Linked List

Programming Fundamentals 2

Goals

- ✯ Manipulation and implementation of doubly linked list data structure.
- ✯ Writing a unit test.
- ✯ **Relevant videos**:
  - Linked list data structure.
  - Object-oriented bottom-up: imperative data types.
  - Object-oriented bottom-up: object type.

Deliverables

1. The code on your Github repository generated by clicking here: `https://classroom.github.com/a/3ggMGKcq`
2. **Reviewer**: Maria Hartmann (mhart01 on Github).
3. **Automated testing** using JUnit and unit testing.

Exercise 1 – From scratch

1. Open the file `LinkedList.java` and implement the missing methods. You must write every public method that you implemented in the class `DynamicArray` during the lab “Dynamic Array”.
2. Test your implementation by running a unit test with the provided file `LinkedListTest.java`.
3. Add two new unit tests.

Exercise 2 – Sentinel node

There is a slightly annoying aspect to the previous implementation: you need to check whether the list is empty or not, because if it is empty then `head` is `null`. It forces you to make a special case for the empty list. In order to avoid that, we can add a “fake” node in the list, carrying no value. Therefore, when the list is empty, it contains only this fake node. This node is called a sentinel node. Modify your previous implementation to use this technique.

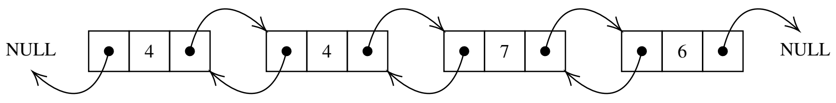
Exercise 3 – Pointing to the last element

When you add an element in a linked list, the operation is very fast once you have reached the position in the list at which you want to insert the element. However, to find that position, you need to loop over elements of the list, which can be time-consuming if the list is very big. In particular, to add an element at the end of the list, you need to loop over the whole list which is not efficient.

1. Modify the implementation of you class `LinkedList` to speed up the operation of adding an element at the end of the list (method `add`), *i.e.*, so that you do not need to loop over all elements of the array.
2. You can use the same unit test with the file `LinkedListTest.java` to check that your new implementation is still working as it should.

Exercise 4 – Doubly Linked List

Another way to improve the efficiency of a linked list is for each node to keep a reference to the previous node. This structure is called a **doubly linked list** and is more convenient for some operations where you need to access the previous node. Indeed, there is no need to traverse the list again to get the previous node, nor to keep track of previous nodes while traversing the list.



1. In the file `DoublyLinkedList.java`, implement a doubly linked list with all methods written in the class `LinkedList`.
2. Test your implementation with the file `DoublyLinkedListTest.java`.
   This file is similar to the file `LinkedListTest.java` which is normal since from the user perspective, the usage of a doubly linked list is similar to the usage of a linked list.