|  |
| --- |
|  |
| 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 Самойленко М. 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.

o Класс ArrayQueueModule должен реализовывать один экземпляр очереди с использованием переменных класса.

o Класс ArrayQueueADT должен реализовывать очередь в виде абстрактного типа данных (с явной передачей ссылки на экземпляр очереди).

o Класс ArrayQueue должен реализовывать очередь в виде класса (с неявной передачей ссылки на экземпляр очереди).

o 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.

o Инвариант, пред- и постусловия записываются в исходном коде в виде комментариев.

o Обратите внимание на инкапсуляцию данных и кода во всех трех реализациях.

● Напишите тесты реализованным классам.

**Задание 2**

4 Определите интерфейс очереди Queue и опишите его контракт.

5 Реализуйте класс LinkedQueue — очередь на связном списке.

6 Выделите общие части классов LinkedQueue и ArrayQueue в базовый класс AbstractQueue.

**Код программы:**

**Класс AbstactQueue**

public abstract class AbstactQueue implements Queue {  
 private int size = 0;  
  
 protected abstract boolean doEnqueue(Object element);  
  
 protected abstract Object doDequeue();  
  
 protected abstract void doPush(Object element);  
  
 public void enqueue(Object ad) {  
 if (doEnqueue(ad)) {  
 size++;  
 }  
 }  
  
 public Object element() {  
 assert size > 0;  
 Object ret = dequeue();  
 if (ret != null)  
 push(ret);  
 return ret;  
 }  
  
 public Object dequeue() {  
 assert size > 0;  
 Object ret= doDequeue();  
 size--;  
 return ret;  
 }  
  
 public int size() {  
 return size;  
 }  
  
 public void clear() {  
 while (size > 0) {  
 dequeue();  
 }  
 }  
  
 public boolean isEmpty() {  
  
 return size == 0;  
 }  
  
 public void push(Object element) {  
 doPush(element);  
 size++;  
 }  
}

**Класс ArrayQueue**

package ru.mirea.java.practice11.Queue;  
  
public class ArrayQueue extends AbstactQueue {  
 final int sizeQ = 5;  
 private int front = -1;  
 private int rear = -1;  
 private Object[] elements = new Object[sizeQ];  
  
 // PRE: None  
 // POST: front != rear  
 // rear - front < size  
 protected boolean doEnqueue(Object element) {  
 if (size() == elements.length) {  
 System.out.println("Очередь заполнена.");  
 return false;  
 }  
 else {  
 if (front == -1)  
 front = 0;  
 rear = (rear + 1) % sizeQ;  
 elements[rear] = element;  
 return true;  
 }  
 }  
 // PRE: size > 0  
 // POST: Object return = queue[head]  
 // newSize = size - 1  
 // queue[front+1]..queue[rear] - immutable  
 protected Object doDequeue() {  
 Object element;  
 if (isEmpty()) {  
 System.*out*.println("Очередь пуста.");  
 return null;  
 }  
 else{  
 element = elements[front];  
 if (front == rear){  
 front = -1;  
 rear = -1;  
 }  
 else  
 front = (front + 1) % sizeQ;  
 }  
 return element;  
 }  
  
 protected void doPush(Object element) {  
 if (front == 0)  
 front = 4;  
 else  
 front = front -1;  
 elements[front] = element;  
 }  
}

**Класс ArrayQueueADT**

package ru.mirea.java.practice11.Queue;  
// INV: FIFO (First in - first out)  
// size >= 0  
// queue[head]..queue[tail] - queue  
// -1 <= front, rear < 5  
public class ArrayQueueADT {  
 private final int sizeQ = 5;  
 private int front = -1;  
 private int rear = -1;  
 private int size = 0;  
 private Object[] elements = new Object[sizeQ];  
  
