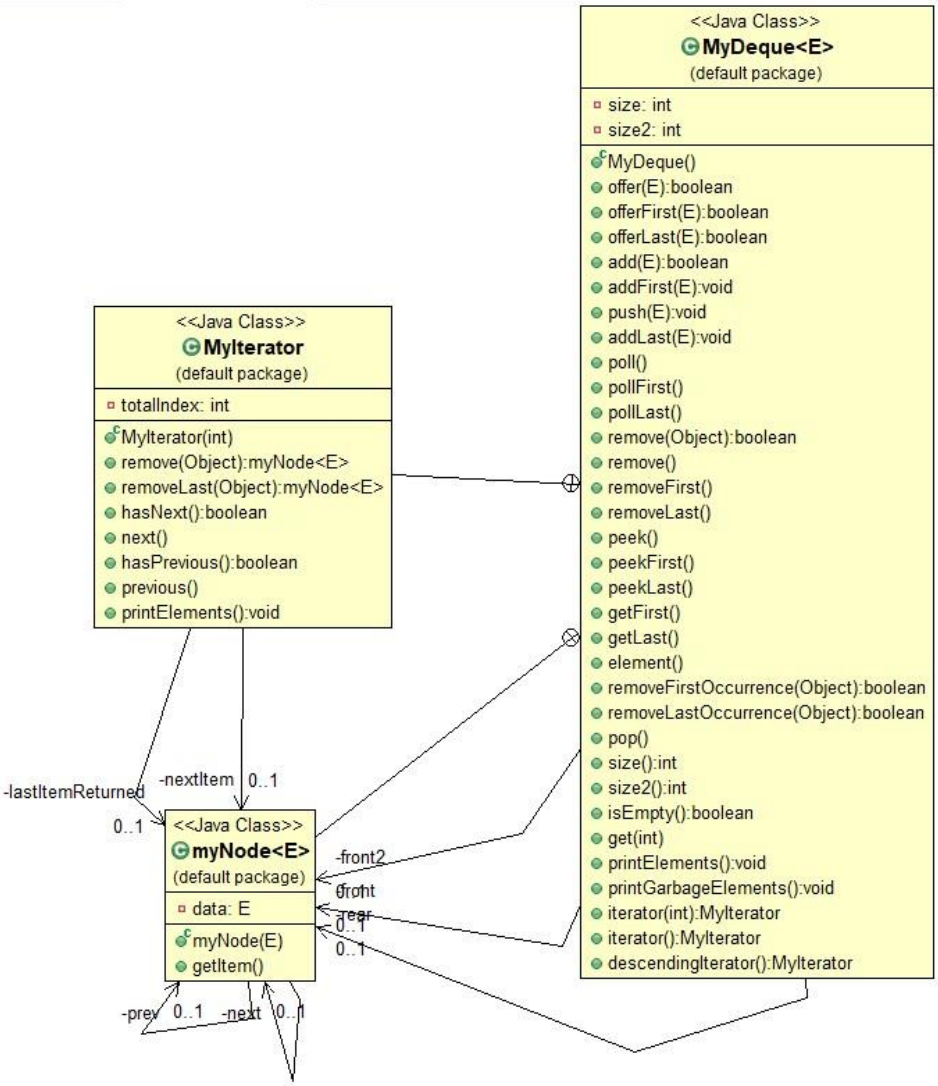
GIT Department of Computer Engineering
CSE 222/505 - Spring 2020
Homework #4 Report

