Dt : 13/10/2023

Ex-program : DemoStack.java

```
package p2;
import java.util.*;
public class DemoStack {
    @SuppressWarnings("removal")
    public static void main(String[] args) {
       Scanner s = new Scanner(System.in);
       try(s;){
        try {
            Stack<Integer> ob = new Stack<Integer>();
            while(true) {
                System.out.println("****Choice****");
                System.out.println("\t1.push(E)"
                     + "\n\t2.pop()"
                     + "\n\t3.peek()"
                     + "\n\t4.search(Object)"
                     + "\n\t5.Exit");
                System.out.println("Enter the
Choice:");
                switch(s.nextInt())
                {
                case 1:
                    System.out.println("Enter the
Element:");
                    ob.push(new ~~Integer~~(s.nextInt()));
                    System.out.println("====Stack-
Elements====");
                    System.out.println(ob.toString());
                    break;
                case 2:
                    if(ob.empty()){
                        System.out.println("Stack is
empty...");
                    }else {
                        ob.pop();
                        System.out.println("====Stack-
Elements====");
System.out.println(ob.toString());
                             }
                             break;
                       case 3:
                             if(ob.empty()) {
                                   System.out.println("Stack is 
empty...");
                             }else {
                                   System.out.println("peek 
ele:"+ob.peek());
                                   System.out.println("====Stack-
Elements====");

System.out.println(ob.toString());
                             }
                             break;
                       case 4:
                             if(ob.empty()) {
                                   System.out.println("Stack is 
empty...");
                             }else {
                                   System.out.println("Enter the 
ele to be Searched:");
                                   Integer el = new 
~~Integer~~(s.nextInt());
                                   int p = ob.search(el);
                                   if(p>0) {
                                         System.out.println("Ele 
found at position : "+p);
                                   }else {
                                         System.out.println("Ele 
not found in Stack...");
                                   }
                             }
                             break;
                       case 5:
                             System.out.println("Stack Operation 
Stopped..");
                             System.exit(0);
```

```
                    default:
                        System.out.println("Invalid
choice...");
                    }//end of switch
                }//end of loop
            }catch(Exception e) {e.printStackTrace();}
            }//end of try with resource
        }
}
```

o/p:

```
****Choice****
        1.push(E)
        2.pop()
        3.peek()
        4.search(Object)
        5.Exit
Enter the Choice:
4
Stack is empty...
****Choice****
        1.push(E)
        2.pop()
        3.peek()
        4.search(Object)
        5.Exit
```

```
Enter the Choice:

1

Enter the Element:

11

====Stack-Elements====

[11]

****Choice****

        1.push(E)

        2.pop()

        3.peek()

        4.search(Object)

        5.Exit

Enter the Choice:

1

Enter the Element:

12

====Stack-Elements====

[11, 12]

****Choice****

        1.push(E)

        2.pop()

        3.peek()
        4.search(Object)

        5.Exit

Enter the Choice:

1

Enter the Element:

13

====Stack-Elements====

[11, 12, 13]

****Choice****

        1.push(E)

        2.pop()

        3.peek()

        4.search(Object)

        5.Exit

Enter the Choice:

1

Enter the Element:

14

====Stack-Elements====

[11, 12, 13, 14]

****Choice****

        1.push(E)
        2.pop()

        3.peek()

        4.search(Object)

        5.Exit

Enter the Choice:

1

Enter the Element:

15

====Stack-Elements====

[11, 12, 13, 14, 15]

****Choice****

        1.push(E)

        2.pop()

        3.peek()

        4.search(Object)

        5.Exit

Enter the Choice:

1

Enter the Element:

16

====Stack-Elements====

[11, 12, 13, 14, 15, 16]
```