 // PRE: queue - not nullable  
 // POST: queue[tail] = element  
 // size' = size + 1  
 // queue[head]..queue[tail-1] - immutable  
 public static void enqueue(ArrayQueueADT queue, Object element) {  
 if (queue.size == queue.elements.length)  
 System.out.println("Очередь заполнена.");  
 else {  
 if (queue.front == -1)  
 queue.front = 0;  
 queue.rear = (queue.rear + 1) % queue.sizeQ;  
 queue.elements[queue.rear] = element;  
 queue.size++;  
 }  
 }  
  
 // PRE: size > 0  
 // queue - not nullable  
 // POST: R = queue[head]  
 // queue - immutable  
 public static Object element(ArrayQueueADT queue) {  
 if ( isEmpty(queue)) {  
 System.out.println("Очередь пуста.");  
 return null;  
 }  
 else {  
 return queue.elements[queue.front];  
 }  
 }  
  
 // PRE: size > 0  
 // queue - not nullable  
 // POST: R = queue[head]  
 // queue[head] = queue[head+1]  
 // size' = size - 1  
 // queue[head+1]..queue[tail] - immutable  
 public static Object dequeue(ArrayQueueADT queue) {  
 Object element;  
 if (isEmpty(queue)) {  
 System.out.println("Очередь пуста.");  
 return null;  
 }  
 else{  
 element = queue.elements[queue.front];  
 if (queue.front == queue.rear){  
 queue.front = -1;  
 queue.rear = -1;  
 }  
 else  
 queue.front = (queue.front + 1) % queue.sizeQ;  
 }  
 queue.size--;  
 return element;  
 }  
  
 // PRE: queue - not nullable  
 // POST: R = size  
 // queue - immutable  
 public static int size(ArrayQueueADT queue) {  
 return queue.size;  
 }  
  
 // PRE: queue - not nullable  
 // POST: queue - immutable  
 // R = (size == 0)  
 public static boolean isEmpty(ArrayQueueADT queue) {  
 return queue.size == 0;  
 }  
  
 // PRE: queue - not nullable  
 // POST: size == 0  
 public static void clear(ArrayQueueADT queue) {  
 while (queue.size > 0) {  
 dequeue(queue);  
 }  
 }  
}

**Класс ArrayQueueModule**

package ru.mirea.java.practice11.Queue;  
// Inv: First in - first out  
// size >= 0  
// -1 <= front, rear < 5  
public class ArrayQueueModule {  
 private static final int *sizeQ* = 5;  
 private static int *front* = -1;  
 private static int *rear* = -1;  
 private static int *size* = 0;  
 private static Object[] *elements* = new Object[*sizeQ*];  
  
 // PRE: None  
 // POST: front != rear  
 // rear - front < size  
 public static void enqueue(Object element) {  
 if (*size* == *elements*.length)  
 System.*out*.println("Очередь заполнена.");  
 else {  
 if (*front* == -1)  
 *front* = 0;  
 *rear* = (*rear* + 1) % *sizeQ*;  
 *elements*[*rear*] = element;  
 *size*++;  
 }  
 }  
  
 // PRE: size > 0  
 // POST: Object return = queue[head]  
 // queue - immutable  
 public static Object element() {  
 if (*isEmpty*()) {  
 System.*out*.println("Очередь пуста.");  
 return null;  
 }  
 else {  
 return *elements*[*front*];  
 }  
  
 }  
  
 // PRE: size > 0  
 // POST: Object return = queue[head]  
 // newSize = size - 1  
 // queue[front+1]..queue[rear] - immutable  
 public static Object dequeue() {  
 Object element;  
 if (*isEmpty*()) {  
 System.*out*.println("Очередь пуста.");  
 return null;  
 }  
 else{  
 element = *elements*[*front*];  
 if (*front* == *rear*){  
 *front* = -1;  
 *rear* = -1;  
 }  
 else  
 *front* = (*front* + 1) % *sizeQ*;  
 }  
 *size*--;  
 return element;  
 }  
  
 // PRE: None  
 // POST: Object return = size  
 // queue - immutable  
 public static int size() {  
 return *size*;  
 }  
  
