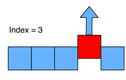
Removing an element from an ArrayList

Elements can be removed from an **ArrayList** in the following ways.

Removing an element at a particular index

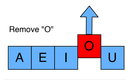
We can use the `remove(int index)` method to remove an element at a particular index. The index should be less than the size of **ArrayList**, otherwise, `IndexOutOfBoundsException` is thrown.



Removing element from a particular index

Removing a particular element from the ArrayList

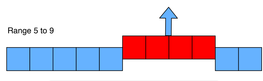
We can also specify the element that we want to remove using the `remove(Object o)` method, and the first occurrence of that element will be removed.



Removing a specific element

Removing all the elements within a range

Let's suppose we need to remove all the elements from index 5 to 9. This can be done using the `removeRange(int fromIndex, int toIndex)` method. This method will remove, from this list, all of the elements whose index is between *fromIndex*, inclusive, and *toIndex*, exclusive. Please note that this method is not defined in the List class. So, it can be used only when the reference type is also **ArrayList** and not List.



Removing elements within a given range

Removing all the elements within a given Collection

We can use the `removeAll(Collection<?> c)` method to remove, from the given list, all of the elements that are contained in the specified collection.

Removing all the elements from the ArrayList

We can use the `clear()` method to remove all the elements from the **ArrayList**.

> We saw that `remove(int index)` removes a method at the given index and `remove(Object o)` removes the given object from the **ArrayList**. Suppose we have an **ArrayList** that contains five elements i.e [13, 21, 43, 2, 9]. Now, if we do `list.remove(2)`, then which overloaded method will be called. Will `remove(int index)` be called or `remove(Object o)` be called? `remove(int index)` will be called because we are passing a primitive to remove method. If we want to delete element 2, we should call `remove(new Integer(2))` because elements are stored in an ArrayList as objects and not primitives.

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

public class ArrayListDemo {

    public static void main(String args[]) {
        List<Integer> list = new ArrayList<>();
        list.add(10);
        list.add(20);
        list.add(30);
        list.add(40);
        list.add(50);
        list.add(60);
        list.add(70);
        list.add(80);

        System.out.println(list);

        list.remove(1);  // This will remove the element at index 1 i.e 20.
        System.out.println(list);

        list.remove(new Integer(30)); // This will remove 30 from the list
        System.out.println(list);

        list.clear(); //This will remove all the elements from the list.
        System.out.println(list);
    }
}
```

Run Save Reset

Replacing all the elements of the ArrayList

A new method, `replaceAll(UnaryOperator<E> operator)`, was added in Java 8. This method takes a single UnaryOperator type argument. The UnaryOperator interface is a functional interface that has a single abstract method, `apply()`, that returns a result of the same object type as the operand.

Let's say we have an **ArrayList** that contains String objects; we need to make all the elements in this list uppercase. In this case, we can use the `replaceAll()` method and provide it with a lambda expression that converts each element into uppercase.

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

public class ArrayListDemo {

    public static void main(String args[]) {
        List<String> list = new ArrayList<>();
        list.add("apple");
        list.add("banana");

        list.replaceAll(element) -> element.toUpperCase());

        System.out.println(list);
    }
}

```

Run Save Reset

Additional operations on ArrayList

Updating an element in ArrayList

To update an element in **ArrayList**, the `set(int index, E e)` method can be used. This method takes the index, which needs to be updated and a new value.

Checking if an element is present in the ArrayList

To check if an element is present in the list, we can use the `contains(Object o)` method. This method returns `true` if the element is present in the list; otherwise, it returns `false`.

If we need the index of the first occurrence of the element, then the `indexOf(Object o)` method can be used. And if we need the last occurrence of the element, the `lastIndexOf(Object o)` can be used.

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

public class ArrayListDemo {

    public static void main(String args[]) {
        List<Integer> list = new ArrayList<>();
        list.add(10);
        list.add(20);
        list.add(30);
        list.add(40);
        list.add(10);

        list.set(1, 100);

    System.out.println(list);

        if (list.contains(30)) {
            System.out.println("List contains 30");
        }

        System.out.println("Index of first occurence of 10 is " + list.indexOf(10));
        System.out.println("Index of last occurence of 10 is " + list.lastIndexOf(10));
    }
}

```

Run Save Reset