Очередь

Интерфейс Java Queue, java.util.Queue, представляет собой структуру данных, предназначенную для вставки элементов в конец очереди и удаления элементов из начала очереди. Это похоже на работу очереди в супермаркете.

Интерфейс является подтипом интерфейса Collection. Он представляет упорядоченную последовательность объектов так же, как список, но его предполагаемое использование немного отличается.

Реализации

Будучи подтипом Collection, все методы в интерфейсе Collection также доступны в интерфейсе Queue.

Поскольку Queue является интерфейсом, вам необходимо создать конкретную реализацию, чтобы использовать его. Вы можете выбрать одну из следующих в API коллекций:

java.util.LinkedList – довольно стандартная реализация очереди;

java.util.PriorityQueue хранит свои элементы внутри в соответствии с их естественным порядком (если они реализуют Comparable) или в соответствии с Comparator, переданным в PriorityQueue.

Есть также реализации Queue в пакете java.util.concurrent, но оставим утилиты параллелизма вне этого урока.

```
Queue queueA = new LinkedList();

Queue queueB = new PriorityQueue();
```

Как добавить элемент

Для добавления элементов в очередь вызывается ее метод add (). Этот метод наследуется от интерфейса коллекции.

```
Queue queueA = new LinkedList();

queueA.add("element 1");

queueA.add("element 2");

queueA.add("element 3");
```

Как получить элемент

Порядок, в котором элементы, добавленные в очередь, хранятся внутри, зависит от реализации. То же самое верно для порядка, в котором элементы извлекаются из очереди.

Как посмотреть на первый элемент

Вы можете посмотреть на элемент в начале очереди, не вынимая его из очереди, element() или метода peek().

Метод element() возвращает первый элемент в очереди. Если очередь пуста, вызывает исключение NoSuchElementException.

```
Queue queue = new LinkedList();

queue.add("element 1");

queue.add("element 2");

queue.add("element 3");

Object firstElement = queue.element();
```

После выполнения этого кода переменная firstElement будет содержать элемент value 1, который является первым элементом в очереди.

Peek() работает так же, как метод element (), за исключением того, что он не создает исключение, если очередь пуста. Вместо этого он просто возвращает null. Вот пример:

```
Queue queue = new LinkedList();

queue.add("element 1");

queue.add("element 2");

queue.add("element 3");

Object firstElement = queue.peek();

Object firstElement = queueA.element();
```

Как итерировать все элементы?

Вы также можете перебирать все элементы очереди, а не просто обрабатывать по одному за раз.

```
Queue queueA = new LinkedList();

queueA.add("element 0");

queueA.add("element 1");

queueA.add("element 2");

//access via Iterator

Iterator iterator = queueA.iterator();

while(iterator.hasNext(){

  String element =(String) iterator.next();

}

//access via new for-loop

for(Object object : queueA) {
```

```
    String element =(String) object;
}
```

При выполнении итерации очереди через ее итератор или цикл for-each (который также использует итератор), последовательность, в которой элементы итерируются, зависит от реализации очереди.

Как удалить элемент?

Вызывается метод remove(). Он удаляет элемент в начале очереди. В большинстве реализаций начало и конец очереди находятся на противоположных концах. Однако возможно реализовать интерфейс очереди, чтобы заголовок и конец находились в одном конце. В этом случае у вас будет стек.

```
Object firstElement = queue.remove();
```

Общая очередь

По умолчанию вы можете поместить любой объект в очередь, но начиная с Java 5, Generics позволяет ограничить типы объектов, которые вы можете вставить в нее. Вот вам пример:

```
Queue queue = new LinkedList();
```

Теперь в эту очередь могут быть вставлены только экземпляры MyObject. Затем вы можете получить доступ к его элементам и повторить их без приведения. Вот как это выглядит:

```
MyObject myObject = queue.remove();

for(MyObject anObject : queue){
   //do someting to anObject...
}
```