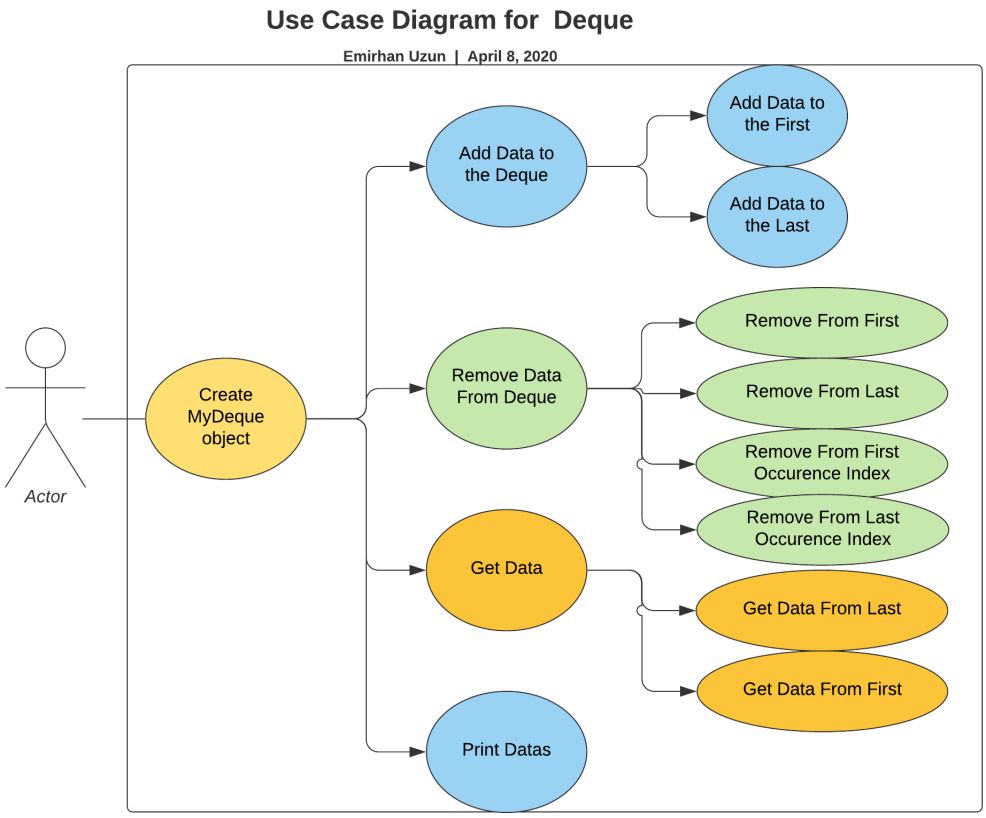
Emirhan Uzun
171044019

HOMEWORK 4 QUESTION 2 REPORT

1) Class Diagram



2) Use Case Diagram



3) Problem Solution Approach

Firstly, I identified the problem. A iterator and a node class are needed to me for this problem. I created doubly linked list for this problem. So in my node class, I have next and prev nodes.

After implementing them, I created a front and a rear node to hold the deque elements. Then I created my second front node to put the removing elements. I kept in size to these two lists.

Then, I put the nodes which removed in my remove methods into my garbage list. Then in my adding methods; if this garbage list is empty, I created a new node. If not, I used these garbage list nodes and I saved time.

I also wrote my own methods fort his problem instead of Java's List methods.Finally, I came to the conclusion and I will show the test results below.

4) Test Cases

- 1.Test exceptions.
- 2.Test add methods (add, addFirst, addLast)
- 2.1. Test offer method (offerFirst, offerLast, offer)
- 3.Test remove methods(remove, removeFirst, removeLast)
- 3.2.Test poll methods (poll, pollFirst, pollLast)
- 4.Test while we are adding element, that uses garbage list nodes
- 5.Test get methods (getFirst, getLast)
- 5.2.Test peek methods (peekFirst, peekLast, peek)
- 5.3.Test get element methods (element)
- 6.4. Test mix add and remove methods
- 7.Test remove first and last occurrence methods

5) Running Command and Results

TEST 1 : TEST EXCEPTION TYPES

```
java.lang.NullPointerException: This data is null !
```

```
MyDeque<Integer> mylist = new MyDeque<Integer>();
try {
    mylist.addFirst(null);

}
catch(Exception e) {
    System.out.println(e);
}
```

First exception is that if the data is null, then send NullPointerException.

```
java.util.NoSuchElementException: This element 5 was not found !
```

```
MyDeque<Integer> mylist = new MyDeque<Integer>();
try {
    mylist.remove(5);

}
catch(Exception e) {
    System.out.println(e);
}
```

The second exception is that if the deque is empty, sen NoSuchElementException

```
java.util.NoSuchElementException: The deque is empty !
```

```
MyDeque<Integer> mylist = new MyDeque<Integer>();
try {
    mylist.remove();

}
catch(Exception e) {
    System.out.println(e);
}
```

The third exception is simple second one. The difference is this method removes last of deque, but if the deque is empty, then send NoSuchElementException

```
java.util.NoSuchElementException: The deque is empty !
```

```
MyDeque<Integer> mylist = new MyDeque<Integer>();
try {
    mylist.getFirst();
}
catch(Exception e) {
    System.out.println(e);
}
```

The fourth exceptions is get method exception example. If the deque is empty, then send NoSuchElementException

