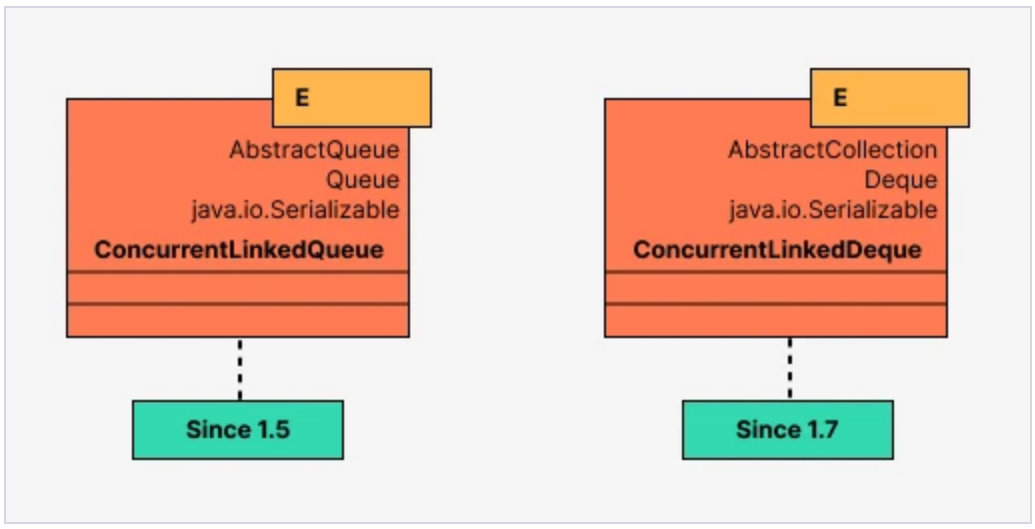
Concurrent Queues

JSP & Servlets
19 уровень, 3 лекция

ОТКРЫТА

Non-Blocking Queues



Потокобезопасные и самое важное неблокирующие имплементации ***Queue*** на связанных нодах (linked nodes).

`ConcurrentLinkedQueue<E>` — тут используется wait-free алгоритм, адаптированный для работы с garbage collector'ом. Этот алгоритм довольно эффективен и очень быстр, так как построен на CAS. Метод `size()` может работать долго, так что лучше постоянно его не дергать.

`ConcurrentLinkedDeque<E>` — Deque расшифровывается как Double ended queue. Это означает, что данные можно добавлять и вытаскивать с обеих сторон. Соответственно, класс поддерживает оба режима работы: FIFO (First In First Out) и LIFO (Last In First Out).

На практике `ConcurrentLinkedDeque` стоит использовать в том случае, если обязательно нужно именно LIFO, так как за счет двунаправленности нод данный класс проигрывает по производительности наполовину по сравнению с `ConcurrentLinkedQueue`.

```
1  import java.util.concurrent.ConcurrentLinkedQueue;
2
3  public class  ConcurrentLinkedQueueExample {
4     public static void main(String[] args) {
5         ConcurrentLinkedQueue<String> queue = new ConcurrentLinkedQueue<>();
6
7         Thread producer = new Thread(new Producer(queue));
8         Thread consumer = new Thread(new Consumer(queue));
9
10        producer.start();
11        consumer.start();
12    }
13 }
14
15 class Producer implements Runnable {
16
17    ConcurrentLinkedQueue<String> queue;
18    Producer(ConcurrentLinkedQueue<String> queue){
19        this.queue = queue;
20    }
```

