**Collection Framework – Part\_17**

* **Searching elements of list:**

Collections class defines the following binary search methods.

public static int binarySearch(List l, Object target);

If the list is sorted according to default natural sorting order, then we have to use this method.

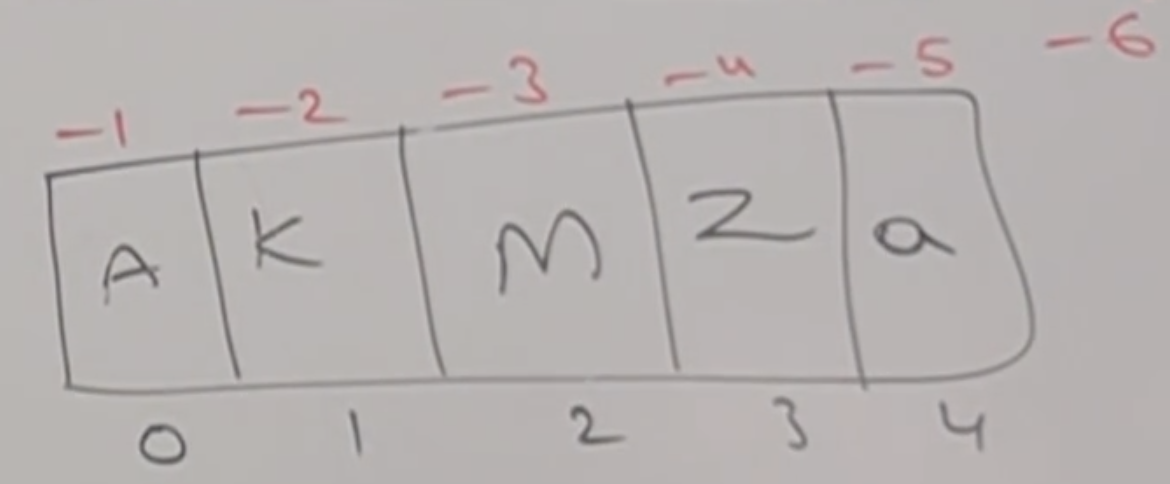
public static int binarySearch(List l, Object target, Comparator c);

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

Conclusions:

1. The above search methods internally will use Binary Search algorithm.
2. Successful search returns index.
3. Unsuccessful search returns insertion point.
4. Insertion point is the location where we can place target element in the sorted list.
5. Before calling binarySearch() method compulsory list should be sorted otherwise will get unpredictable results.
6. If the List is sorted according to Comparator then at the time of search operation also, we have to pass same Comparator object, otherwise we will get unpredictable results.

Insertion Point:



* **Example\_01:**

import java.util.\*;

class CollectionsSearchDemo{

public static void main(String[] args){

ArrayList l = new ArrayList();

l.add(“Z”);

l.add(“A”);

l.add(“M”);

l.add(“K”);

l.add(“a”);

System.out.println(l); //[Z,A,M,K,a]

Collections.sort(l);

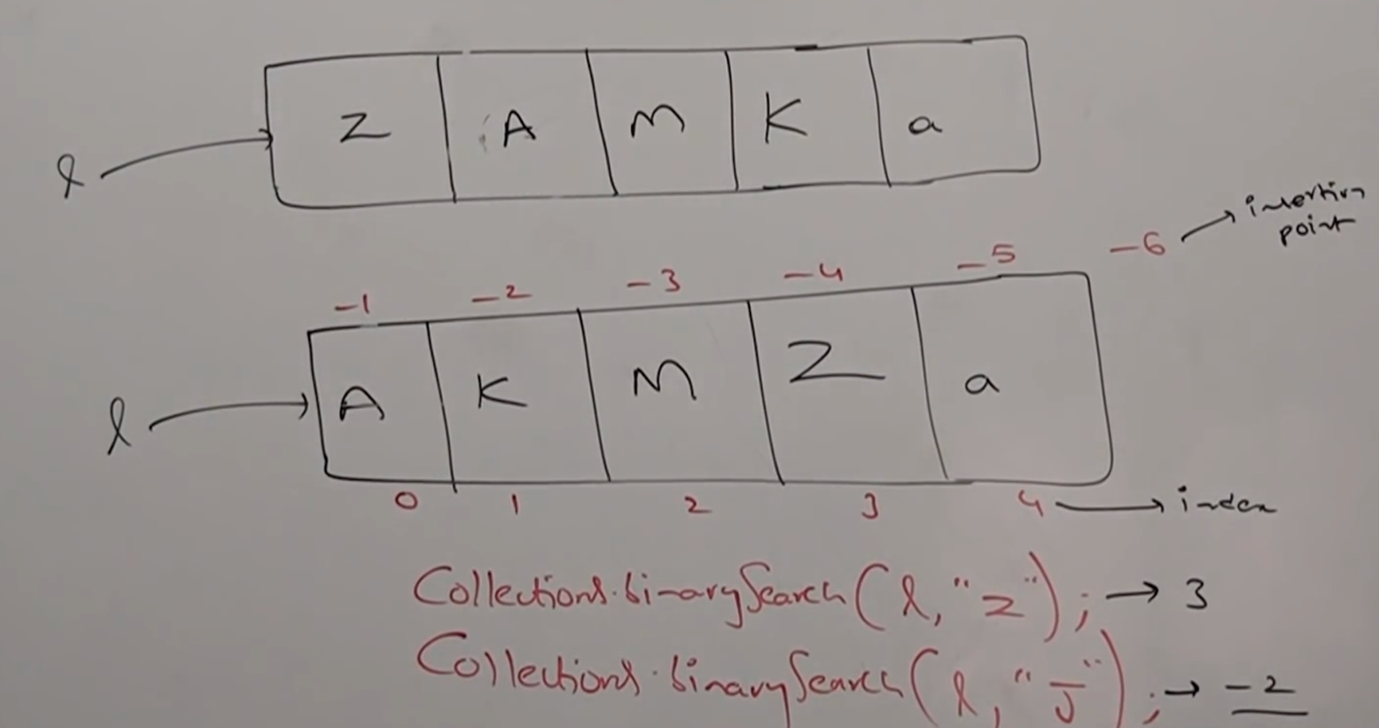
System.out.println(l); //{A,K,M,A,a]

