Assignment 3

Coding Question of arraylist stack and queue

Q.1 : Create an ArrayList and search (if it is present or not)that element and tells the index. tell me if the elements is duplicate tell me the first and last occurrences of that element

Sol :

Code - >

```
import java.util.ArrayList;
import java.util.Scanner;

public class SearchInArrayList {

    public static void search(ArrayList<Integer> al ,int element){
        if(al.isEmpty()){
            return ;
        }
        int f = -1;
        int s = -1;
        for(int i=0; i<al.size(); i++){
            if(al.get(i) == element){
                if (f == -1){
                    f = i;
                } else {
                    s = i;
                }
            }
        }
        if (f==-1){
            System.out.println("Element not present");
            return;
        }
        if (s==-1){
            System.out.println("Only one occurence of element present");
        }
         System.out.println("First occurence of element is at index : " + f);
         System.out.println("Last occurence of element is at index : " + s);


    }
```

```
    public static void main(String[] args) {
        ArrayList <Integer> al = new ArrayList<>();
        Scanner sc = new Scanner(System.in);
        al.add(1);
        al.add(3);
        al.add(2);
        al.add(4);
        al.add(3);
        al.add(5);
        al.add(3);
        System.out.print("Enter element to search : ");
        int element = sc.nextInt();

        search(al,element);
    }
}
```

Output ->

Enter element to search : 3

First occurence of element is at index : 1

Last occurence of element is at index : 6

Process finished with exit code 0

Q.2 : Copy elements from one ArrayList to another

Output would be

Before copy : [One, Two, Three, Four, Five]

After copy : [1, 2, 3, Four, Five]

Sol :

Code - >

```
import java.util.ArrayList;

public class CopyArrayList {
    public static void main(String[] args) {
        ArrayList al = new ArrayList();
        al.add("One");
        al.add("Two");
        al.add("Three");
```

```
        al.add("Four");
        al.add("Five");
        System.out.print("Before copy : ");
        for (int i=0; i<al.size(); i++){
            System.out.print(al.get(i)+" ");
        }
        System.out.println();
        ArrayList alCopy = al;
        alCopy.set(0,1);
        alCopy.set(1,2);
        alCopy.set(2,3);
        System.out.print("After copy : ");
        for (int i=0; i<alCopy.size(); i++){
            System.out.print(alCopy.get(i)+" ");
        }


    }
}
```

Output ->

Before copy : One Two Three Four Five

After copy : 1 2 3 Four Five

Process finished with exit code 0

Q.3 : Swap elements of Java ArrayList At the each definedindex

Before swaping, ArrayList contains : [A, B, C, D, E]

After swaping, ArrayList contains : [E, B, C, D, A]

Sol :

Code - >

```
import java.util.ArrayList;
import java.util.Collections;

public class SwapElements {
    public static void main(String[] args) {
        ArrayList al = new ArrayList();
        al.add('A');
        al.add('B');
        al.add('C');
        al.add('D');
        al.add('E');
        System.out.print("Before swaping, ArrayList contains : ");
```

```
        for (int i=0; i<al.size(); i++){
            System.out.print(al.get(i)+" ");
        }
        System.out.println();

        Collections.reverse(al);
        System.out.print("After swaping, ArrayList contains : ");
        for (int i=0; i<al.size(); i++){
            System.out.print(al.get(i)+" ");
        }


    }
}
```

Output ->

Before swaping, ArrayList contains : A B C D E

After swaping, ArrayList contains : E D C B A

Process finished with exit code 0

Q.4/5 : Check if given Parentheses expression is balanced or not

Input: str = "(())))()()"

Output: Balanced

Input: str = "())((())"

Output: Not Balanced

Sol :

Code - >

```
import java.util.*;
public class BalancedParenthesis {

    public static boolean isValid(String s) {
        Stack <Character> st = new Stack<>();
        int i=0;
        for( i=0; i<s.length(); i++){
            if(s.charAt(i)=='(' || s.charAt(i)=='{' || s.charAt(i)=='['){
                st.push(s.charAt(i));
            } else if (!st.isEmpty() && s.charAt(i) == ')' && st.peek() == '('){
                st.pop();
```

```
            }
            else {
                return false;
            }

        }
        if(st.isEmpty() && i==s.length()){
            return true;
        }
        return false;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the parenthesis to check : ");
        String s = sc.nextLine();
        if(isValid(s)){
            System.out.println("Balanced");
        } else {
            System.out.println("Not Balanced");
        }

    }
}
```

Output-1 ->

Enter the parenthesis to check : (()())

Balanced

Process finished with exit code 0

Output-2 ->

Enter the parenthesis to check : ((()))

Not Balanced

Process finished with exit code 0

Q.6 : Reverse a stack using recursion

Sol :

