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 Деревянкин Н.А.

Принял Степанов П.В.

Практическая работа выполнена «___»_____2021 г.

«_____» «___»_____2021 г.

Отметка о выполнении

Москва – 2021 г.

СОДЕРЖАНИЕ

Цель работы

Цель данной практической работы – Научиться реализовывать очереди.

Задание

Задание 1 – Очередь на массиве

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.

Задание 2 – Очереди

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.

Репозиторий

Ссылка: https://github.com/neluckoff/mirea-javalessons/tree/master/src/ru/luckoff/mirea/practice_11

Выполнение работы

Решение задания 1

Первым делом мною были создан класс ArrayQueueModule в который я добавил методы, поставленные в задании. Я посчитал, что названия методов для меня слегка непривычны, поэтому решил использовать привычные для всех стандартные названия этих функций, такие как: pop, push и peek.

Реализацию всех классов можно будет посмотреть ниже, в коде выполненной работы.

Сейчас ниже мною будет представлена UML Диаграмма полученная в процессе выполнения работы.

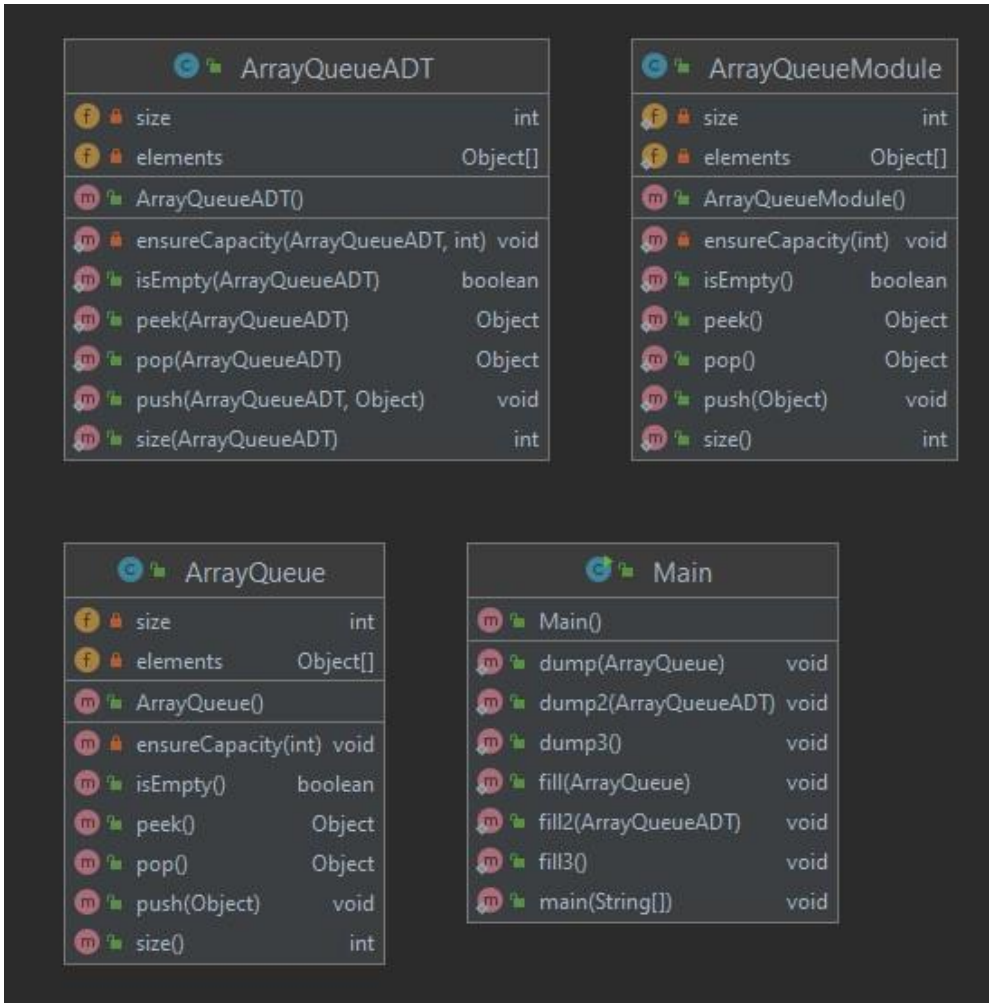


Рисунок 1 – UML Диаграмма первого задания

Решение задания 2

В этом задании я скопировал получившийся класс ArrayQueue и создал новый – LinkedQueue. Так как в поставленной задаче нас просят выделить общие части

этих классов в абстрактный класс AbstractQueue, я решил сделать это первым же делом.

