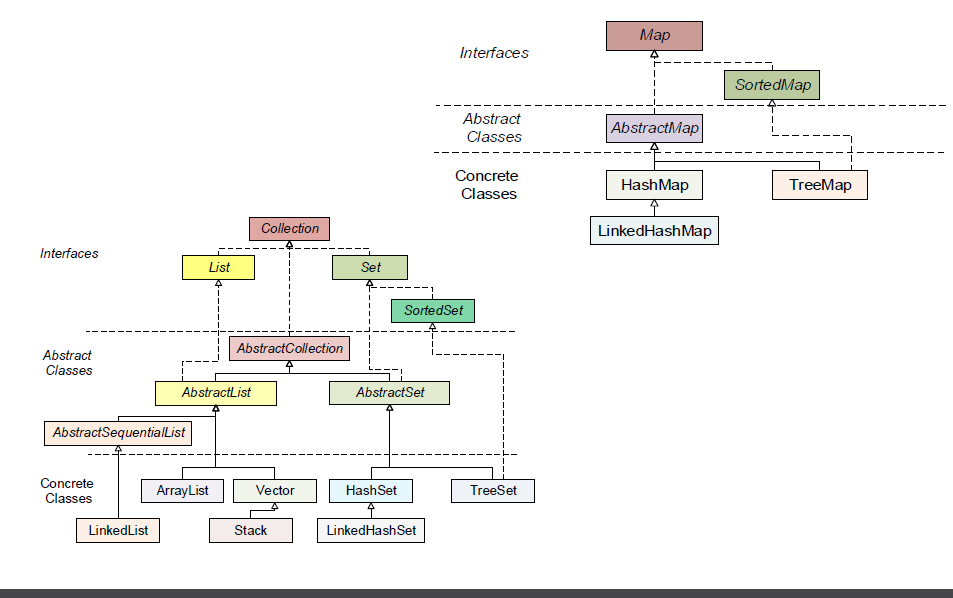
**------------------------------------------------------------------------------**

**Collections Q & A**

**------------------------------------------------------------------------------**

**1. Draw Collections Framework Class Diagram**

****

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

**HashMap:**

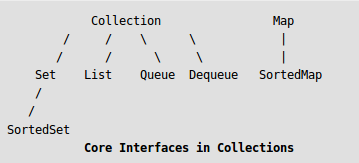
Java HashMap class implements the map interface by using a hashtable. It inherits AbstractMap class and implements Map interface.

The important points about Java HashMap class are:

* A HashMap contains values based on the key.
* It contains only unique elements.
* It may have one null key and multiple null values.
* It maintains no order.

Map:

The java.util.Map interface represents a mapping between a key and a value. The Map interface is not a subtype of the Collection interface. Therefore it behaves a bit different from the rest of the collection types.



A Map cannot contain duplicate keys and each key can map to at most one value. Some implementations allow null key and null value (HashMap and LinkedHashMap) but some do not (TreeMap).

The order of a map depends on specific implementations, e.g TreeMap and LinkedHashMap have predictable order, while HashMap does not.

Exampled class that implements this interface is HashMap, TreeMap and LinkedHashMap.

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

**Differences between HashMap and a Hashtable:**

1) Hashtable is synchronized (i.e. methods defined inside Hashtable), whereas HashMap is not. If you want to make a HashMap thread-safe, use Collections.synchronizedMap(map) or ConcurrentHashMap class.

2) Hashtable does not allow null keys or values. HashMap allows one null key (other null keys will simply overwrite first null key) and any number of null values.

3) Hashtable is legacy class and was not part of the initial Java Collections Framework (later it was included in JDK 1.2). HashMap is part of Collections since it’s birth. Also note that Hashtable extends the Dictionary class, which as the Javadocs state, is obsolete and has been replaced by the Map interface in newer JDK versions.

4) Iterator in the HashMap is fail-fast and throw ConcurrentModificationException if any other Thread modifies the map structurally by adding or removing any element except Iterator’s own remove() method. But this is not a guaranteed behavior and will be done by JVM on best effort. The enumerator for the Hashtable is not fail-fast.

5) Finally, Map fixes a minor deficiency in the Hashtable interface. Hashtable has a method called “contains()” (along with “containsValue()” and “containsKey()“), which returns true if the Hashtable contains a given value. Given its name, you may expect this method to return true if the Hashtable contained a given key, because the key is the primary access mechanism for a Hashtable. The Map interface eliminates this source of confusion by removing this method to and has only “containsValue()” and “containsKey()“.

**Can we make hashmap synchronized?**

HashMap is a non-synchronized collection class. If we need to perform thread-safe operations on it then we must need to synchronize it explicitly.

For example we have a HashMap<Integer, String> it is having integer keys and String type values. In order to synchronize it we are using Collections.synchronizedMap(hashmap) it returns a thread-safe map backed up by the specified HashMap.

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

ArrayList and Vector both implements List interface and maintains insertion order.

