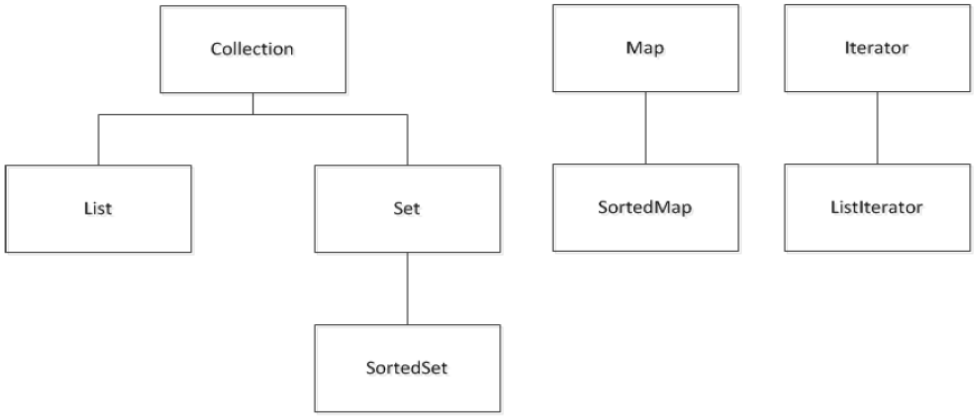
The Java Collections Framework

interfaces:




```
public interface Collection<E> extends Iterable<E> {
        // Basic operations
        int size();
        boolean isEmpty();
        boolean contains(Object element);
        boolean add(E element); // optional
        boolean remove(Object element); // optional
        Iterator<E> iterator();
        // Bulk operations
        boolean containsAll(Collection<E> c);
        boolean addAll(Collection<? extends E> c); // optional
        boolean removeAll(Collection<? extends E> c); // optional
        boolean retainAll(Collection<? extends E> c); // optional
        void clear(); // optional
        // Array operations
        Object[] toArray();
        <E> E[] toArray(E[] a);
}
```

```
public interface List<E> extends Collection<E> {
        // Positional access
        E get(int index);
        E set(int index, E element); // optional
        boolean add(E element); // optional
        void add(int index, E element); // optional
        E remove(int index); // optional
        boolean addAll(int index, Collection<? extends E> c); // optional
        // Search
        int indexOf(Object o);
        int lastIndexOf(Object o);
        // Iteration
        ListIterator<E> listIterator();
        ListIterator<E> listIterator(int index);
        // Range-view
        List<E> subList(int from, int to);
}
```

```java
public interface Set<E> extends Collection<E> {
        // Basic operations
        int size();
        boolean isEmpty();
        boolean contains(Object element);
        boolean add(E element); // optional
        boolean remove(Object element); // optional
        Iterator<E> iterator();
        // Bulk operations
        boolean containsAll(Collection<?> c);
        boolean addAll(Collection<? extends E> c); // optional
        boolean removeAll(Collection<?> c); // optional
        boolean retainAll(Collection<?> c); // optional
        void clear(); // optional
        // Array Operations
        Object[] toArray();
        <T> T[] toArray(T[] a);
}
```

```java
public interface SortedSet<E> extends Set<E> {
    Comparator<? super E> comparator();
    E first();
    E last();
}
```

```java
public interface Map<K, V> {
        // Basic operations
        V put(K key, V value);
        V get(Object key);
        V remove(Object key);
        boolean containsKey(Object key);
        boolean containsValue(Object value);
        int size();
        boolean isEmpty();
        // Bulk operations
        void putAll(Map<? extends K, ? extends V> m);
        void clear();
        // Collection Views
        public Set<K> keySet();
        public Collection<V> values();
        public Set<Map.Entry<K, V>> entrySet();
        // Interface for entrySet elements
        public interface Entry<K, V> {
                K getKey();
                V getValue();
                V setValue(V value);
        }
}
```

```java
public interface SortedMap<K, V> extends Map<K, V>{
    Comparator<? super K> comparator();
    SortedMap<K, V> subMap(K fromKey, K toKey);
    SortedMap<K, V> headMap(K toKey);
    SortedMap<K, V> tailMap(K fromKey);
    K firstKey();
    K lastKey();
}
```

```java
public interface Iterator<E> {
    boolean hasNext();
    E next();
    void remove();
}
```

```java
public interface ListIterator<E> extends Iterator<E> {
    // Query Operations
    boolean hasNext();
    E next();
    boolean hasPrevious();
    E previous();
    int nextIndex();
    int previousIndex();
    // Modification Operations
    void remove();
    void set(E e);
    void add(E e);
}
```



```java
public interface Comparable<T> {
    public int compareTo(T o);
}
```

```java
public interface Comparator<T> {
    int compare(T obj1, T obj2);
}
```

