|  |
| --- |
|  |
| 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, №12**

Выполнил студент группы ИНБО-02-20 Зотов Е. А.

Принял Степанов П.В.

Практические работы выполнены «\_\_\_»\_\_\_\_\_\_\_2020г.

«\_\_\_\_\_\_\_\_\_\_\_\_\_\_» «\_\_\_»\_\_\_\_\_\_\_2020г.

Отметка о выполнении

**Москва – 2020 г.**

**Практическое занятие №11**

**Задание 1**

* Очередь на массиве
* Найдите инвариант структуры данных «очередь». 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**

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

**Код программы:**

**Класс AbstactQueue**

public abstract class AbstractQueue implements Queue {  
 protected int head, tail;  
 protected static final int *START\_CAPACITY* = 16;  
  
}

}

**Класс ArrayQueue**

public class ArrayQueue {  
 private ArrayQueueModule queue;  
  
 public ArrayQueue() {  
 this.queue = ArrayQueueModule.*getInstance*();  
 }  
  
 public Object dequeue() {  
 return queue.dequeue();  
 }  
  
 public Object element() {  
 return queue.element();  
 }  
  
  
 public void enqueue(Object o) {  
 queue.enqueue(o);  
 }  
  
  
 public boolean isEmpty() {  
 return queue.isEmpty();  
 }  
  
 public boolean clear() {  
 return queue.clear();  
 }  
}

**Класс ArrayQueueADT**

public class ArrayQueueADT {  
  
 private ArrayQueueModule queue;  
  
 public ArrayQueueADT(ArrayQueueModule queue){  
 this.queue = queue;  
 }  
  
 public Object dequeue() {  
 return queue.dequeue();  
 }  
  
 public Object element() {  
 return queue.element();  
 }  
  
 public void enqueue(Object o) {  
 queue.enqueue(o);  
 }  
  
 public boolean isEmpty() {  
 return queue.isEmpty();  
 }  
  
 public boolean clear() {  
 return queue.clear();  
 }  
}

**Класс ArrayQueueModule**

package ru.mirea.practice11.task1;  
  
import ru.mirea.practice11.task2.AbstractQueue;  
  
public class ArrayQueueModule extends AbstractQueue {  
 private Object[] q;  
  
 private static ArrayQueueModule *instance*;  
  
 private ArrayQueueModule(){  
 q = new Object[*START\_CAPACITY*];  
 head = tail = 0;  
 }  
  
 public static ArrayQueueModule getInstance(){  
 if(*instance* == null)  
 *instance* = new ArrayQueueModule();  
 return *instance*;  
 }  
  
 @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;  
 }  
  
 @Override  
 public boolean isEmpty() {  
 return head == tail;  
 }  
  
 @Override  
 public void enqueue(Object element){  
 q[tail++] = element;  
 if(tail == q.length) tail = 0;  
 if(tail == head) resize(q.length \* 2);  
 }  
  
 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;  
 }  
  
 @Override  
 public Object element(){  
 if(isEmpty()) throw new IndexOutOfBoundsException("Queue is empty");  
 return q[head];  
 }  
  
 @Override  
 public boolean clear(){  
 boolean r = !isEmpty();  
 head = tail = 0;  
 resize(*START\_CAPACITY*);  
 return r;  
 }  
  
}

}

**Класс LinkedQueue**

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;  
 }  
}

**Интерфейс Queue**

public interface Queue {  
 Object dequeue();  
 Object element();  
 void enqueue(Object o);  
 boolean isEmpty();  
 boolean clear();  
}

}

**Класс Main**

public class Main {  
 public static void main(String[] args) {  
 ArrayQueueModule aqm = ArrayQueueModule.*getInstance*();  
 aqm.enqueue("Test 1");  
 aqm.enqueue("Test 2");  
 aqm.enqueue("Test 3");  
 System.*out*.println(aqm.dequeue());  
  
 ArrayQueueADT adt = new ArrayQueueADT(aqm);  
 System.*out*.println(adt.dequeue());  
  
 ArrayQueue aq = new ArrayQueue();  
 System.*out*.println(aq.dequeue()); }  
}

**Практическое занятие №12**

**Задание**

Разработать класс Person, в котором имеется функция, возвращающая Фамилию И.О. Функция должна учитывать возможность отсутствия значений в полях Имя и Отчество. Программу оптимизировать с точки зрения быстродействия.

**Код программы:**

**Класс Person**

public class Person {  
 private String name, surname, patronymic;  
  
 public Person(String surname) {  
 this.surname = surname;  
 }  
  
 public Person(String name, String surname, String patronymic) {  
 this.name = name;  
 this.surname = surname;  
 this.patronymic = patronymic;  
 }  
 public String getFio(){  
 StringBuilder sb = new StringBuilder(surname);  
 if(name != null && ! name.equals(""))  
 sb.append(" ").append(name);  
 if(patronymic != null && ! patronymic.equals(""))  
 sb.append(" ").append(patronymic);  
 return sb.toString();  
 }  
  
 public static void main(String[] args) {  
 Person p1 = new Person("Иванов");  
 Person p2 = new Person("Николай", "Васильев", "");  
 Person p3 = new Person("Андрей", "Смирнов", "Сергеевич");  
  
 System.*out*.println(p1.getFio());  
 System.*out*.println(p2.getFio());  
 System.*out*.println(p3.getFio());  
 }  
}

**Тест**

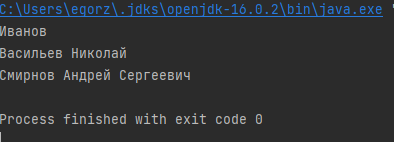


Рис.1- тест

**Выводы**

Я научился создавать очереди одинаковых по применению, но разных по своему строению, так же научился работать с классами String, StringBuffer. StringBuilder.