But there are many differences between ArrayList and Vector classes that are given below.

|  |  |
| --- | --- |
| ArrayList | Vector |
| 1) ArrayList is not synchronized. | Vector is synchronized. |
| 2) ArrayList increments 50% of current array size if number of element exceeds from its capacity. | Vector increments 100% means doubles the array size if total number of element exceeds than its capacity. |
| 3) ArrayList is not a legacy class, it is introduced in JDK 1.2. | Vector is a legacy class. |
| 4) ArrayList is fast because it is non-synchronized. | Vector is slow because it is synchronized i.e. in multithreading environment, it will hold the other threads in runnable or non-runnable state until current thread releases the lock of object. |

**5. What is an Iterator?**

Iterator enables you to cycle through a collection, obtaining or removing elements.

**6. List vs Set vs Map. Purposes and definitions.**

List, Set and Map are the interfaces which implements Collection interface. Below are the difference between List Set and Map.

List Vs Set Vs Map

1) Duplicity: List allows duplicate elements. Any number of duplicate elements can be inserted into the list without affecting the same existing values and their indexes.

Set doesn’t allow duplicates. Set and all of the classes which implements Set interface should have unique elements.

Map stored the elements as key & value pair. Map doesn’t allow duplicate keys while it allows duplicate values.

2) Null values: List allows any number of null values.

Set allows single null value at most.

Map can have single null key at most and any number of null values.

3) Order: List and all of its implementation classes maintains the insertion order.

Set doesn’t maintain any order; still few of its classes sort the elements in an order such as LinkedHashSet maintains the elements in insertion order.

Similar to Set Map also doesn’t stores the elements in an order, however few of its classes does the same. For e.g. TreeMap sorts the map in the ascending order of keys and LinkedHashMap sorts the elements in the insertion order, the order in which the elements got added to the LinkedHashMap.

4) Commonly used classes:

List: ArrayList, LinkedList etc.

Set: HashSet, LinkedHashSet, TreeSet, SortedSet etc.

Map: HashMap, TreeMap, WeakHashMap, LinkedHashMap, IdentityHashMap etc.

When to use List, Set and Map in Java?

1) If you do not want to have duplicate values in the database then Set should be your first choice as all of its classes do not allow duplicates.

2) If there is a need of frequent search operations based on the index values then List (ArrayList) is a better choice.

3) If there is a need of maintaining the insertion order then also the List is a preferred collection interface.

4) If the requirement is to have the key & value mappings in the database then Map is your best bet.

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

Pros of ArrayList:

1. ArrayList uses internally an array for internal storage. That makes it particularly fast for random access

Cons of ArrayList:

1. 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.

2. 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:

1. LinkedList follows a different approach. It's more efficient in adding or deleting elements in the beggining or middle of the collection.

2. 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:

1. 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)).

**8. TreeSet vs LinkedHashSet**

1. Implementation

LinkedHashSet uses Hash table and linked list. It maintains a doubly-linked list running through all of its entries, thus maintaining the order of insertion.

TreeSet is a NavigableSet implementation based on a TreeMap.

2. Ordering

LinkedHashSet maintains the order in which the elements are inserted.

TreeSet sorts the elements in its natural ordering and ascending. You can provide a comparator to sort the elements stored in the TreeMap

3. Null values as Entries

LinkedHashSet allows Null elements to be added.

Null is not allowed in a TreeSet unless you specify a comparator that can handle a Null.

4. Performance

LinkedHashSet can be a little slower as they maintain a linked list which causes some performance overhead. But we never saw a major performance difference between HashSet and LinkedHashSet.

TreeSet are the slowest in comparision with the other two, due to the sorting done.

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

The contract between equals() and hashCode() is:

1) If two objects are equal, then they must have the same hash code.

2) If two objects have the same hash code, they may or may not be equal.

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

Advantages of using ArrayList over arrays.

1) You can define ArrayList as re-sizable array. Size of the ArrayList is not fixed. ArrayList can grow and shrink dynamically.

2) Elements can be inserted at or deleted from a particular position.

3) 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() ).

4) If generics are not used, ArrayList can hold any type of objects.

5) 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.

6) You can traverse an ArrayList in both the directions – forward and backward using ListIterator.

7) ArrayList can hold multiple null elements.

8) ArrayList can hold duplicate elements.

**11. Principle of storing data in a hashtable**

In hashing, large keys are converted into small keys by using hash functions. Thevalues are then stored in a data structure called hash table. The idea of hashing is to distribute entries (key/value pairs) uniformly across an array. Each element is assigned a key (converted key).