 // PRE: None  
 // POST: queue - immutable  
 // boolean return = (size == 0) false/true  
 public static boolean isEmpty() {  
 return *size* == 0;  
 }  
  
 // PRE: None  
 // POST: size == 0  
 public static void clear() {  
 while (*size* > 0) {  
 *dequeue*();  
 }  
 }  
}

**Класс Knot**

package ru.mirea.java.practice11.Queue;  
  
public class Knot {  
 Object key;  
 Knot next;  
 Knot prev;  
  
 public Knot(Object key) {  
 this.key = key;  
 this.next = null;  
 this.prev = null;  
 }  
}

**Класс LinkedQueue**

package ru.mirea.java.practice11.Queue;  
  
public class LinkedQueue extends AbstactQueue {  
 private Knot head = null;  
 private Knot tail = null;  
  
 @Override  
 protected boolean doEnqueue(Object element) {  
 Knot kn = new Knot(element);  
 if (size() == 0){  
 head = kn;  
 tail = kn;  
 }  
 else {  
 tail.next = kn;  
 kn.prev = tail;  
 tail = kn;  
 }  
 return true;  
 }  
  
 @Override  
 protected Object doDequeue() {  
 Object ret = head.key;  
 if (size() != 1){  
 head = head.next;  
 head.prev = null;  
 } else {  
 head = null;  
 tail = null;  
 }  
 return ret;  
 }  
  
 @Override  
 protected void doPush(Object element) {  
 Knot kn = new Knot(element);  
 if (size() == 0){  
 head = kn;  
 tail = kn;  
 }  
 else {  
 kn.next = head;  
 head.prev = kn;  
 head = kn;  
 }  
 }  
}

**Интерфейс Queue**

package ru.mirea.java.practice11.Queue;  
  
// INV: FIFO(First in - First out)  
// size >=0  
public interface Queue {  
 // Pre: None  
 // Post: queue[tail] = newElement  
 // newSize = lastsize + 1  
 void enqueue(Object ad);  
 // PRE: size > 0  
 // POST: returnEleemnt = queue[head]  
 // queue - immutable  
 Object element();  
 // PRE: size > 0  
 // POST: newSize = lastSize - 1  
 // newQueue[head] = lastQueue[head +1]  
 Object dequeue();  
 // PRE: None  
 // Post: returnInt = size  
 int size();  
 // PRE: None  
 // POST: returnInt = false/true(size != 0 : size == 0)  
 boolean isEmpty();  
 // PRE: None  
 // POST: size = 0  
 void clear();  
 // PRE: None  
 // POST: queue[head] = elementWhoNowCalledElement  
 // newSize = lastSize + 1  
 void push(Object element);  
}

**Класс Main**

package ru.mirea.java.practice11;  
  
import ru.mirea.java.practice11.Queue.\*;  
  
public class Main {  
 public static void main(String[] args) {  
 ArrayQueueModule q1= new ArrayQueueModule();  
 ArrayQueueADT q2 = new ArrayQueueADT();  
 ArrayQueue q3 = new ArrayQueue();  
 LinkedQueue q4 = new LinkedQueue();  
 for (int i = 0; i < 30; i = i + 6) {  
 q1.*enqueue*(i);  
 ArrayQueueADT.*enqueue*(q2,i);  
 q3.enqueue(i);  
 q4.enqueue(i);  
 }  
 System.*out*.print('\n'+"Module:" + '\n');  
 for (int i = 0; i < 30; i = i + 6){  
 System.*out*.println(q1.*dequeue*());  
 }  
 System.*out*.print('\n'+"ADT:" + '\n');  
 for (int i = 0; i < 30; i = i + 6){  
 System.*out*.println(ArrayQueueADT.*dequeue*(q2));  
 }  
 System.*out*.print('\n'+"Queue:" + '\n');  
 for (int i = 0; i < 30; i = i + 6){  
 System.*out*.println(q3.dequeue());  
 }  
 System.*out*.print('\n'+"linked:" + '\n');  
 for (int i = 0; i < 30; i = i + 6){  
 System.*out*.println(q4.dequeue());  
 }  
 if (q1.*isEmpty*() && ArrayQueueADT.*isEmpty*(q2) && q3.isEmpty() && q4.isEmpty())  
 System.*out*.println("All queue is empty");  
 System.*out*.println("Function clear");  
 for (int i = 0; i < 30; i = i + 6) {  
 q1.*enqueue*(i);  
 ArrayQueueADT.*enqueue*(q2,i);  
 q3.enqueue(i);  
 q4.enqueue(i);  
 }  
 q1.*clear*();  
 ArrayQueueADT.*clear*(q2);  
 q3.clear();  
 q4.clear();  
 if (q1.*isEmpty*() && ArrayQueueADT.*isEmpty*(q2) && q3.isEmpty() && q4.isEmpty())  
 System.*out*.println("All queue is empty");  
 }  
}

