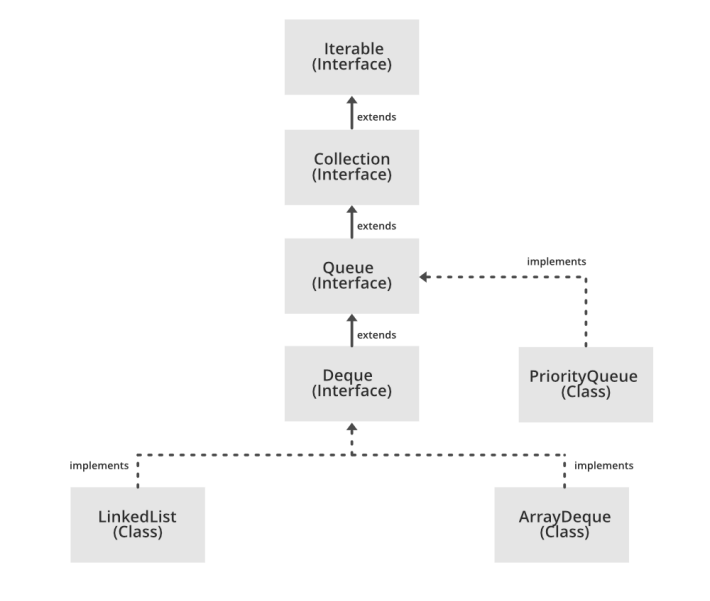
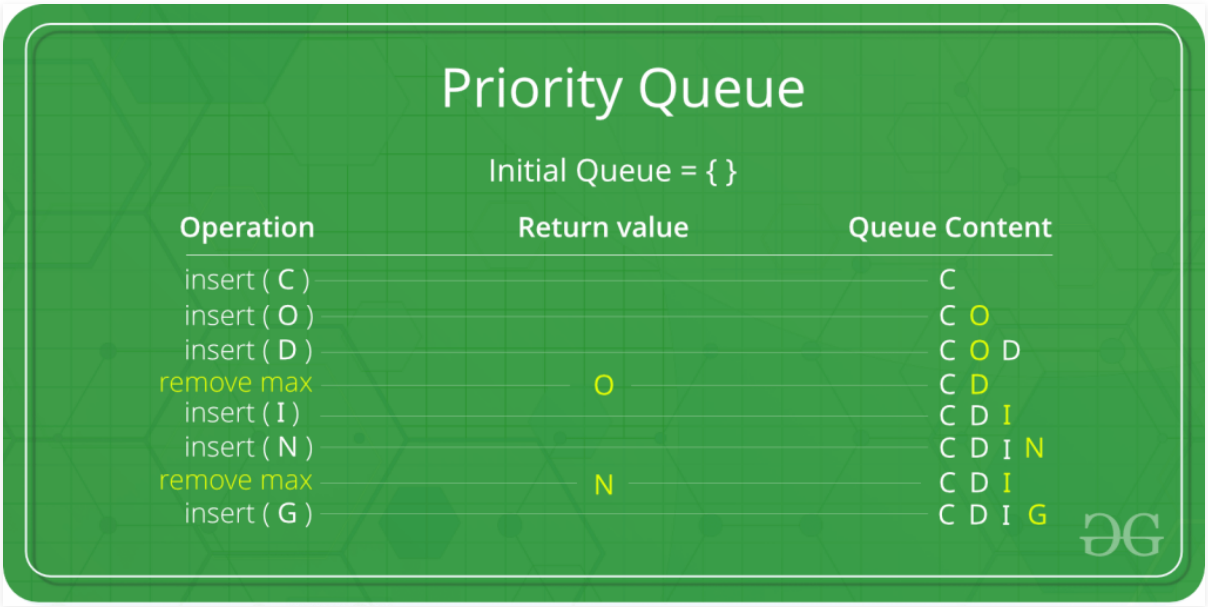
PriorityQueue in Java

Difficulty Level : Medium Last Updated : 27 Oct, 2021

A PriorityQueue is used when the objects are supposed to be processed based on the priority. It is known that a Queue follows the First-In-First-Out algorithm, but sometimes the elements of the queue are needed to be processed according to the priority, that's when the PriorityQueue comes into play. The PriorityQueue is based on the priority heap. The elements of the priority queue are ordered according to the natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used.



In the below priority queue, an element with maximum ASCII value will have the highest priority.



