1. Write a java program to demonstrate the working of Stack using array.

Code:

```
import java.util.Scanner;
class MyStack {
    int maxSize;
    int top;
    int[] stackArray;

    public MyStack(int size) {
        maxSize = size;
        stackArray = new int[maxSize];
        top = -1;
    }
 // PUSH
    public void push(int value) {
        if (isFull()) {
            System.out.println("Stack is Full! Cannot push element!");
        } else {
            stackArray[++top] = value;
        }
    }
    // POP
    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack is Empty! Cannot pop element!!");
            return -1;
        } else {
            return stackArray[top--];
        }
    }
    // PEEK
    public int peek() {
        if (isEmpty()) {
            System.out.println("Stack is Empty! Cannot peek element.");
            return -1;
        } else {
            return stackArray[top];
        }
    }

    public boolean isFull() {
        return (top == maxSize - 1);
    }
```

```
    public boolean isEmpty() {
        return top == -1;
    }
    void displayAll() {
        if (isEmpty()) {
            System.out.println("\nStack is Empty! No elements to display!!");
        } else {
            System.out.println("Elements in the stack are: ");
            for (int i = top; i >= 0; i--) // Changed to i >= 0
                System.out.println(stackArray[i]);
            System.out.println("");
        }
    }
}

public class StackDemo {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        MyStack stack = new MyStack(5);
        int val;
        int choice;
        do {
            System.out.println("\n1. Push");
            System.out.println("2. Pop");
            System.out.println("3. Peek");
            System.out.println("4. Display");
            System.out.println("5. Exit");
            System.out.print("Enter Your Choice: ");
            choice = sc.nextInt();
            switch (choice) {
                case 1:
                    System.out.print("\nEnter Value to be Pushed: ");
                    val = sc.nextInt();
                    stack.push(val);
                    break;
                case 2:
                    val = stack.pop();
                    System.out.println("\nPopped element is: " + val);
                    break;
                case 3:
                    System.out.println("Top element is: " + stack.peek());
                    break;
                case 4:
                    stack.displayAll();
                    break;
```

```
            case 5:
                System.out.println("Exiting the program.");
                break;
            default:
                System.out.println("\nWrong Choice\n");
            }
        } while (choice != 5);
        sc.close();
    }
}
```

Output

```
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter Your Choice: 1

Enter Value to be Pushed: 20

1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter Your Choice: 3
Top element is: 20

1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter Your Choice: 4
Elements in the stack are:
20
```

```
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter Your Choice: 2

Popped element is: 20

1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter Your Choice: 5
Exiting the program.
```

2. Write a java program to demonstrate the working of Queue using array.

Code:

```
import java.util.Scanner;
class MyQueue
{
	int queueSize;int arr[];
	int q_front,q_rear;
	public MyQueue(int size)
	{
		queueSize=size;
		arr=new int[queueSize];
		q_front=0;
		q_rear=-1;
	}
	void enqueue(int val)
	{
		q_rear=q_rear+1;
		arr[q_rear]=val;
	}
	public int dequeue()
	{
		int q_element=arr[q_front];
		q_front=q_front+1;
		return q_element;
	}
	public boolean isEmpty()
	{
		if (q_rear==-1 ||q_front>q_rear)
			return true;
		else
			return false;
	}
	public boolean isFull()
	{
		if (q_rear==queueSize-1)
			return true;
		else
			return false;
	}
	public int size()
	{
		return q_rear+1;
	}
```

```
        public void displayAllElement()
        {
                if (isEmpty())
                {
                        System.out.println("\nQueue is Empty!! No element Display");
                }else
                {
                        System.out.println("Element in the queue are:  ");
                        for(int i=q_front;i<=q_rear;i++)
                        {
                                System.out.print(arr[i]+" ");
                        }
                        System.out.println(" ");
                }
        }
}
public class QueueDemo {
        public static void main(String[] args) {
            Scanner sc = new Scanner(System.in);
            MyQueue queue = new MyQueue(5);
            int val;
            int choice;
            do {
               System.out.println("\n1. Enqueue");
               System.out.println("2. Dequeue");
               System.out.println("3. Check if Queue is Empty");
               System.out.println("4. Check if Queue is Full");
               System.out.println("5. Display All Elements");
               System.out.println("6. Exit");
               System.out.print("Enter Your Choice: ");
               choice = sc.nextInt();
               switch (choice) {
                  case 1:
                     System.out.print("\nEnter Value to be Enqueued: ");
                     val = sc.nextInt();
                     queue.enqueue(val);
                     break;
                  case 2:
                     val = queue.dequeue();
                     System.out.println("\nDequeued element is: " + val);
                     break;
                  case 3:
                     System.out.println("Queue is " + (queue.isEmpty()?"Empty":"Not
Empty"));
                     break;
```

```
                case 4:
                    System.out.println("Queue is " + (queue.isFull() ? "Full" : "Not
Full"));
                    break;
                case 5:
                    queue.displayAllElement();
                    break;
                case 6:
                    System.out.println("Exiting the program.");
                    break;
                default:
                    System.out.println("\nWrong Choice\n");
                }
            } while (choice != 6);
            sc.close();
        }
    }
```

