

GIT Department of Computer Engineering
CSE 222/505 - Spring 2020
Homework 4 Report

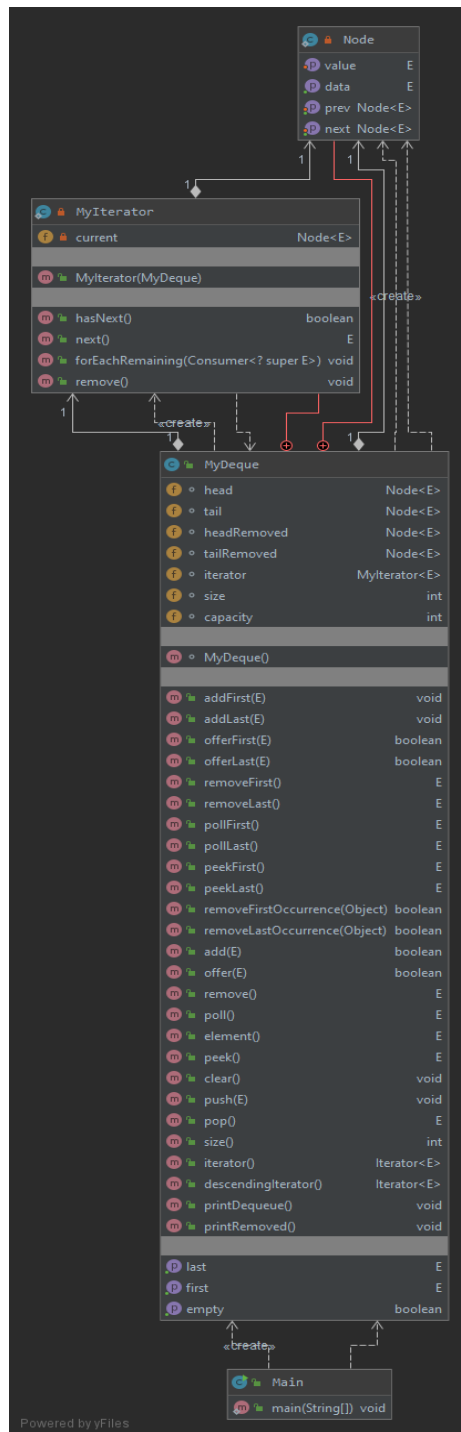
Esra Nur Arıcan
161044028

Question 1

The solution to the 1'st question and its explanation were prepared as a separate pdf. So no explanation needed in this report.

Question 2

1. CLASS DIAGRAMS



In this part of the homework we need to implement a generic deque class which extends from `AbstractCollection` and implements the `Deque` interface. This deque needs to keep two linked lists. One for to keep deque's elements and other for keep removed elements. This deque class `MyDeque` has two private inner class: `Node` class and `Iterator` class to use them in the deque implementation. `MyDeque` class implements all methods that come from `Deque` interface. The difference of this class from `Container Deque` class is, it keeps the removed elements in another linked list and it uses when a new node is needed. `MyIterator` class implements the methods from `Iterator` interface. Also main class written for testing the methods inside.

2. PROBLEM SOLUTIONS APPROACH

To implement `MyDeque` class, first `Node` class written. `Node` class keeps a generic type data and since i choose to use double linked list, it has `prev` and `next` variables in type `Node`.

`MyIterator` class written as inner class because it keeps `Node` and the `Node` class written inner also. So i implement this class as private inner class. And i implemented the methods of `Iterator<E>` interface.

`MyDeque` class keeps two linked lists. One is for elements in the deque and other is elements for removed from deque. To create the deque with linked list, 4 `Node` variable needed. `Head` and `tail` for deque's linked list and `headRemoved`, `tailRemoved` for removed variable's linked list. In the methods of `MyDeque`, there is different approach for adding or removing elements other than ordinary `Deque` class. The difference is after removing an element, we need to add that element to removed linked list and while adding an element, we need to check the removed linked list. If removed linked list has elements, one node should be transferred from removed linked list to deque's linked list. So in other words, when adding an element, we have to delete it from a place, and when deleting an element, we have to add an element to another place.

Since deque has only gives us the chance to achieve elements top or bottom, if a method needs to achieve middle places of deque, exception throwed.