Interface:	**Implementations:**
`List`	`ArrayList, LinkedList`
`Set`	`HashSet, TreeSet, LinkedHashSet`
`Map`	`HashMap, TreeMap, LinkedTreeMap`

Method Summary of java.util.Collections (incomplete)	
static <T> boolean	**addAll**(Collection<? super T> c, T... elements) Adds all of the specified elements to the specified collection.
static <T> int	**binarySearch**(List<? extends Comparable<? super T>> list, T key) Searches the specified list for the specified object using the binary search algorithm.
static <T> int	**binarySearch**(List<? extends T> list, T key, Comparator<? super T> c) Searches the specified list for the specified object using the binary search algorithm.
static <T> void	**copy**(List<? super T> dest, List<? extends T> src) Copies all of the elements from one list into another.
static boolean	**disjoint**(Collection<?> c1, Collection<?> c2) Returns `true` if the two specified collections have no elements in common.
static <T> List<T>	**emptyList**() Returns the empty list (immutable).
static <K,V> Map<K,V>	**emptyMap**() Returns the empty map (immutable).
static <T> Set<T>	**emptySet**() Returns the empty set (immutable).
static <T> void	**fill**(List<? super T> list, T obj) Replaces all of the elements of the specified list with the specified element.
static int	**frequency**(Collection<?> c, Object o) Returns the number of elements in the specified collection equal to the specified object.
static int	**indexOfSubList**(List<?> source, List<?> target) Returns the starting position of the first occurrence of the specified target list within the specified source list, or -1 if there is no such occurrence.
static int	**lastIndexOfSubList**(List<?> source, List<?> target) Returns the starting position of the last occurrence of the specified target list within the specified source list, or -1 if there is no such occurrence.
static <T extends Object & Comparable<? super T>> T	**max**(Collection<? extends T> coll) Returns the maximum element of the given collection, according to the *natural ordering* of its elements.
static <T> T	**max**(Collection<? extends T> coll, Comparator<? super T> comp) Returns the maximum element of the given collection, according to the order induced by the specified comparator.
static <T extends Object & Comparable<? super T>> T	**min**(Collection<? extends T> coll) Returns the minimum element of the given collection, according to the *natural ordering* of its elements.
static <T> T	**min**(Collection<? extends T> coll, Comparator<? super T> comp) Returns the minimum element of the given collection, according to the order induced by the specified comparator.
static <T> List<T>	**nCopies**(int n, T o) Returns an immutable list consisting of `n` copies of the specified object.
static <E> Set<E>	**newSetFromMap**(Map<E,Boolean> map) Returns a set backed by the specified map.