Благодаря этому, класс LinkedQueue я заполнил достаточно быстро, просто переопределяя уже готовые методы.

Реализацию всех классов можно будет посмотреть ниже, в коде выполненной работы.

Сейчас ниже мною будет представлена UML Диаграмма полученная в процессе выполнения работы.

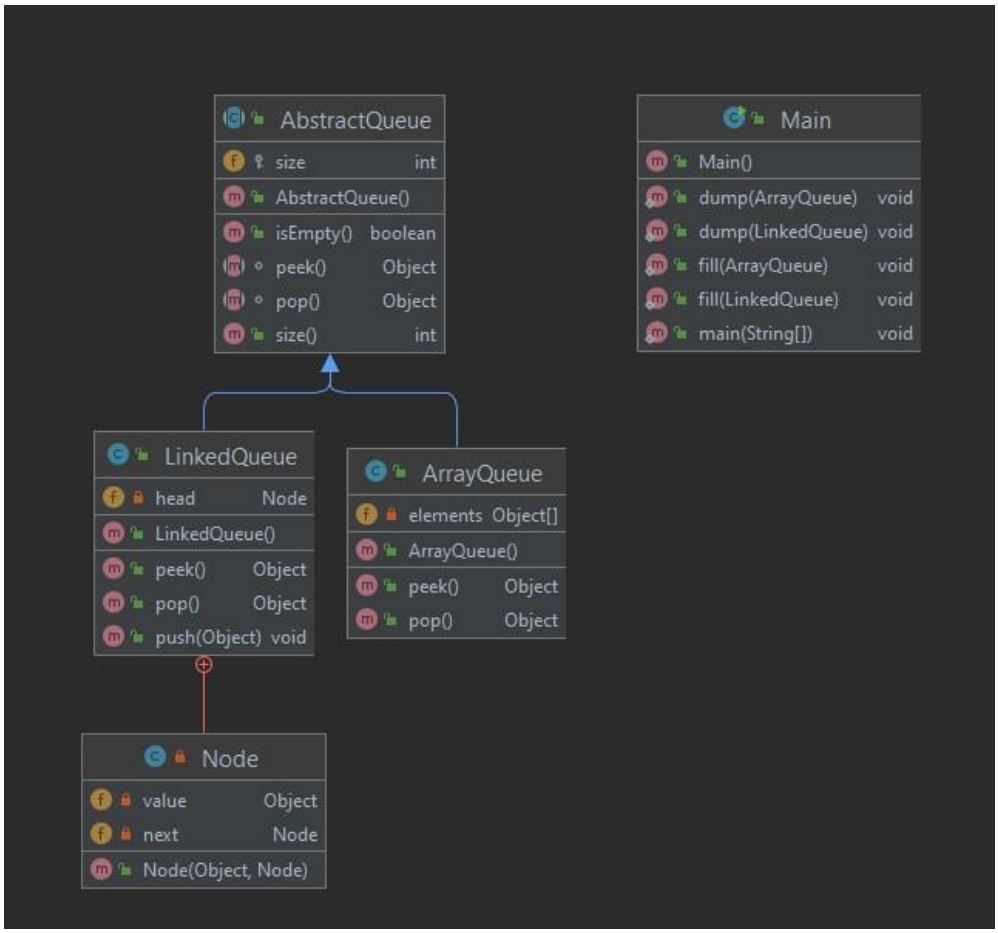


Рисунок 2 – UML Диаграмма второго задания

Код выполненной работы

Здесь в нескольких скриншотах можно увидеть как выглядит код выполненного задания и результат его работы.

