Zadanie1:

Interfejs:

public interface IQueue<T> {  
 boolean isEmpty();  
 boolean isFull();  
 T dequeue() throws EmptyQueueException;  
 void enqueue(T element) throws FullQueueException;  
 int size();  
 T first() throws EmptyQueueException;  
 T top() throws EmptyQueueException;  
 T get(int index) throws EmptyQueueException;  
}

Błędy:

public class FullQueueException extends Exception{  
}

public class EmptyQueueException extends Exception{  
}

Kolejka:

public class ArrayQueue<T> implements IQueue<T> {  
  
 private final T[] array;  
 private final int beginIndex;  
 private int endIndex;  
  
 @SuppressWarnings("unchecked")  
 public ArrayQueue(int size) {  
 array = (T[]) new Object[size];  
 beginIndex = 0;  
 endIndex = 0;  
 }  
  
 @Override  
 public boolean isEmpty() {  
 return beginIndex == endIndex;  
 }  
  
 @Override  
 public boolean isFull() {  
 return endIndex == array.length;  
 }  
  
 @Override  
 public T dequeue() throws EmptyQueueException {  
 if (isEmpty())  
 throw new EmptyQueueException();  
 T retValue = array[beginIndex];  
 if (endIndex - 1 >= 0) System.*arraycopy*(array, 1, array, 0, endIndex - 1);  
 endIndex--;  
 return retValue;  
 }  
  
 @Override  
 public void enqueue(T element) throws FullQueueException {  
 if (isFull())  
 throw new FullQueueException();  
 array[endIndex++] = element;  
 }  
  
 @Override  
 public int size() {  
 return endIndex;  
 }  
  
 @Override  
 public T first() throws EmptyQueueException {  
 if (isEmpty())  
 throw new EmptyQueueException();  
 return array[beginIndex];  
 }  
  
 public T top() throws EmptyQueueException {  
 if (isEmpty())  
 throw new EmptyQueueException();  
 return array[endIndex-1];  
 }  
  
 public T get(int index) throws EmptyQueueException {  
 if (isEmpty())  
 throw new EmptyQueueException();  
 if (index > endIndex || index < 0)  
 throw new IndexOutOfBoundsException();  
 return array[index-1];  
 }  
  
}

Testy pokazujące że działa: (robiłem to w folderze z testami, wszystkie przeszły poprawnie)

import com.company.Z1.ArrayQueue;  
import com.company.Z1.EmptyQueueException;  
import com.company.Z1.FullQueueException;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class ArrayQueueTest {  
  
 ArrayQueue<Integer> arrayQueue;  
  
 @BeforeEach  
 public void SetUp() {  
 arrayQueue = new ArrayQueue<>(4);  
 }  
  
 @Test  
 public void IsEmptyQueueExceptionWorking() {  
 *assertThrows*(  
 EmptyQueueException.class,  
 () -> arrayQueue.dequeue()  
 );  
 }  
  
 @Test  
 public void IsFullQueueExceptionWorking() throws FullQueueException {  
 arrayQueue.enqueue(1);  
 arrayQueue.enqueue(2);  
 arrayQueue.enqueue(3);  
 arrayQueue.enqueue(4);  
  
 *assertThrows*(  
 FullQueueException.class,  
 () -> arrayQueue.enqueue(5)  
 );  
 }  
  
 @Test  
 public void IsSizeWorking() throws FullQueueException, EmptyQueueException {  
 *assertEquals*(0, arrayQueue.size());  
  
 arrayQueue.enqueue(1);  
 arrayQueue.enqueue(2);  
 *assertEquals*(2, arrayQueue.size());  
  
 arrayQueue.dequeue();  
 *assertEquals*(1, arrayQueue.size());  
 }  
  
 @Test  
 public void IsEnqueAndDequeWorking() throws FullQueueException, EmptyQueueException {  
 arrayQueue.enqueue(1);  
 arrayQueue.enqueue(2);  
 arrayQueue.enqueue(3);  
  
 int a = arrayQueue.dequeue();  
 *assertEquals*(a, 1);  
  
 int b = arrayQueue.dequeue();  
 *assertEquals*(b, 2);  
  
 }  
}

Zadanie2:

Interfejs:

public interface IStack<T> {  
 boolean isEmpty();  
 // boolean isFull(); nie potrzebne do stosu nieograniczonego  
 T pop() throws EmptyStackException, EmptyQueueException;  
 void push(T elem) throws FullStackException;  
 int size();  
 T top() throws EmptyStackException, EmptyQueueException;  
}

