1. **Draw Collections Framework Class Diagram**

**Collections**

**List(Interface)**

**Set(Interface)**

**SortedSet(Interface)**

**Vector**

**HashSet**

**ArrayList**

**LinkedList**

**LinkedHashSet**

**TreeSet**

**Map(Interface)**

**SortedMap(Interface)**

**LinkedHashMap**

**TreeMap**

**HashMap**

**HashTable**

2. **What is HashMap and Map?**

**HashMap :** It is a Map based collection class that is used for storing Key & value pairs, it is denoted as HashMap<Key, Value> or HashMap<K, V>.

* This class makes no guarantees as to the order of the map. It is similar to the Hashtable class except that it is unsynchronized and permits nulls(null values and null key).
* It is not an ordered collection which means it does not return the keys and values in the same order in which they have been inserted into the HashMap.
* It does not sort the stored keys and Values. You must need to import java.util.HashMap or its super class in order to use the HashMap class and methods.

**Map :** A map contains values on the basis of key i.e. key and value pair. Each key and value pair is known as an entry. Map contains only unique keys.

* Map is useful if you have to search, update or delete elements on the basis of key.

3. **Difference between HashMap and HashTable? Can we make hashmap synchronized?**

* HashMap is non synchronized. It is not-thread safe and can’t be shared between many threads without proper synchronization code whereas Hashtable is synchronized. It is thread-safe and can be shared with many threads.
* HashMap allows one null key and multiple null values whereas Hashtable doesn’t allow any null key or value.
* HashMap is generally preferred over HashTable if thread synchronization is not needed.

4. **Difference between Vector and ArrayList?**

* **Synchronization :** Vector is synchronized that means at a time only one thread can access the code while arrayList is not synchronized that means multiple threads can work on arrayList

at same time. For example, if one thread is performing add operation, then there can be another thread performing remove operation in multithreading environment.

If multiple threads access arrayList concurrently then we must synchronize the block of the code which modifies the list either structurally or simple modifies element.

* **Performance**: ArrayList is faster as it is non-synchronized while vector operations give slow performance as they are synchronized(thread-safe). If one thread works on vector has acquired lock on it which makes other thread will has to wait till lock is released.
* **Data Growth:** ArrayList and Vector both grow and shrink dynamically to maintain optimal use of storage. But the way they resized is different. ArrayList increments 50% of current array size if number of elements exceeds its capacity while vector increments 100% means doubles the current array size.
* **Traversal:** Vector can use both Enumeration and Iterator for traversing over elements of vector while ArrayList can only use Iterator for traversing.

5. **What is an Iterator?**

* In Java, Iterator is an interface available in Collection framework in java.util package.
* It is a Java Cursor used to iterate a collection of objects.
* It is used to traverse a collection object elements one by one.
* It is available since Java 1.2 Collection Framework.
* It is applicable for all Collection classes. So it is also known as Universal Java Cursor.
* It supports both READ and REMOVE Operations.
* Compare to Enumeration interface, Iterator method names are simple and easy to use.

**Iterator Methods**

* boolean hasNext():Returns true if the iteration has more elements.
* E next(): Returns the next element in the iteration.
* default void remove(): Removes from the underlying collection the last element returned by this iterator.
* default void forEachRemaining(Consumer action): Performs the given action for each remaining element until all elements have been processed or the action throws an exception.

**6. Difference between list vs Set vs Map**

* The Set interface provides an unordered collection of unique objects, i.e. Set doesn't allow duplicates, while Map provides a data structure based on key-value pair and hashing.
* All three List, Set, and Map are interfaces in Java and there are many concrete implementations of them are available in Collection API. ArrayList and LinkedList are two most popular used List implementation while LinkedHashSet, TreeSet, and HashSet are frequently used Set implementation.
* In this Java article, we will see the difference between Map, Set, and List in Java and learn when to use List, Set or Map.
* List and Map are interfaces, which defines core contract e.g. a Set contract says that it can not contain duplicates. Based on our knowledge of List, Set and Map let's compare them on different metrics.
* The main difference between List and Set interface in Java is that List allows duplicates while Set doesn't allow duplicates. All implementation of Set honor this contract.
* While a Map holds two objects per Entry e.g. a key and a value and It may contain duplicate values but keys are always unique. See here for more difference between List and Set data structure in Java.
* Another key difference between List and Set is that List is an ordered collection, List's contract maintains insertion order or element. Set is an unordered collection, you get no guarantee on which order element will be stored.
* Though some of the Set implementation e.g. LinkedHashSet maintains order. Also SortedSet and SortedMap e.g. TreeSet and TreeMap maintain a sorting order, imposed by using Comparator or Comparable.

**7. Pros and cons of ArrayList and LinkedList**

Pros of ArrayList

* ArrayList uses internally an array for internal storage. That makes it particularly fast for random access - get(#n).