Output:

```
1. Enqueue
2. Dequeue
3. Check if Queue is Empty
4. Check if Queue is Full
5. Display All Elements
6. Exit
Enter Your Choice: 1

Enter Value to be Enqueued: 20

1. Enqueue
2. Dequeue
3. Check if Queue is Empty
4. Check if Queue is Full
5. Display All Elements
6. Exit
Enter Your Choice: 5
Element in the queue are:
20

1. Enqueue
2. Dequeue
3. Check if Queue is Empty
4. Check if Queue is Full
5. Display All Elements
6. Exit
Enter Your Choice: 2

Dequeued element is: 20
```

```
1. Enqueue
2. Dequeue
3. Check if Queue is Empty
4. Check if Queue is Full
5. Display All Elements
6. Exit
Enter Your Choice: 3
Queue is Empty

1. Enqueue
2. Dequeue
3. Check if Queue is Empty
4. Check if Queue is Full
5. Display All Elements
6. Exit
Enter Your Choice: 4
Queue is Not Full

1. Enqueue
2. Dequeue
3. Check if Queue is Empty
4. Check if Queue is Full
5. Display All Elements
6. Exit
Enter Your Choice: 6
Exiting the program.
```

3. Write a java program to demonstrate the working of Circular Queue using array.

Code:

```
import java.util.Scanner;
class MyCircularQueue {
    int QUEUESIZE;
    int arr[];
    int q_front;
    int q_rear;
    public MyCircularQueue(int size) {
        QUEUESIZE = size;
        arr = new int[QUEUESIZE];
        q_front = -1;
        q_rear = -1;
    }
    public void enqueue(int val) {
        if (isEmpty()) {
            q_rear = 0;
            q_front = 0;
            arr[q_rear] = val;
        } else {
            q_rear = (q_rear + 1) % QUEUESIZE;
            arr[q_rear] = val;
        }
    }
    public int dequeue() {
        int q_element;
        if (q_front == q_rear) {
            q_element = arr[q_front];
            q_rear = -1;
            q_front = -1;
        } else {
            q_element = arr[q_front];
            q_front = (q_front + 1) % QUEUESIZE;
        }
        return q_element;
    }
    public boolean isEmpty() {
        return q_rear == -1;
    }
```

```
    public boolean isFull() {
        return (q_rear + 1) % QUEUESIZE == q_front;
    }
    public int size() {
        if (q_rear >= q_front)
            return q_rear - q_front + 1;
        else
            return QUEUESIZE - q_front + q_rear + 1;
    }
    public void displayAllElements() {
        if (q_rear == -1) {
            System.out.println("No elements to display!");
            return;
        }
        System.out.println("Elements in the queue are: ");
        for (int i = q_front; i != q_rear; i = (i + 1) % QUEUESIZE) {
            System.out.print(arr[i] + " ");
        }
        System.out.print(arr[q_rear] + " ");
        System.out.println("");
    }
}
public class CircularQueueDemo {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        MyCircularQueue q = new MyCircularQueue(5); // Initialize the circular queue
        int val;
        int choice;
        do {
            System.out.println("\n1. Insert (enqueue)");
            System.out.println("2. Delete (dequeue)");
            System.out.println("3. Display");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            choice = sc.nextInt();
            switch (choice) {
                case 1:
                    if (!q.isFull()) {
                        System.out.print("\nEnter Value to be enqueued (inserted): ");
                        val = sc.nextInt();
                        q.enqueue(val);
                    } else {
                        System.out.println("\nQueue is full, cannot enqueue element!");
                    }
                    break;
```

```
            case 2:
                if (!q.isEmpty()) {
                    val = q.dequeue();
                    System.out.println("\nDequeued element is: " + val);
                } else {
                    System.out.println("\nQueue is empty, cannot dequeue element!");
                }
                break;
            case 3:
                q.displayAllElements();
                break;
            case 4:
                System.out.println("Exiting the program.");
                break;
            default:
                System.out.println("\nWrong choice! ");
                break;
            }
        } while (choice != 4);
    }
}
```

