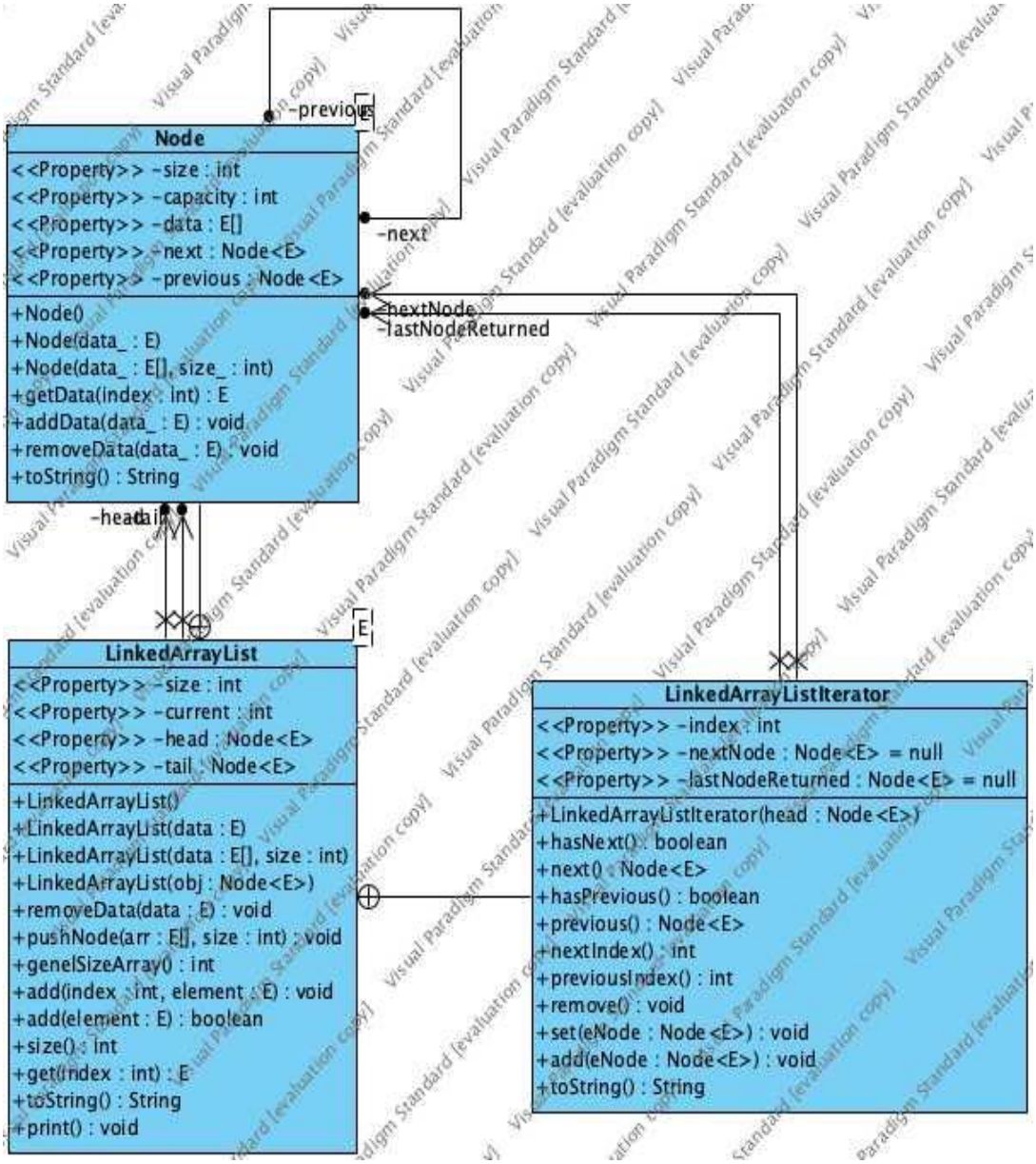
HOMEWORK 3
REPORT

DİLARA KARAKAŞ
171044010

Q3

CLASS DIAGRAM



PROBLEM SOLUTION APPROACH

We class keep a linked list where each node contains an array of elements with constant capacity (5). These arrays should be partially filled arrays. Arrays in the nodes of the linked list, may contain different number of elements. We define a class Node<E> as an inner class that can be placed inside a generic list class. When each node is created, the type parameter E specifies the type of data stored in the node. The keyword static in the class header indicates that the Node<E> class will not reference its outer class. (It can't because it has no methods other than constructors.) In the Java API documentation, static inner classes are also called *nested classes*.

Generally, we want to keep the details of the Node class private. Thus, the qualifier private is applied to the class as well as to the data fields and the constructor. However, the data fields and methods of an inner class are visible anywhere within the enclosing class (also called the *parent class*).

The first constructor stores the data passed to it in the instance variable data of a new node. It also sets the next field to null. The second constructor sets the next field to reference the same node as its second argument. We didn't define a default constructor because none is needed.

