|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| *Федеральное государственное бюджетное образовательное учреждение высшего образования* ***«МИРЭА – Российский технологический университет»***  **РТУ МИРЭА** |

**Институт информационных технологий (ИТ)**

**Кафедра инструментального и прикладного программного обеспечения (ИиППО)**

**Дисциплина «Программирование на языке Джава»**

**ОТЧЕТ**

**ПО ПРАКТИЧЕСКОМУ ЗАНЯТИЮ №7**

Выполнил студент группы ИНБО-02-20 Маркарянц Р.Д.

Принял Степанов П.В.

Практические работы выполнены «\_\_\_»\_\_\_\_\_\_\_2021г.

«\_\_\_\_\_\_\_\_\_\_\_\_\_\_» «\_\_\_»\_\_\_\_\_\_\_2021г.

Отметка о выполнении

**Москва – 2021 г.**

## **Задание**

**Задание 1. Очередь на массиве**

* Найдите инвариант структуры данных «[очередь](http://ru.wikipedia.org/wiki/%D0%9E%D1%87%D0%B5%D1%80%D0%B5%D0%B4%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))». Определите функции, которые необходимы для реализации очереди. Найдите их пред- и постусловия.
* Реализуйте классы, представляющие циклическую очередь с применением массива.
  + Класс ArrayQueueModule должен реализовывать один экземпляр очереди с использованием переменных класса.
  + Класс ArrayQueueADT должен реализовывать очередь в виде абстрактного типа данных (с явной передачей ссылки на экземпляр очереди).
  + Класс ArrayQueue должен реализовывать очередь в виде класса (с неявной передачей ссылки на экземпляр очереди).
  + Должны быть реализованы следующие функции(процедуры) / методы:
    - enqueue – добавить элемент в очередь;
    - element – первый элемент в очереди;
    - dequeue – удалить и вернуть первый элемент в очереди;
    - size – текущий размер очереди;
    - isEmpty – является ли очередь пустой;
    - clear – удалить все элементы из очереди.
  + Инвариант, пред- и постусловия записываются в исходном коде в виде комментариев.
  + Обратите внимание на инкапсуляцию данных и кода во всех трех реализациях.
* Напишите тесты реализованным классам.

**Задание 2. Очереди**

1. Определите интерфейс очереди Queue и опишите его контракт.
2. Реализуйте класс LinkedQueue — очередь на связном списке.
3. Выделите общие части классов LinkedQueue и ArrayQueue в базовый класс AbstractQueue.

**Дополнительные задания**

**Задание 3. Вычисление выражений**

1. **Разработайте классы Const, Variable, Add, Subtract, Multiply, Divide для вычисления выражений с одной переменной.**
2. **Классы должны позволять составлять выражения вида**

new Subtract(

new Multiply(

new Const(2),

new Variable("x")

),

new Const(3)

).evaluate(5)

При вычислении такого выражения вместо каждой переменной подставляется значение, переданное в качестве параметра методу evaluate (на данном этапе имена переменных игнорируются). Таким образом, результатом вычисления приведенного примера должно стать число 7.

1. **Для тестирования программы должен быть создан класс Main, который вычисляет значение выражения x2−2x+1, для x, заданного в командной строке.**
2. **При выполнение задания следует обратить внимание на:**

* **Выделение общего интерфейса создаваемых классов.**
* **Выделение абстрактного базового класса для бинарных операций.**

**Задание 4**

1. **Доработайте предыдущее задание, так что бы выражение строилось по записи вида**

x \* (y - 2)\*z + 1

Для этого реализуйте класс ExpressionParser с методом TripleExpression parse(String).

1. **В записи выражения могут встречаться: умножение \*, деление /, сложение +, вычитание -, унарный минус -, целочисленные константы (в десятичной системе счисления, которые помещаются в 32-битный знаковый целочисленный тип), круглые скобки, переменные (x, y и z) и произвольное число пробельных символов в любом месте (но не внутри констант).**
2. **Приоритет операторов, начиная с наивысшего**

* **унарный минус;**
* **умножение и деление;**
* **сложение и вычитание.**

1. **Для выражения 1000000\*x\*x\*x\*x\*x/(x-1) вывод программы должен иметь следующий вид:**

**x f**

0 0

1 division by zero

2 32000000

3 121500000

4 341333333

5 overflow

6 overflow

7 overflow

8 overflow

9 overflow

10 overflow

## **Ход Работы**

В ходе выполнения работы были получены следующие исходные коды:

package ru.mirea.inbo0220.markaryants.pr11.Task1;  
  