Błędy:

public class FullStackException extends Exception {  
}

public class EmptyStackException extends Exception{  
}

Stos:

public class DynamicStack<T> implements IStack<T>{  
  
 private T[] array;  
 private int topIndex;  
 private int actualSize;  
  
 @SuppressWarnings("unchecked")  
 public DynamicStack(int size) {  
 array = (T[]) new Object[size];  
 topIndex = 0;  
 actualSize = size;  
 }  
  
 @Override  
 public boolean isEmpty() {  
 return topIndex == 0;  
 }  
  
 @Override  
 @SuppressWarnings("unchecked")  
 public T pop() throws EmptyStackException {  
 if (isEmpty())  
 throw new EmptyStackException();  
 T returnValue = array[--topIndex];  
  
 if ( topIndex == 0.25 \* actualSize ) {  
 actualSize \*= 0.5;  
 array = (T[]) new Object[actualSize];  
 }  
  
  
 return returnValue;  
 }  
  
 @Override  
 @SuppressWarnings("unchecked")  
 public void push(T element) {  
 array[topIndex++] = element;  
  
 if ( topIndex == 0.75 \* actualSize) {  
 actualSize \*= 2;  
 array = (T[]) new Object[actualSize];  
 }  
 }  
  
 @Override  
 public int size() {  
 return topIndex;  
 }  
  
 public int MaxSize() {  
 return actualSize;  
 }  
  
 @Override  
 public T top() {  
 return null;  
 }  
}

Demonstracja działania: (testy)

import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class DynamicStackTest {  
 DynamicStack<Integer> stosDynamiczny;  
  
 @BeforeEach  
 public void SetUp() {  
 stosDynamiczny = new DynamicStack<>(4);  
 }  
  
 @Test  
 public void IsGrowing() {  
 *assertEquals*(0, stosDynamiczny.size());  
  
 stosDynamiczny.push(1);  
 stosDynamiczny.push(2);  
 *assertEquals*(2, stosDynamiczny.size());  
 *assertEquals*(4, stosDynamiczny.MaxSize());  
  
 stosDynamiczny.push(3);  
 *assertEquals*(3, stosDynamiczny.size());  
 *assertEquals*(8, stosDynamiczny.MaxSize());  
  
 stosDynamiczny.push(4);  
 *assertEquals*(4, stosDynamiczny.size());  
  
 stosDynamiczny.push(5);  
 *assertEquals*(5, stosDynamiczny.size());  
 }  
  
 @Test  
 public void IsDecreasing() {  
 stosDynamiczny.push(1);  
 stosDynamiczny.push(2);  
 stosDynamiczny.push(3);  
 stosDynamiczny.push(4);  
 stosDynamiczny.push(5);  
 stosDynamiczny.push(6);  
 stosDynamiczny.push(7);  
 stosDynamiczny.push(8);  
 stosDynamiczny.push(9);  
 stosDynamiczny.push(10);  
 stosDynamiczny.push(11);  
 stosDynamiczny.push(12);  
  
 *assertEquals*(32, stosDynamiczny.MaxSize());  
  
 try {  
 stosDynamiczny.pop();  
 stosDynamiczny.pop();  
 stosDynamiczny.pop();  
 stosDynamiczny.pop();  
  
 *assertEquals*(16, stosDynamiczny.MaxSize());  
  
 stosDynamiczny.pop();  
 stosDynamiczny.pop();  
 stosDynamiczny.pop();  
 stosDynamiczny.pop();  
  
 *assertEquals*(8, stosDynamiczny.MaxSize());  
  
 stosDynamiczny.pop();  
 *assertEquals*(8, stosDynamiczny.MaxSize());  
  
  
 } catch (EmptyStackException emptyStackError) {  
 emptyStackError.printStackTrace();  
 }  
  
 }  
  
}

Zadanie 3:

import com.company.Z1.ArrayQueue;  
import com.company.Z1.EmptyQueueException;  
import com.company.Z1.FullQueueException;  
import com.company.Z1.IQueue;  
import com.company.Z2.EmptyStackException;  
import com.company.Z2.FullStackException;  
import com.company.Z2.IStack;  
  
public class StackOnTwoQueues<T> implements IStack<T> {  
  
 private final IQueue<T> iQueue;  
 private final IQueue <T> iQueue2;  
 private int topIndex;  
  
 public StackOnTwoQueues(int size) {  
 iQueue = new ArrayQueue<>(1);  
 iQueue2 = new ArrayQueue<>(size-1);  
 topIndex = 0;  
 }  
  