TEST 2 : TEST ADD METHODS

```
MyDeque<Integer> mylist = new MyDeque<Integer>();
System.out.println("**********ADD - OFFER TEST**********");
    mylist.add(5);
    mylist.add(867);
    mylist.offerFirst(4);
    mylist.offerLast(9384);
    mylist.offerFirst(756);
    mylist.offerLast(1847);
    mylist.addLast(3923);
    mylist.offer(8413);
    mylist.offerLast(777);
    mylist.offerFirst(8413);
    mylist.addFirst(1453);
    mylist.offer(3543);
    mylist.printElements();
System.out.println("*****************************************");
```

```
**********ADD - OFFER TEST**********
The element 5 is inserted to the rear of deque
The element 867 is inserted to the rear of deque
The element 4 is inserted to the front of deque
The element 9384 is inserted to the rear of deque
The element 756 is inserted to the front of deque
The element 1847 is inserted to the rear of deque
The element 3923 is inserted to the rear of deque
The element 8413 is inserted to the rear of deque
The element 777 is inserted to the rear of deque
The element 8413 is inserted to the front of deque
The element 1453 is inserted to the front of deque
The element 3543 is inserted to the rear of deque

This is the deque list elements :
1453 --  8413 --  756 --  4 --  5 --  867 --  9384 --  1847 --  3923 --  8413 --  777 --  3543 --
```

These methods add the element to the deque. Some methods add to the first index of deque and some methods add to the last index of deque.

TEST 3 : TEST REMOVE METHODS

```
System.out.println("***************REMOVE TEST***************");
    mylist.removeFirst();
    mylist.removeLast();
    mylist.remove(9384);
    mylist.pollFirst();
    mylist.pollLast();
    mylist.poll();
    System.out.println("This is the garbage list size : " + mylist.size2());
System.out.println("*****************************************");
```

```
***************REMOVE TEST***************
The first element 1453 is removed and is moved to the garbage list.

This is the garbage list elements :
1453 --

The last element 3543 is removed and is moved to the garbage list.

This is the garbage list elements :
3543 --  1453 --

9384 was removed from queue in index 5

This is the garbage list elements :
9384 --  3543 --  1453 --

The first element 8413 is removed and is moved to the garbage list.

This is the garbage list elements :
8413 --  9384 --  3543 --  1453 --

The last element 777 is removed and is moved to the garbage list.

This is the garbage list elements :
777 --  8413 --  9384 --  3543 --  1453 --

The last element 8413 is removed and is moved to the garbage list.

This is the garbage list elements :
8413 --  777 --  8413 --  9384 --  3543 --  1453 --

This is the garbage list size : 6
*****************************************
```

These methods remove element from the deque. Some methods remove from the first index of deque and some methods remove from the last index of deque. Also this remove methods put the nodes to the garbage list and then if the

user adds element, the program uses the garbage list nodes instead of creating new node.

TEST 4 : TEST USING GARBAGE LIST NODES DURING ADDING

```
System.out.println("****ADD - OFFER (CHECK IF USE GARBAGE LIST)****");
    mylist.add(2456);
    mylist.add(3620);
    mylist.offerFirst(4890);
    mylist.offerLast(3950);
    mylist.offerFirst(1359);
    System.out.println("\nThis is the garbage list size : " + mylist.size2());
    mylist.offerLast(1456);
    mylist.offer(7898);
    mylist.offerLast(6503);
    mylist.offerLast(4890);
    mylist.offerFirst(9384);
    mylist.addFirst(2253);
    mylist.printElements();
System.out.println("*****************************************");
```

```
****ADD - OFFER (CHECK IF USE GARBAGE LIST)****
The node was taken from garbage list
The element 2456 is inserted to the rear of deque
The node was taken from garbage list
The element 3620 is inserted to the rear of deque
The node was taken from garbage list
The element 4890 is inserted to the front of deque
The node was taken from garbage list
The element 3950 is inserted to the rear of deque
The node was taken from garbage list
The element 1359 is inserted to the front of deque

This is the garbage list size : 1
The node was taken from garbage list
The element 1456 is inserted to the rear of deque
The element 7898 is inserted to the rear of deque
The element 6503 is inserted to the rear of deque
The element 4890 is inserted to the rear of deque
The element 9384 is inserted to the front of deque
The element 2253 is inserted to the front of deque

This is the deque list elements :
2253 --  9384 --  1359 --  4890 --  756 --  4 --  5 --  867 --  1847 --  3923 --  2456 --  3620 --  3950 --  1456 --  7898 --  6503 --  4890 --

*****************************************
```

After we remove some elements, we add some and the program uses garbage list instead of creating new node. So it works correctly.

