Java Collection Classes

Java Collections class is a part of collection framework. This class is designed to provide methods for searching, sorting, copying etc. It consists exclusively of built-in static methods that operate on or return collections. It contains polymorphic algorithms that operate on collections.

This class is located into **java.util** package. The declaration of this class is given below.

Collections Class Declaration

```
public class Collections extends Object
```

Copy

It inherits Object class and all the methods of this class throw a **NullPointerException** if the object is null.

Collections Methods

This table contains commonly used methods of Collections class.

Method	Description
addAll()	It adds all of the specified elements to the specified collection.
binarySearch()	It searches the list for the specified object and returns their position in a sorted list.
copy()	It copies all the elements from one list into another list.
disjoint()	It returns true if the two specified collections have no elements in common.
emptyEnumeration()	It fetches an enumeration that has no elements.
emptyIterator()	It fetches an Iterator that has no elements.
emptyList()	It fetches a List that has no elements.

emptyListIterator()	It fetches a List Iterator that has no elements.
emptyMap()	It returns an empty map which is immutable.
emptyNavigableMap()	It returns an empty navigable map which is immutable.
emptyNavigableSet()	It returns an empty navigable set which is immutable in nature.
emptySet()	It returns the set that has no elements.
emptySortedMap()	It returns an empty sorted map which is immutable.
emptySortedSet()	It is used to get the sorted set that has no elements.
enumeration()	It is used to get the enumeration over the specified collection.
fill()	It is used to replace all of the elements of the specified list with the specified elements.
list()	It is used to get an array list containing the elements returned by the specified enumeration in the order in which they are returned by the enumeration.
max()	It is used to get the maximum value of the given collection.

min()	It is used to get the minimum value of the given collection.
nCopies()	It is used to get an immutable list consisting of n copies of the specified object.
replaceAll()	It is used to replace all occurrences of one specified value in a list with the other specified value.
reverse()	It is used to reverse the order of the elements in the specified list.
reverseOrder()	It is used to get the comparator that imposes the reverse of the natural ordering on a collection.
rotate()	It is used to rotate the elements in the specified list by a given distance.
shuffle()	It is used to randomly reorders the specified list elements using a default randomness.
sort()	It is used to sort the elements presents in the specified list of collection in ascending order.
swap()	It is used to swap the elements at the specified positions in the specified list.
synchronizedCollection()	It is used to get a synchronized (thread-safe) collection backed by the specified collection.

synchronizedList()	It is used to get a synchronized (thread-safe) collection backed by the specified list.
synchronizedMap()	It is used to get a synchronized (thread-safe) map backed by the specified map.
synchronizedNavigableMap()	It is used to get a synchronized (thread-safe) navigable map backed by the specified navigable map.
synchronizedNavigableSet()	It is used to get a synchronized (thread-safe) navigable set backed by the specified navigable set.
synchronizedSet()	It is used to get a synchronized (thread-safe) set backed by the specified set.
synchronizedSortedMap()	It is used to get a synchronized (thread-safe) sorted map backed by the specified sorted map.
synchronizedSortedSet()	It is used to get a synchronized (thread-safe) sorted set backed by the specified sorted set.

Example: Sorting List

In this example, we are using sort() method of Collections class that is used to sort elements of a collection. Here we are using Arralist class that stores integer type object and sorted.

```
import java.util.*;
public class Demo {
    public static void main(String a[]){
      // Creating ArrayList
        ArrayList<Integer> list = new ArrayList<>();
        // Adding elements
        list.add(100);

        list.add(2);

        list.add(66);

        list.add(22);

        list.add(10);

        // Displaying list

        System.out.println(list);

        // Sorting list

        Collections.sort(list);

        // Displaying sort data

        System.out.println(list);

    }

}
```

Copy

```
[100, 2, 66, 22, 10]

[2, 10, 22, 66, 100]
```

Example: Finding min and max elements

The collections class provides two methods `max()` and `min()` that can be used to fetch max and min values from a collection. See the below example.

```
import java.util.*;

public class Demo {

    public static void main(String a[]){

      // Creating ArrayList

        ArrayList<Integer> list = new ArrayList<>();

        // Adding elements

        list.add(100);

        list.add(2);

        list.add(66);

        list.add(22);
        list.add(10);

        // Displaying list

        System.out.println(list);

        // Find min element

        int min = Collections.min(list);

        // Find max element

        int max = Collections.max(list);

        // Displaying data

        System.out.println("Minimum element : "+ min);

        System.out.println("Maximum element : "+ max);

    }

}
```

Copy

```
[100, 2, 66, 22, 10]

Minimum element : 2

Maximum element : 100
```

Example: Swapping Elements

To swap elements, we don't need write logic code. Collections class provides built-in swap method that can be used to swap elements from one position to another in a collection. The swap() method takes three arguments: first is reference of object, second is index of first elements and third is index of second elements to be swapped. See the below example.

```
import java.util.*;

public class Demo {

    public static void main(String a[]){

      // Creating ArrayList

        ArrayList<Integer> list = new ArrayList<>();

        // Adding elements

        list.add(100);

        list.add(2);

        list.add(66);
        list.add(22);

        list.add(10);

        // Displaying list

        System.out.println(list);

        // Swapping elements

        Collections.swap(list, 0, 4); // 100 is swapped by 10

        System.out.println("List after swapping : "+ list);

    }

}
```

Copy

```
[100, 2, 66, 22, 10]

List after swapping : [10, 2, 66, 22, 100]
```

Example: Reverse the list

Collections class provides a static method reverse() that is used to get a collection in reverse order. In the below example, we are getting list in reverse order using the reverse() method.

```
import java.util.*;

public class Demo {

    public static void main(String a[]){

      // Creating ArrayList

        ArrayList<Integer> list = new ArrayList<>();

        // Adding elements

        list.add(100);

        list.add(2);

        list.add(66);

        list.add(22);

        list.add(10);

        // Displaying list

        System.out.println(list);

        // Reverse the list
```

```java
        Collections.reverse(list);

        // Displaying data

        System.out.println("List in reverse order "+list);

    }

}
```

Copy

```
OUTPUT

[100, 2, 66, 22, 10]

List in reverse order [10, 22, 66, 2, 100]
```