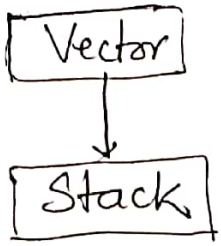
01-05-2020

Today we will try to understand the Stack class.



* Stack is a child class of Vector class.
* It is specially designed class for LIFO (Last In First-Out) order.

Constructor — Only one Constructor:-

```
Stack s=new Stack();
```

Methods :—

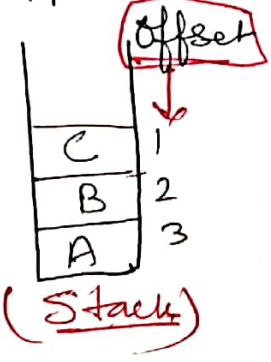
① push(Object ob) — to insert an Object to the Stack. Syntax:- Object push(Object o);

② pop() — To remove and return top of the stack. Syntax:- `Object pop();`

③ peak() — Returns the top of the stack without removal. Syntax:- `Object peak();`

④ empty() — to check whether stack is empty or not.

⑤ search("A"); — If element "A" is available returns the offset, if not → (-1). Syntax:-



(Stack)

Let us demonstrate with the small example :— StackDemo.java

```
import java.util.*;
class StackDemo{
    public static void main(String args[])
    {
```

```
Stack s = new Stack();
s.push("A"); s.push("B"); s.push("C");
System.out.println(s); //printed [A, B, C]
System.out.println(s.search("A")); // prints [3]
System.out.println(s.search("Z")); //prints [-1]
    } //close main
} //close class StackDemo
```

The three Cursors of java available for Collections Framework — to retrieve elements one-by-one from Collection: The three types of Cursors are →

1. Enumeration
2. Iterator
3. ListIterator

Let us understand — Enumeration (legacy) comes in java 1.0 Version

```
Enumeration e = v.elements();
```

Methods:-

1. hasMoreElements();
2. nextElement();

v → Vector object

We can create Enumeration Object by using elements() method of Vector class.

```
public boolean hasMoreElements()
public Object nextElement()
```

} Syntax.

Let us take an example:-

```
Vector v = new Vector();
for(int i=0; i<=10; i++)
{
   v.addElement(i);
}
System.out.println(v); // [0,1,2,3,... 10]
Enumeration e = v.elements();
while (e.hasMoreElements())
{
   Integer i = (Integer)e.nextElement();
   if(i%2==0)
      System.out.println(i);
}
System.out.println(v); // [0,1,2,3,4,... 10]
```

0
2
4
6
8
10

Now, combine the above codes in a single program as :— Enumdemo1.java

```
import java.util.*;
class Enumdemo1{
  public static void main(String args[])
  {
     Vector v = new Vector();
  for(int i=0; i<=10; i++)
  {
     v.addElement(i);
  }
  System.out.println(v);
        Enumeration e = v.elements();
```

```
        while(e.hasMoreElements())
        {
      Integer I = (Integer) e.nextElement();
      if((I % 2) == 0)
      System.out.println(I);
    }
  System.out.println(V);
  }
}
```

Limitations of Enumeration

1. Enumeration concepts are applicable only to legacy classes since it is defined in 1.0 version, hence it is not a universal cursor.
2. By using Enumeration we can get only read access and we cannot perform remove operation.

Note:- To overcome the limitations of Enumeration, we should use **Iterator**

1. We can apply Iterator concept for any Collection object hence it is universal cursor.
2. By using Iterator we can perform both read and remove operations.

Methods of Iterator — Inherited from Collection interface

1. `public Iterator iterator();`

Eg:- `Iterator itr = c.iterator();`

→ Any Collection Object

Methods from Iterator interface — got by own definition

① public boolean hasNext()

② public Object next()

③ public void remove()

We can create Iterator object by using iterator() method of Collection interface.

```
ArrayList al = new ArrayList();
for(int i=0; i<=10; i++)
{
    al.add(i);
}
System.out.println(al); // [0,1,2,3,4,5,6,7,8,9,10]
Iterator itr = al.iterator();
while(itr.hasNext())
{
    Integer num = (Integer)itr.next();
    if(num%2==0)
        System.out.println(num); // 0 2 4 6 8 10
    else
        itr.remove();
System.out.println(al); // [0,2,4,6,8,10]
}
```

Let us arrange the above logics in an example:- IteratorDemo.java

```
import java.util.*;
class IteratorDemo{
public static void main(String args[])
{
    ArrayList al = new ArrayList();
```

```
for (int i=0; i<=10; i++)
  {
    al.add(i);
  }
System.out.println("Array List has :"+al);
Iterator itr = al.iterator(); // iterator makes it separate
   while(itr.hasNext())
   {
      Integer num = (Integer)itr.next();
      if (num %2 == 0)
     System.out.println (num); // if even, printed
        else
        itr.remove(); // if odd then removed
   }
System.out.println("Array List now Contain:"+al);
} //close of main
} //close of class
```

Output is :-

```
Array List has: [0,1,2,3,4,5,6,7,8,9,10]
0
2
4
6
8
10
Array List now Contain: [0,2,4,6,8,10]
```

Problems (Limitations) with Iterator :-

1. Only forward direction.
2. We can perform — read, remove but not replace. We also cannot perform adding new elements.

In other words :-

1. By using Enumeration and Iterator we can move only towards forward direction and we cannot move to the backward direction, and hence these are single direction cursors.

Page 1 of 1

Signature of Examiner Date :

Signature of Head Examiner Date :

Signature of Examiner Date :

Signature of Head Examiner Date :

② By using Iterator we can perform only read and remove operations and we can not perform replacement of new Objects.

Note:- To overcome above limitations of Iterator we should go for ListIterator: which is Bi-directional with Read, Remove, Replace, Addition of new.

Now, we will try to know all about — ListIterator(I)

ListIterator

→ By using ListIterator interface we can move either to the forward direction or to the backward direction, and hence it is bidirectional cursor.

→ By using ListIterator we can perform replacement and addition of new Objects in addition to read and remove operations.

Method:- `public ListIterator listIterator()` → List

eg:- `ListIterator ltr = l.listIterator();`

Methods for Forward ③:
- `public boolean hasNext();`
- `public Object next();`
- `public int nextIndex();`

Iterator(I) (base) → ListIterator(I) (child)

Methods for Backward ③:
- `public boolean hasPrevious();`
- `public Object previous();`
- `public int previousIndex();`

③ other capability methods:
- `public void set(Object newobj);`
- `public void remove();`
- `public void add(Object newobj);`

Let us do this small example :- ListIteratorDemo.java

```
import java.util.*;
class ListIteratorDemo{
  public static void main(String args[])
   {
      LinkedList ob = new LinkedList();
  ob.add("Shershah"); ob.add("College");
  ob.add("Sasaram"); ob.add("Rohtas");
   ob.add("Bihar"); ob.add("821115");
 System.out.println("Original Linked List Elements:"+ob);
 ListIterator ltr = ob.listIterator();
   while(ltr.hasNext())
    { String str=(String)ltr.next();
       if(str.equals("821115")){ltr.remove();}
 else if(str.equals("Rohtas")){ltr.add("BCA");}
 else if(str.equals("Sasaram")){ltr.set("BBA");}
    }
 System.out.println("Final LinkedList elements:"+ob);
  }
}
```

removes 821115

adds BCA after Rohtas

Replaces Sasaram with BBA

Output will be :-

Original Linked List Elements: [Shershah, College, Sasaram, Rohtas, Bihar, 821115]

Final LinkedList Elements: [Shershah, College, BBA, Rohtas, BCA, Bihar]

XX