TEST 5 : TEST GET METHODS

```
System.out.println("****************GET AND PEEK TEST***************");
    mylist.printElements();
    System.out.println("The first element is (getFirst method) " + mylist.getFirst());
    System.out.println("The last element is (getLast method) " + mylist.getLast());
    System.out.println("The first element is (peekFirst method " + mylist.peekFirst());
    System.out.println("The last element is (peekLast method) " + mylist.peekLast());
    System.out.println("The last element is (peek method ) " + mylist.peek());
    System.out.println("The last element is (element method ) " + mylist.element());
    System.out.println("The 5.index element is (get method ) " + mylist.get(5));
System.out.println("*****************************************");
```

```
***************GET AND PEEK TEST***************

This is the deque list elements :
2253 --  9384 --  1359 --  4890 --  756 --  4 --  5 --  867 --  1847 --  3923 --  2456 --  3620 --  3950 --  1456 --  7898 --  6503 --  4890

The first element is (getFirst method) 2253
The last element is (getLast method) 4890
The first element is (peekFirst method 2253
The last element is (peekLast method) 4890
The last element is (peek method ) 4890
The last element is (element method ) 2253
The 5.index element is (get method ) 4
*****************************************
```

These methods returns the elements of deque. All of them works correctly.

TEST 6 : TEST MIX ADD AND REMOVE METHODS

```
System.out.println("****MIX ADD AND REMOVE TEST****");
    mylist.offerFirst(777);
    mylist.addFirst(458);
    mylist.addLast(756);
    mylist.addLast(4);
    mylist.addLast(1359);
    mylist.removeLast();
    mylist.removeLast();
    mylist.offerLast(1359);
    mylist.removeFirst();
    mylist.printElements();
System.out.println("*********************************************");
```

```
****MIX ADD AND REMOVE TEST****
The element 777 is inserted to the front of deque
The element 458 is inserted to the front of deque
The element 756 is inserted to the rear of deque
The element 4 is inserted to the rear of deque
The element 1359 is inserted to the rear of deque
The last element 1359 is removed and is moved to the garbage list.

This is the garbage list elements :
1359 --

The last element 4 is removed and is moved to the garbage list.

This is the garbage list elements :
4 --  1359 --

The node was taken from garbage list
The element 1359 is inserted to the rear of deque
The first element 458 is removed and is moved to the garbage list.

This is the garbage list elements :
458 --  1359 --


This is the deque list elements :
777 --  2253 --  9384 --  1359 --  4890 --  756 --  4 --  5 --  867 --  1847 --  3923 --  2456 --  3620 --  3950 --  1456 --  7898 --  6503 --  4890 --  756 --  1359

*********************************************
```

To sure the use garbage list, I write that add and remove methods again

TEST 7 : TEST REMOVE FIRST AND LAST OCCURRENCE METHODS

```
System.out.println("****************REMOVE OCCURENCE TEST***************");
    mylist.printElements();
    mylist.removeFirstOccurrence(756);
    mylist.printElements();
    mylist.removeLastOccurrence(4890);
    mylist.printElements();
System.out.println("*****************************************************");
```

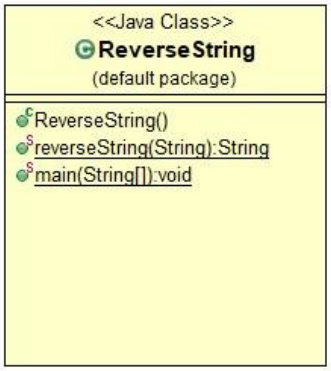
```
***************REMOVE OCCURENCE TEST***************

This is the deque list elements :
777 --  2253 --  9384 --  1359 --  4890 --  756 --  4 --  5 --  867 --  1847 --  3923 --  2456 --  3620 --  3950 --  1456 --  7898 --  6503 --  4890 --  756 --  1359

756 was removed from queue in index 5

This is the garbage list elements :
756 --  458 --  1359 --


This is the deque list elements :
777 --  2253 --  9384 --  1359 --  4890 --  4 --  5 --  867 --  1847 --  3923 --  2456 --  3620 --  3950 --  1456 --  7898 --  6503 --  4890 --  756 --  1359 --

4890 was removed from queue in index 16

This is the garbage list elements :
4890 --  756 --  458 --  1359 --


This is the deque list elements :
777 --  2253 --  9384 --  1359 --  4890 --  4 --  5 --  867 --  1847 --  3923 --  2456 --  3620 --  3950 --  1456 --  7898 --  6503 --  756 --  1359 --

*****************************************************
```

These methods removes the first or last occurence elements. I use and it works correctly.

- **COMMAND –**
  In this homework, I learned the deque struct and combine with iterator and node. Also I learned some methods differences. For example add and offer method. And finally, using that garbage list, we save the time and it is very important in this job. Because we have to minimize the run time of programs.

HOMEWORK 4 QUESTION 3.1 REPORT (Reverse String)

1)Class Diagram



2)Problem Solution Approach

In this part, first I thought about problem. So I decided to split sentence to the words not letters.

