Java Collections Part 3(**Collection Interface**)

Let us start understanding Java Collections framework by Collection<E> Interface.

As we discussed in [this blog](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-2-interfaces.html) Collection<E> is root interface in collections hierarchy. Collection is container which allows group of objects to reside inside it. Collection can be seen in different ways. Collections can be Ordered (sequence or order of insertion in maintained) or Unordered. Some collections might allow you the duplicate elements while others might now allow.

Now I will explain the Collection<E> interface.

This is the signature of Collection<E> interface. It extends another interface called Iterable<E>. We will dive deeper in Iterable<E> interface later on. Right now we will focus on Collection<E>.

**public** **interface** Collection<E> **extends** Iterable<E>

<E> defines that any element can be contained in Collection.

**Understanding methods declared in Collection<E> interface**

**int** size(); - this method returns total number of elements respective collection holds.

**boolean** isEmpty(); - this method returns true if the collection is empty.

**boolean** contains(Object o); - this method return true if the Object o resides in collection.

Iterator<E> iterator(); - Iterator<E> is used to iterate over collection elements. But it does not give any guarantee that the elements in Collection will be iterated in same order that they were inserted. This method returns the iterator.

Object[] toArray(); - This method is used to return the Object[] of Collection. Object[] because any object in Collection will by default extend Object class. **Actually, all classes in Java extends Object class, except Object class. Object class does not extend any class.**

<T> T[] toArray(T[] a); - This method is used to return the T[] i.e. type array specified in the argument. This method is different from Object[] toArray() because this method return the actually type of array which was used to declare the collection.

**boolean** add(E e); - This method adds the element e into the collection. Upon successful insertion of element in collection it will return true.

**boolean** remove(Object o); - This method removes the Object o which is passed as parameter to this method. It returns true if the underlying collection is modified, else returns false.

**boolean** containsAll(Collection<?> c); - This method is used to check whether this collection contains all elements in another collection. Returns true if it contains all.

**boolean** addAll(Collection<? **extends** E> c); - This method is used for bulk operation. It takes another collection of type <? extends E> i.e any type that extends E. we use <? extends E> if we need to get elements from Collection. So here we use this bounded wildcard and get all elements from collection c specified in parameter and insert into another collection. It returns true if collection is changed because of call to this method.

**boolean** removeAll(Collection<?> c); - This method is used to remove all elements from collection that contains elements in Collection c(passed as parameter in method). After this operation the collection will have no elements that are in collection c. Returns true if the collection was changed due to the call to this method.

**boolean** retainAll(Collection<?> c); - this method is used to retain the elements in this collection that are contained in specified collection. Means it removes elements from this collection that are not contained in specified collection. Returns true of this collection is changed as result of this method call.

**void** clear(); - this method is used to remove all elements from collection. Once this method is executed resulting collection will be empty.

**boolean** equals(Object o); - Object o is used to compare for the elements in Collection. If match happens then it returns true else false.

**int** hashCode(); - returns the hashCode for the collection.