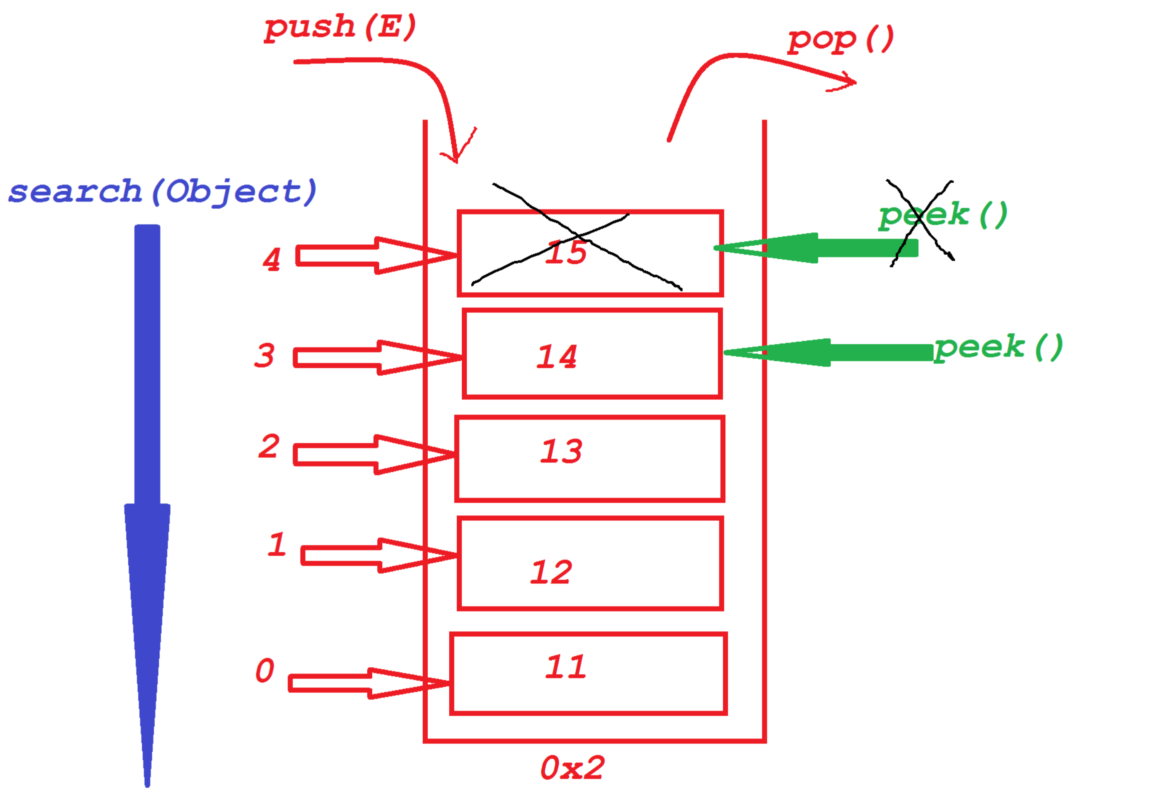
```
****Choice****
        1.push(E)
        2.pop()
        3.peek()
        4.search(Object)
        5.Exit
Enter the Choice:
1
Enter the Element:
17
====Stack-Elements====
[11, 12, 13, 14, 15, 16, 17]
****Choice****
        1.push(E)
        2.pop()
        3.peek()
        4.search(Object)
        5.Exit
Enter the Choice:
1
Enter the Element:
18
```

```
====Stack-Elements====

[11, 12, 13, 14, 15, 16, 17, 18]

****Choice****

        1.push(E)

        2.pop()

        3.peek()

        4.search(Object)

        5.Exit

Enter the Choice:

1

Enter the Element:

19

====Stack-Elements====

[11, 12, 13, 14, 15, 16, 17, 18, 19]

****Choice****

        1.push(E)

        2.pop()

        3.peek()

        4.search(Object)

        5.Exit

Enter the Choice:

1
```

```
Enter the Element:
20
====Stack-Elements====
[11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
****Choice****
        1.push(E)
        2.pop()
        3.peek()
        4.search(Object)
        5.Exit
Enter the Choice:
4
Enter the ele to be Searched:
13
Ele found at position : 8
****Choice****
        1.push(E)
        2.pop()
        3.peek()
        4.search(Object)
        5.Exit
Enter the Choice:
```

```
4

Enter the ele to be Searched:

100

Ele not found in Stack...

****Choice****

        1.push(E)

        2.pop()

        3.peek()

        4.search(Object)

        5.Exit

Enter the Choice:

5

Stack Operation Stopped..
```

