```java
public interface ArrayStack<T> {
    boolean isEmpty();
    boolean isFull();
    T pop();
    void push(T elem) throws FullStackException;
    int size();
    T top();
}
```

```java
public class EmptyQueueException extends Exception{
    public EmptyQueueException() {
    }

    public EmptyQueueException(String message) {
        super(message);
    }

    @Override
    public String getMessage() {
        return super.getMessage();
    }

    @Override
    public void printStackTrace() {
        super.printStackTrace();
    }
}
```

```java
public class FullQueueException extends Exception{
    public FullQueueException() {
    }

    public FullQueueException(String message) {
        super(message);
    }

    @Override
    public String getMessage() {
        return super.getMessage();
    }

    @Override
    public void printStackTrace() {
        super.printStackTrace();
    }
}
```

```java
public class FullStackException extends Exception{
    public FullStackException(String message) {
        super(message);
    }

    @Override
    public void printStackTrace() {
        super.printStackTrace();
    }

    @Override
    public String getMessage() {
        return super.getMessage();
    }
}
```

```java
public interface IQueue<T>{
    boolean isEmpty();
    boolean isFull();
    T dequeue() throws EmptyQueueException;
    void enqueue(T elem) throws FullQueueException;
    int size();
    T first() throws EmptyQueueException;
}
```

ZADANIE 1:

```java
package Zadanie1;

import Interfaces.EmptyQueueException;
import Interfaces.FullQueueException;
import Interfaces.IQueue;

import java.util.Arrays;
import java.util.StringJoiner;

public class LimitedQueue<T> implements IQueue<T> {

    private static final int DEFAULT_CAPACITY = 16;
    T[] queue;
    int endIndex;

    public LimitedQueue(int size) {
        queue=(T[])new Object[size+1];
        endIndex = queue.length;
    }

    public LimitedQueue() {
        this(DEFAULT_CAPACITY);
    }

    @Override
    public boolean isEmpty() {
        return queue[queue.length-1]==null;
    }

    @Override
    public boolean isFull() {
        return queue[0]!=null;
    }

    @Override
    public T dequeue() throws EmptyQueueException {
        if(isEmpty())
            throw new EmptyQueueException();
            T retValue=queue[queue.length-1];
            for (int i = queue.length-1; i > 0; i--) {
                queue[i] = queue[i-1];
            }
            queue[0] = null;
            endIndex++;

        return retValue;
    }

    @Override
    public void enqueue(T elem) throws FullQueueException {
        if(isFull())
            throw new FullQueueException();
        queue[--endIndex] = elem;
    }
    @Override
    public int size() {
        return (endIndex + queue.length) % queue.length;
    }

    @Override
    public T first() throws EmptyQueueException {
        if(isEmpty())
            throw new EmptyQueueException();
        return queue[queue.length-1];
    }

    @Override
    public String toString() {
        return
Arrays.toString(Arrays.copyOfRange(queue,endIndex,queue.length));
    }
}
```

TEST ZADANIE 1:

```java
public class LimitedQueueTest {
    public static void main(String[] args) throws FullQueueException,
EmptyQueueException {
        LimitedQueue<Integer> lq = new LimitedQueue<>();
        System.out.println(lq);
        for (int i = 0; i < 17; i++) {
            lq.enqueue(i);
            System.out.println(lq);
        }

        for (int i = 0; i < 5; i++) {
            lq.dequeue();
            System.out.println(lq.first());
            System.out.println(lq);
        }


    }
}
```

KONSOLA ZADANIE 1:

```
[]

[0]

[1, 0]

[2, 1, 0]

[3, 2, 1, 0]

[4, 3, 2, 1, 0]

[5, 4, 3, 2, 1, 0]

[6, 5, 4, 3, 2, 1, 0]

[7, 6, 5, 4, 3, 2, 1, 0]

[8, 7, 6, 5, 4, 3, 2, 1, 0]

[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

[11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

[12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

[13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

[14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

[16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

1

[16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

2

[16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2]

3

[16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3]

4

[16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4]

5

[16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5]
```