 @Override  
 public boolean isEmpty() {  
 return iQueue.isEmpty();  
 }  
  
 public boolean isFull() {  
 return iQueue2.isFull();  
 }  
  
 @Override  
 public T pop() throws EmptyStackException, EmptyQueueException {  
 if (isEmpty())  
 throw new EmptyStackException();  
  
 if (topIndex == 1) {  
 try {  
 return iQueue.dequeue();  
 } catch (EmptyQueueException e) {  
 e.printStackTrace();  
 }  
 }  
  
 --topIndex;  
  
 return iQueue2.get(topIndex);  
 }  
  
  
  
 @Override  
 public void push(T elem) throws FullStackException {  
  
 if (isFull())  
 throw new FullStackException();  
  
 if (topIndex == 0) {  
 topIndex++;  
 try {  
 iQueue.enqueue(elem);  
 return;  
 } catch (FullQueueException e) {  
 e.printStackTrace();  
 }  
 }  
  
 topIndex++;  
  
 try {  
 iQueue2.enqueue(elem);  
 } catch (FullQueueException e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Override  
 public int size() {  
 return topIndex;  
 }  
  
 @Override  
 public T top() throws EmptyStackException, EmptyQueueException {  
 if (isEmpty())  
 throw new EmptyStackException();  
  
 if (topIndex == 1) {  
 try {  
 return iQueue.dequeue();  
 } catch (EmptyQueueException e) {  
 e.printStackTrace();  
 }  
 }  
  
 return iQueue2.dequeue();  
 }  
}

Pokazanie że działa: (test)

import com.company.Z1.EmptyQueueException;  
import com.company.Z2.EmptyStackException;  
import com.company.Z2.FullStackException;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class StackOnTwoQueuesTest {  
  
 StackOnTwoQueues<Integer> stack;  
  
 @BeforeEach  
 public void setUp() {  
 stack = new StackOnTwoQueues<>(4);  
 }  
  
 @Test  
 public void isThrowingEmptyStackError() {  
 *assertThrows*(  
 EmptyStackException.class,  
 () -> stack.pop()  
 );  
  
 *assertThrows*(  
 EmptyStackException.class,  
 () -> stack.top()  
 );  
 }  
  
 @Test  
 public void isThrowingFullStackError() throws FullStackException {  
 stack.push(1);  
 stack.push(2);  
 stack.push(3);  
 stack.push(4);  
  
 *assertThrows*(  
 FullStackException.class,  
 () -> stack.push(5)  
 );  
 }  
  
 @Test  
 public void isPopAndPushWorking() throws FullStackException, EmptyStackException, EmptyQueueException {  
  
 stack.push(1);  
 stack.push(2);  
 stack.push(3);  
  
  
 int a = stack.pop();  
 *assertEquals*(a,3);  
  
 int b = stack.pop();  
 *assertEquals*(b, 2);  
  
 int c = stack.pop();  
 *assertEquals*(c, 1);  
 }  
  
}

Zadanie 4:

Klasa pomocnicza:

import com.company.Z1.ArrayQueue;  
import com.company.Z1.EmptyQueueException;  
import com.company.Z1.FullQueueException;  
import com.company.Z1.IQueue;  
import com.company.Z2.DynamicStack;  
import com.company.Z2.EmptyStackException;  
import com.company.Z2.FullStackException;  
import com.company.Z2.IStack;  
  
import java.io.IOException;  
import java.io.StreamTokenizer;  
import java.io.StringReader;  
  
public class MyAnalyzer {  
 private static class Operator {  
 final char \_ch;  
  
 public Operator(char ch) {  
 \_ch = ch;  
 }  
  
 int getPriority() {  
 if (\_ch == '\*' || \_ch == '/') return 2;  
 if (\_ch == '+' || \_ch == '-') return 1;  
 return 0;  
 }  
  
 @Override  
 public String toString() {  
 return "" + \_ch;  
 }  
 }  
  
 private static class LeftBracket {  
 @Override  
 public String toString() {  
 return "(";  
 }  
 }  
  
