compact1, compact2, compact3

java.util

Class PriorityQueue<E>

java.lang.Object
    java.util.AbstractCollection<E>
        java.util.AbstractQueue<E>
            java.util.PriorityQueue<E>

Type Parameters:

`E - the type of elements held in this collection`

All Implemented Interfaces:

`Serializable, Iterable<E>, Collection<E>, Queue<E>`

```
public class PriorityQueue<E>
extends AbstractQueue<E>
implements Serializable
```

An unbounded priority queue based on a priority heap. The elements of the priority queue are ordered according to their natural ordering, or by a `Comparator` provided at queue construction time, depending on which constructor is used. A priority queue does not permit `null` elements. A priority queue relying on natural ordering also does not permit insertion of non-comparable objects (doing so may result in `ClassCastException`).

The *head* of this queue is the *least* element with respect to the specified ordering. If multiple elements are tied for least value, the head is one of those elements -- ties are broken arbitrarily. The queue retrieval operations `poll`, `remove`, `peek`, and `element` access the element at the head of the queue.

A priority queue is unbounded, but has an internal *capacity* governing the size of an array used to store the elements on the queue. It is always at least as large as the queue size. As elements are added to a priority queue, its capacity grows automatically. The details of the growth policy are not specified.

This class and its iterator implement all of the *optional* methods of the `Collection` and `Iterator` interfaces. The Iterator provided in method `iterator()` is *not* guaranteed to traverse the elements of the priority queue in any particular order. If you need ordered traversal, consider using `Arrays.sort(pq.toArray())`.

Note that this implementation is not synchronized. Multiple threads should not access a `PriorityQueue` instance concurrently if any of the threads modifies the queue. Instead, use the thread-safe `PriorityBlockingQueue` class.

Implementation note: this implementation provides O(log(n)) time for the enqueuing and dequeuing methods (`offer`, `poll`, `remove()` and `add`); linear time for the `remove(Object)` and `contains(Object)` methods; and constant time for the retrieval methods (`peek`, `element`, and `size`).

This class is a member of the Java Collections Framework.

Since:

1.5

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor and Description
`PriorityQueue``()` Creates a `PriorityQueue` with the default initial capacity (11) that orders its elements according to their **natural ordering**.
`PriorityQueue``(`**`Collection`**`<? extends `**`E`**`> c)` Creates a `PriorityQueue` containing the elements in the specified collection.
`PriorityQueue``(`**`Comparator`**`<? super `**`E`**`> comparator)` Creates a `PriorityQueue` with the default initial capacity and whose elements are ordered according to the specified comparator.
`PriorityQueue``(int initialCapacity)` Creates a `PriorityQueue` with the specified initial capacity that orders its elements according to their **natural ordering**.
`PriorityQueue``(int initialCapacity, `**`Comparator`**`<? super `**`E`**`> comparator)` Creates a `PriorityQueue` with the specified initial capacity that orders its elements according to the specified comparator.
`PriorityQueue``(`**`PriorityQueue`**`<? extends `**`E`**`> c)` Creates a `PriorityQueue` containing the elements in the specified priority queue.
`PriorityQueue``(`**`SortedSet`**`<? extends `**`E`**`> c)` Creates a `PriorityQueue` containing the elements in the specified sorted set.

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description

`boolean`	**`add`**`(`**`E`**` e)` Inserts the specified element into this priority queue.
`void`	**`clear`**`()` Removes all of the elements from this priority queue.
`Comparator``<? super `**`E`**`>`	**`comparator`**`()` Returns the comparator used to order the elements in this queue, or `null` if this queue is sorted according to the **natural ordering** of its elements.
`boolean`	**`contains`**`(`**`Object`**` o)` Returns `true` if this queue contains the specified element.
`Iterator``<`**`E`**`>`	**`iterator`**`()` Returns an iterator over the elements in this queue.
`boolean`	**`offer`**`(`**`E`**` e)` Inserts the specified element into this priority queue.
`E`	**`peek`**`()` Retrieves, but does not remove, the head of this queue, or returns `null` if this queue is empty.
`E`	**`poll`**`()` Retrieves and removes the head of this queue, or returns `null` if this queue is empty.
`boolean`	**`remove`**`(`**`Object`**` o)` Removes a single instance of the specified element from this queue, if it is present.
`int`	**`size`**`()` Returns the number of elements in this collection.
`Spliterator``<`**`E`**`>`	**`spliterator`**`()` Creates a ***late-binding*** and *fail-fast* **`Spliterator`** over the elements in this queue.
`Object``[]`	**`toArray`**`()` Returns an array containing all of the elements in this queue.
`<T> T[]`	**`toArray`**`(T[] a)` Returns an array containing all of the elements in this queue; the runtime type of the returned array is that of the specified array.

