|  |
| --- |
|  |
| 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 ЗАДАНИЮ №8**

Выполнил студент группы ИНБО-02-20 Веселкин.И.А

Принял Степанов П.В.

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

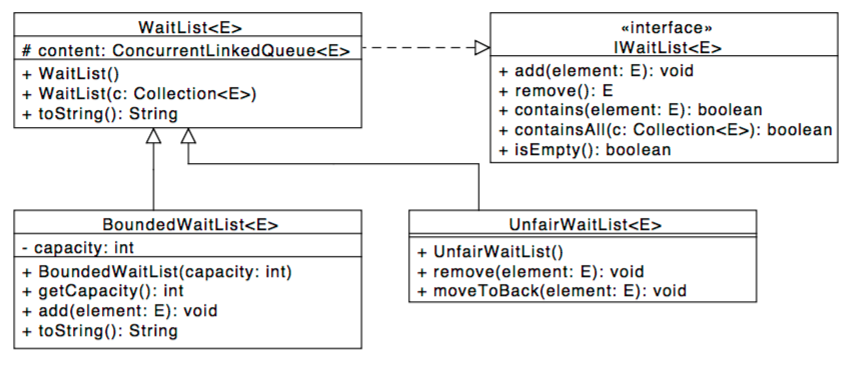
«\_\_\_\_\_\_\_\_\_\_\_\_\_\_» «\_\_\_»\_\_\_\_\_\_\_2021г.

Отметка о выполнении

**Москва – 2021 г.**

## **Задание**

1. Исследуйте UML диаграмму классов на рисунке 1 и понаблюдайте, как она выражает то, что мы говорили выше в словах. Убедитесь, что вы понимаете все аспекты диаграммы.
2. Расширить и модифицировать исходный код WaitList, как необходимо, чтобы полностью реализовать всю схему UML. Включить комментарии Javadoc. Обратите внимание на переключение ролей после реализации каждого интерфейса / класса!
3. Изучение работу метода main(), которая использует ваши новые классы и интерфейс.



1. **Ход Работы**

Реализация класса BoundedWaitList:

import java.util.Collection;  
import java.util.Iterator;  
import java.util.Scanner;  
  
public class BoundedWaitList<E> extends WaitList<E>{  
 private int capacity;  
 public BoundedWaitList(int capacity){  
 this.capacity = capacity;  
 }  
 public BoundedWaitList(Collection<E> c){  
 System.*out*.print("Размер списка: ");  
 Scanner in = new Scanner(System.*in*);  
 capacity = in.nextInt();  
 Iterator<E> it = c.iterator();  
 while (it.hasNext() && content.size()<capacity)  
 content.add(it.next());  
 }  
 public int getCapacity() {  
 return capacity;  
 }  
 public void add(E element){  
 if (content.size() < capacity)  
 content.add(element);  
 else  
 System.*out*.println("Список заполнен");  
 }  
  
 @Override  
 public String toString() {  
 return "BoundedWaitList{" +  
 "capacity=" + capacity +  
 "} " + content;  
 }  
}

Реализация интерфейса IWaitList:

import java.util.Collection;  
  
public interface IWaitList<E> {  
 public void add(E element);  
 public E remove();  
 public boolean contains(E element);  
 public boolean containsAll(Collection<E> c);  
 public boolean isEmpty();  
}

Реализация класса UnfairWaitList:

import java.util.Iterator;  
  
public class UnfairWaitList<E> extends WaitList<E>{  
 public UnfairWaitList() {  
 }  
 public void remove(E element) {  
 Iterator<E> it = content.iterator();  
 while (it.hasNext())  
 if (it.next() == element)  
 it.remove();  
 }  
 public void moveToBack(E element) {  
 Iterator<E> it = content.iterator();  
 while (it.hasNext()) {  
 E a = it.next();  
 if (a == element){  
 it.remove();  
 content.add(a);  
 break;  
 }  
 }  
  
 }  
}

Реализация класса WaitList:

import java.util.Collection;  
import java.util.Iterator;  
import java.util.concurrent.ConcurrentLinkedQueue;  
  
public class WaitList<E> implements IWaitList<E>{  
 protected ConcurrentLinkedQueue<E> content = new ConcurrentLinkedQueue<E>();  
 @Override  
 public void add(E element) {  
 content.add(element);  
 }  
  
 @Override  
 public E remove() {  
 return content.poll();  
 }  
  
 @Override  
 public boolean contains(E element) {  
 return content.contains(element);  
 }  
  
 @Override  
 public boolean containsAll(Collection<E> c) {  
 Iterator<E> it = c.iterator();  
 while (it.hasNext())  
 if (!content.contains(it.next()))  
 return false;  
 return true;  
 }  
  
 @Override  
 public boolean isEmpty() {  
 return content.isEmpty();  
 }  
 public WaitList() {  
 }  
 public WaitList(Collection<E> c) {  
 Iterator<E> it = c.iterator();  
 while (it.hasNext())  
 content.add(it.next());  
 }  
  
 @Override  
 public String toString() {  
 return "WaitList{" +  
 "content=" + content +  
 '}';  
 }  
}

}

Реализация тестового класса MainPract8:

import java.util.ArrayList;  
import java.util.Collection;  
  
public class MainPract8 {  
 public static void main(String[] args) {  
 ArrayList<Integer> arr = new ArrayList<Integer>();  
 arr.add(2);  
 arr.add(4);  
 arr.add(8);  
 WaitList<Integer> o1 = new WaitList<Integer>(arr);  
 o1.add(2);  
 System.*out*.println(o1.toString());  
 if (!o1.isEmpty())  
 System.*out*.println("remove: " + o1.remove());  
 System.*out*.println("contains: " + o1.contains(33));  
 System.*out*.println("containsAll: " + o1.containsAll(arr));  
 System.*out*.println("isEmpty: " + o1.isEmpty());  
 System.*out*.println(o1.toString());  
 System.*out*.println("");  
 BoundedWaitList<Integer> o2 = new BoundedWaitList<Integer>(arr);  
 o2.add(2);  
 System.*out*.println(o2.toString());  
 if (!o1.isEmpty())  
 System.*out*.println("remove: " + o2.remove());  
 System.*out*.println("contains: " + o2.contains(50));  
 System.*out*.println("containsAll: " + o2.containsAll(arr));  
 System.*out*.println("isEmpty: " + o2.isEmpty());  
 System.*out*.println(o2.toString());  
 System.*out*.println("");  
 UnfairWaitList<Integer> o3 = new UnfairWaitList<Integer>();  
 o3.add(1);  
 o3.add(2);  
 o3.add(3);  
 o3.add(4);  
 o3.add(5);  
 o3.add(10000);  
 o3.add(20000);  
 System.*out*.println(o3.toString());  
 if (!o1.isEmpty())  
 System.*out*.println("remove: " + o3.remove());  
 System.*out*.println("contains: " + o3.contains(100));  
 System.*out*.println("containsAll: " + o3.containsAll(arr));  
 System.*out*.println("isEmpty: " + o3.isEmpty());  
 System.*out*.println(o3.toString());  
 o3.remove(2);  
 System.*out*.println("Remove - 2 " + o3.toString());  
 o3.moveToBack(3);  
 System.*out*.println("moveToBack - 5 " + o3.toString());  
 }  
}

## **Вывод**

В ходе выполнения работы я изучил различные виды списков ожидания.

C полным кодом программы можно ознакомится по ссылке: https://github.com/gost214/Practika-INBO-02-20-Veselkin/tree/main/Pract8