We conclude our illustration of the double-linked list data structure. we will write the get, set, remove, and add methods.

A double-linked list object would consist of a separate object with data fields head (a reference to the first list Node), tail(a reference to the last list Node), and size (the number of Nodes). Because both ends of the list are directly accessible, now insertion at either end is **O**(1); insertion else- where is still **O**(*n*).

The Java API also contains the ListIterator<E> interface, which is an extension of theIterator<E> interface that overcomes these limitations. Like the Iterator, the ListIterator should be thought of as being positioned between elements of the linked list. The positions are assigned an index from 0 to size, where the position just before the first element has index 0 and the position just after the last element has index size. The next method moves the iterator forward and returns the element that was jumped over. The previous method moves the iterator backward and also returns the element that was jumped over.

TEST CASE

Three different arrays were created. These were primarily pushes. The addition of the element was then observed. When the index is not given, the element is added to the end, and when given, it is added to the given index. The remaining elements were scrolled. When the Array is full, a new node is created. Remove Also checks for the given element and deletes the element it first found. Shifts the rest of the elements. If there are no elements the node, it will delete that node. The rest of our override function is tried to work our functions. All these operations were applied one by one for integer, double and string.

TEST RESULTS

```
**********INTEGER TESTING******

******ADDING******

Array1 Push

0 1 2 3 null


Array2 Push

0 1 2 3 null
2 3 4 5 null


Array3 Push

0 1 2 3 null
2 3 4 5 null
9 10 null null null

Adding to 4 (Index: 11)

0 1 2 3 null
2 3 4 5 null
4 9 10 null null


Adding to 9 (Index: 1)

9 0 1 2 3
2 3 4 5 null
4 9 10 null null
```

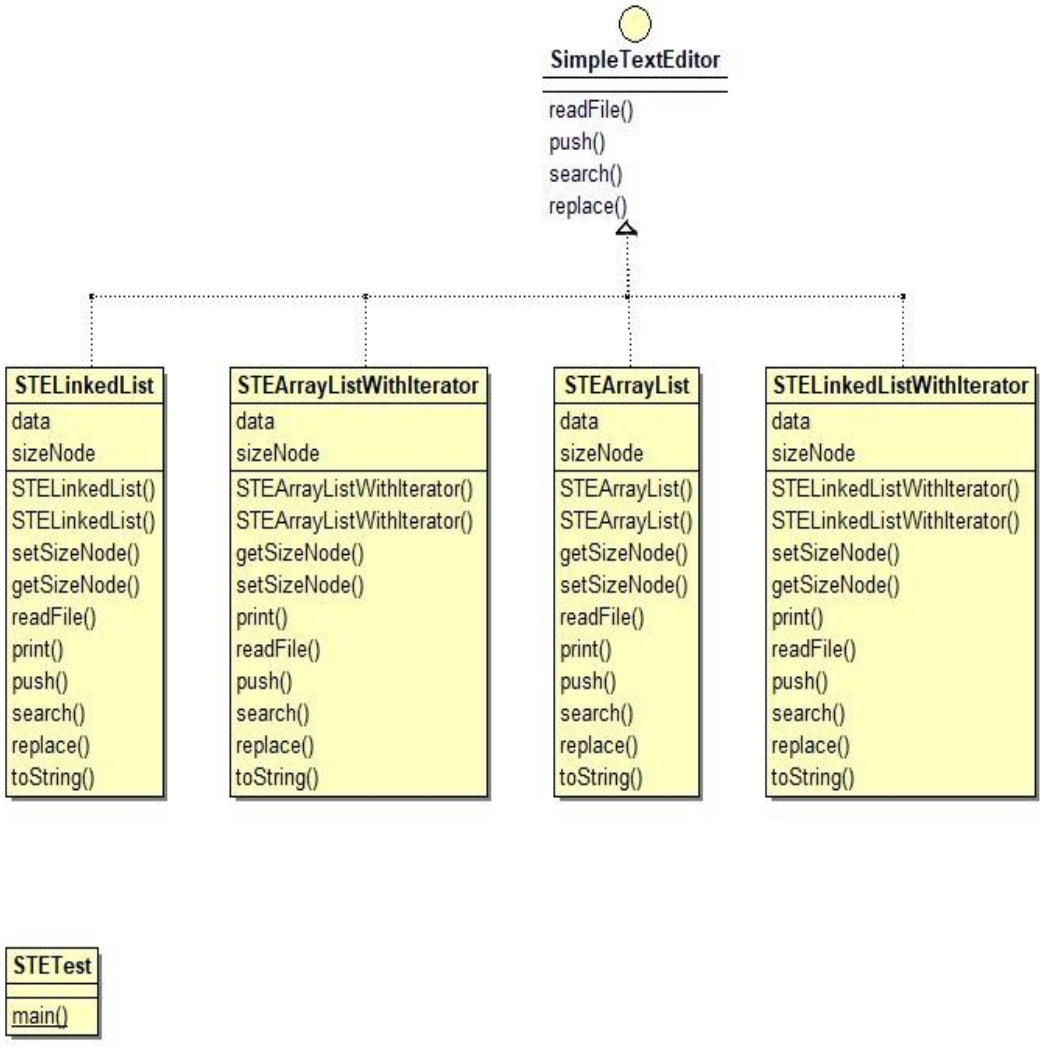
```
Adding to 6 (Index: 1)

6 9 0 1 2
3 2 3 4 5
4 9 10 null null


Adding to 41 (Index: 15)

6 9 0 1 2
3 2 3 4 5
4 9 10 null 41


Adding to 6 (Index: 14)

6 9 0 1 2
3 2 3 4 5
4 9 10 6 41


Adding to 34 (Index: 16)

6 9 0 1 2
3 2 3 4 5
4 9 10 6 41
34 null null null null
```

```
******REMOVING******

Removing to 6

9 0 1 2 null
3 2 3 4 5
4 9 10 6 41
34 null null null null


Removing to 2

9 0 1 null null
3 2 3 4 5
4 9 10 6 41
34 null null null null


Removing to 34

9 0 1 null null
3 2 3 4 5
4 9 10 6 41


Removing to 2

9 0 1 null null
3 3 4 5 null
4 9 10 6 41


Removing to 3

9 0 1 null null
3 4 5 null null
4 9 10 6 41
```

```
******GETTING******

9 0 1 null null
3 4 5 null null
4 9 10 6 41
Getting of 6. element (Index: 6)
Element : 3

Getting of 2. element (Index: 2)
Element : 0

Getting of 5. element (Index: 5)
Element : null
```