Methods inherited from class java.util.AbstractQueue

`addAll, element, remove`

Methods inherited from class java.util.AbstractCollection

`containsAll, isEmpty, removeAll, retainAll, toString`

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Methods inherited from interface java.util.Collection

`containsAll, equals, hashCode, isEmpty, parallelStream, removeAll, removeIf, retainAll, stream`

Methods inherited from interface java.lang.Iterable

`forEach`

Constructor Detail

PriorityQueue

```
public PriorityQueue()
```

Creates a `PriorityQueue` with the default initial capacity (11) that orders its elements according to their natural ordering.

PriorityQueue

```
public PriorityQueue(int initialCapacity)
```

Creates a `PriorityQueue` with the specified initial capacity that orders its elements according to their natural ordering.

Parameters:

`initialCapacity` - the initial capacity for this priority queue

Throws:

`IllegalArgumentException` - if `initialCapacity` is less than 1

PriorityQueue

```
public PriorityQueue(Comparator<? super E> comparator)
```

Creates a `PriorityQueue` with the default initial capacity and whose elements are ordered according to the specified comparator.

Parameters:

`comparator` - the comparator that will be used to order this priority queue. If `null`, the natural ordering of the elements will be used.

Since:

1.8

PriorityQueue

```
public PriorityQueue(int initialCapacity,
                     Comparator<? super E> comparator)
```

Creates a `PriorityQueue` with the specified initial capacity that orders its elements according to the specified comparator.

Parameters:

`initialCapacity` - the initial capacity for this priority queue

`comparator` - the comparator that will be used to order this priority queue. If `null`, the natural ordering of the elements will be used.

Throws:

`IllegalArgumentException` - if `initialCapacity` is less than 1

PriorityQueue

```
public PriorityQueue(Collection<? extends E> c)
```

Creates a `PriorityQueue` containing the elements in the specified collection. If the specified collection is an instance of a `SortedSet` or is another `PriorityQueue`, this priority queue will be ordered according to the same ordering. Otherwise, this priority queue will be ordered according to the natural ordering of its elements.

Parameters:

`c` - the collection whose elements are to be placed into this priority queue

Throws:

`ClassCastException` - if elements of the specified collection cannot be compared to one another according to the priority queue's ordering

`NullPointerException` - if the specified collection or any of its elements are null

PriorityQueue

```
public PriorityQueue(PriorityQueue<? extends E> c)
```

Creates a `PriorityQueue` containing the elements in the specified priority queue. This priority queue will be ordered according to the same ordering as the given priority queue.

Parameters:

`c` - the priority queue whose elements are to be placed into this priority queue

Throws:

`ClassCastException` - if elements of `c` cannot be compared to one another according to `c`'s ordering

`NullPointerException` - if the specified priority queue or any of its elements are null

PriorityQueue

```
public PriorityQueue(SortedSet<? extends E> c)
```

Creates a `PriorityQueue` containing the elements in the specified sorted set. This priority queue will be ordered according to the same ordering as the given sorted set.

Parameters:

`c` - the sorted set whose elements are to be placed into this priority queue

Throws:

`ClassCastException` - if elements of the specified sorted set cannot be compared to one another according to the sorted set's ordering

`NullPointerException` - if the specified sorted set or any of its elements are null

Method Detail

add

```
public boolean add(E e)
```

Inserts the specified element into this priority queue.

Specified by:

`add` in interface `Collection<E>`

Specified by:

`add` in interface `Queue<E>`

Overrides:

`add` in class `AbstractQueue<E>`

Parameters:

`e` - the element to add

Returns:

`true` (as specified by `Collection.add(E)`)

Throws:

`ClassCastException` - if the specified element cannot be compared with elements currently in this priority queue according to the priority queue's ordering

`NullPointerException` - if the specified element is null

offer

```
public boolean offer(E e)
```

Inserts the specified element into this priority queue.

Specified by:

`offer` in interface `Queue<E>`

Parameters:

`e` - the element to add

Returns:

`true` (as specified by `Queue.offer(E)`)

Throws:

`ClassCastException` - if the specified element cannot be compared with elements currently in this priority queue according to the priority queue's ordering

`NullPointerException` - if the specified element is null

peek

```
public E peek()
```

Description copied from interface: `Queue`

Retrieves, but does not remove, the head of this queue, or returns `null` if this queue is empty.

Specified by:

`peek` in interface `Queue<E>`

Returns:

the head of this queue, or null if this queue is empty

remove

```
public boolean remove(Object o)
```

Removes a single instance of the specified element from this queue, if it is present. More formally, removes an element `e` such that `o.equals(e)`, if this queue contains one or more such elements. Returns `true` if and only if this queue contained the specified element (or equivalently, if this queue changed as a result of the call).