static <T> boolean	**replaceAll**(List<T> list, T oldVal, T newVal) Replaces all occurrences of one specified value in a list with another.
static void	**reverse**(List<?> list) Reverses the order of the elements in the specified list.
static <T> Comparator<T>	**reverseOrder**() Returns a comparator that imposes the reverse of the *natural ordering* on a collection of objects that implement the Comparable interface.
static <T> Comparator<T>	**reverseOrder**(Comparator<T> cmp) Returns a comparator that imposes the reverse ordering of the specified comparator.
static void	**rotate**(List<?> list, int distance) Rotates the elements in the specified list by the specified distance.
static void	**shuffle**(List<?> list) Randomly permutes the specified list using a default source of randomness.
static void	**shuffle**(List<?> list, Random rnd) Randomly permute the specified list using the specified source of randomness.
static <T> Set<T>	**singleton**(T o) Returns an immutable set containing only the specified object.
static <T> List<T>	**singletonList**(T o) Returns an immutable list containing only the specified object.
static <K,V> Map<K,V>	**singletonMap**(K key, V value) Returns an immutable map, mapping only the specified key to the specified value.
static <T extends Comparable<? super T>> void	**sort**(List<T> list) Sorts the specified list into ascending order, according to the *natural ordering* of its elements.
static <T> void	**sort**(List<T> list, Comparator<? super T> c) Sorts the specified list according to the order induced by the specified comparator.
static void	**swap**(List<?> list, int i, int j) Swaps the elements at the specified positions in the specified list.
static <T> Collection<T>	**synchronizedCollection**(Collection<T> c) Returns a synchronized (thread-safe) collection backed by the specified collection.
static <T> List<T>	**synchronizedList**(List<T> list) Returns a synchronized (thread-safe) list backed by the specified list.
static <K,V> Map<K,V>	**synchronizedMap**(Map<K,V> m) Returns a synchronized (thread-safe) map backed by the specified map.
static <T> Set<T>	**synchronizedSet**(Set<T> s) Returns a synchronized (thread-safe) set backed by the specified set.
static <K,V> SortedMap<K,V>	**synchronizedSortedMap**(SortedMap<K,V> m) Returns a synchronized (thread-safe) sorted map backed by the specified sorted map.
static <T> SortedSet<T>	**synchronizedSortedSet**(SortedSet<T> s) Returns a synchronized (thread-safe) sorted set backed by the specified sorted set.

static <T> Collection<T>	**unmodifiableCollection**(Collection<? extends T> c) Returns an unmodifiable view of the specified collection.
static <T> List<T>	**unmodifiableList**(List<? extends T> list) Returns an unmodifiable view of the specified list.
static <K,V> Map<K,V>	**unmodifiableMap**(Map<? extends K,? extends V> m) Returns an unmodifiable view of the specified map.
static <T> Set<T>	**unmodifiableSet**(Set<? extends T> s) Returns an unmodifiable view of the specified set.
static <K,V> SortedMap<K,V>	**unmodifiableSortedMap**(SortedMap<K,? extends V> m) Returns an unmodifiable view of the specified sorted map.
static <T> SortedSet<T>	**unmodifiableSortedSet**(SortedSet<T> s) Returns an unmodifiable view of the specified sorted set.

Method Summary of class java.util.Arrays

static <T> List<T>	**asList**(T... a) Returns a fixed-size list backed by the specified array.
static int	**binarySearch**(TYPE[] a, TYPE key) Searches the specified array of bytes for the specified value using the binary search algorithm.
static <T> int	**binarySearch**(T[] a, int fromIndex, int toIndex, T key, Comparator<? super T> c) Searches a range of the specified array for the specified object using the binary search algorithm.
static <T> int	**binarySearch**(T[] a, T key, Comparator<? super T> c) Searches the specified array for the specified object using the binary search algorithm.
static TYPE[]	**copyOf**(TYPE[] original, int newLength) Copies the specified array, truncating or padding with zeros (if necessary) so the copy has the specified length.
static TYPE[]	**copyOfRange**(TYPE[] original, int from, int to) Copies the specified range of the specified array into a new array.
static boolean	**deepEquals**(Object[] a1, Object[] a2) Returns `true` if the two specified arrays are *deeply equal* to one another.
static int	**deepHashCode**(Object[] a) Returns a hash code based on the "deep contents" of the specified array.
static String	**deepToString**(Object[] a) Returns a string representation of the "deep contents" of the specified array.
static boolean	**equals**(TYPE[] a, TYPE[] a2) Returns `true` if the two specified arrays of booleans are *equal* to one another.
static void	**fill**(TYPE[] a, TYPE val) Assigns the specified TYPE value to each element of the specified array of booleans.
static void	**fill**(TYPE[] a, int fromIndex, int toIndex, TYPE val) Assigns the specified TYPE value to each element of the specified range of the specified array of TYPEs.
static int	**hashCode**(TYPE[] a) Returns a hash code based on the contents of the specified array.
static void	**sort**(TYPE[] a) Sorts the specified array of TYPEs into ascending numerical order.
static void	**sort**(TYPE[] a, int fromIndex, int toIndex) Sorts the specified range of the specified array of TYPEs into ascending numerical order.
static String	**toString**(TYPE[] a) Returns a string representation of the contents of the specified array.