My base case is if the last string is equal to null or doesn't contain the white space to stop recursion.

My small problems is that find the index of White space in given string, then seperate from the beginning of the string to this index and call the recursion method with rest of string + first seperated Word.

To summarize, string comes to the method. The method checks the string and then split the words until doesn't contain white space or equal to null (These are my base case) then send rest of string to the recursion method.

3)Test Cases

Test 1 : Test example sentence

Test 2 : Test given sentence in homework

4)Running Command and Results

```java
public class ReverseString {

    /**
     * This method, returns the reversing string. This uses recursion.
     * @param str Given string to reverse
     * @return Returns the reversed string
     */
    public static String reverseString(String str) {
        if (str == null)
            return "";
        if (!str.contains(" "))
            return str;
        //Find the index of white space. Then split the first word from beginning to index.
        int spaceIndex = str.indexOf(" ");
        String firstWord = str.substring(0, spaceIndex);
        //Calls the recursive method with first word. At last, return this string.
        return reverseString(str.substring(spaceIndex + 1)) + " " + firstWord;
    }

    public static void main(String[] args) {
        String str = "this function writes the sentence in reverse";
        System.out.println(reverseString(str));
    }

}
```

This is my method to see.

TEST 1 : TEST EXAMPLE SENTENCE

```java
public static void main(String[] args) {
    String str = "Gebze Technical University Computer Engineering";

    System.out.println(reverseString(str));
}
```

```
Engineering Computer University Technical Gebze
```

Method works correctly in this example

TEST 2 : TEST THE GIVEN SENTENCE IN HOMEWORK

```java
public static void main(String[] args) {
    String str = "this function writes the sentence in reverse";
    System.out.println(reverseString(str));
}
```

```
reverse in sentence the writes function this
```

This method also works correctly in this test.

HOMEWORK 4 QUESTION 3.2 REPORT (Elfish Word Test)

1)Class Diagram

2)Problem Solution Approach

First I return integer in this part. Because if a word is an elfish, then it contains just 3 letters. So, at the end I return an integer

My base case is checks elfish letters length. Because in every call this method, I send the elfish letters from 1.index to end. So it decrease one by one every call.

My small problem is if a word contains the elfish letters first character, then counter is 1. Else return 0. And if it is 1, then return the count and recursion method (without elfish letters first character. Because I split this.)

3)Test Cases

Test 1 : Test one of given word in homework pdf

Test 2 : Test another given word in homework pdf

Test 3 : Test a word which is not elfish word.

4)Running Command and Results

This is my method :

```
public static int ElfishTest(String word, String elfishLetters, int elfishLetterCount) {

    //Base case : If the elfish letters length is less than 1, then return.
    if(elfishLetters.length() < 1) {
        return 0;
    }

    //Checks the string contains the elfish letters.
    if(word.contains(elfishLetters.substring(0, 1))) {
        elfishLetterCount = 1;
    }
    else {
        return 0;
    }

    //Calls the recursion with counter.
    return elfishLetterCount + ElfishTest(word, elfishLetters.substring(1, elfishLetters.length()), 0);
}
```

TEST 1 : TEST ONE OF GIVEN WORD IN HOMEWORK

```
public static void main(String args[]) {

    String word = "whiteleaf";

    //String word = "tasteful";
    //String word = "unfriendly";
    //String word = "waffle";

    String elfishLetters = "elf";

    int elfishLetterCount = ElfishTest(word, elfishLetters, 0);

    if(elfishLetters.length()== elfishLetterCount) {
        System.out.println(word + " has  'e', 'l' and 'f' letters.\n"
                + "So this is elifsh!");
    }else{
        System.out.println(word + " doesn't have at least one of that 'e','l'and 'f' letters.\n"
                + " So this is not elifsh!");
    }
}
```

```
whiteleaf has  'e', 'l' and 'f' letters.
So this is elifsh!
```

TEST 2 : TEST ANOTHER GIVEN WORD IN HOMEWORK

```
public static void main(String args[]) {

    //String word = "whiteleaf";

    String word = "tasteful";
    //String word = "unfriendly";
    //String word = "waffle";

    String elfishLetters = "elf";

    int elfishLetterCount = ElfishTest(word, elfishLetters, 0);

    if(elfishLetters.length()== elfishLetterCount) {
        System.out.println(word + " has  'e', 'l' and 'f' letters.\n"
                + "So this is elifsh!");
    }else{
        System.out.println(word + " doesn't have at least one of that 'e','l'and 'f' letters.\n"
                + " So this is not elifsh!");
    }
}
```

```
tasteful has  'e', 'l' and 'f' letters.
So this is elifsh!
```