Cons of ArrayList

* ArrayList is slower for modification operations like add or delete elements in the beginning or middle of the collection. This is due to the need of relocate all subsequent elements one position to the right (or left in case of deletion) in order to make space to the new element.
* Similar to before described process. ArrayList has some performance downside when the internal array is completely full, and therefore has to create a bigger array and relocate all elements to new array.

Pros of LinkedList

* LinkedList follows a different approach. It's more efficient in adding or deleting elements in the beggining or middle of the collection. If you ever programmed from scratch a list data structure, you will remember you have nodes with pointers/references to the next element.
* In this case what is done to insert a new node in the middle of the list is to create a new node pointing to the next element (->c), and the before element is updated to point to the new created node (->b).
* Given the nature of the internal structure which is not restricted to an initial size, LinkedList has no growing problems as ArrayList.

Cons of LinkedList

* Random access to LinkedList elements are expensive, because in worst case scenarios the entire list has to be traversed to retrieve the desired element (O(n)).
* We could say that we should use ArrayList if we have many random accesses. If we think our lists are going to grow unexpectedly, we should favor LinkedList. This is just one scenario.
* We could have both needs in which case a combination of both approaches could be use. Like using LinkedList to create the list, and then use ArrayList for read access.

8. **TreeSet Vs LinkedHashedSet**

* Both TreeSet, and LinkedHashSet are not synchronized. They can not be shared between multiple threads until specifically synchronized. It's easy to create synchronized Set, though, all you need to do is use java.util.Collections utility class as shown below :

Synchronizing LinkedHashSet in Java

Set s = Collections.synchronizedSet(new LinkedHashSet(...));

Synchronizing TreeSet in Java

Set s = Collections.synchronizedSet(new TreeSet(...));

* Ordering

TreeSet sorts all object based upon there natural ordering by using compareTo() method,

or custom order by using compare() method Comparator passed to them. LinkedHashSet also provides ordering support to keep elements in the order they are added into Collection.

this property is also derived from the fact that they are backed by respective Map implementation.

* Null Element

This property can be deduced form LinkedHashMap, and TreeMap since HashSet internally uses HashMap, LinkedHashSet internally uses LinkedHashMap and TreeSet internally uses TreeMap.

LinkedHashMap allows one null key and so are these two Set implementations. On the other hand, since TreeMap doesn't allow null keys, TreeSet doesn't allow null elements and

throws java.lang.NullPointerException when you try to add a null object. The main reason of this is the use of compareTo() and compare() method, which throws NullPointerException if one element is null, but it truly depends on implementation.

* Implementation

HashSet internally uses a HashMap with dummy value object, while LinkedHashSet uses a LinkedHashMap to guarantee insertion order.

When you iterate through HashSet order is unpredictable but when you iterate through LinkedHashSet.

**9. What are relationships between equals and hash codes?**

* equals(Object obj): a method provided by java.lang.Object that indicates whether some other object passed as an argument is "equal to" the current instance.

The default implementation provided by the JDK is based on memory location — two objects are equal if and only if they are stored in the same memory address.

* hashcode(): a method provided by java.lang.Object that returns an integer representation of the object memory address. By default, this method returns a random integer that is unique for each instance. This integer might change between several executions of the application and won't stay the same

The Contract Between equals() and hashcode()

* The default implementation is not enough to satisfy business needs, especially if we're talking about a huge application that considers two objects as equal when
* some business fact happens. In some business scenarios, developers provide their own implementation in order to force their own equality mechanism regardless the memory addresses.

**10. What are the advantages of ArrayList over arrays ?**

* Defined ArrayList as re-sizable array. Size of the ArrayList is not fixed. ArrayList can grow and shrink dynamically.
* Elements can be inserted at or deleted from a particular position.
* ArrayList class has many methods to manipulate the stored objects.
* ArrayList class has methods to perform solo modifications ( add(), remove()… ), bulk modifications ( addAll(), removeAll(), retainAll()… ),searching( indexOf(), lasIndexOf() ) and iterations( iterator() ).If generics are not used, ArrayList can hold any type of objects.
* Many are of the assumption that multiple insertion and removal operations on ArrayList will decrease the performance of an application.
* But, there will be no significant change in the performance of an application if you use ArrayList instead of arrays. Below example shows time taken to add 1000 string elements to ArrayList and array.

11. **Principles of hashtables**

* Hashtable is an implementation of a key-value pair data structure in java. You can store and retrieve a ‘value’ using a ‘key’ and it is an identifier of the value stored.

It is obvious that the ‘key’ should be unique.

* b. Java.util.Hashtable extends Dictionary and implements Map. Objects with non-null value can be used as a key or value.

Key of the Hashtable must implement hashcode() and equals() methods. By the end of this article you will find out the reason behind this condition.

