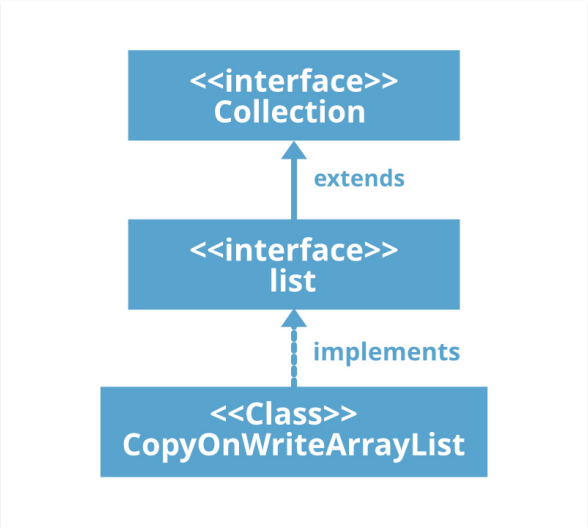
CopyOnWriteArrayList in Java

Last Updated : 05 Feb, 2021

CopyOnWriteArrayList class is introduced in JDK 1.5, which implements the **List interface**. It is an enhanced version of **ArrayList** in which all modifications (add, set, remove, etc) are implemented by making a fresh copy. It is found in **java.util.concurrent** package. It is a data structure created to be used in a concurrent environment.



Here are few points about CopyOnWriteArrayList:

- As the name indicates, CopyOnWriteArrayList creates a Cloned copy of underlying ArrayList, for every update operation at a certain point both will be synchronized automatically, which is taken care of by JVM. Therefore, there is no effect for threads that are performing read operation.
- It is costly to use because for every update operation a cloned copy will be created. Hence, CopyOnWriteArrayList is the best choice if our frequent operation is read operation.
- The underlined data structure is a grow-able array.
- It is a thread-safe version of ArrayList.
- Insertion is preserved, duplicates, null, and heterogeneous Objects are allowed.
- The main important point about CopyOnWriteArrayList is the Iterator of CopyOnWriteArrayList can not perform remove operation otherwise we get Run-time exception saying **UnsupportedOperationException.** add() and set() methods on CopyOnWriteArrayList iterator also throws **UnsupportedOperationException.** Also Iterator of CopyOnWriteArrayList will never throw **ConcurrentModificationException**.

Declaration:

public class CopyOnWriteArrayList<E> extends Object implements List<E>, RandomAccess, Clo neable, Serializable

Here, E is the type of elements held in this collection.

Note: The class implements **Serializable**, **Cloneable**, **Iterable<E>**, **Collection<E>**, List<E>, **RandomAccess** interfaces.

Constructors:

1. CopyOnWriteArrayList(): Creates an empty list.

```
CopyOnWriteArrayList c = new CopyOnWriteArrayList();
```

2. CopyOnWriteArrayList(Collection obj): Creates a list containing the elements of the specified collection, in the order, they are returned by the collection's iterator.

```
CopyOnWriteArrayList c = new CopyOnWriteArrayList(Collection obj);
```

3. CopyOnWriteArrayList(Object[] obj);: Creates a list holding a copy of the given array.

```
CopyOnWriteArrayList c = new CopyOnWriteArrayList(Object[] obj);
```

Example:

```
// Java program to illustrate
// CopyOnWriteArrayList class
import java.util.*;
import java.util.concurrent.CopyOnWriteArrayList;

public class ConcurrentDemo extends Thread {

    static CopyOnWriteArrayList<String> l
        = new CopyOnWriteArrayList<String>();

    public void run()
    {
        // Child thread trying to
        // add new element in the
        // Collection object
        l.add("D");
    }

    public static void main(String[] args)
        throws InterruptedException
    {
        l.add("A");
        l.add("B");
        l.add("c");

        // We create a child thread
        // that is going to modify
        // ArrayList l.
        ConcurrentDemo t = new ConcurrentDemo();
        t.run();
```

```
        Thread.sleep(1000);

        // Now we iterate through
        // the ArrayList and get
        // exception.
        Iterator itr = l.iterator();
        while (itr.hasNext()) {
            String s = (String)itr.next();
            System.out.println(s);
            Thread.sleep(1000);
        }
        System.out.println(l);
    }
}
```

