newTicket.next = head;
        }
        System.out.println("Ticket booked successfully!");
    }

    static void removeTicket(int id) {
        if (head == null) {
            System.out.println("No tickets found.");
            return;
        }
        Ticket curr = head, prev = null;
        do {
            if (curr.ticketId == id) {
                if (curr == head && curr.next == head)
                    head = null;
                else {
                    if (curr == head) {
                        Ticket last = head;
                        while (last.next != head)
                            last = last.next;
                        head = head.next;
                        last.next = head;
                    } else
                        prev.next = curr.next;
                }
                System.out.println("Ticket removed!");
                return;
            }
            prev = curr;
            curr = curr.next;
```

```
        } while (curr != head);
        System.out.println("Ticket ID not found.");
    }

    static void displayTickets() {
        if (head == null) {
            System.out.println("No tickets booked.");
            return;
        }
        Ticket temp = head;
        System.out.println("Booked Tickets:");
        do {
            System.out.println("ID: " + temp.ticketId + ", Name: " +
temp.customerName +
                    ", Movie: " + temp.movieName + ", Seat: " +
temp.seatNumber +
                    ", Time: " + temp.bookingTime);
            temp = temp.next;
        } while (temp != head);
    }

    static void searchTicket(String keyword) {
        if (head == null) {
            System.out.println("No tickets found.");
            return;
        }
        boolean found = false;
        Ticket temp = head;
        do {
            if (temp.customerName.equalsIgnoreCase(keyword) ||
temp.movieName.equalsIgnoreCase(keyword)) {
                System.out.println("Ticket found -> ID: " + temp.ticketId +
", Name: " + temp.customerName +
                        ", Movie: " + temp.movieName + ", Seat: " +
temp.seatNumber +
                        ", Time: " + temp.bookingTime);
                found = true;
            }
            temp = temp.next;
        } while (temp != head);
        if (!found) System.out.println("No matching ticket found.");
```

```
    }

    static void countTickets() {
        if (head == null) {
            System.out.println("Total Tickets: 0");
            return;
        }
        int count = 0;
        Ticket temp = head;
        do {
            count++;
            temp = temp.next;
        } while (temp != head);
        System.out.println("Total Tickets: " + count);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (true) {
            System.out.println("\n1. Add Ticket\n2. Remove Ticket\n3.
Display Tickets\n4. Search Ticket\n5. Count Tickets\n6. Exit");
            switch (sc.nextInt()) {
                case 1:
                    System.out.print("Enter ID, Name, Movie, Seat, Time:
");
                    addTicket(sc.nextInt(), sc.next(), sc.next(),
sc.next(), sc.next());
                    break;
                case 2:
                    System.out.print("Enter Ticket ID to remove: ");
                    removeTicket(sc.nextInt());
                    break;
                case 3:
                    displayTickets();
                    break;
                case 4:
                    System.out.print("Enter Name or Movie to search: ");
                    searchTicket(sc.next());
                    break;
                case 5:
                    countTickets();
```

```
                    break;
                case 6:
                    System.out.println("Exiting.");
                    return;
            }
        }
    }
}
```