// INV: FIFO (First in - first out)  
// 0 <= size  
// queue[head]..queue[tail] - queue  
public class ArrayQueue {  
 private ArrayQueueModule queue;  
  
 public ArrayQueue(){  
 this.queue = ArrayQueueModule.*getInstance*();  
 }  
  
 // PRE: size > 0  
 // POST: R = queue[head]  
 // queue[head] = queue[head+1]  
 // queue[head+1]..queue[tail] - immutable  
 public Object dequeue() {  
 return queue.dequeue();  
 }  
  
 // PRE: size > 0  
 // POST: R = queue[head]  
 // queue - immutable  
 public Object element() {  
 return queue.element();  
 }  
  
 // PRE: None  
 // POST: queue[tail] = element  
 // queue[head]..queue[tail-1] - immutable  
 public void enqueue(Object o) {  
 queue.enqueue(o);  
 }  
  
 // PRE: None  
 // POST: queue - immutable  
 // R = (head == tale)  
 public boolean isEmpty() {  
 return queue.isEmpty();  
 }  
  
 // PRE: None  
 // POST: size == 0  
 public boolean clear() {  
 return queue.clear();  
 }  
}

package ru.mirea.inbo0220.markaryants.pr11.Task1;  
  
public class ArrayQueueAdt {  
  
 private ArrayQueueModule queue;  
  
 public ArrayQueueAdt(ArrayQueueModule queue){  
 this.queue = queue;  
 }  
  
 // PRE: size > 0  
 // POST: R = queue[head]  
 // queue[head] = queue[head+1]  
 // queue[head+1]..queue[tail] - immutable  
 public Object dequeue() {  
 return queue.dequeue();  
 }  
  
 // PRE: size > 0  
 // POST: R = queue[head]  
 // queue - immutable  
 public Object element() {  
 return queue.element();  
 }  
  
 // PRE: None  
 // POST: queue[tail] = element  
 // queue[head]..queue[tail-1] - immutable  
 public void enqueue(Object o) {  
 queue.enqueue(o);  
 }  
  
 // PRE: None  
 // POST: queue - immutable  
 // R = (head == tale)  
 public boolean isEmpty() {  
 return queue.isEmpty();  
 }  
  
 // PRE: None  
 // POST: size == 0  
 public boolean clear() {  
 return queue.clear();  
 }  
}

package ru.mirea.inbo0220.markaryants.pr11.Task1;  
  
import ru.mirea.inbo0220.markaryants.pr11.Task2.AbstractQueue;  
  
// INV: FIFO (First in - first out)  
// 0 <= size <= q.length - 1  
// queue[head]..queue[tail] - queue  
// queue - Singleton  
// q.length = 2\*\*x  
public class ArrayQueueModule extends AbstractQueue {  
 private Object[] q;  
  
 private static ArrayQueueModule *instance*;  
  
 // PRE: queue - null  
 // POST: queue.size = 0;  
 // q.length = 16;  
 private ArrayQueueModule(){  
 q = new Object[*START\_CAPACITY*];  
 head = tail = 0;  
 }  
  
 // Pre: none;  
 // Post: instance - Singleton;  
 // queue - immutable  
 public static ArrayQueueModule getInstance(){  
 if(*instance* == null)  
 *instance* = new ArrayQueueModule();  
 return *instance*;  
 }  
  
 // PRE: size > 0  
 // POST: R = queue[head]  
 // queue[head] = queue[head+1]  
 // queue[head+1]..queue[tail] - immutable  
 @Override  
 public Object dequeue(){  
 if(isEmpty()) throw new IndexOutOfBoundsException("Queue is empty!");  
 Object r = q[head++];  
 if(head == q.length) head = 0;  
 if(Math.*abs*(head-tail) < q.length / 2) resize(q.length / 2);  
 return r;  
 }  
  
 // PRE: None  
 // POST: queue - immutable  
 // R = (head == tale)  
 @Override  
 public boolean isEmpty() {  
 return head == tail;  
 }  
  
 // PRE: None  
 // POST: queue[tail] = element  
 // queue[head]..queue[tail-1] - immutable  
 @Override  
 public void enqueue(Object element){  
 q[tail++] = element;  
 if(tail == q.length) tail = 0;  
 if(tail == head) resize(q.length \* 2);  
 }  
  