In general, all methods are similar to each other. Adding and deleting elements is the basic process, but return value differs in some methods, or exceptions thrown shows differ in some methods.

3. TEST CASES

-`MyDeque` object created in type of `Integer`.

-`addFirst()` method called.

-`addLast()` method called.

-`offerFirst()` method called.

-`offerLast()` method called.

-`removeFirst()` method called.

-`removeLast()` method called.

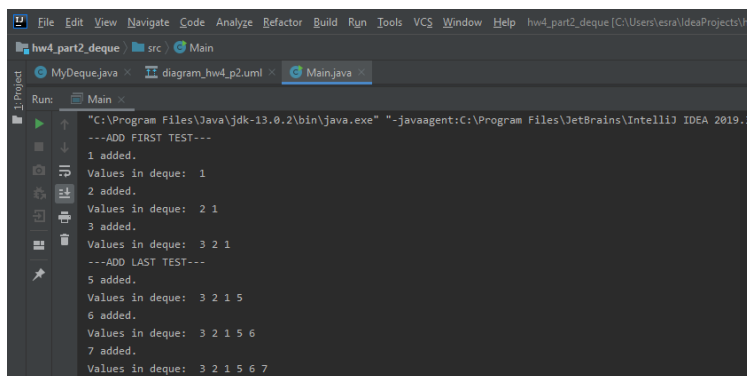
-`pollFirst()` method called.

-`pollLast()` method called.

- getFirst() method called.
- getLast() method called.
- peekFirst() method called.
- peekLast () method called.
- add () method called.
- offer () method called.
- remove() method called.
- poll () method called.
- element() method called.
- peek() method called.
- clear() method called. size() method called after this to show that deque is cleared.
- push() method called.
- pop() method called.
- isEmpty() method called.
- iterator() method called.
- After every method called, deque printed to show all additions, removings work correctly.

4. RUNNING AND RESULTS

addFirst and addLast results:

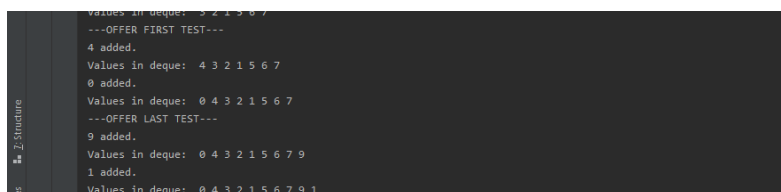


```

C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2019.3
---ADD FIRST TEST---
1 added.
Values in deque: 1
2 added.
Values in deque: 2 1
3 added.
Values in deque: 3 2 1
---ADD LAST TEST---
5 added.
Values in deque: 3 2 1 5
6 added.
Values in deque: 3 2 1 5 6
7 added.
Values in deque: 3 2 1 5 6 7

```

offerFirst and offerLast results:

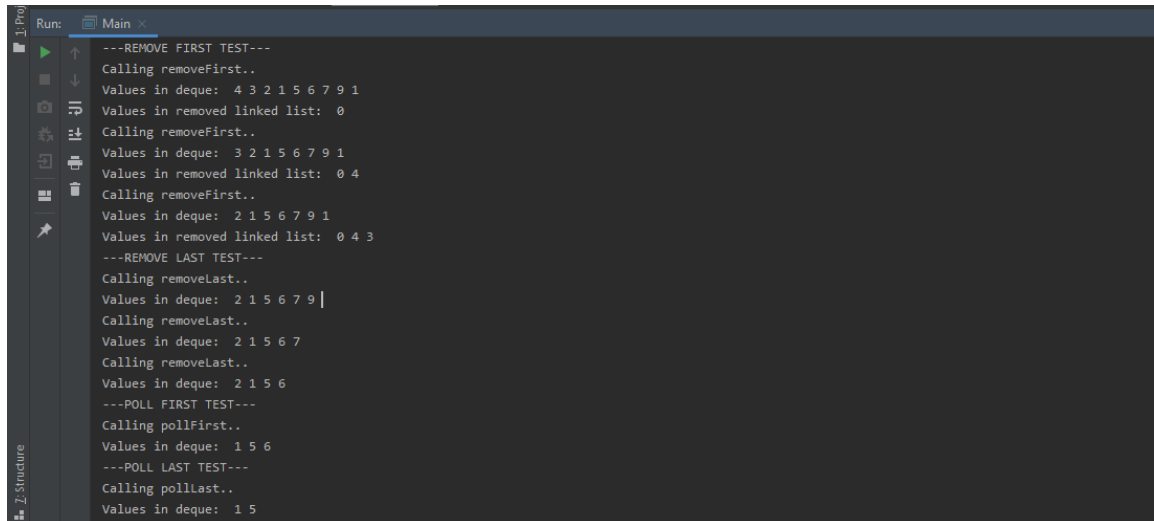


```

Values in deque: 3 2 1 5 6 7
---OFFER FIRST TEST---
4 added.
Values in deque: 4 3 2 1 5 6 7
0 added.
Values in deque: 0 4 3 2 1 5 6 7
---OFFER LAST TEST---
9 added.
Values in deque: 0 4 3 2 1 5 6 7 9
1 added.
Values in deque: 0 4 3 2 1 5 6 7 9 1

