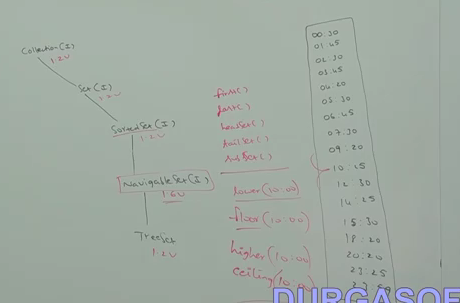
**1.6 Enhancement in Collection Framework:**

1. Navigable Set Interface

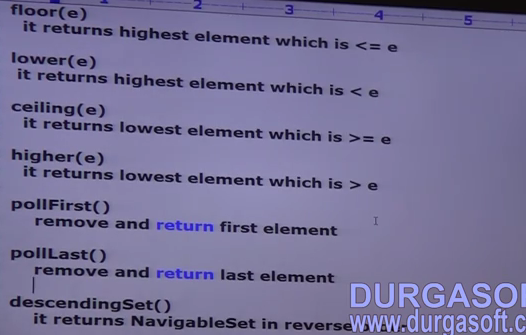
2. Navigable Map Interface

**NavigableSet (I):**

1. It is the child interface of SortedSet interface
2. Provides navigable functions

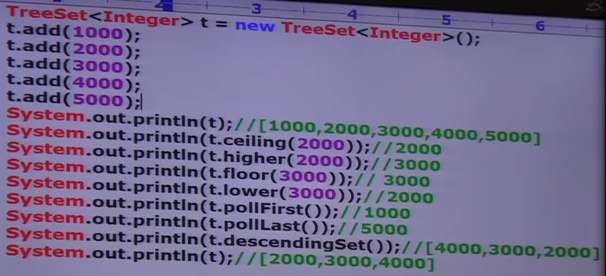
****

**Methods:**



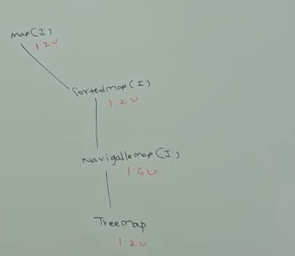
* Above methods, we can apply in TreeSet only cause Treeset is the implementation class for navigable set interface.

**Example:**

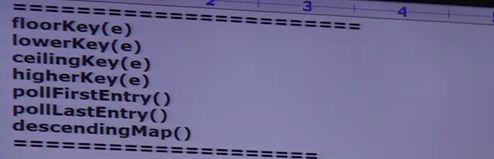
****

**NavigableMap (I):**

1. It is the child interface of sorted map interface
2. Provides navigable functions

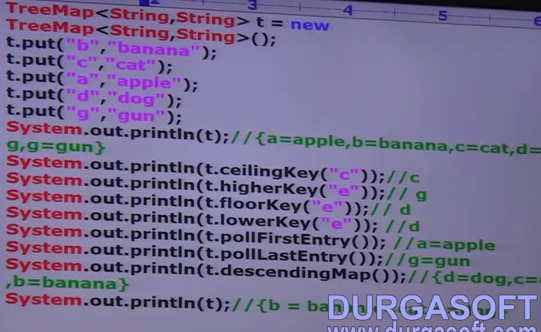


**Methods:**

****

* Above methods, we can apply in Treemap only cause Treemap is the implementation class for navigable map interface.

Example:



