Okeoghene Excel Omorobe

CMSC 204

Assignment 3

Learning Experience

Implementing Doubly Linked List was a highly enlightening experience that further solidified my understanding of data structures, iterators, and exception handling. While working on BasicDoubleLinkedList, I needed a clear grasp of how nodes are linked in a doubly linked list to ensure smooth forward and backward navigation. Debugging insert and delete operations helped me understand edge cases like handling an empty list, deleting a single element, and keeping head/tail pointers in sync. This reinforced how crucial it is to manage pointers (prev and next) carefully to prevent breaking the list structure.

Implementing SortedDoubleLinkedList was particularly challenging since it required inserting all elements in sorted order while still maintaining proper doubly linked list functionality. One key takeaway was learning how to traverse the list using a comparator to find the correct insertion position. Another was overriding addToFront() and addToEnd() to throw exceptions, which helped me understand how to properly restrict operations in a subclass while adhering to good object-oriented design principles.

Finally, JUnit testing played a major role in the learning process. Debugging iterator traversal issues, handling NoSuchElementException, and ensuring robust exception handling improved my problem-solving skills. Tracking down failing test cases using print statements and systematic checks taught me the importance of step-by-step debugging and carefully analyzing program flow. Overall, this project provided hands-on experience with linked list data structures, iterators, exception handling, and unit testing—valuable skills for writing efficient and bug-free code.

Pseudocode

Class BasicDoubleLinkedList<T>

**Attributes:**

* head: Node<T> (reference to the first element)
* tail: Node<T> (reference to the last element)
* size: int (number of elements in the list)

**Nested Class Node<T>**

* Attributes:
  + data: T (stores the value)
  + prev: Node<T> (reference to the previous node)
  + next: Node<T> (reference to the next node)
* Constructor:
  + Node(T data, Node<T> prev, Node<T> next)

**Nested Class DoubleLinkedListIterator**

* Implements ListIterator<T>
* Attributes:
  + current: Node<T> (tracks position in the list)
* Methods:
  + hasNext(): Returns true if there is a next node
  + next(): Returns data of the next node and moves the pointer
  + hasPrevious(): Returns true if there is a previous node
  + previous(): Returns data of the previous node and moves the pointer
  + remove(): Throws UnsupportedOperationException

**Methods**

1. addToFront(T data)
   * Create a new node with data
   * If list is empty, set head and tail to new node
   * Otherwise, adjust pointers (prev and next) accordingly
   * Increment size
2. addToEnd(T data)
   * Create a new node with data
   * If list is empty, set head and tail to new node
   * Otherwise, adjust pointers (prev and next) accordingly
   * Increment size
3. getFirst()
   * Return head.data
   * If list is empty, return null
4. getLast()
   * Return tail.data
   * If list is empty, return null
5. remove(T target, Comparator<T> comparator)
   * Traverse the list using a loop
   * If comparator.compare(target, currentNode.data) == 0, remove the node
   * Adjust prev and next pointers
   * Decrement size
6. toArrayList()
   * Create an empty list result
   * Traverse from head to tail, adding each node's data to result
   * Return result

Class SortedDoubleLinkedList<T> extends BasicDoubleLinkedList<T>

**Attributes:**

* comparator: Comparator<T> (used to maintain order)

**Methods**

1. **add(T data)**
   * Create a new node with data
   * If list is empty, set head and tail to the new node
   * Otherwise, traverse the list and find the correct position based on comparator
   * Insert the node in sorted order
   * Increment size
2. **Override addToFront() and addToEnd()**
   * Throw UnsupportedOperationException