1. What is the Java Collections Framework?

The Java Collections Framework is a set of classes and interfaces that implement commonly reusable collection data structures. It includes interfaces like List, Set, and Map, and their implementations such as ArrayList, HashSet, and HashMap. The framework provides a standardized way to handle groups of objects.

2. What are the main interfaces of the Java Collections Framework?

The main interfaces are:

\*Collection\*: The root interface for manipulating a group of objects.

\*List\*: An ordered collection (also known as a sequence) that allows positional access and duplicate elements.

\*Set\*: A collection that does not allow duplicate elements.

\*Map\*: An object that maps keys to values, with no duplicate keys allowed.

\*Queue\*: A collection for holding elements prior to processing, typically in a FIFO order.

\*Deque\*: A double-ended queue that allows insertion and removal from both ends.

3. What is the difference between List and Set?

\*List\*: An ordered collection that allows duplicate elements and maintains the insertion order. Example implementations are ArrayList and LinkedList.

\*Set\*: A collection that does not allow duplicate elements and does not maintain any order. Example implementations are HashSet and TreeSet.

4. How does HashMap work internally?

HashMap works on the principle of hashing. It uses an array of buckets to store key-value pairs. When you insert a key-value pair, the hash code of the key is calculated and the bucket location is determined using this hash code. If a collision occurs (i.e., two keys have the same hash code), it uses a linked list to store the entries. In Java 8, if the number of entries in a bucket exceeds a certain threshold, the linked list is replaced by a balanced tree (Red-Black Tree) to improve performance.

5. What are the differences between ArrayList and LinkedList?

\*ArrayList\*: Implements a resizable array. It provides fast random access to elements (O(1) time complexity for get operations) but slow insertions and deletions (O(n) time complexity in the worst case).

\*LinkedList\*: Implements a doubly-linked list. It provides fast insertions and deletions (O(1) time complexity for add/remove operations at the beginning or end of the list) but slow random access to elements (O(n) time complexity for get operations).

6. What is the difference between HashMap and TreeMap?

\*HashMap\*: Uses a hash table to store key-value pairs. It provides O(1) time complexity for put and get operations but does not maintain any order of keys.

\*TreeMap\*: Uses a Red-Black Tree to store key-value pairs. It provides O(log n) time complexity for put and get operations and maintains keys in a sorted order.

7. What is the ConcurrentHashMap class?

ConcurrentHashMap is a thread-safe variant of HashMap that allows concurrent read and write operations without locking the entire map. It uses a technique called lock stripping, which divides the map into multiple segments and locks only the segment that is being modified, thus allowing higher concurrency and better performance in multi-threaded environments.

8. Explain the fail-fast behavior of iterators in Java.

Fail-fast iterators throw a ConcurrentModificationException if the underlying collection is modified while iterating over it (except through the iterator's own remove method). This behavior is intended to prevent unpredictable results by ensuring that a collection cannot be modified concurrently with iteration.

9. How does the Collections class differ from the Collection interface?

\*Collection interface\*: The root interface in the collection hierarchy that defines standard operations on collections (e.g., add, remove, size).

\*Collections class\*: A utility class that consists of static methods for manipulating collections, such as sorting, searching, and synchronization. It provides various algorithms and implementations for collection operations.

10. What is the difference between Iterator and ListIterator?

\*Iterator\*: Provides a way to traverse elements sequentially in a collection. It supports the `next()`, `hasNext()`, and `remove()` methods.

\*ListIterator\*: Extends Iterator and provides additional methods for bidirectional traversal (both forward and backward) of a list. It supports `previous()`, `hasPrevious()`, `nextIndex()`, `previousIndex()`, `add()`, and `set()` methods.

11. Explain the differences between synchronized and concurrent collections in Java.

\*Synchronized Collections\*: Typically created by using the `Collections.synchronizedXXX()` methods. These collections are thread-safe by ensuring that every method is synchronized, which can lead to performance bottlenecks under high contention.

\*Concurrent Collections\*: Part of the `java.util.concurrent` package, these collections are designed for high concurrency and provide better performance under multi-threaded scenarios by using more sophisticated techniques like non-blocking algorithms and lock stripping. Examples include `ConcurrentHashMap`, `CopyOnWriteArrayList`, and `ConcurrentLinkedQueue`.

12. What is a PriorityQueue in Java?

PriorityQueue is an unbounded queue based on a priority heap. The elements are ordered according to their natural ordering or by a Comparator provided at the time of queue construction. It provides O(log n) time complexity for the insertion and removal of elements.

13. What is the purpose of the Collections.unmodifiableXXX() methods?

The `Collections.unmodifiableXXX()` methods return an unmodifiable view of the specified collection. This view does not allow modifications, meaning any attempt to add, remove, or modify elements will result in an UnsupportedOperationException. These methods are used to create read-only collections.

14. Can you explain the concept of "autoboxing" in Java with respect to collections?

Autoboxing is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes. For example, in the context of collections, if you have a `List<Integer>`, you can add an int directly (`list.add(5)`), and the compiler will automatically convert it to `Integer`.

15. What is a WeakHashMap and when would you use it?

A WeakHashMap is a variant of HashMap that uses weak references for its keys. This means that if a key is no longer referenced outside of the WeakHashMap, it can be garbage collected, and the corresponding entry will be automatically removed from the map. This is useful for implementing caches or memory-sensitive applications where you want to allow entries to be garbage collected when they are no longer in use.