Output

```
A
B
c
D
[A, B, c, D]
```

Iterating over CopyOnWriteArrayList: We can iterate over CopyOnWriteArrayList using iterator() method. The important point to be noted is that the iterator we create is an immutable snapshot of the original list. Because of this property, we can see that **GfG** is not printed at the first iteration.

```
// Java program to illustrate
// CopyOnWriteArrayList class
import java.io.*;
import java.util.*;
import java.util.concurrent.*;

class Demo {
    public static void main(String[] args)
    {

        CopyOnWriteArrayList<String> list
            = new CopyOnWriteArrayList<>();

        // Initial Iterator
        Iterator itr = list.iterator();
        list.add("GfG");
        System.out.println("List contains: ");
        while (itr.hasNext())
```

```
            System.out.println(itr.next());

        // iterator after adding an element
        itr = list.iterator();
        System.out.println("List contains:");
        while (itr.hasNext())
            System.out.println(itr.next());
    }
}
```

Output

```
List contains:
List contains:
GfG
```

Methods of CopyOnWriteArrayList:

METHOD	DESCRIPTION
add(E e)	Appends the specified element to the end of this list.
add(int index, E element)	Inserts the specified element at the specified position in this list.
addAll(Collection<? extends E> c)	Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
addAll(int index, Collection<? extends E> c)	Inserts all of the elements in the specified collection into this list, starting at the specified position.
addAllAbsent(Collection<? extends E> c)	Appends all of the elements in the specified collection that are not already contained in this list, to the end of this list, in the order that they are returned by the specified collection's iterator.
addIfAbsent(E e)	Appends the element, if not present.
clear()	Removes all of the elements from this list.
clone()	Returns a shallow copy of this list.
contains(Object o)	Returns true if this list contains the specified element.
containsAll(Collection<?> c)	Returns true if this list contains all of the elements of the specified collection.
equals(Object o)	Compares the specified object with this list for equality.

METHOD	DESCRIPTION
forEach(Consumer<? super E> action)	Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.
get(int index)	Returns the element at the specified position in this list.
hashCode()	Returns the hash code value for this list.
indexOf(E e, int index)	Returns the index of the first occurrence of the specified element in this list, searching forwards from the index, or returns -1 if the element is not found.
indexOf(Object o)	Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
isEmpty()	Returns true if this list contains no elements.
iterator()	Returns an iterator over the elements in this list in the proper sequence.
lastIndexOf(E e, int index)	Returns the index of the last occurrence of the specified element in this list, searching backward from the index, or returns -1 if the element is not found.
lastIndexOf(Object o)	Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
listIterator()	Returns a list iterator over the elements in this list (in proper sequence).
listIterator(int index)	Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list.
remove(int index)	Removes the element at the specified position in this list.
remove(Object o)	Removes the first occurrence of the specified element from this list, if it is present.
removeAll(Collection<?> c)	Removes from this list all of its elements that are contained in the specified collection.
removeIf(Predicate<? super E> filter)	Removes all of the elements of this collection that satisfy the given predicate.
replaceAll(UnaryOperator<E> operator)	Replaces each element of this list with the result of applying the operator to that element.
retainAll(Collection<?> c)	Retains only the elements in this list that are contained in the specified collection.

METHOD	DESCRIPTION
set(int index, E element)	Replaces the element at the specified position in this list with the specified element.
size()	Returns the number of elements in this list.
sort(Comparator<? super E> c)	Sorts this list according to the order induced by the specified Comparator.
spliterator()	Returns a Spliterator over the elements in this list.
subList(int fromIndex, int toIndex)	Returns a view of the portion of this list between fromIndex, inclusive, and toIndex, exclusive.
toArray()	Returns an array containing all of the elements in this list in proper sequence (from first to the last element).
toArray(T[] a)	Returns an array containing all of the elements in this list in proper sequence (from first to the last element); the runtime type of the returned array is that of the specified array.
toString()	Returns a string representation of this list.

Methods inherited from interface java.util.Collection:

METHOD	DESCRIPTION
parallelStream()	Returns a possibly parallel Stream with this collection as its source.
stream()	Returns a sequential Stream with this collection as its source.

Note: We should use **CopyOnWriteArrayList** when we prefer to use a data structure similar to ArrayList in a concurrent environment.

Must Read:

- Difference between ArrayList and CopyOnWriteArrayList