Q2

CLASS DIAGRAM



PROBLEM SOLUTION APPROACH

We implement SimpleTextEditor interface. We have implemented four classes from this interface. Two of these classes contain arrayList. Their only difference is that iterator is used in one and not in the other. Similarly, the other two classes are created but LinkedList is used instead of arrayList. These classes include file reading, word search and returning the first index, replacing two desired chars, and adding strings to the desired location. The necessary print function has been added for the test.

TEST CASE

Theorical

Without Iterator = O(n)

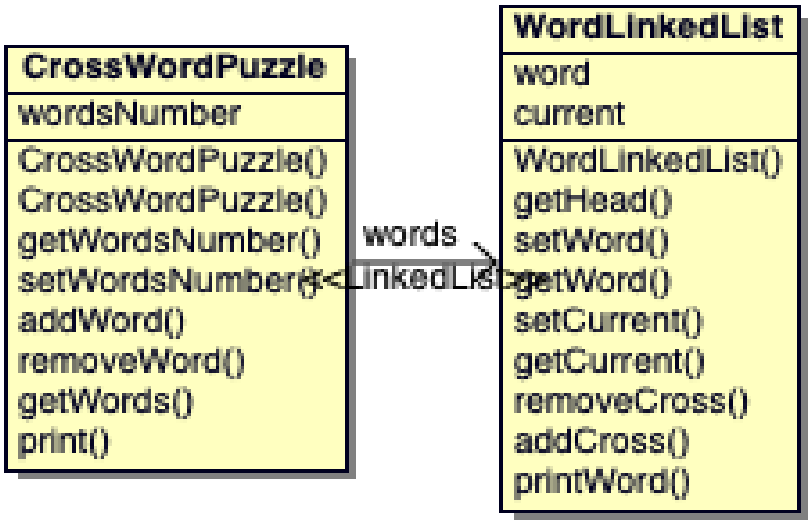
With Iterator = O(n^2)

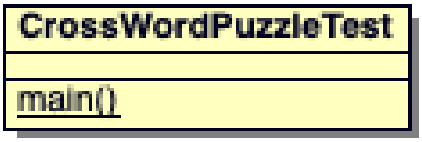
TEST RESULTS

```
************************** TIME TEST FOR FILE 1************************
************* STRArrayList Test Without Iterator *************
368182
************* STRArrayList Test With Iterator *************
288680
************* STRLinkList Test Without Iterator *************
366133
************* STRLinkList Test With Iterator *************
346608
************************** TIME TEST FOR FILE 2************************
************* STRArrayList Test Without Iterator *************
418783
************* STRArrayList Test With Iterator *************
229863
************* STRLinkList Test Without Iterator *************
295494
************* STRLinkList Test With Iterator *************
244421
Nis 02, 2020 6:25:04 ÖS STFTest main
```

Q3

CLASS DIAGRAM





PROBLEM SOLUTION APPROACH

As in the first question, we defined Node inner class in our WordLinkedList class. This class includes a char, a Node next, Node previous, and Node cross. If node cross is not null, crossIndex is not -1. We check whether there is a cross with -1. When we implemented it, we needed the Iterator class (like the first question). We made this a private inner class. Our CrossWordPuzzle class contains a LinkedList of the WordLinkedList class type.

TEST CASE

We added "hello" as the first word. Then "hair". Hair is expected as the cross of Hello, index: 0. Then added "car". Car is expected as a cross of hair, index: 1.

TEST RESULTS