|  |
| --- |
|  |
| 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 №7, №8**

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

Принял Степанов П.В.

Практические работы выполнены «\_\_\_»\_\_\_\_\_\_\_2021 г.

«\_\_\_\_\_\_\_\_\_\_\_\_\_\_» «\_\_\_»\_\_\_\_\_\_\_2021 г.

Отметка о выполнении

**Москва – 2021 г.**

**Практическое занятие №7**

**Цель работы:** изучение на практике приемов работы со стандартными

контейнерными классами Java Collection Framework.

**Задание.**

Напишите программу в виде консольного приложения, которая

моделирует карточную игру «пьяница» и определяет, кто выигрывает. В игре

участвует 10 карт, имеющих значения от 0 до 9, большая карта побеждает

меньшую; карта «0» побеждает карту «9».

Карточная игра “ В пьяницу”. В этой игре карточная колода раздается

поровну двум игрокам. Далее они открывают по одной верхней карте, и тот, чья

карта старше, забирает себе обе открытые карты, которые кладутся под низ его

колоды. Тот, кто остается без карт, - проигрывает.

Для простоты будем считать, что все карты различны по номиналу и что

самая младшая карта побеждает самую старшую карту (“шестерка берет туз”).

Игрок, который забирает себе карты, сначала кладет под низ своей колоды

карту первого игрока, затем карту второго игрока (то есть карта второго игрока

оказывается внизу колоды).

**Входные данные.**

Программа получает на вход две строки: первая строка содержит 5 карт

первого игрока, вторая - 5 карт второго игрока. Карты перечислены сверху вниз,

то есть каждая строка начинается с той карты, которая будет открыта первой.

**Выходные данные.**

Программа должна определить, кто выигрывает при данной раздаче, и

вывести слово first или second, после чего вывести количество ходов, сделанных

до выигрыша. Если на протяжении 106 ходов игра не заканчивается, программа

должна вывести слово botva

**Код программы:**

**Класс Main**

public class Main {  
 public static void main(String[] args) {  
 ArrayDeque<Integer> qP1 = new ArrayDeque<>(), qP2 = new ArrayDeque<>();  
 ArrayDeque<Integer> deqP1 = new ArrayDeque<>(), deqP2 = new ArrayDeque<>();  
 int steps = 0;  
 Scanner in = new Scanner(System.*in*);  
 int x;  
 for(int i = 0; i < 5;i++) {  
 x = in.nextInt();  
 qP1.addLast(x);  
 deqP1.addFirst(x);  
 }  
 for(int i = 0; i < 5;i++) {  
 x = in.nextInt();  
 qP2.addLast(x);  
 deqP2.addFirst(x);  
 }  
 while(steps != 106 && !(qP1.isEmpty() || qP2.isEmpty() || deqP1.isEmpty() || deqP2.isEmpty())) {  
 steps += 1;  
 if (qP1.getFirst() == 0 && qP2.getFirst() == 9 || qP1.getFirst() > qP2.getFirst() && (qP1.getFirst() != 9 || qP2.getFirst() != 0)) {  
 qP1.addLast(qP1.pollFirst());  
 qP1.addLast(qP2.pollFirst());  
 } else {  
 qP2.addLast(qP1.pollFirst());  
 qP2.addLast(qP2.pollFirst());  
 }  
 if (deqP1.getLast() == 0 && deqP2.getLast() == 9 || deqP1.getLast() > deqP2.getLast() && (deqP1.getLast() != 9 || deqP2.getLast() != 0)){  
 deqP1.addFirst(deqP1.pollLast());  
 deqP1.addFirst(deqP2.pollLast());  
 } else {  
 deqP2.addFirst(deqP1.pollLast());  
 deqP2.addFirst(deqP2.pollLast());  
 }  
 }  
 System.*out*.println("Queue");  
 if (steps == 106)  
 System.*out*.println("botva");  
 else if (qP1.isEmpty())  
 System.*out*.println("second " + steps);  
 else if (qP2.isEmpty())  
 System.*out*.println("first " + steps);  
 System.*out*.println("Deque");  
 if (steps == 106)  
 System.*out*.println("botva");  
 else if (deqP1.isEmpty())  
 System.*out*.println("second " + steps);  
 else if (deqP2.isEmpty())  
 System.*out*.println("first " + steps);  
 }  
}