Declaration:

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

where E is the type of elements held in this queue
```

The class implements **Serializable**, **Iterable<E>**, **Collection<E>**, Queue<E> interfaces.

Few **important points on Priority Queue** are as follows:

- PriorityQueue doesn't permit null.
- We can't create PriorityQueue of Objects that are non-comparable
- PriorityQueue are unbound queues.
- 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.
- Since PriorityQueue is not thread-safe, so java provides PriorityBlockingQueue class that implements the BlockingQueue interface to use in java multithreading environment.
- The queue retrieval operations poll, remove, peek, and element access the element at the head of the queue.
- It provides O(log(n)) time for add and poll methods.
- It inherits methods from **AbstractQueue**, **AbstractCollection**, **Collection** and **Object** class.

Constructors:

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

```
PriorityQueue<E> pq = new PriorityQueue<E>();
```

2. PriorityQueue(Collection<E> c): Creates a PriorityQueue containing the elements in the specified collection.

PriorityQueue<E> pq = new PriorityQueue<E>(Collection<E> c);

3. PriorityQueue(int initialCapacity): Creates a PriorityQueue with the specified initial capacity that orders its elements according to their natural ordering.

PriorityQueue<E> pq = new PriorityQueue<E>(int initialCapacity);

4. PriorityQueue(int initialCapacity, Comparator<E> comparator): Creates a PriorityQueue with the specified initial capacity that orders its elements according to the specified comparator.

PriorityQueue<E> pq = new PriorityQueue(int initialCapacity, Comparator<E> comp arator);

5. PriorityQueue(PriorityQueue<E> c): Creates a PriorityQueue containing the elements in the specified priority queue.

PriorityQueue<E> pq = new PriorityQueue(PriorityQueue<E> c);

6. PriorityQueue(SortedSet<E> c): Creates a PriorityQueue containing the elements in the specified sorted set.

PriorityQueue<E> pq = new PriorityQueue<E>(SortedSet<E> c);

Example:

The example below explains the following basic operations of the priority queue.

- boolean add(E element): This method inserts the specified element into this priority queue.
- public peek(): This method retrieves, but does not remove, the head of this queue, or returns null if this queue is empty

returns null if this queue is empty.

- public poll(): This method retrieves and removes the head of this queue, or returns null if this queue is empty.

```
// Java program to demonstrate the
// working of PriorityQueue
import java.util.*;

class PriorityQueueDemo {

      // Main Method
    public static void main(String args[])
    {
        // Creating empty priority queue
        PriorityQueue<Integer> pQueue = new PriorityQueue<Integer>();

        // Adding items to the pQueue using add()
        pQueue.add(10);
        pQueue.add(20);
        pQueue.add(15);

        // Printing the top element of PriorityQueue
        System.out.println(pQueue.peek());

        // Printing the top element and removing it
        // from the PriorityQueue container
        System.out.println(pQueue.poll());

        // Printing the top element again
        System.out.println(pQueue.peek());
    }
}
```

Output:

```
10
10
15
```

Operations on PriorityQueue

Let's see how to perform a few frequently used operations on the Priority Queue class.

1. Adding Elements: In order to add an element in a priority queue, we can use the add() method. The insertion order is not retained in the PriorityQueue. The elements are stored based on the priority order which is ascending by default.

```
// Java program to add elements
// to a PriorityQueue
import java.util.*;
import java.io.*;

public class PriorityQueueDemo {

    public static void main(String args[])
    {
        PriorityQueue<String> pq = new PriorityQueue<>();

        pq.add("Geeks");
        pq.add("For");
        pq.add("Geeks");

        System.out.println(pq);
    }
}
```

Output:

```
[For, Geeks, Geeks]
```

2. Removing Elements: In order to remove an element from a priority queue, we can use the remove() method. If there are multiple such objects, then the first occurrence of the object is removed. Apart from that, the poll() method is also used to remove the head and return it.

```
// Java program to remove elements
// from a PriorityQueue

import java.util.*;
import java.io.*;

public class PriorityQueueDemo {

    public static void main(String args[])
    {
        PriorityQueue<String> pq = new PriorityQueue<>();

        pq.add("Geeks");
        pq.add("For");
        pq.add("Geeks");

        System.out.println("Initial PriorityQueue " + pq);

          // using the method
        pq.remove("Geeks");

        System.out.println("After Remove - " + pq);

        System.out.println("Poll Method - " + pq.poll());

        System.out.println("Final PriorityQueue - " + pq);
    }
}
```

Output:

```
Initial PriorityQueue [For, Geeks, Geeks]
After Remove - [For, Geeks]
Poll Method - For
Final PriorityQueue - [Geeks]
```

3. Accessing the elements: Since Queue follows the First In First Out principle, we can access only the head of the queue. To access elements from a priority queue, we can use the peek() method.

