Project: Linked List/Array Hybrid
Due: 11:59 pm 12/12/19
Total Points: 10
Projects must be submitted on BlackBoard as a ZIPPED FOLDER with the folder name as X{LastName_FirstName}
for example your student id is Alex Chen than the folder name is XChen_Alex
Within the folder will only be source code, NO .class files. The files in the folder will be:

1) X{LastName_FirstName}.java
2) Utility.java
3) LinkedListArray.java
4) Node.java

Any projects submitted that **DOES NOT** have this naming convention will not be graded.
If you do not submit anything, you will receive 1 point for the project. Any projects that **do not compile or work** will receive a 0. Excuses such as "It compiles on my computer" or "It worked last time" will not be accepted. Your program must work on all machines not just yours.

If you are using an IDE such as eclipse, before submitting, remove all package statements from all files.
Late penalty Any project submitted:
1 day late will receive a max possible score of 8 and extra credit will not be rewarded
2 days late will receive a max possible score of 7 and extra credit will not be rewarded
3 days late will receive a max possible score of 6 and extra credit will not be rewarded
Any project submitted after 3 days will not be graded.
Cheating Any one caught cheating, copying code or letting others copy, will receive a 0 and reported. Collaborating with others is encourage on a high level, but code and implementation should never be shared.

Project Specs:

Create a linked list that has both the features of a linked list but with the added bonus of looking up an element by using its index (analogous to how arrays work).

Features of this linked list:

- **Doubly Linked**
- **Constant time insertion and removal of the first and last node**
- **Constant time remove of any given node**
- **Constant time insertion of any new node, given a location to insert in front or behind of**
- **Constant time look up of any node given its position on the list**

API of this Data Structure:

bool isEmpty() : return true if list is empty, otherwise false

bool contains(Node<T> node) : returns true if the given node is in the list , false otherwise

void insert(T x) : insert an element to the rear of the list

void insertHead(T x) : insert an element to the head of the list

void addBefore(T x, Node<T> node) : insert the new element before the given node

void addAfter(T x, Node<T> node) : insert the new element after the given node

bool removeFirst() : Remove the first element in the list. Returns true if removal was successful, false otherwise

bool removeLast() : Remove the last element in the list. Returns true if removal was successful, false otherwise

bool remove(int position) : Removes the element at the given position : Returns true if removal was successful, false otherwise

bool remove(Node<T> node) : Removes the given node from the list : Returns true if removal was successful, false otherwise.

void sort () : Sort the current state of the linked list

T get(int position) : Return the data at that given position

<u>Constraints:</u>
All methods MUST be constant in time complexity with the exception of addBefore or addAfter; these two methods may be in linear time. You data structure should work for any data type not only primitives.

The utility class is for you to write out tests for this data structure:

- **You must write at least 5 static methods in that class that uses your data structure.**
- **Each test should test out different cases or condition your list can experience**
- **Test that it can handle any situation such as calling remove on an empty list, or get from an unknown position.**
- **Show that your data structure works the way you expect it to work in normal circumstances.**