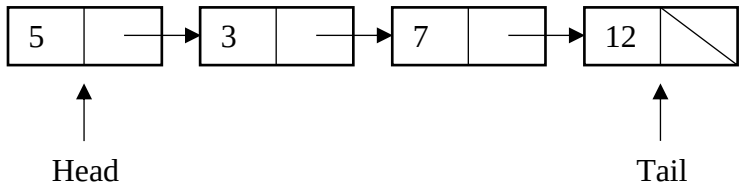
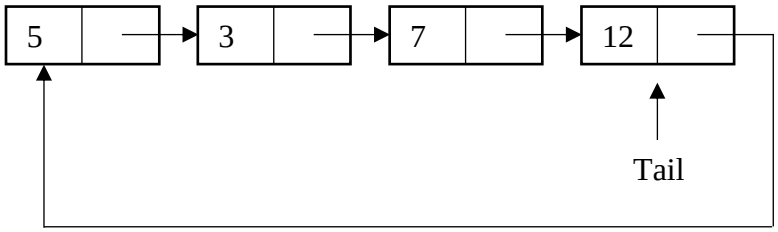
Circular Linked List Assignment

Overview

A circular linked list is essentially a singly linked list in which the next pointer of the tail node is set to point to the head node of the linked list rather than set to null. The first figure below shows a linked list as a singly linked list. The second figure below shows the singly linked list from the first figure as a circular linked list. In this assignment, you will write code for an implementation of a circular linked list.



Singly Linked List



Circular Linked List

Design

1. You will write the implementation for the CircularLinkedList class. Download the files List.java and CircularLinkedList.java for the framework for the CircularLinkedList class.

   You will write the code for all the methods of the CircularLinkedList class except for the methods:

   ```
   public boolean add(AnyType newValue)
   private Node<AnyType> getNode(int index)
   ```

   The definition of each method you have to implement in the CircularLinkedList class:

   - void clear() – Remove all values from the list, creating an empty list.
   - int size() – Return the number of data values currently in the list.
   - boolean isEmpty() – Return true if the list is empty; otherwise return false.
   - AnyType get(int index) – Return the data value at position index in the list.
   - AnyType set(int index, AnyType newValue) – Return the old data value at position index and replace it with the data value newValue.
   - void add(int index, AnyType newValue) – Add data value newValue at the position specified by index; shifting to the right by one position all the values from position index to the end of the list.
   - AnyType remove(int index) – Remove and return the data value at position index in the list.
   - void rotate() – Moves the value at the head of the list to the tail of the list without removing and adding anything to the list.
   - Node<AnyType> getNode(int index, int lower, int upper) – Return the pointer to the node at position index in the list.

   If needed, you can write additional methods to help in your implementation of any method of the CircularLinkedList class but they must be defined as **private**. You **cannot** add any additional data members to the CircularLinkedList class. You **cannot** add any nested classes to the CircularLinkedList class.

   You **cannot** modify the nested Node class.

   You will also write the code for all the methods of the nested LinkedListIterator class. The definition of each method you have to implement in the LinkedListIterator class:

   - boolean hasNext() – Returns true if the iteration has more elements.
   - AnyType next() – Returns the next element in the interation.
   - void remove() – Removes from the underlying list the last element returned by the Iterator.

   You **cannot** add any additional data members and methods to the nested LinkedListIterator class. You **cannot** add any nested classes to the LinkedListIterator class.

2. The input to your program will be read from a plain text file called **assignment1.txt**. This is the statement you'll use to open the file:

   ```
   FileInputStream fstream = new FileInputStream("assignment1.txt");
   ```

   Assuming you're using Eclipse to create your project, you will store the input file assignment1.txt in the parent directory of your source code (**.java** files) which happens to be the main directory of your project in Eclipse. If you're using some other development environment, you will have to figure out where to store the input file.

The input consists of one or more commands (**clear**, **size**, **is_empty**, **get**, **set**, **add**, **add_last**, **remove**, **rotate**, **print_list**) with each command on a separate line. Each command can appear more than once in the input file.

The code you write to implement each command is **NOT** written in the CircularLinkedList class but elsewhere in the program. The code for a command will call the appropriate method of the CircularLinkedList class in order to accomplish the goal of the command.

Have the output for each command print on a separate line. Print a blank line after the output printed for the command.

The **clear** command has your program empty the contents of the circular linked list and print the message **The circular linked list is now empty**.
For example: `clear`

The **size** command has your program print the number of values in the circular linked list.
For example, if the circular linked list contains 5 strings: `size`
Prints: `The circular linked list contains 5 strings.`

The **is_empty** command has your program print the message **The circular linked list is empty** or **The circular linked list is not empty**.
For example: `is_empty`

The **get** command is the word get followed by a space followed by an integer. The integer in the command is a position number. The get command has your program print the string at that position in the circular linked list.
For example, if the string at position 5 is **New York City**: `get 5`
Prints: `The string at position 5 in the circular linked list is "New York City".`

The **set** command is the word set followed by a space followed by an integer followed by a space followed by the rest of the characters on the line, a string. The integer in the command is a position number. The set command has your program replace the string at that position in the circular linked list with the string given in the command.
For example, if the string at position 4 is **New York City**: `set 4 Salt Lake City`
Prints: `The string "Salt Lake City" replaces the string "New York City" at position 4 in the circular linked list.`