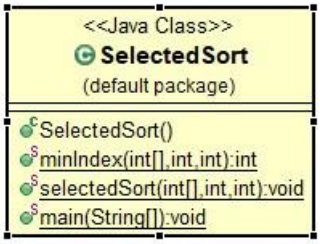
TEST 3 : TEST A WORD WHICH IS NOT ELFISH WORD

```
public static void main(String args[]) {

    String word = "Computer";
    //String word = "whiteleaf";

    //String word = "tasteful";
    //String word = "unfriendly";
    //String word = "waffle";

    String elfishLetters = "elf";

    int elfishLetterCount = ElfishTest(word, elfishLetters, 0);

    if(elfishLetters.length()== elfishLetterCount) {
        System.out.println(word + " has  'e', 'l' and 'f' letters.\n"
                + "So this is elifsh!");
    }else{
        System.out.println(word + " doesn't have at least one of that 'e','l'and 'f' letters.\n"
                + " So this is not elifsh!");
    }
}
```

```
Computer doesn't have at least one of that 'e','l'and 'f' letters.
 So this is not elifsh!
```

HOMEWORK 4 QUESTION 3.3 REPORT (Selected Sort)

1)Class Diagram



2)Problem Solution Approach

In this part, I use 2 recursive method. From main, I send the first recursive method with array, last and first index of array parameters.

In this method my base case is if start index is greater than finish index, then return. Because every call this method, I increase the start index one by one.

In this method, my algorithm is that first initialize the min index with call the other min index recursive method, and then if min index is not equal than start index, swap them. Last call again recursion method with increasing start index. Because we find the minimum number of array and put the start index.

The other min index recursion method finds the minimum index of array elements.

İn this method my base case is if start index is equal to the finish index, then return start.

My algorithm is I compare returning index's elements and current index's elements. Which one is smaller, then return this index every time.

3)Test Cases

Test 1 : Test example array

Test 2 : Test given array in homework

4)Running Command and Results

This is my methods :

```
public static int minIndex(int sortingList[], int start, int finish)
{
    if (start == finish)
        return start;

    // Find minimum of remaining elements
    int k = minIndex(sortingList, start + 1, finish);

    // Return minimum of current and remaining.
    if(sortingList[start] < sortingList[k] )    return start;
    else return k;
}
```

```
public static void selectedSort(int[] sortingList, int start, int finish) {
        if (start >= finish)    return;
        else{
            int k = minIndex(sortingList, start, finish);
            if (k != start){
               // swap
               int temp = sortingList[k];
               sortingList[k] = sortingList[start];
               sortingList[start] = temp;
            }


           // Sort the remaining list
           selectedSort(sortingList, start + 1, finish);

        }
}
```

TEST 1 : TEST EXAMPLE ARRAY

```
public static void main(String [] args){
        //int[] sortingList = {9,5,3,7,4,8,6};

        int[] sortingList = {65,49,31,46,5,32,4,52,6};
        System.out.printf("This is our initially array : ");
        for (int i=0; i<sortingList.length; i++) {
                System.out.printf("%d ",sortingList[i]);
        }


        selectedSort(sortingList,0,sortingList.length -1);

        System.out.printf("\n\nThe sorting list is : ");
        for (int i=0; i<sortingList.length; i++) {
                System.out.printf("%d ",sortingList[i]);
        }
}
```

```
This is our initially array : 65 49 31 46 5 32 4 52 6

The sorting list is : 4 5 6 31 32 46 49 52 65
```

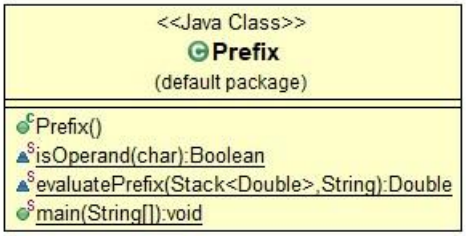
TEST 2 : TEST GIVEN ARRAY IN HOMEWORK

```
public static void main(String [] args){
        int[] sortingList = {9,5,3,7,4,8,6};


        System.out.printf("This is our initially array : ");
        for (int i=0; i<sortingList.length; i++) {
                System.out.printf("%d ",sortingList[i]);
        }


        selectedSort(sortingList,0,sortingList.length -1);

        System.out.printf("\n\nThe sorting list is : ");
        for (int i=0; i<sortingList.length; i++) {
                System.out.printf("%d ",sortingList[i]);
        }
}
```

```
This is our initially array : 9 5 3 7 4 8 6

The sorting list is : 3 4 5 6 7 8 9
```

HOMEWORK 4 QUESTION 3.4 REPORT (Prefix Expression)

1)Class Diagram



2)Problem Solution Approach

In this method, firstly I send the expression and empty stack to the method that is recursive. I have two method in this program and the other method is not recursive.

