CSCI 2120:
Software Design & Development II

UNIT2: Data management

Collections Framework & Generics
Priority Queue

Overview

1. Introduction
2. Realtime Examples of PriorityQueue in Java
3. Hierarchy of Java PriorityQueue
4. PriorityQueue class declaration
5. Features of PriorityQueue
6. PriorityQueue Constructors
7. PriorityQueue Methods
8. PriorityQueue Examples

Introduction

A **PriorityQueue in Java** is a queue or collection of elements in which elements are stored in order of their priority. It is an abstract data type similar to an ordinary queue except for its removal algorithm.

An ordinary queue is a first-in-first-out (FIFO) data structure. In FIFO, elements are added to the end of the queue and removed from the beginning.

But, In Java `PriorityQueue`, elements are stored in order of their priority. When accessing elements, the element with the highest priority is removed first before the element with lower priority.

PriorityQueue was added in Java 1.5 version and it is part of the Java Collection Framework. It is present in `java.util.PriorityQueue` package.

Realtime Examples of PriorityQueue in Java

1. A typical example of a priority queue is work schedule. Works are added in random order but each work has a priority. Urgent work is assigned the highest priority and is done first.

2. In a hospital, the emergency room assigns priority numbers to patients. The patient with the highest priority is checked up first.

Similarly, Java `PriorityQueue` class provides an implementation of a priority queue. We can create a queue with its own priority queue in java.

Realtime Examples of PriorityQueue in Java

Priority is decided by Comparator provided in the constructor `PriorityQueue(initialCapacity, Comparator)` when the priority queue is instantiated.

If no comparator is provided, `PriorityQueue` orders its elements according to their natural ordering using the `Comparable` interface. In simple words, by default, the priority is based on the natural order of the elements.

For example, if all elements are the type of Integer and no comparator is provided at the time of construction, natural ordering of elements is used to prioritize them.

The element at the head of the priority queue is the smallest element with respect to specified ordering. That is, the element having the smallest value will be assigned with the highest priority and removed first from the queue.

Realtime Examples of PriorityQueue in Java

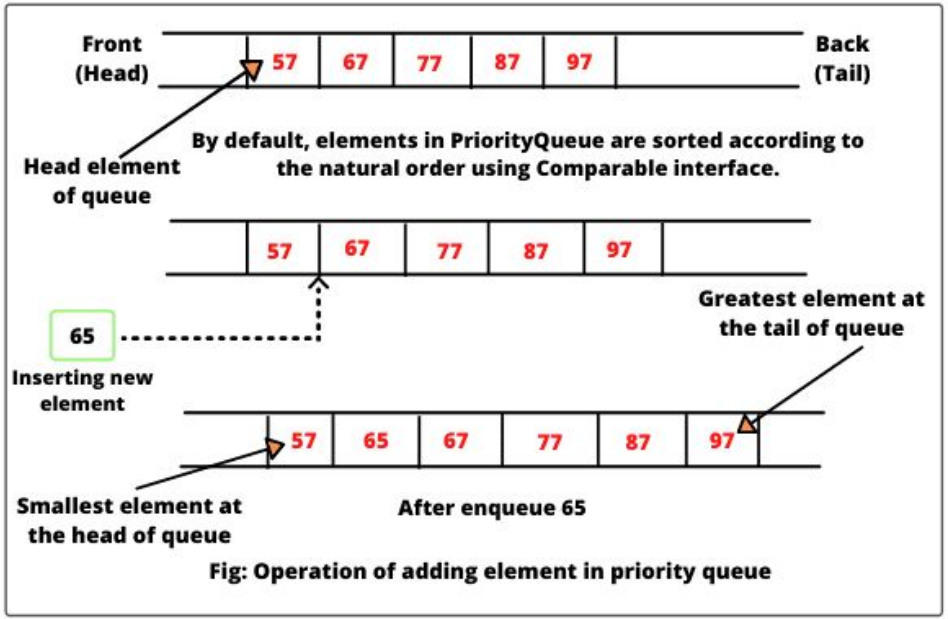
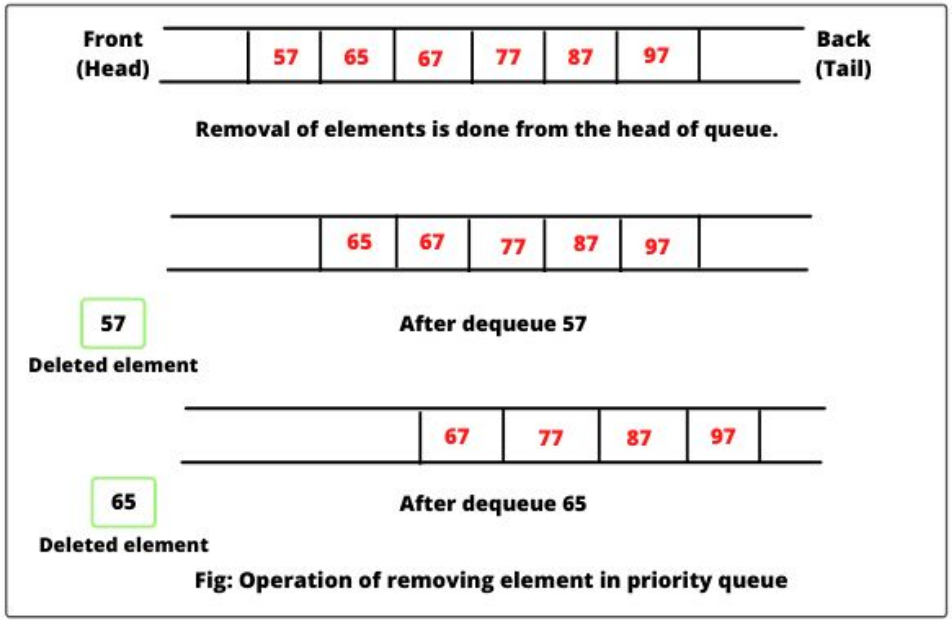


