

## Utility Classes

07 July 2025 18:48

**Collections:** Collections class defines several utility methods for collections objects like Sorting, searching, reversing, etc.

Sorting elements of List:

i. public static void sort(List l);

① D.N.S.O

② Objects should be homogenous and comparable

otherwise Runtime exception: ClassCastException.

③ If List contain 'null' as object then

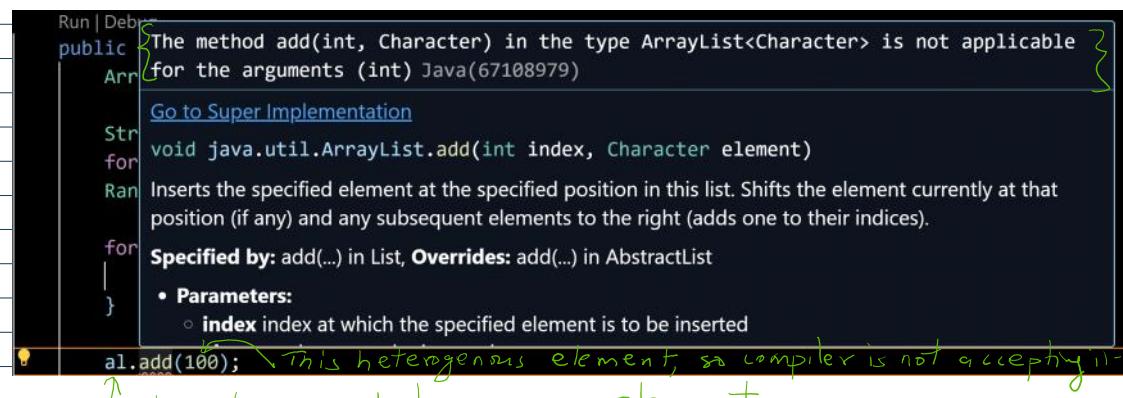
Runtime exception: NullPointerException.

ii. public static void sort(List l, Comparator c); → C.S.O

```
package UtilityClasses;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;
public class CollectionsSortDemo {
    public static void main(String[] args) {
        ArrayList<Character> al = new ArrayList<Character>();
        String alpha = "";
        for(char i = 'A'; i <= 'Z'; alpha+=i,i++);
        Random r = new Random();
        for(int i = 0; i < 10 + r.nextInt(10); i++){
            al.add(alpha.charAt(r.nextInt(26)));
        }
        System.out.println("Before sorting: " + al);
        Collections.sort(al);
        System.out.println("After sorting: " + al);
    }
}
```

Output:

```
Before sorting: [K, M, B, L, Y, A, C, V, M, B, X, V]
After sorting: [A, B, B, C, K, L, M, M, V, V, X, Y]
```



- **index** index at which the specified element is to be inserted

*al.add(100); This heterogeneous element, so compiler is not accepting it.*

*It will store only homogenous elements.*

```
package UtilityClasses;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;
public class CollectionsSortDemo {
    public static void main(String[] args) {
        ArrayList<Character> al = new ArrayList<Character>();
        String alpha = "";
        for(char i = 'A'; i <= 'Z'; alpha+=i,i++);
        Random r = new Random();
        for(int i = 0; i < 10 + r.nextInt(10); i++){
            al.add(alpha.charAt(r.nextInt(26)));
        }
        al.add(100);
        System.out.println("Before sorting: " + al);
        Collections.sort(al);
        System.out.println("After sorting: " + al);
    }
}
```

Output:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
  The method add(int, Character) in the type ArrayList<Character> is not applicable for the arguments (int)
  at UtilityClasses.CollectionsSortDemo.main(CollectionsSortDemo.java:18)
```

Customized Sorting Order:

```
package UtilityClasses;
import java.util.Comparator;
public class MyComparator implements Comparator{
    @Override
    public int compare(Object o1, Object o2) {
        Character c1 = (Character)o1;
        Character c2 = (Character)o2;
        return c2.compareTo(c1);
    }
}
```

```
package UtilityClasses;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;
public class SortWithComparatorDemo {

    public static void main(String[] args) {
        ArrayList<Character> al = new ArrayList<Character>();
        String alpha = "";
        for(char i = 'A'; i <= 'Z'; alpha+=i,i++);
        Random r = new Random();
        for(int i = 0; i < 10 + r.nextInt(10); i++){
            al.add(alpha.charAt(r.nextInt(26)));
        }
    }
}
```

```

        }
        System.out.println("Before sorting: " + al);
        Collections.sort(al, new MyComparator());
        System.out.println("After sorting: " + al);
    }
}

```

Output:

```

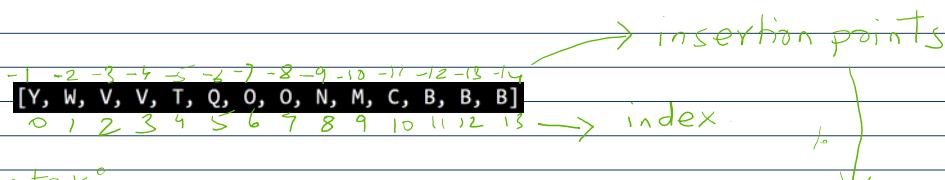
Before sorting: [M, Y, V, Q, B, O, V, B, O, C, T, N, W, B]
After sorting: [Y, W, V, V, T, Q, O, O, N, M, C, B, B, B]

```

### Searching elements in List

starting element = 0

last element = (-1)  
(-n+1)



#### ① Syntax:-

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

```
Collections.binarySearch (l, "Z"); // -1
```

```
Collections.binarySearch (l, "J"); // -11
```

```
Collections.binarySearch (l, "N"); // 8
```

#### ② Syntax:-

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

↑  
sorting order of the list is customized  
by the comparator

### Conclusion:-

1. The above search methods 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 the target element in sorted list.
5. Before calling binarySearch() method, compulsory list should be sorted, otherwise we will get unpredictable results.
6. If the list is not sorted according to the comparator then it

sorted, otherwise we will get unpredictable results.

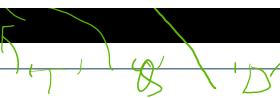
6. If the list is sorted according to the Comparator, then at the time of calling binarySearch() method, we have to pass same Comparator object, otherwise we will get unpredictable results.

```
package UtilityClasses;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;
public class CollectionSearchDemo {
    public static void main(String[] args) {
        ArrayList<Character> al = new ArrayList<>();
        String bigAlpha="";
        for (char i = 'A'; i <= 'Z'; i++) {
            bigAlpha += i;
        }
        Random r = new Random();
        for (int i = 0; i < 10 + r.nextInt(20); i++) {
            al.add(bigAlpha.charAt(r.nextInt(26)));
        }
        System.out.println("List before sorting: " + al);
        Collections.sort(al);
        System.out.println("List after sorting: " + al );
        int in = Collections.binarySearch(al, 'D');
        if(in > -1){
            System.out.println("Element is found at index: "+in);
        }
        else{
            System.out.println("Element insertion point is: "+in);
        }
        in = Collections.binarySearch(al, 'Q');
        if(in > -1){
            System.out.println("Element is found at index: "+in);
        }
        else{
            System.out.println("Element insertion point is: "+in);
        }
        in = Collections.binarySearch(al, 'T');
        if(in > -1){
            System.out.println("Element is found at index: "+in);
        }
        else{
            System.out.println("Element insertion point is: "+in);
        }
    }
}
```

Output:

```
List before sorting: [I, I, V, L, L, O, A, H, F, E, J, P, L, N, A, J, G]
List after sorting: [A, E, F, G, H, I, I, J, J, L, L, N, O, P, V]
Element insertion point is: 3
Element insertion point is: -17
Element insertion point is: -17
```

```
Element insertion point is: -17
```



Another execution:

```
List before sorting: [P, G, X, A, J, Y, V, E, S, Q, J, B, B, Z, W, C, W, V, L, N, K]
List after sorting: [A, B, B, C, E, G, J, J, K, L, N, P, Q, S, V, V, W, W, X, Y, Z]
Element insertion point is: -5
Element is found at index: 12
Element insertion point is: -15
```

Collections.binarySearch(list, se, comparator)

```
package UtilityClasses;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;
public class CollectionSearchDemoUsingComparator {
    public static void main(String[] args) {
        ArrayList<Character> al = new ArrayList<>();
        String bigAlpha="";
        for (char i = 'A'; i <= 'Z'; i++) {
            bigAlpha += i;
        }
        Random r = new Random();
        for (int i = 0; i < 10 + r.nextInt(20); i++) {
            al.add(bigAlpha.charAt(r.nextInt(26)));
        }
        System.out.println("List before sorting: " + al);
        MyComparator c = new MyComparator();
        Collections.sort(al, c);
        System.out.println("List after sorting: " + al );
        int in = Collections.binarySearch(al, 'D', c);
        if(in > -1){
            System.out.println("Element is found at index: "+in);
        }
        else{
            System.out.println("Element insertion point is: "+in);
        }
        in = Collections.binarySearch(al, 'Q', c);
        if(in > -1){
            System.out.println("Element is found at index: "+in);
        }
        else{
            System.out.println("Element insertion point is: "+in);
        }
        in = Collections.binarySearch(al, 'T', c);
        if(in > -1){
            System.out.println("Element is found at index: "+in);
        }
        else{
            System.out.println("Element insertion point is: "+in);
        }
    }
}

package UtilityClasses;
import java.util.Comparator;
public class MyComparator implements Comparator{
    @Override
```