ZADANIE 2:

```java
package Zadanie2;

import Interfaces.ArrayStack;

import java.util.Arrays;
import java.util.EmptyStackException;
import java.util.StringJoiner;

public class DynamicStack<T> implements ArrayStack<T> {

    private static final int DEFAULT_CAPACITY = 4;
    private T[] stack;
    private int topIndex;

    public DynamicStack() {
        this(DEFAULT_CAPACITY);
    }

    public DynamicStack(int initialSize){
        stack=(T[])(new Object[initialSize]);
        topIndex=0;
    }

    @Override
    public boolean isEmpty() {
        return topIndex==0;
    }

    @Override
    public boolean isFull() {
        return topIndex==stack.length;
    }

    @Override
    public T pop() {
        if(isEmpty())
            throw new EmptyStackException();
        T ret = stack[topIndex-1];
        stack[--topIndex] = null;
        if(topIndex<=(stack.length/4))
        {
            stack = Arrays.copyOf(stack, stack.length/2);
        }
        return ret;
    }

    @Override
    public void push(T elem){
        stack[topIndex++] = elem;
        if(topIndex>=((stack.length*3)/4))
        {
            stack = Arrays.copyOf(stack, stack.length*2);
        }
    }

    @Override
    public int size() {
        return topIndex;
    }

    @Override
    public T top() {
        return stack[topIndex-1];
    }

    @Override
    public String toString() {
        return "Stack=
"+Arrays.toString(Arrays.copyOfRange(stack,0,topIndex)).toString();
    }
}
```

TEST ZADANIE 2:

```
public class DynamicStackTest {
    public static void main(String[] args) {
        DynamicStack<Integer> ds = new DynamicStack<>();

        for (int i = 0; i < 30; i++) {
            ds.push(i);
            System.out.println(ds);
        }
        for (int i = 0; i < 5; i++) {
            ds.pop();
            System.out.println(ds);
        }

    }
}
```

KONSOLA ZADANIE 2:

```
Stack= [0]
Stack= [0, 1]
Stack= [0, 1, 2]
Stack= [0, 1, 2, 3]
Stack= [0, 1, 2, 3, 4]
Stack= [0, 1, 2, 3, 4, 5]
Stack= [0, 1, 2, 3, 4, 5, 6]
Stack= [0, 1, 2, 3, 4, 5, 6, 7]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
29]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27]
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
```

```
Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]

Stack= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
```

ZADANIE 3:

```java
public class StackWithQueues<T> {
    private LimitedQueue<T> q1 = new LimitedQueue<>();
    private LimitedQueue<T> q2 = new LimitedQueue<>();

    private int curr_size;

    public StackWithQueues() {
        curr_size = 0;
    }

    public void push(T x) throws FullQueueException, EmptyQueueException {

        curr_size++;
        if(q1.isEmpty()){
            q1.enqueue(x);
        }else{
            for (int i = 0; i < curr_size-1; i++) {
                q2.enqueue(q1.dequeue());
            }
            q1.enqueue(x);

            for (int i = 0; i < curr_size-1; i++) {
                q1.enqueue(q2.dequeue());
            }
        }

    }

    public void pop() throws EmptyQueueException {
        if (q1.isEmpty())
            return;
        q1.dequeue();
        curr_size--;
    }

    public T top() throws EmptyQueueException {
        if (q1.isEmpty()){
            throw new EmptyQueueException();}
        return q1.first();
    }

    public int size() {
        return curr_size;
    }

    @Override
    public String toString() {
        return "q1=" + q1+"\nq2=" + q2+"\nsize="+size();
    }
}
```

TEST ZADANIE 3:

```java
public class StackWithQueuesTest {
    public static void main(String[] args) {
        StackWithQueues<Integer> s = new StackWithQueues<>();

        try{
            s.push(1);
            System.out.println(s);
            s.push(2);
            System.out.println(s);
            s.push(3);
            System.out.println(s);
            System.out.println("Rozmiar stosu: " + s.size());
            System.out.println(s.top());
            s.pop();
            System.out.println(s.top());
            s.pop();
            System.out.println(s.top());
            System.out.println("current size: " + s.size());
        }catch (Exception ex){
            ex.printStackTrace();
        }

    }
}
```