```
// Java program to access elements
// from a PriorityQueue
import java.util.*;

class PriorityQueueDemo {

      // Main Method
    public static void main(String[] args)
    {

        // Creating a priority queue
        PriorityQueue<String> pq = new PriorityQueue<>();
        pq.add("Geeks");
        pq.add("For");
        pq.add("Geeks");
        System.out.println("PriorityQueue: " + pq);

        // Using the peek() method
        String element = pq.peek();
        System.out.println("Accessed Element: " + element);
    }
}
```

Output:

```
PriorityQueue: [For, Geeks, Geeks]
Accessed Element: For
```

4. Iterating the PriorityQueue: There are multiple ways to iterate through the PriorityQueue. The most famous way is converting the queue to the array and traversing using the for loop. However, the queue also has an inbuilt iterator which can be used to iterate through the queue.

```
// Java program to iterate elements
// to a PriorityQueue

import java.util.*;

public class PriorityQueueDemo {
```

```
    // Main Method
    public static void main(String args[])
    {
        PriorityQueue<String> pq = new PriorityQueue<>();

        pq.add("Geeks");
        pq.add("For");
        pq.add("Geeks");

        Iterator iterator = pq.iterator();

        while (iterator.hasNext()) {
            System.out.print(iterator.next() + " ");
        }
    }
}
```

Output:

```
For Geeks Geeks
```

Methods in PriorityQueue class

METHOD	DESCRIPTION
add(E e)	Inserts the specified element into this priority queue.
clear()	Removes all of the elements from this priority queue.
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.
contains(Object o)	Returns true if this queue contains the specified element.
forEach (Consumer<? super E> action)	Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.
iterator()	Returns an iterator over the elements in this queue.

METHOD	DESCRIPTION
offer(E e)	Inserts the specified element into this priority queue.
remove(Object o)	Removes a single instance of the specified element from this queue, if it is present.
removeAll (Collection<?> c)	Removes all of this collection's elements that are also contained in the specified collection (optional operation).
removeIf (Predicate<? super E> filter)	Removes all of the elements of this collection that satisfy the given predicate.
retainAll (Collection<?> c)	Retains only the elements in this collection that are contained in the specified collection (optional operation).
spliterator()	Creates a late-binding and fail-fast Spliterator over the elements in this queue.
toArray()	Returns an array containing all of the elements in this queue.
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 Declared in class java.util.AbstractQueue

METHOD	DESCRIPTION
addAll(Collection<? extends E> c)	Adds all of the elements in the specified collection to this queue.
element()	Retrieves, but does not remove, the head of this queue.
remove()	Retrieves and removes the head of this queue.

Methods Declared in class java.util.AbstractCollection

METHOD	DESCRIPTION

METHOD	DESCRIPTION
containsAll(Collection<? > c)	Returns true if this collection contains all of the elements in the specified collection.
isEmpty()	Returns true if this collection contains no elements.
toString()	Returns a string representation of this collection.

Methods Declared in interface java.util.Collection

METHOD	DESCRIPTION
containsAll(Collection<? > c)	Returns true if this collection contains all of the elements in the specified collection.
equals(Object o)	Compares the specified object with this collection for equality.
hashCode()	Returns the hash code value for this collection.
isEmpty()	Returns true if this collection contains no elements.
parallelStream()	Returns a possibly parallel Stream with this collection as its source.
size()	Returns the number of elements in this collection.
stream()	Returns a sequential Stream with this collection as its source.
toArray(IntFunction<T[]> generator)	Returns an array containing all of the elements in this collection, using the provided generator function to allocate the returned array.

Methods Declared in interface java.util.Queue

METHOD	DESCRIPTION
peek()	Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.

METHOD	DESCRIPTION
poll()	Retrieves and removes the head of this queue, or returns null if this queue is empty.

Applications:

- Implementing Dijkstra's and Prim's algorithms.
- Maximize array sum after K negations

Related Articles:

- Java.util.PriorityQueue class in Java
- Implement PriorityQueue through Comparator in Java

This article is contributed by **Mehak Kumar.** Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.