System.out.println(Collections.binarySearch(l, “Z”)); //3

System.out.println(Collections.binarySearch(l,”J”)); //-2

}

}

****

* **Example\_02:**

import java.util.\*;

class CollectionsSearchDemo{

public static void main(String[] args){

ArrayList l = new ArrayList();

l.add(15);

l.add(0);

l.add(20);

l.add(10);

l.add(5);

System.out.println(l); [15,0,20,10,5]

Collections.sort(l, new MyComparator());//[20,15,10,5,0]

System.out.println(Collections.binarySearch(l,10,new MyComparator())); // 2

System.out.println(Collections.binarySearch(l, 13, new MyComparator())); // -3

System.out.println(Collections.binarySearch(l, 17)); //unpredictable

}

}

class MyComparator implement Comparator{

public int compare(Object obj1, Object obj2){

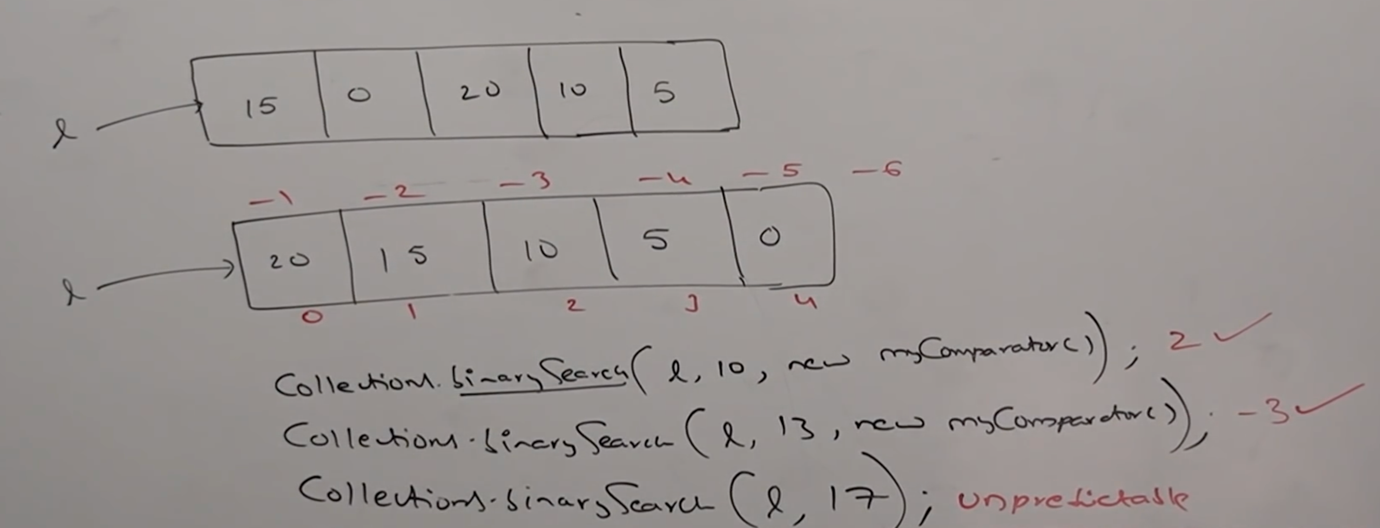
Integer I1 = (Integer) obj1;