KONSOLA ZADANIE 3:

```
q1=[1]

q2=[]

size=1

q1=[1, 2]

q2=[]

size=2

q1=[1, 2, 3]

q2=[]

size=3

Rozmiar stosu: 3

3

2

1

current size: 1
```

ZADANIE 4:

```
package Zadanie4;

import Zadanie2.DynamicStack;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Zadanie4 {
    static DynamicStack<String> stack = new DynamicStack<>();

    public static void main(String[] args) {
        StringBuilder result = new StringBuilder();
        String statement = readFromFile();
        boolean sign = true;
        statement = statement.replaceAll(" ","");

        for (int i=0; i<statement.length(); i++) {
            if (statement.charAt(i) == '(') {
                stack.push("(");
                sign = true;
                result.append(" ");
            } else if (statement.charAt(i) == ')') {
                result.append(" ").append(getFromStackUntilBracket());
                sign = false;
            } else if ((statement.charAt(i) == '+' ||
                    statement.charAt(i) == '-' ||
                    statement.charAt(i) == '*' ||
                    statement.charAt(i) == '/') && !sign) {
                result.append(" ").append(getFromStack(statement.sub-
string(i, i + 1)));
                sign = true;
            } else {
                if (sign && statement.charAt(i) == '-') {
                    result.append(" ");
                }
                result.append(statement.charAt(i));
                sign = false;
            }
        }
        result.append(getAllFromStack());
        result = new StringBuilder(result.toString().replaceAll("  ", "
"));
        System.out.println("ONP: " + result);
        System.out.println("Wartość: "+ calculate(result.toString()));
    }

    private static String getFromStackUntilBracket() {
        String result = "";
        String c = null;
        if (!stack.isEmpty()) {
            c = (String) stack.pop();
            while (!c.equals("(")){
                result = result + " " + c;
                if (stack.isEmpty()) break;
                c = (String) stack.pop();
            }
        }
        if (result.length() > 0) {
            result = " " + result;
        }
        return result;
    }

    private static String getFromStack(String operator) {
        StringBuilder result = new StringBuilder();
        String c = null;
        if (!stack.isEmpty()) {
            c = stack.pop();
            while (((operator.equals("+") || operator.equals("-")) &&
!c.equals("(")) ||
                    ((operator.equals("/") || operator.equals("*")) &&
(c.equals("/") || c.equals("*")))){
                result.append(" ").append(c);
                if (stack.isEmpty()) break;
                c = (String) stack.pop();
            }
            stack.push(c);
        }
        stack.push(operator);

        return result.toString();
    }

    private static String getAllFromStack() {
        StringBuilder result = new StringBuilder();
        String c = null;
        while (!stack.isEmpty()){
            c = (String) stack.pop();
            result.append(" ").append(c);
        }
        return result.toString();
    }

    private static double calculate(String result) {
        result = result.trim();
        DynamicStack<Double> values = new DynamicStack<>();
        String[] tokens = result.split(" +");
        double answer = 0.0;

        for (String token: tokens)
        {
            if(!token.equals("+") && !token.equals("-") && !to-
ken.equals("*") && !token.equals("/"))
            {
                double value = Double.parseDouble(token);
                values.push(value);
            }
            else
            {
                double a = values.pop();
                double b = values.pop();
                switch (token) {
                    case "+" -> answer = a + b;
                    case "-" -> answer = b - a;
                    case "*" -> answer = a * b;
                    case "/" -> answer = b / a;
                }
                values.push(answer);
            }
        }
        return answer;

    }

    private static String readFromFile() {
        try {
            File myObj = new File("plik.txt");
            Scanner myReader = new Scanner(myObj);
            while (myReader.hasNextLine()) {
                String data = myReader.nextLine();
                return data;
            }
            myReader.close();
        } catch (FileNotFoundException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
        return "";
    }

}
```

PLIK:

6 / (4 * 2)

KONSOLA:

```
ONP: 6 4 2  * /
Wartość: 0.75
```