Fig: Operation of adding element in priority queue

Look at the figure where a new element is inserted in the queue and `PriorityQueue` orders its elements according to the natural order.

The element having the greatest value will be assigned with the lowest priority and it will be placed at the tail of priority queue.

Realtime Examples of PriorityQueue in Java



Removal of elements takes place from the front end of the queue. Look at the below figure to understand better.

If multiple elements are tied for the highest priority, one of those elements is arbitrarily chosen as the least element. Similarly, when there is a tie among elements at the tail of the priority queue, it is also arbitrarily chosen.

Hierarchy of PriorityQueue

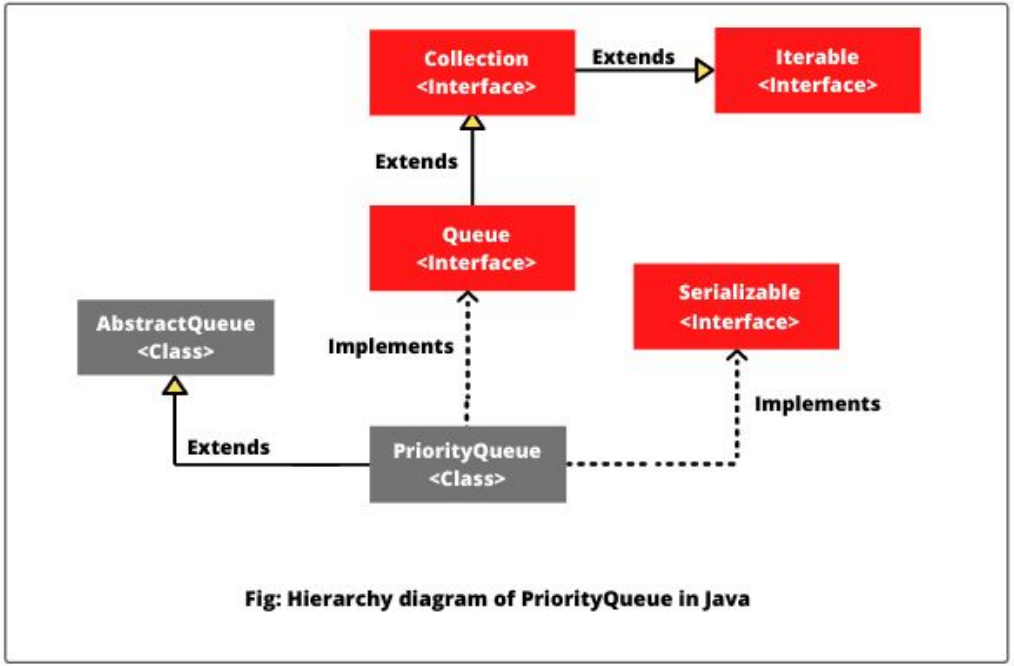


Fig: Hierarchy diagram of PriorityQueue in Java

PriorityQueue extends AbstractQueue class to support a priority-based queue. It implements the Queue interface, Serializable interface but not implements Cloneable.

The hierarchy diagram of PriorityQueue in java is shown in the below figure.

PriorityQueue class declaration

PriorityQueue is a generic class that can be declared as follows:

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

Here, E represents the type of objects stored in the queue.

