Important

There are a few guidelines you must follow in this homework. If you fail to follow any of the following guidelines you will receive a **0** for the entire assignment.

1. All submitted code must compile under **JDK 7**. This includes unused code, don't submit extra files that don't compile. (Java is backwards compatabile so if it compiles under JDK 6 it *should* compile under JDK 7)
2. Don't include any package declarations in your classes.
3. Don't change any *existing* class headers or method signatures. (It is fine to add extra methods and classes)
4. Don't import anything that would trivialize the assignment. (e.g. don't import `java.util.LinkedList` for a Linked List assignment. Ask if you are unsure.)
5. You must submit your source code, the `.java` files, not the compiled `.class` files.

After you submit your files redownload them and run them to make sure they are what you intended to submit. We are not responsible if you submit the wrong files.

Linked List

A linked list is a data structure consisting of a group of nodes which together represent a list. Under the simplest form, each node is composed of some data and a link to the next node in the list.

The principal benefit of a linked list over a conventional array is that the list elements can easily be inserted or removed without reallocation or reorganization of the entire structure, because the data items need not be stored contiguously in memory.

In this assignment you will be implementing a **doubly linked list, with both a head and tail pointer.**

Node

For this assignment you will need to write a Node class. This can be a private class inside of LinkedList, or it can be a full public class in its own file. The choice is yours.

The Node class will keep a reference to the previous node in the list, the next node in the list, and a piece of data. Here is a very simple private node class that you can stick in your LinkedList class and use to complete this assignment just fine; however, I would recommend adding some useful constructors.

```
private class Node {
   Node next, prev;
   T data;
}
```

Hint: If you choose to make your node class its own file, it will need to be generic.

Fields

The Linked List class should have 3 fields: head, tail, and size.

```
private Node head, tail;    // These are always the first and last nodes
private int size;           // This is the number of items in the list
```

Constructors

There is no need to have any constructors other than the default constructor for a Linked List. All of the fields can just take their default initial value. If you choose to create your own constructor(s) for testing purposes, **make sure you do not hide the default constructor or your code won't compile**.

To be safe you can just copy the following constructor into your code and it should be fine. (assuming you are using the recommended fields)

```
public LinkedList() {
   // Do nothing.
}
```

Methods

Read the javadocs!

There are a large number of methods in this assignment, I will try to provide a usefull tip/hint for each of them. These are just suggestions that may or may not make the assignment easier for you.

```
void add(T)
   // this is the same as addLast
void add(T, int)
   // if index is 0 or size, use addFirst or addLast
void addFirst(T), void addLast(T)
   // special case for an empty list

void clear()
   // set all the fields to null, and size to 0

boolean contains(Object)
   // don't forget about nulls, can safely use:
   // (data == item || (data != null && data.equals(item)))
   // could use a for-each loop over this
int indexOf(Object)
   // can use a for-each loop over this, return the first occurence

T get(int)
   // check index, then do a simple for loop
T getFirst()
   // this can throw an exception if the list is empty
T getLast()
   // this can throw an exception if the list is empty

T remove(Object)
   // using indexOf(Object) this becomse the same as remove(int)
T remove(int)
   // using get(int) this becomes the same as remove(Object)
T removeFirst(), T removeLast()
   // special case if size is 1

boolean isEmpty()
   // one line: return size == 0;

Iterator<T> iterator()
   // See the guide on Iterable and Iterator
```

Exceptions

In this assignment rather than ignoring bad cases, an exception should be thrown. For example if someone trys to add to get an item at an invalid index an IndexOutOfBoundsException will be thrown. Example code follows. See the javadocs of each method for exactly when you should throw an exception.

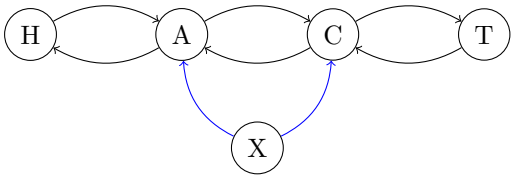
```
public T get(int index) {
    if (index < 0 || index >= size) {
        throw new IndexOutOfBoundsException();
    }
    // The rest of the method goes here.
}
```

Add

This is a simple diagram on how add works.

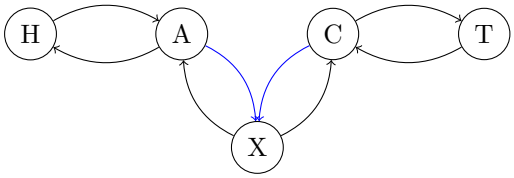
Initial List

Adding X - step 1 - create new links for X




```
x.prev = a;
x.next = c;
```

Adding X - step 2 - move links from the existing nodes




```
a.next = x;
c.prev = x;
```

Deliverables

You must submit all of the following files.

1. `LinkedList.java`
2. `Node.java (optional)`

You may attach them each individually, or submit them in a zip archive.