```

removeFirst and removeLast results:



```
Run: Main x
---REMOVE FIRST TEST---
Calling removeFirst..
Values in deque: 4 3 2 1 5 6 7 9 1
Values in removed linked list: 0
Calling removeFirst..
Values in deque: 3 2 1 5 6 7 9 1
Values in removed linked list: 0 4
Calling removeFirst..
Values in deque: 2 1 5 6 7 9 1
Values in removed linked list: 0 4 3
---REMOVE LAST TEST---
Calling removeLast..
Values in deque: 2 1 5 6 7 9 |
Calling removeLast..
Values in deque: 2 1 5 6 7
Calling removeLast..
Values in deque: 2 1 5 6
---POLL FIRST TEST---
Calling pollFirst..
Values in deque: 1 5 6
---POLL LAST TEST---
Calling pollLast..
Values in deque: 1 5
```


pollFirst and pollLast results:



```
Run: Main x
Values in deque: 2 1 5 6
---POLL FIRST TEST---
Calling pollFirst..
Values in deque: 1 5 6
---POLL LAST TEST---
Calling pollLast..
Values in deque: 1 5
```

getFirst, getLast , peekFirst and peekLast results:

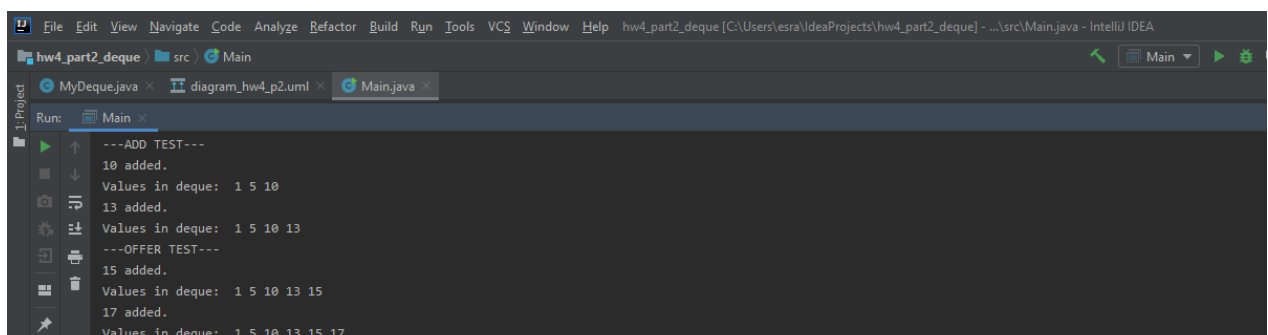
getFirst and peekFirst should show first element , getLast and peekLast should show last element in deque.



```
Run: Main x
Values in deque: 1 5
---GET FIRST TEST---
Calling getFirst..
1
---GET LAST TEST---
Calling getLast..
5
---PEEK FIRST TEST---
Calling peekFirst..
1
---PEEK LAST TEST---
Calling peekLast..
5
```

Add and offer methods results:

Add is equivalent to addFirst and offer is equivalent to addLast.



```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help hw4_part2_deque [C:\Users\esra\IdeaProjects\hw4_part2_deque] - ...src\Main.java - IntelliJ IDEA
hw4_part2_deque / src / Main
MyDeque.java x diagram_hw4_p2.uml x Main.java x
Run: Main x
---ADD TEST---
10 added.
Values in deque: 1 5 10
13 added.
Values in deque: 1 5 10 13
---OFFER TEST---
15 added.
Values in deque: 1 5 10 13 15
17 added.
Values in deque: 1 5 10 13 15 17
```

Size method results:

```
Values in deque: 1 5 10 13 15 17
---SIZE TEST---
Values in deque: 1 5 10 13 15 17
size called: 6
```

Remove, poll, element, peek methods results:

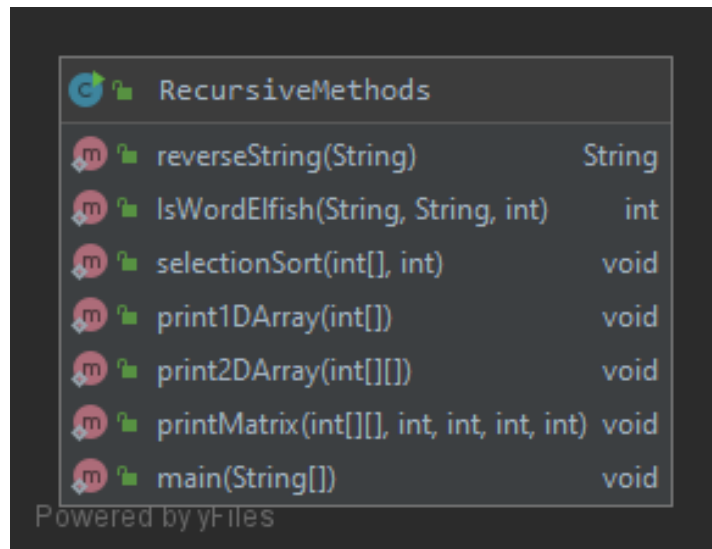
```
---REMOVE TEST---
Remove called.
Values in deque: 5 10 13 15 17
Remove called.
Values in deque: 10 13 15 17
---POLL TEST---
poll called.
Values in deque: 13 15 17
poll called.
Values in deque: 15 17
---ELEMENT TEST---
element called.
15
---PEEK TEST---
peek called.
15
```

Clear, push pop, isEmpty , iterator methods results:

```
---CLEAR TEST---
This deque is empty.
size called after clearing deque: 0
isEmpty called: true
---PUSH TEST---
3 pushed.
Values in deque: 3
6 pushed.
Values in deque: 6 3
5 pushed.
Values in deque: 5 6 3
---POP TEST---
pop called.
Values in deque: 6 3
---IS EMPTY TEST---
Values in deque: 6 3
isEmpty called: false
---ITERATOR TEST---
Reaching the deque's head with iterator.
6
Process finished with exit code 0
```

Question 3

1. CLASS DIAGRAMS



Class RecursiveMethods written to implement Q3's problems. This class has methods reverseString, isWordElfish, selectionSort, printMatrix. All of these methods are recursive. Also there are two print array methods written to print 1D and 2D arrays to the screen. Main method calls these recursive methods for testing.

2. PROBLEM SOLUTIONS APPROACH

2.1. Reversing a string

In this problem, we need to take a string input for reversing. If the string is empty, recursion stops. So this condition is the base case. Other way, incoming string splitted to words by controlling the space character. Splitted word assigned to result variable and remained string send as parameter to the function again (recursively). Thus, each time the final word is added to the result and the sentence is inverted at the end. If there is no word to divide at the end of the sentence, the function ends.

2.2. Determining if a word is elfish or not

To determine if a word is elfish, that word has to include the -e, -l, -f letters. So this recursive method takes the word as input and the string control letters ("elf" for this case). Every recursive call input controlled if it has the letter inside from control. The string to be checked is shrunk after checking a character and checked again. For example if the input has -e character inside, recursive call works for the "lf" string, and then "f" for the last time. And recursion stops after that control. If every letter occurs in the input (we understand that by using count, count must be 3 for elf check) we can say that the word is elfish.

