Priority Queue in java

A PriorityQueue is used when the objects are supposed to be processed based on the priority. **It is known that a queue follows 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.

- 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.
- The queue retrieval operations poll, remove, peek, and element access the element at the head of the queue.
- It inherits methods from AbstractQueue, AbstractCollection, Collection and Object class.

Basic Operations on PriorityQueue:

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

Let us understand with the help of an example

Data:120,60,160,30,80,140,180

```
import java.util.PriorityQueue;
class Demo
{
    public static void main(String[] args)
    {
        PriorityQueue pq = new PriorityQueue();
        pq.add(120);
        pq.add(60);
        pq.add(160);
        pq.add(30);
        pq.add(80);
        pq.add(140);
        pq.add(180);
        System.out.println(pq);
    }
}
```

Output:

```
[30, 60, 140, 120, 80, 160, 180]
Press any key to continue . . .
```

Note: The order of storing the values is explained in the video with more clarity.

Methods in PriorityQueue class:

1. **boolean add(E element):** This method inserts the specified element into this priority queue.
2. **public remove()**: This method removes a single instance of the specified element from this queue, if it is present
3. **public poll():** This method retrieves and removes the head of this queue, or returns null if this queue is empty.

4. **public peek()**: This method retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
5. **Iterator iterator()**: Returns an iterator over the elements in this queue.
6. **boolean contains(Object o):** This method returns true if this queue contains the specified element
7. **void clear():** This method is used to remove all of the contents of the priority queue.
8. **boolean offer(E e):** This method is used to insert a specific element into the priority queue.
9. **int size():** The method is used to return the number of elements present in the set.
10. **toArray()**: This method is used to return an array containing all of the elements in this queue.
11. **Comparator comparator()**: The method is used to return the comparator that can be used to order the elements of the queue.

TreeSet in java

Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements the NavigableSet interface**. The objects of the TreeSet class are stored in ascending order.**

- Java TreeSet class contains unique elements only like HashSet.
- Java TreeSet class access and retrieval times are quiet fast.
- Java TreeSet class doesn't allow null element.
- Java TreeSet class is non-synchronized.
- Java TreeSet class maintains ascending order.

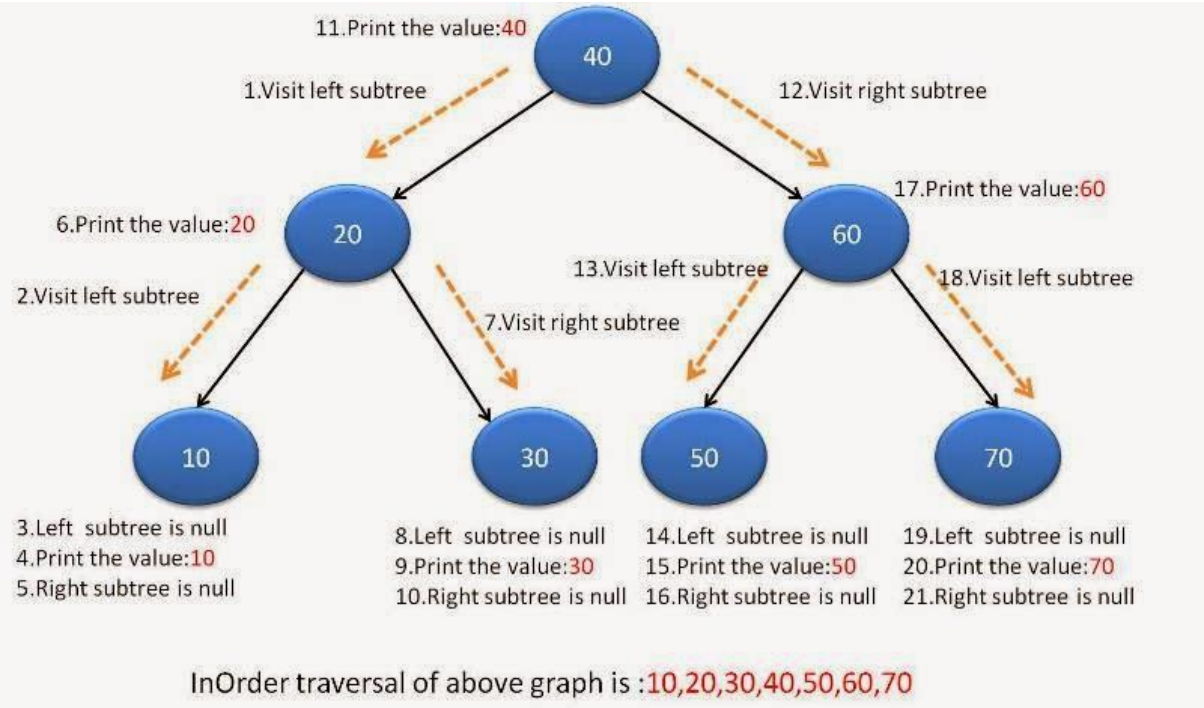
Let us see an example for treeset,

```
import java.util.TreeSet;
class Demo
{
    public static void main(String[] args)
    {
        TreeSet ts = new TreeSet();
        ts.add(120);
        ts.add(60);
        ts.add(160);
        ts.add(30);
        ts.add(80);
        ts.add(140);
        ts.add(180);
        System.out.println(ts);
    }
}
```