* Generally a Hashtable in java is created using the empty constructor Hashtable(). Which is a poor decision and an often repeated mistake. Hashtable has two other constructors
* Hashtable(int initialCapacity) and Hashtable(int initialCapacity, float loadFactor). Initial capacity is number of buckets created at the time of Hashtable instantiation.
* Bucket is a logical space of storage for Hashtable.

**12. Differences between Hashtable, ConcurrentHashMap and Collections.synchronizedMap()**

* Difference between HashMap and ConcurrentHashMap is that later is thread-safe and can be used in a concurrent environment without external synchronization.
* Though it doesn't provide the same level of synchronization as achieved by using Hashtable but it's enough for the most practical purpose.
* HashMap can be synchronized by wrapping it on Collections.synchornizedMap(HashMap) which will return a collection which is almost equivalent to Hashtable,
* Where every modification operation on Map is locked on Map object while in case of ConcurrentHashMap, thread-safety is achieved by dividing whole Map into different partition based upon Concurrency level and only locking particular portion instead of locking the whole Map.

**Difference between ConcurrentHashMap and HashMap in Java Collection**

* ConcurrentHashMap is more scalable and performs better than Synchronized HashMap in the multi-threaded environment while in Single threaded environment both HashMap and ConcurrentHashMap gives comparable performance, where HashMap only slightly better.
* Hashtable and Collections.synchronizedMap() provide the same degree of synchronization. If you were to wrap Hashtable through Collections. Synchronized you would have the same degree, but with another redundant layer, of synchronization.
* The main difference between Hashtable and Collections.synchronizedMap(HashMap) exist more at the API level.
* Because Hashtable is part of Java's legacy code, you'll see that the Hashtable API is enhanced to implement the Map interface, to become part of Java's collections framework.
* This means that if you were to wrap Hashtable through Collections.synchronizedMap(), the API of the wrapped Hashtable would become limited to the Map API.
* So if the API of Hashtable is encompassed in your definition of behavior, then it is obviously altered/limited.

**13. How are hash codes computed?**

Object (Java Platform SE 8 ) does not say anything particular about how hash values are calculated, but it does say “…This is typically implemented by converting the internal address of the object into an integer, but this implementation technique is not required by the Java programming language.”. But in the case of java.lang.String, String (Java Platform SE 8 ) says “The hash code for a String object is computed as s[0]\*31^(n-1) + s[1]\*31^(n-2) + ... + s[n-1] using int arithmetic, where s[i] is the ith character of the string, n is the length of the string, and ^ indicates exponentiation. (The hash value of the empty string is zero.). syntax - public int hashCode()

**14. Is it possible that hash code is not unique?**

* Since there are only 2^32 different ints and there may be more than 2^32 live objects in any VM instance, it is technically impossible to guarantee a unique hash code for each object.
* Even if the default hash code may be based on the internal address of the object, it is not identical to the internal address.
* It is not required that if two objects are unequal according to the equals(java.lang.Object) method, then calling the hashCode method on each of the two objects
* must produce distinct integer results.

**15. Can we put two elements with equal hash code to one hash map?**

* It is perfectly legal for two elemets can the same hash map.
* If two objects are equal (using the equals() method) then they have the same hashcode.

**16. Iterator and modification of a List. ConcurentModificationException.**

* remove() method is introduced in iterator. Using this method we can remove element from the underlying [collection](https://crunchify.com/how-to-convert-hashmap-to-arraylist-in-java/) which we are iterating.

Enumeration has two methods and both are available in iterator. Method names for both of them are shortened.

* We cannot add or remove elements to the underlying collection when we are using an iterator.

**17. What is the significance of ListIterator? What is the difference b/w Iterator and ListIterator?**

* Like Iterator, ListIterator is a Java Iterator, which is used to iterate elements one-by-one from a List implemented object.
* It is available since Java 1.2.
* It extends Iterator interface.
* It is useful only for List implemented classes.
* Unlike Iterator, It supports all four operations: CRUD (CREATE, READ, UPDATE and DELETE).
* Unlike Iterator, It supports both Forward Direction and Backward Direction iterations.
* It is a Bi-directional Iterator.
* It has no current element; its cursor position always lies between the element that would be returned by a call to previous() and the element that would be returned by a call to next().

**Difference b/w Iterator and ListIterator.**

* Iterator - Introduced in Java 1.2. ListIterator - Introduced in Java 1.2.
* Iterator - It is an Iterator for whole Collection API.ListIterator - It is an Iterator for only List implemented classes.
* Iterator - It is an Universal Iterator. ListIterator - It is NOT an Universal Iterator.
* Iterator - It supports only Forward Direction Iteration. ListIterator - It supports both Forward and Backward Direction iterations.
* Iterator - It’s a Uni-Directional Iterator. ListIterator - It’s a Bi-Directional Iterator.
* Iterator - It supports only READ and DELETE operations. ListIterator - It supports all CRUD operations.
* Iterator - We can get Iterator by using iterator() method. ListIterator - We can ListIterator object using listIterator() method.