Output:

```
1. Insert (enqueue)
2. Delete (dequeue)
3. Display
4. Exit
Enter your choice: 1

Enter Value to be enqueued (inserted): 20

1. Insert (enqueue)
2. Delete (dequeue)
3. Display
4. Exit
Enter your choice: 3
Elements in the queue are:
20

1. Insert (enqueue)
2. Delete (dequeue)
3. Display
4. Exit
Enter your choice: 2

Dequeued element is: 20
```

```
1. Insert (enqueue)
2. Delete (dequeue)
3. Display
4. Exit
Enter your choice: 3
No elements to display!

1. Insert (enqueue)
2. Delete (dequeue)
3. Display
4. Exit
Enter your choice: 4
Exiting the program.
```

4. Write a java program to demonstrate the application of stack for evaluating postfix expression.

Code:

```
import java.util.Scanner;
import java.lang.Math;
class MyStack2 {
    private int maxSize;
    private int top;
    private int[] stackArray;
    public MyStack2(int size) {
        maxSize = size;
        stackArray = new int[maxSize];
        top = -1;
    }
    public void push(int value) {
        if (isFull()) {
            System.out.println("Stack is full! Cannot push element!");
        } else {
            stackArray[++top] = value;
        }
    }
    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack is empty! Cannot pop element!");
            return -1;
        } else {
            return stackArray[top--];
        }
    }
    public int peek() {
        if (isEmpty()) {
            System.out.println("Stack is empty!");
            return -1;
        } else {
            return stackArray[top];
        }
```

```
    }
    public boolean isFull() {
        return (top == maxSize - 1);
    }
    public boolean isEmpty() {
        return top == -1;
    }
    public void displayAll() {
        if (top == -1) {
            System.out.println("\nStack is empty! No elements to display!\n");
        } else {
            System.out.println("Elements in the stack are: ");
            for (int i = top; i >= 0; i--) {
                System.out.println(stackArray[i] + " ");
            }
            System.out.println("");
        }
    }
}
public class PostfixEval {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        MyStack2 st = new MyStack2(5);
        System.out.println("Please enter postfix expression: ");
        String exp = sc.nextLine();
        for (int i = 0; i < exp.length(); i++) {
            char c = exp.charAt(i);
            if (Character.isDigit(c)) {
                st.push(c - '0');
            } else {
                int val1 = st.pop();
                int val2 = st.pop();
                System.out.println(val2 + " " + c + " " + val1);
                switch (c) {
                    case '+':
                        st.push(val2 + val1);
                        break;
                    case '-':
                        st.push(val2 - val1);
```

```
                break;
            case '/':
                st.push(val2 / val1);
                break;
            case '*':
                st.push(val2 * val1);
                break;
            case '^':
                int power = (int) Math.pow(val2, val1);
                st.push(power);
                break;
            default:
                System.out.println("Invalid operator found!");
        }
      }
    }
    System.out.println("Answer = " + st.pop());
    sc.close();
  }
}
```

Output:

```
Please enter postfix expression:
42+3*5/2-3+
4 + 2
6 * 3
18 / 5
3 - 2
1 + 3
Answer = 4
```

5. Write a java program to demonstrate the application of stack for parenthesis balancing.

Code:

```
import java.util.Scanner;
class MyStack3 {
   int maxSize;
```

```
int top;
char[] stackArray;
public MyStack3(int size) {
   maxSize = size;
   stackArray = new char[maxSize];
   top = -1;
}
// PUSH
public void push(char value) {
   if (isFull()) {
      System.out.println("Stack is Full! Cannot push element!");
   } else {
      stackArray[++top] = value;
   }
}
// POP
public char pop() {
   if (isEmpty()) {
      System.out.println("Stack is Empty! Cannot pop element!!");
      return 'E';
   } else {
      return stackArray[top--];
   }
}
// PEEK
public char peek() {
   if (isEmpty()) {
      System.out.println("Stack is Empty! Cannot peek element.");
      return 'E';
   } else {
      return stackArray[top];
   }
}
public boolean isFull() {
   return (top == maxSize - 1);
}
public boolean isEmpty() {
   return top == -1;
}
```

```
    void displayAll() {
        if (isEmpty()) {
            System.out.println("\nStack is Empty! No elements to display!!");
        } else {
            System.out.println("Elements in the stack are: ");
            for (int i = top; i >= 0; i--)
                System.out.println(stackArray[i]);
            System.out.println("");
        }
    }
}
public class ParenthesisBalanecing {
                public static void main (String [] args) {
                        Scanner sc =new Scanner(System.in);
                        MyStack3 st= new MyStack3(10);
                        System.out.println("Please Enter a postfix ecpression");
                        String exp=sc.nextLine();
                        int isValid=1;
                        char c;
                        for( int i=0;i<exp.length();i++)
                        {
                                c=exp.charAt(i);
                                char stChar;
                                if(c=='{' || c=='[' || c=='(')
                                {
                                        st.push(c);
                                }
                                else
                                {
                                        if (st.isEmpty())
                                        {
                                                isValid=0;
                                                break;
                                        }
                                        else
                                        {
                                                stChar =st.pop();
```

```
                                        if((stChar!='(' && c==')') ||(stChar!='{' &&
c=='}')||(stChar!='[' && c==']'))
                                        {
                                                isValid=0;
                                                break;
                                        }
                                }
                        }
                }
                if(isValid==1 && st.isEmpty())
                        System.out.println("\nParemthesis are Balanced");
                else {
                        System.out.println("\nParenthisis are not balanced!");
                        sc.close();
                }
        }
}
```