Diagram:



Stack<Integer> ob = new Stack<Integer>();

===

Note:

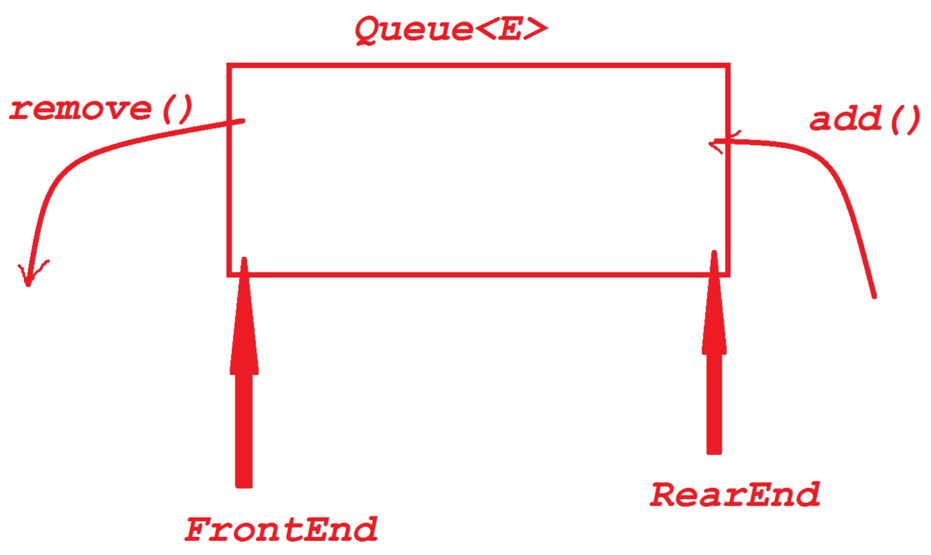
=>Stack<E> and Queue<E> is used in Algorithmic design part of Product-Based-Engineering.

===

3.Queue<E>:

=>Queue<E> is an interface from java.util package and which organizes elements based on the algorithm First-in-first-out or Last-in-last-out

Diagram:



=>The following are some important methods of Queue<E>:

public abstract boolean add(E);

public abstract boolean offer(E);

public abstract E remove();

public abstract E poll();

public abstract E element();

public abstract E peek();

=>'PriorityQueue<E>' is the implementation class of Queue<E> and which organizes elements based on Elements-priority.

=>In realtime 'PriorityQueue<E>' is used in Server-Designs.

---

define Deque<E>?

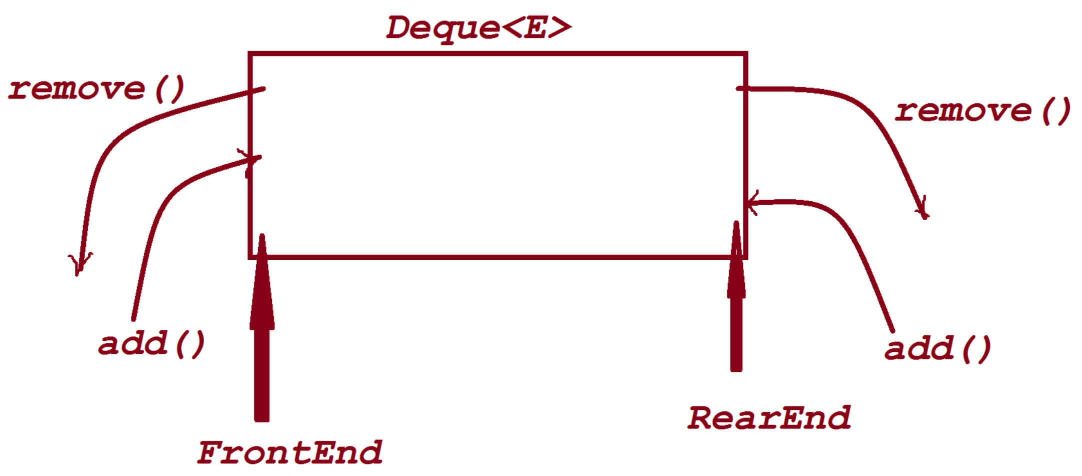
=>Deque<E> is an interface from java.util package and which is extended

from Queue<E>.

