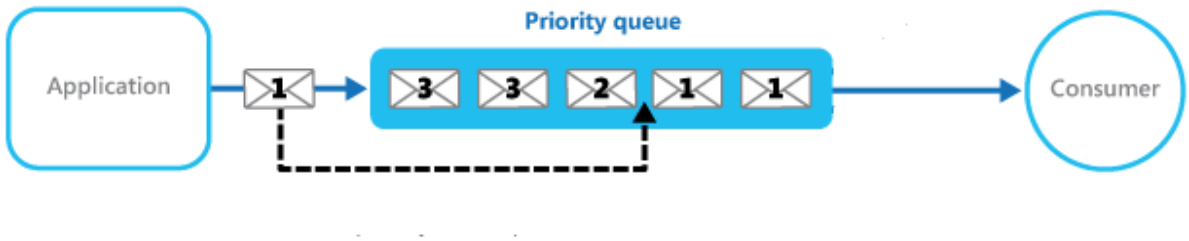
HowToDoInJava

Java PriorityBlockingQueue class

Last Updated: December 26, 2020 | By: Lokesh Gupta | Java Collections | Blocking IO, Java Collections

Java PriorityBlockingQueue class is **concurrent** blocking queue data structure implementation in which objects are processed based on their **priority**. The "blocking" part of the name is added to imply the **thread will block waiting until there's an item available on the queue**.

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



Priority Blocking Queue

1. PriorityBlockingQueue Features

Let's note down few important points on the PriorityBlockingQueue.

- PriorityBlockingQueue is an unbounded queue and grows dynamically. The default initial capacity is `'11'` which can be overridden using **initialCapacity** parameter in appropriate constructor.
- It supplies blocking retrieval operations.
- It does not allow NULL objects.
- Objects added to PriorityBlockingQueue MUST be comparable otherwise it throws `ClassCastException`.
- The objects of the priority queue are ordered **by default in natural order**.
- A Comparator can be used for custom ordering of objects in the queue.
- The **head** of the priority queue is the **least** element based on the natural ordering or comparator based ordering. When we poll the queue, it returns the head object from the queue.
- If multiple objects are present of same priority the it can poll any one of them randomly.
- PriorityBlockingQueue is **thread safe**.
- The Iterator provided in method **iterator()** is not guaranteed to traverse the elements of the PriorityBlockingQueue in any particular order. If you need ordered traversal, consider using **Arrays.sort(pbq.toArray()).**
- The **drainTo()** can be used to remove some or all elements in priority order and place them in another collection.

2. Java PriorityBlockingQueue Example

Let's see how object's ordering impacts the add and remove operations in PriorityBlockingQueue. In given examples, the objects 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 + "]";
    }
}
```

2.1. Natural Ordering

Java PriorityBlockingQueue example to add and poll elements which are compared based on their natural ordering.

```
PriorityBlockingQueue Example

PriorityBlockingQueue<Employee> PriorityBlockingQueue = new PriorityBlockingQueue<

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

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

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

```
}
```

Program Output.

```
Console

Employee [id=1, name=AAA, dob=2018-10-31]
Employee [id=2, name=FFF, 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]
```

2.2. PriorityBlockingQueue Comparator example

Let's redefine the custom ordering using Java 8 lambda based comparator syntax and verify the result. We are using constructor **PriorityBlockingQueue(int initialCapacity, Comparator comparator)**.

```
PriorityBlockingQueue Example

//Comparator for name field
Comparator<Employee> nameSorter = Comparator.comparing(Employee::getName);

PriorityBlockingQueue<Employee> PriorityBlockingQueue = new PriorityBlockingQueue<

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

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

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

Program Output.

```
Console

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]
```

2.3. PriorityBlockingQueue drainTo() example

Java example to use drianTo() method to take multiple elements from queue in one statement.

```
PriorityBlockingQueue Example

PriorityBlockingQueue<Integer> priorityBlockingQueue = new PriorityBlockingQueue<>

priorityBlockingQueue.add(1);
priorityBlockingQueue.add(3);
priorityBlockingQueue.add(2);
priorityBlockingQueue.add(6);
priorityBlockingQueue.add(4);
priorityBlockingQueue.add(5);

ArrayList<Integer> list = new ArrayList<>();

//Drain first 3 elements
priorityBlockingQueue.drainTo(list, 3);

System.out.println(list);

//Drain all elements
priorityBlockingQueue.drainTo(list);

System.out.println(list);
```

Program Output.

