|  |
| --- |
|  |
| 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 Баринов.И.В.

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 Practise11.Exercise2;  
  
public abstract class AbstractQueue<E> {  
 abstract void add(E element);  
  
 abstract E element();  
  
 abstract boolean offer(E element);  
  
 abstract E peek();  
  
 abstract E poll();  
  
 abstract E remove();  
}

----

package Practise11.Exercise2;  
  
import java.util.NoSuchElementException;  
  
public class ArrayQueue<E> extends AbstractQueue<E> {  
 private E[] array;  
 private int size;  
 private int capacity;  
 private E first;  
 private E last;  
  
 public ArrayQueue(int capacity) {  
 first = null;  
 last = null;  
 this.capacity = capacity;  
 size = 0;  
 array = (E[]) new Object[capacity];  
 }  
  
 @Override  
 void add(E element) {  
 if (size < capacity) {  
 array[size] = element;  
 size++;  
 }  
 }  
  
 @Override  
 E element() {  
 if (size != 0){  
 return array[0];  
 } else {  
 throw new NoSuchElementException("Queue is empty");  
 }  
 }  
  
 @Override  
 boolean offer(E element) {  
 if (size < capacity) {  
 array[size] = element;  
 size++;  
 return true;  
 } else {  
 return false;  
 }  
 }  
  
 void clear() {  
 array = (E[]) new Object[0];  
 size = 0;  
 }  
  
 @Override  
 E peek() {  
 if (size != 0) {  
 return array[0];  
 }  
 return null;  
 }  
  
 @Override  
 E poll() {  
 if (size != 0) {  
 E temp = array[0];  
 if (size - 1 >= 0) System.*arraycopy*(array, 1, array, 0, size - 1);  
 size--;  
 return temp;  
 }  
 return null;  
 }  
  
 @Override  
 E remove() {  
 E temp = array[0];  
 if (size - 1 >= 0) System.*arraycopy*(array, 1, array, 0, size - 1);  
 size--;  
 return temp;  
 }  
  
 public String toString() {  
 if (size != 0) {  
 for (int i = 0; i < size; i++) {  
 System.*out*.print(array[i] + " ");  
 }  
 }  
 return "ArrayQueue is empty";  
 }  
  
}

----

package Practise11.Exercise2;  
  
import java.util.NoSuchElementException;  
  
public class LinkedQueue<E> extends AbstractQueue<E> {  
 private int size;  
 private int capacity;  
 private Node first;  
 private Node last;  
  
 private class Node {  
 private E data;  
 private Node next;  
 }  
  
 public LinkedQueue() {  
 first = null;  
 last = null;  
 size = 0;  
 }  
  
 public LinkedQueue(int capacity) {  
 first = null;  
 last = null;  
 size = 0;  
 this.capacity = capacity;  
 }  
  
 public boolean isEmpty() {  
 return first == null;  
 }  
  
 public int getSize() {  
 return size;  
 }  
  
 @Override  
 void add(E element) {  
 Node oldLast = last;  
 last = new Node();  
 last.data = element;  
 last.next = null;  
 if (isEmpty()) {  
 first = last;  
 } else {  
 oldLast.next = last;  
 }  
 size++;  
 }  
  
 @Override  
 E element() {  
 if (!isEmpty()) {  
 return first.data;  
 } else {  
 throw new NoSuchElementException("Queue is empty");  
 }  
 }  
  
 @Override  
 boolean offer(E element) {  
 if (element == null) {  
 throw new NullPointerException("Null object can not br added to this queue");  
 }  
 if (size < capacity) {  
 Node oldLast = last;  
 last = new Node();  
 last.data = element;  
 last.next = null;  
 if (isEmpty()) {  
 first = last;  
 } else {  
 oldLast.next = last;  
 }  
 size++;  
 return true;  
 } else {  
 return false;  
 }  
 }  
  
 @Override  
 E peek() {  
 if (!isEmpty()) {  
 return first.data;  
 }  
 return null;  
 }  
  
 @Override  
 E poll() {  
 if (!isEmpty()) {  
 Node temp = first;  
 first = first.next;  
 size--;  
 return temp.data;  
 } else {  
 return null;  
 }  
 }  
  
 @Override  
 E remove() {  
 if (!isEmpty()) {  
 Node temp = first;  
 first = first.next;  
 size--;  
 return temp.data;  
 } else {  
 throw new NoSuchElementException("Queue is empty");  
 }  
 }  
  
 @Override  
 public String toString() {  
 Node current = first;  
 while (current != null) {  
 System.*out*.print(current.data + " ");  
 current = current.next;  
 }  
 return "LinkedQueue{" +  
 "size=" + size +  
 ", capacity=" + capacity + '}';  
 }  
}

