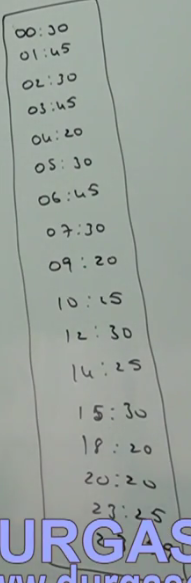
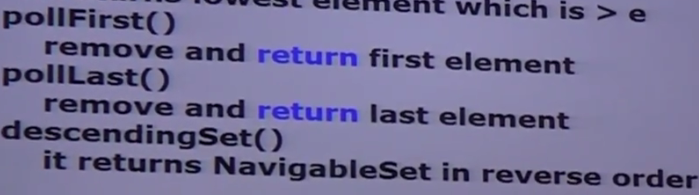
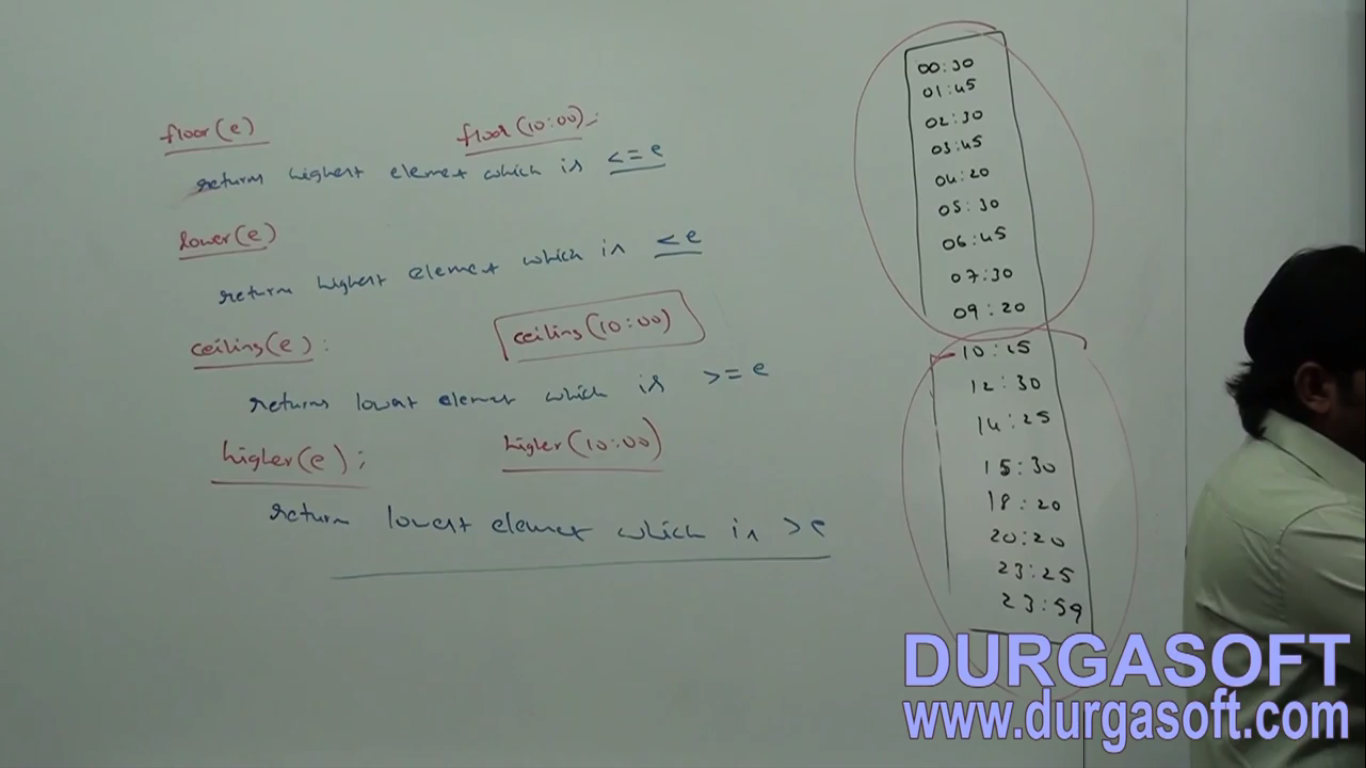
1. **In 1.6 version , new concepts came**
   1. NavigableSet
   2. NavigableMap
2. d