```
Console
```

```
[1, 2, 3]
[1, 2, 3, 4, 5, 6]
```

2.4. PriorityBlockingQueue blocking retrieval example

Java example to take elements from PriorityBlockingQueue using blocking retrieval. A thread will wait until there is an element present in the queue.

In given example, a thread is waiting on queue in infinite loop using **take()** method. It wait for 1 seconds before checking again. As soon as we add elements to queue, it poll the item and print to the console.

```
PriorityBlockingQueue Example

import java.util.concurrent.PriorityBlockingQueue;
import java.util.concurrent.TimeUnit;

public class PriorityQueueExample
{
    public static void main(String[] args) throws InterruptedException
    {
        PriorityBlockingQueue<Integer> priorityBlockingQueue = new PriorityBlockin

        new Thread(() ->
        {
          System.out.println("Waiting to poll ...");

          try
          {
              while(true)
              {
                  Integer poll = priorityBlockingQueue.take();
                  System.out.println("Polled : " + poll);

                  Thread.sleep(TimeUnit.SECONDS.toMillis(1));
              }

          } catch (InterruptedException e) {
              e.printStackTrace();
          }

        }).start();

        Thread.sleep(TimeUnit.SECONDS.toMillis(2));
        priorityBlockingQueue.add(1);
```

```
            Thread.sleep(TimeUnit.SECONDS.toMillis(2));
            priorityBlockingQueue.add(2);

            Thread.sleep(TimeUnit.SECONDS.toMillis(2));
            priorityBlockingQueue.add(3);
        }
    }
```

Program Output.