=>Deque<E> means double-ended-queue and which organizes elements on both

ends.

Diagram:



=>The following are some important methods of Deque<E>:

public abstract void addFirst(E);

```
   public abstract void addLast(E);

   public abstract boolean offerFirst(E);

   public abstract boolean offerLast(E);

   public abstract E removeFirst();

   public abstract E removeLast();

   public abstract E pollFirst();

   public abstract E pollLast();

   public abstract E getFirst();

   public abstract E getLast();

   public abstract E peekFirst();

   public abstract E peekLast();

   public abstract boolean removeFirstOccurrence(java.lang.Object);

   public abstract boolean removeLastOccurrence(java.lang.Object);


   public abstract java.util.Iterator<E> iterator();

   public abstract java.util.Iterator<E> descendingIterator();
```

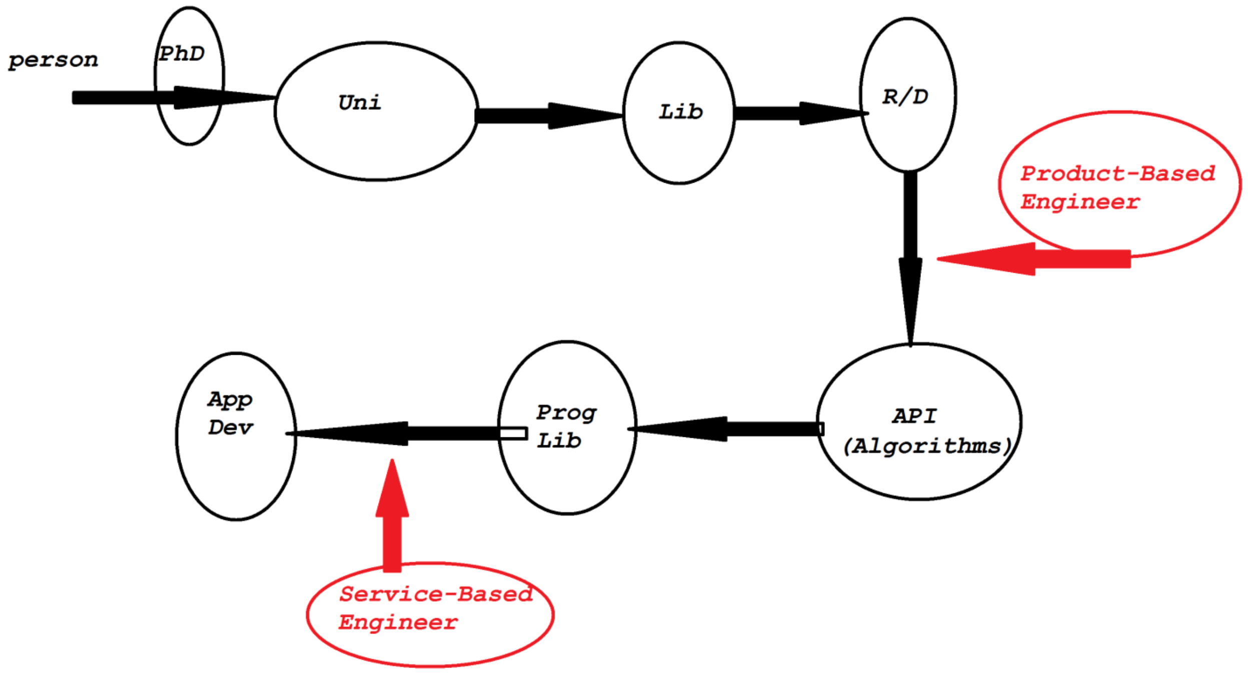
--

=>The following are the implementation classes of Deque<E>:

(a)ArrayDeque<E> - Organizes elements in Sequence

(b)LinkedList<E> - Organizes elements in NonSequence

===
===


person
PhD
Uni
Lib
R/D
Product-Based Engineer
API (Algorithms)
Prog Lib
App Dev
Service-Based Engineer