2.3. Selection sort

In recursive selection sort general idea is finding minimum element and searching remained array. By this way we don't have to check sorted parts of the array. For example if we have an array for four elements, recursive selection sort works this way:

Find the minimum element in arr[0...4] and place it at beginning

Find the minimum element in arr[1...4] and place it at beginning of arr[1...4]

Find the minimum element in arr[2...4] and place it at beginning of arr[2...4]

Find the minimum element in arr[3...4] and place it at beginning of arr[3...4]

This way, our array will be sorted recursively.

In every recursive call start index of the array increases. If the start index has achieved to array's length recursion stops. This condition is the base case for recursive selection sort.

Small problems for this method is finding the minimum element.

2.4. Printing matrix in spiral form

To printing given matrix in spiral form, we have to print row, column and changing the index of row and column then print them again. So if the index of row bigger than row size or index of the column is bigger than the column size recursion must stop. These conditions are base case. First we print top row, then right column. And after that we should increase row index and decrease column index to be able to print the inner square. In recursive call, we call the method by updated index values.

3. TEST CASES

-Reversing string method called for two different strings.

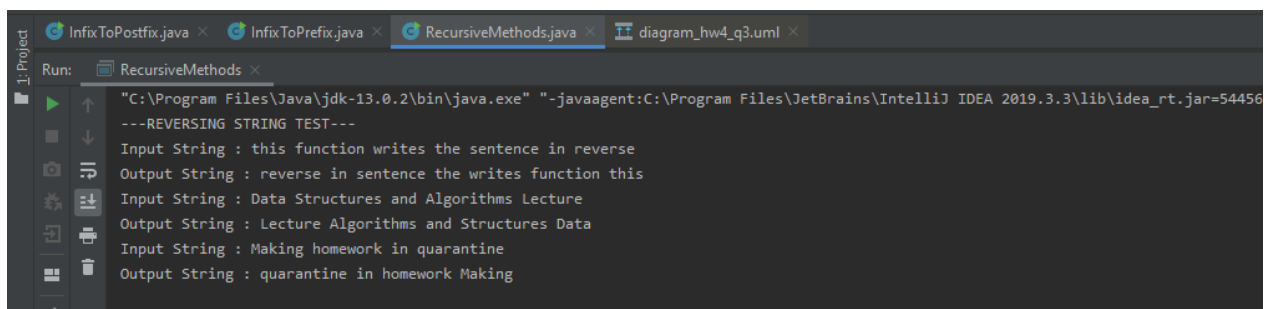
-Determine elfish word method called for three different words. Two of them were elfish and one wasn't elfish.

-Selection sort method called for two different arrays.

-Printing matrix method called for two times. One for 4*4 matrix and other was 5*4.

4. RUNNING AND RESULTS

Reverse sentence results:



```
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2019.3.3\lib\idea_rt.jar=54456"
---REVERSING STRING TEST---
Input String : this function writes the sentence in reverse
Output String : reverse in sentence the writes function this
Input String : Data Structures and Algorithms Lecture
Output String : Lecture Algorithms and Structures Data
Input String : Making homework in quarantine
Output String : quarantine in homework Making
```


Determine elfish word results:

```
output string: quarantine in homework taking  
---CHECKING ELFISH WORD TEST---  
waffle is elfish.  
unfriendly is elfish.  
another is not elfish!
```

Selection sort results:

```
---SELECTION SORT TEST---  
Printing array1 before selection sort:  
3 1 5 -2 7 0  
Printing array1 after selection sort:  
-2 0 1 3 5 7  
Printing array2 before selection sort:  
5 1 11 15 3 9 20 33 12 21 2  
Printing array2 after selection sort:  
1 2 3 5 9 11 12 15 20 21 33
```

Printing matrix results:

```
---PRINTING MATRIX SPIRAL FORM TEST---  
Testing with 4*4 matrix  
Printing matrix:  
1 4 7 10  
2 5 8 11  
3 6 9 10  
16 15 13 7  
  
Printing matrix spiral form:  
1 4 7 10 11 10 7 13 15 16 3 2 5 8 9 6  
  
Testing with 5*4 matrix  
Printing matrix:  
1 2 3 4  
5 6 7 8  
9 10 11 12  
13 14 15 16  
17 18 19 20  
  
Printing matrix spiral form:  
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10  
Process finished with exit code 0
```