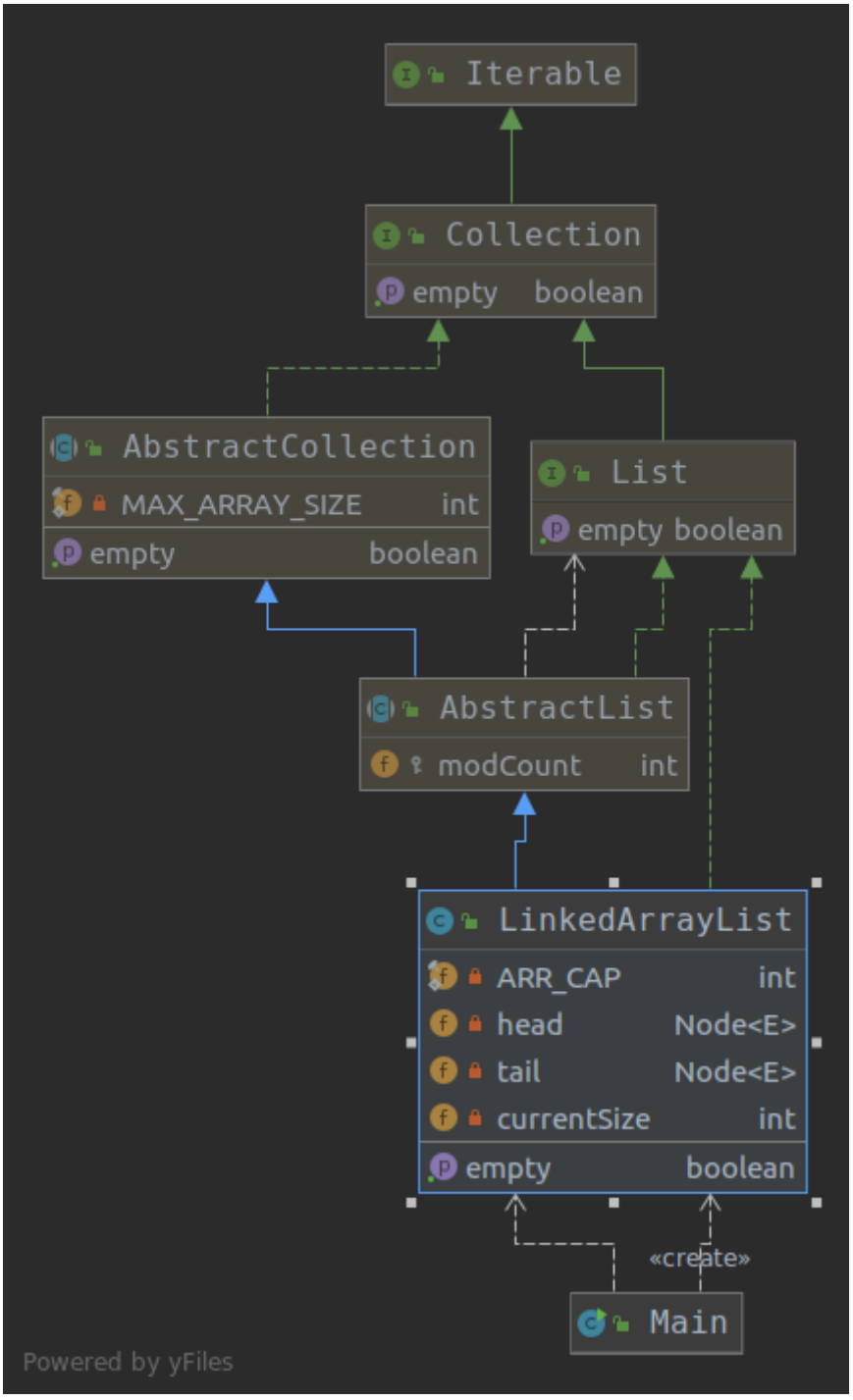
GIT Department of Computer Engineering
CSE 222/505 - Spring 2020
Homework 3 Report

