COMP2402/2002 - Summer 2012 - Assignment 2

Posted: Saturday, May 19^th
Due: Thursday, June 2^nd (up load it on Web-CT)

- Must be uploaded on Web CT
- Make a folder neamed as
  (< $StudentID$ >_<First Name>_<Last Name>_<Assignment #>)
- Put all the source files into that folder
- Add a text file with your Id and full name and any comment you have about your assignment
- Zip the folder
- Upload it on Web-CT.

(All implementation will be done in Java, in code that will run on the SCS lab machines)

In this assignment you will practice implementing two List-like data structures: A **DoublyLinkedList** and a **TripleArrayDeque**.

Both of these structres will implement a **SimpleList** interface, which is a simplified version of the java.util.List interface. However, how your two structres implement the SimpleList interface will be different.

The SimpleList interface

This interface is described in the file **SimpleList.java**.

- Although there are many methods in the the **SimpleList** interface, most are not very complicated.
- To keep things simple, just implement a default (no argument) constructor for your data structures.
- You can start with the skeleton code presented in DoublyLinkedList.java and TripleArrayDeque.java.

Doubly Linked List

Background

A **LinkedList** stores its data items in a sequence of nodes. In **Singly-Linked Lists**, each node only knows about the node that comes after it in the sequence. In a **Doubly-LinkedList**, every node will know both the node that comes after it, as well as the node that comes before it, in the sequence. It may make your code a bit easier to use a **dummy** node at the start of your **DoublyLinkedList**, as described in the course notes.

To implemented

- You will implement a **DoublyLinkedList**.
- Make sure it is named "**DoublyLinkedList**" and saved in a file **DoublyLinkedList.java**.
- Your **DoublyLinkedList** must implement the interface of **SimpleList.java**.

This should be fairly straight-forward since they are covered in the notes.

TripleArrayDeque

Background

Deques are **Double-Ended Queues**. They are efficient at adding and removing from the front and the back of a sequence of items. **ArrayDeques** are data structures that implement the Deque using an array. Since arrays are fixed in size, if more items are added than the current capacity of the array, the array must be resized. **DualArrayDeques** are **ArrayDeques** that use two array-based structures to store the items. The first array-based structure stores the first items in the sequence, and the second array-based structure store the last items in the sequence.

Both the first and last array-based structures should hold around the same number of elements. That is, they should be balanced whenever one of the array-based structure have 3 times more elements than the other. If many more items are added/removed from the front than the end (or vice versa), the two array-based structures may each end up being unbalanced, and require a re-balancing operation. **DualArrayDeques** are discussed in the course notes.

In a **TripleArrayDeque**, like the **DualArrayDeques**, the sequence of items is stored across multiple array-based structures. However, in a **TripleArrayDeque**, there are three such lists structures, one for items at the front, one for items in the middle, and one for items at the back.

To implement

- You will implement a **TripleArrayDeque**
- Instead of using three array-based structures you will use three **DoublyLinkedList** (the one you implemented before).
- Make sure it is named "**TripleArrayDeque**" and saved in a file **TripleArrayDeque.java**.
- Your **TripleArrayDeque** must implement the interface of **SimpleList.java**.
- It will store the sequence of items across three DoublyLinkedList (the one you implemented before) structures.
- It will have to handle rebalancing of the items. That is, you must rebalance the three lists whenever any of the lists have 3 times less element than the other two lists together. For example if in list A there are 5 elements and list B have 7 and C have 8 you must rebalance.

Please include a README file which tells the TA any important details about your submission, and mentions what works and what doesn't (and when).