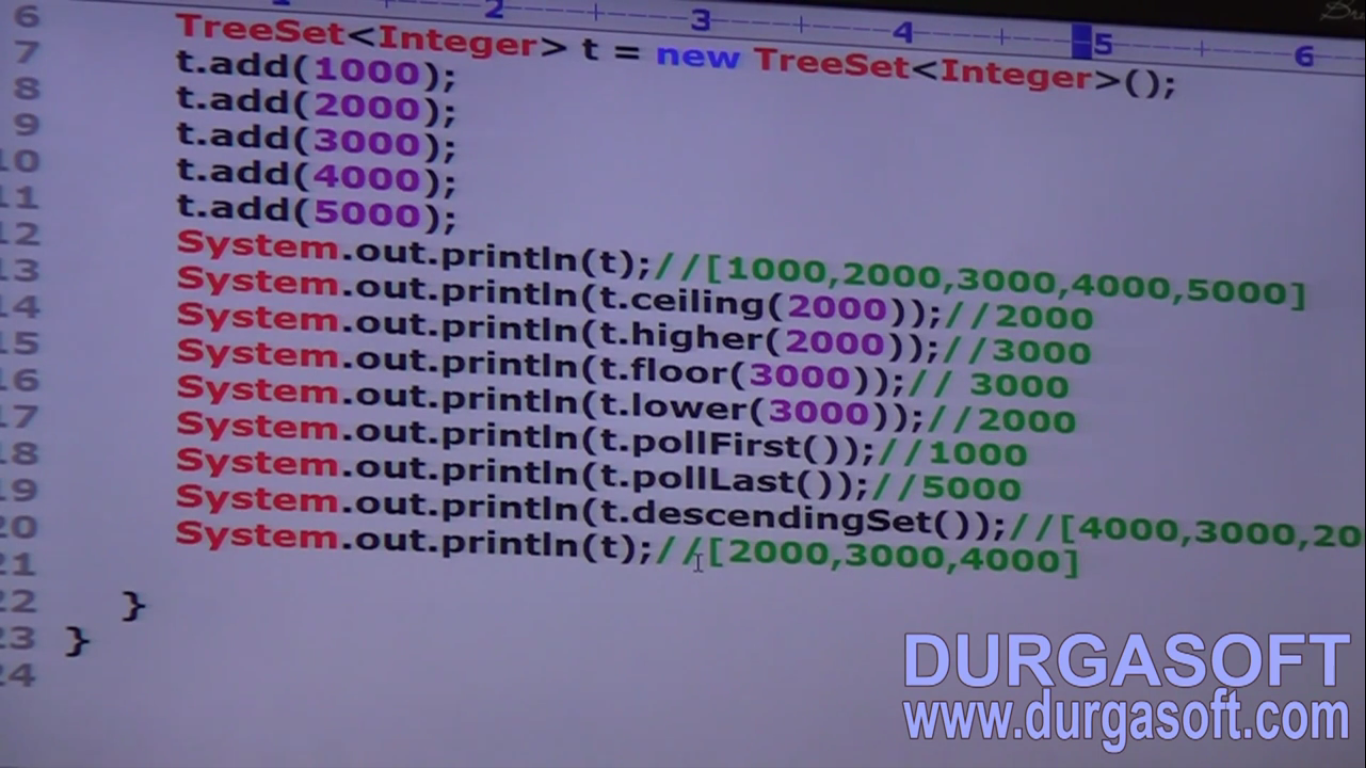
NavigableSet

1. It’s child interface of SortedSet(I) and it defines several methods navigation purposes.   
   
2. **Scenario**:   
   Followings are the flight Scheduling Table  
   
3. **Methods**:

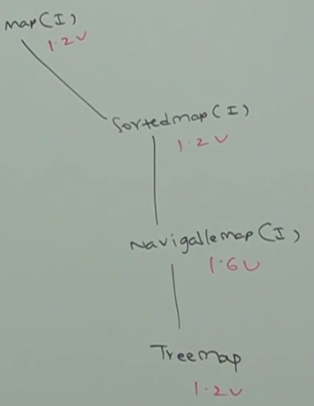
Now suppose we want to know (the last flight before 10:00), (the first flight after 10:00) 🡪 there is navigation like before, after  
In **SortedSet** there is no method fulfilling such requirement. The methods in **SortedSet** are  
 first(), last() 🡪 would give first and last from whole set.  
 headset(10:00)🡪 subset before 10:00, tailSet(10:00): subset after 10:00,   
 subset(10:00, 20:00) Subset b/w 10:00 and 20:00  
None of the above methods fulfills our requirement.  
  
So the above question can be answered with methods in **NavigableSet**lower(10:00): It says that the last flight before 10:00  
NavigationSet provides methods for navigation purpose.

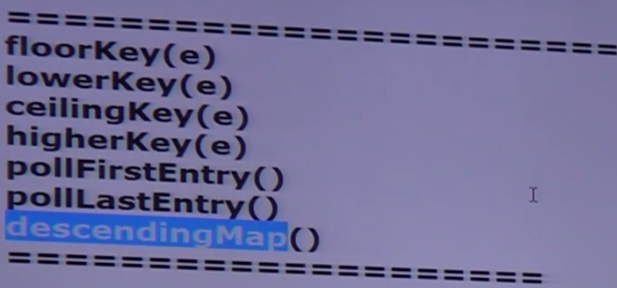
* 1. **floor(e):** Returns highest element which is <= e
     1. floor(10:00): first flight which is at 10:00 or before 10:00
  2. **lower(e)**: Returns highest element which is <e
     1. lower(10:00): first flight which is before 10:00
  3. **ceiling(e)**: Returns lowest element which is >=e
     1. ceiling(10:00): first flight which is at 10:00 or after 10:00
  4. **higher(e)**: Returns lowest element which is > e
     1. higher(10:00): first flight which is after 10:00
  5. **d**