```
Console

Waiting to poll ...
Polled : 1
Polled : 2
Polled : 3
```

3. Java PriorityBlockingQueue Constructors

PriorityBlockingQueue class provides 4 different ways to construct a priority queue in Java.

- **PriorityBlockingQueue()** : constructs empty queue with the default initial capacity (11) that orders its elements according to their natural ordering.
- **PriorityBlockingQueue(Collection c)** : constructs empty queue containing the elements in the specified collection.
- **PriorityBlockingQueue(int initialCapacity)** : constructs empty queue with the specified initial capacity that orders its elements according to their natural ordering.
- **PriorityBlockingQueue(int initialCapacity, Comparator comparator)** : constructs empty queue with the specified initial capacity that orders its elements according to the specified comparator.

4. Java PriorityBlockingQueue Methods

PriorityBlockingQueue class has below given important methods, you should know.

- **boolean add(object)** : Inserts the specified element into this priority queue.
- **boolean offer(object)** : Inserts the specified element into this priority queue.
- **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, waiting up to the specified wait time if necessary for an element to become available.
- **Object poll(timeout, timeUnit)** : Retrieves and removes the head of this queue, waiting up to the specified wait time if necessary for an element to become available.
- **Object take()** : Retrieves and removes the head of this queue, waiting if necessary until an element becomes available.
- **void put(Object o)** : Inserts the specified element into this priority queue.
- **void clear()** : Removes all of the elements from this priority queue.
- **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.
- **int drainTo(Collection c)** : Removes all available elements from this queue and adds them to the given collection.
