HOMEWORK 4

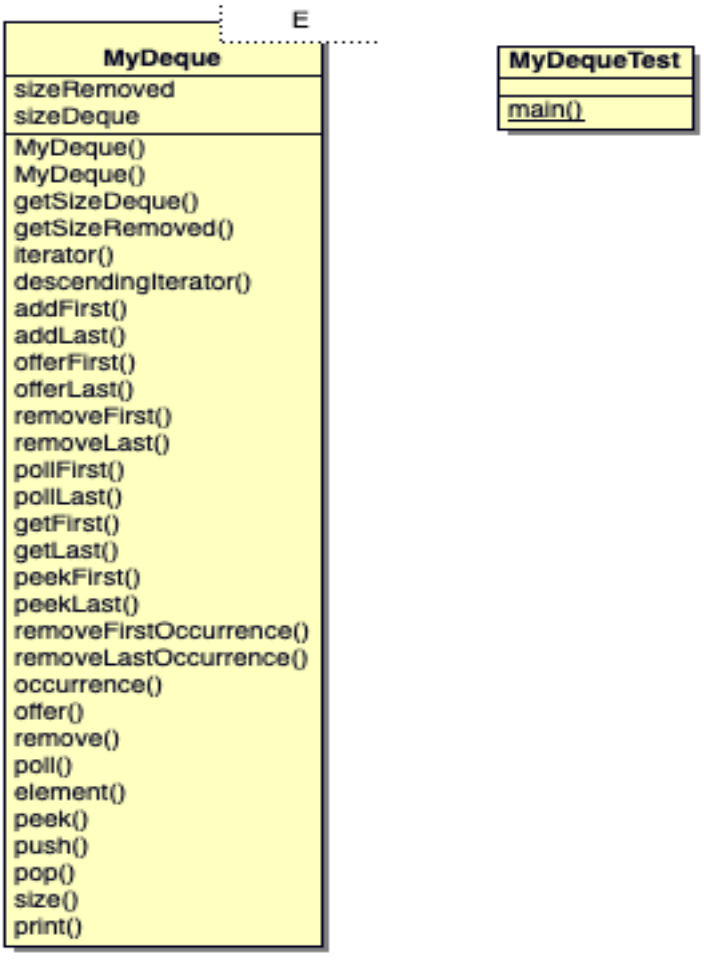
REPORT

171044010

DİLARA KARAKAŞ

QUESTION 2

1) CLASS DIAGRAM



2)PROBLEM SOLUTION APPROACH

Java provides the Deque interface. The name deque (pro- nounced "deck") is short for double-ended queue, which means that it is a data structure that allows insertions and removals from both ends (front and rear). Methods are provided to insert, remove, and examine elements at both ends of the deque. Method names that end in *first* access the front of the deque, and method names that end in *last* access the rear of the deque.

Method	Behavior
boolean offerFirst(E item)	Inserts item at the front of the deque. Returns true if successful; returns false if the item could not be inserted
boolean offerLast(E item)	Inserts item at the rear of the deque. Returns true if successful; returns false if the item could not be inserted
void addFirst(E item)	Inserts item at the front of the deque. Throws an exception if the item could not be inserted
void addLast(E item)	Inserts item at the rear of the deque. Throws an exception if the item could not be inserted
E pollFirst( )	Removes the entry at the front of the deque and returns it; returns null if the deque is empty
E pollLast( )	Removes the entry at the rear of the deque and returns it; returns null if the deque is empty
E removeFirst( )	Removes the entry at the front of the deque and returns it if the deque is not empty. If the deque is empty, throws a NoSuchElementException
E removeLast( )	Removes the item at the rear of the deque and returns it. If the deque is empty, throws a NoSuchElementException
E peekFirst( )	Returns the entry at the front of the deque without removing it; returns null if the deque is empty
E peekLast( )	Returns the item at the rear of the deque without removing it; returns null if the deque is empty
E getFirst( )	Returns the entry at the front of the deque without removing it. If the deque is empty, throws a NoSuchElementException
E getLast( )	Returns the item at the rear of the deque without removing it. If the deque is empty, throws a NoSuchElementException
boolean removeFirstOccurrence	Removes the first occurrence of item in the deque. Returns true (Object item) if the item was removed
boolean removeLastOccurrence(Object item)	Removes the last occurrence of item in the deque. Returns true if the item was removed
Iterator<E> iterator( )	Returns an iterator to the elements of this deque in the proper sequence
Iterator<E> descendingIterator( )	Returns an iterator to the elements of this deque in reverse sequential order

The Difference Between Queue and Deque

The Deque interface extends the Queue interface, which means that a class that implements Deque also implements Queue. The Queue methods are equivalent to Deque methods. If elements are always inserted at the front of a deque and removed from the rear (FIFO), then the deque functions as a queue. In this case, you could use either method add or addLast to insert a new item.

Queue Method	Equivalent Deque Method
add(e)	addLast(e)
offer(e)	offerLast(e)
remove()	removeFirst()
poll()	pollFirst()
element()	getFirst()
peek()	peekFirst()

3) TEST CASE

MyDeque deletes rare and front. MyDeque rare and erases from the front. It stores each deleted node elsewhere. It does the adding process in the same way. If there are deleted nodes, they will use them first. These should be observed during the test.

