Set Implementations

1. [HashSet](https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html)
2. [TreeSet](https://docs.oracle.com/javase/8/docs/api/java/util/TreeSet.html)
3. [LinkedHashSet](https://docs.oracle.com/javase/8/docs/api/java/util/LinkedHashSet.html)
4. [EnumSet](https://docs.oracle.com/javase/8/docs/api/java/util/EnumSet.html)
5. [CopyOnWriteArraySet](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CopyOnWriteArraySet.html)

List Implementations

1. [ArrayList](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html)
2. [LinkedList](https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html)
3. [CopyOnWriteArrayList](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CopyOnWriteArrayList.html)

**Map Implementations**

1. [HashMap](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html)
2. [TreeMap](https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html)
3. [LinkedHashMap](https://docs.oracle.com/javase/8/docs/api/java/util/LinkedHashMap.html)
4. [EnumMap](https://docs.oracle.com/javase/8/docs/api/java/util/EnumMap.html)
5. [WeakHashMap](https://docs.oracle.com/javase/8/docs/api/java/util/WeakHashMap.html)
6. [IdentityHashMap](https://docs.oracle.com/javase/8/docs/api/java/util/IdentityHashMap.html)
7. [ConcurrentMap](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentMap.html)

# Queue Implementations

1. [LinkedBlockingQueue](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/LinkedBlockingQueue.html) — an optionally bounded FIFO blocking queue backed by linked nodes
2. [ArrayBlockingQueue](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ArrayBlockingQueue.html) — a bounded FIFO blocking queue backed by an array
3. [PriorityBlockingQueue](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/PriorityBlockingQueue.html) — an unbounded blocking priority queue backed by a heap
4. [DelayQueue](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/DelayQueue.html) — a time-based scheduling queue backed by a heap
5. [SynchronousQueue](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/SynchronousQueue.html) — a simple rendezvous mechanism that uses the BlockingQueue interface
6. [PriorityQueue](https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html)
7. [LinkedTransferQueue](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/LinkedTransferQueue.html) — an unbounded TransferQueue based on linked nodes

# Deque Implementations

1. LinkedList
2. ArrayDeque
3. [LinkedBlockingDeque](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/LinkedBlockingDeque.html)

# Wrapper Implementations

The synchronization wrappers add automatic synchronization (thread-safety) to an arbitrary collection. Each of the six core collection interfaces — [Collection](https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html), [Set](https://docs.oracle.com/javase/8/docs/api/java/util/Set.html), [List](https://docs.oracle.com/javase/8/docs/api/java/util/List.html), [Map](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html), [SortedSet](https://docs.oracle.com/javase/8/docs/api/java/util/SortedSet.html), and [SortedMap](https://docs.oracle.com/javase/8/docs/api/java/util/SortedMap.html) — has one static factory method.

public static <T> Collection<T> synchronizedCollection(Collection<T> c);

public static <T> Set<T> synchronizedSet(Set<T> s);

public static <T> List<T> synchronizedList(List<T> list);

public static <K,V> Map<K,V> synchronizedMap(Map<K,V> m);

public static <T> SortedSet<T> synchronizedSortedSet(SortedSet<T> s);

public static <K,V> SortedMap<K,V> synchronizedSortedMap(SortedMap<K,V> m);

## Unmodifiable Wrappers

Unlike synchronization wrappers, which add functionality to the wrapped collection, the unmodifiable wrappers take functionality away. In particular, they take away the ability to modify the collection by intercepting all the operations that would modify the collection and throwing an UnsupportedOperationException. Unmodifiable wrappers have two main uses, as follows:

* To make a collection immutable once it has been built. In this case, it's good practice not to maintain a reference to the backing collection. This absolutely guarantees immutability.
* To allow certain clients read-only access to your data structures. You keep a reference to the backing collection but hand out a reference to the wrapper. In this way, clients can look but not modify, while you maintain full access.

Like synchronization wrappers, each of the six core Collection interfaces has one static factory method.

public static <T> Collection<T> unmodifiableCollection(Collection<? extends T> c);

public static <T> Set<T> unmodifiableSet(Set<? extends T> s);

public static <T> List<T> unmodifiableList(List<? extends T> list);

public static <K,V> Map<K, V> unmodifiableMap(Map<? extends K, ? extends V> m);

public static <T> SortedSet<T> unmodifiableSortedSet(SortedSet<? extends T> s);

public static <K,V> SortedMap<K, V> unmodifiableSortedMap(SortedMap<K, ? extends V> m);

## Checked Interface Wrappers

The Collections.checked interface wrappers are provided for use with generic collections. These implementations return a dynamically type-safe view of the specified collection, which throws a ClassCastException if a client attempts to add an element of the wrong type. The generics mechanism in the language provides compile-time (static) type-checking, but it is possible to defeat this mechanism. Dynamically type-safe views eliminate this possibility entirely.

<https://docs.oracle.com/javase/tutorial/collections/implementations/index.html>