|  |
| --- |
|  |
| 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 ПРАКТИЧЕСКИМ ЗАНЯТИЯМ №11, №12**

Выполнил студент группы ИНБО-02-20 Чернов Я. А.

Принял Степанов П.В.

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

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

Отметка о выполнении

**Москва 2021 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**

* 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 Main**

**package** ru.mirea.Practice11.Task12;  
  
**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(**"Queues 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(**"Queues empty"**);  
 }  
}

**Класс ArrayQueue**

**package** ru.mirea.Practice11.Task12;  
  
  
**public class** ArrayQueue **extends** AbstractQueue {  
 **final int sizeQ** = 5;  
 **private int front** = -1;  
 **private int rear** = -1;  
 **private** Object[] **elements** = **new** Object[**sizeQ**];**protected boolean** doEnqueue(Object element) {  
 **if** (size() == **elements**.**length**) {  
 System.***out***.println(**"Queue full"**);  
 **return false**;  
 }  
 **else** {  
 **if** (**front** == -1)  
 **front** = 0;  
 **rear** = (**rear** + 1) % **sizeQ**;  
 **elements**[**rear**] = element;  
 **return true**;  
 }  
 }  
**protected** Object doDequeue() {  
 Object element;  
 **if** (isEmpty()) {  
 System.***out***.println(**"Queue empty"**);  
 **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.Practice11.Task12;  
  
**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**];  
  
 **public static void** enqueue(ArrayQueueADT queue, Object element) {  
 **if** (queue.**size** == queue.**elements**.**length**)  
 System.***out***.println(**"Queue full"**);  
 **else** {  
 **if** (queue.**front** == -1)  
 queue.**front** = 0;  
 queue.**rear** = (queue.**rear** + 1) % queue.**sizeQ**;  
 queue.**elements**[queue.**rear**] = element;  
 queue.**size**++;  
 }  
 }  
  
 **public static** Object element(ArrayQueueADT queue) {  
 **if** ( *isEmpty*(queue)) {  
 System.***out***.println(**"Queue empty"**);  
 **return null**;  
 }  
 **else** {  
 **return** queue.**elements**[queue.**front**];  
 }  
 }  
  
 **public static** Object dequeue(ArrayQueueADT queue) {  
 Object element;  
 **if** (*isEmpty*(queue)) {  
 System.***out***.println(**"Queue empty"**);  
 **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;  
 }  
  
 **public static int** size(ArrayQueueADT queue) {  
 **return** queue.**size**;  
 }  
  
 **public static boolean** isEmpty(ArrayQueueADT queue) {  
 **return** queue.**size** == 0;  
 }  
  
 **public static void** clear(ArrayQueueADT queue) {  
 **while** (queue.**size** > 0) {  
 *dequeue*(queue);  
 }  
 }  
}

**Класс ArrayQueueModule**

**package** ru.mirea.Practice11.Task12;  
  
**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***];  
  
 **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*++;  
 }  
 }  
  
 **public static** Object element() {  
 **if** (*isEmpty*()) {  
 System.***out***.println(**"Queue empty"**);  
 **return null**;  
 }  
 **else** {  
 **return** *elements*[*front*];  
 }  
  
 }  
  
 **public static** Object dequeue() {  
 Object element;  
 **if** (*isEmpty*()) {  
 System.***out***.println(**"Queue empty"**);  
 **return null**;  
 }  
 **else**{  
 element = *elements*[*front*];  
 **if** (*front* == *rear*){  
 *front* = -1;  
 *rear* = -1;  
 }  
 **else** *front* = (*front* + 1) % ***sizeQ***;  
 }  
 *size*--;  
 **return** element;  
 }  
  
