Lab 3: Intersection of Linked Lists

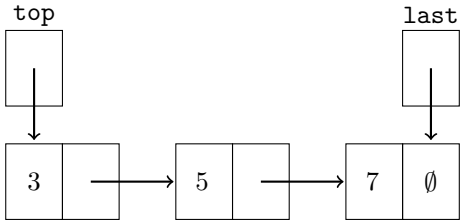
Objective

To implement intersection of sets implemented as simple linked lists with no dummy nodes.

Intersection of Two Linked Lists

File `Lab03.java` contains a nearly-complete application that creates pairs of sets of integers (implemented as simple linked lists) and then tests union and intersection methods on them. Union is already implemented, you will implement the intersection method only.

If the set is $\{3, 5, 7\}$, then the linked list implementation looks like the following:



This set implementation uses a simple linked with

- `top`, a pointer to the first `Node` in the list,
- `last`, a pointer to the last `Node` in the list, and
- No dummy nodes.

Furthermore, the list is *ordered* and does not allow duplicates. (Basically, class `Set` is a renamed class `LinkedList`.)

Inside the `Set` class is a `private Node` class with `public` instance members, `item` and `next`. Because the `Node` class is `private` inside the `LinkedList` class, no code outside the `LinkedList` class can access or know about `Node`s. Because `item` and `next` are `public`, you can access the `item` and `next` of any node anywhere inside the `LinkedList` class. (This structure is not good object-oriented practice. However, it is very simple and will allow you to transfer your knowledge to non-object-oriented languages easily.)

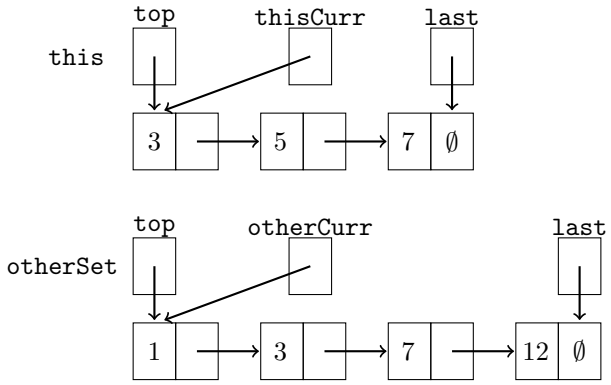
The method you must write: In the `Set` class, you must complete method `intersection`, which is passed two `Set`s:

- One set is pointed at by implicit parameter `this`, and
- The second set is passed in parameter `otherSet`.

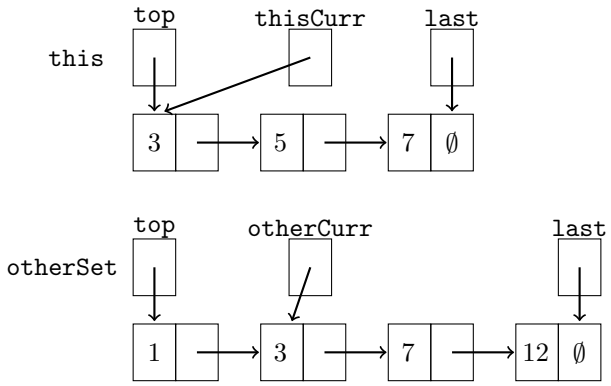
It returns another set containing all the elements that are in **both** `this` and `otherSet`, without changing `this` or `otherSet`. For example, if `this` is a linked list representing $\{1, 3, 5\}$ and `otherSet` is a linked list representing $\{3, 55\}$, then method `intersection` should return a linked list representing $\{3\}$ (and `this` and `otherSet` should be unchanged).

How does it work? A bit like merge in merge sort. Since the linked lists are ordered linked lists, you simply need to loop through `this` and `otherSet` at the same time, adding a new element to the (initially empty) result set whenever you see that `this` and `otherSet` have the same element.

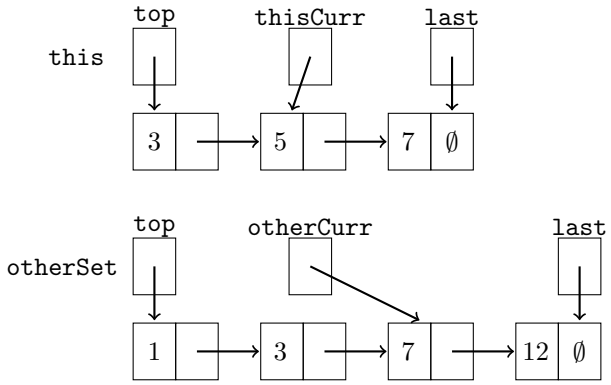
The intersection begins with pointers `thisCurr` and `otherCurr` at the start of the two sets:



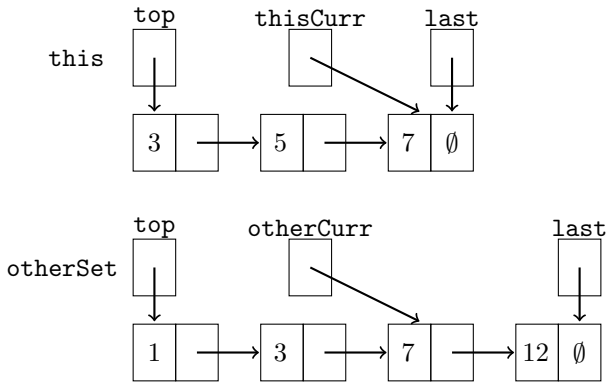
Since `otherCurr`'s item is smaller than `thisCurr`'s item, nothing is added to the intersection set and `otherCurr` is moved to the next node:



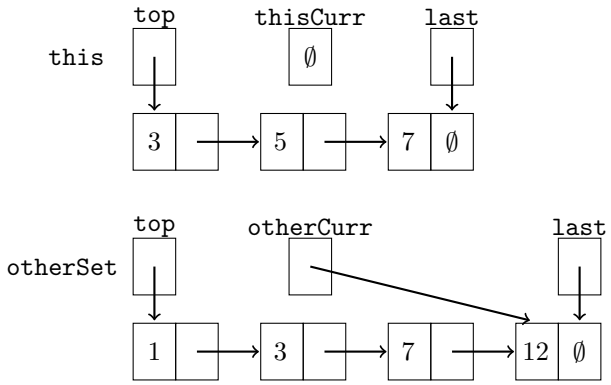
Since `thisCurr`'s item is equal to `otherCurr`'s item, the item is added to the intersection set and both `thisCurr` and `otherCurr` are moved to the next node:



Since `thisCurr`'s item is less than `otherCurr`'s item, nothing is added to the intersection set and `thisCurr` is moved to the next node:



Since `thisCurr`'s item is equal to `otherCurr`'s item, the item is added to the intersection set and both `thisCurr` and `otherSet` are moved to the next node:



Since `thisCurr` is now `null`, there cannot be any more items that are common to both sets, so the method returns the intersection set (which now contains 3 and 7).

So you need a loop that goes until one of `thisCurr` or `otherCurr` is `null`, moving the pointers as appropriate and only adding an item to the intersection set when `thisCurr`'s and `otherCurr`'s items are equal. (You can use `private` helper method `addLast` to add items to the intersection set, since the items you are adding are added in increasing order.)

Reminder: Your method should have only ONE `return` statement. You should not have any `break` or `continue` statements.