|  |
| --- |
|  |
| 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 №11**

Выполнил студент группы ИНБО-02-20 Судариков Д.А.

Принял Степанов П.В.

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

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

Отметка о выполнении

**Москва – 2021 г.**

## **Задание**

Получить навыки работы с очередями

## **Ход Работы**

В ходе выполнения работы были получены следующие исходные коды:

package ru.mirea.lab.lab11.ex1;  
  
// 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.lab.lab11.ex1;  
  
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.lab.lab11.ex1;  
import ru.mirea.lab.lab11.ex2.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 lab11.ex2;  
  
public class Test {  
 public static void main(String[] args) {  
 System.*out*.println("LinkedQueue Test");  
 LinkedQueue<Integer> queue = new LinkedQueue<>(5);  
 queue.add(1);  
 queue.add(2);  
 queue.add(3);  
 queue.add(4);  
 queue.add(5);  
 System.*out*.println(queue);  
 queue.remove();  
 System.*out*.println("remove(): " + queue);  
 System.*out*.println("element(): " + queue.element());  
 System.*out*.println("poll(): " + queue.poll());  
 System.*out*.println("peek(): " + queue);  
 System.*out*.println("getSize(): " + queue.getSize());  
 if (queue.isEmpty()) {  
 System.*out*.println("Queue is empty");  
 } else {  
 System.*out*.println("Queue is not empty");  
 }  
  
 System.*out*.println("\n\nArrayQueue Test");  
 ArrayQueue<Integer> array = new ArrayQueue<>(5);  
 array.add(1);  
 array.add(2);  
 array.add(3);  
 array.add(4);  
 array.add(5);  
 System.*out*.println(array);  
 array.offer(6);  
 System.*out*.println("element(): " + array.element());  
 System.*out*.println("offer(): " + array);  
 System.*out*.println("poll(): " + array.poll());  
 System.*out*.println(array);  
 System.*out*.println(array.peek());  
 System.*out*.println(array);  
 array.clear();  
 System.*out*.println("clear(): " + array);  
 }  
}

----

package ru.mirea.lab.lab11.ex1;  
  
public class TestQueue {  
 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.lab.lab11.ex2;  
  
public abstract class AbstractQueue implements Queue {  
 protected int head, tail;  
 protected static final int *START\_CAPACITY* = 16;  
  
}

----

package ru.mirea.lab.lab11.ex2;  
  
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.lab.lab11.ex2;  
  
public interface Queue {  
 Object dequeue();  
 Object element();  
 void enqueue(Object o);  
 boolean isEmpty();  
 boolean clear();  
}

package ru.mirea.lab.lab11.ex3;  
  
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.lab.lab11.ex3;  
  
public class Const extends Operand {  
 public Const(double value){  
 this.value = value;  
 }  
}

package ru.mirea.lab.lab11.ex3;  
  
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.lab.lab11.ex3;  
  
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.lab.lab11.ex3;  
  
public abstract class Operand {  
 protected double value;  
  
 public double getValue() {  
 return value;  
 }  
}

package ru.mirea.lab.lab11.ex3;  
  
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.lab.lab11.ex3;  
  
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.lab.lab11.ex3;  
  
import java.util.Scanner;  
  
public class Test3 {  
 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.lab.lab11.ex3;  
  
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;  
 }  
}

## **Вывод**

В данной практической работе мной были получены навыки по работе с очередями.