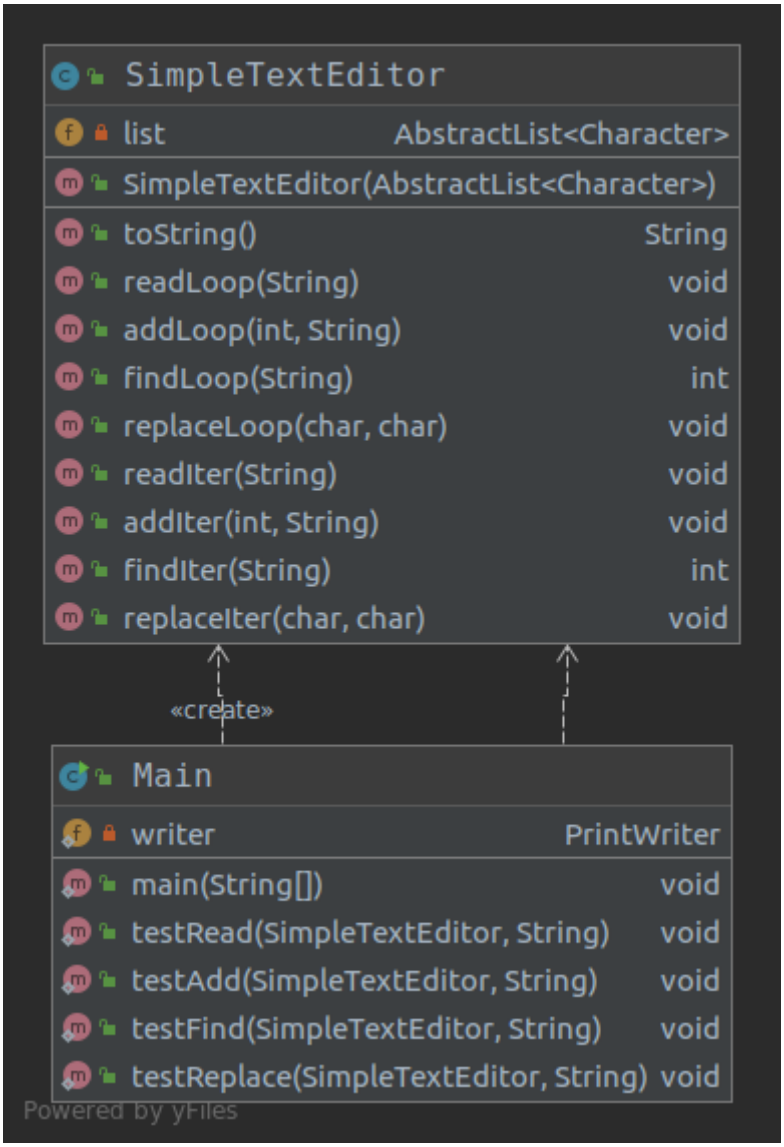
Buğra Eren Yılmaz
1801042669

1. Class Diagrams

Q1.



Q2.


SimpleTextEditor
list AbstractList<Character>
SimpleTextEditor(AbstractList<Character>)
toString() String
readLoop(String) void
addLoop(int, String) void
findLoop(String) int
replaceLoop(char, char) void
readIter(String) void
addIter(int, String) void
findIter(String) int
replaceIter(char, char) void
«create»
Main
writer PrintWriter
main(String[]) void
testRead(SimpleTextEditor, String) void
testAdd(SimpleTextEditor, String) void
testFind(SimpleTextEditor, String) void
testReplace(SimpleTextEditor, String) void
Powered by yFiles

2. Problem Solution Approach

Q1 Problem Solution.

For the solution of these problem I clearly needed a way to make custom implementation of the List class from java collection hierarchy.

First I started with creating a inner class for the custom implementation of Iterator, LinkedArrayListIterator.

Then I created the LinkedArrayListListIterator inner class, which is custom implementation of ListIterator.
I extended the LinkedArrayListIterator for this class, and added both-ways-traversing ability to this iterator.

Now that I have basic abstractions of the list traversing for my custom list class, I started building the Node structure.
Node is a inner class that holds references to next and previous nodes. It also holds the data as an Array.

Then I started implementing the list interface methods for extending the class from AbstractList.
Implemented all of the methods needed for list with custom data structure.

Q2 Problem Solution.

