Java Collection Framework ArrayList

This class provides implementation of an array based data structure that is used to store elements in linear order. This class implements List interface and an abstract AbstractList class. It creates a dynamic array that grows based on the elements strength.

Java ArrayList class Declaration

```
public class ArrayList<E> extends AbstractList<E> implements List<E>,
RandomAccess, Cloneable, Serializable
```

Copy

1. ArrayList supports dynamic array that can grow as needed.
2. It can contain Duplicate elements and it also maintains the insertion order.
3. Manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.
4. ArrayLists are not synchronized.
5. ArrayList allows random access because it works on the index basis.

ArrayList Constructors

ArrayList class has three constructors that can be used to create Arraylist either from empty or elements of other collection.

```
ArrayList()  // It creates an empty ArrayList

ArrayList( Collection C ) // It creates an ArrayList that is
initialized with elements of the Collection C

ArrayList( int capacity ) // It creates an ArrayList that has the
specified initial capacity
```

Copy

Example: Creating an ArrayList

Lets create an ArrayList to store string elements. Here list is empty because we did not add elements to it.

```
import java.util.*;

class Demo

{

  public static void main(String[] args)
```

```
  {

    // Creating an ArrayList

    ArrayList< String> fruits = new ArrayList< String>();



    // Displaying Arraylist

    System.out.println(fruits);

  }

}
```

Copy

```
[]
```

ArrayList Methods

The below table contains methods of Arraylist. We can use them to manipulate its elements.

Method	Description
void add(int index, E element)	It inserts the specified element at the specified position in a list.
boolean add(E e)	It appends the specified element at the end of a list.
boolean addAll(Collection<? extends E> c)	It appends all of the elements in the specified collection to the end of this list.
boolean addAll(int index, Collection<? extends E> c)	It appends all the elements in the specified collection, starting at the specified position of the list.
void clear()	It removes all of the elements from this list.
void ensureCapacity(int requiredCapacity)	It enhances the capacity of an ArrayList instance.

E get(int index)	It fetches the element from the particular position of the list.
boolean isEmpty()	It returns true if the list is empty, otherwise false.
int lastIndexOf(Object o)	It returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
Object[] toArray()	It returns an array containing all of the elements in this list in the correct order.
<T> T[] toArray(T[] a)	It returns an array containing all of the elements in this list in the correct order.
Object clone()	It returns a shallow copy of an ArrayList.
boolean contains(Object o)	It returns true if the list contains the specified element
int indexOf(Object o)	It returns the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
E remove(int index)	It removes the element present at the specified position in the list.
boolean remove(Object o)	It removes the first occurrence of the specified element.
boolean removeAll(Collection<?> c)	It removes all the elements from the list.

boolean removeIf(Predicate<? super E> filter)	It removes all the elements from the list that satisfies the given predicate.
protected void removeRange(int fromIndex, int toIndex)	It removes all the elements lies within the given range.
void replaceAll(UnaryOperator<E> operator)	It replaces all the elements from the list with the specified element.
void trimToSize()	It trims the capacity of this ArrayList instance to be the list's current size.

Add Elements to ArrayList

To add elements into ArrayList, we are using add() method. It adds elements into the list in the insertion order.

```
import java.util.*;

class Demo

{

  public static void main(String[] args)

  {

    // Creating an ArrayList

    ArrayList< String> fruits = new ArrayList< String>();

    // Adding elements to ArrayList

    fruits.add("Mango");

    fruits.add("Apple");

    fruits.add("Berry");

    // Displaying ArrayList

    System.out.println(fruits);

  }

}
```

Copy

```
[Mango, Apple, Berry]
```

Removing Elements

To remove elements from the list, we are using remove method that remove the specified elements. We can also pass index value to remove the elements of it.

```
import java.util.*;

class Demo

{

  public static void main(String[] args)

  {

    // Creating an ArrayList

    ArrayList< String> fruits = new ArrayList< String>();

    // Adding elements to ArrayList

    fruits.add("Mango");

    fruits.add("Apple");

    fruits.add("Berry");

    fruits.add("Orange");

    // Displaying ArrayList

    System.out.println(fruits);

    // Removing Elements

    fruits.remove("Apple");

    System.out.println("After Deleting Elements: \n"+fruits);

    // Removing Second Element

    fruits.remove(1);

    System.out.println("After Deleting Elements: \n"+fruits);


  }

}
```

Copy

```
[Mango, Apple, Berry, Orange]

After Deleting Elements:

[Mango, Berry, Orange]

After Deleting Elements:

[Mango, Orange]
```

Traversing Elements of ArrayList

Since ArrayList is a collection then we can use loop to iterate its elements. In this example we are traversing elements. See the below example.

```
import java.util.*;

class Demo

{

  public static void main(String[] args)

  {

    // Creating an ArrayList

    ArrayList< String> fruits = new ArrayList< String>();

    // Adding elements to ArrayList

    fruits.add("Mango");

    fruits.add("Apple");

    fruits.add("Berry");

    fruits.add("Orange");

    // Traversing ArrayList

    for(String element : fruits) {

      System.out.println(element);

    }

  }

}
```

Copy

```
Mango

Apple

Berry

Orange
```

Get size of ArrayList

Sometimes we want to know number of elements an ArrayList holds. In that case we use size() then returns size of ArrayList which is equal to number of elements present in the list.

```
import java.util.*;

class Demo

{

  public static void main(String[] args)

  {

    // Creating an ArrayList

    ArrayList< String> fruits = new ArrayList< String>();

    // Adding elements to ArrayList

    fruits.add("Mango");

    fruits.add("Apple");

    fruits.add("Berry");

    fruits.add("Orange");

    // Traversing ArrayList

    for(String element : fruits) {

      System.out.println(element);

    }

    System.out.println("Total Elements: "+fruits.size());

  }

}
```

Copy

```
Mango

Apple

Berry
```

```
Orange

Total Elements: 4
```

Sorting ArrayList Elements

To sort elements of an ArrayList, Java provides a class Collections that includes a static method sort(). In this example, we are using sort method to sort the elements.

```
import java.util.*;

class Demo

{

  public static void main(String[] args)

  {

  // Creating an ArrayList

  ArrayList< String> fruits = new ArrayList< String>();

  // Adding elements to ArrayList

  fruits.add("Mango");

  fruits.add("Apple");

  fruits.add("Berry");

  fruits.add("Orange");

  // Sorting elements

  Collections.sort(fruits);

  // Traversing ArrayList

  for(String element : fruits) {

      System.out.println(element);

    }

  }

}
```

Copy

```
OUTPUT-

Apple

Berry

Mango

Orange
```