

	Ordering	Random Access	Key-Value Pairs	Allows Duplicates	Allows Null Values	Thread Safe	Blocking Operations	Upper Bounds	Usage Scenarios
Most Commonly Known Collections									
<a href="#">ArrayList</a>	YES	YES	NO	YES	YES	NO	NO	NO	<ul style="list-style-type: none"> <li>* Default choice of List implementation</li> <li>* To store a bunch of things</li> <li>* Repetitions matters</li> <li>* Insertion order matters</li> <li>* Best implementation in case of huge lists which are read intensive (elements are accessed more frequently than inserted deleted)</li> </ul>
<a href="#">HashMap</a>	NO	YES	YES	NO	YES	NO	NO	NO	<ul style="list-style-type: none"> <li>* Default choice of Map implementation</li> <li>* Majorly used for simple in-memory caching purpose.</li> </ul>
<a href="#">Vector</a>	YES	YES	NO	YES	YES	YES	NO	NO	<ul style="list-style-type: none"> <li>* Historical implementation of List</li> <li>* A good choice for thread-safe implementation</li> </ul>
<a href="#">Hashtable</a>	NO	YES	YES	NO	NO	YES	NO	NO	<ul style="list-style-type: none"> <li>* Similar to HashMap</li> <li>* Do not allow null values or keys</li> <li>* Entire map is locked for thread safety</li> </ul>
Most Talked About Collections									
<a href="#">HashSet</a>	NO	YES	NO	NO	YES	NO	NO	NO	<ul style="list-style-type: none"> <li>* To store bunch of things</li> <li>* A very nice alternative for ArrayList if</li> <li>** Do not want repetitions</li> <li>** Ordering does not matter</li> </ul>
<a href="#">TreeSet</a>	YES	YES	NO	NO	NO	NO	NO	NO	<ul style="list-style-type: none"> <li>* To store bunch of things in sorted order</li> <li>* A very nice alternative for ArrayList if</li> <li>** Do not want repetitions</li> <li>** Sorted order</li> </ul>
<a href="#">LinkedList</a>	YES	NO	NO	YES	YES	NO	NO	NO	<ul style="list-style-type: none"> <li>* Sequential Access</li> <li>* Faster adding and deleting of elements</li> <li>* Slightly more memory than ArrayList</li> <li>* Add/Remove elements from both ends of the queue</li> <li>* Best alternative in case of huge lists which are more write intensive (elements added / deleted are more frequent than reading elements)</li> </ul>
<a href="#">ArrayDeque</a>	YES	YES	NO	YES	NO	NO	NO	NO	<ul style="list-style-type: none"> <li>* Random Access</li> <li>* Faster searching and retrieval of elements</li> <li>* Add/Remove elements from both ends of the queue</li> <li>* Best alternative in case of huge lists which are more read intensive</li> </ul>
<a href="#">Stack</a>	YES	NO	NO	YES	YES	YES	NO	NO	<ul style="list-style-type: none"> <li>* Similar to a Vector</li> <li>* Last-In-First-Out implementation</li> </ul>
<a href="#">TreeMap</a>	YES	YES	YES	NO	NO	NO	NO	NO	<ul style="list-style-type: none"> <li>* A very nice alternative for HashMap if sorted keys are important</li> </ul>
Special Purpose Collections									
<a href="#">WeakHashMap</a>	NO	YES	YES	NO	YES	NO	NO	NO	<ul style="list-style-type: none"> <li>* The keys that are not referenced will automatically become eligible for garbage collection</li> <li>* Usually used for advanced caching techniques to store huge data and want to conserve memory</li> </ul>
<a href="#">Arrays</a>	YES	YES	NO	YES	YES	NO	NO	YES	<ul style="list-style-type: none"> <li>* A Utility class provided to manipulate arrays</li> <li>** Searching</li> <li>** Sorting</li> <li>** Converting to other Collection types such as a List</li> </ul>
<a href="#">Properties</a>	NO	YES	YES	NO	NO	YES	NO	NO	<ul style="list-style-type: none"> <li>* Properties are exactly same as the Hashtable</li> <li>* Keys and Values are String</li> <li>* Can be loaded from a input stream</li> <li>* Usually used to store application properties and configurations</li> </ul>
Thread Safe Collections									