The **add** command is the word add followed by a space followed by an integer followed by a space followed by the rest of the characters on the line, a string. The integer in the command is a position number. The add command has your program add that string at that position in the circular linked list.
For example: `add 7 San Francisco`
Prints: `Added the string "San Francisco" at position 7 in the circular linked list.`

The **add_last** command is the word add_last followed by a space followed by the rest of the characters on the line, a string. The add_last command has your program add that string at the end of the circular linked list.
For example: `add_last Washington, D.C.`
Prints: `Added the string "Washington, D.C." to the end of the circular linked list.`

The **remove** command is the word remove followed by a space followed by an integer. The integer in the command is a position number. The remove command has your program remove the string at that position in the circular linked list and print out that string.
For example, if the string at position 5 is **New York City**: `remove 5`
Prints: `Removed the string "New York City" at position 5 from the circular linked list.`

The **rotate** command has your program move the value at the head of the circular linked list to the tail of the circular linked list and print the message **The value at the head of the circular linked list was rotated to the tail of the circular linked list**.
For example: `rotate`

The **print_list** command has your program print the contents of the circular linked list. For this command, the program creates an iterator of the circular linked list and uses the iterator to get and print each string from the circular linked list. The program prints **The circular linked list contains:** on the first line followed by each string in the circular linked list printed on a separate line.
For example: `print_list`

3. Create a driver class and make the name of the driver class **Assignment1** and it should only contain only one method:
   `public static void main(String args[]).`
   The main method will open the file **assignment1.txt**, create a CircularLinkedList object which stores strings, and have a loop to process each command in the file. The main method itself should be fairly short and will pass the command to another class to perform the command.

4. **Tip:** Make your program as modular as possible, not placing all your code in one .java file. You can create as many classes as you need in addition to the classes described above. Methods should be reasonably small following the guidance that "A function should do one thing, and do it well."

5. Do **NOT** use your own packages in your program. If you see the keyword **package** on the top line of any of your .java files then you created a package. Create every .java file in the **src** folder of your Eclipse project

6. Do **NOT** use any graphical user interface code in your program!

Grading Criteria

The total project is worth 20 points, broken down as follows:

If the program does not compile successfully then the grade for the assignment is zero.

If the program compiles successfully then the grade computes as follows:
Proper submission instructions:

1. Was the file submitted a zip file, 1 point.
2. The zip file has the correct filename, 1 point.
3. The contents of the zip file are in the correct format, 1 point.
4. The keyword **package** should not appear at the top of any of the .java files, 1 point.

Program execution:

5. The program uses the correct input file name, 2 points.
6. The program properly opens and reads from the input file, 1 point.
7. The program properly reads and processes the input, 2 points.
8. The program produces the correct result for the input, 2 points.

Code implementation:

9. The driver file has the correct filename, **Assignment1.java**, 1 point.
10. The driver file contains only the method **main** performing the exact tasks as described in the assignment description, 1 point.
11. The code performs all the tasks as described in the assignment description, 3 points.
12. The code is free from logical errors, 2 points.

Code readability (2 points):

13. Essential guidelines:
    a. Good variable, method, and class names.
    b. Variables, classes, and methods that have a single purpose.
    c. Consistent indentation and formatting style.
    d. Reduction of the nesting level in code.

Late submission penalty: assignments submitted after the due date are subjected to a 2 point deduction for each day late.

Submission Instructions

Go to the folder containing the .java files of your assignment and select all (and **ONLY**) the .java files which you created for the assignment in order to place them in a Zip file. The file should **NOT** be a **7z** or **rar** file! Then, follow the directions below for creating a zip file depending on the operating system running on the computer containing your assignment's .java files.

Creating a Zip file in Microsoft Windows (any version):

1. Right-click any of the selected .java files to display a pop-up menu.
2. Click on **Send to**.
3. Click on **Compressed (zipped) Folder**.
4. Rename your Zip file as described below.
5. Follow the directions below to submit your assignment.

Creating a Zip file in Mac OS X:

1. Click **File** on the menu bar.
2. Click on **Compress ? Items** where ? is the number of .java files you selected.
3. Mac OS X creates the file **Archive.zip**.
4. Rename **Archive** as described below.
5. Follow the directions below to submit your assignment.

Save the Zip file with the filename having the following format:

your last name,
followed by an underscore _,
followed by your first name,
followed by an underscore _,
followed by the word **Assignment1**.

For example, if your name is John Doe then the filename would be: **Doe_John_Assignment1**

Once you submit your assignment you will not be able to resubmit it!
Make absolutely sure the assignment you want to submit is the assignment you want graded.
There will be **NO** exceptions to this rule!

You will submit your Zip file via your CUNY Blackboard account.
Follow these instructions:

Log onto your CUNY BlackBoard account.
Click on the CSCI 340 course link in the list of courses you're taking this semester.
Click on **Content** in the green area on the left side of the webpage.
You will see the **Assignment 1 – Circular Linked List Assignment**.
Click on the assignment.
Upload your Zip file and then click the submit button to submit your assignment.

Due Date: Submit this assignment by Tuesday, October 17, 2017.