Specified by:

`remove` in interface `Collection<E>`

Overrides:

`remove` in class `AbstractCollection<E>`

Parameters:

`o` - element to be removed from this queue, if present

Returns:

`true` if this queue changed as a result of the call

contains

```
public boolean contains(Object o)
```

Returns `true` if this queue contains the specified element. More formally, returns `true` if and only if this queue contains at least one element `e` such that `o.equals(e)`.

Specified by:

`contains` in interface `Collection<E>`

Overrides:

`contains` in class `AbstractCollection<E>`

Parameters:

`o` - object to be checked for containment in this queue

Returns:

`true` if this queue contains the specified element

toArray

```
public Object[] toArray()
```

Returns an array containing all of the elements in this queue. The elements are in no particular order.

The returned array will be "safe" in that no references to it are maintained by this queue. (In other words, this method must allocate a new array). The caller is thus free to modify the returned array.

This method acts as bridge between array-based and collection-based APIs.

Specified by:

`toArray` in interface `Collection<E>`

Overrides:

`toArray` in class `AbstractCollection<E>`

Returns:

an array containing all of the elements in this queue

toArray

```
public <T> T[] toArray(T[] a)
```

Returns an array containing all of the elements in this queue; the runtime type of the returned array is that of the specified array. The returned array elements are in no particular order. If the queue fits in the specified array, it is returned therein. Otherwise, a new array is allocated with the runtime type of the specified array and the size of this queue.

If the queue fits in the specified array with room to spare (i.e., the array has more elements than the queue), the element in the array immediately following the end of the collection is set to `null`.

Like the `toArray()` method, this method acts as bridge between array-based and collection-based APIs. Further, this method allows precise control over the runtime type of the output array, and may, under certain circumstances, be used to save allocation costs.

Suppose `x` is a queue known to contain only strings. The following code can be used to dump the queue into a newly allocated array of `String`:

```
  String[] y = x.toArray(new String[0]);
```

Note that `toArray(new Object[0])` is identical in function to `toArray()`.

Specified by:

`toArray` in interface `Collection<E>`

Overrides:

`toArray` in class `AbstractCollection<E>`

Type Parameters:

`T` - the runtime type of the array to contain the collection

Parameters:

`a` - the array into which the elements of the queue are to be stored, if it is big enough; otherwise, a new array of the same runtime type is allocated for this purpose.

Returns:

an array containing all of the elements in this queue

Throws:

`ArrayStoreException` - if the runtime type of the specified array is not a supertype of the runtime type of every element in this queue

`NullPointerException` - if the specified array is null

iterator

`public Iterator<E> iterator()`

Returns an iterator over the elements in this queue. The iterator does not return the elements in any particular order.

Specified by:

`iterator in interface Iterable<E>`

Specified by:

`iterator in interface Collection<E>`

Specified by:

`iterator in class AbstractCollection<E>`

Returns:

`an iterator over the elements in this queue`

size

`public int size()`

Description copied from interface: `Collection`

Returns the number of elements in this collection. If this collection contains more than `Integer.MAX_VALUE` elements, returns `Integer.MAX_VALUE`.

Specified by:

`size in interface Collection<E>`

Specified by:

`size in class AbstractCollection<E>`

Returns:

`the number of elements in this collection`

clear

`public void clear()`

Removes all of the elements from this priority queue. The queue will be empty after this call returns.

Specified by:

`clear in interface Collection<E>`

Overrides:

`clear in class AbstractQueue<E>`

poll

`public E poll()`

Description copied from interface: `Queue`

Retrieves and removes the head of this queue, or returns `null` if this queue is empty.

Specified by:

`poll` in interface `Queue<E>`

Returns:

the head of this queue, or null if this queue is empty

comparator

```
public Comparator<? super E> comparator()
```

Returns the comparator used to order the elements in this queue, or `null` if this queue is sorted according to the natural ordering of its elements.

Returns:

the comparator used to order this queue, or null if this queue is sorted according to the natural ordering of its elements

spliterator

```
public final Spliterator<E> spliterator()
```

Creates a *late-binding* and *fail-fast* `Spliterator` over the elements in this queue.

The `Spliterator` reports `Spliterator.SIZED`, `Spliterator.SUBSIZED`, and `Spliterator.NONNULL`. Overriding implementations should document the reporting of additional characteristic values.

Specified by:

`spliterator` in interface `Iterable<E>`

Specified by:

`spliterator` in interface `Collection<E>`

Returns:

a Spliterator over the elements in this queue

Since:

1.8

Submit a bug or feature
For further API reference and developer documentation, see Java SE Documentation. That documentation

contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.