Linked Lists – Class Replication

Purpose

The Linked List is the foundation for many other data structures. By understanding how they work from the ground up, it paves the way for understanding it's many variations and how its evolution into entirely new data structures.

How to Get started

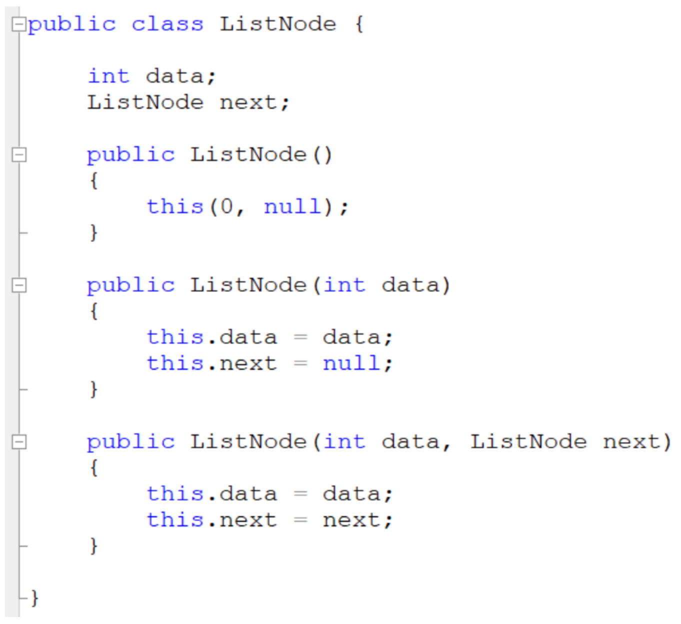
Start with the LinkedListReplicated.java file. Create a Java project that also contains the ListNode.java file and make sure you can compile and run this program.

Basic Specifications

Create a new class, in a separate file named **LinkedIntList.java**. This class must implement a connected list of `ListNode` objects, and you must not import the built-in LinkedList class.

The list is internally implemented as a chain of linked nodes. The `LinkedIntList` keeps a reference to its `front` ("node") as an attribute of the class; `null` is at the end of the list and a `null` front reference signifies an empty list.

The `ListNode` class implements the following code:



```java
public class ListNode {

    int data;
    ListNode next;

    public ListNode()
    {
        this(0, null);
    }

    public ListNode(int data)
    {
        this.data = data;
        this.next = null;
    }

    public ListNode(int data, ListNode next)
    {
        this.data = data;
        this.next = next;
    }

}
```

You may only import the exceptions from util. Nothing else!

Comment each method in this file. *Also provide a comment at the top of this file that gives the name of the file, your name, and an explanation for the purpose of this file as part of the whole application.* The methods to provide are the following:

- **LinkedIntList**(): Default Constructor, sets `front` to `null`.
- **LinkedIntList**(int value): Constructor that sets `front` to a new node with *value* as the parameter.
- **add**(int value): |**void**| Adds the given value to the end of the list. What happens if the list is empty?
- **add**(int index, int value): |**void**| Inserts the specified value in the indicated position of the list. If the index is out of bounds, throw an IndexOutOfBoundsException.
- **get**(int index): |**int**| Return the value at the specified index. If the index is out of bounds, throw an IndexOutOfBoundsException.
- **indexOf** (int value): |**int**| Returns the index of the first occurrence of the value in the list, or -1 if it never occurs**.**
- **remove**(int index): |**int**| Removes the node at the specified index and returns it. If the index is out of bounds, throw an IndexOutOfBoundsException.
- **size**( ): |**int**| Returns the number nodes currently chained in the list.
- **toString**( ): |**String**| Returns a String of the list in this format "item1, item2, …, item99". You should NOT be printing in this method.

Make use of your new `LinkedIntList` class and show the tests that you used to verify each method is working as intended in the main method.

Miscellaneous Advice

Think carefully first and then code. Drawing a graph or diagram on paper can help a LOT when thinking about Linked Lists. Remember that if nothing is pointing to a node, it is inaccessible.

You can always create more methods than the required methods.

You can add a `size` field to the `LinkedIntList` class to return its size more efficiently.

Grading Rubric

In this assignment, you can earn 100 points as follows.

- **5** points: File Information correctly commented at the top of the file
- **5** points: Default and Secondary Constructor working as intended.
- **60** points: LinkedIntList.java file and class exist. All specified methods should work correctly (add, add, get, indexOf, remove, size, toString). If any of these are missing or incorrect, subtract 8 for each one missing or incorrect.
- **20** points: Testing each method in your `main` method, including both constructors. Testing is incredibly important with this assignment.
- **10** points for writing very clear readable code with clear comments in English for each new class, method and function, and static variable. Any other variables should have names suggestive of their meaning, with comments unless the variable is a loop variable (i.e., int i) or a size or length (int n).

BONUS

Implement the methods `clear`() and `sort`(): +7 for both