|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |

Отчет по выполнению практической работы 8

**Тема. ИСПОЛЬЗОВАНИЕ КОНТЕЙНЕРНЫХ КЛАССОВ ПРИ ПРОГРАММИРОВАНИИ НА JAVA**

Дисциплина Программирование на языке Джава

Выполнил

|  |  |
| --- | --- |
| студент | Болотов Михаил |
|  | Фамилия И.О. |
| группа | ИКБО-06-19 |
|  | Номер группы |

Москва 2020

**Содержание**

[​ Теория 2](#__RefHeading___Toc4128_811265792)

[​ Задание 2](#__RefHeading___Toc4136_811265792)

[​ Код 3](#__RefHeading___Toc35859_811265792)

[​ UnfairWaitList.java 3](#__RefHeading___Toc68406_811265792)

[​ WaitList.java 4](#__RefHeading___Toc68408_811265792)

[​ Main.java 6](#__RefHeading___Toc68410_811265792)

[​ IWaitList.java 7](#__RefHeading___Toc68412_811265792)

[​ BoundedWaitList.java 8](#__RefHeading___Toc68414_811265792)

[​ Скриншот 9](#__RefHeading___Toc3748_811265792)

[​ Заключение 9](#__RefHeading___Toc3750_811265792)

[​ Библиографический список 9](#__RefHeading___Toc3752_811265792)

# Теория

Вначале, рассмотрите класс, представленный на диаграмме - общий класс список ожидания. Допустим мы решили, что нам понадобиться еще два вида списков ожидания:

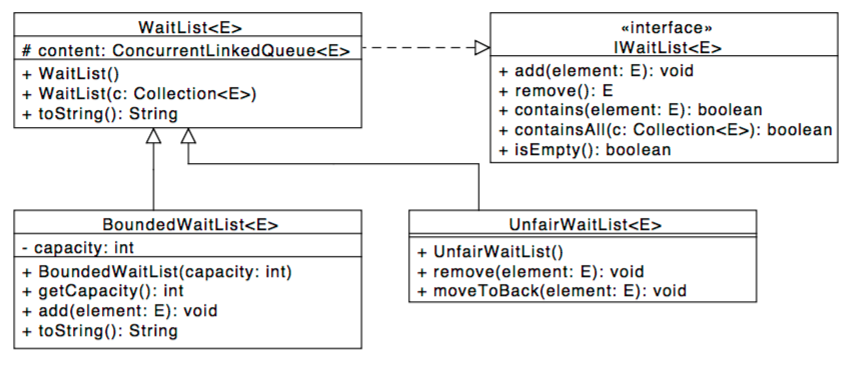
• BoundedWaitList: Этот список ожидания имеет ограниченную емкость, указываемую в момент создания. Он не принимает более элементов, чем заранее задано (возможное количество потенциальных элементов в списке ожидания).

• UnfairWaitList: В этом списке ожидания, можно удалить элемент, который не является первым в очереди - и помните он не может вернуться обратно! (Возможны различные реализации, но в вашей реализации необходимо удалить первое вхождение данного элемента.) Также возможно, чтобы например, первый элемент будет отправлен обратно в конец списка.

После описания всей задачи в целом, мы сможем решить, что мы нам нужен интерфейс IWaitList, и затем нужно создать три разных класса для трех списков ожидания. Также предполагается, что один из списков ожидания должен быть супер классом для двух других списков ожидания.

# 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.



# Код

### UnfairWaitList.java

package dev.ky3he4ik.pr.pr8;  
  
import java.util.concurrent.ConcurrentLinkedQueue;  
  
/\*\*  
 \* Список ожидания, в котором можно любой элемент удалить либо переместить в конец  
 \*/  
public class UnfairWaitList<E> extends WaitList<E> {  
 protected ConcurrentLinkedQueue<E> exp = new ConcurrentLinkedQueue<>();  
  
 /\*\*  
 \* Удаление @param element  
 \*/  
 public void remove(E element) {  
 if (!content.peek().equals(element)) {  
 content.remove(element);  
 exp.add(element);  
 }  
 }  
  
 @Override  
 public void add(E element) {  
 if (!exp.contains(element))  
 super.add(element);  
 }  
  
 /\*\*  
 \* Перемещение в конец @param element  
 \*/  
 public void moveToBack(E element) {  
 if (content.remove(element))  
 content.add(element);  
 }  
}

### WaitList.java

package dev.ky3he4ik.pr.pr8;  
  
import java.util.Collection;  
import java.util.concurrent.ConcurrentLinkedQueue;  
  
/\*\*  
 \* Обычная реализация  
 \* @param <E>  
 \*/  
public class WaitList<E> implements IWaitList<E> {  
 /\*\*  
 \* Сама очередь  
 \*/  
 protected ConcurrentLinkedQueue<E> content = new ConcurrentLinkedQueue<>();  
  
 /\*\*  
 \* Конструктор по умолчанию  
 \*/  
 WaitList() {  
 }  
  
 /\*\*  
 \* Коснстуктор с заполнением из коллекции @param c  
 \*/  
 WaitList(Collection<E> c) {  
 content.addAll(c);  
 }  
  
 /\*\*  
 \* Добавление @param element в очередь  
 \*/  
 @Override  
 public void add(E element) {  
 content.add(element);  
 }  
  
