

## Lab 2 – Stack ADT

### Aim

Understand java generic type and Stack ADT.

### Resources

All Java files you need are found on Learning Mall.

### Tips

Check lecture notes for lecture 4 to understand Stack before starting.

### Stack Implementation

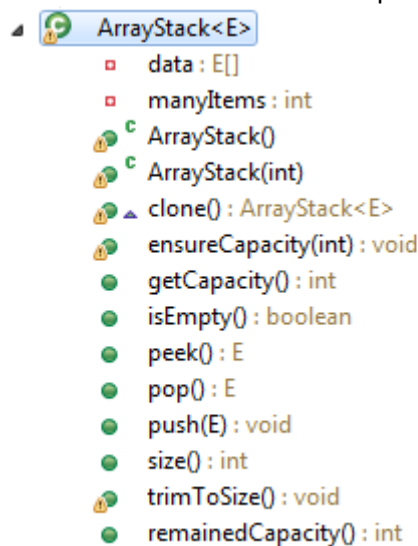
At this Lab session, we are going to implement the *ArrayStack* class, which uses **array** to store the data of our Stack structure. An *ArrayStack* is a generic collection of references to objects (objects has the same type). The collection represents a last-in-first-out (LIFO) stack of objects.

You will have to use *ArrayBag.java* from last lab as reference.

Task: Create a class called *ArrayStack* (Similar to *ArrayBag* class).

Your class should contain:

- An array called *data* to store the elements. An attribute *manyItems* stores how many items are in the stack.
- Two constructors and **all the methods** in below picture.



```
ArrayStack<E>
  data : E[]
  manyItems : int
  ArrayStack()
  ArrayStack(int)
  clone() : ArrayStack<E>
  ensureCapacity(int) : void
  getCapacity() : int
  isEmpty() : boolean
  peek() : E
  pop() : E
  push(E) : void
  size() : int
  trimToSize() : void
  remainedCapacity() : int
```

Note:

- clone():
  - Generate a copy of this array stack. Please refer to *ArrayBag* class, we will discuss about this method in detail in the coming tutorial.
  - **Returns:** a copy of the stack.
- peek():
  - Looks at the object at the top of this stack without removing it from the stack.
  - **Returns:** the object at the top of this stack (the last item of the array).

- **Throws:** *EmptyStackException* - if this stack is empty.
- **pop():**
  - Removes the object at the top of this stack and returns that object as the value of this function.
  - **Returns:** The object at the top of this stack (the last item of the array).
  - **Throws:** *EmptyStackException* - if this stack is empty.
- **push():**
  - Pushes an item onto the top of this stack. This has exactly the same effect as add an element to the stack.
  - **Parameters:** item - the item to be pushed onto this stack.
  - **Returns:** the item argument.
- **remainingCapacity():**
  - Find out how many items can be pushed to the stack still.
  - **Returns:** the remaining capacity of the stack.

Test:

Write a test class similar to below example to test your methods.

```
1 package tutorial3;
2
3
4 public class ArrayStackTest {
5
6     public static void main(String[] args){
7         ArrayStack<String> arrayStack = new ArrayStack<>();
8
9         for(String s: "This is a test for arraystack".split(" "))
10             arrayStack.push(s);
11
12         while(!arrayStack.isEmpty())
13             System.out.print(arrayStack.pop()+" ");
14     }
15 }
16
17
```

Problems @ Javadoc Declaration Console

<terminated> ArrayStackTest [Java Application] C:\Program Files\Java\jdk1.8.0\_131\bin\javaw.exe (M:  
arraystack for test a is This

Extra:

1. Have a look the API of `java.util.LinkedList` class. If you use `LinkedList` to implement our `Stack`, the implementation of stack will be very simple. See what methods (if we use `LinkedList` to store the data) you can use to implement a stack.
2. Compare below two structures. Discuss with your friends and find out what are the difference of these two ADT.

The image shows two side-by-side panels of Java API documentation. The left panel displays the `ArrayStack<E>` class, and the right panel displays the `ArrayBag<E>` class. Both classes are generic and implement a stack-like structure.

**ArrayStack<E>**

- `data : E[]`
- `manyItems : int`
- `ArrayStack()`
- `ArrayStack(int)`
- `clone() : ArrayStack<E>`
- `ensureCapacity(int) : void`
- `getCapacity() : int`
- `isEmpty() : boolean`
- `peek() : E`
- `pop() : E`
- `push(E) : void`
- `size() : int`
- `trimToSize() : void`
- `remainedCapacity() : int`

**ArrayBag<E>**

- `data : Object[]`
- `manyItems : int`
- `ArrayBag()`
- `ArrayBag(int)`
- `add(E) : void`
- `addAll(ArrayBag<E>) : void`
- `addMany(E...) : void`
- `clone() : ArrayBag<E>`
- `countOccurrences(E) : int`
- `ensureCapacity(int) : void`
- `getCapacity() : int`
- `grab() : E`
- `remove(E) : boolean`
- `size() : int`
- `trimToSize() : void`
- `union(ArrayBag<E>, ArrayBag<E>) <E> : ArrayBag<E>`

Note: If you cannot finish all the tasks, please do them as homework.