For the solution of these problem I needed a way to make a text container class. This class should be able to use any custom containers for holding data like LinkedLists or ArrayLists or any kind of AbstractList class.

To achieve this, I used polymorphism to get the List from user. Hold the list as AbstractList and got the list from Constructor by dependency injection.

For implementing methods with both using loops and iterators, I started looking into ListIterator documents and used it for adding, removing, traversing type of operations for iterator versions of methods.

Implementing the loop version of methods uses normal add remove and traverse operation from list class.

To measure the running time I used System class from java to get the current time in nano time. Then measured the elapsed time from start and end times.

For constructing the log file I used PrintWriter class from java.io and write all of the logs to a final log file named “logfile”.

3. Test Cases

Q1 tests.

```
public static void testAdd() {
    print("Test: Add object to list");
    LinkedArrayList<Integer> myList = new LinkedArrayList<>();

    for(int i = 1; i < 30; i++) {
        myList.add(i);
    }

    System.out.println(myList);
    print("End of test\n\n");
}
```

```
public static void testRemoveByIndex() {
    print("Test: Remove object from list by index");
    LinkedArrayList<Integer> myList = new LinkedArrayList<>();

    for(int i = 1; i < 30; i++) {
        myList.add(i);
    }

    print("Initial list:");
    print(myList.toString());

    for(int i = 11; i <= 20; i++) {
        myList.remove(i);
    }
    print("Removed elements from index 11 to 20:");
    print(myList.toString());

    print("End of test\n\n");
}
```

```java
public static void testRemoveByObject() {
    print("Test: Remove object from list");
    LinkedArrayList<Integer> myList = new LinkedArrayList<>();

    for(int i = 1; i < 30; i++) {
        myList.add(i);
    }

    print("Initial list:");
    print(myList.toString());

    myList.remove(Integer.valueOf(11));
    myList.remove(Integer.valueOf(12));
    myList.remove(Integer.valueOf(16));
    myList.remove(Integer.valueOf(5));

    print("Removed elements 11-12-16-5:");
    print(myList.toString());

    print("End of test\n\n");
}
```

```java
public static void testIteratorPrintForwards() {
    print("Test: Print list forwards with iterator");
    LinkedArrayList<Integer> myList = new LinkedArrayList<>();

    for(int i = 1; i < 30; i++) {
        myList.add(i);
    }

    Iterator<Integer> iteratorFoo = myList.iterator();
    while (true) {
        if(!iteratorFoo.hasNext()){
            System.out.print(iteratorFoo.next().toString() + " ");
            break;
        }
        System.out.print(iteratorFoo.next().toString() + " ");
    }
    System.out.println();
    print("End of test\n\n");
}
```

```java
public static void testIteratorPrintBackwards() {
    print("Test: Print list backwards from given index with iterator");
    LinkedArrayList<Integer> myList = new LinkedArrayList<>();

    for(int i = 1; i < 30; i++) {
        myList.add(i);
    }

    ListIterator<Integer> listIterator = myList.listIterator(11);
    while (true) {
        if(!listIterator.hasPrevious()){
            System.out.print(listIterator.previous().toString() + " ");
            break;
        }
        System.out.print(listIterator.previous().toString() + " ");
    }
    System.out.println();
    print("End of test\n\n");
}
```

```java
public static void testClearList() {
    print("Test: Clear list");
    LinkedArrayList<Integer> myList = new LinkedArrayList<>();

    for(int i = 1; i < 30; i++) {
        myList.add(i);
    }

    myList.clear();

    System.out.println(myList);
    print("End of test\n\n");
}
```

```java
public static void testRemoveNodeDemo() {
    print("Test: When an array in node is fully cleared, remove that node");
    LinkedArrayList<Integer> myList = new LinkedArrayList<>();

    for(int i = 1; i < 30; i++) {
        myList.add(i);
    }

    print("Initial list:");
    print(myList.toString());

    for(int i = 11; i ⩽ 20; i++) {
        myList.remove(Integer.valueOf(i));
    }

    print("Removed middle node by removing all elements inside the array of it");
    System.out.println(myList);
    print("End of test\n\n");
}
```

```java
public static void testAddToAnyIndex() {
    print("Test: When adding to an array that has empty space, just insert and shift");
    LinkedArrayList<Integer> myList = new LinkedArrayList<>();

    for(int i = 1; i < 30; i++) {
        myList.add(i);
    }

    myList.remove(Integer.valueOf(15));
    myList.remove(Integer.valueOf(16));
    print("Initial list:");
    print(myList.toString());

    myList.add(15, 999);

    print("Insert 999 to index 15, it will shift the array forward because array contains empty space.");
    System.out.println(myList);
    print("End of test\n\n");
}
```

```java
public static void testAddToAnyIndexAndExhaustArray() {
    print("Test: When adding to a full array, new node is inserted");
    LinkedArrayList<Integer> myList = new LinkedArrayList<>();

    for(int i = 1; i < 30; i++) {
        myList.add(i);
    }

    print("Initial list:");
    print(myList.toString());

    myList.add(15, 999);

    print("Insert 999 to index 15, it will exhaust the array, shift it and insert new node.");
    System.out.println(myList);
    print("End of test\n\n");

}
```