**Utility Classes:**

1. Collections
2. Arrays

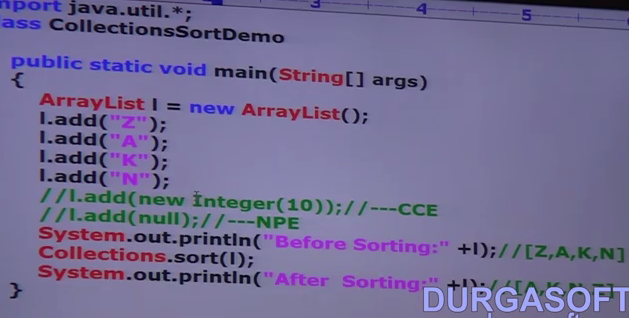
**Collections:**

1. Provides utility methods like sorting ,searching for Collection Objects

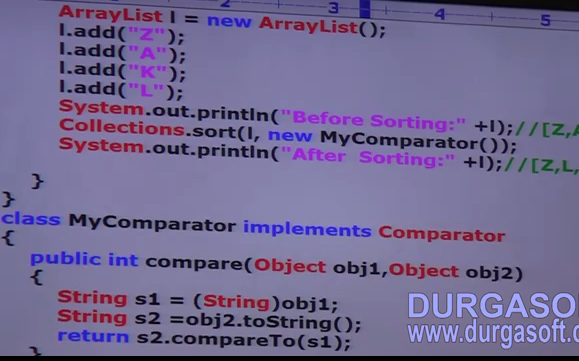
**Sorting Elements of list:**

1. Public static void sort(List l) ->
2. Sort based on default natural sorting order
3. Objects should be homogeneous and comparable otherwise throw RE : ClassCast Exception
4. List should not contain null otherwise throw RE: NPE
5. Public static void sort(list l, Comparator)
   1. Sort based on customized sorting order given in comparator
   2. Objects are not need be homogeneous and comparable

Example:



Example 2:



**Searching Elements of list:**

1. Public static int binarySearch(List l, Object target)

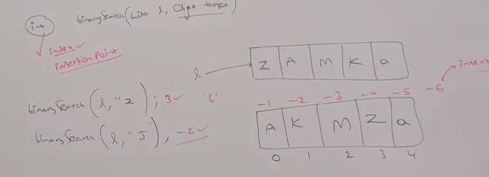
* We have to use this method, if the list is sorted according to default natural sorting order

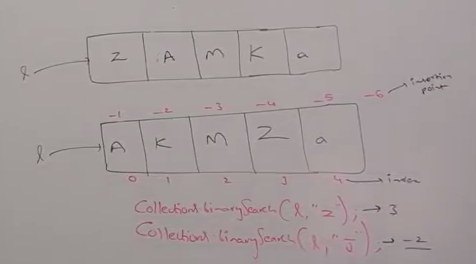
1. Public static int binarySearch(List l, Object target, Comparator)

* We have to use this method, if the list is sorted according to customized sorting order

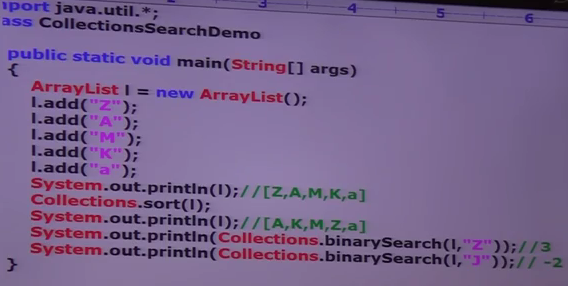
**Conclusions:**

1. Above search methods works on binary search algm.
2. Successful search return index of target element, and unsuccessful search return insertion point.
3. Insertion point is a location, where we can place the target element in the sorted list.
4. Before calling binary search method, compulsory the list should be in sorting order otherwise we will get unpredictable results. We won’t get any exceptions.
5. If the list is sorted according to comparator, then at the time of search operation also, we have to pass the same comparator object otherwise we will get unpredictable results.

