Interfaces of Java Collection Framework

The Collections framework has a lot of Interfaces, setting the fundamental nature of various collection classes. Lets study the most important Interfaces in the Collections framework.

The Collection Interface

It is at the top of collection heirarchy and must be implemented by any class that defines a collection. Its general declaration is,

```
interface Collection <E>
```

Copy

Collection Interface Methods

1. Following are some of the commonly used methods in this interface.

Methods	**Description**
boolean add(E obj)	Used to add objects to a collection. Returns true if obj was added to the collection. Returns false if obj is already a member of the collection, or if the collection does not allow duplicates.
boolean addAll(Collection C)	Add all elements of collection C to the invoking collection. Returns true if the element were added. Otherwise, returns false.
boolean remove(Object obj)	To remove an object from collection. Returns true if the element was removed. Otherwise, returns false.
boolean removeAll(Collection C)	Removes all element of collection C from the invoking collection. Returns true if the collection's elements were removed. Otherwise, returns false.
boolean contains(Object obj)	To determine whether an object is present in collection or not. Returns true if obj is an element of the invoking collection. Otherwise, returns false.
boolean isEmpty()	Returns true if collection is empty, else returns false.
int size()	Returns number of elements present in collection.
void clear()	Removes total number of elements from the collection.

Object[] toArray()	Returns an array which consists of the invoking collection elements.
boolean retainAll(Collection c)	Deletes all the elements of invoking collection except the specified collection.
Iterator iterator( )	Returns an iterator for the invoking collection.
boolean equals(Object obj)	Returns true if the invoking collection and obj are equal. Otherwise, returns false.
Object[] toArray(Object array[])	Returns an array containing only those collection elements whose type matches of the specified array.

The List Interface

It extends the **Collection** Interface, and defines storage as sequence of elements. Following is its general declaration,

```
interface List <E>
```

Copy

1. Allows random access and insertion, based on position.
2. It allows Duplicate elements.

List Interface Methods

3. Apart from methods of Collection Interface, it adds following methods of its own.

Methods	Description
Object get(int index)	Returns object stored at the specified index
Object set(int index, E obj)	Stores object at the specified index in the calling collection
int indexOf(Object obj)	Returns index of first occurrence of obj in the collection
int lastIndexOf(Object obj)	Returns index of last occurrence of obj in the collection
List subList(int start, int end)	Returns a list containing elements between start and end index in the collection

The Set Interface

This interface defines a Set. It extends **Collection** interface and doesn't allow insertion of duplicate elements. It's general declaration is,

```
interface Set <E>
```

Copy

1. It doesn't define any method of its own. It has two sub interfaces, **SortedSet** and **NavigableSet**.
2. **SortedSet** interface extends **Set** interface and arranges added elements in an ascending order.
3. **NavigabeSet** interface extends **SortedSet** interface, and allows retrieval of elements based on the closest match to a given value or values.

The Queue Interface

It extends **collection** interface and defines behaviour of queue, that is first-in, first-out. It's general declaration is,

```
interface Queue <E>
```

Copy

Queue Interface Methods

There are couple of new and interesting methods added by this interface. Some of them are mentioned in below table.

Methods	Description
Object poll()	removes element at the head of the queue and returns **null** if queue is empty
Object remove()	removes element at the head of the queue and throws **NoSuchElementException** if queue is empty
Object peek()	returns the element at the head of the queue without removing it. Returns **null** if queue is empty
Object element()	same as peek(), but throws **NoSuchElementException** if queue is empty
boolean offer(E obj)	Adds object to queue.

The Dequeue Interface

It extends **Queue** interface and implements behaviour of a double-ended queue. Its general declaration is,

```
interface Dequeue <E>
```

Copy

1. Since it implements Queue interface, it has the same methods as mentioned there.
2. Double ended queues can function as simple queues as well as like standard Stacks.