TRINITY COLLEGE

DEPARTMENT OF COMPUTER SCIENCE

CPSC 215: Data Structures and Algorithms

Instructor: Dr. Chandranil Chakraborttii

Spring 2025

Laboratory 4

Objective:

- To familiarize yourself with Double Linked Lists.
- To implement a set of generic objects in a Java class that supports basic operations using Double Linked Lists.

Academic Honesty Policy

In working on programming assignments, you may discuss broad issues of interpretation and understanding and general approaches to a solution. However, conversion to a specific solution or to program code must be your own work. Programming assignments are expected to be the work of the individual student, designed, and coded by him or her alone. Violations are easy to identify and will be dealt with promptly according to the Academic Integrity and Intellectual Dishonesty outlined in the Student Handbook.

- Copying another person's programs or encouraging or assisting another person to commit plagiarism is cheating. In particular, the following activities are strictly prohibited:
- Giving and receiving help in the actual development of code or writing of an assignment.
- Looking at another person's code or showing your code to another person.
- Sharing a copy of all or part of your code regardless of whether that copy is on paper or in a computer file.
- Turning in the work of any other person(s) (former students, friends, textbook authors, people on the Internet, etc.) and representing it as your own work.
- Fabricating compilation or execution results.
- Use of AI ChatBots for any help regarding the assignment is **strictly prohibited.**

The penalty for cheating is a failing grade (F) for the course, and the student is asked to appear at an Academic Dishonesty Hearing. In addition, the College also places a record of the incident in the student's permanent record. It is your responsibility to protect your work from unauthorized access. Do not discard copies of your programs in public places. Do not leave computers unattended and copies of output lying around.

Step 1: Getting Started

- Open Eclipse and create a Java Project (File -> New -> Java Project)
- Give a project name (For example: Lab_4_CPSC_215)
- Go to Project -> Right Click -> New Package
- Give a package name (For example: Lab_4_CPSC_215)
- Now to create the classes, go to package -> Right Click -> New Java class.
- Paste contents of the given files. Keep the first line in place (package package_name;)
- Understand the code structure before starting to write code.
- Complete code to complete the classes labelled "**Need to complete**" below.

Input Files (Starter Code)

- List.java **(No need to change)**
- DLink.java - **(No need to change)**
- DLList.java **(Need to change)**
- DLListMain.java – Contains the main function **(No Need to change)**

Step 2: Review the DLink and List class

- Understand the node structure used in a doubly linked list. (DLink)
- Understand the interface methods and to be implemented in the DLList class.

Step 4: Review the Implemented methods in DLList class

- Review the completed methods in DLList class.

Step 5: Implement new Methods (DLList.java)

Methods to Implement:

- **removeLast():** Removes the last element from the linked list and updates the tail reference.
- **removeFirst():** Removes the first element from the linked list and updates the head reference.
- **findMiddle():** Finds and returns the middle element of the linked list.
- **insertAtPosition(int pos, T data):** Inserts a new node with the specified data at the given position.
- **displayEvenElements():** Displays elements for even-indexed nodes.
- **removeDuplicates():** Removes duplicate elements from the linked list while preserving order.
- **bubbleSort():** Sorts the linked list using the Bubble Sort algorithm.

Additional Notes:

- The implementation must support **generic types** (T extends Comparable<T>).
- **Sorting must use Bubble Sort** (manual implementation, no Arrays.sort()).

Step-by-Step Algorithm (Bubble Sort):

- **Check for an empty or single-node list**: If the list has **0 or 1 elements**, it's already sorted. (Edge case check)
- **Outer Loop (cnt - 1 passes)**: [where `cnt` is the number of elementz]
    - **Each pass moves largest element to correct position** at the end of list.
    - **Reduce the number of comparisons** on each subsequent pass.
- **Inner Loop (Adjacent Comparison)**:
    - Start from the **head** and **traverse until the last unsorted node**.
    - Compare **current node's value** with the **next node's value**:
        - If the current value is **greater**, **swap them**.
        - **Continue until all elements are sorted**.
- **Optimization using swapped flag**:
    - If **no swaps happen in a full pass**, the list is already sorted, and we **exit early**.

CompareTo: The compareTo method returns an integer value:
If element1 is lexicographically less than element2, compareTo returns negative integer.
If element1 is lexicographically equal to element2, it returns 0.
If element1 is lexicographically greater than element2, it returns a positive integer.

Step 6: Testing for correctness

- Use class DLListMain to test the correctness of your Main class.
- The class DLListMain is already completed, so you should not have to make any more modifications.
-

Step 7: Documentation and Reflection

- Be sure the code is runnable, testable, formatted, and easily readable.
- Include JavaDoc to explain the purpose and functionality of each class, method, and exception as per standards (DLList).
- Reflect on the challenges faced and lessons learned during the implementation.

