ArrayList<E> class

ArrayList<E> class is concrete implementation of List<E> interface. It uses array to perform list operations. It also permits duplicate and null elements. ArrayList<E> class is unsynchronized.

ArrayList<E> class’s objects has a maximum capacity.

The implementation is not thread-safe. If you want to use synchronized version of ArrayList<E> then you can use Vector<E> or you can convert given ArrayList<E> object by Collections.synchronizedList(List<T> list);

ArrayList<E> must not be used in concurrent environment or in multithreaded environment. As it is not synchronized the list can be modified structurally (add element, delete element or resize backing array).

It is best that we synchronize list at the time of creation.

We can do that as List<Integer> list=Collections.synchronizedList(new ArrayList<Integer>(..));

Iterator and ListIterator provided by ArrayList are fail-fast. Means that If the list is modified structurally when list is been traversed by Iterator or ListIterator any other than iterator’s own remove() method it throws ConcurrentModificationException. So iterator fails quickly so as to save us from any non-deterministic behavior in future. But the exception is thrown on best effort basics and no guarantee can be made for correctness of list.

Few of the methods of ArrayList class runs in constant time i.e. in O(1) time. They are size, get, set, isEmpty, iterator and listIterator.

I have explained different method implementation of ArrayList<E> class here and here.

I also have written post on how to traverse ArrayList<E> in 5 different ways.

Next I will write down about the different constructors used in ArrayList<E>.

First is no argument constructor which creates an empty list of capacity 10.

Second is int argument constructor that takes initial capacity of ArrayList<E>. If initial capacity is provided in negative then IllegalArgumentException is thrown.

Third constructor take argument as Collection. So elements in specified collection are copied into this ArrayList<E> object. The insertion order is not guaranteed as it depends on underlying implementation of Collection.