Q2 tests.

Read from file tests

```
public static void testRead(SimpleTextEditor editor, String type) {
    writer.println("Test: read from file with " + type);

    writer.println("Reading with loop implementation");
    long start = System.nanoTime();
    editor.readLoop("filetoread");
    long end = System.nanoTime();
    long time = end - start;
    double timeInSecond = (double) time / 1000000;
    writer.println(editor.toString());
    writer.println("Elapsed time: " + timeInSecond);

    start = System.nanoTime();
    writer.println("Reading with iterator implementation");
    editor.readIter("filetoread");
    end = System.nanoTime();
    time = end - start;
    timeInSecond = (double) time / 1000000;
    writer.println(editor.toString());
    writer.println("Elapsed time: " + timeInSecond);
}
```

Insert string tests

```java
public static void testAdd(SimpleTextEditor editor, String type) {
    writer.println("Test: add string to index 5 with " + type);

    writer.println("Adding string with loop implementation");
    long start = System.nanoTime();
    editor.addLoop(5, "INSERTED");
    long end = System.nanoTime();
    long time = end - start;
    double timeInSecond = (double) time / 1000000;
    writer.println(editor.toString());
    writer.println("Elapsed time: " + timeInSecond);

    start = System.nanoTime();
    writer.println("Adding string with iterator implementation");
    editor.addIter(5, "INSERTED");
    end = System.nanoTime();
    time = end - start;
    timeInSecond = (double) time / 1000000;
    writer.println(editor.toString());
    writer.println("Elapsed time: " + timeInSecond);
}
```

Find string tests.

```
public static void testFind(SimpleTextEditor editor, String type) {
    writer.println("Test: find string INSERTED with " + type);

    writer.println("Finding string with loop implementation");
    long start = System.nanoTime();
    int index = editor.findLoop("INSERTED");
    index++;
    long end = System.nanoTime();
    long time = end - start;
    double timeInSecond = (double) time / 1000000;
    writer.println("Found at index: " + index);
    writer.println("Elapsed time: " + timeInSecond);

    start = System.nanoTime();
    writer.println("Finding string with iterator implementation");
    index = editor.findIter("INSERTED");
    end = System.nanoTime();
    time = end - start;
    timeInSecond = (double) time / 1000000;
    writer.println(editor.toString());
    writer.println("Found at index: " + index);
    writer.println("Elapsed time: " + timeInSecond);
}
```

Replace Character test.

```java
public static void testReplace(SimpleTextEditor editor, String type) {
    writer.println("Test: replace all character e with character * with " + type);

    writer.println("Replacing with loop implementation");
    long start = System.nanoTime();
    editor.replaceLoop('e', '*');
    long end = System.nanoTime();
    long time = end - start;
    double timeInSecond = (double) time / 1000000;
    writer.println("Elapsed time: " + timeInSecond);

    start = System.nanoTime();
    writer.println("Replacing with iterator implementation");
    editor.replaceIter('e', '*');
    end = System.nanoTime();
    time = end - start;
    timeInSecond = (double) time / 1000000;
    writer.println(editor.toString());
    writer.println("Elapsed time: " + timeInSecond);
}
```

4. Running results

Q1 Results.

