2 Ways to Remove Elements/Objects From ArrayList in Java [Example]

There are *two ways to remove objects from ArrayList in Java*, first, by using the `remove()` method, and second by using `Iterator`. ArrayList provides overloaded `remove()` method, one accepts the index of the object to be removed i.e. `remove(int index)`, and the other accept objects to be removed, i.e. `remove(Object obj)`. The rule of thumb is, If you know the index of the object, then use the first method which accept index, otherwise use the second method. By the way, you must remember to use ArrayList remove methods, only when you are not iterating over ArrayList if you are iterating then use `Iterator.remove()` method, failing to do so may result in `ConcurrentModificationException` in Java.

Another gotcha can have occurred due to autoboxing. If you look closely that two remove methods, `remove(int index)` and `remove(Object obj)` are indistinguishable if you are trying to remove from an ArrayList of `Integers`.

Suppose you have three objects in ArrayList i.e. `[1,2,3]` and you want to remove the second object, which is `2`. You may call `remove(2)`, which is actually a call to `remove(Object)` if consider autoboxing, but will be interpreted as a call to remove 3rd element, by interpreting as `remove(index)`.

I have discussed this problem earlier in my article about best practices to follow while overloading methods in Java. Because of lesser-known widening rules and autoboxing, the poorly overloaded methods can create a lot of ambiguity.

Code Example To Remove Elements from ArrayList in Java

Let's test the above theory with a simple code example of `ArrayList` with Integers. The following program has an ArrayList of Integers containing 1, 2, and 3 i.e. `[1, 2, 3]`, which corresponds exactly to the index.

```
package test;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author http://java67.blogspot.com
 */

public class JavaTutorial{

    /**
     * @param args the command line arguments
     */

    public static void main(String[] args) {

        List<Integer> numbers = new ArrayList<Integer>();
        numbers.add(1);
        numbers.add(2);
        numbers.add(3);

        System.out.println("ArrayList contains : " + numbers);

        // Calling remove(index)
        numbers.remove(1); //removing object at index 1 i.e. 2nd Object, which is 2

        //Calling remove(object)
        numbers.remove(3);

    }

}

Output:
ArrayList contains : [1. 2, 3]
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index: 3, Size: 2
        at java.util.ArrayList.rangeCheck(ArrayList.java:635)
        at java.util.ArrayList.remove(ArrayList.java:474)
        at test.Test.main(Test.java:33)

Java Result: 1
```

You can see that the second call is also treated as `remove(index)`. The best way to remove ambiguity is to take out autoboxing and provide an actual object, as shown below.

```
System.out.println("ArrayList Before : " + numbers);
```

```
// Calling remove(index)
numbers.remove(1); //removing object at index 1 i.e. 2nd Object, which is 2

//Calling remove(object)
numbers.remove(new Integer(3));

System.out.println("ArrayList After : " + numbers);

Output :
ArrayList Before : [1, 2, 3]
ArrayList After : [1]
```

This time, it works, but I am afraid of lazy developers like me, which take autoboxing for granted. Now let's take a look at removing the object from ArrayList while Iterating over them. You must be familiar with Iterator in Java, before proceeding further.

Remove Object From ArrayList using Iterator

This is actually a subtle detail of Java programming, not obvious for first-timers, as the compiler will not complain, even if you use the `remove()` method from `java.util.ArrayList`, while using Iterator.

You will only realize your mistake, when you see `ConcurrentModificationException`, which itself is misleading and you may spend countless hours finding another thread, which is modifying that `ArrayList`, because of Concurrent word. Let's see an example.

```
public static void main(String[] args) {

        List<Integer> numbers = new ArrayList<Integer>();
        numbers.add(101);
        numbers.add(200);
        numbers.add(301);
        numbers.add(400);

        System.out.println("ArrayList Before : " + numbers);

        Iterator<Integer> itr = numbers.iterator();

        // remove all even numbers
        while (itr.hasNext()) {
            Integer number = itr.next();

            if (number % 2 == 0) {
                numbers.remove(number);
            }
        }

        System.out.println("ArrayList After : " + numbers);
```

```
    }
```

```
Output :

ArrayList Before : [101, 200, 301, 400]

Exception in thread "main" java.util.ConcurrentModificationException
        at java.util.ArrayList$Itr.checkForComodification(ArrayList.java:859)
        at java.util.ArrayList$Itr.next(ArrayList.java:831)
        at Testing.main(Testing.java:28)
```

You can `ConcurrentModificationException`, due to call to `remove()` method from `ArrayList`. This is easy in simple examples like this, but in a real project, it can be really tough. Now, to fix this exception, just replace the call of `numbers.remove()` to `itr.remove()`, this will remove the current object you are Iterating, as shown below :

```
System.out.println("ArrayList Before : " + numbers);

Iterator<Integer> itr = numbers.iterator();

// remove all even numbers
while (itr.hasNext()) {
    Integer number = itr.next();

       if (number % 2 == 0) {
       itr.remove();
    }

}

System.out.println("ArrayList After : " + numbers);

Output
ArrayList Before : [101, 200, 301, 400]
ArrayList After : [101, 301]
```

That's all on this post about **How to remove objects from ArrayList in Java**. We have learned two ways to remove an object or element from ArrayList. By the way, You should always use `remove(index)` to delete objects, if you are not iterating, otherwise, always use Iterator's `remove()` method for removing objects from ArrayList.

By the way, the above tips will work with any index-based List implementation.