Features of PriorityQueue

There are few important features of Priority Queue that are as follows:

1. The underlying data structure for implementing PriorityQueue in Java is Binary Heap.

2. PriorityQueue doesn’t permit null elements.

3. It does not allow the insertion of non-comparable objects when relying on natural ordering.

4. PriorityQueue is an implementation class for unbounded priority queue but it has an internal capacity that governs the size of an array used to store priority queue elements. The capacity value is always at least as large as the queue size. As elements are inserted into the priority queue, its internal capacity grows automatically.

5. Element at the head of priority queue is least element.

Features of PriorityQueue

There are few important features of Priority Queue that are as follows:

6. Element at the tail of priority queue is greatest element.

7. Removal of elements always takes place from the front end (head) of queue.

8. PriorityQueue is not synchronized. That means it is not thread-safe. For working in a multithreading environment, Java provides a `PriorityBlockingQueue` class that implements the BlockingQueue interface.

9. It provides `O(log(n))` time for enqueuing and dequeuing operations.

10. The iterator returned by the iterator() method does not guarantee that it will traverse elements in their sorted order.

PriorityQueue Constructors

PriorityQueue defines six constructors that are as follows:

PriorityQueue Constructors

1. PriorityQueue(): This form of constructor is used to create an empty, default priority queue with an initial capacity of 11 elements. It orders its elements according to their natural ordering.

The general syntax to create a PriorityQueue instance with initial capacity of 11 is as follows:

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

PriorityQueue Constructors

2. PriorityQueue(int initialCapacity): This constructor is used to create a default priority queue with the specified initial capacity that orders its elements according to their natural ordering.

This constructor throws an exception named IllegalArgumentException when initialCapacity is less than 1.

PriorityQueue Constructors

3. PriorityQueue(Collection<? extends E> c): This constructor is used to create a priority queue with the specified collection c.

It throws ClassCastException when collection c's elements cannot be compared with one another based on the priority queue ordering. When c or any of its elements contain null object, this constructor throws NullPointerException.

PriorityQueue Constructors

4. PriorityQueue(int initialCapacity, Comparator<? super E> comparator): This constructor creates a priority queue with the specified initial capacity that orders its elements according to the specified comparator.

When comparator contains null reference, natural ordering is used to sort elements. This form of constructor also throws `IllegalArgumentException` when initialCapacity is less than 1.

PriorityQueue Constructors

5. PriorityQueue(PriorityQueue<? extends E> pq): This constructor creates a priority queue containing the elements in the specified priority queue pq.

The elements in this priority queue will be ordered according to the same ordering as pq. It also throws `ClassCastException` when pq's elements cannot be compared with one another based on the pq's ordering.

When pq or any of its elements contain null object, this constructor throws `NullPointerException`.

PriorityQueue Constructors

6. PriorityQueue(SortedSet<? extends E> ss): This constructor creates a priority queue containing the specified sorted set ss's elements. The elements in this priority queue will be ordered according to same ordering as ss.

When ss's elements cannot be compared with one another based on the ss's ordering, this constructor throws `ClassCastException`. When ss or any of its elements contain null object, this constructor throws `NullPointerException`.

PriorityQueue Methods

Method	Description
boolean add(E e)	This method is used to add the specified element into the priority queue.
void clear()	This method is used to remove all the elements from the priority queue.
comparator()	It returns the comparator used to order the elements of queue. If the queue is sorted according to the natural ordering of its elements, it returns null object.
boolean contains(Object o)	It returns true if the queue contains the specified element.
Iterator<E> iterator()	It returns an iterator over the elements in the priority queue.

PriorityQueue Methods

Method	Description
boolean offer(E e)	It is used to add the specified element into the priority queue.
E peek()	This method retrieves, but does not remove, element at the head of the queue. It returns null element if the queue is empty.
E poll()	This method retrieves and removes element at the head of the queue. It returns null if this queue is empty.
boolean remove(Object o)	It is used to remove the specified element from the queue.
int size()	It returns the number of elements in the queue.

PriorityQueue Methods

Method	Description
Object[] toArray()	It returns an array containing all the elements present in the queue
<T> T[ ] toArray(T[ ] a)	This method returns an array containing all the elements present in the queue; the runtime type of the returned array is that of the specified array.

PriorityQueue Methods

Methods inherited from AbstractQueue class:

```
addAll, remove, element
```

Methods inherited from AbstractCollection class:

```
isEmpty, removeAll, retainAll, containsAll, toString
```

Methods inherited from Object class:

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

Methods inherited from Collection interface:

```
equals, hashCode, isEmpty, removeAll, retainAll, containsAll
```

PriorityQueue Example

Example 1: PriorityQueue, Iterator

Let’s take some example programs to perform various operations based on the above methods provided by Java PriorityQueue.