 // na razie mamy tylko implementacje stosu i kolejki  
 // o ograniczonej pojemności  
 public final int MAX\_NUMBER\_OF\_TOKENS = 100;  
  
 public IQueue<Object> analize(String inputStr) {  
  
 StreamTokenizer st = new StreamTokenizer(new StringReader(inputStr));  
 st.ordinaryChar('/'); // traktuj ‘/’ jako zwykły znak  
 st.ordinaryChar('-'); // traktuj ‘-’ jako zwykły znak  
 IQueue<Object> queue = new ArrayQueue<>(MAX\_NUMBER\_OF\_TOKENS);  
 IStack<Object> stack = new DynamicStack<>(MAX\_NUMBER\_OF\_TOKENS);  
  
 try {  
 while (st.nextToken() != StreamTokenizer.*TT\_EOF*) {  
 if (st.ttype == StreamTokenizer.*TT\_NUMBER*) {  
 queue.enqueue(st.nval);  
 }  
 else if (st.ttype == '(') stack.push(new LeftBracket());  
 else if (st.ttype == ')') {  
 Object elem;  
 do {  
 elem = stack.pop();  
 if (!(elem instanceof MyAnalyzer.LeftBracket)) queue.enqueue(elem);  
 } while (!(elem instanceof MyAnalyzer.LeftBracket));  
 }  
  
 else // Operator  
 {  
 MyAnalyzer.Operator oper1 = new Operator((char) st.ttype);  
 int priorityOper1 = oper1.getPriority();  
 Object topElem;  
 while (!stack.isEmpty() &&  
 ((topElem = stack.top()) instanceof MyAnalyzer.Operator) &&  
 ((MyAnalyzer.Operator) topElem).getPriority() >= priorityOper1) {  
 queue.enqueue(stack.pop());  
 }  
 stack.push(oper1);  
 }  
 }  
 // na koniec przerzucenie elementów ze stosu na koniec kolejki  
 while (!stack.isEmpty()) {  
 queue.enqueue(stack.pop());  
 }  
 } catch (IOException | FullQueueException | EmptyStackException | FullStackException | EmptyQueueException e) {  
 e.printStackTrace();  
 }  
 return queue;  
 }  
  
 public String toRPNString(IQueue<Object> queue){  
 StringBuilder buffer = new StringBuilder();  
 IQueue<Object> copyQueue= new ArrayQueue<>(100);  
 try {  
 while(!queue.isEmpty()){  
 Object elem;  
 elem = queue.dequeue();  
 buffer.append(elem).append(" ");  
 copyQueue.enqueue(elem);  
 }  
 while(!copyQueue.isEmpty())  
 queue.enqueue(copyQueue.dequeue());  
 } catch (EmptyQueueException | FullQueueException e) {  
 // *TODO Auto-generated catch block* e.printStackTrace();  
 }  
 buffer.setLength(buffer.length() - 1);  
 return buffer.toString();  
 }  
  
 public float result(String input) throws EmptyStackException {  
  
 String[] tokens = input.split(" ");  
 DynamicStack <Float> dynamicStack = new DynamicStack<>(10);  
 float a, b;  
  
 for (String s :  
 tokens) {  
 switch (s) {  
 case "+" -> {  
 a = dynamicStack.pop();  
 b = dynamicStack.pop();  
 dynamicStack.push(b + a);  
 }  
 case "-" -> {  
 a = dynamicStack.pop();  
 b = dynamicStack.pop();  
 dynamicStack.push(b - a);  
 }  
 case "\*" -> {  
 a = dynamicStack.pop();  
 b = dynamicStack.pop();  
 dynamicStack.push(b \* a);  
 }  
 case "/" -> {  
 a = dynamicStack.pop();  
 b = dynamicStack.pop();  
 dynamicStack.push(b / a);  
 }  
 default -> dynamicStack.push(Float.*parseFloat*(s));  
 }  
  
 }  
  
 return dynamicStack.pop();  
 }  
  
}

Main:

import com.company.Z1.IQueue;  
import com.company.Z2.EmptyStackException;  
import com.company.Z4.MyAnalyzer;  
  
import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.IOException;  
  
public class Main {  
  
 public static String Load(String path) {  
 String line = null;  
  
 try(BufferedReader bufferedReader = new BufferedReader(new FileReader(path))) {  
 line = bufferedReader.readLine();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 return line;  
 }  
   
 public static void main(String[] args) {  
  
 String input = *Load*("C:\\Users\\remig\\OneDrive - Politechnika Wroclawska\\Semestr2\\LaboUBiele\\Z3\\dane.txt");  
 System.*out*.println("Postać infiksowa: " + input);  
  
 MyAnalyzer myAnalyzer = new MyAnalyzer();  
 IQueue<Object> queue = myAnalyzer.analize(input);  
 String show = myAnalyzer.toRPNString(queue);  
  
 System.*out*.println("Postać postfiksowa: " + show);  
  
 try {  
 System.*out*.println("Wynik: " + myAnalyzer.result(show));  
 } catch (EmptyStackException e) {  
 e.printStackTrace();  
 }  
 }  
}