- **int drainTo(Collection c, int maxElements)** : Removes at most the given number of available elements from this queue and adds them to the given collection.
- **int remainingCapacity()** : Always returns `Integer.MAX_VALUE` because a PriorityBlockingQueue is not capacity constrained.
- **Object[] toArray()** : Returns an array containing all of the elements in this queue.

> Please note the **difference between take() and poll()** methods. The poll() retrieves and removes the head of this queue, or returns null if this queue is empty. It is not blocking operation.
>
> The take() retrieves and removes the head of this queue, waiting if necessary until an element becomes available. It is blocking operation.

5. Conclusion

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

We also learned few important methods and constructors of PriorityBlockingQueue class.

Drop me your questions in comments section.

Happy Learning !!

References:

PriorityBlockingQueue Class Java Docs

Recommended Reading:

1. [Solved]: javax.xml.bind.JAXBException: class java.util.ArrayList nor any of its super class is known to this context
2. Java ArrayBlockingQueue class
3. Java TransferQueue – Java LinkedTransferQueue class
4. Java TreeMap class
5. Java CopyOnWriteArrayList class
6. Java CopyOnWriteArraySet class
7. Java LinkedHashMap class
8. Java LinkedHashSet class
9. Java TreeSet class
0. Java LinkedList class

1 thought on “Java PriorityBlockingQueue class”

Pon

May 16, 2019 at 7:31 pm

missing information about difference between add, offer, put

Reply

Leave a Comment

Name *

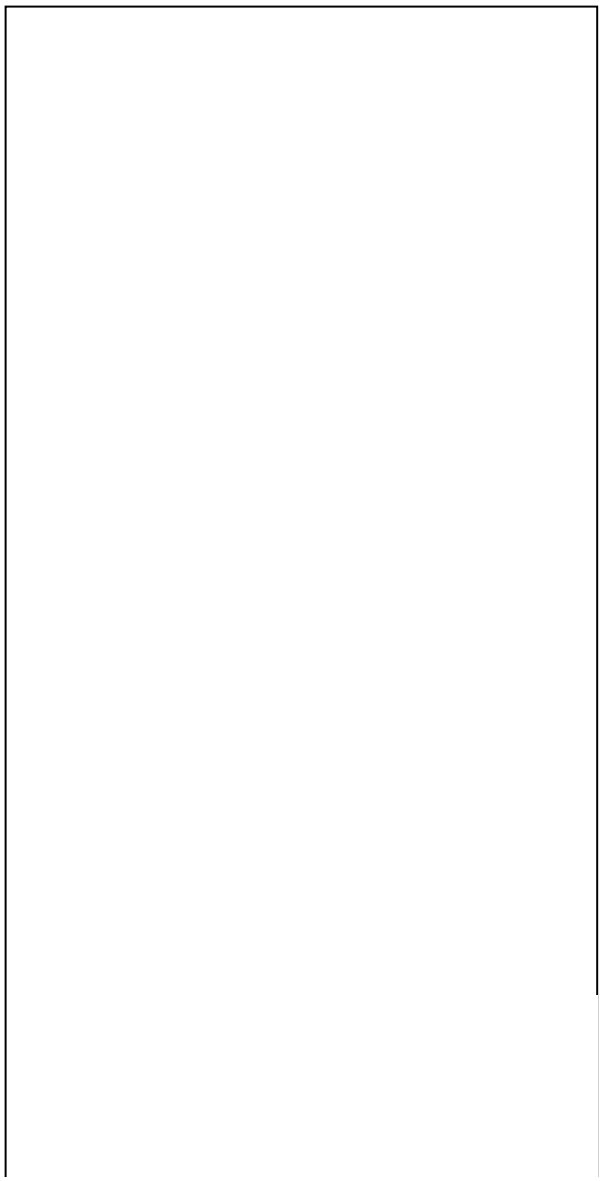
Email *

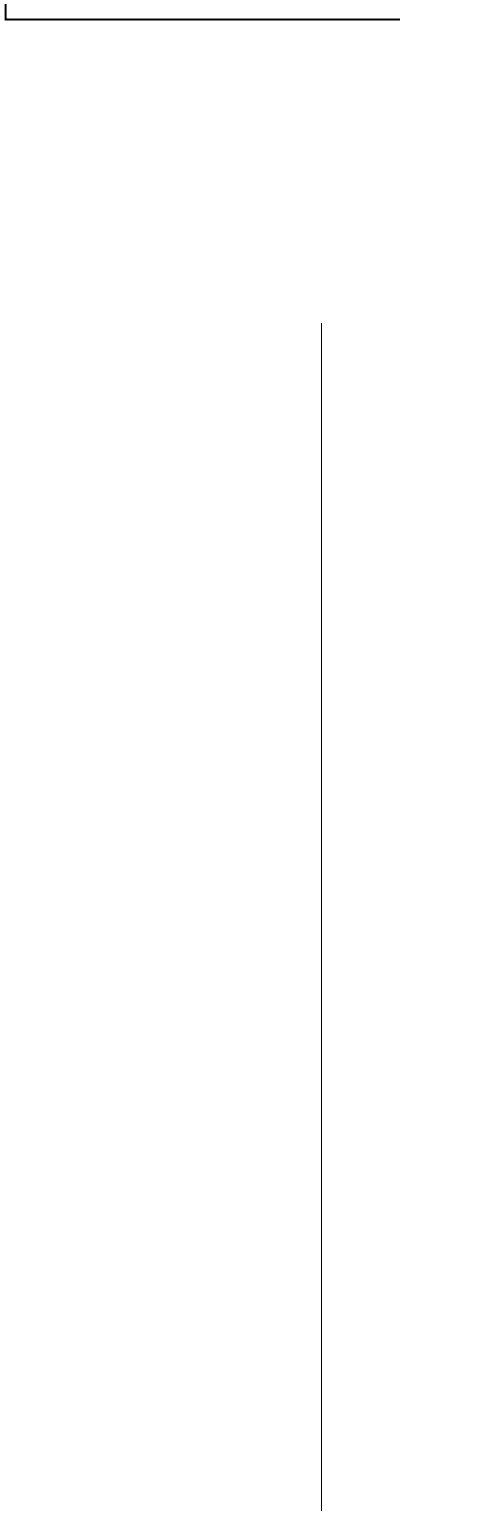
Website

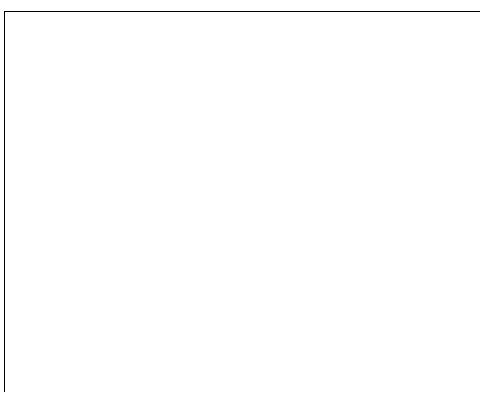
Add me to your newsletter and keep me updated whenever you publish new blog posts

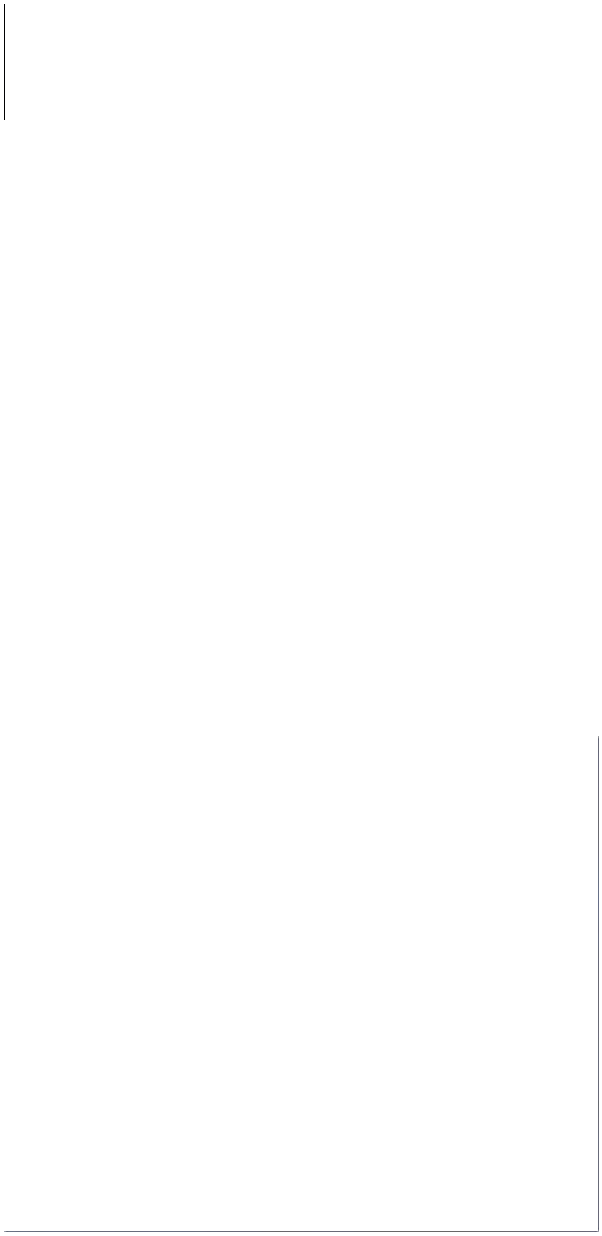
Post Comment

Search …









HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- About Me
- Contact Us
- Privacy policy