一 概述

1. JUC 指的是 JDK1.5 中提供的一套并发包及其子包：java.util.concurrent，java.util.concurrent.lock，java.util.concurrent.atomic
2. JUC 的主要内容：阻塞式队列、并发映射、锁、执行器服务、原子性操作

二 BlockingQueue （接口）- 阻塞式队列

1 概述

1) 满足队列的特点：FIFO(First In First Out)
2) BlockingQueue 在使用的时候需要容量，且容量是固定的，不可扩容
3) 阻塞：如果队列为空，则试图获取元素的线程会被阻塞；如果队列已满，则试图放入元素的线程会被阻塞
4) BlockingQueue 中不允许元素为 null
5) 适应于生产消费模型
6) 重要方法：

	抛出异常	返回特殊值	永久阻塞	定时阻塞
添加	add - IllegalStateException	offer - false	put	offer
获取	remove - NoSuchElementException	poll - null	take	poll

2 常用方法测试

2.A add/remove

2.A.a add：

```
public static void main(String[] args) throws InterruptedException {
    // 容量指定之后不可变
    BlockingQueue<String> queue =
            new ArrayBlockingQueue<>(5);
    // 添加元素
    queue.add("a");
    queue.add("a");
    queue.add("a");
    queue.add("a");
    queue.add("a");
```

```
    // 队列已满
    // 抛出异常 - IllegalStateException: Queue full
    queue.add("b");
}
```

```
E:\JAVA\jdk1.8.0_152\bin\java.exe ...
Exception in thread "main" java.lang.IllegalStateException: Queue full
    at java.util.AbstractQueue.add(AbstractQueue.java:98) <1 internal call>
    at cn.tedu.blockingqueue.BlockingQueueDemo.main(BlockingQueueDemo.java:22)
```

2.A.b remove：

```
public static void main(String[] args) throws InterruptedException {

    LinkedBlockingQueue<String> queue =
            new LinkedBlockingQueue<>();
    queue.remove();

}
```

```
E:\JAVA\jdk1.8.0_152\bin\java.exe ...
Exception in thread "main" java.util.NoSuchElementException
    at java.util.AbstractQueue.remove(AbstractQueue.java:117)
    at cn.tedu.blockingqueue.BlockingQueueDemo2.main(BlockingQueueDemo2.java:12)
```

2.B offer/poll

2.B.a offer

```
public static void main(String[] args) throws InterruptedException {

    // 容量指定之后不可变
    BlockingQueue<String> queue =
            new ArrayBlockingQueue<>(5);

    // 添加元素
    queue.add("a");
    queue.add("a");
    queue.add("a");
    queue.add("a");
    queue.add("a");
    // 队列满 offer 添加返回 false
    // 返回 false
    boolean b = queue.offer("c");
    System.out.println(b);//false
}
```

2.B.b poll

```
public static void main(String[] args) throws InterruptedException {

    LinkedBlockingQueue<String> queue =
            new LinkedBlockingQueue<>();
    // 队列为空 返回 null，元素不能为 null
```

```
        System.out.println(queue.poll());//null

    }
```

2.C put/take

2.C.a put

```
public static void main(String[] args) throws InterruptedException {

    // 容量指定之后不可变
    BlockingQueue<String> queue =
            new ArrayBlockingQueue<>(5);
    // 添加满元素
    queue.add("a");
    queue.add("a");
    queue.add("a");
    queue.add("a");
    queue.add("a");
    // 队列满产生阻塞
    queue.put("d");
    System.out.println(queue);
}
```

```
E:\JAVA\jdk1.8.0_152\bin\java.exe ...
```

2.C.b take

```
public static void main(String[] args) throws InterruptedException {
    LinkedBlockingQueue<String> queue =
            new LinkedBlockingQueue<>();
    // 队列没有元素 永久阻塞
    System.out.println(queue.take());
}
```

```
E:\JAVA\jdk1.8.0_152\bin\java.exe ...
```

