Set:

- It is the child interface of collection interface.
- If we want to represent a group of individual object as a single entity where duplicates are not allowed and insertion order is not preserved then we will use set inter face.
- Set interface does not contain any new method and we have to use only collection interface methods.

HashSet:

- Child class of set.
- Underlying datastructure is hashtable.
- Duplicate objects are not allowed.
- Insertion order is not preserved and it is based on hashcode of objects.
- Null insertion is possible.
- Heterogineous objects are allowed.
- Implements serialaizable and clonable but not random access interface.
- It is best choice if our frequent operation is search operation.
- Default initial capacity is 16 and fill ratio is 0.75.

Constructors for Hashset:

- HashSet h=new HashSet()//Will create an empty hashset of intial capacity 16 and fill ratio is .75%.
- HashSet h=new HashSet(int initialcapacity)//creats a new empty hash set with specified initial capacity but the fill ratio will remain same.
- HashSet h=new HashSet(int initialcapacity,float fillRatio)//Will create a empty hash set with specified capacity and specified fill ratio.
- HashSet h=new HashSet(Collection c)//Will create a equivalent hashset for given collection.

LinkedHashSet:

- It is the child class of hashSet.
- If we want to represent a group of individual objects as a single entity where duplicates are not allowed but insertion order is preserved then we should go for linkedhashset.

- Underlying datastructure is a combination of linkedlist and hashtable.
- In general we can use Linked hashset to develop cashibased application where duplicates are not allowed and insertion order preserved.

SortedSet:

- SortedSet is the child interface of set interface.
- If we want a group of individual object to be represented as a single entity where duplicates are not allowed but the objects are inserted by following some sorting order then we will go for sortedSet.
- If we don't define the sorting order it will automatically insert the objects as per the default natural sorting order.
- Default natural sorting order means in case of numbers it will sort them as ascending order and in case of String it will follow the alphabetical order.

Methods related to sortedset:

- SortedSet interface defines some specific methods.

TreeSet:

- It is the child class of Navigable Set interface.
- Underlying data structure is Balanced Tree.
- Duplicate objects are not allowed.
- Insertion order not preserved.
- Heterogeneous objects are not allowed.
- Null insertion is possible only once.
- TreeSet implements serialaizable and clonable but not Random access.
- All objects will be inserted based on some sorting order,It may be default sorting order or customized sorting order.
- Default sorting in case of integer is ascending order and in case of String alphabetical order.

Constructors:

- TreeSet t=new TreeSet();//It will create a empty treeset object which will follow natural sorting order.
- TreeSet t=new TreeSet(Comparator c);//It will create a empty treeSet object where the sorting order will be difined by the comparator object.
- TreeSet t=new TreeSet(Collection c);
- TreeSet t=new TreeSet(SortedSet s);

e.g:

```
        TreeSet <Object> t=new TreeSet<>();
        t.add(10);
        t.add(12);
        t.add(9);
        //t.add("Tusar");//It will throw classcast
exception because heterogeneous objects are not allowed
        System.out.println(t);//[9, 10, 12] default
sorting ascending order.
```

Null Acceptence:

- For non empty treeset if we try to add null then we will get null pointer exception.
- For empty treeset we can add null in treset but after that if we try to add any other elements then we will get null pointer exception.

From java 7 version onwards the null insertion is not allowed in treeset.

E.G:

```
TreeSet <Object> t=new TreeSet<>();
t.add(new StringBuffer("A"));
        t.add(new StringBuffer("B"));
        t.add(new StringBuffer("C"));
        System.out.println(t);//Class cast exception
```

- The above example will give classcast exception because here we are trying to add String buffer object and String buffer class doesn't implements comparable interface.
- If we are depending on default natural sorting order compulsory the object should be homogeneous and comparable otherwise we will get class cast exception.
- We can say a object is comparable if and only if the corresponding class implements the comparable interface.
- String class and all wrapper classes already implements comparable but the string buffer class does not implements comparable interface.

Comparable(Interface):

- It is present in java.lang package and it contains only one method compareTo().
- It is meant for default natural sorting order.

Syntax:

public int compareTo(Object obj);

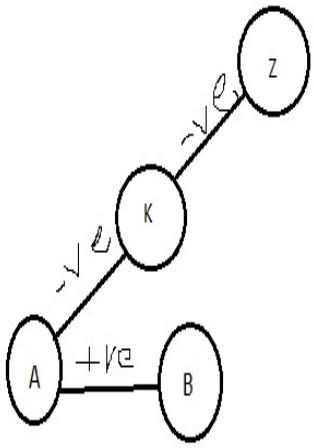
obj1.compareTo(obj2);
Here obj1 means the object which we are trying to isert and obj2 means which is already inserted.

- Return –ve if obj1 has to come before obj2.
- Return +ve if obj1 has to come after obj2.
- Return 0 if obj1 and obj2 both are same and in this case insertion will not happen.