Output:

```
Please Enter a postfix ecpression
{[()]}

Paremthesis are Balanced
```

```
Please Enter a postfix ecpression
{[(])}

Parenthisis are not balanced!
```

6. Write a java program to demonstrate the application of stack for conversion of Infix into postfix expression.

Code:

```
import java.util.Scanner;

class MyStack1 {
   int maxSize;
   int top;
   char [] stackArray;
   public MyStack1(int size) {
      maxSize = size;
      stackArray = new char[maxSize];
```

```
    top = -1;
}
// PUSH
public void push(char value) {
    if (isFull()) {
        System.out.println("Stack is Full! Cannot push element!");
    } else {
        stackArray[++top] = value;
    }
}
// POP
public char pop() {
    if (isEmpty()) {
        System.out.println("Stack is Empty! Cannot pop element!!");
        return 'E';
    } else {
        return stackArray[top--];
    }
}
// PEEK
public char peek() {
    if (isEmpty()) {
        System.out.println("Stack is Empty! Cannot peek element.");
        return 'E';
    } else {
        return stackArray[top];
    }
}
public boolean isFull() {
    return (top == maxSize - 1);
}
public boolean isEmpty() {
    return top == -1;
}
void displayAll() {
    if (isEmpty()) {
        System.out.println("\nStack is Empty! No elements to display!!");
    } else {
        System.out.println("Elements in the stack are: ");
```

```
            for (int i = top; i >= 0; i--) // Changed to i >= 0
                System.out.println(stackArray[i]);
            System.out.println("");
        }
    }
}
public class InfixToPostfix {
          int prec(char c) {
        if (c == '^')
            return 3;
        else if (c == '/' || c == '*')
            return 2;
        else if (c == '+' || c == '-')
            return 1;
        else
            return -1;
    }
          void infixToPostfix(String s,MyStack1 st) {
               StringBuilder result= new StringBuilder();

               for(int i=0;i<s.length();i++)
               {
                    char c=s.charAt(i);
                    if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <=
'9'))
                      result.append(c);
                    else if(c=='(')
                              st.push('(');
                    else if (c == ')') {
                      while (st.peek() != '(') {
                         result.append(st.pop());
                      }
                      st.pop();
                   }
                    else {
                      while (!st.isEmpty() && (prec(c) < prec(st.peek()) ||
                                        prec(c) == prec(st.peek()))) {
                         result.append(st.pop());
                      }
```

```
                st.push(c);
            }
        }
        while (!st.isEmpty()) {
            result.append(st.pop());
        }
        System.out.println(result.toString());
    }
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        MyStack1 stack1=new MyStack1(15);
        System.out.println("Please Enter infix Expression");
        String exp=sc.nextLine();
        InfixToPostfix ip =new InfixToPostfix();
        ip.infixToPostfix(exp, stack1);
        sc.close();
    }
}
```

Output:

```
Please Enter infix Expression
a+b*(c^d-e)^(f+g*h)-i
abcd^e-fgh*+^*+i-
```

Conclusion:

Stacks and queues are fundamental data structures used to manage data efficiently. A **stack** follows the **LIFO (Last In, First Out)** principle, meaning the most recently added element is the first to be removed. This structure is commonly used in applications like expression evaluation, function call handling, and undo mechanisms in software.

In contrast, a **queue** operates on the **FIFO (First In, First Out)** principle, where elements are processed in the order they arrive, making it ideal for scheduling tasks, managing resources, and handling tasks sequentially. Both data structures serve distinct purposes: stacks are suited for tasks that require reverse-order processing, while queues are optimal for tasks that need to be handled in the same order they were received.