1. Collection?

   A collection is an object that groups multiple elements into a single unit. The Java Collections Framework is a unified architecture for representing and manipulating collections. It contains the interfaces, their implementations, and algorithms to process the data stored in a collection

2. Difference between Collection and Collections?

   A Collection is an interface, whereas Collections is a class.

   A Collection interface provides the standard functionality of a data structure to List, Set, and Queue. However, the Collections class provides the utility methods that can be used to search, sort, and synchronize collection elements.

3. ArrayList?

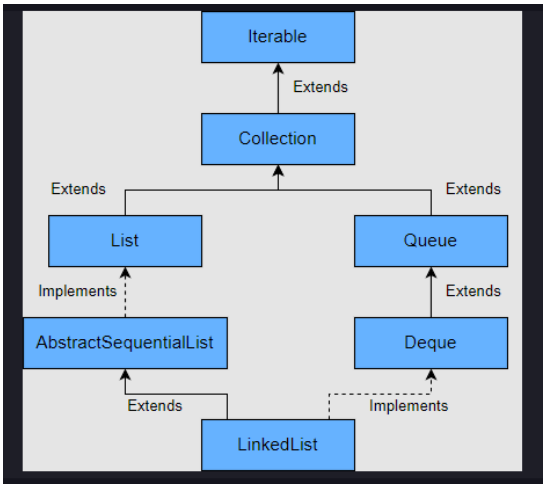
   ArrayList is the most widely used implementation of the List interface. Some of the salient features of an ArrayList are:

   - Elements are stored in the order of insertion.
   - It allows the storage of duplicate elements.
   - ArrayList also supports null elements.

4. LinkedList?

   The LinkedList class in Java implements the List and the Deque interface. Some of the salient features of a LinkedList are:

   - The elements are inserted in the order of insertion.
   - It supports duplicate elements.
   - We can add any number of null elements.



5. Hashset?

   HashSet is a class in the java.utils package, which implements the Set interface. Some of the features of HashSet are:

   - HashSet does not allow duplicate elements.
   - HashSet allows only one null element.
   - The elements are inserted in random order in a HashSet.
   - A HashSet is internally backed by a HashMap.

6. Creation of Hashset?

   The simplest way to create a HashSet is by using the no-arg constructor. This constructor creates a HashSet with an initial capacity of 16 and a load factor of 0.75.

   Below is the code syntax to create a HashSet.

   Set<Integer> set= new HashSet<>();

7. Load Factor?

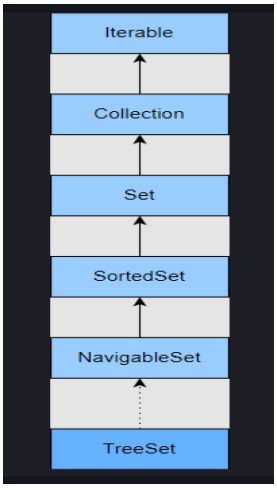
   Load factor is a number that defines when a Set should be resized. If the load factor is 0.75, then the Set should be resized when it is 75% full.

8. Treeset?

   Java TreeSet class implements the Set interface that uses a tree for storage. It inherits the AbstractSet class and implements the NavigableSet interface.

   Some of the features of TreeSet are:

   - TreeSet does not allow duplicate elements.
   - TreeSet class doesn't allow null elements.
   - Since elements are stored in a tree, the access and retrieval times are quite fast in a TreeSet.
   - The elements are stored in ascending order in a TreeSet.



9. Difference between HashSet and TreeSet?

    - The HashSet allows one null element, whereas a TreeSet does not allow a null element.
    - The elements are stored in random order in a HashSet, whereas it is stored in sorted order in TreeSet.
    - HashSet is faster than Treeset for the operations like add, remove, contains, size, etc.

10. Treeset Prerequisite?

    Since all the elements are stored in sorted order in a TreeSet, storing elements should implement either the **Comparable** interface or a custom **Comparator** while creating the TreeSet.

11. Hashmap?

    HashMap is a class in the java.utils package that implements the Map interface. It is used to store the key-value pair. Let's suppose we need to store the stock prices of some companies. In this case, we can use a HashMap. The company name will be the key and the stock price will be the value.

    Some of the features of HashMap are:

    - The keys should be unique.
    - HashMap allows only one null key.
    - The values can be null or duplicate.
    - The keys are stored in random order.

12. HashMap Internal Working?

    The basic principle used by a HashMap to store elements is Hashing. Hashing is a way to assign a unique code for any variable or object after applying any formula to its properties. The unique code is called HashCode.