Code - >

```
import java.util.Scanner;
import java.util.Stack;

public class ReverseStack {
    static Stack<Integer> st = new Stack<>();
    public static void pushEle(int x)
    {
        if (st.isEmpty()) {
            st.push(x);
        }
        else {
            int top = st.peek();
            st.pop();
            pushEle(x);
            st.push(top);
        }
    }

    public static void reverse()
    {
        if (!st.isEmpty()) {
            int x = st.peek();
            st.pop();
            reverse();
            pushEle(x);
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of elements to be pushed in stack : ");
        int n = sc.nextInt();
        for (int i=1; i<=n; i++){
            System.out.print("Push element " + i + " : ");
            int ele = sc.nextInt();
            pushEle(ele);
        }
        reverse();
        System.out.println("Reversed Stack : ");
        for (int i=1; i<=n; i++){
            System.out.print(st.pop()+" ");
        }
    }
}
```

Output ->

Enter number of elements to be pushed in stack : 6

Push element 1 : 2

Push element 2 : 3

Push element 3 : 5

Push element 4 : 6

Push element 5 : 7

Push element 6 : 1

Reversed Stack :

1 7 6 5 3 2

Process finished with exit code 0

Q.7 : Design a stack that returns the minimum element

Sol :

Code - >

```
import java.util.Scanner;
import java.util.Stack;

public class MinimumEle {
    public static void main(String[] args) {
        Stack<Integer> st = new Stack<>();
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of elements to be pushed in stack : ");
        int n = sc.nextInt();
        for (int i=1; i<=n; i++){
            System.out.print("Push element " + i + " : ");
            int ele = sc.nextInt();
            st.push(ele);
        }
        System.out.println();
        int minVal = Integer.MAX_VALUE;
        while(!st.isEmpty()){
            minVal =Math.min(st.pop(),minVal);
        }
        System.out.println("Minimum value in stack is : " + minVal);
    }
}
```

Output ->

Enter number of elements to be pushed in stack : 5

Push element 1 : 24

Push element 2 : 4

Push element 3 : 3

Push element 4 : 11

Push element 5 : 5

Minimum value in stack is : 3

Process finished with exit code 0

Q.8 : Implement a Stack using queue Data Structure

Sol :

Code - >

```
      import java.util.*;
import java.util.LinkedList;

public class StackClass {
   public static  class Stack {

       //Define the data members
       static Queue<Integer> q1
              = new LinkedList<Integer>();
       static Queue<Integer> q2
              = new LinkedList<Integer>();
       static int size;



       /*---------------- Public Functions of Stack ----------------*/


       public static int getSize() {
          //Implement the getSize() function
          return size;
       }

       public static boolean isEmpty() {
          //Implement the isEmpty() function
          return size == 0;
       }
```

```
    public static void push(int element) {
        //Implement the push(element) function
        q1.add(element);
        size++;

    }

    public static int pop() {
        //Implement the pop() function
        if (q1.isEmpty()) {
            return -1;
        }
        while (q1.size() != 1) {
            q2.add(q1.remove());
        }
        int top = (int) q1.remove();

        while (!q2.isEmpty()) {
            q1.add(q2.remove());
        }
        size = size - 1;
        return top;
    }

    public static int top() {
        //Implement the top() function
        if (q1.isEmpty()) {
            return -1;
        }
        while (q1.size() != 1) {
            q2.add(q1.remove());
        }
        int top = (int) q1.peek();
        q2.add(q1.remove());
        Queue<Integer> q = q1;
        q1 = q2;
        q2 = q;
        return top;

    }
}


    public static void main(String[] args) {
       Stack st = new Stack();
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of elements to be pushed in stack : ");
        int n = sc.nextInt();
        for (int i = 1; i <= n; i++) {
            System.out.print("Push element " + i + " : ");
```

```
            int ele = sc.nextInt();
            st.push(ele);
        }
        System.out.println("size of stack  is : " + st.getSize());
        System.out.println("Top of the stack is : " + st.top());
        System.out.println("Popped element is : " + st.pop());
        System.out.println("Top of the stack is : " + st.top());
        System.out.println("size of stack  is : " + st.getSize());
        if (!st.isEmpty()) {
            System.out.println("Stack not empty");
        } else {
            System.out.println("Stack is empty");
        }
        while (!st.isEmpty()){
            st.pop();
        }

        if (!st.isEmpty()) {
            System.out.println("Stack not empty");
        } else {
            System.out.println("All elements are popped ");
        }



    }

}
```

Output ->

Enter number of elements to be pushed in stack : 5

Push element 1 : 2

Push element 2 : 4

Push element 3 : 1

Push element 4 : 7

Push element 5 : 9

size of stack is : 5

Top of the stack is : 9

Popped element is : 9

Top of the stack is : 7

size of stack is : 4

Stack not empty

All elements are popped

Process finished with exit code 0