GIT Department Of Computer Engineering
CSE 222/505 - Spring 2020
Homework 3 Report

Fatih Kaan Salgır
171044009

PART 1

1 Problem Solution Approach

The assignment is developing an data structure which consist of arrays of nodes. Since it extends `AbstractList<E>`, I have started to implement methods not implemented in `AbstractList<E>`. These methods are;

- `void add(int index, E e)`
- `E get(int index)`
- `E remove(int index)`
- `E set(int index, E element)`
- `int size()`

All other methods are already inherited or implemented in `AbstractList<E>` class, or depends on the listed methods.

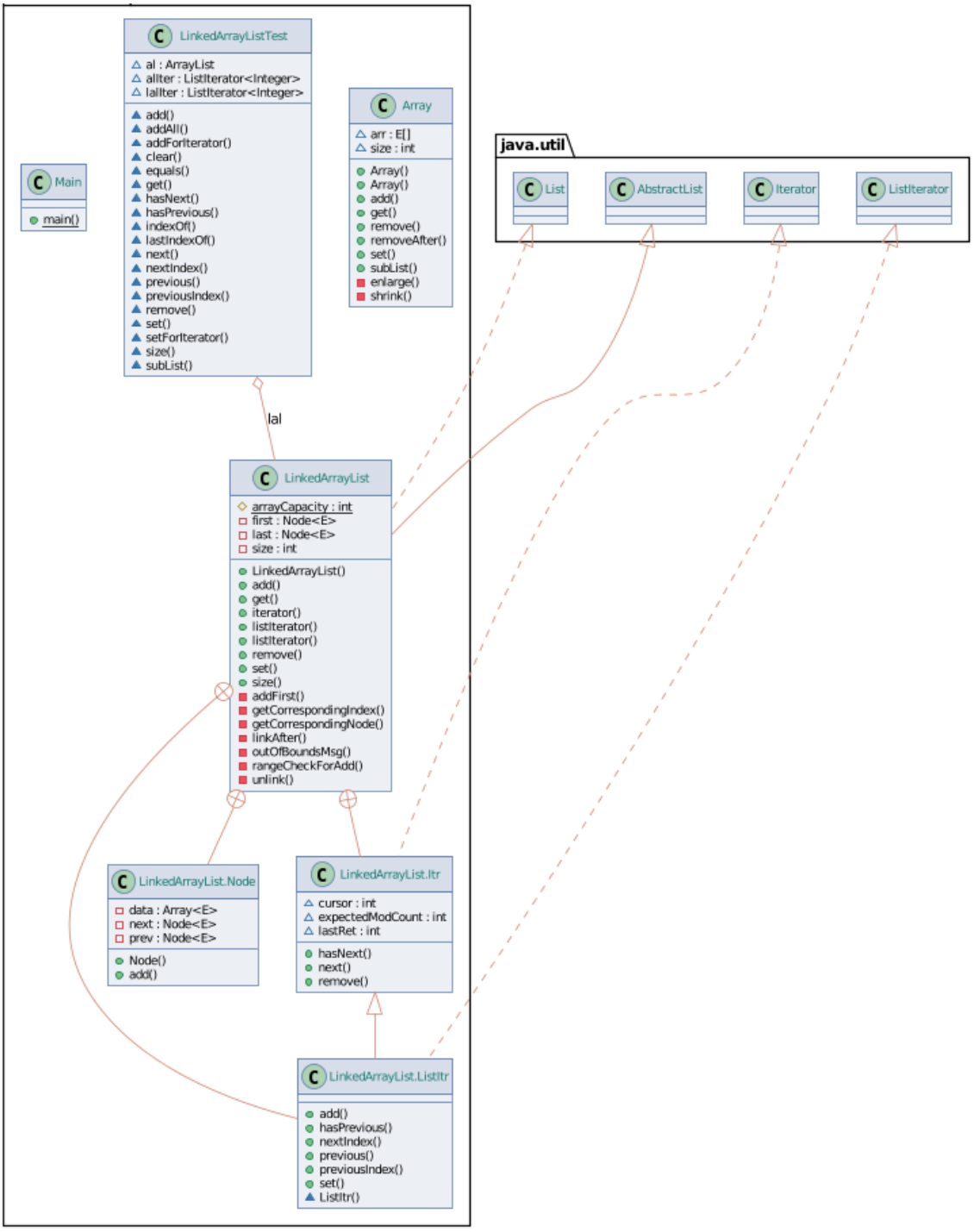
I separated array to another class since its behavior is different than linked list structure. I made Node class and fields private, because its better practice to encapsulate classes.

