Top 50 Interview Questions and Answers on Java Collections Framework

1. What is the Java Collections Framework?
2. List the main interfaces in the Java Collections Framework.
3. What is the difference between Collection and Collections?
4. What is the difference between List and Set?
5. What is the difference between ArrayList and LinkedList?
6. How does HashMap work internally?
7. Explain the difference between HashMap and Hashtable.
8. What is the load factor in HashMap?
9. What is the difference between HashSet and TreeSet?
10. How does LinkedHashMap maintain insertion order?
11. What is fail-fast behavior in Java Collections?
12. What are the key differences between List and Queue?
13. What is the purpose of the Map interface?
14. Explain the difference between Comparator and Comparable.
15. How does the `contains()` method work in a Set?
16. What happens when you insert a null key into a HashMap?
17. How do you convert an ArrayList to an array?
18. What are the different ways to traverse a List?
19. What is a priority queue, and how is it different from a regular queue?
20. Explain the differences between `Iterator` and `ListIterator`.
21. How do you remove duplicates from a List?
22. What is the difference between shallow copy and deep copy in collections?
23. What is the difference between Collections.sort() and List.sort()?
24. How can you synchronize a List in Java?
25. What is the difference between `subList()` and `slice()`?
26. Explain the `poll()` and `remove()` methods in the Queue interface.
27. What is the significance of the `retainAll()` method?
28. What is a `WeakHashMap`, and when would you use it?

29. Explain the purpose of the `Deque` interface.
30. How do you implement a stack using the Collections Framework?
31. What is a `LinkedHashSet`, and how does it differ from HashSet?
32. What is the difference between `clone()` and `copyOf()` in collections?
33. How do you sort a Set using a custom comparator?
34. Explain the concept of `ConcurrentHashMap`.
35. What is a `NavigableSet` and how does it extend `SortedSet`?
36. How can you check if two collections are equal?
37. What are the performance implications of using different collection types?
38. Explain the concept of a `Map.Entry` in a HashMap.
39. What is the difference between `remove()` and `clear()` in collections?
40. How can you find the intersection of two Sets?
41. What are the characteristics of a `Stack` in Java?
42. What is the difference between a `List` and an `Array`?
43. How can you iterate over a Map in Java?
44. What is the purpose of the `Collections` utility class?
45. How do you implement a queue using the Collections Framework?
46. What is a `SortedMap`, and how does it work?
47. Explain the difference between `toArray()` and `toArray(T[])`.
48. How do you create an immutable list in Java?
49. What are the advantages of using a TreeSet over a HashSet?
50. What is the `Stream` API, and how does it relate to collections?

Here are the answers corresponding to the top 50 interview questions on the Java Collections Framework:

1. **The Java Collections Framework is a unified architecture for representing and manipulating collections, providing interfaces and classes for data structures and algorithms.**
2. **The main interfaces are: Collection, List, Set, Map, Queue, and Deque.**

3. `Collection` is a root interface for collections, while `Collections` is a utility class providing static methods for operations on collections.
4. `List` allows duplicate elements and maintains order, while a `Set` does not allow duplicates and does not guarantee order.
5. `ArrayList` is backed by a dynamic array, providing fast random access, while `LinkedList` is backed by a doubly-linked list, providing efficient insertions and deletions.
6. `HashMap` uses an array of buckets, where each bucket holds a linked list of entries. It calculates an index using a hash function and resolves collisions with chaining.
7. `HashMap` is not synchronized and allows null keys/values, while `Hashtable` is synchronized and does not allow null keys/values.
8. The load factor is a measure of how full the hash table is allowed to get before its capacity is automatically increased. The default load factor is 0.75.
9. `HashSet` is implemented using a hash table and does not guarantee order, while `TreeSet` maintains a sorted order of elements using a red-black tree.
10. `LinkedHashMap` maintains insertion order by using a doubly-linked list that runs through its entries.
11. Fail-fast behavior occurs when a collection is modified while being iterated, causing a `ConcurrentModificationException`.
12. A `List` maintains the order of elements and allows duplicates, while a `Queue` is designed for holding elements prior to processing and typically follows FIFO ordering.
13. The `Map` interface represents a collection of key-value pairs, providing methods for storing, retrieving, and manipulating key-value associations.
14. `Comparator` is used to define a custom order for objects, while `Comparable` is used to define a natural ordering for objects.
15. The `contains()` method checks if a specified element exists in a `Set` by computing its hash code and looking for it in the appropriate bucket.
16. When you insert a null key into a `HashMap`, it allows one null key, while `Hashtable` does not permit null keys or values.