 // Pre: none;  
 // Post: q.length == nSize;  
 // queue - immutable;  
 private void resize(int nSize) {  
 Object[] nq = new Object[nSize];  
 System.*arraycopy*(q, head, nq, 0, Math.*abs*(head-tail));  
 tail = Math.*abs*(head-tail);  
 head = 0;  
 q = nq;  
 }  
  
 // PRE: size > 0  
 // POST: R = queue[head]  
 // queue - immutable  
 @Override  
 public Object element(){  
 if(isEmpty()) throw new IndexOutOfBoundsException("Queue is empty!");  
 return q[head];  
 }  
  
 // PRE: None  
 // POST: size == 0  
 // q.length = 16  
 @Override  
 public boolean clear(){  
 boolean r = !isEmpty();  
 head = tail = 0;  
 resize(*START\_CAPACITY*);  
 return r;  
 }  
  
}

package ru.mirea.inbo0220.markaryants.pr11.Task1;  
  
public class Test {  
 public static void main(String[] args) {  
 ArrayQueueModule aqm = ArrayQueueModule.*getInstance*();  
 aqm.enqueue("Text1");  
 aqm.enqueue("Text2");  
 aqm.enqueue("Text3");  
 System.*out*.println(aqm.dequeue());  
  
 ArrayQueueAdt adt = new ArrayQueueAdt(aqm);  
 System.*out*.println(adt.dequeue());  
  
 ArrayQueue aq = new ArrayQueue();  
 System.*out*.println(aq.dequeue()); }  
}

package ru.mirea.inbo0220.markaryants.pr11.Task2;  
  
public abstract class AbstractQueue implements Queue {  
 protected int head, tail;  
 protected static final int *START\_CAPACITY* = 16;  
}

package ru.mirea.inbo0220.markaryants.pr11.Task2;  
import java.util.LinkedList;  
  
public class LinkedQueue extends AbstractQueue {  
 private LinkedList<Object> q;  
  
 public LinkedQueue(){  
 q = new LinkedList<>();  
 head = tail = 0;  
 }  
  
 @Override  
 public Object dequeue() {  
 if(isEmpty()) throw new IndexOutOfBoundsException("Queue is empty!");  
 return q.remove(head);  
 }  
  
 @Override  
 public Object element() {  
 if(isEmpty()) throw new IndexOutOfBoundsException("Queue is empty!");  
 return q.get(head); }  
  
 @Override  
 public void enqueue(Object o) {  
 q.add(o);  
 tail++;  
 }  
  
 @Override  
 public boolean isEmpty() {  
 return tail == head;  
 }  
  
 @Override  
 public boolean clear() {  
 boolean r = ! isEmpty();  
 q.clear();  
 head = tail = 0;  
 return r;  
 }  
}

package ru.mirea.inbo0220.markaryants.pr11.Task2;  
  
public interface Queue {  
 Object dequeue();  
 Object element();  
 void enqueue(Object o);  
 boolean isEmpty();  
 boolean clear();  
}

package ru.mirea.inbo0220.markaryants.pr11.Task3;  
  
public class Add extends Operator {  
 public Add(Operand left, Operand right) {  
 super(left, right);  
 }  
  
 public Add(Operand left, Variable vRight) {  
 super(left, vRight);  
 }  
  
 public Add(Variable vLeft, Variable vRight) {  
 super(vLeft, vRight);  
 }  
  
 public Add(Variable vLeft, Operand right) {  
 super(vLeft, right);  
 }  
  
 @Override  
 public double getValue() {  
 return left.getValue() + right.getValue();  
 }  
}

package ru.mirea.inbo0220.markaryants.pr11.Task3;  
  
public class Const extends Operand {  
 public Const(double value){  
 this.value = value;  
 }  
}

package ru.mirea.inbo0220.markaryants.pr11.Task3;  
  
public class Divide extends Operator {  
 public Divide(Operand left, Variable vRight) {  
 super(left, vRight);  
 }  
  
 public Divide(Variable vLeft, Variable vRight) {  
 super(vLeft, vRight);  
 }  
  
 public Divide(Variable vLeft, Operand right) {  
 super(vLeft, right);  
 }  
  
 public Divide(Operand left, Operand right) {  
 super(left, right);  
 }  
  
 @Override  
 public double getValue() {  
 return left.getValue() / right.getValue();  
 }  
}