Полученный код для задания №1

```
public class ArrayQueue {
    private int size;
    private Object[] elements = new Object[5];

    public void push(ArrayQueue this, Object element) {
        assert element != null;
        this.ensureCapacity(size + 1);
        this.elements[this.size++] = element;
    }

    private void ensureCapacity(int capacity) {
        if (capacity > this.elements.length) {
            this.elements = Arrays.copyOf(this.elements, newLength: 2 * capacity);
        }
    }

    public Object peek() {
        assert size > 0;
        return elements[size - 1];
    }

    public Object pop() {
        assert size > 0;

        Object value = peek();
        elements[--size] = 0;
        return value;
    }

    public int size() { return size; }

    public boolean isEmpty() { return size == 0; }
}
```

Рисунок 3 – Класс ArrayQueue

```
//TESTING - ArrayQueue
public static void fill(ArrayQueue stack) {
    for (int i = 0; i < 10; i++) {
        stack.push(i);
    }
}

public static void dump(ArrayQueue stack) {
    while (!stack.isEmpty()) {
        System.out.println(stack.size() + " " + stack.peek() + " " + stack.pop());
    }
}
```

Рисунок 4 – Тест класса ArrayQueue

```java
public class ArrayQueueADT {
    private int size;
    private Object[] elements = new Object[10];

    public static void push(ArrayQueueADT stack, Object element) {
        assert element != null;
        ensureCapacity(stack, capacity: stack.size + 1);
        stack.elements[stack.size++] = element;
    }

    private static void ensureCapacity(ArrayQueueADT stack, int capacity) {
        if (capacity > stack.elements.length) {
            stack.elements = Arrays.copyOf(stack.elements, newLength: 2 * capacity);
        }
    }

    public static Object pop(ArrayQueueADT stack) {
        assert stack.size > 0;

        return stack.elements[--stack.size];
    }

    public static Object peek(ArrayQueueADT stack) {
        assert stack.size > 0;

        return stack.elements[stack.size - 1];
    }
    public static int size(ArrayQueueADT stack) { return stack.size; }

    public static boolean isEmpty(ArrayQueueADT stack) { return stack.size == 0; }

}
```

Рисунок 5 – Класс ArrayQueueADT

```java
//TESTING - ArrayQueueADT
public static void fill2(ArrayQueueADT stack) {
    for (int i = 0; i < 10; i++) {
        ArrayQueueADT.push(stack, i);
    }
}

public static void dump2(ArrayQueueADT stack) {
    while (!ArrayQueueADT.isEmpty(stack)) {
        System.out.println(ArrayQueueADT.size(stack) + " " +
                ArrayQueueADT.peek(stack) + " " + ArrayQueueADT.pop(stack));
    }
}
```

Рисунок 6 – Тест класса ArrayQueueADT

```java
public class ArrayQueueModule {
    private static int size;
    private static Object[] elements = new Object[5];

    public static void push(Object element) {
        assert element != null;
        ensureCapacity(size + 1);
        elements[size++] = element;
    }

    private static void ensureCapacity(int capacity) {
        if (capacity > elements.length) {
            elements = Arrays.copyOf(elements, newLength: 2 * capacity);
        }
    }

    public static Object pop() {
        assert size > 0;
        return elements[--size];
    }

    public static Object peek() {
        assert size > 0;
        return elements[size - 1];
    }

    public static int size() { return size; }

    public static boolean isEmpty() { return size == 0; }

}
```

Рисунок 7 – Класс ArrayQueueModule

```java
//TESTING - ArrayQueueModule
public static void fill3() {
    for (int i = 0; i < 10; i++) {
        ArrayQueueModule.push(i);
    }
}

public static void dump3() {
    while (!ArrayQueueModule.isEmpty()) {
        System.out.println(ArrayQueueModule.size() + " "
                + ArrayQueueModule.peek() + " " + ArrayQueueModule.pop());
    }
}
```

Рисунок 8 – Тест класса ArrayQueueModule

```java
public static void main(String[] args) {
    System.out.println("TESTING - ArrayQueue");
    ArrayQueue stack = new ArrayQueue();
    fill(stack);
    dump(stack);

    System.out.println("TESTING - ArrayQueueADT");
    ArrayQueueADT stack2 = new ArrayQueueADT();
    fill2(stack2);
    dump2(stack2);

    System.out.println("TESTING - ArrayQueueModule");
    fill3();
    dump3();
}
```

Рисунок 9 – Метод main

```
TESTING - ArrayQueue
10 9 9
9 8 8
8 7 7
7 6 6
6 5 5
5 4 4
4 3 3
3 2 2
2 1 1
1 0 0
```

Рисунок 10 – Тест ArrayQueue

```
TESTING - ArrayQueueADT
10 9 9
9 8 8
8 7 7
7 6 6
6 5 5
5 4 4
4 3 3
3 2 2
2 1 1
1 0 0
```

