Joanna Klukowska
joannakl@cs.nyu.edu

Project 2: Circular Doubly Linked List

Due date: March 3, 11:55PM EST.

You may discuss any of the assignments with your classmates and tutors (or anyone else) but all work for all assignments must be entirely your own . Any sharing or copying of assignments will be considered cheating (this includes posting of partial or complete solutions on Piazza, GitHub or any other public forum). If you get significant help from anyone, you should acknowledge it in your submission (and your grade will be proportional to the part that you completed on your own). You are responsible for every line in your program: you need to know what it does and why. You should not use any data structures and features of Java that have not been covered in class (or the prerequisite class). If you have doubts whether or not you are allowed to use certain structures, just ask your instructor.

The goal of this project is to implement a circular doubly linked list. Your implementation should be based on a doubly linked list implementation specified in the textbook. The differences from that implementation are described in the body of this specification.

Objectives

The goal of this programming project is for you to master (or at least get practice on) the following tasks:

- reading, understanding and modifying provided code
- writing classes
- providing a reference based implementation of a circular doubly linked list data structure

Start early! This project may not seem like much coding, but debugging and testing always takes time.

For the purpose of grading, your project should be in the package called `project2`. This means that each of your submitted source code files should start with a line:

```
package project2;
```

`Tester` class

The project comes with a simple `Tester` class. This class is designed to use a few of the methods of the class that you are creating. It is NOT designed to extensively test the class. You should write your own tests to convince yourself that your implementation is correct.

The sequence of instructions in the `Tester` class should produce the following output in the console:

Output:

```
[]
[A, B]
[]
[E, D, C, B, A, Z, Y, X, W, V]
[]
[V, W, X, Y, Z, A, B, C, D, E]
[V, W, X, Y, Z, A, B, C, D, E]
[W, X, Y, Z, A, B, C, D, E, K]
[V, W, X, M, Z, A, B, C, D, E]
[X, Y, Z, A, B, C, D, E, K, W]
[D, E, K, W, X, Y, Z, A, B, C]
```

Joanna Klukowska
joannakl@cs.nyu.edu

`CircularDoublyLinkedList` class

The class that you need to implement is the `CircularDoublyLinkedList` class.

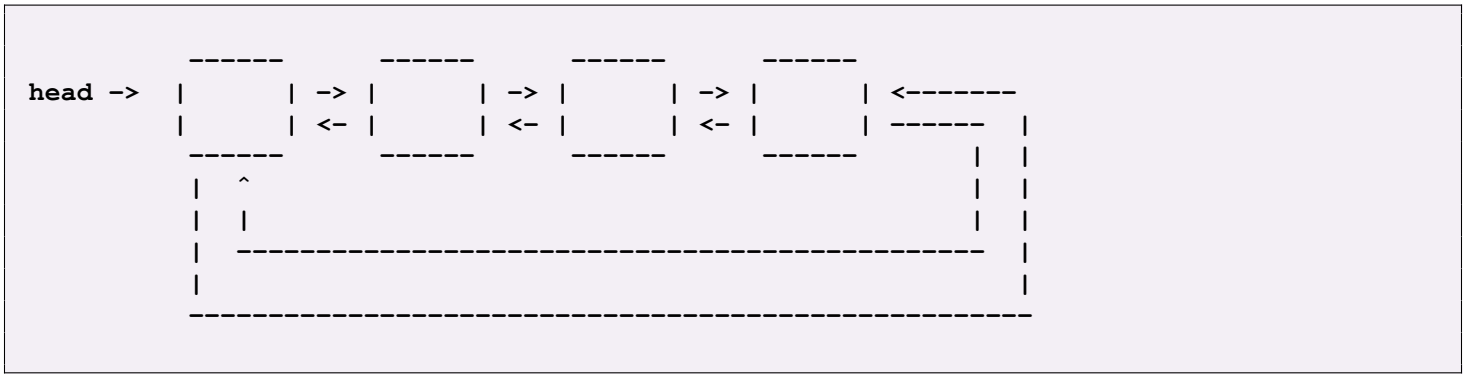
Doubly Linked - No Sentinels

You should base your implementation on `DoublyLinkedList<E>` class that is implemented in the textbook (section 3.4.1). Your class should implement all of the methods provided by the implementation in the book. The major difference between the book's and your implementations is that your implementation **should not use the sentinel nodes**. Your implementation will also need to provide additional methods described below.