MyDeque : 1 0

OfferAdd(2) : 2 1 0

OfferLast(3) : 2 1 0 3

PollFirst() : 1 0 3 Removed : 2

RemovedLast() : 1 0 Removed : 2 3

AddLast(10) : 1 0 10 Removed : 3

4) TEST RESULTS

```
******** TEST INTEGER ********
** My Deque **
 0
** My Removed **


**** addFirst(1) ****
** My Deque **
 1 0


** My Removed **


**** offerFirst (2) ****
** My Deque **
 2 1 0


** My Removed **


**** offerLast (3) ****
** My Deque **
 2 1 0 3


** My Removed **


**** addLast (5) ****
** My Deque **
 2 1 0 3 5


** My Removed **


**** pollFirst ****
** My Deque **
 1 0 3 5
```

```
**** pollFirst ****
** My Deque **
 1 0 3 5


** My Removed **
 2

**** removeLast ****
** My Deque **
 1 0 3


** My Removed **
 5 2



**** addLast (10) ****
** My Deque **
 1 0 3 10


** My Removed **
 2

**** removeFirst ****
** My Deque **
 0 3 10


** My Removed **
 1 2



**** addFirst (5) ****
** My Deque **
 5 0 3 10


** My Removed **
 2
```

```
**** getFirst  ****
5
**** getLast  ****
10
**** peekFirst  ****
5
**** peekLast  ****
10
*** removeLastOccurrence (0)***
** My Deque **
 5 3 10


** My Removed **
 0 2



*** removeFirstOccurrence (3)***
** My Deque **
 5 10


** My Removed **
 3 0 2






******** TEST DOUBLE ********
** My Deque **
 0.0
** My Removed **


**** addFirst(1.1) ****
** My Deque **
 1.1 0.0


** My Removed **
```

```
** My Removed **


**** offerFirst (2.0) ****
** My Deque **
 2.0 1.1 0.0


** My Removed **


**** offerLast (3.3) ****
** My Deque **
 2.0 1.1 0.0 3.3


** My Removed **


**** addLast (5.2) ****
** My Deque **
 2.0 1.1 0.0 3.3 5.2


** My Removed **


**** pollFirst ****
** My Deque **
 1.1 0.0 3.3 5.2


** My Removed **
 2.0

**** removeLast ****
** My Deque **
 1.1 0.0 3.3


** My Removed **
 5.2 2.0
```

```
**** addLast (10.1) ****
** My Deque **
 1.1 0.0 3.3 10.1


** My Removed **
 2.0

**** removeFirst ****
** My Deque **
 0.0 3.3 10.1


** My Removed **
 1.1 2.0



**** addFirst (5.4) ****
** My Deque **
 5.4 0.0 3.3 10.1


** My Removed **
 2.0

**** getFirst  ****
5.4
**** getLast  ****
10.1
**** peekFirst  ****
5.4
**** peekLast  ****
10.1
*** removeLastOccurrence (0.0)***
** My Deque **
 5.4 3.3 10.1


** My Removed **
 0.0 2.0
```

```
*** removeFirstOccurrence (3.3)***
** My Deque **
 5.4 10.1


** My Removed **
 3.3 0.0 2.0




Process finished with exit code 0
```

QUESTION 3

1) CLASS DIAGRAM

2)PROBLEM SOLUTION APPROACH

- REVERSE STRING
  - The string is assigned to the word string array according to the spaces. Array was suppressed in reverse.
- WORD ELFISH
  - The word is taken as a string. Char char control is done. Returns true if it contains 'l', 'e', and 'f', false if it does not.
- SORTING
  - The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning.
  - In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.
- PREFIX
  - Step 1: Put a pointer P at the end of the end
  - Step 2: If character at P is an operand push it to Stack
  - Step 3: If the character at P is an operator pop two elements from the Stack. Operate on these elements according to the operator, and push the result back to the Stack
  - Step 4: Decrement P by 1 and go to Step 2 as long as there are characters left to be scanned in the expression.
  - Step 5: The Result is stored at the top of the Stack, return it
  - Step 6: End

- POSTFIX
  - Create a stack to store operands (or values).
  - Scan the given expression and do following for every scanned element.
    - If the element is a number, push it into the stack
    - If the element is a operator, pop operands for the operator from stack. Evaluate the operator and push the result back to the stack
  - When the expression is ended, the number in the stack is the final answer.
- SPIRAL
  - Right: x : same y : =< row-1
    - Print
    - Else down
  - Up: y : same x : >= shell
    - Print
    - Else shell ++ if loop is ok.
  - Down : y: same x : =< column – shell
    - Print
    - Else left
  - Left : x : same y : >= shell – 1
    - Print
    - Else up
  - The mathematics of the coordinates was found in this way. The function stopped when each print was held in one variable and all the data was pressed.

3) TEST CASE

Reverse before

this function writes the sentence in reverse

Reverse after

reverse in sentence the writes function this

Elfish control

Yes elfish

True

no ***ish

False

Sorted Before

4 5 9 1

Sorted After

1 4 5 9

Postfix

5712*-1/+6+31/-

13

Prefix

-++5/-7*1216/31

13

1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 16

17 18 19 20

12348121620191817139567111514 10

4) TEST RESULTS

```
Reverse Before
this function writes the sentence in reverse
Reverse After
reverse in sentence the writes function this
Elfish Control
yes elfish
true
no ***ish
false
Sorted Before
4 5 9 1
Sorted After
1 4 5 9
Postfix Control
5712*-1/+6+31/-
13
Prefix Control
-++5/-7*1216/31
13
Spiral Test
1  2  3  4
5  6  7  8
9  10  11  12
13  14  15  16
17  18  19  20


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


1  2  3  4
5  6  7  8
9  10  11  12
13  14  15  16
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10



Process finished with exit code 0
```