Concurrent Collections

LinkedList and ArrayList, as well as TreeSet and HashSet, are also **NOT thread-safe**.

Each of these can be used with a synchronized wrapper, which you can get from the Collections helper class.

The synchronized wrappers provide a thread-safe option for you, with less impact on the design, if you need to make existing code work concurrently.

If you're starting with new code though, I'd recommend using concurrent collections.

Concurrent Collections for Arrays and Lists

For lists, there are two concurrent collection choices, depending on the type of work which needs to be done in parallel.

- Use `ConcurrentLinkedQueue` when you'll have frequent insertions and removals, such as producer-consumer scenarios, or task scheduling.
- Use `CopyOnWriteArrayList` when you have a read-heavy workload with infrequent modifications. This type of list is useful for scenarios like configuration management, or read-only views of data.

For an array, you can use one of the concurrent list options above

- Or Use an `ArrayBlockingQueue`. This is a fixed-size queue, that blocks under two circumstances. The first is if you try to poll or remove an element from an empty queue. The second is if you try to offer, or add an element to a full queue. This is designed as a First In First Out or FIFO queue.

Adding an Element to the ArrayBlockingQueue

On this slide, you can see four methods you can use, to add an element to the ArrayBlockingQueue.

	Blocks?	Returns	Throws InterruptedException?	Adds To Queue
`add(E e)`	No	boolean	No, throws IllegalStateException (Unchecked)	Yes
`offer(E e)`	No	boolean	No	Yes
`offer(E e, long timeout, TimeUnit unit)`	Temporarily	boolean	Yes	Yes
`put(E e)`	Yes	void	Yes	Yes