Integer I2 = (Integer) obj2;

return I2.compareTo(I2);

}

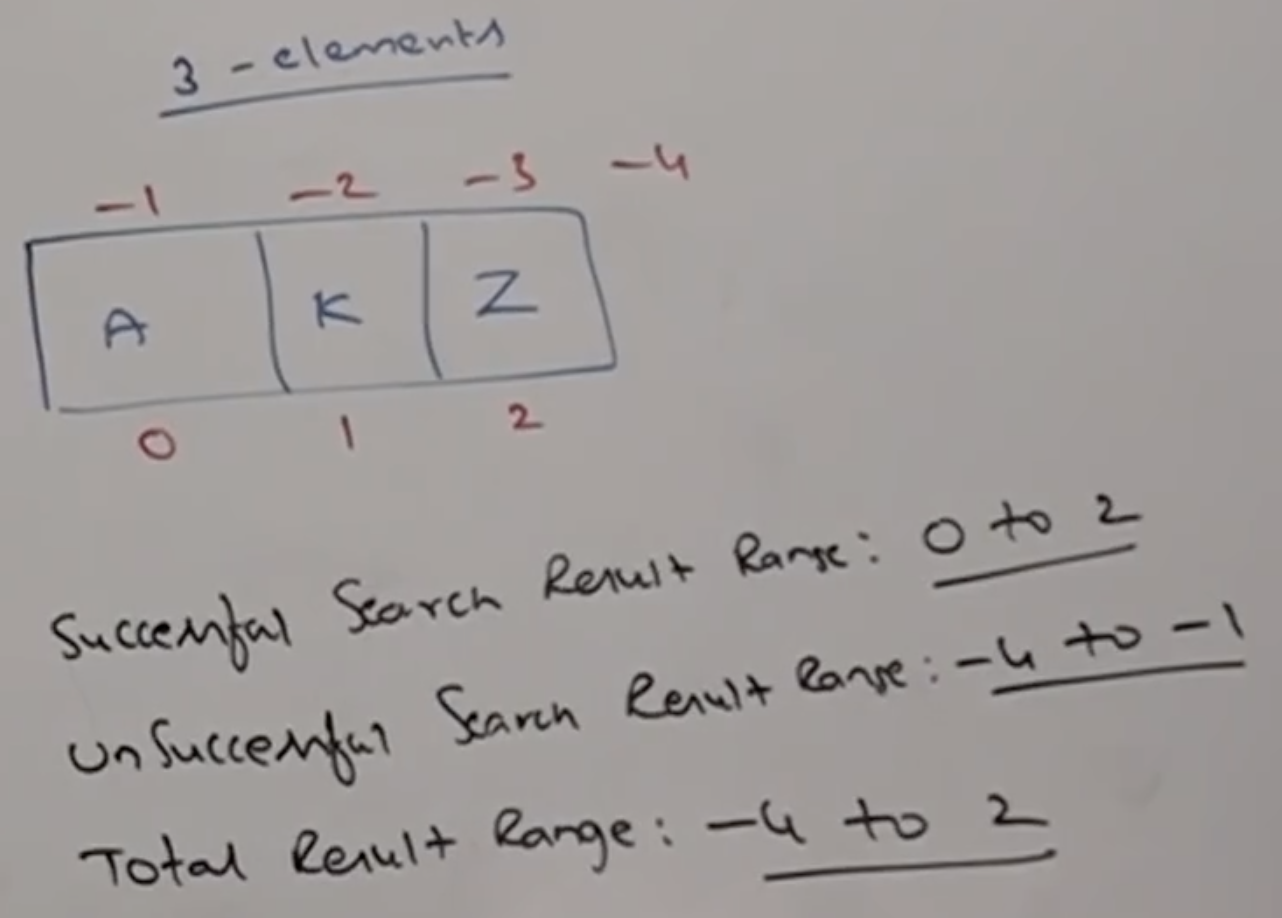
}

****

* **Note:**

For the list of “n” elements, the case of binary search method

1. Successful search result range 0 to n-1
2. Unsuccessful search result range –(n+1) to n-1
3. Total result range –(n+1) to n-1



* **Reversing elements of List:**

Collections class defines the following reverse() method to reverse elements of List.

public static void reverse(List l);

* **Example:**

import java.util.\*;

class CollectionsReverseDemo{

public static void main(String[] args){

ArrayList l = new ArrayList();

l.add(15);

l.add(0);

l.add(20);

l.add(10);

l.add(5);

System.out.println(l); // [15,0,20,10,5]

Collections.reverse(l);

System.out.println(l); // [5,10,20,0,15]

}

}

* **reverse() vs reverseOrder():**

We can use reverse method to reverse order of element of List.

Whereas we can use reverseOrder() method to get reversed comparator.