 /\*\*  
 \* Удаление первого элемента  
 \* @return удаленный элемент  
 \*/  
 @Override  
 public E remove() {  
 return content.poll();  
 }  
  
 /\*\*  
 \* @return содержит ли @param element  
 \*/  
 @Override  
 public boolean contains(E element) {  
 return content.contains(element);  
 }  
  
 /\*\*  
 \* @return содержит ли все элементы из @param c  
 \*/  
 @Override  
 public boolean containsAll(Collection<E> c) {  
 return content.containsAll(c);  
 }  
  
 /\*\*  
 \* @return пустая ли очередь  
 \*/  
 @Override  
 public boolean isEmpty() {  
 return content.isEmpty();  
 }  
  
 /\*\*  
 \* @return приведенная в строку очередь  
 \*/  
 @Override  
 public String toString() {  
 return content.toString();  
 }  
}

### Main.java

package dev.ky3he4ik.pr.pr8;

public class Main {

public static void main(String[] args) {

WaitList<Integer> waitList = new WaitList<>();

assert waitList.isEmpty();

assert !waitList.contains(5);

waitList.add(5);

assert waitList.contains(5);

boolean t = waitList.remove() == 5;

assert t;

BoundedWaitList<Integer> waitList2 = new BoundedWaitList<>(3);

assert waitList2.isEmpty();

assert waitList2.getCapacity() == 3;

assert !waitList2.contains(5);

waitList2.add(5);

assert waitList2.contains(5);

t = waitList2.remove() == 5;

assert t;

waitList2.add(5);

waitList2.add(6);

waitList2.add(7);

try {

waitList2.add(8);

assert false;

} catch (IllegalStateException e) {

}

UnfairWaitList<Integer> waitList3 = new UnfairWaitList<>();

assert waitList3.isEmpty();

assert !waitList3.contains(5);

waitList3.add(5);

assert waitList3.contains(5);

t = waitList3.remove() == 5;

assert t;

waitList3.add(5);

waitList3.add(6);

waitList3.add(7);

waitList3.remove(5);

waitList3.moveToBack(6);

assert waitList3.remove() == 7;

assert waitList3.remove() == 6;

System.out.println("WaitLists tested successfully");

}

}

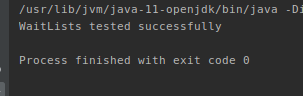
### IWaitList.java

package dev.ky3he4ik.pr.pr8;  
  
import java.util.Collection;  
  
/\*\*  
 \* Интерфейс для списка ожидания  
 \* @param <E> - ожидающий элемент  
 \*/  
public interface IWaitList<E> {  
 /\*\*  
 \* добавление элемента @param element  
 \*/  
 void add(E element);  
  
 /\*\*  
 \* Извлечение и возврат первого элемента в очереди  
 \*/  
 E remove();  
  
 /\*\*  
 \* Проверка, есть ли @param element в очереди  
 \*/  
 boolean contains(E element);  
  
 /\*\*  
 \* Проверка, есть ли @param c в очереди  
 \*/  
 boolean containsAll(Collection<E> c);  
  
 /\*\*  
 \* Проверка, пустая ли очередь  
 \*/  
 boolean isEmpty();  
}

### BoundedWaitList.java

package dev.ky3he4ik.pr.pr8;  
  
/\*\*  
 \* Список ожидания с ограниченной емкостью  
 \*/  
public class BoundedWaitList<E> extends WaitList<E> {  
 /\*\*  
 \* ёмкость очереди  
 \*/  
 private int capacity;  
  
 /\*\*  
 \* Создает очередь с ёмкостью @param capacity  
 \*/  
 public BoundedWaitList(int capacity) {  
 this.capacity = capacity;  
 }  
  
 /\*\*  
 \* @return ёмкость очередь  
 \*/  
 public int getCapacity() {  
 return capacity;  
 }  
  
 /\*\*  
 \* Добавляет @param element в очередь  
 \*/  
 @Override  
 public void add(E element) {  
 if (content.size() < capacity)  
 super.add(element);  
 else  
 throw new IllegalStateException("BoundedWaitList overflow. Maximal capacity: " + capacity);  
 }  
  
 /\*\*  
 \* @return приведенная в строку очередь  
 \*/  
 @Override  
 public String toString() {  
 return "[" + capacity + "] " + content;  
 }  
}

# Скриншот



# Заключение

В данной практической работе я освоил на практике приемы работы со контейнерными классами.

# Библиографический список

1. Зорина Н.В. Курс лекций по Объектно-ориентированному программированию на Java, МИРЭА, Москва, 2016
2. Программирование на языке Java: работа со строками и массивами. Методические указания. [Электронный ресурс] : Учебно-методические пособия — Электрон. дан. — СПб. : ПГУПС, 2015. — 24 с.
3. Кожомбердиева, Г.И. Программирование на языке Java: создание графического интерфейса пользователя: учеб. пособие. [Электронный ресурс] : Учебные пособия / Г.И. Кожомбердиева, М.И. Гарина. — Электрон. дан. — СПб.: ПГУПС, 2012. — 67 с.
4. Вишневская, Т.И. Технология программирования. Часть 1. [Электронный ресурс] / Т.И. Вишневская, Т.Н. Романова. — Электрон. дан. — М. : МГТУ им. Н.Э. Баумана, 2007. — 59 с.