Department of Software Engineering Mehran University of Engineering and Technology, Jamshoro

Course: SWE224 – Data Structure & Algorithms			
Instructor	Mariam Memon	**Practical/Lab No.**	07
Date		**CLOs**	3
Signature		**Assessment Score**	01

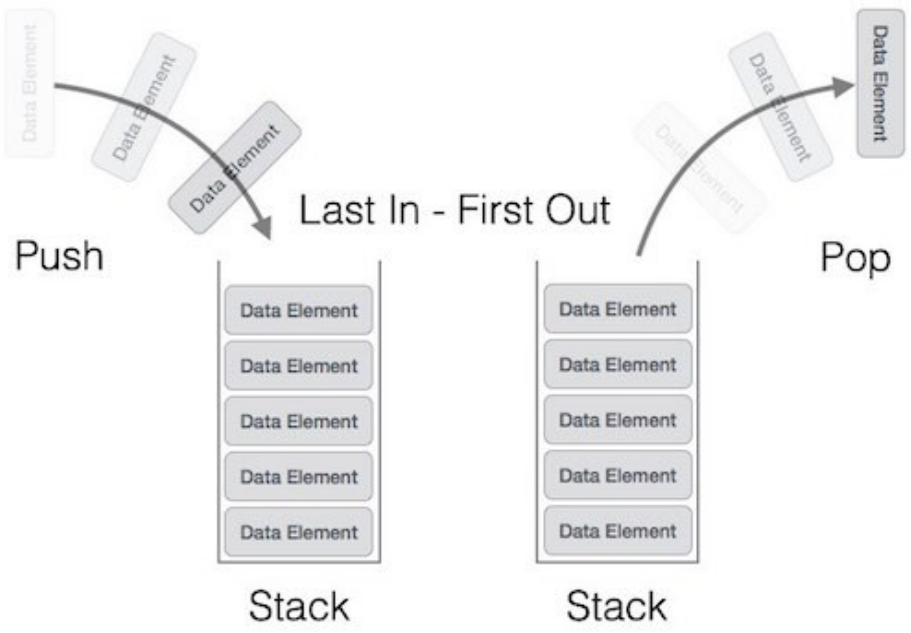
Topic	Implementation of Stacks
Objectives	To implement stack data structure and its various methods in Java.

Lab Discussion: Theoretical concepts and Procedural steps

Stack

A stack is an Abstract Data Type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc.

A real-world stack allows operations at one end only. For example, we can place or remove a card or plate from the top of the stack only. Likewise, Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack. This feature makes it LIFO data structure. LIFO stands for Last-in-first-out. Here, the element which is placed (inserted or added) last, is accessed first. In stack terminology, insertion operation is called **PUSH** operation and removal operation is called **POP** operation.



A stack can be implemented by means of Array, Structure, Pointer, and Linked List. Stack can either be a fixed size one or it may have a sense of dynamic resizing.

Basic Operations

- **push()** − Pushing (storing) an element on the stack.
- **pop()** − Removing (accessing) an element from the stack.
- **peek()** − get the top data element of the stack, without removing it.
- **size()** − returns the number of elements in the stack.

Stack Implementation in Java using Arrays

```
public class ArrayStack implements Stack
{
      private Object[] a;
      private int size;

      public ArrayStack(int capicity){
            a = new Object[capicity];
      }

      public boolean isEmpty()
      {
            return (size == 0);
      }

      public Object peek()
      {
            if(size == 0)
                  throw new IllegalStateException("Stack is empty");
            return a[size-1];
      }

      public Object pop()
      {
            Object object = a[--size];
            a[size] = null;
            return object;
      }

      public void push(Object object)
```

```
	{
		if(size == a.length) resize();
		a[size++] = object;
	}

	public int size(){
		return size;
	}

	private void resize()
	{
		Object[] aa = a;
		a = new Object[2*aa.length];
		System.arraycopy(aa,0,a,0,size);
	}

}
```

Stack Implementation in Java using Linked Structures

```
public class LinkedStack implements Stack
{
	private Node top;
	private int size;

	private static class Node{
		Object object;
		Node next;
		Node(Object object, Node next)
		{
			this.object = object;
			this.next  =next;
		}
	}

	public boolean isEmpty()
	{
		return (size == 0);
	}

	public Object peek()
	{
		if(size == 0)
			throw new IllegalStateException("Stack is empty");
```

```
            return top.object;
      }

      public Object pop()
      {
            if(size == 0)
                  throw new IllegalStateException("Stack is empty");
            Object oldTop = top.object;
            top = top.next;
            --size;
            return oldTop;
      }

      public void push(Object object)
      {
            top = new Node(object, top);
            ++size;
      }

      public int size(){
            return size;
      }
}
```

Lab Tasks

Task 1: Write and test the following methods in both ArrayStack and LinkedStack class:

1. toString()
2. equals()
3. bottomElement()
4. removeBottomElement()
5. secondElement()
6. removeSecondElement()

Task 2: Implement and test LinkedStack toLinkedStack() method in ArrayStack class.

Task 3: Implement and test ArrayStack toArrayStack() method in LinkedStack class.

Task 4: Explore java.util.Stack class; Create a stack of any type using this class and apply any 5 of its methods.

Lab Rubrics for Evaluation

Rubrics	Proficient	Adequate	Poor
Programming Algorithms	Completely accurate and efficient design of algorithms **(0.4)**	Accurate but inefficient algorithms **(0.2)**	Unable to design algorithms for the given problems **(0.0)**
Originality	Submitted work shows large amount of original thought **(0.3)**	Submitted work shows some amount of original thought **(0.2)**	Submitted work shows no original thought **(0.0)**
Troubleshooting	Easily traced and corrected faults **(0.3)**	Traced and corrected faults with some guidance **(0.2)**	No troubleshooting skills **(0.0)**