CopyOnWriteArrayList :

Sometimes we want to add or remove elements from the list if we find some specific element, in that case we should use concurrent collection class – CopyOnWriteArrayList. This is a thread-safe variant of java.util.ArrayList in which all mutative operations (add, set, and so on) are implemented by making a fresh copy of the underlying array.

CopyOnWriteArrayList introduces extra overload to the processing but it's very effective when number of modifications are minimal compared to number of traversal operations.

Notice that it allows the modification of list, but it doesn't change the iterator and we get same elements as it was on original list.

It's immutable snapshot style iterator method uses a reference to the state of the array at the point that the iterator was created. This helps in usecases when traversal operations vastly outnumber list update operations and we do not want to synchronize the traversals and still want thread safety while updating the list.

Copy on Write ArrayList is a thread safe version of the Arraylist.

The important things to learn about Java CopyOnWriteArrayList class are:

- CopyOnWriteArrayList class implement `List` and `RandomAccess` interfaces and thus provide all functionalities available in ArrayList class.
- Using CopyOnWriteArrayList is costly for update operations, because each mutation creates a cloned copy of underlying array and add/update element to it.
- It is thread-safe version of ArrayList. Each thread accessing the list sees its own version of snapshot of backing array created while initializing the iterator for this list.
- Because it gets snapshot of underlying array while creating iterator, it does not throw ConcurrentModificationException.
- Mutation operations on iterators (remove, set, and add) are not supported. These methods throw `UnsupportedOperationException`.
- CopyOnWriteArrayList is a concurrent replacement for a synchronized List and offers better concurrency when iterations outnumber mutations.

- It allows duplicate elements and heterogeneous Objects (use generics to get compile time errors).
- Because it creates a new copy of array everytime iterator is created, performance is slower than ArrayList.

It provides better performance when there are more iterations than mutations, because it is thread safe and many threads can read it concurrently.
It refers to the state of the array at the point when the iterator was created.

The iterator of this CopyOnWriteArrayList is a snapshot style iterator, it is a snapshot of the original collection and never reflects the changes made to the collection after the iterator was created. This iterator is fail-safe.

Iterator methods like remove, set, add etc are not supported with this. It will throw UnSupportedOperationException, as it operates on copy of the original collection, and original collection cannot be modifed via this iterator.
However, if we add / remove / update from the original List, then it works fine, because it is not concerned with the original List, it is just operating on the copy , hence adding anything to the original List object / removing etc, won't affect this iterator, whereas it is the opposite in fail-fast iterators.

In fail fast iterator, adding anything or removing or modifying the original collection with the collection object while iterating would throw an error, because their iterator iterates over the original collection, and checks the mod count before every next() call (modcount gets updated after every update on the collection)
.
Since this iterator is not affected by mutations, it can be accessed by multiple threads.

Internal Working:

Add method is implemented in a thread safe fashion.
It uses a re-entrant lock for thread safety.
On addition of every element, the length of the array increases by 1.

Java CopyOnWriteArrayList Usecases

We can prefer to use CopyOnWriteArrayList over normal ArrayList in following cases:

1. When list is to be used in concurrent environemnt.

2. Iterations outnumber the mutation operations.
3. Iterators must have snapshot version of list at the time when they were created.
4. We don’t want to synchronize the thread access programatically.

Java CopyOnWriteArrayList Performance

Due to added step of creating a new backing array everytime the list is updated, it performs worse than ArrayList.

There is no performance overhead on read operations and both classes perform same.