Sublist() Method in Java: ArrayList and List

Description: Sublist is a Java method in the List interface that can be used on ArrayLists, LinkedLists, Vectors, and Stacks. It returns a view of the portion between two indexes specified by arguments.

What is the subList() method?

The Collections Framework is a very popular component in Java API. List interface and ArrayList class are probably the most important tools in the Collections Framework. subList is a method in the List interface that lets you create a new list from a portion of an existing list. However, this newly created list is only a view with a reference to the original list.

For example, take the list of [1,2,3,4,5,6]. Suppose that you want to create a new list without the first and last elements. In such a scenario, the list.subList() method will help you. subList(fromIndex, toIndex) method has only one form and it takes two arguments, which are the first index**(fromIndex)** and the last index**(toIndex)**. It will return the portion between the **fromIndex** and the **toIndex** as a new list.

There is an important point to remember. The newly created list will include the fromIndex and exclude the toIndex.

So the algorithm for the above scenario will be something like this.

List = [1,2,3,4,5,6]
newList = List.subList(1,5)

Since subList is a method of List interface, you can use it on ArrayList, LinkedList, Stack, and Vector objects. However, in this article, we will mainly focus on ArrayList and LinkedList objects.

Example of the subList method on an ArrayList object.

We are declaring an ArrayList of countries. Then we try to return the portion between the 2nd and 4th elements.

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
         // create an ArrayList
        ArrayList<String> list = new ArrayList<String>();
```

```
          // add values to ArrayList
          list.add("USA");
          list.add("UK");
          list.add("France");
          list.add("Germany");
          list.add("Russia");
         System.out.println("List of the countries:" + list);
          //Return the subList : 1 inclusive and 3 exclusive
         ArrayList<String> new_list = new
ArrayList<String>(list.subList(1, 3));
         System.out.println("The subList of the list: "+new_list);
     }
 }
```

The output of the above code will be

List of the countries:[USA, UK, France, Germany, Russia]
The subList of the list: [UK, France]

In an ArrayList, the index value of the first element is 0. Therefore, the index values of the second and fourth elements are 1 and 3 respectively. So, we invoke the sublist() method as **list.subList(1, 3)**.

However, remember that the subList method returns the portion excluding the toIndex which is the fourth element**("Germany")** in this case. Thus, it will output "**UK"** and "**France"** only. Since the returned output is a List itself, you can call any List methods directly on it.

So what will happen if we use the same index for both the parameters? Will that index be included or excluded in the returned list? Let's find out.

```
//execute subList() method with the same argument for both parameters.
ArrayList<String> new_list2 = new ArrayList<String>(list.subList(3, 3));
System.out.println("The subList of the list: "+new_list2);
```

The Output is
The subList of the list: [ ]

The output is an empty list. Even though fromIndex selects the 4th element, the subList() method will remove it as it is also the toIndex.

Example of the subList method on a LinkedList object.

In this example, we will use the sublist method on a LinkedList element. Again, It will return the list between the specified index fromIndex(inclusive) and toIndex(exclusive).

Remember that we said the list returned by the subList() method is only a view that has a reference to the original list. If you do any changes to the sublist, it will affect the original list as well. We will test that too in this example.

```java
import java.util.LinkedList;
import java.util.Iterator;
import java.util.List;

public class Main {

 public static void main(String[] args) {

    // Create a LinkedList
    LinkedList<String> linkedlist = new LinkedList<String>();

    // Add elements to LinkedList
    for(int i = 0; i<7; i++){
      linkedlist.add("Node "+ (i+1));
    }

    // Displaying LinkedList elements
    System.out.println("Elements of the LinkedList:");
    Iterator it= linkedlist.iterator();
    while(it.hasNext()){
       System.out.print(it.next()+ " ");
    }

    // invoke subList() method on the linkedList
    List sublist = linkedlist.subList(2,5);

    // Displaying SubList elements
    System.out.println("\nElements of the sublist:");
    Iterator subit= sublist.iterator();
    while(subit.hasNext()){
       System.out.print(subit.next()+" ");
    }

    /* The changes you made to the sublist will affect the      original
LinkedList
     * Let’s take this example - We
     * will remove the element "Node 4" from the sublist.
     * Then we will print the original LinkedList.
     * Node 4 will not be in the original LinkedList too.
     */
    sublist.remove("Node 4");
    System.out.println("\nElements of the LinkedList LinkedList After
removing Node 4:");
    Iterator it2= linkedlist.iterator();
    while(it2.hasNext()){
       System.out.print(it2.next()+" ");
    }
 }
}
```

The output will look like this:

```
Elements of the LinkedList:
Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7
Elements of the sublist:
Node 3 Node 4 Node 5
Elements of the LinkedList LinkedList After removing Node 4:
Node 1 Node 2 Node 3 Node 5 Node 6 Node 7
```

What will happen if the indexes are out of bound in subList()?

The subList method returns two types of exceptions. Let’s have a look at them.

Consider a situation if the specified indexes are out of the range of the List element **(fromIndex < 0 || toIndex > size)**. Then it will throw an IndexOutOfBoundExecption.

```
//using subList() method with fromIndex <0
```

```
ArrayList<String> new_list2 = new ArrayList<String>(list.subList(-1, 3));
System.out.println("Portion of the list: "+new_list2);

Exception in thread "main" java.lang.IndexOutOfBoundsException: fromIndex
= -1

// using subList() method with toIndex > size
ArrayList<String> new_list2 = new ArrayList<String>(list.subList(3, 6));
System.out.println("Portion of the list: "+new_list2);

Exception in thread "main" java.lang.IndexOutOfBoundsException: toIndex =
6
```

Also, if the fromIndex is greater than the toIndex **(fromIndex > toIndex)**, the subList() method throws an IllegalArgumentException error.

```
//If fromIndex > toIndex
ArrayList<String> new_list2 = new ArrayList<String>(list.subList(5, 3));
System.out.println("Portion of the list: "+new_list2);

Exception in thread "main" java.lang.IllegalArgumentException:
fromIndex(5) > toIndex(3)
```

Conclusion

In this article, we discussed the subList() method and how to use it. subList() method eliminates the need for explicit range operations (It’s a type of operations that commonly exist for arrays). The most important thing to remember is the subList method doesn’t return a new instance but a view with a reference to the original list. Therefore, overusing the subList method on the same list can cause a thread stuck in your Java application.