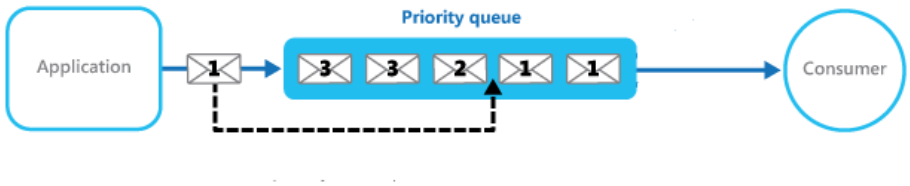
HowToDoInJava

Java PriorityQueue

Last Updated: November 1, 2021 By: Lokesh Gupta Java Collections Java Collections, Java Queue

Java PriorityQueue class is a queue data structure implementation that processes the queue items based on their **priorities**. Note that `PriorityQueue` is different from other standard queues which implement the FIFO (First-In-First-Out) algorithm.



Priority Queue

In *PriorityQueue*, the added *items are retrieved according to their priorities*. By default, the priority is determined by objects' natural ordering. Default priority can be overridden by a Comparator provided at queue construction time.

Important

It is important to note that the items of a `PriorityQueue` may not be sorted by their priorities. However, items are always retrieved in sorted order.

Table Of Contents

1. How to Create and Use PriorityQueue

To create a priority queue, use one of the constructors. We can optionally pass the *Comparator* instance for custom ordering of the items.

A Simple Example of PriorityQueue

```
import java.util.PriorityQueue;

public class PriorityQueueExample
{
    public static void main(String[] args)
    {
        // Creating a priority queue
        PriorityQueue<Integer> numbers = new PriorityQueue<>();

        // Using the add() method
        numbers.add(3);
        numbers.add(2);
        System.out.println("PriorityQueue: " + numbers);

        // Using the offer() method
        numbers.offer(1);
        System.out.println("PriorityQueue: " + numbers);

        //Retrieve the items
        System.out.println("Item: " + numbers.poll());
        System.out.println("Item: " + numbers.poll());
        System.out.println("Item: " + numbers.poll());
    }
}
```

Output

```
PriorityQueue: [2, 3]
PriorityQueue: [1, 3, 2]

Item: 1
Item: 2
Item: 3
```

Notice the sequence of items in the priority queue is not always in sorted order, but when we retrieved the items then items are retrieved always in sorted order.

2. PriorityQueue Features

Let's note down a few important features of the *PriorityQueue*.

- *PriorityQueue* is an **unbounded queue** that grows dynamically.
- The **default initial capacity is `'11'`** which can be overridden using **initialCapacity** parameter in appropriate constructor.
- It **does not allow NULL** objects.
- The **queue items must be** Comparable, to determine their priorities.
- By deafult, the items in the priority queue are **ordered in natural order**.
- A *Comparator* can be used for custom ordering of objects in the queue.

- *PriorityQueue* relying on natural ordering does not permit insertion of non-comparable objects (doing so may result in `ClassCastException`).
- The **queue retrieval operations** `poll`, `remove`, `peek`, and `element` access the element at the **head** of the queue.
- The *head of the PriorityQueue is the least element* based on the natural ordering or the *Comparator* based ordering.
- If multiple objects are present of same priority then queue can poll any one of them randomly.
- PriorityQueue is **not thread safe**. Use `PriorityBlockingQueue` in concurrent environment.
- It provides **O(log(n)) time performance** for *add* and *poll* methods.
- 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())`.

3. PriorityQueue Example with Custom Objects

Let's see how the priorities of the items impact the `add()` and `remove()` operations. In the given examples, the queue items are of type `Employee`.

`Employee` class implements **Comparable** interface which makes objects comparable by Employee `'id'` field, by default.

Employee.java

```
public class Employee implements Comparable<Employee> {

    private Long id;
    private String name;
    private LocalDate dob;

    public Employee(Long id, String name, LocalDate dob) {
        super();
        this.id = id;
        this.name = name;
        this.dob = dob;
    }

    @Override
    public int compareTo(Employee emp) {
        return this.getId().compareTo(emp.getId());
    }

    //Getters and setters

    @Override
    public String toString() {
        return "Employee [id=" + id + ", name=" + name + ", dob=" + dob + "]";
    }
}
```

3.1. PriorityQueue with Natural Ordering

Java PriorityQueue example to add and poll elements while the items are compared based on their natural ordering. Here the natural ordering is based on the provided *compareTo()* method which compares the employees by *id*.

```
PriorityQueue<Employee> priorityQueue = new PriorityQueue<>();

priorityQueue.add(new Employee(1l, "AAA", LocalDate.now()));
priorityQueue.add(new Employee(4l, "CCC", LocalDate.now()));
priorityQueue.add(new Employee(5l, "BBB", LocalDate.now()));
priorityQueue.add(new Employee(2l, "FFF", LocalDate.now()));
priorityQueue.add(new Employee(3l, "DDD", LocalDate.now()));
priorityQueue.add(new Employee(6l, "EEE", LocalDate.now()));

while(true)
{
    Employee e = priorityQueue.poll();
    System.out.println(e);

    if(e == null) break;
}
```

Program Output.