Рисунок 11 – Тест ArrayQueueADT

```
TESTING - ArrayQueueModule
10 9 9
9 8 8
8 7 7
7 6 6
6 5 5
5 4 4
4 3 3
3 2 2
2 1 1
1 0 0
```

Рисунок 12 – Тест ArrayQueueModule

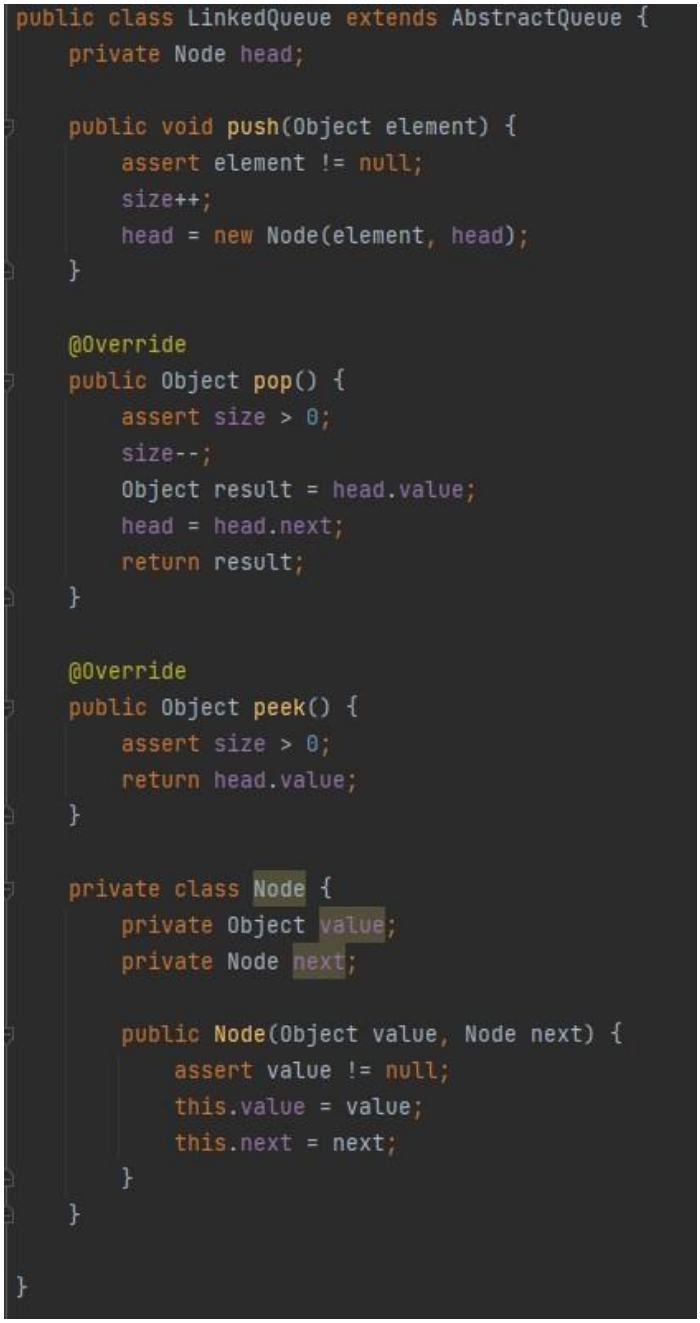
Полученный код для задания №2

```
public abstract class AbstractQueue {
    protected int size;

    public int size() { return size; }

    public boolean isEmpty() { return size == 0; }

    abstract Object pop();
    abstract Object peek();
}
```

Рисунок 13 – Абстрактный Класс AbstactQueue

```java
public class ArrayQueue extends AbstractQueue {
    private Object[] elements = new Object[5];

    public void push(ArrayQueue this, Object element) {
        assert element != null;
        this.ensureCapacity(size + 1);
        this.elements[this.size++] = element;
    }

    private void ensureCapacity(int capacity) {
        if (capacity > this.elements.length) {
            this.elements = Arrays.copyOf(this.elements, newLength: 2 * capacity);
        }
    }

    @Override
    public Object peek() {
        assert size > 0;
        return elements[size - 1];
    }

    @Override
    public Object pop() {
        assert size > 0;

        Object value = peek();
        elements[--size] = 0;
        return value;
    }
}
```

