Collections

java.util.*

- ArrayList
- HashSet
- LinkedList
- LinkedHashSet
- TreeSet
- Vector

java.util.Iterator & java.lang.Iterable

Iterator enables you to cycle through a collection, obtaining or removing elements.

```
public interface Iterator<E> {

    boolean hasNext();

    E next();

    void remove();

...
}
```

Implementing this interface allows an object to be the target of the "for-each loop" statement.

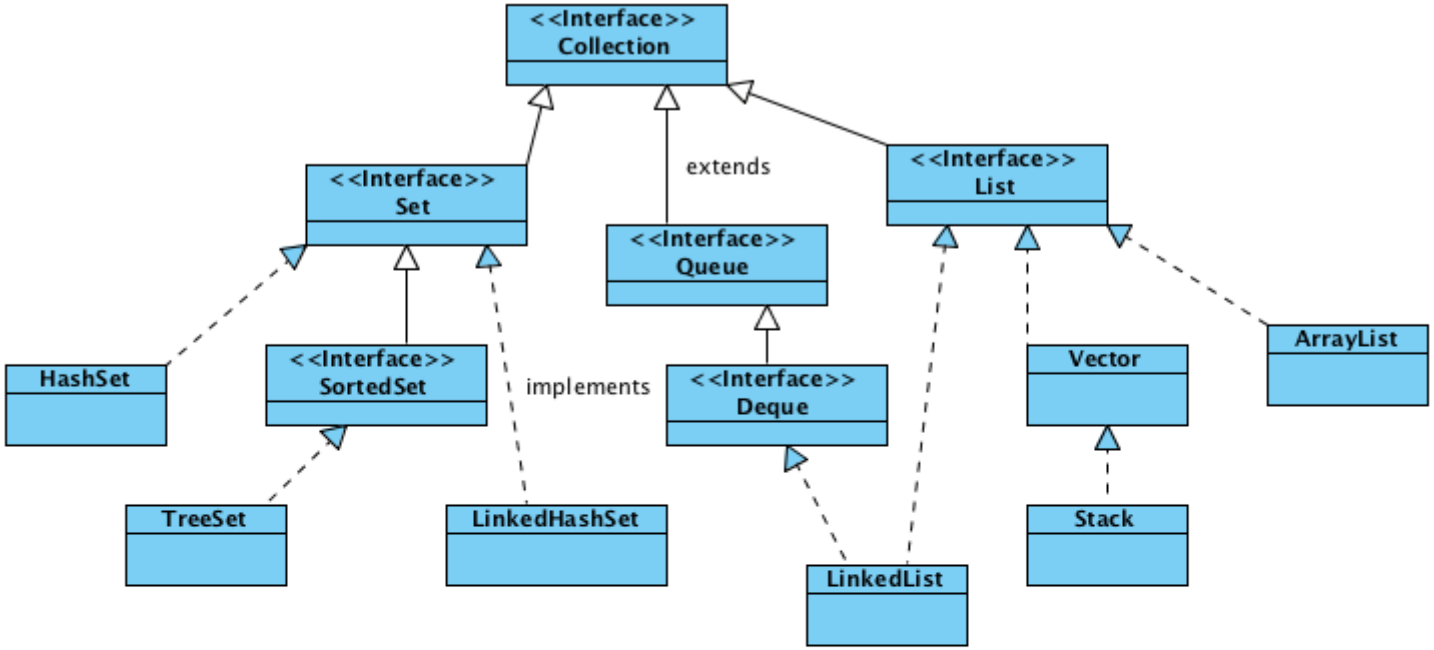
```
public interface Iterable<T> {

    Iterator<T> iterator();

...

}
```

Hierarchy


<<Interface>>
Collection
<<Interface>>
Set
extends
<<Interface>>
List
<<Interface>>
Queue
HashSet
<<Interface>>
SortedSet
implements
<<Interface>>
Deque
Vector
ArrayList
TreeSet
LinkedHashSet
LinkedList
Stack

Interface Collection before 1.8

```
public interface Collection<E> extends Iterable<E> {

    int size();

    boolean isEmpty();

    boolean contains(Object o);

    Iterator<E> iterator();

    <T> T[] toArray(T[] a);

    boolean add(E e);

    boolean remove(Object o);

    boolean containsAll(Collection<?> c);

    boolean addAll(Collection<? extends E> c);

    boolean removeAll(Collection<?> c);

    void clear();

...
}
```

Interface Collection after 1.8

```
public interface Collection<E> extends Iterable<E> {

...

   default boolean removeIf(Predicate<? super E> filter) {
       Objects.requireNonNull(filter);
       boolean removed = false;
       final Iterator<E> each = iterator();
       while (each.hasNext()) {
           if (filter.test(each.next())) {
               each.remove();
               removed = true;
           }
       }
       return removed;
   }

   @Override
   default Spliterator<E> spliterator() {
       return Spliterators.spliterator(this, 0);
   }

   default Stream<E> stream() {
       return StreamSupport.stream(spliterator(), false);
   }

   default Stream<E> parallelStream() {
       return StreamSupport.stream(spliterator(), true);
   }
}
```

Spliterator

Spliterators, like Iterators, are for traversing the elements of a source.

The Spliterator API was designed to support efficient

- parallel traversal in addition to sequential traversal,
- decomposition as well as single-element iteration.

Collections comparison

	add	remove	get	contains	ordered	sorted
ArrayList	O(1)	O(n)	O(1)	O(n)	yes	no
HashSet	O(1)	O(1)	-	O(1)	no	no
LinkedList	O(1)	O(1)	O(n)	O(n)	yes	no
LinkedHashSet	O(1)	O(1)	-	O(1)	yes	no
TreeSet	O(log n)	O(log n)	-	O(log n)	yes*	yes
Vector**	O(1)	O(n)	O(1)	O(n)	yes	no

* TreeSet ordered because it is sorted

** Vector is the same as ArrayList but synchronized