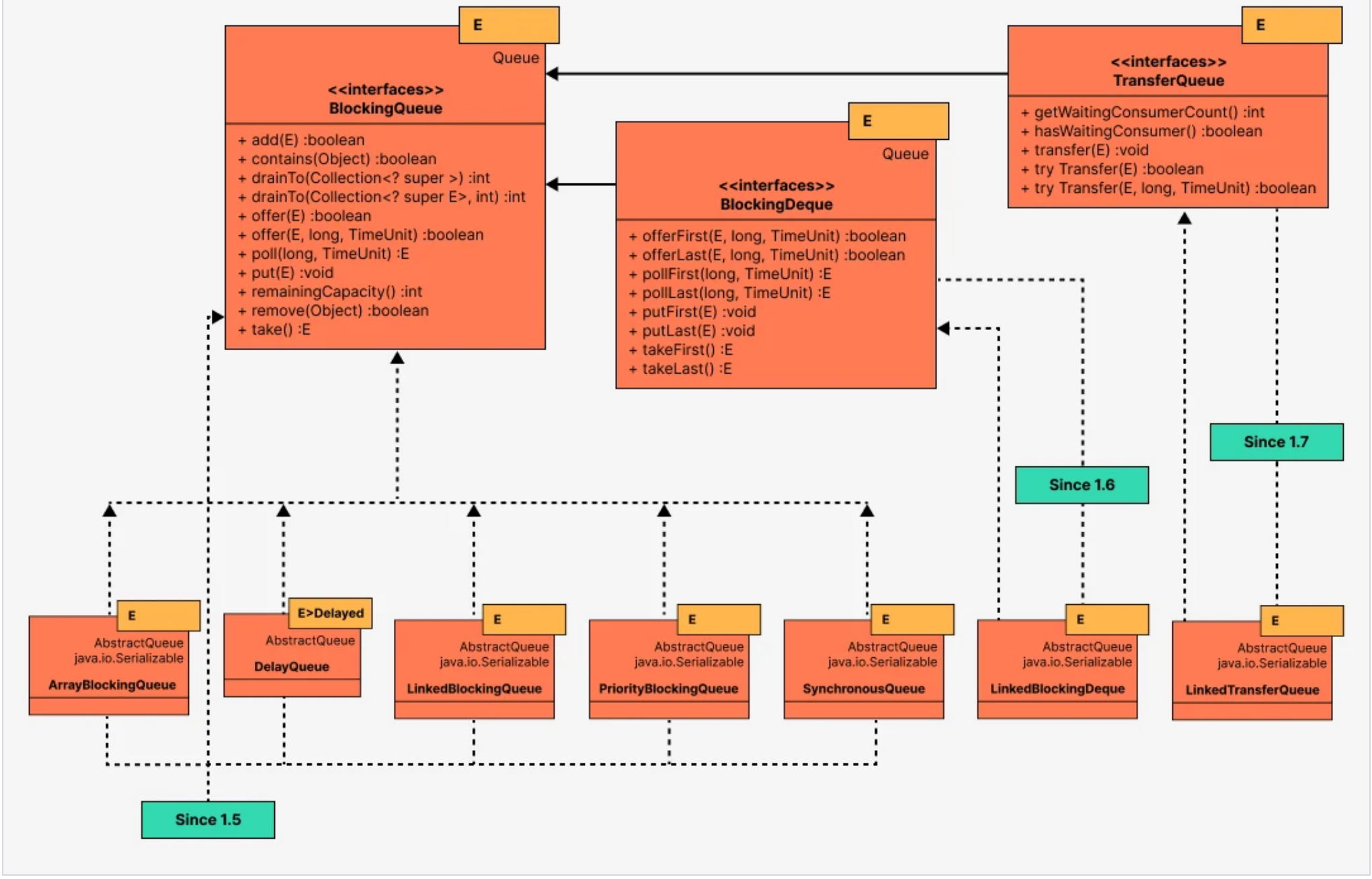
```
    public void run() {
        System.out.println("Класс для добавление элементов в очередь");
        try {
            for (int i = 1; i < 5; i++) {
                queue.add("Элемент #" + i);
                System.out.println("Добавили: Элемент #" + i);
                Thread.sleep(300);
            }
        } catch (InterruptedException ex) {
            ex.printStackTrace();
            Thread.currentThread().interrupt();
        }
    }
}

class Consumer implements Runnable {

    ConcurrentLinkedQueue<String> queue;
    Consumer(ConcurrentLinkedQueue<String> queue){
        this.queue = queue;
    }

    public void run() {
        String str;
        System.out.println("Класс для получения элементов из очереди");
        for (int x = 0; x < 5; x++) {
            while ((str = queue.poll()) != null) {
                System.out.println("Вытянули: " + str);
            }
            try {
                Thread.sleep(600);
            } catch (InterruptedException ex) {
                ex.printStackTrace();
                Thread.currentThread().interrupt();
            }
        }
    }
}
```

Blocking Queues



Интерфейс ***BlockingQueue<E>*** — при большом количестве данных `ConcurrentLinkedQueue` не хватает.

