Java Collections - Queue

Jakob Jenkov
Last update: 2014-06-23

The `java.util.Queue` interface is a subtype of the `java.util.Collection` interface. It represents an ordered list of objects just like a `List`, but its intended use is slightly different. A queue is designed to have elements inserted at the end of the queue, and elements removed from the beginning of the queue. Just like a queue in a supermarket.

Here is a list of the topics covered in this text:

Queue Implementations

Being a `Collection` subtype all methods in the `Collection` interface are also available in the `Queue` interface.

Since `Queue` is an interface you need to instantiate a concrete implementation of the interface in order to use it. You can choose between the following `Queue` implementations in the Java Collections API:

- java.util.LinkedList
- java.util.PriorityQueue

`LinkedList` is a pretty standard queue implementation.

`PriorityQueue` stores its elements internally according to their natural order (if they implement `Comparable`), or according to a `Comparator` passed to the `PriorityQueue`.

There are also `Queue` implementations in the `java.util.concurrent` package, but I will leave the concurrency utilities out of this tutorial.

Here are a few examples of how to create a `Queue` instance:

```
Queue queueA = new LinkedList();
Queue queueB = new PriorityQueue();
```

Adding and Accessing Elements

To add elements to a `Queue` you call its `add()` method. This method is inherited from the `Collection` interface. Here are a few examples:

```
Queue queueA = new LinkedList();

queueA.add("element 1");
queueA.add("element 2");
queueA.add("element 3");
```

The order in which the elements added to the `Queue` are stored internally, depends on the implementation. The same is true for the order in which elements are retrieved from the queue. You should consult the JavaDoc's for more information about the specific `Queue` implementations.

You can peek at the element at the head of the queue without taking the element out of the queue. This is done via the `element()` method. Here is how that looks:

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

To take the first element out of the queue, you use the `remove()` method which is

All Trails | Trail TOC | Page TOC | Previous | Next

You can also iterate all elements of a queue, instead of just processing one at a time. Here is how that looks:

```
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;
}
```

When iterating the queue via its `Iterator` or via the for-loop (which also uses the Iterator behind the scene, the sequence in which the elements are iterated depends on the queue implementation.

Removing Elements

To remove elements from a queue, you call the `remove()` method. This method removes the element at the head of the queue. In most `Queue` implementations the head and tail of the queue are at opposite ends. It is possible, however, to implement the `Queue` interface so that the head and tail of the queue is in the same end. In that case you would have a stack.

Here is a remove example();

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

Generic Queue

By default you can put any `Object` into a `Queue`, but from Java 5, Java Generics makes it possible to limit the types of object you can insert into a `Queue`. Here is an example:

```
Queue<MyObject> queue = new LinkedList<MyObject>();
```

This `Queue` can now only have `MyObject` instances inserted into it. You can then access and iterate its elements without casting them. Here is how it looks:

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

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

For more information about Java Generics, see the **Java Generics Tutorial**.

More Details in the JavaDoc

There is a lot more you can do with a `Queue`, but you will have to check out the JavaDoc for more details. This text focused on the two most common operations: Adding / removing elements, and iterating the elements.

Next: **Java Collections - Deque**

All Trails | Trail TOC | Page TOC | Previous | Next