----

package Practise11.Exercise2;  
  
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 Practise11;  
  
import java.util.Arrays;  
  
public class ArrayQueue {  
 private int size;  
 private Object[] elements = new Object[1];  
  
 public void enqueue(Object element) {  
 assert element != null;  
 ensureCapacity(size+1);  
 elements[size++] = element;  
 }  
  
 private void ensureCapacity(int capacity) {  
 if (capacity <= elements.length) {  
 return;  
 }  
 Object[] newElements = new Object[capacity];  
 if (size >= 0) System.*arraycopy*(elements, 0, newElements, 0, size);  
 elements = newElements;  
 }  
  
 public Object element() {  
 assert size > 0;  
 return elements[0];  
 }  
  
 public Object dequeue() {  
 assert size > 0;  
 Object temp = elements[0];  
 Object[] newElements = new Object[size-1];  
 System.*arraycopy*(elements, 1, newElements, 0, size - 1);  
 elements = newElements;  
 size--;  
 return temp;  
 }  
  
 public int getSize() {  
 return size;  
 }  
  
 public boolean isEmpty() {  
 return size == 0;  
 }  
  
 public void clear() {  
 elements = new Object[0];  
 size = 0;  
 }  
  
 @Override  
 public String toString() {  
 return "ArrayQueue{" +  
 "size=" + size +  
 ", elements=" + Arrays.*toString*(elements) +  
 '}';  
 }  
}

----

package Practise11;  
  
import java.util.Arrays;  
  
public class ArrayQueueADT {  
 private int size;  
 private Object[] elements = new Object[1];  
  
 public static void push(ArrayQueueADT queue, Object element) {  
 assert element != null;  
 *ensureCapacity*(queue, queue.size + 1);  
 queue.elements[queue.size++] = element;  
 }  
  
 private static void ensureCapacity(ArrayQueueADT queue, int capacity) {  
 if (capacity <= queue.elements.length) {  
 return;  
 }  
 Object[] newElements = new Object[capacity];  
 if (queue.size >= 0) System.*arraycopy*(queue.elements, 0, newElements, 0, queue.size);  
 queue.elements = newElements;  
 }  
  
 public static Object element(ArrayQueueADT queue) {  
 assert queue.size > 0;  
 return queue.elements[0];  
 }  
  
 static Object dequeue(ArrayQueueADT queue) {  
 assert queue.size > 0;  
 Object temp = queue.elements[0];  
 Object[] newElements = new Object[queue.size - 1];  
 System.*arraycopy*(queue.elements, 1, newElements, 0, queue.size - 1);  
 queue.elements = newElements;  
 queue.size--;  
 return temp;  
 }  
  
 static boolean isEmpty(ArrayQueueADT queue) {  
 return queue.size == 0;  
 }  
  
 public static int getSize(ArrayQueueADT queue) {  
 return queue.size;  
 }  
  
 static void clear(ArrayQueueADT queue) {  
 queue.elements = new Object[0];  
 queue.size = 0;  
 }  
  
 @Override  
 public String toString() {  
 return "ArrayQueueADT{" +  
 "size=" + size +  
 ", elements=" + Arrays.*toString*(elements) +  
 '}';  
 }  
}

----

package Practise11;  
  
import java.util.Arrays;  
  
public class ArrayQueueModule {  
 static int *size*;  
 static Object[] *elements* = new Object[1];  
  
 static void enqueue(Object element) {  
 assert element != null;  
 *ensureCapacity*(*size*+1);  
 *elements*[*size*++] = element;  
 }  
  
 static void ensureCapacity(int capacity) {  
 if (capacity <= *elements*.length) {  
 return;  
 }  
 Object[] newElements = new Object[capacity];  
 if (*size* >= 0) System.*arraycopy*(*elements*, 0, newElements, 0, *size*);  
 *elements* = newElements;  
 }  
  
 static Object element() {  
 assert *size* > 0;  
 return *elements*[0];  
 }  
  
 static Object dequeue() {  
 assert *size* > 0;  
 Object temp = *elements*[0];  
 Object[] newElements = new Object[*size*-1];  
 System.*arraycopy*(*elements*, 1, newElements, 0, *size* - 1);  
 *elements* = newElements;  
 *size*--;  
 return temp;  
 }  
  
 static int getSize() {  
 return *size*;  
 }  
  
 static boolean isEmpty() {  
 return *size* == 0;  
 }  
  
 static void clear() {  
 *elements* = new Object[0];  
 *size* = 0;  
 }  
  
 @Override  
 public String toString() {  
 return super.toString();  
 }  
}

----

package Practise11;  
  