**Тесты**

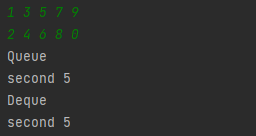


Рисунок 1 – выигрывает 2 игрок

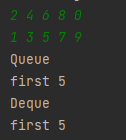


Рисунок 2 – выигрывает 1 игрок

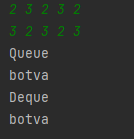


Рисунок 3 – никто не выигрывает

**Практическое занятие №8**

**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 программы:**

**Интерфейс IWaitList**

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();  
}

**Класс WaitList**

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 +  
 '}';  
 }  
}

**Класс UnfairWaitList**

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;  
 }  
 }  
  
 }  
}

**Класс BoundedWaitList**

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;  
 }  
}

**Класс Main**

public class Main {  
 public static void main(String[] args) {  
 ArrayList<Integer> b = new ArrayList<Integer>();  
 b.add(5); b.add(7); b.add(8);  
 WaitList<Integer> a = new WaitList<Integer>(b);  
 a.add(78);  
 System.*out*.println(a.toString());  
 if (!a.isEmpty())  
 System.*out*.println("remove: " + a.remove());  
 System.*out*.println("contains: " + a.contains(67));  
 System.*out*.println("containsAll: " + a.containsAll(b));  
 System.*out*.println("isEmpty: " + a.isEmpty());  
 System.*out*.println(a.toString());  
 System.*out*.println("--------------------------------------");  
 BoundedWaitList<Integer> a2 = new BoundedWaitList<Integer>(b);  
 a2.add(78);  
 System.*out*.println(a2.toString());  
 if (!a.isEmpty())  
 System.*out*.println("remove: " + a2.remove());  
 System.*out*.println("contains: " + a2.contains(67));  
 System.*out*.println("containsAll: " + a2.containsAll(b));  
 System.*out*.println("isEmpty: " + a2.isEmpty());  
 System.*out*.println(a2.toString());  
 System.*out*.println("--------------------------------------");  
 UnfairWaitList<Integer> a3 = new UnfairWaitList<Integer>();  
 a3.add(78); a3.add(5); a3.add(7); a3.add(8); a3.add(789); a3.add(6009); a3.add(72389);  
 System.*out*.println(a3.toString());  
 if (!a.isEmpty())  
 System.*out*.println("remove: " + a3.remove());  
 System.*out*.println("contains: " + a3.contains(67));  
 System.*out*.println("containsAll: " + a3.containsAll(b));  
 System.*out*.println("isEmpty: " + a3.isEmpty());  
 System.*out*.println(a3.toString());  
 a3.remove(7);  
 System.*out*.println("Remove - 7 " + a3.toString());  
 a3.moveToBack(8);  
 System.*out*.println("moveToBack - 8 " + a3.toString());  
 }  
}

**Тест**

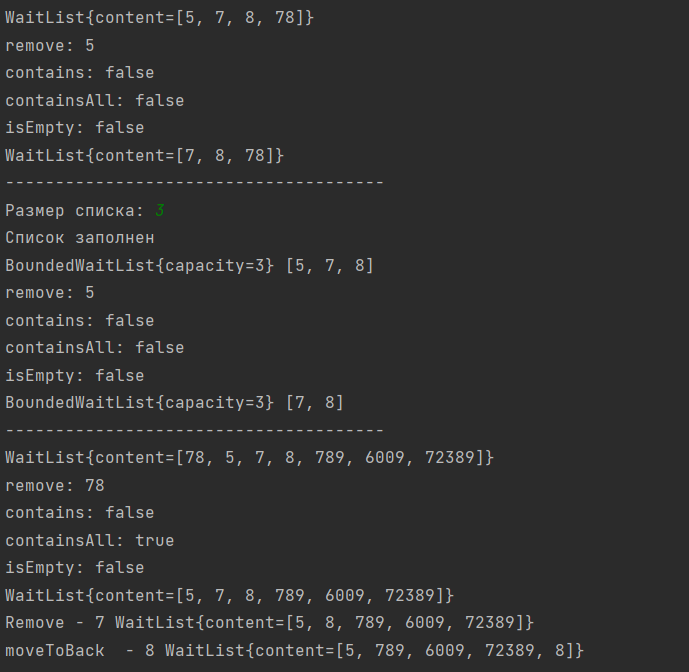


Рисунок 4 – тест

**Выводы**

Я научился работать со стандартными контейнерными классами Java Collection Framework и научился делать списки ожидание разного вида.

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