**18. What is the Collections API?**

* The Collections API provides a number of interfaces (including Collection, List, Map and Set) to define a standard way of using a range of concrete data structures.
* The interfaces and classes of the Collections API belong to the java.util package

**Characteristics of collections include:**

* Ordered - It is possible to iterate over the elements of an ordered collection in a predictable order.
* Uniqueness of elements - Some collections do not allow duplicate elements. Objects are considered as duplicates if, according to their equals(Object) methods, they are equal.
* Array-based storage - Some collections use an array internally to store their elements. The array is resized to accommodate more elements. Array storage is generally fast to access but slow to remove or insert elements.
* Linked-list storage -In linked-lists each element is stored in another object that has a reference to the next and (in a double-linked list) previous element. Linked-lists are efficient at removing or inserting elements but slower for access.

**19. How can we access elements of a collection?**

* There are 4 ways to retrieve any elements from a collection object: For-each and using cursors.
* For each loop is meant for traversing items in a collection.
* Cursor is an interface and it is used to retrieve data from collection object,one by one. Cursor has 3 types - Iterator interface, ListIterator Interface and EnumerationIterator Interface.

**20. What is the difference between a queue and a stack?**

* Both Stack and Queue are built on top of basic data structure like an array or linked list. Since both Stack and Queue can be bounded and unbounded,
* it makes sense to use an array for bounded stack and queue and may be linked list (it suits problem domain) for an unbounded queue.
* The Java Collection API contains an implementation of both stack and queue data structure. It has a class called java.util.Stack which represents
* Stack and then it has a Queue interface, with a couple of implementation e.g. BlockingQueue, LinkedList, and PriorityQueue

**21. What is the Properties class?**

* Properties is a subclass of Hashtable. It is used to maintain lists of values in which the key is a String and the value is also a String. The Properties class is used by many other Java classes.
* For example, it is the type of object returned by System.getProperties( ) when obtaining environmental values.

**22. Which implementation of the List interface provides for the fastest insertion of a new element into the middle of the list?**

* Vector
* ArrayList
* LinkedList
* None of the above
* ArrayList and Vector both use an array to store the elements of the list. When an element is inserted into the middle of the list
* the elements that follow the insertion point must be shifted to make room for the new element. The LinkedList is implemented using a doubly linked list; an insertion requires only the updating of the links at the point of insertion. Therefore, the LinkedList allows for fast insertions and deletions.

**23. How can we use hashset in collection interface?**

* Implements Set Interface.
* Underlying data structure for HashSet is hashtable.
* As it implements the Set Interface, duplicate values are not allowed.
* Objects that you insert in HashSet are not guaranteed to be inserted in same order. Objects are inserted based on their hash code.
* NULL elements are allowed in HashSet.
* HashSet also implements Searlizable and Cloneable interfaces.

**25. Can you limit the initial capacity of vector in java?**

* A Vector uses an array internally which has a maximum size of exactly Integer.MAX\_VALUE, so it can't support more than that many elements

**26. What method should the key class of Hashmap override?**

* hashCode() and equals() methods.

**27. What is the difference between Enumeration and Iterator?**

* Introduction-IIterator interface is introduced from JDK 1.2 where as Enumeration interface is there from JDK 1.0.
* remove() method - This is the main difference between Enumeration and Iterator interface. Enumeration only traverses the Collection object. You can’t do any modifications to Collection while traversing the Collection using Enumeration. Where as Iterator interface allows us to remove an element while traversing the Collection object. Iterator has remove() method which is not there in the Enumeration interface. Below is the list of Enumeration and Iterator methods.
* c) Iterator - hasNext(),next() and remove() Enumeration - hasMoreElements(), nextElement() and (Not Available)
* Legacy Interface - Enumeration is a legacy interface used to traverse only the legacy classes like Vector, HashTable and Stack. Where as Iterator is not a legacy code which is used to traverse most of the classes in the collection framework. For example, ArrayList, LinkedList, HashSet, LinkedHashSet, TreeSet, HashMap, LinkedHashMap, TreeMap etc.

**28. Collections class and Arrays class**

* Array does not have methods (no API) such as the ones provided by Collection classes.

Collection framework classes either use array underneath or use more complex data structure. When an array is simply...an array.

* Arrays can store primitives

Collections can not store primitives (although they can store the primitive wrapper classes, such as Integer etc)

* Arrays - Avoid using them unless you have to

Collections - Use them in preference to arrays

* Arrays are ultimately the only way of storing a group of primitives/references in one object, but they are the most basic option. Although arrays may give you some speed advantages, unless you need super-fast code.
* Collections are preferred because they have so much convenience.