CMSC 132, Object-Oriented Programming II **Summer 2018**

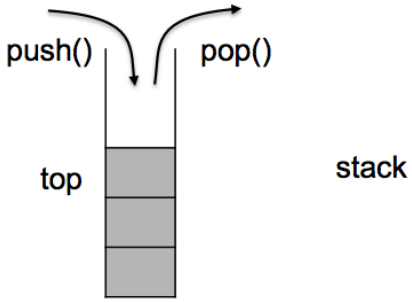
Lecture 9:

Lecturer: Anwar Mamat

9.1 STACK

A stack is a Last-In-First-Out (LIFO) data structure. It supports push, pop operations which only operations occur only at one end of the structure, referred to as the top of the stack, as shown in Figure 9.1.

Figure 9.1: Stack



Listing 1: Stack Interface

```
1 public interface Stack<T> extends Iterable<T> {
2     void push(T t);
3     T pop();// throws EmptyStackException;
4     T peek() ;//throws EmptyStackException;
5     boolean isEmpty();
6     int size();
7 }
```

If we have a stack S:

```
1 S.push(10);
```

```
1 [  ]
2 [10] <-- top
```

If we add numbers 20,30

```
1 S.push(20);
2 S.push(30);
```

```
1 [  ]
2 [30] <-- top
3 [20]
4 [10]
```

If we pop a number:

```
S.pop();
```

```
[  ]
[20]<-- top
[10]
```

Peek returns the top item, but it does not remove the item. If we pop a number:

```
int i = S.peek();//returns the top item;
```

```
[  ]
[20]<-- top
[10]
```

9.1.1 Array based implementation of Stack

In this section, we use a resizable array to implement stack. Array "Items" holds the elements of the stack. items[N] always points to an empty cel in the "Items" array. When an item is pushed, the item is stored in items[N], and N is increments. Pop does the opposite.

Listing 2: Array Based Stack

```
import java.util.Iterator;
import java.util.NoSuchElementException;
public class ArrayStack<T> implements Stack<T> {
    private T[] items;
    private int N; // number of elements in the stack
    public ArrayStack(){
        items = (T[])new Object[2];
        N = 0;
    }
    private void resize(int capacity){
        T[] temp = (T[]) new Object[capacity];
        for(int i =0; i < N; i++){
            temp[i] = items[i];
        }
        items = temp;
    }

    public void push(T item){
        if(N == items.length){
            resize(2 * items.length);
        }
        items[N++] = item;
    }

    public boolean isEmpty(){
        return N == 0;
    }
    public T pop(){
        if(isEmpty()) throw new NoSuchElementException();
        T item = items[--N];
        items[N] = null;
        return item;
    }

    public T peek()//alias top
    {
        if(isEmpty()) throw new NoSuchElementException("Stack Underflow");
```

```
38          return items[N-1];
39      }
40
41      @Override
42      public Iterator<T> iterator() {
43          return new StackIterator();
44      }
45
46      @Override
47      public int size() {
48          return N;
49      }
50      /*
51      *   Stack iterator
52      */
53      private class StackIterator implements Iterator<T>{
54          private int index;
55          StackIterator(){
56              index = N;
57          }
58          @Override
59          public boolean hasNext() {
60              return index > 0;
61          }
62
63          @Override
64          public T next() {
65              if(!hasNext()){
66                  throw new NoSuchElementException();
67              }
68              return items[--index];
69          }
70
71          @Override
72          public void remove() {
73              throw new UnsupportedOperationException("Not supported yet.");
74          }
75
76      }
77      public static void main(String[] args) {
78          Stack<Integer> as = new ArrayStack();
79          for(int i = 1; i <= 5; i++){
80              as.push(i);
81          }
82         while(!as.isEmpty()){
83              System.out.println(as.peek());
84              System.out.println(as.peek());
85              //System.out.println(as.pop());
86          }
87          for(Integer i: as){
88              System.out.println(i);
89          }
90      }
91
92  }
```

9.1.2 Linked List based implementation of Stack

In this section, we implement the stack using singly linked list. We always add and remove the nodes at the head of the linked list.

Listing 3: Linked List Based Stack

```
1  import java.util.Iterator;
```

```
2  import java.util.NoSuchElementException;
3  public class LinkedStack<T> implements Stack<T> {
4      private int N;
5      private Node first;
6
7      @Override
8      public Iterator<T> iterator() {
9          //TO DO
10     }
11
12     private class Node{
13         private T data;
14         private Node next;
15         Node(T item){
16             data = item;
17             next = null;
18         }
19     }
20     LinkedStack(){
21         first = null;
22         N = 0;
23     }
24
25     public void push(T item){
26         Node t = new Node(item);
27         Node old = first;
28         first = t;
29         first.next = old;
30         N++;
31     }
32
33     public boolean isEmpty(){
34         return first == null;
35     }
36     public T pop(){
37         if(isEmpty()){
38             throw new NoSuchElementException();
39         }
40         T item = first.data;
41         first = first.next;
42         N--;
43         return item;
44     }
45     public int size(){
46         return N;
47     }
48
49     public T peek(){
50         if(isEmpty()){
51             throw new NoSuchElementException();
52         }
53         return first.data;
54     }
55
56     public static void main(String[] args) {
57         LinkedStack<Integer> ls = new LinkedStack();
58         for(int i = 1; i <= 7; i++){
59             ls.push(i);
60         }
61         System.out.println("size:" + ls.size());
62         System.out.println("\n");
63         while(!ls.isEmpty()){
64             System.out.print(ls.peek()+",");
65             System.out.print(ls.pop()+",");
66         }
67     }
68 }
```