Рисунок 14 – Класс ArrayQueue второго задания



```
public class LinkedQueue extends AbstractQueue {
    private Node head;

    public void push(Object element) {
        assert element != null;
        size++;
        head = new Node(element, head);
    }

    @Override
    public Object pop() {
        assert size > 0;
        size--;
        Object result = head.value;
        head = head.next;
        return result;
    }

    @Override
    public Object peek() {
        assert size > 0;
        return head.value;
    }

    private class Node {
        private Object value;
        private Node next;

        public Node(Object value, Node next) {
            assert value != null;
            this.value = value;
            this.next = next;
        }
    }

}
```

Рисунок 15 – Класс LinkedQueue

```
//TESTING - ArrayQueue
public static void fill(ArrayQueue stack) {
    for (int i = 0; i < 10; i++) {
        stack.push(i);
    }
}

public static void dump(ArrayQueue stack) {
    while (!stack.isEmpty()) {
        System.out.println(stack.size() + " " + stack.peek() + " " + stack.pop());
    }
}
```

Рисунок 16 – Тест класса ArrayQueue второго задания

```
//TESTING - LinkedQueue
public static void fill1(LinkedQueue stack) {
    for (int i = 0; i < 10; i++) {
        stack.push(i);
    }
}

public static void dump1(LinkedQueue stack) {
    while (!stack.isEmpty()) {
        System.out.println(stack.size() + " " + stack.peek() + " " + stack.pop());
    }
}
```

Рисунок 17 – Тест класса LinkedQueue

```
public static void main(String[] args) {
    System.out.println("TESTING - ArrayQueue with AbstractQueue");
    ArrayQueue stack = new ArrayQueue();
    fill(stack);
    dump(stack);

    System.out.println("TESTING - LinkedQueue with AbstractQueue");
    LinkedQueue stack2 = new LinkedQueue();
    fill1(stack2);
    dump1(stack2);
}
```

Рисунок 18 – Метод main второго задания

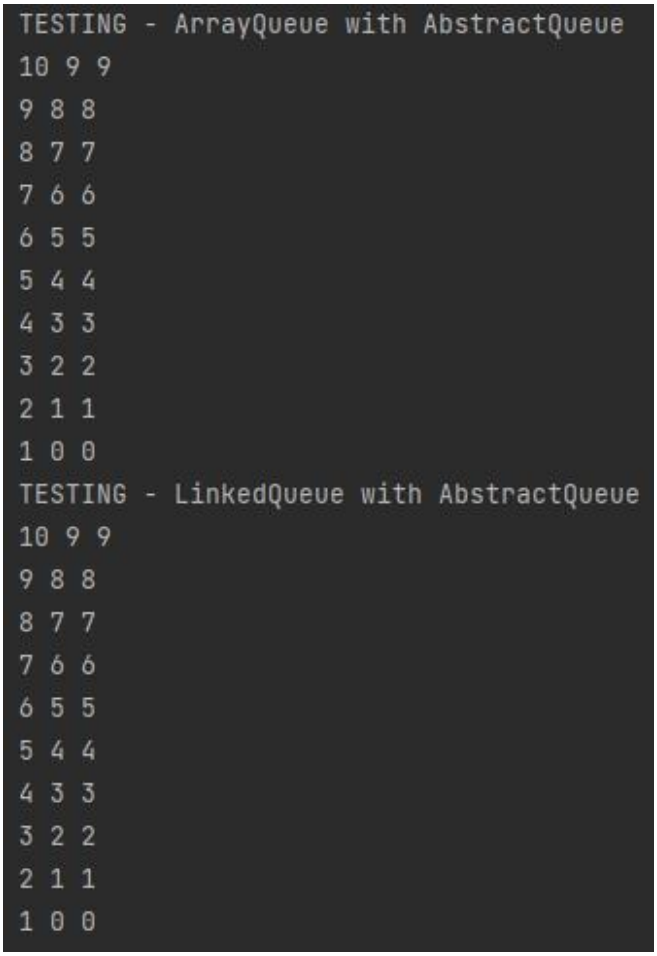


Рисунок 19 – Результат запуска второго задания

Вывод

В результате выполнения данной практической работы я научился реализовывать очереди как на массиве, так и на связном списке, а также применять их на практике в языке программирования Java.

GITHUB - https://github.com/dronikosha/JavaPractice