In my recursive method, my base case is if the expression length is less than 1, return null.

The expression size decreases one by one because I take the one character of them and call recursive function with rest of the expression.

Later, I check the characters from end to beginning. If it is operand (I check this in other method which is isOperand method) than I push to the stack and return that. But if it is operator, I pop two elements of that stack and process with incoming operator.

3)Test Cases

Test 1-2 : Test Different Expressions

4)Running Command and Results

This is my method :

```
public static Double evaluatePrefix(Stack<Double> Stack,String expression)
{
    //Base case is if the expression doesn't have more elements
    if (expression.length() < 1)
        return null ;

    //Recursion call
    evaluatePrefix(Stack,expression.substring(1));

    //Check the element that if is operand push to the stack.
    if (isOperand(expression.charAt(0)))
        Stack.push((double)(expression.charAt(0) - 48));
    else {

       //If the element is not operand than pop two element and process with incoming operator
        double o1 = Stack.peek();
        Stack.pop();
        double o2 = Stack.peek();
        Stack.pop();

        switch (expression.charAt(0)) {
        case '+':
            Stack.push(o1 + o2);
            break;
        case '-':
            Stack.push(o1 - o2);
            break;
        case '*':
            Stack.push(o1 * o2);
            break;
        case '/':
            Stack.push(o1 / o2);
            break;
        }
    }

    return Stack.peek();
}
```

TEST 1 : TEST WITH AN EXAMPLE

```
public static void main(String[] args)
{
    String exprsn = "*/93+77";
    Stack<Double> Stack = new Stack<Double>();
    System.out.println("The prefix value is : " + evaluatePrefix(Stack,exprsn));
}
```

```
The prefix value is : 42.0
```

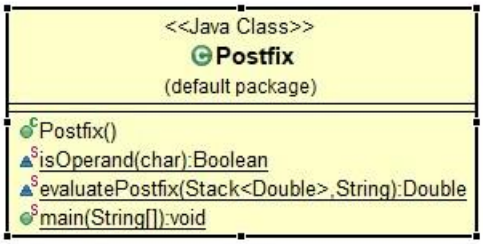
TEST 2 : TEST WITH ANOTHER EXAMPLE

```
public static void main(String[] args)
{
    String exprsn = "*/64+98";
    Stack<Double> Stack = new Stack<Double>();
    System.out.println("The prefix value is : " + evaluatePrefix(Stack,exprsn));
}
```

```
The prefix value is : 25.5
```

HOMEWORK 4 QUESTION 3.5 REPORT (Postfix Expression)

1)Class Diagram



2)Problem Solution Approach

In this method, firstly I send the expression and empty stack to the method that is recursive. I have two method in this program and the other method is not recursive.

In my recursive method, my base case is if the expression is empty, then return stack which is result.

The expression size decreases one by one because I take the one character of them and call recursive function with rest of the expression.

Later, I check the characters from end to beginning. If it is operand (I check this in other method which is isOperand method) than I push to the stack and call recursive method. But if it is operator, I pop two elements of that stack and process with incoming operator.

3)Test Cases

Test 1-2 : Test Different Expressions

4)Running Command and Results

This is my method :

```
public static Double evaluatePostfix(Stack<Double> Stack,String expression)
{
    //Base case is if the expression doesn't have more elements then returns the stack which is answer
    if (expression.isEmpty())
        return Stack.peek() ;

    //Check the element that if is operand push to the stack.
    if (isOperand(expression.charAt(0)))
        Stack.push((double)(expression.charAt(0) - 48)); //48 means in ASCII is 0. So we push stack as a numbe
    else {

         //If the element is not operand than pop two element and process with incoming operator
        double o1 = Stack.peek();
        Stack.pop();
        double o2 = Stack.peek();
        Stack.pop();

        //I would like to specify that I firstly wrote o2 and o1.
        //Because when we pop two numbers, we firstly put the last pop element which is o2
        switch (expression.charAt(0)) {
        case '+':
            Stack.push(o2 + o1);
            break;
        case '-':
            Stack.push(o2 - o1);
            break;
        case '*':
            Stack.push(o2 * o1);
            break;
        case '/':
            Stack.push(o2 / o1);
            break;
        }
    }

    //Call recursive method with first index of expression
    return evaluatePostfix(Stack,expression.substring(1));
}
```