package ru.mirea.inbo0220.markaryants.pr11.Task3;  
  
public class Multiply extends Operator {  
 public Multiply(Operand left, Operand right) {  
 super(left, right);  
 }  
  
 public Multiply(Operand left, Variable vRight) {  
 super(left, vRight);  
 }  
  
 public Multiply(Variable vLeft, Variable vRight) {  
 super(vLeft, vRight);  
 }  
  
 public Multiply(Variable vLeft, Operand right) {  
 super(vLeft, right);  
 }  
  
 @Override  
 public double getValue() {  
 return left.getValue() \* right.getValue();  
 }  
}

package ru.mirea.inbo0220.markaryants.pr11.Task3;  
  
public class Variable {  
 private String name;  
 private int pow;  
  
 public Variable(String name){  
 this.name = name;  
 pow = 1;  
 }  
  
 public Variable(String name, int pow) {  
 this.name = name;  
 this.pow = pow;  
 }  
  
 public Const toConst(double v){  
 return new Const(Math.*pow*(v, pow));  
 }  
  
 public String getName() {  
 return name;  
 }  
}

package ru.mirea.inbo0220.markaryants.pr11.Task3;  
  
public class Subtract extends Operator {  
 public Subtract(Operand left, Operand right) {  
 super(left, right);  
 }  
  
 public Subtract(Operand left, Variable vRight) {  
 super(left, vRight);  
 }  
  
 public Subtract(Variable vLeft, Variable vRight) {  
 super(vLeft, vRight);  
 }  
  
 public Subtract(Variable vLeft, Operand right) {  
 super(vLeft, right);  
 }  
  
 @Override  
 public double getValue() {  
 return left.getValue() - right.getValue();  
 }  
}

package ru.mirea.inbo0220.markaryants.pr11.Task3;  
  
public abstract class Operator extends Operand{  
 protected Operand left, right;  
 protected Variable vLeft, vRight;  
 public Operator(Operand left, Operand right){  
 this.left = left;  
 this.right = right;  
 }  
  
 public Operator(Operand left, Variable vRight) {  
 this.left = left;  
 this.vRight = vRight;  
 }  
  
 public Operator(Variable vLeft, Variable vRight) {  
 this.vLeft = vLeft;  
 this.vRight = vRight;  
 }  
 public Operator (Variable vLeft, Operand right){  
 this.vLeft = vLeft;  
 this.right = right;  
 }  
  
 @Override  
 public abstract double getValue();  
  
 public double evaluate(double v){  
 if(right == null) right = vRight.toConst(v);  
 if(left == null) left = vLeft.toConst(v);  
 if(left instanceof Operator) ((Operator) left).evaluate(v);  
 if(right instanceof Operator)((Operator) right).evaluate(v);  
 return getValue();  
 }  
}

package ru.mirea.inbo0220.markaryants.pr11.Task3;  
  
public abstract class Operand {  
 protected double value;  
  
 public double getValue() {  
 return value;  
 }  
}

package ru.mirea.inbo0220.markaryants.pr11.Task3;  
import java.util.Scanner;  
  
public class Test {  
 public static void main(String[] args) {  
 System.*out*.println(  
 new Subtract(  
 new Multiply(  
 new Const(2),  
 new Variable("x")  
 ),  
 new Const(3)  
 ).evaluate(5)  
 );  
  
 Scanner sc = new Scanner(System.*in*);  
 double v = sc.nextDouble();  
  
 System.*out*.println(  
 new Add(  
 new Subtract(  
 new Variable("x", 2),  
 new Multiply(  
 new Const(2),  
 new Variable("x")  
 )  
 ),  
 new Const(1)  
 ).evaluate(v)  
 );  
 }  
}

package ru.mirea.inbo0220.markaryants.pr11.Task4;  
  
import ru.mirea.inbo0220.markaryants.pr11.Task3.\*;  
  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.Stack;  
import java.util.regex.Pattern;  
  
public class Parser {  
  
 private ArrayList<String > prefix;  
 private int creatingStep;  
  
 public Parser(String s){  
 prefix = *infixToPreFix*(s);  
 }  
  
 public Double calc(HashMap<String , Double> v){  
 creatingStep = 0;  
 return createOpers(v).getValue();  
 }  
  