Когда потоки не справляются с поставленной задачей, ты легко можешь получить **OutOfMemmoryException**. И чтобы такие случаи не возникали, у нас для работы есть ***BlockingQueue*** с наличием разных методов для заполнения и работы с очередью и блокировками по условиям.

BlockingQueue не признает нулевых элементов (null) и вызывает **NullPointerException** при попытке добавить или получить такой элемент. Нулевой элемент возвращает метод poll, если в течение таймаута не был размещен в очереди очередной элемент.

Реализации BlockingQueue<E>

Давай разберем подробно каждую из реализаций нашей ***BlockingQueue***:

`ArrayBlockingQueue<E>` — класс блокирующей очереди, построенный на классическом кольцевом буфере. Здесь нам доступна возможность управлять “честностью” блокировок. Если fair=false (по умолчанию), то очередность работы потоков не гарантируется.

`DelayQueue<E extends Delayed>` — класс, который позволяет вытаскивать элементы из очереди только по прошествии некоторой задержки, определенной в каждом элементе через метод `getDelay` интерфейса ***Delayed***.

`LinkedBlockingQueue<E>` — блокирующая очередь на связанных нодах, реализованная на “two lock queue” алгоритме: первый лок — на добавление, второй — на вытаскивание элемента из очереди. За счет локов, по сравнению с `ArrayBlockingQueue`, данный класс имеет высокую производительность, но для него необходимо большее количество памяти. Размер очереди задается через конструктор и по умолчанию равен Integer.MAX_VALUE.

`PriorityBlockingQueue<E>` — многопоточная обертка над `PriorityQueue`. ***Comparator*** отвечает за то, по какой логике будет добавлен элемент. Первым же из очереди выходит самый наименьший элемент.

`SynchronousQueue<E>` — очередь работает по принципу FIFO(first-in-first-out). Каждая операция вставки блокирует поток “Producer”до тех пор, пока поток “Consumer” не вытащит элемент из очереди и наоборот, “Consumer” будет ждать пока “Producer” не вставит элемент.

BlockingDeque<E> — интерфейс, который описывает дополнительные методы для двунаправленной блокирующей очереди. Данные можно вставлять и вытаскивать с обеих сторон очереди.

`LinkedBlockingDeque<E>` — двунаправленная блокирующая очередь на связанных нодах, реализованная как простой двунаправленный список с одним локом. Размер очереди задается через конструктор и по умолчанию равен Integer.MAX_VALUE.

TransferQueue<E> — интерфейс интересен тем, что при добавлении элемента в очередь существует возможность заблокировать вставляющий поток **Producer** до тех пор, пока другой поток **Consumer** не вытащит элемент из очереди. Также можно добавить проверку на определенный тайм-аут или выставить проверку на наличие ожидающих **Consumer**. Как итог, мы получаем механизм передачи данных с поддержкой асинхронных и синхронных сообщений.

`LinkedTransferQueue<E>` — реализация ***TransferQueue*** на основе алгоритма Dual Queues with Slack. Активно использует CAS (смотрите выше) и парковку потоков, когда они находятся в режиме ожидания.

‹ Предыдущая лекция

Следующая лекция ›

JavaRush — это интерактивный онлайн-курс по изучению Java-программирования с нуля. Он содержит 1200 практических задач с проверкой решения в один клик, необходимый минимум теории по основам Java и мотивирующие фишки, которые помогут пройти курс до конца: игры, опросы, интересные проекты и статьи об эффективном обучении и карьере Java-девелопера.

ПОДПИСЫВАЙТЕСЬ

ЯЗЫК ИНТЕРФЕЙСА

Русский

СКАЧИВАЙТЕ НАШИ ПРИЛОЖЕНИЯ