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Hashtable** | **HashMap** | **ConcurrentHashMap** |
| **1. Thread-safe** | **Yes**(synchronized keyword in all methods to provide thread safety) | **No**(only suitable for single threaded environment) | **Yes**(use java.concurrent.ReenterantLock to avoid concurrent access) |
| **2. Performance** | slow, any operation blocks whole Map | so a new operation has to wait to finish of previous one. | Fast , as no locks are used, Faster than Hashtable , as locks are applied on different segments and any two operations across different segments are parallel. |
| **3. null** | No | Yes | No |
| **4. Since** | Yes | No , Introduced in JDK 1.2 | No , added in JDK 1.5 with java.util.concurrent package. |
| **5. Iterators** | Fail Fast | Fail Fast | Fail safe. |

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

haschode() Returns a hash code value for the object. This method is supported for the benefit of hash tables such as those provided by HashMap.

The general contract of hashCode is:

* Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.
* If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.
* 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. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables.

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

Yes, when we override the hashcode() method, it is possible that hash code is not unique.we can always just do a "return 1" in the overriden hashcode() method in the class. But that violates the basic principle of having a unique hashcode to differentiate between objects.

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

Two unequal objects can have equal objects, so we can put two elements with equal has code to one hash map. Since HashMap use a linked list to store in bucket, value object will be stored in next node of linked list

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

java.util.ConcurrentModificationException is a very common exception when working with java collection classes. Java Collection classes are fail-fast, which means if the Collection will be changed while some thread is traversing over it using iterator, the iterator.next() will throwConcurrentModificationException

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

Using ListIterator we can iterate in both the directions, in forward direction as well as backward direction.

difference b/w Iterator and ListIterator:

1) Iterator is used for traversing List and Set both.

We can use ListIterator to traverse List only, we cannot traverse Set using ListIterator.

2) We can traverse in only forward direction using Iterator.

Using ListIterator, we can traverse a List in both the directions (forward and Backward).

3) We cannot obtain indexes while using Iterator

We can obtain indexes at any point of time while traversing a list using ListIterator. The methods nextIndex() and previousIndex() are used for this purpose.

4) We cannot add element to collection while traversing it using Iterator, it throws ConcurrentModificationException when you try to do it.

We can add element at any point of time while traversing a list using ListIterator.

5) We cannot replace the existing element value when using Iterator.

By using set(E e) method of ListIterator we can replace the last element returned by next() or previous() methods.

6) Methods of Iterator:

hasNext()

next()

remove()

**Methods of ListIterator:**

add(E e)

hasNext()

hasPrevious()

next()

nextIndex()

previous()

previousIndex()

remove()

set(E e)

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

A *collection* (also known as a *container*) is a single object representing a group of objects (such as the familiar Vector class). A collections API is a unified framework for representing and manipulating collections, allowing them to be manipulated independent of the details of their representation.

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

We can access the elements of a collection using for-each, Iterator, ListIterator and EnumerationIterator

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

**Stack** – Represents the collection of elements in Last In First Out order.   
Operations includes testing null stack, finding the top element in the stack, removal of top most element and adding elements on the top of the stack.  
  
**Queue** - Represents the collection of elements in First In First Out order.  
Operations include testing null queue, finding the next element, removal of elements and inserting the elements from the queue.   
Insertion of elements is at the end of the queue  
Deletion of elements is from the beginning of the queue.

**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?**

LinkedList allows for fast insertions. The LinkedList is implemented using a doubly linked list; an insertion requires only the updating of the links at the point of insertion.

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

Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.

* HashSet stores the elements by using a mechanism called **hashing.**
* HashSet contains unique elements only.

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

Yes. By using constructor we can limit the initial capacity of the vector

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

We need to override equals() and hashcode() methods of a class whose objects we want to use as Key in a hashmap... This is required because hashmap uses these 2 methods to retrieve the stored values

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

Major difference between Enumeration and iterator is Iterator has a remove() method while Enumeration doesn't. Enumeration acts as Read-only interface, because it has the methods only to traverse and fetch the objects, where as by using Iterator we can manipulate the objects like adding and removing the objects from collection e.g. Arraylist.

Also Iterator is more secure and safe as compared to Enumeration because it does not allow other thread to modify the collection object while some thread is iterating over it and throws ConcurrentModificationException. This is by far most important fact for me for deciding between Iterator vs Enumeration in Java.

**28. Collections class and Arrays class**

Difference between Collections class and Arrays class

|  |  |  |
| --- | --- | --- |
|  | **Array** | **Collection** |
| 1 | Arrays are fixed in size and hence once we created an array we are not allowed to increase or decrease the size based on our requirement. | Collections are grow-able in nature and hence based on our requirement we can increase or decrease the size. |
| 2 | Arrays can hold both primitives as well as objects. | Collections can hold only objects but not primitive. |
| 3 | Performance point of view arrays faster than collection | Performance point of view collections are slower than array |
| 4 | Arrays can hold only homogeneous elements. | Collections can hold both homogeneous and heterogeneous elements. |
| 5 | Memory point of view arrays are not recommended to use. | Memory point of view collections are recommended to use. |
| 6 | For any requirement, there is no ready method available. | For every requirement ready made method support is available. |