import java.util.Arrays;  
  
public class Test {  
 public static void main(String[] args) {  
 System.*out*.println("ArrayQueueModule Test");  
 ArrayQueueModule.*enqueue*(5);  
 ArrayQueueModule.*enqueue*(58);  
 ArrayQueueModule.*enqueue*(59);  
 ArrayQueueModule.*enqueue*(591);  
 ArrayQueueModule.*enqueue*(592);  
 ArrayQueueModule.*enqueue*(595);  
 System.*out*.println("elements: " + Arrays.*toString*(ArrayQueueModule.*elements*));  
 System.*out*.println("element(): " + ArrayQueueModule.*element*());  
 System.*out*.println("elements after element(): " + Arrays.*toString*(ArrayQueueModule.*elements*));  
 System.*out*.println("dequeue(): " + ArrayQueueModule.*dequeue*());  
 System.*out*.println("elements after dequeue(): " + Arrays.*toString*(ArrayQueueModule.*elements*));  
 System.*out*.println("getSize(): " + ArrayQueueModule.*getSize*());  
 if (ArrayQueueModule.*isEmpty*()) {  
 System.*out*.println("Empty " + ArrayQueueModule.*getSize*());  
 } else {  
 System.*out*.println("Not empty " + ArrayQueueModule.*getSize*());  
 }  
 ArrayQueueModule.*clear*();  
 System.*out*.println("elements after clear(): " + Arrays.*toString*(ArrayQueueModule.*elements*));  
 if (ArrayQueueModule.*isEmpty*()) {  
 System.*out*.println("Empty " + ArrayQueueModule.*getSize*());  
 } else {  
 System.*out*.println("Not empty "+ ArrayQueueModule.*getSize*());  
 }  
 System.*out*.println("---------------------------------------------------------");  
  
 System.*out*.println("ArrayQueueADT Test");  
 ArrayQueueADT arrayQueueADT = new ArrayQueueADT();  
 ArrayQueueADT.*push*(arrayQueueADT, 1);  
 ArrayQueueADT.*push*(arrayQueueADT, 2);  
 ArrayQueueADT.*push*(arrayQueueADT, 3);  
 ArrayQueueADT.*push*(arrayQueueADT, 4);  
 System.*out*.println(arrayQueueADT);  
 System.*out*.println("element(): " + ArrayQueueADT.*element*(arrayQueueADT));  
 System.*out*.println("dequeue(): " + ArrayQueueADT.*dequeue*(arrayQueueADT));  
 System.*out*.println("dequeue(): " + ArrayQueueADT.*dequeue*(arrayQueueADT));  
 System.*out*.println("after dequeue(): " + arrayQueueADT);  
 if (ArrayQueueADT.*isEmpty*(arrayQueueADT)) {  
 System.*out*.println("Empty " + ArrayQueueADT.*getSize*(arrayQueueADT));  
 } else {  
 System.*out*.println("Not empty "+ ArrayQueueADT.*getSize*(arrayQueueADT));  
 }  
 ArrayQueueADT.*clear*(arrayQueueADT);  
 if (ArrayQueueADT.*isEmpty*(arrayQueueADT)) {  
 System.*out*.println("Empty " + ArrayQueueADT.*getSize*(arrayQueueADT));  
 } else {  
 System.*out*.println("Not empty "+ ArrayQueueADT.*getSize*(arrayQueueADT));  
 }  
 System.*out*.println("---------------------------------------------------------");  
  
 System.*out*.println("ArrayQueue Test");  
 ArrayQueue arrayQueue = new ArrayQueue();  
 arrayQueue.enqueue(11);  
 arrayQueue.enqueue(12);  
 arrayQueue.enqueue(13);  
 arrayQueue.enqueue(14);  
 System.*out*.println(arrayQueue);  
 System.*out*.println("element(): " + arrayQueue.element());  
 System.*out*.println("dequeue(): " + arrayQueue.dequeue());  
 System.*out*.println(arrayQueue);  
 if (arrayQueue.isEmpty()) {  
 System.*out*.println("Empty " + arrayQueue.getSize());  
 } else {  
 System.*out*.println("Not empty "+ arrayQueue.getSize());  
 }  
 arrayQueue.clear();  
 if (arrayQueue.isEmpty()) {  
 System.*out*.println("Empty " + arrayQueue.getSize());  
 } else {  
 System.*out*.println("Not empty "+ arrayQueue.getSize());  
 }  
 }  
}

## **Вывод**

Получены навыки по работе с очередями

GitHub

https://github.com/BarinovIvan/RuPractises/tree/master/src/Practise11