<a href="#">CopyOnWriteArrayList</a>	YES	YES	NO	YES	YES	YES	NO	NO	<ul style="list-style-type: none"> <li>* A thread safe variant of ArrayList</li> <li>* Best use for</li> <li>** Small lists which are read intensive</li> <li>** requires thread-safety</li> </ul>
<a href="#">ConcurrentHashMap</a>	NO	YES	YES	NO	NO	YES	NO	NO	<ul style="list-style-type: none"> <li>* A thread safe variant of Hashtable</li> <li>* Best use for</li> <li>** requires thread-safety</li> <li>** Better performance at high load due to a better locking mechanism</li> </ul>
<a href="#">ConcurrentSkipListMap</a>	YES	YES	YES	NO	NO	YES	NO	NO	<ul style="list-style-type: none"> <li>* A thread safe variant of TreeMap</li> <li>* Best use for</li> <li>** requires thread-safety</li> </ul>
<a href="#">ConcurrentSkipListSet</a>	YES	NO	NO	NO	NO	YES	NO	NO	<ul style="list-style-type: none"> <li>* A thread safe variant of TreeSet</li> <li>* Best use for</li> <li>** Do not want repetitions</li> <li>** Sorted order</li> <li>** Requires thread-safety</li> </ul>
<a href="#">CopyOnWriteArraySet</a>	YES	YES	NO	NO	YES	YES	NO	NO	<ul style="list-style-type: none"> <li>* A thread-safe implementation of a Set</li> <li>* Best use for</li> <li>** Small lists which are read intensive</li> <li>** requires thread-safety</li> <li>** Do not want repetitions</li> </ul>
<a href="#">ConcurrentLinkedQueue</a>	YES	NO	NO	YES	NO	YES	NO	NO	<ul style="list-style-type: none"> <li>* A thread-safe variant of PriorityQueue</li> <li>* Best use for</li> <li>** Small lists</li> <li>** No random access</li> <li>** requires thread-safety</li> </ul>
<a href="#">ConcurrentLinkedDeque</a>	YES	NO	NO	YES	NO	YES	NO	NO	<ul style="list-style-type: none"> <li>** A thread-safe variant of LinkedList</li> <li>* Best use for</li> <li>** Small lists</li> <li>** No random access</li> <li>** Insertions, retrieval on both sides of the queue</li> <li>** requires thread-safety"</li> </ul>
Blocking Collections									
<a href="#">ArrayBlockingQueue</a>	YES	NO	NO	YES	NO	YES	YES	YES	<ul style="list-style-type: none"> <li>* Best use for Producer - Consumer type of scenarios with</li> <li>** Lower capacity bound</li> <li>** Predictable capacity</li> <li>* Has a bounded buffer. Space would be allocated during object creation</li> </ul>
<a href="#">LinkedBlockingQueue</a>	YES	NO	NO	YES	NO	YES	YES	YES	<ul style="list-style-type: none"> <li>* Best use for Producer - Consumer type of scenarios with</li> <li>** Large capacity bound</li> <li>** Unpredictable capacity</li> <li>* Upper bound is optional</li> </ul>
<a href="#">LinkedTransferQueue</a>	YES	NO	NO	YES	NO	YES	YES	YES	<ul style="list-style-type: none"> <li>* Can be used in situations where the producers should wait for consumer to receive elements. e.g. Message Passing</li> </ul>
<a href="#">PriorityBlockingQueue</a>	YES	NO	NO	YES	NO	YES	YES	NO	<ul style="list-style-type: none"> <li>** Best use for Producer - Consumer type of scenarios with</li> <li>** Large capacity bound</li> <li>** Unpredictable capacity</li> <li>** Consumer needs elements in sorted order</li> </ul>
<a href="#">LinkedBlockingDeque</a>	YES	NO	NO	YES	NO	YES	YES	YES	<ul style="list-style-type: none"> <li>* A Deque implementation of LinkedBlockingQueue</li> <li>** Can add elements at both head and tail</li> </ul>
<a href="#">SynchronousQueue</a>	YES	NO	NO	YES	NO	YES	YES	NO	<ul style="list-style-type: none"> <li>* Both producer and consumer threads will have to wait for a handoff to occur</li> <li>* If there is no consumer waiting. The element is not added to the collection.</li> </ul>
<a href="#">DelayQueue</a>	YES	NO	NO	YES	NO	YES	YES	NO	<ul style="list-style-type: none"> <li>* Similar to a normal LinkedBlockingQueue</li> <li>* Elements are implementations of Delayed interface</li> <li>* Consumer will be able to get the element only when it's delay has expired</li> </ul>
						Source:	<a href="http://www.janeve.me/articles/which-java-collection-to-use">http://www.janeve.me/articles/which-java-collection-to-use</a>		