Lab 05

Stack

Objective:

This lab will introduce you the concept of Stack data structure

Activity Outcomes:

This lab teaches you the following topics:

- How to access the top of the stack
- How to push data onto the stack
- How to pop data from the stack

1) Useful Concepts

A *stack* is a ordered collection of elements in which the elements can be accessed from Top of Stack, it works in LIFO order. Its operations are:

push - push an element onto the top of the stack;

pop - pop an element from the top of the stack;

top - retrieve the element at the top of the stack;

delete - delete the whole stack.

Activity 1:

Implement a Stack in Array, use class in the implementation so that Stack can be treated as an object and its operations as public interface for the user.

Interface in java:

```
public interface Stack {

    public void push(char x);
    public char pop();
    public boolean isBalanced(String expr);
}
```

```
public class StackArray implements Stack {

	private int tos;
	private char[] values;



	public StackArray(int x) {
		this.tos = -1;
		this.values = new char[x];
	}//end constructor

	@Override
	public void push(char x) {

		if(tos==9)
			System.out.println("Stack overflow condition");

		else
			values[++tos] = x;

	}//end push

	@Override
	public char pop() {
			if(tos==-1) {
				System.out.println("Stack underflow
condition");
			}
			else {
				return values[tos--];
			}
		return 0;
	}

	@Override
	public boolean isBalanced(String expr) {
		// TODO Auto-generated method stub
		return false;
```

```
    }

    public static void main(String[] args) {

        StackArray array = new StackArray(10);
        array.push('B');
        array.push('S');
        array.push('S');
        array.push('E');
        array.push('3');
        array.push('Z');

        System.out.println(array.pop());
        System.out.println(array.pop());
        System.out.println(array.pop());
        System.out.println(array.pop());
        System.out.println(array.pop());
        System.out.println(array.pop());
        System.out.println(array.pop());


    }

}//end class
```

Output:

Z3ESSB

Stack Underflow Condition

Activity 2:

Implement Dynamic Stack.

```
public class Node {

    char value;
    Node nextNode;
```

```
        public Node(char value) {
            this.value = value;
            this.nextNode = null;
        }
    }
    public class StackLinkedList implements Stack {

         private Node head;



         public StackLinkedList() {
              head = null;
         }


         @Override
         public void push(char x) {
              Node temp = new Node(x);
              temp.nextNode = head;
              head = temp;

         }

         @Override
         public char pop() {

              if(head==null) {
                   System.out.println("Stack is Empty ...
    underflow condition");
              }
              else {
                   Node temp = head;
                   char t = temp.value;
                   head = temp.nextNode;
                   return t;
              }
                        return 0;
         }
         @Override
         public boolean isBalanced(String expr) {
              // TODO Auto-generated method stub
              return false;
         }

         public static void main(String[] args) {

              StackLinkedList linkedList = new StackLinkedList();
              linkedList.push('H');
              linkedList.push ('e');
              linkedList.push ('l');
              linkedList.push ('l');
              linkedList.push ('o');


              System.out.print(linkedList.pop());
              System.out.print(linkedList.pop());
              System.out.print(linkedList.pop());
              System.out.print(linkedList.pop());
              System.out.print(linkedList.pop()+"\n");
              System.out.print(linkedList.pop());

         }
    }
```

Output:

olleH

Activity 3:

```
StackLinkedList linkedList2 = new StackLinkedList();
        char[] str = ("I Love Programming").toCharArray();
        int len = str.length;


        for (int i = 0; i<len;i++)
```

```
                linkedList2.push(str[i]);

            for(int j=0;j<len;j++)
                System.out.print(linkedList2.pop());
```

Output:

gnimmargorP evoL I

Activity 4:

Using the stack check that the given expression has balanced parenthesis or not

```
    public boolean isBalanced(char[] expr) {
        StackLinkedList s = new StackLinkedList();
        char ch;

          for (int i = 0; i < expr.length; i++) {
               if (expr[i] == '(' || expr[i] == '{' ||
expr[i] == '[') {
                   s.push(expr[i]);
                   continue;
               }

               if (s.isEmpty()) {
                 System.out.println("Emptu");
                   return false;
               }

               switch (expr[i]) {
                   case ')':
                       ch = s.top();
                       s.pop();
                       if (ch == '{' || ch == '[') {
                           return false;
                       }
                       break;
                   case '}':
                       ch = s.top();
                       s.pop();
                       if (ch == '(' || ch == '[') {
                           return false;
```

```
                            }
                            break;
                        case ']':
                            ch = s.top();
                            s.pop();
                            if (ch == '(' || ch == '{') {
                                return false;
                            }
                            break;
                    }
                }
            return s.isEmpty();
        }

//Activity main method
        String expr = "[0]{}{[00]()}";
        System.out.println();
        System.out.println("Input expression : "+ expr);
        if(linkedList2.isBalanced(expr.toCharArray()))
        System.out.println("Balanced");
        else
        System.out.println("Not Balanced");
```

***Input**: exp = "[0]{}{[00]()}"*
***Output**: Balanced **Input**: exp = "[(]) "*
***Output**: Not Balanced*

2) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab

Lab Task 1

Write an application using Stack that checks whether the entered string of brackets is balanced, e.g.

[{( )}] is Balanced while {[( )] } is not Balanced because the priority of the pranthesis is not maintained.

Lab Task 2

Use dynamic stack and implement Infix to Postfix conversion algorithm and test it for various inputs.

Lab Task 3

For a given postfix expression, use dynamic stack to evaluate a numerical result for given values of variables.