Adding objects to list, 1 to 29.

```
Test: Add object to list
[1,2,3,4,5,6,7,8,9,10]->[11,12,13,14,15,16,17,18,19,20]->[21,22,23,24,25,26,27,28,29,*]->
End of test
```

Remove objects from list, 11-12-16-5.

```
Test: Remove object from list
Initial list:
[1,2,3,4,5,6,7,8,9,10]->[11,12,13,14,15,16,17,18,19,20]->[21,22,23,24,25,26,27,28,29,*]->
Removed elements 11-12-16-5:
[1,2,3,4,6,7,8,9,10,*]->[13,14,15,17,18,19,20,*,*,*]->[21,22,23,24,25,26,27,28,29,*]->
End of test
```

Remove indexes from list, index 11 to 20.

```
Test: Remove object from list by index
Initial list:
[1,2,3,4,5,6,7,8,9,10]->[11,12,13,14,15,16,17,18,19,20]->[21,22,23,24,25,26,27,28,29,*]->
Removed elements from index 11 to 20:
[1,2,3,4,5,6,7,8,9,10]->[11,13,15,17,19,*,*,*,*,*]->[22,23,24,25,26,27,28,29,*,*]->
End of test
```

Forward traversing with Iterator.

```
Test: Print list forwards with iterator
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
End of test
```

Backwards traversing with ListIterator.

```
Test: Print list backwards from given index with iterator
12 11 10 9 8 7 6 5 4 3 2 1
End of test
```

Remove elements that results to removing a whole node.

```
Test: When an array in node is fully cleared, remove that node
Initial list:
[1,2,3,4,5,6,7,8,9,10]->[11,12,13,14,15,16,17,18,19,20]->[21,22,23,24,25,26,27,28,29,*]->
Removed middle node by removing all elements inside the array of it
[1,2,3,4,5,6,7,8,9,10]->[21,22,23,24,25,26,27,28,29,*]->
End of test
```

Insert element and shift list.

```
Test: When adding to an array that has empty space, just insert and shift
Initial list:
[1,2,3,4,5,6,7,8,9,10]->[11,12,13,14,17,18,19,20,*,*]->[21,22,23,24,25,26,27,28,29,*]->
Insert 999 to index 15, it will shift the array forward because array contains empty space.
[1,2,3,4,5,6,7,8,9,10]->[11,12,13,14,17,999,18,19,20,*]->[21,22,23,24,25,26,27,28,29,*]->
End of test
```

Insert element and exhaust the array.

```
Test: When adding to a full array, new node is inserted
Initial list:
[1,2,3,4,5,6,7,8,9,10]->[11,12,13,14,15,16,17,18,19,20]->[21,22,23,24,25,26,27,28,29,*]->
Insert 999 to index 15, it will exhaust the array, shift it and insert new node.
[1,2,3,4,5,6,7,8,9,10]->[11,12,13,14,15,999,16,17,18,19]->[20,*,*,*,*,*,*,*,*,*]->[21,22,23,24,25,26,27,28,29,*]->
End of test
```

Q2 Results.

Elapsed times in milliseconds.

Read from file tests

```
Test: read from file with ArrayList
Reading with loop implementation
[a, b, c, d, e, e, e, e, b, u, g, r, a, f, e, g]
Elapsed time: 0.81627
Reading with iterator implementation
[a, b, c, d, e, e, e, e, b, u, g, r, a, f, e, g]
Elapsed time: 0.214352

Test: read from file with LinkedList
Reading with loop implementation
[a, b, c, d, e, e, e, e, b, u, g, r, a, f, e, g]
Elapsed time: 0.100679
Reading with iterator implementation
[a, b, c, d, e, e, e, e, b, u, g, r, a, f, e, g]
Elapsed time: 0.083977
```

Adding/Inserting String tests

```
Test: add string to index 5 with ArrayList
Adding string with loop implementation
[a, b, c, d, e, I, N, S, E, R, T, E, D, e, e, e, b, u, g, r, a, f, e, g]
Elapsed time: 0.015008
Adding string with iterator implementation
[a, b, c, d, e, I, N, S, E, R, T, E, D, I, N, S, E, R, T, E, D, e, e, e, b, u, g, r, a, f, e, g]
Elapsed time: 0.018064

Test: add string to index 5 with LinkedList
Adding string with loop implementation
[a, b, c, d, e, I, N, S, E, R, T, E, D, e, e, e, b, u, g, r, a, f, e, g]
Elapsed time: 0.013906
Adding string with iterator implementation
[a, b, c, d, e, I, N, S, E, R, T, E, D, I, N, S, E, R, T, E, D, e, e, e, b, u, g, r, a, f, e, g]
Elapsed time: 0.015329
```