Output:

```
[30, 60, 80, 120, 140, 160, 180]
Press any key to continue . . .
```

Let us see how exactly we got this order



Methods of TreeSet in java

Methods	Description
boolean add(E e)	It is used to add the specified element to this set if it is not already present.
boolean addAll(Collection<? extends E> c)	It is used to add all of the elements in the specified collection to this set.
E ceiling(E e)	It returns the equal or closest greatest element of the specified element from the set, or null there is no such element.
Comparator<? super E> comparator()	It returns comparator that arranged elements in order.
Iterator descendingIterator()	It is used iterate the elements in descending order.
NavigableSet descendingSet()	It returns the elements in reverse order.
E floor(E e)	It returns the equal or closest least element of the specified element from the set, or null there is no such element.
SortedSet headSet(E toElement)	It returns the group of elements that are less than the specified element.
NavigableSet headSet(E toElement, boolean inclusive)	It returns the group of elements that are less than or equal to(if, inclusive is true) the specified element.
E higher(E e)	It returns the closest greatest element of the specified element from the set, or null there is no such element.
Iterator iterator()	It is used to iterate the elements in ascending

	order.
E lower(E e)	It returns the closest least element of the specified element from the set, or null there is no such element.
E pollFirst()	It is used to retrieve and remove the lowest(first) element.
E pollLast()	It is used to retrieve and remove the highest(last) element.
Spliterator spliterator()	It is used to create a late-binding and fail-fast spliterator over the elements.
NavigableSet subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive)	It returns a set of elements that lie between the given range.
SortedSet subSet(E fromElement, E toElement))	It returns a set of elements that lie between the given range which includes fromElement and excludes toElement.
SortedSet tailSet(E fromElement)	It returns a set of elements that are greater than or equal to the specified element.
NavigableSet tailSet(E fromElement, boolean inclusive)	It returns a set of elements that are greater than or equal to (if, inclusive is true) the specified element.
boolean contains(Object o)	It returns true if this set contains the specified element.
boolean isEmpty()	It returns true if this set contains no elements.
boolean remove(Object o)	It is used to remove the specified element from this set if it is present.
void clear()	It is used to remove all of the elements from this set.
Object clone()	It returns a shallow copy of this TreeSet instance.
E first()	It returns the first (lowest) element currently in this sorted set.
E last()	It returns the last (highest) element currently in this sorted set.
int size()	It returns the number of elements in this set.