- If we are depending on default natural sorting orderer in treeset then internally jvm will call compareTo method.

```
TreeSet <Object> t=new TreeSet<>();

t.add("z");          k.compareTo(z)//(-ve)
t.add("k");          A.compareTo(z)//(-ve)
t.add("A");          A.compareTo(k)//-ve
                     A.compareTo(z)//-ve
t.add("A");          A.compareTo(k)//-ve
                     A.compareTo(A)//0
t.add("B");          B.compareTo(z)//-ve
                     B.compareTo(k)//-ve
                     B.compareto(A)//+ve
System.out.println(t);
```



[A,B,k,z]

Internally jvm will call compareTo method

Comparator:

- Comparator present in java.util package and it defines two methods compare() and equals().

- Public int compare(Object obj1,Object obj2)
  - Return –ve if obj1 has to come before ob2.
  - Return +ve if obj1 has to come after obj2.
  - Return 0 if obj1 and obj2 are same.
- Public boolean equals(Object obj)
- Whenever we are implementing comparator interface compulsory we should provide the implementation for the compare method and we are not required to provide implementation for equals methods because it is already available to our class, Because every class in java extends object class and equals method is present in the object class so by default it is available to our class.

Comparable vs Comparator:

- For predefined comparable classes default natural sorting order already available.
- If we are not satisfied with that default natural sorting order we can define our own sorting using comparator.
- For predefined non comparable classes (like string buffer) default natural sorting order not already available, we can define our own sorting with the help of comparator.
- For our own classes like employee the person who is writing the class is responsible to define default natural sorting order by implementing comparable interface.
- The person who is using our class if he is not satisfied with our sorting then he can use comparator for customized sorting order.

Comparision Table of Set implemented classes:

Properties	HashSet	LinkedHashSet	TreeSet
1. Duplicate objects.	**NA**	**NA**	**NA**
2. Insertion order.	**Not Preserved**	**Preserved**	**Not Preserved**
3. Sorting Order	**Not Followed**	**Not Followed**	**Follow sorting order (Default sorting and customized sorting)**
4. Underlying Data Structure	**HashTable**	**LinkedList and HashTable**	**Balanced Tree**
5. Heterogeneous Objects	**Allowed**	**Allowed**	**Not Allowed**

Queue(1.5 version enhancements):

- It is the child interface of collection.
- If we want to represent a group of individual objects prior to processing then we should go for queue.
- Before sending sms we have to store all mobile numbers in a datastructure, in which order we added mobile number in same order only the msg should be delivered for this first in first out requirement queue is the best choice.
- Useually que follows first in first out order but based on our requirement we can implement our own priority order also(Priority queue).
- From 1.5 verson onwards linkedlist class also implements queue interface.Linkedlist based implementation of queue always follows first in first out order.

Queue interface specific methods:

- Boolean offer(Object O)//To add element in the queue
- Object peek()//To return the head element from the queue, If the queue is empty the it will return null.
- Object element()//To return the head element from the queue if the queue is empty then it will give runtime exception which is NosuchElement Exception.
- Object poll()//To remove and return the head element from the queue if the queue is empty then it will give null.
- Object remove()//To remove and return the head element from the queue if the queue is empty then it will give runtime exception which is NosuchElement Exception.

Priority Queue:

- If we want to represent a group of individual objects prior to processing then we should go for priority queue.
- The priority can be either default natural sorting order or customized sorting order defined by comparator.
- Insertion order is not preserved and it is based on some priority.
- Duplicate objects are not allowed.
- If we are depending on default natural sorting order, cumpolsory object should be homogeneous and comparable otherwise we will get runtime exception saying classcast exception.

- If we are defining our own sorting by comparator then objects need not be homogeneous and comparable.
- Null is not allowed even as the first element also.

Constructors of priority Queue:

- **PriorityQueue q=new PriorityQueue()//Creates an empty priorityque with default initial capacity of 11 and all objects will be inserted according to default natural sorting order.**
- PriorutyQueue q=new PriorityQueue(int initial capacity).
- PriritQueue q=new priorityQueue(int initialcapacity,Comparator c).
- PriritQueue q=new priorityQueue(SortedSet s).
- PriritQueue q=new priorityQueue(Collection C).

E.g:

```
public static void main(String[] args) {
        PriorityQueue<String>q=new PriorityQueue<>();
        q.offer("Tusar");
        q.offer("Rocky");
        q.offer("Sangram");
        System.out.println(q);//[Rocky, Tusar, Sangram]
        System.out.println(q.peek());//Rocky
        System.out.println(q.element());//Rocky
        System.out.println(q.poll());//Rocky
        System.out.println(q);//[Tusar,Sangram]
        System.out.println(q.remove());//Tusar
        System.out.println(q);//[Sangram]

    }
```