**Тесты**

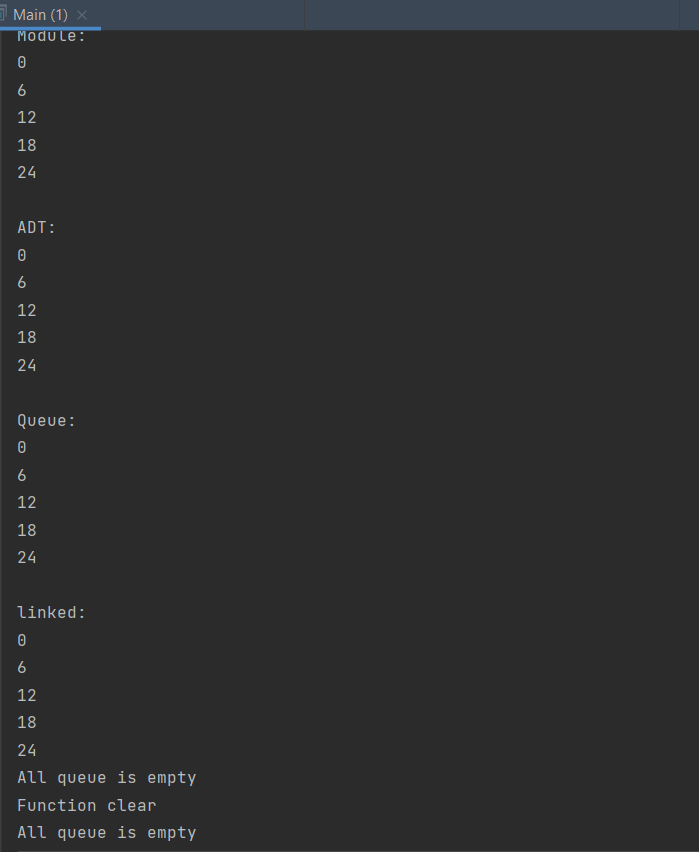


Рис.1 – тест

**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**

package ru.mirea.java.practice12;  
  
public class Person {  
 private String surname;  
 private String name;  
 private String middleName;  
  
 public Person(String surname, String name, String middleName) {  
 this.surname = surname;  
 this.name = name;  
 this.middleName = middleName;  
 }  
 Person(String surname){  
 this(surname,null,null);  
 }  
  
 String FIO() {  
 if (name != null && middleName != null) {  
 return surname + " " + name + " " + middleName;  
 } else  
 return surname;  
 }  
}

**Класс Main**

package ru.mirea.java.practice12;  
  
public class Main {  
 public static void main(String[] args) {  
 Person p1 = new Person("Samoilenko", "Maxim", "Andreevich");  
 Person p2 = new Person("Utkin");  
 System.*out*.println(p1.FIO());  
 System.*out*.println(p2.FIO());  
 }  
}

**Тест**

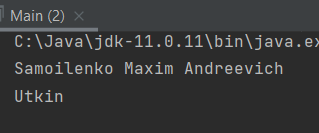


Рис.2- тест

**Выводы**

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