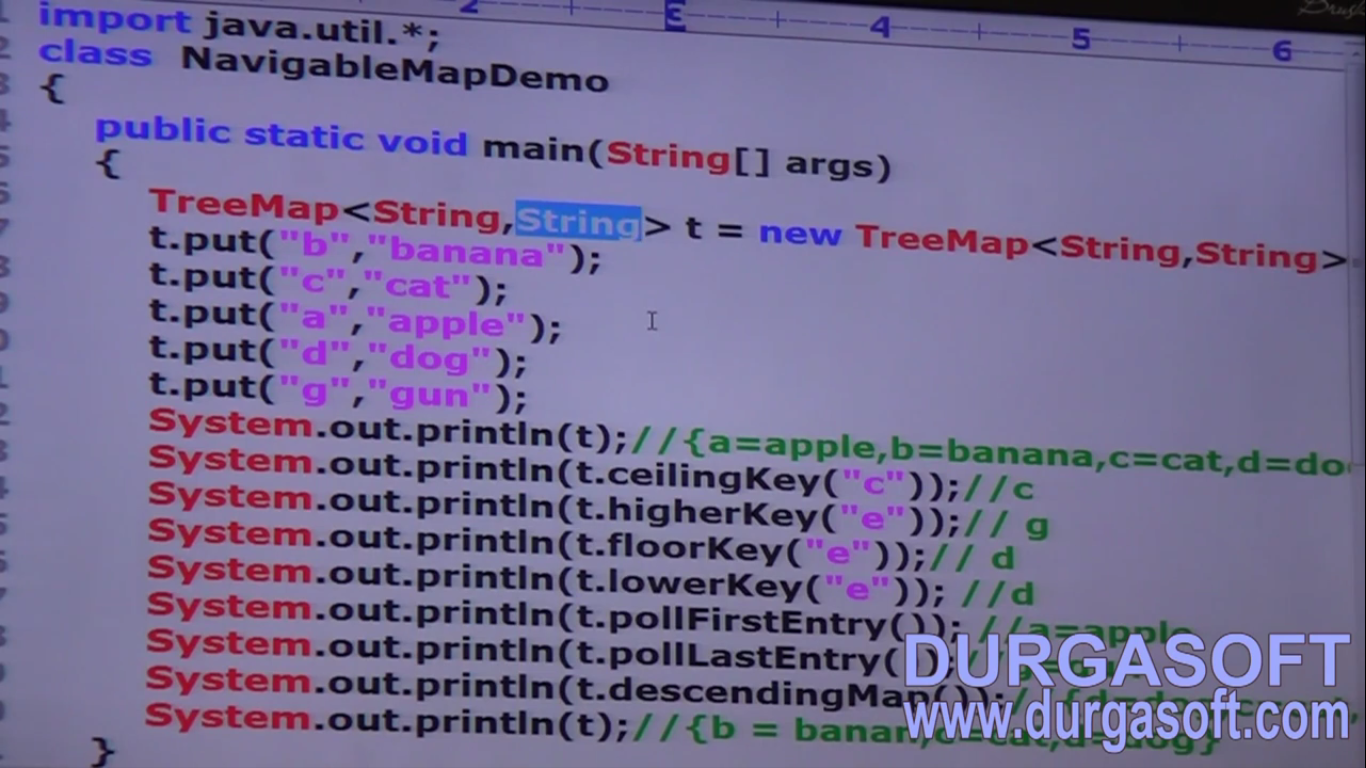


1. **Example:**   
   
2. **d**

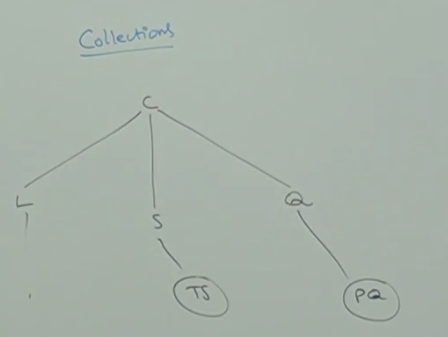
NavigableMap

1. NavigableMap is the child interface of **SortedMap**.
2. It defines several methods for navigation purposes.
3. To know more about why navigable map came into picture, go to the topic 🡪 NavigableSet. Theory is same.
4. **Place of NavigableMap in hierarchy.**   
   
5. **Methods**:  
    NOTE: Here in NavigableMap, we have key and value. Entry is sorted according to Key. So we will navigate the map via key.



1. **Example**:  
   
2. d

utility class

1. **2 Utility classes**
   1. **Collections**
   2. **Arrays**
2. **Scenario**:
   1. Consider the following diagram  
      
   2. We have List, Set, Queue.   
      In set 🡪 TreeSet, In Queue 🡪 PriorityQueue provide mechanism to sort their elements but there is no mechanism to sort elements in List. List can maintain insertion order.
   3. **Solution:** Collections. It defines several utility methods for collection objects like
      1. **Sorting**
      2. **Searching**
      3. **Reversing**
      4. **etc.**
   4. **Example**: Sorting elements of List
3. **Collections class define the following two sorting-methods**
   1.   
      In this case, List should compulsory contain homogeneous comparable objects otherwise we will get **ClassCastException**. Why? Because this method sorts the objects in the list by calling compareTo(Object) method on list element assuming the element to be Comparable(I) type.  
      Second, as the comparison will be made using default natural sorting order, so null must not be in the list otherwise we will get NullPointerException.
   2. To sort based on customized sorting order.

# Demo Program for sorting elements of List according to Default Natural Sorting Order.

# Demo to sort List according to comparator

Jatin 🡪 ClassCastException in TreeSet and TreeMap

1. When we talk about ClassCastException related to sorting in TreeSet and TreeMap, there are two cases when this exception can arise.

**NOTE**: We are assuming that we are talking about **Default Natural Sorting Order**.

* 1. **Case01**:
     1. When TreeSet and TreeMap assume that we are depending on Default Natural Sorting Order and elements in the collection can be sorted no problem as the object class implements **Comaparable**(I) but we added some heterogeneous type of object.
  2. **Case02**:
     1. When TreeSet and TreeMap assume that we are depending on Default Natural Sorting Order but the class of elements in collection doesn’t implement **Comparable**(I).