TEST 1 : TEST WITH AN EXAMPLE

```
public static void main(String[] args)
{
    String exprsn = "852/+3*7-";
    Stack<Double> Stack = new Stack<Double>();
    System.out.println("The postfix value is : " + evaluatePostfix(Stack,exprsn));
}
```

```
The postfix value is : 24.5
```

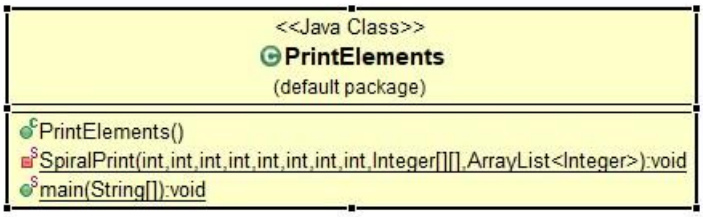
TEST 2: TEST WITH ANOTHER EXAMPLE

```
public static void main(String[] args)
{
    String expression = "87*9+3/4-";
    //String expression = "852/+3*7-";
    Stack<Double> Stack = new Stack<Double>();
    System.out.println("The postfix value is : " + evaluatePostfix(Stack,expression));
}
```

```
The postfix value is : 17.666666666666668
```

HOMEWORK 4 QUESTION 3.6 REPORT (Clockwise Array)

1)Class Diagram



2)Problem Solution Approach

In this method, firstly I would like to say I am not satisfied that I don't like using many parameters in one method. But every time instead of creating again and again in recursive method, I sent from main one time and that's it.

In my recursive method, my base case is if step number which is index is equal to array size, then return.

My directions are 6 2 4 8 like keyboard layout (6-right, 2-down, 4-left, 8-up)

In this method, initially I go to the right side until my y index is greater than array's column index. Because that means we don't have more elements to the right. Every step that I call same direction recursive method, index increases one by one. And then I turn the down direction with recursive method. I go to the this direction until x index is greater than array's row index. And now I go to the left direction but these recursive method runs until if y index is less than tour count. Tour count means that when this program complete a tour (we can just think like clock), that means we don't go to these indexes again.For example we complete one tour, tour count is 1, and we won't go index[0][0] or [2][0] etc. When I go to the up side, the same thing is valid. For instance if the tour count 1, then we won't go to the [0][1] or [0][2] etc.

3)Test Cases

Test 1 : Test Homework Array

4)Running Command and Results

This is my method :

```
private static void SpiralPrint(int row, int column, int xIndex, int yIndex, int index, int direction, int tourCount,
        int size,Integer[][] array,ArrayList<Integer> temp) {
    if(index == size) return ;
    else if(direction == 6) {
        if(yIndex < column) {
            temp.add(array[xIndex][yIndex]);
            SpiralPrint(row,column,xIndex,yIndex+1,index+1,6,tourCount,size,array,temp);
        }
        else SpiralPrint(row,column,xIndex+1,yIndex-1,index,2,tourCount,size,array,temp);
    }
    else if(direction == 2)
    {
        if(xIndex < row)
        {
            temp.add(array[xIndex][yIndex]);
            SpiralPrint(row,column,xIndex+1,yIndex,index+1,2,tourCount,size,array,temp);
        }
        else SpiralPrint(row,column,xIndex-1,yIndex-1,index,4,tourCount,size, array ,temp);
    }
    else if(direction == 4) {
        if(yIndex >= tourCount ) {
            temp.add(array[xIndex][yIndex]);
            SpiralPrint(row,column,xIndex,yIndex-1,index+1,4,tourCount,size,array,temp);
        }
        else SpiralPrint(row,column,xIndex-1,yIndex+1,index,8,tourCount,size, array ,temp);
    }
    else if(direction == 8) {
        if(xIndex > tourCount ) {
            temp.add(array[xIndex][yIndex]);
            SpiralPrint(row,column,xIndex-1,yIndex,index+1,8,tourCount,size,array,temp);
        }
        else SpiralPrint(row-1,column-1,tourCount+1,tourCount+1,index,6,tourCount+1,size, array ,temp);
    }


}
```

TEST 1 : TEST HOMEWORK ARRAY

```
public static void main(String[] args) {
    Integer[][] array = {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16},{17,18,19,20}};
    int size = array.length*array[0].length;
    ArrayList<Integer> temp = new ArrayList<Integer>(array.length*array[0].length) ;
    SpiralPrint(array.length,array[0].length,0,0,0,6,0,size,array,temp);


    for(int i =0 ; i< temp.size() ; ++i) {
        System.out.printf("%d ",temp.get(i));
    }
}
```

```
1 2 3 4 8 12 16 20 19 18 17 13 9 5 6 7 11 15 14 10
```

This method also works correctly and this is my last method of this HOMEWORK 4