17. You can convert an `ArrayList` to an array using the `toArray()` method, which can be called with or without an array parameter.
18. You can traverse a `List` using a for loop, enhanced for loop, `Iterator`, or `ListIterator`.
19. A priority queue orders its elements based on their natural ordering or a specified comparator, allowing access to the highest (or lowest) priority element.
20. The `Iterator` provides a way to traverse a collection, while `ListIterator` can traverse a list in both directions and has additional methods for manipulating the list.
21. To remove duplicates from a `List`, you can convert it to a `Set` and then back to a `List`.
22. A shallow copy duplicates only the references to the objects, while a deep copy duplicates the objects themselves.
23. `Collections.sort()` is a static method for sorting a list, while `List.sort()` is an instance method that sorts the list directly.
24. You can synchronize a `List` using `Collections.synchronizedList(new ArrayList<>())` or by using `CopyOnWriteArrayList`.
25. `subList()` returns a view of a portion of the list, while `slice()` is not a method in the Collections Framework.
26. The `poll()` method retrieves and removes the head of the queue, returning `null` if the queue is empty, while `remove()` throws an exception if the queue is empty.
27. The `retainAll()` method retains only the elements that are contained in the specified collection, removing all other elements.
28. A `WeakHashMap` allows its keys to be garbage collected when they are no longer in use, which helps to prevent memory leaks.
29. The `Deque` interface represents a double-ended queue that allows adding/removing elements from both ends, providing more flexibility than a standard queue.
30. You can implement a stack using the `Deque` interface by using methods like `push()`, `pop()`, and `peek()`.

31. A `LinkedHashSet` maintains insertion order using a linked list while providing the same performance characteristics as `HashSet`.
32. The `clone()` method creates a shallow copy of the collection, while `copyOf()` creates a new immutable collection based on the existing one.
33. You can sort a `Set` by first converting it to a `List` and then using the `Collections.sort()` method with a custom comparator.
34. A `ConcurrentHashMap` allows concurrent access by dividing the map into segments, enabling multiple threads to access the map without locking the entire structure.
35. A `NavigableSet` is a subinterface of `SortedSet` that provides navigation methods for finding elements based on their value.
36. You can check if two collections are equal using the `equals()` method, which checks for both size and content equality.
37. The performance implications depend on the specific implementation: `ArrayList` provides fast access but slow insertions; `LinkedList` provides fast insertions but slow access.
38. The `Map.Entry` interface represents a key-value pair in a map, allowing access to both the key and value.
39. The `remove()` method removes a specific element, while `clear()` removes all elements from the collection.
40. To find the intersection of two sets, you can use the `retainAll()` method, which modifies the original set to retain only the elements that are also contained in the specified collection.
41. A Stack follows LIFO (last-in-first-out) order, allowing elements to be added and removed from the same end. It supports operations like `push()`, `pop()`, and `peek()`.
42. The main difference between a List and an Array is that a List is a resizable collection that can grow and shrink dynamically, while an Array has a fixed size.
43. You can iterate over a Map using the `entrySet()`, `keySet()`, or `values()` methods combined with an enhanced for loop or an iterator.

44. The `Collections` utility class provides static methods for operations on collections, such as sorting, searching, and creating synchronized collections.
45. You can implement a queue using the `Deque` interface by using methods like `offer()`, `poll()`, and `peek()`.
46. A `SortedMap` is a subinterface of `Map` that maintains its entries in ascending key order, typically implemented by `TreeMap`.
47. The difference between `toArray()` and `toArray(T[])` is that the first returns an array of `Object`, while the second returns an array of the specified type.
48. To create an immutable list in Java, you can use `List.of()` or `Collections.unmodifiableList()` methods.
49. The advantages of using a `TreeSet` over a `HashSet` include sorted order of elements and the ability to perform range queries.
50. The Stream API allows for functional-style operations on collections, enabling processing of sequences of elements in a declarative manner.