List [Interface]: ArrayList

Definition

It's an interface that extends the Collection and contains ordered collection of elements including duplicate values.

Class ArrayList<E>

Is the implementation of List interface where the elements can be dynamically added or removed from the list. Java List provides control over the position where you can insert an element. You can access by their index and search elements in the list. The size of the list is increased dynamically if the elements are added more than the initial size.

Implements all optional list operations, and permits all elements, including null. This class provides methods to manipulate the size of the array that is used internally to store the list. This class is roughly equivalent to Vector, except that is unsynchronized.

The size, isEmpty, get, set, iterator, and listIterator operations run in constant time. The add operation runs in amortized constant time, that is, adding n elements requires O(n) time. All of the other operations run-in linear time. The constant factor is low compared to that for the LinkedList implementation.

The implementation is not synchronized. If multiple threads access and ArrayList instance concurrently, and at least one of the threads modifies the list structurally. It must be synchronized externally. This is accomplished by synchronizing on some object that naturally encapsulates the list. If not such object exist, the list should be "wrapped" using the Collections.synchronizedList method.

```
List  list = Collections.synchronizedList(new ArrayList(…));
```

Constructor summary

ArrayList ()	Constructs an empty list with an initial capacity of ten.
ArrayList (Collection < ? extends E > c)	Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.
ArrayList (int initalCapacity)	Constructs an empty list with the specified initial capacity.

Methods summary

Modifier and Type	Method and Description
boolean	**add**(E e) Appends the specified element to the end of this list.
void	**add**(int index, E element) Inserts the specified element at the specified position in this list.
boolean	**addAll**(Collection < ? extends E > c) Appends all of the elements in the specified collections to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean	**addAll** (int index, Collection < ? extends E > c) Insert all of the elements in the specified collection into this list, starting at the specified position.
void	**clear**() Removes all of the elements from this list
Object	**clone**() Returns a shallow copy of this ArrayList instance
boolean	**contains** (Object o) Returns true if this list contains the specified element
void	**ensureCapacity**(int minCapacity) Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
void	**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.
E	**get** (int index) Returns the element at the specified position in this list.
int	**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.
boolean	**isEmpty** ( ) Returns true if this list contains no elements
Iterator<E>	**iterator** ( ) Returns an iterator over the elements in this list in proper sequence.

When is its use recommended?

- ArrayList provides constant time for search operation, so it's better to use ArrayList if searching is more frequent operation than add and remove operation.
- When we don't know the size of the array because is a dynamic array, so we don't have to specify the size while creating it.
- When we want to access randomly to the Array. ArrayList has 0(1)-time complexity to access elements via the get and set methods.
- When we work with data that duplicate values matters.