I used `display()` method which is not part of methods I had to implemented it, but I have used this method to debug almost every method, to observe current states of the list structure. It prints total size, size of each node and elements in the nodes. One of its example output is shown below;

```
12[3, 1, 2, 8, null]4->[7, 6, 5, 10, 8]5->[9, 3, 2, null, null]3->
```

Every method with index as parameter has a check mechanism, if given index is in bounds to operate that particular method. I have implemented this control either with `Objects.checkIndex(index, size)` or `rangeCheckForAdd(index)`.

2 Class Diagrams



3 Test Cases

To test all the methods I have created `LinkedArrayListTest` class which uses JUnit methods. I have created an `ArrayList` and `LinkedArrayList` to compare if the list is operates as expected. I called same methods over these two and check them if they are same. After the method calls I used `assertEquals(lal, al)`.

```
@org.junit.jupiter.api.Test void get() {

void set() {
  add();
  assertEquals(al.set(3, 11), lal.set(3, 11));
  assertEquals(al, lal);
}

@org.junit.jupiter.api.Test
void remove() {
  add();
  assertEquals(al.remove(1), lal.remove(1));
  assertEquals(al.remove(new Integer(3)), lal.remove(new Integer(3)));
}

@org.junit.jupiter.api.Test
void size() {
  add();
  assertEquals(al.size(), lal.size());
  assertEquals(al, lal);
}
```

Listing 1: Some of the JUnit methods for demonstration

I have created some text cases for some adding and removing operations which is present on my main method.

Test Case ID	Test Scenario	Test Steps	Expected Results	Actual Results	Pass/Fail
T01	add end of the list	1. add(1) 2. add(2)	[1,2]	As expected	Pass
T02	remove from end of the list	1. add(1) 2. add(2) 3. remove(1)	[1]	As expected	Pass
T03	add beginning of the list	1. add(1) 2. add(0, 2)	[2,1]	As expected	Pass
T04	add middle of the list	1. add(0) 2. add(2) 3. add(1,1)	[0,1,2]	As expected	Pass
T05	add or remove greater or less than size	add(-1,0) add(100,0) remove(-2) remove(100)	Throw exception	As expected	Pass

4 Running and Results

```java
System.out.println("Test Cases");

System.out.println("T01");
LinkedArrayList<Integer> lal = new LinkedArrayList<>();
lal.add(1);
lal.add(2);
System.out.println(lal);
lal.clear();

System.out.println("T02");
lal.add(1);
lal.add(2);
lal.remove( index: 1);
System.out.println(lal);
lal.clear();

System.out.println("T03");
lal.add(1);
lal.add( index: 0,  e: 2);
System.out.println(lal);
lal.clear();

System.out.println("T04");
lal.add(0);
lal.add(2);
lal.add( index: 1,  e: 1);
System.out.println(lal);
lal.clear();

System.out.println("T05");
lal.add( index: -1,  e: 0);
lal.add( index: 100,  e: 0);
lal.remove( index: -2);
lal.remove( index: 100);
```

Figure 1: main - test cases

```
Test Cases
T01
[1, 2]
T02
[1]
T03
[2, 1]
T04
[0, 1, 2]
T05
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index: -1, Size: 0
    at com.company.LinkedArrayList.rangeCheckForAdd(LinkedArrayList.java:249)
    at com.company.LinkedArrayList.add(LinkedArrayList.java:53)
    at com.company.Main.main(Main.java:36)
```