2.D offer/poll（定时）

```
// 定时阻塞
boolean b = queue.offer("e", 5, TimeUnit.SECONDS);
// 定时阻塞
System.out.println(queue.poll(5, TimeUnit.SECONDS));
```

3 常用实现类

3.A ArrayBlockingQueue - 阻塞式顺序队列：

1. 底层必然基于数组来存储数据
2. 使用的时候需要指定容量

3.B LinkedBlockingQueue - 阻塞式链式队列：

1. 底层必然基于节点来存储数据
2. 在使用的时候可以指定容量也可以不指定。如果指定容量，则容量不可变；如果不指定容量，则容量默认为 `Integer.MAX_VALUE = 231-1` 不可变。因为实际开发中，一般不会在队列中存储 21 亿个元素，所以一般认为此时的容量是无限的

3.C PriorityBlockingQueue - 具有优先级的阻塞式队列：

1. 底层是基于数组来存储元素
2. 使用的时候可以指定容量也可以不指定。如果不指定则默认初始容量是 11
3. `PriorityBlockingQueue` 会对放入的元素来进行排序，要求元素对应的类实现 `Comparable` 接口，覆盖 `compareTo` 方法指定比较规则。对于不能修改的类，如果要修改排序规则，可在创建 `PriorityBlockingQueue` 对象时指定 `Comparator` 接口
4. `PriorityBlockingQueue` 在迭代遍历的时候不保证排序

3.D SynchronousQueue - 同步队列

1. 在使用的时候不需要指定容量，容量默认为 1 且只能为 1

4 PriorityBlockingQueue - 具有优先级的阻塞式队列

4.A 遍历(for i)

输出结果，为排序后

```
    public static void main(String[] args) throws InterruptedException
{

        PriorityBlockingQueue<String> queue =
                new PriorityBlockingQueue<>();
        queue.put("c");
        queue.put("a");
        queue.put("b");
        queue.put("e");
```

```
        queue.put("f");
        queue.put("d");
        queue.put("g");
        for (int i = 0; i < 5; i++) {
            System.out.println(queue.take());
        }

    }
```

```
E:\JAVA\jdk1.8.0_152\bin\java.exe ...
a
b
c
d
e
f
g
```

4.B 遍历(for each)

未排序

```
public static void main(String[] args) throws InterruptedException {

                        PriorityBlockingQueue<String>    queue    =
                                new   PriorityBlockingQueue<>();
                                        queue.put("c");
                                        queue.put("a");
                                        queue.put("b");
                                        queue.put("e");
                                        queue.put("f");
                                        queue.put("d");
                                        queue.put("g");
                    for    (int    i    =    0;    i    <    5;    i++)    {
                                    System.out.println(queue.take());
                                                                        }

    }
```

```
E:\JAVA\jdk1.8.0_152\bin\java.exe ...
a
c
b
e
f
d
g
```

4.C 排序（实现 Comparable 接口）

实现 compareTo 方法

```
class Student implements Comparable<Student> {
```

```
// 指定排序规则
// 按照年龄升序
// 升序：this - o
// 降序：o - this
@Override
public int compareTo(Student o) {

    return this.age - o.age;
}
```

4.D 排序（自定义 Comparator 实现类）

当 Student 中的类不允许更改，这时可以通过 *Comparator 覆盖*

```
// 按照分数降序排序
// 比较器
Comparator<Student> c = new Comparator<Student>() {
    @Override
    public int compare(Student o1, Student o2) {
        return o2.getScore() - o1.getScore();
    }
};
```

```
// 在定义队列的时候，传入的比较器的优先级要高于 Comparable
PriorityBlockingQueue<Student> queue =
        new PriorityBlockingQueue<>(5, c);
```

三 （扩展）BlockingDeque - 阻塞式双端队列

1. 特点：允许从两端放入元素也允许从两端获取元素
2. 遵循阻塞特点，在使用的时候需要指定容量