Example 1: PriorityQueue, Iterator

```
import java.util.Iterator;
import java.util.PriorityQueue;
public class PriorityQueueTester1 {
   public static void main(String[] args) {
       // Create a Queue. This priority queue stores Strings objects.
       PriorityQueue<String> pq = new PriorityQueue<>();

       // Adds elements to the priority queue.
       pq.offer("USA");
       pq.offer("India");
       pq.offer("England");
       pq.offer("Germany");
       pq.offer("Australia");

       System.out.println("Priority queue: " +pq);

       // Iterating elements of priority queue.
       System.out.println("Iterating elements of priority queue");
       Iterator<String> iterator = pq.iterator();
       while (iterator.hasNext()) {
           System.out.print(iterator.next() + " ");
       }
   }
}
```

Example 1: PriorityQueue, Iterator

Output:

```
Priority queue: [Australia, England, India, USA, Germany]
Iterating elements of priority queue
Australia England India USA Germany
```

Explanation:

Notice in the output that when you print the queue, its elements are not ordered according to their priority.

This is because a queue is never used to iterate over its elements. PriorityQueue class does not guarantee any ordering of elements when we use an iterator. Therefore, when we print the priority queue, its elements are not ordered according to their priority.

However, when we will use peek() or remove() method, the correct element will be peeked at or removed, which is based on the element's priority.

Example 2: PriorityQueue, peek or remove the priority element

Let’s create a program where we will peek or remove the correct element based on priority.

Example 2: PriorityQueue, peek or remove the priority element

```
import java.util.PriorityQueue;
public class PriorityQueueTester2 {
   public static void main(String[] args) {
       PriorityQueue<String> pq = new PriorityQueue<>();

       pq.offer("USA");
       pq.offer("India");
       pq.offer("England");
       pq.offer("Germany");
       pq.offer("Australia");

       System.out.println("Elements in queue: " +pq);

       while (pq.peek() != null) {
           System.out.println("Head Element: " + pq.peek());
           System.out.println("Removed Element from Queue: " +pq.remove());
           System.out.println("Priority queue: " + pq);
       }
   }
}
```

Example 2: PriorityQueue, peek or remove the priority element

Output:

```
Elements in queue: [Australia, England, India, USA, Germany]
Head Element: Australia
Removed Element from Queue: Australia
Priority queue: [England, Germany, India, USA]
Head Element: England
Removed Element from Queue: England
Priority queue: [Germany, USA, India]
Head Element: Germany
Removed Element from Queue: Germany
Priority queue: [India, USA]
Head Element: India
Removed Element from Queue: India
Priority queue: [USA]
Head Element: USA
Removed Element from Queue: USA
Priority queue: []
```

Explanation:

As you can observe in the output, when we use the peek() and remove() methods, the correct element is peeked at and removed from the queue, which is based on the element's priority.

This is because PriorityQueue removes one element from it, processes that element, and then removes another element.

Example 3: PriorityQueue, Comparable, Comparator

Let’s create a program where we use Comparable and Comparator for priority.

Example 3: PriorityQueue, Comparable, Comparator

```
import java.util.Collections;
import java.util.PriorityQueue;
public class PriorityQueueTester3 {
   public static void main(String[] args) {
       PriorityQueue<Integer> pq = new PriorityQueue<>();  // Create a Queue. This priority queue stores Integers.
       pq.offer(50);                                       // Adds elements to the priority queue.
       pq.offer(100);
       pq.offer(60);
       pq.offer(20);
       pq.offer(10);
       System.out.println("Priority queue using Comparable:");
       while(pq.size() > 0) {
           System.out.print(pq.remove() + " ");
       }
       PriorityQueue<Integer> pq2 = new PriorityQueue<>(5, Collections.reverseOrder());
       pq2.offer(50);
       pq2.offer(100);
       pq2.offer(60);
       pq2.offer(20);
       pq2.offer(10);
       int size = pq2.size();
       System.out.println("\nSize of priority queue: " +size);
       System.out.println("\nPriority queue using Comparator:");
       while(pq2.size() > 0) {
           System.out.print(pq2.remove() + " ");
       }
   }
}
```

Example 3: PriorityQueue, Comparable, Comparator

Output:

```
Priority queue using Comparable:
10 20 50 60 100
Size of priority queue: 5

Priority queue using Comparator:
100 60 50 20 10
```

END