Figure 2: Output of main method after tested cases

PART 2

1 Problem Solution Approach

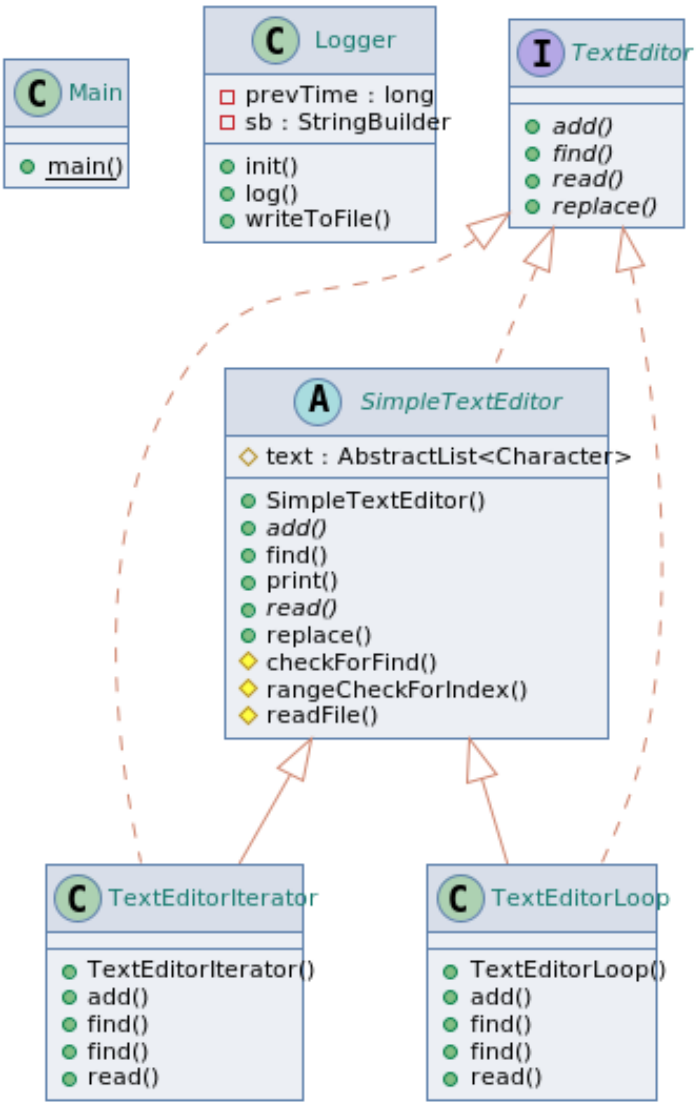
The assignment is developing an simple text editor. We supposed to do it by using different lists and different iteration methods. Therefore I decided to separate these methods into different classes. I started by coding and interface since we know basics of text editor. Method calls for lists should be same, so I separated classes by their iteration type. I have preferred to use an `AbstractList<E>` and use desired list by calling as a constructor parameter.

While I coding classes, I have realized that I needed some common methods for different classes such as; `readFile(), rangeCheck()` methods. Therefore I have created an abstract class on top of these classes, and put common methods in this abstract class. At the end, I have 2 main classes which are `TextEditorIterator` and `TextEditorLoop` according to their way of iteration. I didn't implement replace method 2 times to avoid code repetition. Replace method already calls appropriate find method according to type of the iteration.

In `TextEditorIterator` class I mostly used enhanced for loop, since it works with iterators.

For testing methods, I used a lorem ipsum file. It consist of 100 paragraphs which is relatively long text so that, I can compare the runtimes of methods more accurately. I thought comparing runtimes as milliseconds would be enough. For measuring runtime and logging, I created a `Loger` class.

2 Class Diagrams



3 Test Cases

To test all the methods I called methods with same parameters for different types of lists and different types of iteration. Between these calls I called `log()` method to measure runtime of methods.

I have created some text cases for operations. They are present on my main method.

Test Case ID	Test Scenario	Test Steps	Expected Results	Actual Results	Pass/Fail
T01	find	find("libero(f)")	32611	As expected	Pass
T02	add text at the beginning of the file	add(0,"deneme")	return 1, adds begining of the list	As expected	Pass
T03	add text in the middle of the list	add(32611, ****")	adds position 32611, return 1	As expected	Pass
T04	replace	replace('o','_')	2281 repalce chars	As expected	Pass
T05	replace nonexistent char	replace('8','a')	0	As expected	Pass
T06	find nonexistent char	find("asdf")	-1	As expected	Pass
T07	calling methods greater or less than size	add(-1,"x") add(999999, "x")	Throw exception	As expected	Pass

```
System.out.println("Test Cases");
iterAL.add(0, "deneme");
loopAL.add(0, "deneme");
System.out.println(iterAL.replace('8', 'a'));
System.out.println(iterAL.find("asdf"));
System.out.println(loopAL.replace('8', 'a'));
System.out.println(loopAL.find("asdf"));
iterAL.add(-1, "x");
iterAL.add(999999, "x");
```

```
String found at: 32611
2281 replaced chars.
String added specified position.
**ArrayList and iterator is used**
String found at: 32611
2281 replaced chars.
String added specified position.
**LinkedList and loop is used**
String found at: 32611
2281 replaced chars.
String added specified position.
**ArrayList and loop is used**
String found at: 32611
2281 replaced chars.
String added specified position.
```