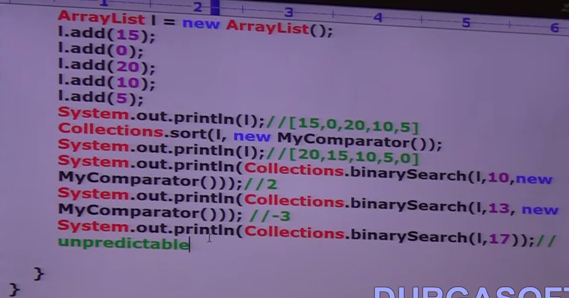
****

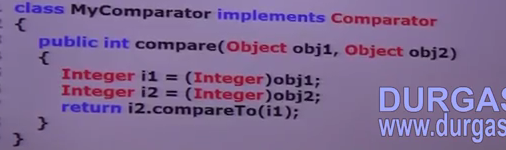
****

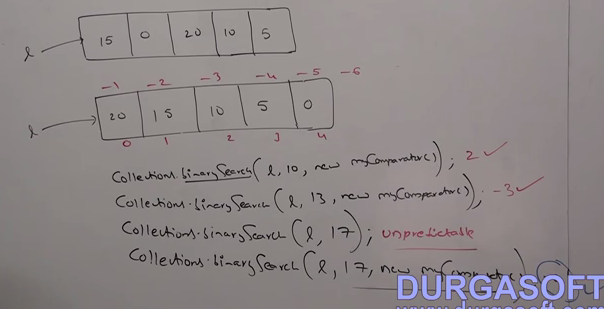
**Example:**

****

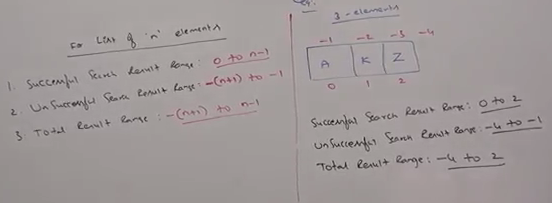
**Example 2:**

****

****

****

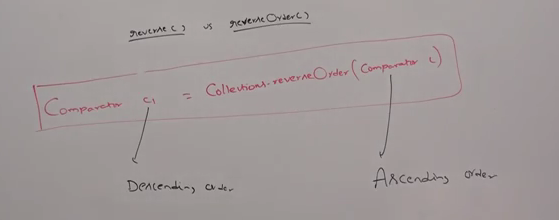
**Binary Search – For list of N elements:**

****

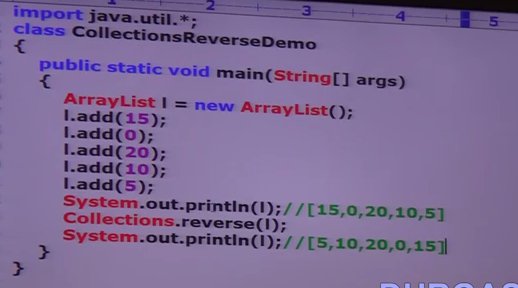
**Reversing Elements of List:**

**Methods:**

1. Public static void reverse(List l) -> reverse the order of elements in the list
2. Public Comparator reverseOrder(Comparator c) -> to get the reversed comparator



Example:



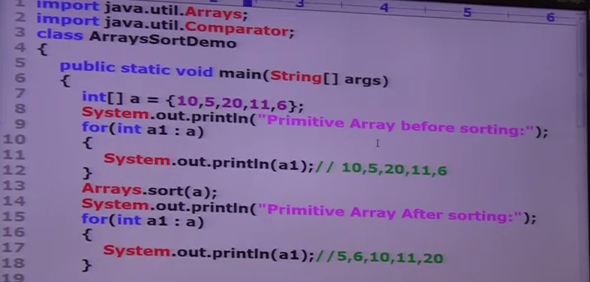
**Arrays:**

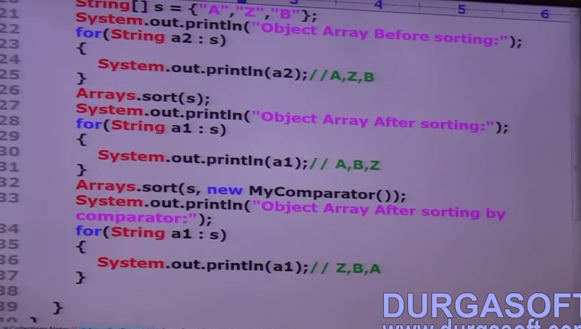
* Provide utility methods for Arrays

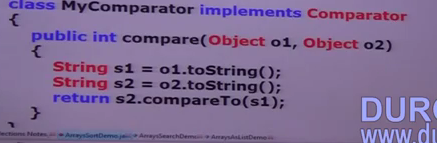
**Sorting Elements of Array:**

1. Public static void sort(primitive[] p) (
   1. primitive means – byte[], char[], int[]
   2. Sort based on Default natural sorting order
   3. Only for primitive types
2. Public static void sort(Object[] o)
   1. Sort based on Default natural sorting order
   2. For object types
3. Public static void sort(Object[] o, Comparator c)
   1. Sort based on Customized sorting order
   2. For object types

