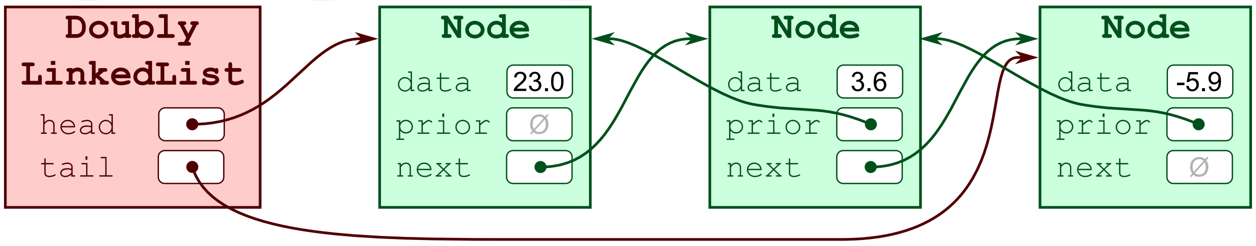
HW 9: Doubly Linked List

You must make a generically-typed `DoublyLinkedList` object. This is a linked list, in which every node contains both a pointer to the prior node and one to the next node. (The prior node of the head and the next node of the tail should both be null.) This allows your functions to operate more quickly. The public methods you implement should be:

- `void add(E value, int location)`, which adds a new `Node` to an arbitrary location, containing the given `E`.
- `void addToHead(E value)`, which adds a new `Node` to the head, containing the given `E`.
- `void addToTail(E value)`, which adds a new `Node` to the tail, containing the given `E`.
- `boolean contains(E value)`, which returns `true` if the passed value is to be found anywhere in the list. Note that this should return `true` if it encounters any object with the same value (i.e. it uses the object's `equals()` function).
- `int find(E value)`, which returns the index of the first `Node` containing the given value, or -1 if it is not found.
- `E get(int location)`, which returns the `E` located at the given location, without modifying the list.
- `E getHead()`, which returns the `E` located at the head, without modifying the list.
- `int getSize()`, which returns the size of the list.
- `E getTail()`, which returns the `E` located at the tail, without modifying the list.
- `E remove(int location)`, which deletes the `Node` at the given location, returning the `E` that was contained therein.
- `E removeHead()`, which deletes the `Node` at the head, returning the `E` that was contained therein.
- `E removeTail()`, which deletes the `Node` at the tail, returning the `E` that was contained therein.
- `String toString()`, which returns a `String` representing this `DoublyLinkedList`, with each object in parentheses separated by "arrows" (`->`). (The `DoublyLinkedList` in the figure would return the `String` `"(23.0) -> (3.6) -> (-5.9)"`.)



- `static void main(String[] args)`, which is a unit-testing `main()` function, that adequately tests all of the above functions. Make sure that it prints out text that is understandable to a reader, without having to read the source code. (i.e. it should have output such as `"6.5 removed from tail"`, or `"9.2 added to head"`, or `"List is now (9.2) -> (6.1) -> (0.3)"`.)

The functions `add()`, `get()`, and `remove()` should be linear-time functions. However, they should all have the capability to start the traversal at the beginning or the end—whichever one will be faster. The functions `contains()`, `find()`, and `toString()` should start at the head. The other functions should be constant time functions.

Note that seven of these functions can just be one-liners, that call a different function in a certain way.