CS401 Lab 8: Linked List Data Structure Implementations

Overview

- This lab is to be completed individually.
- Focus: Understanding and implementing various types of Linked List data structures.
- Objective: Create three different linked list implementations for managing Employee objects.

Part 1: Sorted Linked List

Objectives

1. Implement a basic *LinkedList* class
2. Create a *SortedLinkedList* class that inherits from LinkedList
3. Maintain elements in sorted order based on Employee IDs

Requirements

1. Base LinkedList Implementation:

```
public class LLNode<T> {
  protected T info;
  protected LLNode<T> link;

  public LLNode(T info) {
    this.info = info;
    link = null;
  }

  public void setInfo(T info) { this.info = info; }
  public T getInfo() { return info; }
  public void setLink(LLNode<T> link) { this.link = link; }
  public LLNode<T> getLink() { return link; }
}
```

2. Create a *SortedLinkedList* class that uses the *LLNode* structure
3. Maintain sorted order based on Employee IDs
4. Required Methods:
   - *add(Employee emp)*: Adds employee in sorted position
   - *remove(int id)*: Removes employee with given ID
   - *contains(int id)*: Checks if employee with given ID exists

4. Operations:
   - Read employee data from emp.txt
   - Add employees while maintaining sorted order
   - Print the sorted list

Expected Output

```
Sorted Linked List Contents:
1. [Employee ID: 1001, Name: John Doe]
2. [Employee ID: 1002, Name: Jane Smith]
...
```

Part 2: Circular Linked List

Objectives

Implement a circular linked list where the last node points back to the first node.

Requirements

1. Extend LLNode to create CircularLLNode with necessary modifications
2. Implement circular linkage (last node points to first)
3. Operations:
    - Insert first 8 elements from emp.txt
    - Display all elements
    - Delete one element by ID
    - Verify circular nature of the list

Expected Output

```
Initial Circular List:
[ID: 1001] -> [ID: 1002] -> ... -> [ID: 1008] -> (back to start)

After Deleting ID 1003:
[ID: 1001] -> [ID: 1002] -> [ID: 1004] -> ... -> [ID: 1008] -> (back to start)
```

Part 3: Doubly Linked List

Objectives

Implement a doubly linked list where each node contains references to both next and previous nodes.

Requirements

1. Extend *LLNode* to create *DoublyLLNode*, adding the *prev* reference
2. Required Methods:
    - *add(T element)*: Adds element to the list
    - *remove(T element)*: Removes element from the list
    - *contains(T element)*: Checks if element exists
    - *displayForward()*: Prints list from front to back
    - *displayBackward()*: Prints list from back to front

Expected Output

```
Forward Traversal:
1 <-> 2 <-> 3 <-> 4 <-> 5
```

```
Backward Traversal:
5 <-> 4 <-> 3 <-> 2 <-> 1
```

General Requirements

1. Code Documentation:
    - Include inline comments for all methods
    - Document the purpose of each class
    - Explain complex algorithms

2. Error Handling:
    - Handle empty list cases
    - Manage invalid operations
    - Validate input data

Submission Requirements

1. Source Code Files:
    - LLNode.java (base class)
    - EmployeeNode.java (if extending LLNode)
    - Employee.java (if using separate class)
    - SortedLinkedList.java
    - CircularLLNode.java
    - DoublyLLNode.java
    - Main.java (test program)
    - Compiled Bytecode
    - Executable JAR file

3. Documentation:
    - PDF file containing program outputs for all three parts
    - README file with:
        - Program description
        - Compilation instructions
        - Execution instructions
        - JAR file execution command
    - emp.txt (input file)
    - Ensure proper node linkage in all implementations, and test edge cases (empty list, single element, etc.)