Iteration using `forEach()`

We can use the `forEach(Consumer<? super E> action)` method to iterate over a CopyOnWriteArrayList. This method was added in Java 8 and takes a lambda expression of type Consumer as the parameter.

```
import java.util.List;
import java.util.concurrent.CopyOnWriteArrayList;

public class CopyOnWriteArrayListDemo {

    public static void main(String args[]) {
        List<String> list = new CopyOnWriteArrayList<>();
        list.add("Apple");
        list.add("Banana");
        list.add("Orange");

        list.forEach(System.out::println);
    }
}
```

Run Save Reset

Iteration using `iterator()`

The `iterator()` method returns an iterator that provides a snapshot of the state of the list when the iterator was constructed. No synchronization is needed while traversing the iterator because the iteration is being done on a snapshot.

In the ArrayList iteration lesson, we saw that if after creating an iterator, an element is added to the ArrayList then `ConcurrentModificationException` is thrown. This is not the case with CopyOnWriteArrayList because the addition of elements takes place on the copy and not the actual array. But the iterator will be able to access only the elements which were present at the time of the creation of the iterator. Let's understand this further through an example.

```
import java.util.Iterator;
import java.util.List;
import java.util.concurrent.CopyOnWriteArrayList;

public class CopyOnWriteArrayListDemo {

    public static void main(String args[]) {
        List<String> list = new CopyOnWriteArrayList<>();
        list.add("Apple");
        list.add("Banana");
        list.add("Orange");

        //Created an iterator
        Iterator<String> itr = list.iterator();
        //Adding elements after creating iterator. ConcurrentModificationException will not be thrown.
        list.add("Papaya");

        //Iterating the list through the iterator that was created earlier. Papaya will not be present.
        while(itr.hasNext()) {
            System.out.println(itr.next());
        }

        System.out.println("Again getting the iterator");
        //Again creating the iterator. This time papaya will be present.
        itr = list.iterator();
        while(itr.hasNext()) {
            System.out.println(itr.next());
        }
```

Run Save Reset

There is one interesting thing about this iterator that makes it different from the iterator of other List implementations such as ArrayList or LinkedList. The iterator returned by the `iterator()` method of `CopyOnWriteArrayList` class does not support the remove method. In this lesson, we discussed that if we want to delete an element from the ArrayList while iterating, then we should use the iterator's remove method. The same does not hold true for a CopyOnWriteArrayList.

In CopyOnWriteArrayList, we can directly remove the element from the list while iterating and `ConcurrentModificationException` will not be thrown as shown in the below example.

> The reason for this is that we are iterating over a snapshot of the List but when we remove an element, it is removed from the copy of the List. So no `ConcurrentModificationException` is thrown.

```
import java.util.Iterator;
import java.util.List;
import java.util.concurrent.CopyOnWriteArrayList;

public class CopyOnWriteArrayListDemo {

    public static void main(String args[]) {
        List<String> list = new CopyOnWriteArrayList<>();
        list.add("Apple");
        list.add("Banana");
        list.add("Orange");

        //Created an iterator
        Iterator<String> itr = list.iterator();

        while(itr.hasNext()) {
            System.out.println(itr.next());
            list.remove("Orange");
        }
        System.out.println("Again creating the iterator");
        //Created an iterator
        itr = list.iterator();

        while(itr.hasNext()) {
            System.out.println(itr.next());

        }
```

Run Save Reset

The one last thing that we need to discuss is why the iterator of CopyOnWriteArrayList does not have the remove method. Firstly it is not needed as in a CopyOnWriteArrayList we can directly remove the element from the List without fearing any exception. Secondly, if there was a remove method in the `iterator()` then it will not be very useful. It will remove the element from the snapshot that is created for iteration. The actual array which holds the data will not be changed.