Circular

Your class should implement **a circular doubly linked list**. This means that the last node's next reference should point back to the first node, and the first node's previous reference should point to the last node.

Output:



Generic Class

The `DoublyLinkedList<E>` class in the textbook is a generic class. Your own class also needs to be generic.

Additional Methods

Your class should provide all of the public methods provided by the `DoublyLinkedList<E>` class in your textbook. You may need to change the implementation of some of them to make them work without sentinels and with a circular aspect of your own class.

Your class should provide a slightly different implementation of the **`toString()`** method. The `toString()` method should return a string representation of the collection of the elements stored in the list. The string representation should consist of a list of the elements in the order they are stored in the list, enclosed in square brackets ("[]"). Adjacent elements should separated by the characters ", " (comma and space). Elements are converted to strings based on their own `toString()` method.

In addition, the `CircularDoublyLinkedList` class should provide the implementation of the following methods.

`public boolean equals ( Object o )` This method returns true if the two list (the one on which the method is called and the one that is passed as the parameter) are identical. The lists are considered to be identical if they are of the same length and they contain the same elements in the same order. This method overrides the `equals()` method defined in the `Object` class.

`public CircularDoublyLinkedList<E> clone() throws CloneNotSupportedException` This method returns a shallow clone of the list on which it is called. See section 3.6 in the textbook. In order to implement this method correctly, the class needs to implement the `Cloneable` interface. This method overrides the `clone()` method defined in the `Object` class.

Joanna Klukowska
joannakl@cs.nyu.edu

`public E get ( int index )` The method returns the element at the specified position in this list. The first node in the list is at position zero. The method should throw an instance of **`IndexOutOfBoundsException`** if the **`index`** is out of range **`(index < 0 || index >= size())`**.

`public E set ( int index, E element )` The method replaces the element at the specified position in this list with the specified element. The first node in the list is at position zero. The method returns the element previously at the specified position. The method should throw an instance of **`IndexOutOfBoundsException`** if the **`index`** is out of range **`(index < 0 || index >= size())`**.

`public void rotate()` The method should move the first element to the end of the list. If the list is empty or has only one element, it should remain unchanged.

`public void rotateBackward()` The method should move the last element to the front of the list. If the list is empty or has only one element, it should remain unchanged.

Programming Rules

You should follow the rules outlined in the document *Code conventions* posted on the course website at https://joannakl.github.io/cs102_s18/notes/CodeConventions.pdf.

You may use any exception-related classes.

You must implement your own circular doubly linked list!

Working on This Assignment

You should start right away!

You should modularize your design so that you can test it regularly. You can write method stubs that do not perform actual operations, but allow your code to compile together with the **`Tester`** class.

Make sure that at all times you have a working program. You can implement methods that perform one task at a time. This way, if you run out of time, at least parts of your program will be functioning properly.

You should make sure that your program's results are consistent with what is described in this specification by running the program on carefully designed test inputs and examining the outputs produced to make sure they are correct. The goal in doing this is to try to find the mistakes you have most likely made in your code.

You should backup your code after each time you spend some time working on it. Save it to a flash drive, email it to yourself, upload it to your Google drive, do anything that gives you a second (or maybe third copy). Computers tend to break just a few days or even a few hours before the due dates - make sure that you have working code if that happens.

Grading

If your program does not compile or if it crashes (almost) every time it is run, you will get a zero on the assignment.

If the program does not adhere to the specification, the grade will be low and will depend on how easy it is to figure out what the program is doing.

60 points class correctness: correct behavior of methods of the required classes and correct behavior of the program

20 points design and the implementation of the required class

20 points proper documentation, program style and format of submission

Joanna Klukowska
joannakl@cs.nyu.edu

How and What to Submit

For the purpose of grading, your project should be in the package called `project2`. This means that each of your submitted source code files should start with a line:

```
package project2;
```

Your should submit all your source code files (the ones with .java extensions only) in a single **zip** file to Gradescope. The Gradescope autograder will be available a few days before the assignment due date.

You can produce a zip file directly from Eclipse:

- right click on the name of the package (inside the `src` folder) and select Export...
- under General pick Archive File and click Next
- in the window that opens select appropriate files and settings:
  - in the right pane pick ONLY the files that are actually part of the project, but make sure that you select all files that are needed
  - in the left pane, make sure that no other directories are selected
  - click Browse and navigate to a location that you can easily find on your system (Desktop or folder with the course materials or ...)
  - in Options select ”Save in zip format”, ”Compress the contents of the file” and ”Create only selected directories”
- click Finish