 public ArrayList<String> getPrefix() {  
 return prefix;  
 }  
  
 private Operand createOpers(HashMap<String , Double> v){  
 String s = prefix.get(creatingStep++);  
 switch (s){  
 case "+": return new Add(createOpers(v), createOpers(v));  
 case "-": return new Subtract(createOpers(v), createOpers(v));  
 case "/": return new Divide(createOpers(v), createOpers(v));  
 case "\*": return new Multiply(createOpers(v), createOpers(v));  
 default:  
 if(Pattern.*matches*("\\d+", s)) return new Const(Double.*parseDouble*(s));  
 else {  
 if(!v.containsKey(s))  
 throw new VariableNotFoundException("Переменной с именем "+s+" не задано значение!");  
 return new Variable(s).toConst(v.get(s));  
 }  
 }  
 }  
  
 private static ArrayList<String> infixToPreFix(String expression){  
 ArrayList<String > result = new ArrayList<>();  
 StringBuilder input = new StringBuilder(expression);  
 input.reverse();  
 Stack<Character> stack = new Stack<Character>();  
 char [] charsExp = new String(input).toCharArray();  
 for (int i = 0; i < charsExp.length; i++) {  
 if (charsExp[i] == '(') {  
 charsExp[i] = ')';  
 i++;  
 }  
 else if (charsExp[i] == ')') {  
 charsExp[i] = '(';  
 i++;  
 }  
 }  
 for (int i = 0; i < charsExp.length; i++) {  
 char c = charsExp[i];  
 //check if char is operator or operand  
 if (*priority*(c) > 0) {  
 while (!stack.isEmpty() && *priority*(stack.peek()) >= *priority*(c)) {  
 result.add(stack.pop().toString());  
 }  
 stack.push(c);  
 } else if (c == ')') {  
 char x = stack.pop();  
 while (x != '(') {  
 result.add(((Character) x).toString());  
 x = stack.pop();  
 }  
 } else if (c == '(') {  
 stack.push(c);  
 } else {  
 StringBuilder b = new StringBuilder();  
 //character is neither operator nor "("  
 do{  
 b.append(charsExp[i++]);  
 }  
 while(i < charsExp.length && "1234567890,".contains(((Character) charsExp[i]).toString()) &&  
 "1234567890,".contains(((Character) charsExp[i-1]).toString()));  
 result.add(b.toString());  
 i--;  
 }  
 }  
 for (int i = 0; i <=stack.size() ; i++) {  
 result.add(stack.pop().toString());  
 }  
 for(int i = 0; i < result.size(); i++){  
 result.set(i, new StringBuilder(result.get(i)).reverse().toString());  
 }  
 for(int i = 0; i < result.size()/2; i++){  
 String tmp = result.get(i);  
 result.set(i, result.get(result.size()-1-i));  
 result.set(result.size()-1-i, tmp);  
 }  
 result.removeIf(e->Pattern.*matches*(" +", e));  
 return result;  
 }  
  
 private static int priority(char operator) {  
 switch (operator) {  
 case '\*' :  
 case '/' : return 2;  
 case '+' :  
 case '-' : return 1;  
 }  
 return 0;  
 }  
}

package ru.mirea.inbo0220.markaryants.pr11.Task4;  
  
public class VariableNotFoundException extends IllegalArgumentException{  
 VariableNotFoundException(String msg){  
 super(msg);  
 }  
}

package ru.mirea.inbo0220.markaryants.pr11.Task4;  
import java.util.HashMap;  
import java.util.Scanner;  
import java.util.regex.Pattern;  
  
public class Test {  
 public static void main(String[] args) {  
 System.*out*.println("Enter a string:");  
 Scanner sc = new Scanner(System.*in*);  
 String s = sc.nextLine();  
 Parser p = new Parser(s);  
 System.*out*.println(p.getPrefix());  
 System.*out*.println("ВВодите построчно значения:\nИмя переменной\nЗначение\n...\nend - конец ввода");  
 HashMap<String, Double> m = new HashMap<>();  
 while(!(s = sc.nextLine()).equals("end")){  
 m.putIfAbsent(s, sc.nextDouble());  
 sc.nextLine();  
 }  
 try{  
 System.*out*.println(p.calc(m));  
  
 }catch (VariableNotFoundException e){  
 System.*out*.println(e.getMessage());  
 }  
 }  
}

## **Вывод**

Выполнив практическую работу, я получил навыки работы с инвариантом класса и инкапсуляцией, а также улучшил навыки в интерфейсах и абстрактных классах на языке Java.