    Some of the properties of HashCode are:

    - If two objects are equal, then they should have the same hashcode.
    - If two objects have the same hashcode, then it is not necessary for them to be equal.

13. Hashmap Internal Flow?

Creating a HashMap#
We already know that the HashMap stores key-value pairs. HashMap has a nested static class called Node as shown below.

```
static class Node<K,V> implements Map.Entry<K,V> {
        final int hash;
        final K key;
        V value;
        Node<K,V> next;
...some more code
```

This class has a key and a value field. It also has a next field that is used to point to the next Node.

HashMap also has a field called table as shown below. It is basically an array of Node objects that are not yet initialized.

transient Node<K,V>[] table;

When we create a HashMap using the no-arg constructor, then the only thing that happens is that the loadFactor is assigned DEFAULT_LOAD_FACTOR, which is .75
.75
.

The table array that we discussed above is not initialized at the time of the creation of HashMap.

```
public HashMap() {
      this.loadFactor = DEFAULT_LOAD_FACTOR; // all other fields defaulted
   }
```

Inserting into a HashMap#
When an element is inserted into the Hashmap for the first time, the array table is initialized with size 16. So now there are 16 buckets from index 0 to 15.

If the key that we are inserting is null, then it is inserted at index 0 because the hashcode of null is 0. If the key is not null, then the hash of the key is calculated, and based on the hash value the bucket is decided.

If there is no other element in that bucket, then a new Node is created, and it is inserted in that bucket.

Now, let's say we insert another key that has the same hashcode as the previous key. In this case, this key will go to the same bucket and see that there is already an element there. This is referred to as collision.

In case of collision, it checks if the existing key in the bucket is equal to the key that we are trying to store. If yes, then the value of the key is updated. If the key is different, then it is added at the end of the existing key in the bucket to form a LinkedList.

There is one improvement made in Java 8. If the size of the LinkedList in a particular bucket becomes more than TREEIFY_THRESHOLD, then the LinkedList is converted to a red-black tree. TREEIFY_THRESHOLD is a constant with a default value of 8. This value can't be changed as it is a final variable. This means that if the size of the Linked List becomes more than 8, then it is converted into a tree.

Fetching a value from HashMap#
When we need to get a value from HashMap, then the hashcode of the key is calculated. Based on the hashcode the index is decided, and we go to that index. Now it is possible that there are zero or more keys stored at that index. We match our key with all the keys at that index using the equals() method. If the match is found, then we return the value of that key. If the key is not found, then null is returned.

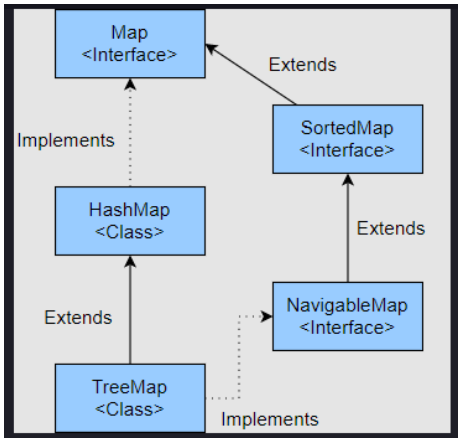
Resizing a HashMap#
We already know that a HashMap is resized when it is about to get full. When a HashMap will be resized depends upon the load factor. If the current capacity is 16, and the load factor is 0.75, then the HashMap will be resized when it has 12 elements (16 * 0.75).

When a HashMap is resized, its capacity is always doubled. So if the current capacity is 16, then the new capacity will be 32. Now all the elements that are stored in the HashMap will be rearranged amongst these 32 buckets. This is a time-consuming operation as the bucket for each key is calculated and rearranged.

14. Treemap?

TreeMap is a class in the java.utils package that stores the keys in sorted order. Some of the features of TreeMap are:

- The entries in TreeMap are sorted in the natural ordering of its keys.
- It does not allow null keys, however there can be null values.
- The TreeMap is not thread-safe, although it can be made thread-safe using the synchronizedMap() method of the Collections class.
- Since a TreeMap stores the keys in sorted order, the objects that we are storing in the TreeMap should either implement the Comparable interface or we should pass a Comparator while creating the TreeMap object.



15. LinkedHashMap?
We have already discussed that a HashMap does not maintain insertion order and TreeMap stores the elements in sorted order. Now, if we want to store the elements in a Map in insertion order, then a LinkedHashMap can be used. LinkedHashMap is a class in the java.utils package that implements the Map interface and extends the HashMap class. It is similar to HashMap with the additional feature of maintaining the order of elements inserted into it.

