UNIVERSITY OF VIRGINIA, DEPARTMENT OF COMPUTER SCIENCE

Linked Lists, Stacks, Queues - Implementations

Nada Basit and Mark Floryan

January 27, 2022

1 SUMMARY

For this homework, you will be implementing multiple data structures. You will begin by implementing a **generic doubly-linked list**. Afterwards, you will implement a **queue** (using your linked-list). We will **NOT** be asking you to implement a **stack** because the implementation is so similar to the Vector you have already done:

1. Download the starter code and import the project into Eclipse
2. Implement the LinkedList.java class
3. Verify your implementation using the provided tester class
4. Implement the Queue class by utilizing your working Linked List
5. Verify your implementation using the provided tester class
6. **FILES TO DOWNLOAD:** LLStacksQueues.zip
7. **FILES TO SUBMIT:** ListIterator.java, LinkedList.java, Queue.java

1.1 ListNode.java

Once you've opened the project, take a look at the *ListNode* class. This class represents a single node in our Linked List, including references to the previous and next nodes in the list. Familiarize yourself with this class but **do not change the code in this class**. ListNode.java is implemented for you and does not require any updates.

```
public class ListNode<T> {

    /* Data being stored in this node */
    private T data;

    /* Reference to the next node in the list */
    protected ListNode<T> next;
    protected ListNode<T> prev;

    public ListNode(T data) {
        this.data = data;
        this.next = null;
        this.prev = null;
    }

    /* Getters */
    public T getData() { return this.data; }
}
```

1.2 ListIterator.java

Once you've imported the provided code into Eclipse, you may want to start on ListIterator.java (though you don't have to). A ListIterator is an object that points to (references) one element in your linked list, and provides methods to grab the element at that index, move the iterator forward one position, backward one position, etc. The provided tester uses this iterator class to test your code, and so it must be implemented correctly. The methods you will be asked to implement are:

```
    /**
     * These two methods tell us if the iterator has run off
     * the list on either side
     */
    public boolean isPastEnd();
    public boolean isPastBeginning();
```

```
8       /**
         * Get the data at the current iterator position
10       */
        public T value();
12
        /**
14       * These two methods move the cursor of the iterator
         * forward / backward one position
16       */
        public void moveForward();
18      public void moveBackward();
```

1.3 LINKEDLIST.JAVA

Next, implement all of the methods in LinkedList.java. This class will implement the provided List interface, which is duplicated for your convenience here. Your task is to implement each of these methods.

```
public interface List<T> {
2
    /**
4    * Returns the size of this list, i.e., the number
     * of nodes currently between the head and tail
6    * @return
     */
8   public int size();

10  /**
     * Clears out the entire list
12   */
    public void clear() ;
14
    /**
16   * Inserts new data at the end of the
     * list (i.e., just before the dummy tail node)
18   * @param data
     */
20  public void insertAtTail(T data);

22  /**
     * Inserts data at the front of the
24   * list (i.e., just after the dummy head node
```

```
     * @param data
     */
    public void insertAtHead(T data);

    /**
     * Inserts node such that index becomes the
     * position of the newly inserted data
     * @param data
     * @param index
     */
    public void insertAt(int index, T data);

    public T removeAtTail();

    public T removeAtHead();

    /**
     * Returns index of first occurrence of
     * the data in the list, or -1 if not present
     * @param data
     * @return
     */
    public int find(T data);

    /**
     * Returns the data at the given index, null if
     * anything goes wrong (index out of bounds, empty list, etc.)
     * @param index
     * @return
     */
    public T get(int index);
}
```

In addition, there are a few Linked List specific methods that you need to implement. They are all of the methods that involve a ListIterator in some way. The are enumerated below:

```
    /**
     * Inserts data after the node pointed to by iterator
     */
    public void insert(ListIterator<T> it, T data);

    /**
     * Remove based on Iterator position
     * Sets the iterator to the node AFTER the one removed
```

```
     */
    public T remove(ListIterator<T> it);

    /* Return iterators at front and end of list */
    public ListIterator<T> front();
    public ListIterator<T> back();
```

1.4 Testing Your Implementation

Once you implement these methods, you can test them by running the main method in the provider ListTester.java class. This simple tester will execute the methods in your list and compare them to those in Java's built in LinkedList to make sure the results are as expected.

Remember, that the tester is not perfect! If it fails on a particular method, it COULD be the case that a different method actually is causing the problem. For example, perhaps your insert has a small issue but that problem doesn't reveal itself until we try to remove items from the list. Keep this in mind as you test your code and work to fix bugs. Also remember that bugs could exist in your ListIterator class, and not in the method that the tester reports as incorrect.

One strategy might be to implement your linked list methods in the order that they are tested. That way, you can see the tester say "this method is correct" before moving on to the next one.

Notice that we expect your Linked List to be a generic class, meaning any type of Object can be stored within your Linked List.

1.5 Queue.java

Your last task is to implement the *Queue* class inside the Queue.java file. The methods you are responsible for are listed below. This Queue **must be a linked-list based queue** and you should be using your custom Linked List class to support this Queue (we've provided the import statement for you).

```
    public class Queue<T>{
        public Queue();

        public int size();

        public void enqueue(T data);
```

```
8           public T dequeue();
    }
```

Notice that this implementation should be *very simple* if you are correctly making use of your Linked List methods.

1.6 Testing and submitting

Run the tester class again to ensure your *Queue* is working correctly. You are now ready to submit.

You should submit **three files** for this homework: **ListIterator.java, LinkedList.java, and Queue.java**.

1.7 Gradescope

You should submit your code to *Gradescope*. If you are having trouble with your submission, you should double check the following common problems:

1. Make sure you are only submitting the three requested files, and they are name *ListIterator.java*, *LinkedList.java*, and *Queue.java* exactly.

2. Make sure you keep any *package* statements in your code before submitting. The autograder expects your files to have the package statements that are provided in the downloaded project.

3. Make sure your output is in the correct format. You should not be printing ANYTHING else or the autograder will think your output is incorrect.