**Example:**

****

****

****

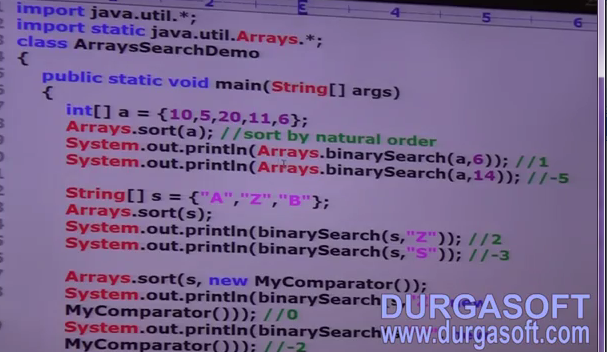
**Note: We can sort primitive array only based on default natural sorting order whereas we can sort object arrays either based on default natural or customized sorting order.**

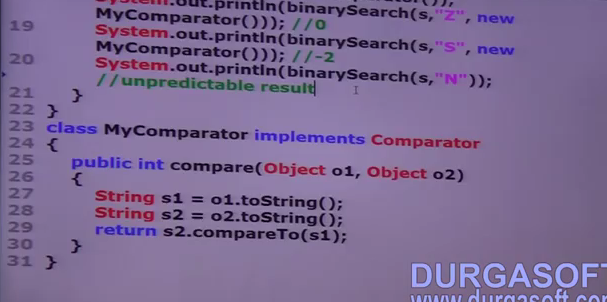
**Searching Elements of Array:**

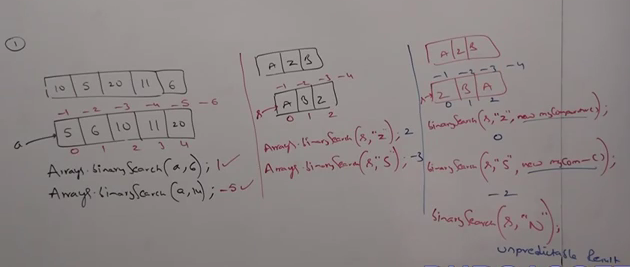
1. Public static int binarySearch(primitive[] p, primitive target)
2. Public static int binarySearch(Object[] o, Object target)
3. Public static int binarySearch(Object[] o, Object target, Comparator c)

* **All rules for binarySearch() for arrays are exactly same as binarySearch() of Collections class.**

**Example:**

****

****

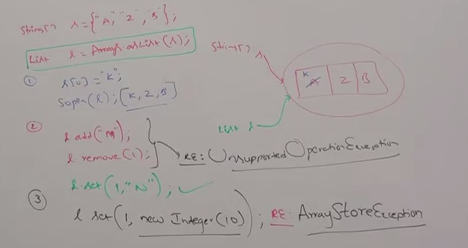
****

**Conversion of List to Array:**

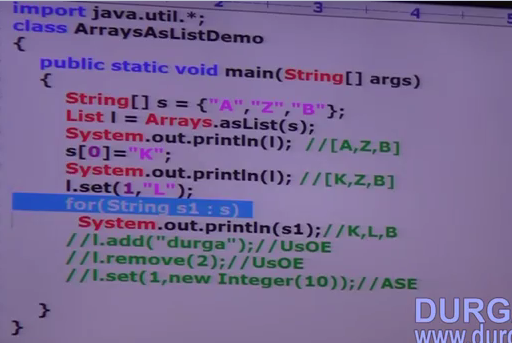
1. Object [] toList()

**Conversion of Array to List:**

1. Public static List asList(Object o[])
   1. Why the method name is asList instead of toList? -> Strictly Speaking, this method won’t create an independent List object. For the existing Array, we are getting List view only not List instance.
   2. By using array reference, if we perform any change automatically that change will be reflected to the List. Similiarly, by using List reference, if we perform any change, the same will reflected in Array.
   3. By using List reference, we cannot perform any operation which varies the Array size otherwise we will get RE: UnSupportedOperation Exception. Like **remove(), add() in List** class.
   4. But set(int index, Obejct element) will work because it just replace the value in the given index and the size wont be altered and also the inserting element should be the type belonging to array otherwise we get RE:ArrayStoreException. That is By using list reference, we are not allowed to replace with heterogeneous objects.

****

**Example:**

****