CS 221
Programming Assignment

Double Linked List
(100 points)

"The way to succeed is to double your error rate."
--Thomas J. Watson

Objectives

- Create a doubly linked list implementation of the `IndexedUnorderedList` interface called `IUDoubleLinkedList`
- Use your test suite to ensure correct functionality of your `IUDoubleLinkedList` class
- Create a fully functional `ListIterator` for your `IUDoubleLinkedList` class
- Create test classes to test the functionality of your `ListIterator`

Tasks

For this programming assignment, you will:

1. Implement a generic list class called `IUDoubleLinkedList`.
    - Your `IUDoubleLinkedList` class should use a doubly linked list to implement all of the methods required by the `IndexedUnorderedList` interface.
    - Create a class called `DLLNode` to use in your implementation.

2. Test your implementation using the test classes created in the previous

homeworks.
   - Use all 21 change scenario classes that should include at least 600 test cases.
   - Add the class names for all of test classes that you created to the provided `iudoublelinkedlist.xml` file.
   - Run the tests to be sure your list class is working properly.

3. Implement a fully functional `ListIterator` for your `IUDoubleLinkedList` class.
   - Your `ListIterator` should implement the Java `ListIterator` interface.
   - The `IndexedUnorderedList` interface has been updated to include a `listIterator` method that should return a reference to your `ListIterator`. You will need to download the new version of this interface and use it in your implementation. For the `iterator` method already required by the interface, you should return your `ListIterator` object cast as a `Iterator` object.
   - Your implementation should also include the optional `remove`, `add`, and `set` methods.

4. Create `ListIterator` test classes for each of the change scenarios that you have completed.
   - The `ListIterator` test classes should be named `ListItrTest_ChangeScenario_X`, where `X` is the number of the scenario.
   - Each `ListIterator` test class should include test methods for all of the required `ListIterator` states (see below) related to a given change scenario. Your test cases only have to include tests for the new methods required by the interface (i.e. `hasPrevious`, `previous`, `add`, `nextIndex`, `previousIndex`, and `set`), not the ones that are part of the `Iterator` tests (i.e. `hasNext`, `next`, and `remove`).
   - The `TestCase` class already contains test cases for the `Iterator`, but no test cases for the `ListIterator` methods. You will have to create these methods for this class.
   - Here are example `ListIterator` test classes for Change Scenario 1, Change Scenario 9, and Change Scenario 38.
   - Add the class names for all of your iterator test classes to `iudoublelinkedlist.xml` and run the tests to ensure that the `ListIterator` works for your `IUDoubleLinkedList` class.

Iterator Testing

In order to test the `ListIterator` for your `IUDoubleLinkedList`, you have to consider not only the state of the list after a given change scenario, but also the state of the `ListIterator` itself.

For change scenarios that result in an empty list:

- There are two `ListIterator` states: 2 States X 6 methods = 12 Tests
  - `init`, an initialized `ListIterator`
  - `add`, a call to the ~~`next`~~ `add()` method, after the `ListIterator` was initialized

For change scenarios that result in a single-element list:

- There are four `ListIterator` states: 4 States X 6 methods = 24 Tests
  - `init`, an initialized iterator
  - `add`, a call to the ~~`next`~~ `add()` method, after the `ListIterator` was initialized
  - `nextAdd`, a call to `next` followed a call to `add`, after the `ListIterator` was initialized
  - `nextPrev`, a call to `next` followed a call to `previous`, after the `ListIterator` was initialized

For change scenarios that result in a two-element list:

- There are eight `ListIterator` states: 8 States X 6 methods = 42 Tests
  - `init`, an initialized iterator
  - `add`, a call to the ~~`next`~~ `add()` method, after the `ListIterator` was initialized
  - `nextAdd`, a call to `next` followed a call to `add`, after the `ListIterator` was initialized
  - `nextPrev`, a call to `next` followed a call to `previous`, after the `ListIterator` was initialized
  - `nextNextPrev`, two calls to `next` followed a call to `previous`, after the `ListIterator` was initialized
  - `nextPrevAdd`, a all to `next`, then to `previous`, and finally a call to `add`, after the `ListIterator` was initialized
  - `nextNextAdd`,two call to `next`, and then a call to `add`, after the `ListIterator` was initialized
  - `nextNextAddPrev`,two call to `next`, a call to `add`, and then a call to `previous`, after the `ListIterator` was initialized

For change scenarios that result in a three-element list:

- We will consider eleven iterator states: 11 States X 6 methods = 66 Tests
  - `init`, an initialized iterator
  - `add`, a call to the ~~`next`~~ `add()` method, after the `ListIterator` was initialized
  - `nextAdd`, a call to `next` followed a call to `add`, after the `ListIterator` was initialized
  - `nextPrev`, a call to `next` followed a call to `previous`, after the `ListIterator` was initialized
  - `nextNextPrev`, two calls to `next` followed a call to `previous`, after the `ListIterator` was initialized

- `nextPrevAdd`, a all to `next`, then to `previous`, and finally a call to `add`, after the `ListIterator` was initialized
- `nextNextAdd`,two call to `next`, and then a call to `add`, after the `ListIterator` was initialized
- `nextNextAddPrev`,two call to `next`, a call to `add`, and then a call to `previous`, after the `ListIterator` was initialized
- `nextNextNextPrev`, three calls to `next`, followed by a call to `previous`, after the `ListIterator` was initialized
- `nextNextNextPrevAdd`, three calls to `next`, followed by a call to `previous` and one to `add`, after the `ListIterator` was initialized
- ~~`nextNextAdd`,~~ two calls to `next`, and a call to `add`, after the `ListIterator` was initialized

nextNextNextAdd

and for each of these states, you will have to write test cases for all of the methods required by the `ListIterator` interface.

All of these `ListIterator` states are listed in the public enumeration `ListItrState` in the `TestCase` class, as well as the method `initListItr` that will return a `ListIterator` in a given state. See the provided `ListIterator` test classes for examples.

Files

In order to create your `IUDoubleLinkedList` class, you may need the following files:

- EmptyCollectionException.java
- ElementNotFoundException.java
- IndexedUnorderedList.java

For testing, you will need these files:

- TestCase.java
- iudoublelinkedlist.xml

The change scenario classes haven't changed from last time, but you can download those files as well as all of the above files in this zip file.

README

Your plain text README should follow the formatting and content guidelines given here, with special attention given to the following:

- Design: your design for this program.

- **Testing: a detailed description of your testing procedure. You should describe how you tested your class and what bugs you found and fixed by so doing.**

- **Discussion: a discussion of your experience writing this assignment. This could be in the form of a journal.**

Grading

Points will be awarded according to the following breakdown:

Tasks	Points
Documentation - README, Javadocs, comments	20
ListTester - complete, including `ListIterator` functionality	20
`IUDoubleLinkedList` and `ListIterator` Functionality	50
Programming Practices - code formatting, naming conventions, encapsulation, etc.	10

Required Files

Please submit the following files:

- **IUDoubleLinkedList.java**
- **TestCase.java (updated)**
- **DLLNode.java**
- **iudoublelinkedlist.xml (updated)**
- **README**
- **Any other files required to compile / run your `ListTester` class using the `IUDoubleLinkedList` class.**

Submission

Submit all files from the same directory. Do not include any unnecessary files. Use the submission command given on your section's class web page from the directory containing your files.