```

public int compare(Object o1, Object o2) {
    Character c1 = (Character)o1;
    Character c2 = (Character)o2;
    return c2.compareTo(c1);
}
}

```

Output:

```

List before sorting: [R, N, W, U, E, R, A, Z, E, C, K]
List after sorting: [Z, W, U, R, R, N, K, E, E, C, A]
Element insertion point is: -10
Element insertion point is: -6
Element insertion point is: -4

```

Another execution:

```

List before sorting: [A, U, A, E, B, H, S, B, B, M, D, B, S, U, L, X, F, F, L, U, Q]
List after sorting: [X, U, U, U, S, S, Q, M, L, L, H, F, F, E, D, B, B, B, B, A, A]
Element is found at index: 14
Element is found at index: 6
Element insertion point is: -5

```

#### Reversing elements in List

```
public static void reverse(List L);
```

```

package UtilityClasses;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;
public class ReversingListDemo {
    public static void main(String[] args) {

        ArrayList<Character> al = new ArrayList<>();
        String bigAlpha="";
        for (char i = 'A'; i <= 'Z'; i++) {
            bigAlpha +=i;
        }
        Random r = new Random();
        for (int i = 0; i < 10 + r.nextInt(20); i++) {
            al.add(bigAlpha.charAt(r.nextInt(26)));
        }
        System.out.println("List before reversing: "+ al);
        Collections.reverse(al);
        System.out.println("List after reverse: " + al);
    }
}

```

Output:

```

List before reversing: [I, L, R, D, Q, Q, S, E, G, C, J, V]
List after reverse: [V, J, C, G, E, S, Q, Q, D, R, L, I]

```

Another implementation:

```

package UtilityClasses;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;
public class ReversingListDemo {
    public static void main(String[] args) {

```

```

ArrayList<Character> al = new ArrayList<>();
String bigAlpha="";
for (char i = 'A'; i <= 'Z'; i++) {
    bigAlpha +=i;
}
Random r = new Random();
for (int i = 0; i < 10 + r.nextInt(20); i++) {
    al.add(bigAlpha.charAt(r.nextInt(26)));
}
System.out.println("List before reversing: " + al);
Collections.reverse(al);
System.out.println("List after reverse: " + al);
Collections.sort(al);
System.out.println("Default natural sorting order: " + al);
Collections.reverse(al);
System.out.println("Descending order: " + al);
}
}

```

Output:

```

List before reversing: [F, D, S, Y, R, O, Z, O, N, G, P, V, R, Y, J]
List after reverse: [J, Y, R, V, P, G, N, O, Z, O, R, Y, S, D, F]
Default natural sorting order: [D, F, G, J, N, O, O, P, R, R, S, V, Y, Y, Z]
Descending order: [Z, Y, Y, V, S, R, R, P, O, O, N, J, G, F, D]

```

reverseOrder() method is used to get the reversed comparator object.

```
Comparator c1 = Collections.reverseOrder(Comparator c);
```

↑ ascending order      ↑ descending order

```

package UtilityClasses;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Random;
public class CollectionsReverseOderUsingComparator {
    public static void main(String[] args) {

        ArrayList<Character> al = new ArrayList<>();
        String bigAlpha="";
        for (char i = 'A'; i <= 'Z'; i++) {
            bigAlpha +=i;
        }
        Random r = new Random();
        for (int i = 0; i < 10 + r.nextInt(20); i++) {
            al.add(bigAlpha.charAt(r.nextInt(26)));
        }
        System.out.println("List before sorting: "+al);
        Collections.sort(al, new MyComparator());
        System.out.println("List in descending order: "+al);
        Comparator c = Collections.reverseOrder( new MyComparator());
        Collections.sort(al, c);
        System.out.println("List in ascending order: "+al);
    }
}

```

Output:

```
List before sorting: [O, B, R, U, X, Y, P, U, H, B, G, D]
List in descending order: [Y, X, U, U, R, P, O, H, G, D, B, B]
List in ascending order: [B, B, D, G, H, O, P, R, U, U, X, Y]
```

## Arrays the Utility Class

Arrays class is an utility class to define several utility methods for arrays or array.

1. Sorting elements of Array: Arrays class defines the following sort methods to sort elements of primitive type and Object type.
  - a. `public static void sort(primitive_data_type[] arr);` to sort according to DNSO.
  - b. `public static void sort(Object[] arr);` to sort according to CSO.
  - c. `public static void sort(primitive_data_type[] arr, Comparator c);` to sort according to CSO.