**Concurrent collections**

**Concurrent collections** are specialized data structures designed to handle concurrent **access by multiple threads**. They are part of the **java.util.concurrent** package and provide thread-safe operations without the need for explicit synchronization.

**Common collections:**

**ConcurrentHashMap:** A thread-safe version of HashMap that allows concurrent read and write operations.

**CopyOnWriteArrayList:** An implementation of List that makes a fresh copy of the underlying array with every mutation. It's highly efficient for scenarios where iteration is far more common than modification.

**BlockingQueue (ArrayBlockingQueue):** An interface that extends Queue with operations that wait for the queue to become non-empty when retrieving an element and wait for space to become available in the queue when storing an element. ArrayBlockingQueue and LinkedBlockingQueue are notable implementations.

**ConcurrentLinkedQueue:** A suitable choice for high-concurrency scenarios, it provides an ordered list of elements with efficient non-blocking operations.

**ConcurrentSkipListMap** and **ConcurrentSkipListSet**: Thread-safe variants of TreeMap and TreeSet that provide concurrent access with sorted order. They are based on skip list data structures, allowing for efficient concurrent access.

**Concurrent vs Synchronized Collections**

**Design and Implementation**:

**Synchronized Collections**: These are part of the **java.util.Collections** utility class, which provides methods to wrap standard collections **like List, Set, and Map** with **synchronized versions**. The synchronization is achieved using intrinsic locks (synchronized keyword), which lock the entire collection for each operation.

**Concurrent Collections**: These are part of the **java.util.concurrent** package and are designed specifically for **concurrent access**. They use techniques like fine-grained locking, lock-free algorithms, or segmented locking.

**Performance**:

**Synchronized Collections**: synchronized collections can become a bottleneck under high contention, as only one thread can access the collection at a time.

**Concurrent Collections**: These collections are optimized for concurrent access, allowing multiple threads to read and write simultaneously without blocking each other.

**Popular Synchronized Collections**

1. **SynchronizedList**: created using Collections.synchronizedList(new ArrayList<>()).
2. **SynchronizedSet**: created using Collections.synchronizedSet(new HashSet<>()).
3. **SynchronizedMap**:created using Collections.synchronizedMap(new HashMap<>()).