Sorting an ArrayList in ascending order

The `Collections` class contains a `sort(List<T> list)` method, which is used to sort an **ArrayList**. This method takes an **ArrayList** as input and sorts it in ascending order.

In the `sort(List<T> list)` method, **T** represents the type of object that is stored in the **ArrayList**. The `Collections.sort(List<T> t)` method takes an **ArrayList** of type **T** objects as the input. It is a must that **T** should implement the Comparable interface; otherwise, the code will not compile.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class ArrayListDemo {

    public static void main(String args[]) {
        List<Integer> list = new ArrayList<>();
        list.add(34);
        list.add(12);
        list.add(9);
        list.add(76);
        list.add(29);
        list.add(75);

        Collections.sort(list);
        System.out.println("ArrayList in asc order: " + list);
    }
}
```

Run Save Reset

There is another way to sort an **ArrayList** using **streams**, which is a Java 8 feature. Once we create a stream then we can use the `sorted()` method of the **Stream** class, which returns the stream of objects in sorted order.

```
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

public class ArrayListDemo {

    public static void main(String args[]) {
        List<Integer> list = new ArrayList<>();
        list.add(34);
        list.add(12);
        list.add(9);
        list.add(76);
        list.add(29);
        list.add(75);

        List<Integer> sortedList = list.stream().sorted().collect(Collectors.toList());
        System.out.println("ArrayList in asc order: " + sortedList);
    }
}
```

Run Save Reset

Sorting an ArrayList in descending order

There is another overloaded version of the `sort()` method, i.e., `sort(List<T> list, Comparator<? super T> c)`, which takes a List and Comparator object as the input.

We will discuss Comparator in detail in upcoming lessons, which will make the below example more clear.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class ArrayListDemo {

    public static void main(String args[]) {
        List<Integer> list = new ArrayList<>();
        list.add(34);
        list.add(12);
        list.add(9);
        list.add(76);
        list.add(29);
        list.add(75);

        Collections.sort(list, Collections.reverseOrder());
        System.out.println("ArrayList in desc order: " + list);
    }
}
```

Run Save Reset

The ArrayList can be sorted in reverse order using streams by passing `Comparator.reverseOrder()` to the `sorted()` method.

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.stream.Collectors;

public class ArrayListDemo {

    public static void main(String args[]) {
        List<Integer> list = new ArrayList<>();
        list.add(34);
        list.add(12);
        list.add(9);
        list.add(76);
        list.add(29);
        list.add(75);

        List<Integer> sortedList = list.stream()
                .sorted(Comparator.reverseOrder())
                .collect(Collectors.toList());
        System.out.println("ArrayList in asc order: " + sortedList);
    }
}
```

Run Save Reset

In Java 8, the `sort(Comparator<? super E> c)` method was added to the **List** interface. If we look at the implementation of the `Collections.sort()` method, then we will find that it internally calls the `sort()` method of the List interface. The code is shown below.

```
public static <T extends Comparable<? super T>> void sort(List<T> list) {
    list.sort(null);
}
```

Let's see how the `sort()` method of the **List** interface sorts a list. When the `sort()` method is called, an array containing all elements in this list is created and sorted. After sorting the array, the list is iterated and each element is reset from the corresponding position in the array.

The elements are first copied to an array and then sorted because it takes less time to sort a linked list using this approach.