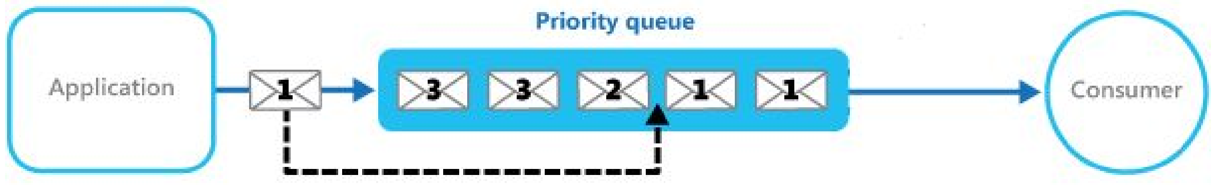
Java PriorityQueue class

Java PriorityQueue class is a queue data structure implementation in which objects are processed based on their priority. It is different from standard queues where FIFO (First-In-First-Out) algorithm is followed.

In a priority queue, added objects are 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.



1. PriorityQueue Features

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

- PriorityQueue is an unbounded queue and grows dynamically. The default initial capacity is `'11'` which can be overridden using initialCapacity parameter in appropriate constructor.
- It does not allow NULL objects.
- Objects added to PriorityQueue MUST be comparable.
- 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.

- PriorityQueue is not thread safe. Use `PriorityBlockingQueue` in concurrent environment.
- It provides O(log(n)) time for add and poll methods.

2. Java PriorityQueue Example

Let’s see how object’s ordering impacts the add and remove operations in PriorityQueue. In given examples, the objects are of type `Employee`. Employee class implements Comparable interface which makes objects comparable by employee `'id'` field, by default

```
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 PriorityQueue example to add and poll elements which are compared based on their natural ordering.

```
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.

```
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. Custom Ordering using Comparator

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

```
//Comparator for name field

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.

```
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]
```

3. Java PriorityQueue Constructors

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

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

4. Java PriorityQueue Methods

PriorityQueue 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, or returns null if this queue is empty.
- 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.
- 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.
- Object[] toArray() : Returns an array containing all of the elements in this queue.

5. 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.