 **public static int** size() {  
 **return** *size*;  
 }  
   
 **public static boolean** isEmpty() {  
 **return** *size* == 0;  
 }  
   
 **public static void** clear() {  
 **while** (*size* > 0) {  
 *dequeue*();  
 }  
 }  
}

**Класс LinkedQueue**

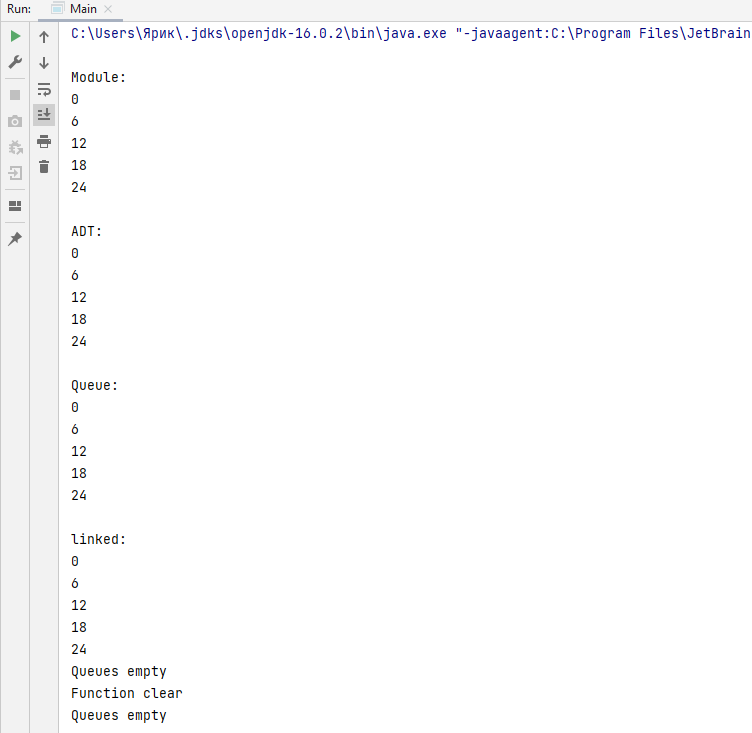
**package** ru.mirea.Practice11.Task12;  
  
**public class** LinkedQueue **extends** AbstractQueue {  
 **private** Node **head** = **null**;  
 **private** Node **tail** = **null**;  
  
 @Override  
 **protected boolean** doEnqueue(Object element) {  
 Node kn = **new** Node(element);  
 **if** (size() == 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) {  
 Node kn = **new** Node(element);  
 **if** (size() == 0){  
 **head** = kn;  
 **tail** = kn;  
 }  
 **else** {  
 kn.**next** = **head**;  
 **head**.**prev** = kn;  
 **head** = kn;  
 }  
 }  
}

**Интерфейс Queue**

**package** ru.mirea.Practice11.Task12;  
  
**public interface** Queue {  
 **void** enqueue(Object ad);  
 Object element();  
 Object dequeue();  
 **int** size();  
 **boolean** isEmpty();  
 **void** clear();  
 **void** push(Object element);  
}

**Класс Node**

**package** ru.mirea.Practice11.Task12;  
  
**public class** Node {  
 Object **key**;  
 Node **next**;  
 Node **prev**;  
  
 **public** Node(Object key) {  
 **this**.**key** = key;  
 **this**.**next** = **null**;  
 **this**.**prev** = **null**;  
 }  
}

**Тестирование** ****

**Рисунок 1 – Тестирование**

**Практическое занятие №12**

**Задание**

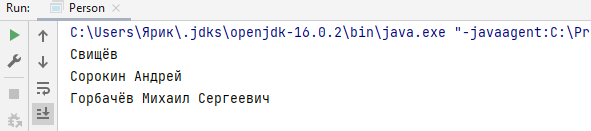
Разработать класс Person, в котором имеется функция, возвращающая Фамилию И.О. Функция должна учитывать возможность отсутствия значений в полях Имя и Отчество. Программу оптимизировать с точки зрения быстродействия.

**Код программы**

**Класс Person**

**package** ru.mirea.Practice1215.Task1;  
  
**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(**"Свищёв"**);  
 Person p2 = **new** Person(**"Андрей"**, **"Сорокин"**, **""**);  
 Person p3 = **new** Person(**"Михаил"**, **"Горбачёв"**, **"Сергеевич"**);  
  
 System.***out***.println(p1.getFio());  
 System.***out***.println(p2.getFio());  
 System.***out***.println(p3.getFio());  
 }  
}

**Тестирование**



**Рисунок 2- Тестирование**

**Выводы**

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

Git: <https://github.com/YaroslavChernov0220/PracticeMIREA>