Step 8: Optional – Complete if you finish before 4 pm

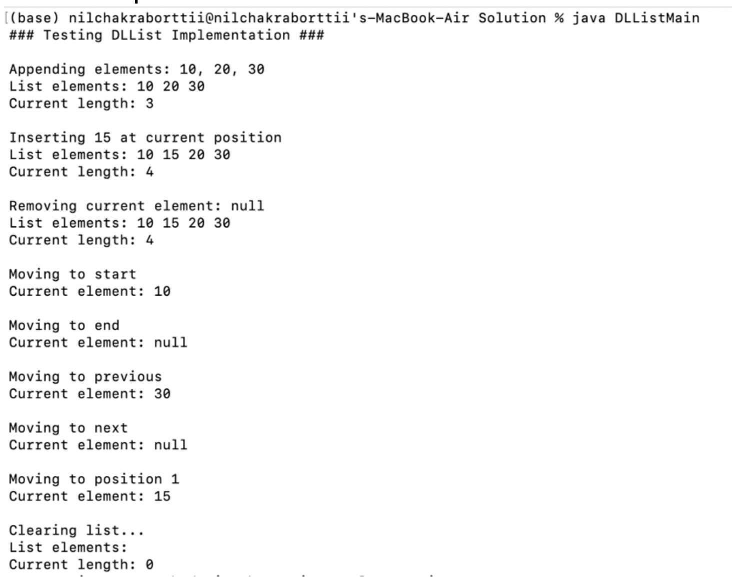
- **reverse():** Reverses the linked list in place.

Submission Guidelines:

- Make sure you completed all methods in all the required classes.
- Make sure you have added JavaDoc in your code.
- Upon the completion of your lab, upload your submission to the Moodle as a zip file.
- See your TA and get your Lab graded before leaving the lab.

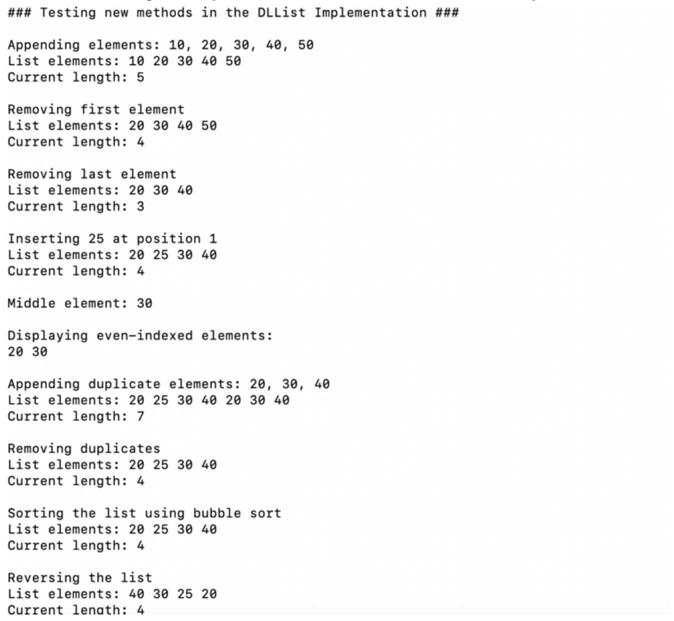
Sample output:

Initial Output:



```
(base) nilchakraborttii@nilchakraborttii's-MacBook-Air Solution % java DLListMain
### Testing DLList Implementation ###

Appending elements: 10, 20, 30
List elements: 10 20 30
Current length: 3

Inserting 15 at current position
List elements: 10 15 20 30
Current length: 4

Removing current element: null
List elements: 10 15 20 30
Current length: 4

Moving to start
Current element: 10

Moving to end
Current element: null

Moving to previous
Current element: 30

Moving to next
Current element: null

Moving to position 1
Current element: 15

Clearing list...
List elements:
Current length: 0
```

Final Output (in addition to above)



```
### Testing new methods in the DLList Implementation ###

Appending elements: 10, 20, 30, 40, 50
List elements: 10 20 30 40 50
Current length: 5

Removing first element
List elements: 20 30 40 50
Current length: 4

Removing last element
List elements: 20 30 40
Current length: 3

Inserting 25 at position 1
List elements: 20 25 30 40
Current length: 4

Middle element: 30

Displaying even-indexed elements:
20 30

Appending duplicate elements: 20, 30, 40
List elements: 20 25 30 40 20 30 40
Current length: 7

Removing duplicates
List elements: 20 25 30 40
Current length: 4

Sorting the list using bubble sort
List elements: 20 25 30 40
Current length: 4

Reversing the list
List elements: 40 30 25 20
Current length: 4
```