Output

```
Employee [id=1, name=AAA, dob=2021-11-01]
Employee [id=2, name=FFF, dob=2021-11-01]
Employee [id=3, name=DDD, dob=2021-11-01]
Employee [id=4, name=CCC, dob=2021-11-01]
Employee [id=5, name=BBB, dob=2021-11-01]
Employee [id=6, name=EEE, dob=2021-11-01]
```

3.2. PriorityQueue with Custom Ordering

Let's redefine the custom ordering using Java 8 lambda based comparator syntax and verify the result.

PriorityQueue Custom Ordering Example

```
//Comparing by employee names
Comparator<Employee> nameSorter = Comparator.comparing(Employee::getName);

PriorityQueue<Employee> priorityQueue = new PriorityQueue<>( nameSorter );

priorityQueue.add(new Employee(1l, "AAA", LocalDate.now()));
priorityQueue.add(new Employee(4l, "CCC", LocalDate.now()));
priorityQueue.add(new Employee(5l, "BBB", LocalDate.now()));
priorityQueue.add(new Employee(2l, "FFF", LocalDate.now()));
priorityQueue.add(new Employee(3l, "DDD", LocalDate.now()));
priorityQueue.add(new Employee(6l, "EEE", LocalDate.now()));
```

```
while(true)
{
    Employee e = priorityQueue.poll();
    System.out.println(e);

    if(e == null) break;
}
```

Program Output.

Output

```
Employee [id=1, name=AAA, dob=2018-10-31]
Employee [id=5, name=BBB, dob=2018-10-31]
Employee [id=4, name=CCC, dob=2018-10-31]
Employee [id=3, name=DDD, dob=2018-10-31]
Employee [id=6, name=EEE, dob=2018-10-31]
Employee [id=2, name=FFF, dob=2018-10-31]
```

4. PriorityQueue Constructors

PriorityQueue class provides 6 different ways to construct a priority queue in Java.

1. **PriorityQueue()** : constructs empty queue with the default initial capacity (11) that orders its elements according to their natural ordering.
2. **PriorityQueue(Collection c)** : constructs empty queue containing the elements in the specified collection.
3. **PriorityQueue(int initialCapacity)** : constructs empty queue with the specified initial capacity that orders its elements according to their natural ordering.
4. **PriorityQueue(int initialCapacity, Comparator comparator)** : constructs empty queue with the specified initial capacity that orders its elements according to the specified comparator.
5. **PriorityQueue(PriorityQueue c)** : constructs empty queue containing the elements in the specified priority queue.
6. **PriorityQueue(SortedSet c)** : constructs empty queue containing the elements in the specified sorted set.

5. PriorityQueue Methods

PriorityQueue class has below given important methods, we should know.

5.1. Adding Items

- **boolean add(object)** : Inserts the specified element into this priority queue. If the queue is full, it throws an exception.

- **boolean offer(object)** : Inserts the specified element into this priority queue. If the queue is full, it returns `false`.

5.2. Accessing Items

- **Object element()** : Retrieves, but does not remove, the head of this queue, or throws *NoSuchElementException* if this queue is empty.
- **Object peek()** : Retrieves, but does not remove, the head of this queue, or returns *null* if this queue is empty.

5.3. Removing Items

- **boolean remove(object)** : Removes a single instance of the specified element from this queue, if it is present.
- **Object poll()** : Retrieves and removes the head of this queue, or returns null if this queue is empty.
- **void clear()** : Removes all of the elements from this priority queue.

5.4. Other Methods

- **Comparator 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 iterator()** : Returns an iterator over the elements in this queue.
- **int size()** : Returns the number of elements in this queue.
- **Object[] toArray()** : Returns an array containing all of the elements in this queue.

6. Conclusion

In this **Java queue tutorial**, we learned to use **PriorityQueue class** which is able to store elements either by default natural ordering or custom ordering specified a comparator.

We also learned a few important methods and constructors of `PriorityQueue` class.

Drop me your questions in the comments section.

Happy Learning !!

Recommended Reading:

1. Java TransferQueue – Java LinkedTransferQueue class
2. Java Iterate List Examples
3. Java TreeMap class
4. Java Spliterator interface
5. Java PriorityBlockingQueue class
6. Java ArrayBlockingQueue class
7. Java CopyOnWriteArrayList class
8. Java CopyOnWriteArraySet class
9. Create List with Single Element
0. Java Comparator Interface

2 thoughts on “Java PriorityQueue”

Sid

November 1, 2019 at 6:07 am

Awesome. Learnt an application of lamda expession

Reply

nitinsridar

July 17, 2019 at 8:50 am

Please add use case and performance

Reply

Leave a Comment

Name *

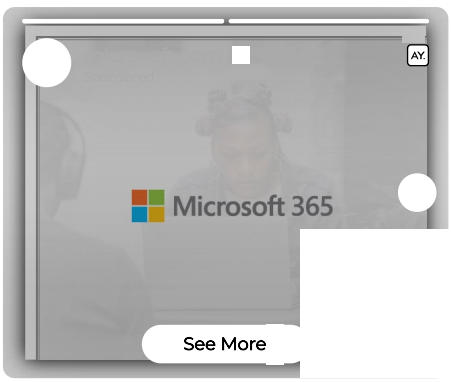
Email *

Website

Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search ...



- About Me
- Contact Us
- Privacy policy
- Advertise
- Guest Posts

Blogs

REST API Tutorial