Finding string tests.

```
Test: find string INSERTED with ArrayList
Finding string with loop implementation
Found at index: 6
Elapsed time: 19.082662
Finding string with iterator implementation
[a, b, c, d, e, I, N, S, E, R, T, E, D, I, N, S, E, R, T, E, D, e, e, e, b, u, g, r, a, f, e, g]
Found at index: 6
Elapsed time: 7.618248

Test: find string INSERTED with LinkedList
Finding string with loop implementation
Found at index: 6
Elapsed time: 0.835416
Finding string with iterator implementation
[a, b, c, d, e, I, N, S, E, R, T, E, D, I, N, S, E, R, T, E, D, e, e, e, b, u, g, r, a, f, e, g]
Found at index: 6
Elapsed time: 0.333565
```

Replacing all characters tests.

```
Test: replace all character e with character * with ArrayList
Replacing with loop implementation
Elapsed time: 0.048231
Replacing with iterator implementation
[a, b, c, d, *, I, N, S, E, R, T, E, D, I, N, S, E, R, T, E, D, *, *, *, b, u, g, r, a, f, *, g]
Elapsed time: 0.032822

Test: replace all character e with character * with LinkedList
Replacing with loop implementation
Elapsed time: 0.047439
Replacing with iterator implementation
[a, b, c, d, *, I, N, S, E, R, T, E, D, I, N, S, E, R, T, E, D, *, *, *, b, u, g, r, a, f, *, g]
Elapsed time: 0.02131
```

Experimental running time results for Q2.

	A	B	C
1		LinkedList	ArrayList
2	ReadIterator	0.0839ms	0.2143ms
3	ReadLoop	0.1006ms	0.8162ms
4	AddIterator	0.0139ms	0.0150ms
5	AddLoop	0.0153ms	0.0180ms
6	FindIterator	0.3335ms	7.618ms
7	FindLoop	0.8354ms	19.082ms
8	ReplaceIterator	0.0213ms	0.0328ms
9	ReplaceLoop	0.0474ms	0.0482ms

Analysis of the performance for Q2.

LinkedList and Iterator used.

- Read: O(n), it needs to take the char and go to ith place from head and add the char with iterators add function. It adds N times.

- Add: O(1), it needs to go to the ith place and add the char with iterators add function. I is a constant index.

- Find: O(1), it needs to get and check equality for each traversed item. Get is constant time.

- Replace: O(1), it needs to get and check equality for each traversed item, and set if the equality check is true. Get and set are both constant time.

LinkedList and loops used.

Because of loops, get operator will go to head and start traversing towards to tail every time.

- Read: O(n^2). it needs to take the char go back to head and start traversing to tail every time for adding a character.

- Add: O(mn). it needs to go to head, start traversing to position, add the char. And repeat these for strings length times.

- Find: O(n). It just needs to go to head and start traversing until found.

- Replace: O(mn), go to place, get and set, repeat

Arraylist and Loop used.
Because of random access inside an array is always constant time, array list is much better at loops than LinkedList, but for iterators it is nearly same.
- **Read:** O(n), reads the stuff and uses set operator to set, which is constant time.
- **Add:** O(nm), since set and get operator are constant time, we need n shifts for m length string insertion.
- **Find:** O(n), traverse the array until found.
- **Replace:** O(n), again set and get operator are constant time for arrays.

Arraylist and Iterator used.
Using the iterator is nearly same for arraylist if we used the iterator by starting it from a given index. Because this operation is constant time it is similar to random access.
- **Read:** O(n), reads the stuff and uses set operator to set, which is constant time.
- **Add:** O(n), insert at given index and shift the array forward this can be done in one go.
- **Find:** O(n), traverse the array until found.
- **Replace:** O(n), set and get for iterator is much less costly then linkedlist counter-part