4 Running and Results

4.1 Theoretic Analysis

ArrayList - Iterator

- `read`: $\theta(n)$
- `add`: $\theta(1)$, as long as there is enough space in the array. $\theta(n)$ otherwise.
- `find`: $O(n)$
- `replace`: $O(n)$

ArrayList - Loop

- `read`: $\theta(n)$
- `add`: $\theta(1)$, as long as there is enough space in the array. $\theta(n)$ otherwise.
- `find`: $O(n)$
- `replace`: $O(n)$

LinkedList - Iterator

- `read`: $\theta(n)$
- `add`: $O(n)$
- `find`: $O(n^2)$
- `replace`: $O(n^2)$

LinkedList - Loop

- `read`: $\theta(n)$
- `add`: $O(n)$
- `find`: $O(n^2)$
- `replace`: $O(n^2)$

4.2 Analysis of runtimes

`read` method is almost same, it has the same complexity for all of them.

`add` method for `ArrayList` is constant time, it is '$< 1ms$' both with iterator and loop. Since `get` methods works in constant time. In `LinkedList` although it has the complexity of $O(n)$ still pretty fast to iterate through nodes. Therefore the differnce between runtimes is not observable.

`find, replace`: Main reason of getting such a long runtimes for `LinkedList` is initalizing `ListIterator` with an index. Every time second iterator is initalized in find method, it has to iterate to specific position from beginning.

Since we don't use remove and add extensively, we don't make use of benefits of linked list.

```
Running Time (ms)     Message
           19 ms:     read method, LinkedList with Iterator
          812 ms:     find method, LinkedList with Iterator
         1476 ms:     replace method, LinkedList with Iterator
            0 ms:     add method, LinkedList with Iterator
            5 ms:     read method, ArrayList with Iterator
            6 ms:     find method, ArrayList with Iterator
            7 ms:     replace method, ArrayList with Iterator
            0 ms:     add method, ArrayList with Iterator
            6 ms:     read method, LinkedList with loop
          732 ms:     find method, LinkedList with loop
         1465 ms:     replace method, LinkedList with loop
            0 ms:     add method, LinkedList with loop
            3 ms:     read method, ArrayList with loop
            4 ms:     find method, ArrayList with loop
            2 ms:     replace method, ArrayList with loop
            0 ms:     add method, ArrayList with loop
```

