Computer Science 21A (Spring Term, 2018)
Data Structures

Programming Assignment 1: Part One

Implementing Uno with Linked Lists

This assignment will test your ability to implement many versions of the LinkedList and other data structures, as well as use your implementation in practice.

The assignment will be divided into two parts. For part one, you will be implementing the generic data structures and testing them with JUnit tests. For part two, you will be using your own implementation of the data structures to further implement the logic, wrapping classes, and client code for the game of Uno.

Part One (due Thursday, February 15 @ 11:30pm)

You need to implement the following classes (for each data structure you should create a testing class that uses JUnit to test its correct behavior):

SinglyLinkedNode<T>
Nodes for the SinglyLinkedList, described below.

1. SinglyLinkedNode(T data) - constructor that initializes the node.
2. T getData() - returns the data in the node
3. void setNext(SinglyLinkedNode<T> nextNode) - sets the node as the “next” node in the list, returned by getNext()
4. SinglyLinkedNode<T> getNext() - returns the next node in the list
5. toString() - return the string of the data inside it.

SinglyLinkedList<T>

1. SinglyLinkedList() - constructor (optional)
2. SinglyLinkedNode<T> getHead() - returns the first node in the list (null if empty)
3. void regularInsert(T data) - insert “data” at the end.
4. void randomInsert(T data) - insert “data” at a random point in the LinkedList. It should be equally likely that a node will be inserted at any of the possible locations, including the front and the end.
5. void remove(T data) - delete the “data” node from the list.
6. int size() - gets size of linked list.
7. toString() - return a representation of the list

DoublyLinkedNode<T implements Comparable<T>>

1. DoublyLinkedNode(T data) - constructor that initializes the node.
2. T getData() - returns the data in the node
3. void setNext(DoublyLinkedNode<T> nextNode) - sets the node as the "next" node in the list.
4. void setPrevious(DoublyLinkedNode<T> prevNode) - sets prevNode as the "previous" node in the list.
5. DoublyLinkedNode<T> getNext() - returns the next node in the list
6. DoublyLinkedNode<T> getPrev() - returns the previous node in the list
7. toString() - return the string of the data inside of it

DoublyLinkedOrderedList<T implements Comparable<T>>
This is a special kind of a doubly linked list. This list should guarantee that elements are always sorted. To achieve this, we simply define the insert method so that every element is inserted in the right spot. This means that insertion will be O(n).

1. DoublyLinkedOrderedList() - constructor
2. DoublyLinkedNode<T> getHead() - returns the first node in the list (null if empty)
3. void insert(T data) - insert "data" into the correct spot in the list, you need to make sure that after each insertion the list is sorted by inserting into the correct place
4. DoublyLinkedNode<T> delete(T data) - find and delete the node with the given "data"
5. toString() - a representation of the list

Queue<T>
This will be an array implementation queue. You need to implement the following methods:

1. queue(int size) - Constructor that creates the internal array of size "size", as well as any other variables needed in the queue.
2. void enqueue(T data) - enqueue data
3. T dequeue() - dequeue first item in the queue
4. int getSize() - return size of the queue
5. boolean isEmpty() - returns true if queue is empty
6. boolean isFull() - return true if queue is full.

Note: Your JUnit tests should be comprehensive (this means that you should test edge cases, and make sure every method is working as expected). In order to test your data structures you can initialize them with a type Integer, i.e. "Queue<Integer> n = new Queue<>(10);".

Submission:

Submit via Latte a zip file with all the following files:

- SinglyLinkedNode.java
- SinglyLinkedList.java
- DoublyLinkedNode.java
- DoublyLinkedOrderedList.java
- Queue.java

The zip file should be titled <your-username>-PA1-Part1.zip - for example, if your brandeis email is dilant@brandeis.edu, your zip file should be called dilant-PA1-Part1.zip

IMPORTANT

1. Your code should be well commented:
    - Add your name and email address at the beginning of each .java file.
    - Write comments within your code when needed.
        - You must use Javadoc comments for your classes and methods.
        - You must write each method's running time in your Javadoc comments for that method.
2. Excluding arrays, you may only use data structures that you implement yourself.
3. For each data structure you should create a testing class that uses JUnit to test its correct behavior.

Your data structures should not have hardcoded data types. For example, your implementation of a LinkedList should be able to handle Integers, Doubles, Strings, or any sort of Java object.