Some of the important features of a LinkedHashMap are:

- It does not allow duplicate keys.
- It may have one null key and multiple null values.
- It is non-synchronized.

16. Difference between ConcurrentHashMap and SynchronizedMap?

    - In a SynchronizedMap, the entire Map is locked. So every read/write operation needs to acquire a lock, which makes it very slow. On the other hand in a ConcurrentHashMap, only a segment of the Map is locked. Two parallel threads can access or update elements in a different segment, so it performs better.

    - SynchronizedMap returns Iterator, which fails fast on concurrent modification. ConcurrentHashMap doesn't throw a ConcurrentModificationException if one thread tries to modify it while another is iterating over it.

    - ConcurrentHashMap does not allow null keys or null values while SynchronizedMap allows one null key.

17. IdentityHashMap?

    IdentityHashMap is another type of Map that implements Map, Serializable, and Cloneable interfaces and extends the AbstractMap class. The main feature of this map is that while storing elements, the equality of the keys is checked on the basis of reference instead of the equals method. What this means is that if we have two keys, key1 and key2, then key1 will be considered equal to key2 if key1 == key2. In other words, if both the keys reference the same object.

    This means that IdentityHashMap intentionally violates Map's general contract which mandates the use of the equals method when comparing objects. This class is designed for use only in rare cases wherein reference-equality semantics are required.

    Some of the features of IdentityHashMap are:

    - The IdentityHashMap stores the elements in random order.
    - The IdentityHashMap allows a single null key.
    - The IdentityHashMap is not thread-safe.

18. Difference between IdentityHashmap and HashMap?

    - IdentityHashMap uses reference equality to compare keys and values while HashMap uses object equality to compare keys and values.
    - IdentityHashMap does not use the hashCode() method. Instead it uses System.identityHashCode() to find the bucket location.
    - IdentityHashMap does not require keys to be immutable as it does not rely on the equals() and hashCode() methods. To safely store the object in HashMap, keys must be immutable.
    - The default initial capacity of HashMap is 16; whereas, for IdentityHashMap, it is 32.

19. EnumMap?

    The EnumMap is a special kind of Map in which the keys are only of the Enum type. Although it is possible to use the Enum type as a key in other Map implementations such as a HashMap or TreeMap, but a separate Map implementation was created for performance reasons.

The EnumMap is highly efficient, and we will discuss more when we will look at how it works internally. Let's look at some of the features of EnumMap.

- EnumMap does not allow null keys, but it allows null values.
- The keys are stored in their natural order. In the case of an Enum, the natural order of keys means the order where enum constant is declared inside Enum type.
- The EnumMap is not synchronized.
- All keys of each EnumMap instance must be keys of a single Enum type.
- Iterators returned by the collection views are inconsistent. They will never throw ConcurrentModificationException, and they may or may not show the effects of any modifications to the map that occur while the iteration is in progress.
- Java EnumMap implementation provides constant-time performance for the basic operations (like get and put).

20. Arrays – Searching?

The Arrays class provides the binarySearch() method to search for a particular element in an array. There are a lot of overloaded binarySearch() methods to handle all the primitive types. Some of the important points to note about the binarySearch() method are:

- The array that is passed to the method should be sorted. If the array is not sorted, then the result is undefined.

- This method returns the index where the element is present in the array. If the element is not present in the array, then the index of the first element greater than the key is returned.

- If the array contains multiple elements with the specified value, there is no guarantee which one will be found.

- ClassCastException is thrown if the search key is not comparable to the elements of the array.

As the name suggests, the binarySearch() method uses the binary search algorithm to search for an element in the array. It is far better than a linear search. The complexity of the linear search algorithm is O(n), whereas the complexity of the binary search algorithm is O(log n).

21. Sorting an Array?

The Arrays class has a sort() method that is used to sort the arrays of objects and primitives. If we are sorting a primitive array, then quicksort is used. And if we are sorting an object array, then merge sort is used.

Although quicksort is faster in both cases, it is not a stable algorithm. Merge sort is a stable algorithm, so it is used in the case of sorting an object array. In the case of the primitive array, we don't care about stability, so quicksort is used.

22. Stable Algorithms?

    Stable sorting algorithms are algorithms that maintain the relative order of equal elements. For example, we have an array [1,4,6,8,6], which we need to sort. Now after sorting this array, the result is [1,4,6,6,8]. Although there are two sixes in the array, we don't care which six came first in the sorted array. But in the case of an object array, the relative order of elements also matters. If two objects are the same in an object array, then their relative order should be the same in the sorted array.