Listing 2: Log file

PART 3

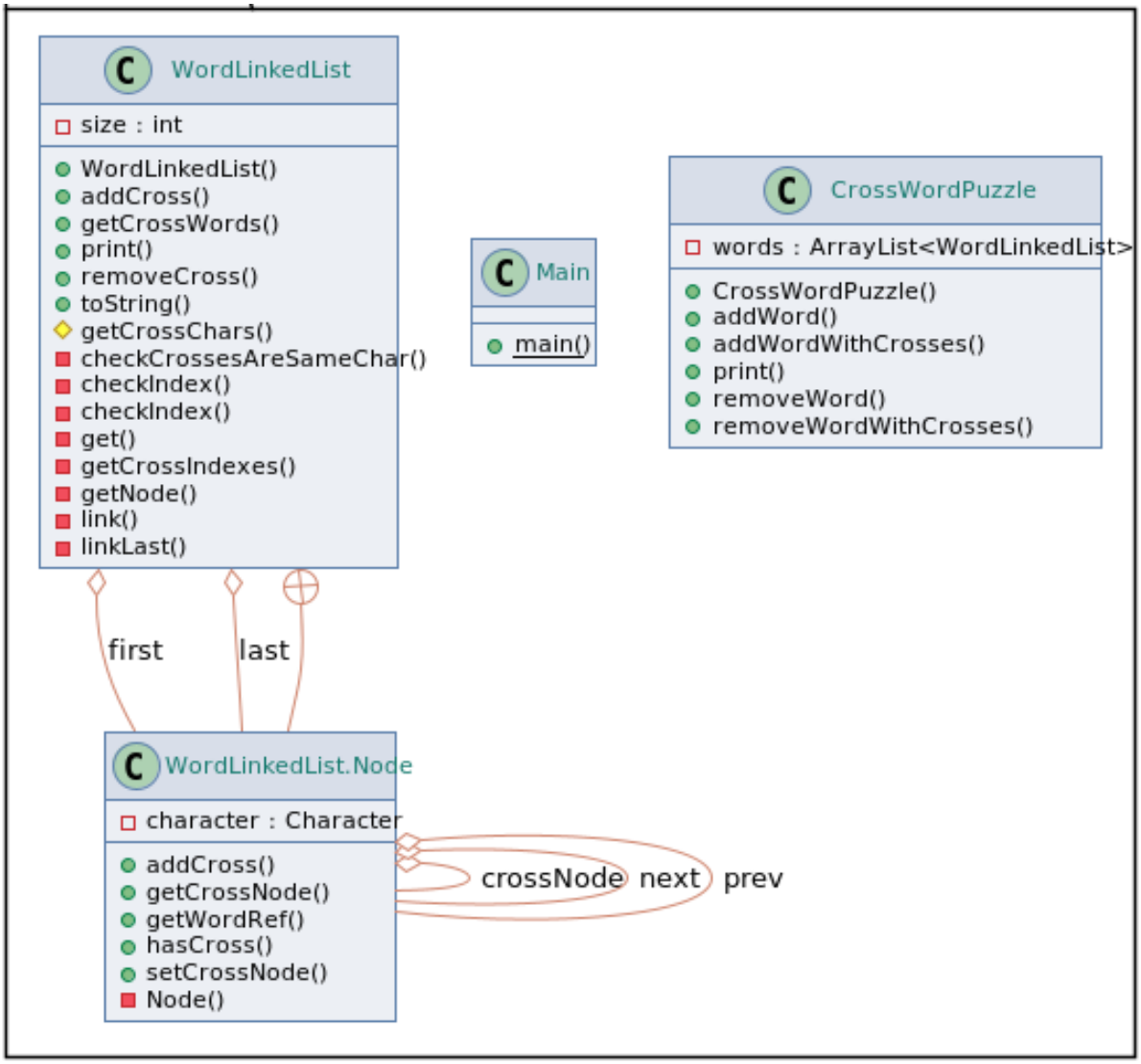
1 Problem Solution Approach

I started by creating a `Node` class which stores a character, both next and previous references to other character and cross node if there is any. While implementing `public` methods I added error handling methods. These methods are `checkCrossesAreSameChar(..)` which checks while adding a cross word if the characters are the same and `checkIndex(..)` which checks if indexes are in bounds. Both of these methods throws exception in necessary conditions.

In `CrossWordPuzzle` there are two adding and two removing operation. The difference between them is; while one of them is adding with all of its cross words other is only adds that word.

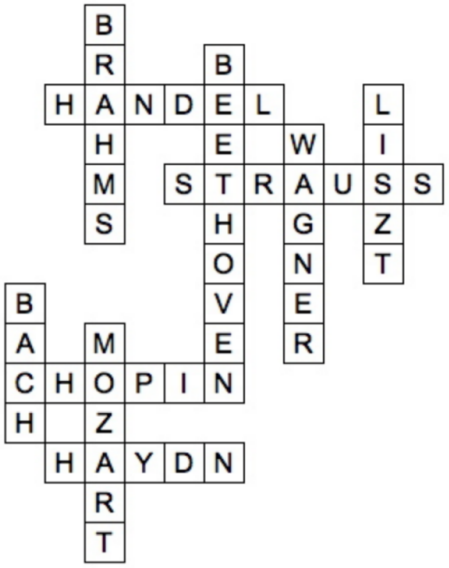
To test methods, I build the same puzzle in the assignment, added words with its cross words. Then I tested with another custom puzzle.

2 Class Diagrams



3 Test Cases, Running and Results

Along with the demonstration in the assignment custom test case I used;




```
**ADD TEST**
PUZZLES: { 1:FUN 6:CROSSWORD }
FUN: { 1:PUZZLES }
CROSSWORD: { 1:ARE 4:PUZZLES }
ARE: { 1:CROSSWORD }

**REMOVE TEST**
FUN removed
PUZZLES: { 6:CROSSWORD }
CROSSWORD: { 1:ARE 4:PUZZLES }
ARE: { 1:CROSSWORD }

**SECOND PUZZLE**
BRAHMS: { 2:HANDEL }
HANDEL: { 1:BRAHMS 4:BEETHOVEN }
BEETHOVEN: { 1:HANDEL 3:STRAUSS 8:CHOPIN }
STRAUSS: { 1:BEETHOVEN 3:WAGNER 5:LISZT }
CHOPIN: { 0:BACH 2:MOZART 5:BEETHOVEN }
WAGNER: { 1:STRAUSS }
LISZT: { 2:STRAUSS }
BACH: { 2:CHOPIN }
MOZART: { 1:CHOPIN 3:HAYDN }
HAYDN: { 1:MOZART }
```

Listing 3: Main output