|  |  |
| --- | --- |
| Map interface: | 1. Map interface is a special type of collection which is used to store key-value pairs. 2. It does not extend Collection interface for this reason 3. Main classes implementing Map interface are:**HashMap, Hashtable, EnumMap, IdentityHashMap, LinkedHashMap and Properties.** |
| What are IdentityHashMap and WeakHashMap? | **IdentityHashMap** is similar to HashMap except that it uses **reference equality** when comparing elements.  Does not use equals() method when comparing objects. |
| **WeakHashMap?** | WeakHashMap is an implementation of the Map interface that stores only weak references to its keys. Storing only weak references allows a key-value pair to be garbage collected when its key is no longer referenced outside of the WeakHashMap. This class is intended primarily for use with key objects whose equals methods test for object identity using the == operator. |
| How to design a good key for hashmap? | 1. he most important constraint is you must be able to fetch the value object back in future. 2. a good key object**must provide same hashCode() again and again** |
| Difference between HashMap and HashTable? | Hashtable is synchronized, whereas HashMap is not.  Hashtable does not allow null keys or values. HashMap allows one null key and any number of null values.  The third significant difference between HashMap vs Hashtable is that Iterator in the HashMap is a fail-fast iterator while the enumerator for the Hashtable is not. |
| Difference between Vector and ArrayList? | All the methods of Vector is synchronized. But, the methods of ArrayList is not synchronized.  By default, Vector doubles the size of its array when it is re-sized internally. But, ArrayList increases by half of its size when it is re-sized |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Iterator in collection**

|  |  |
| --- | --- |
| Difference between Iterator and Enumeration? | Iterators allow the caller to remove elements from the underlying collection during the iteration with its remove() method. You can not add/remove elements from a collection when using enumerator.  Enumeration is available in legacy classes i.e Vector/Stack etc. whereas Iterator is available in all modern collection classes.  Another minor difference is that Iterator has improved method names e.g. Enumeration.hasMoreElement() has become Iterator.hasNext(), Enumeration.nextElement() has become Iterator.next() etc. |
| Difference between Iterator and ListIterator? | We can use Iterator to traverse Set and List and also Map type of Objects. But List Iterator can be used to traverse for List type Objects, but not for Set type of Objects.  By using Iterator we can retrieve the elements from Collection Object in forward direction only whereas List Iterator, which allows you to traverse in either directions using hasPrevious() and previous() methods.  ListIterator allows you modify the list using add() remove() methods. Using Iterator you can not add, only remove the elements. |
| iterator fail-fast | Fail-fast Iterators fail as soon as they realized that structure of Collection has been changed since iteration has begun.  Structural changes means **adding, removing or updating** any element from collection while one thread is Iterating over that collection.  Fail-fast behavior is implemented by keeping a modification count and if iteration thread realizes the change in modification count it throws **ConcurrentModificationException**.  They use original collection to traverse over the elements of the collection.  These iterators don’t require extra memory.  Ex : Iterators returned by ArrayList, Vector, HashMap.  If you remove an element via Iterator remove() method, exception will not be thrown. However, in case of removing via a particular collection remove() method, ConcurrentModificationException will be thrown |
| **Fail Safe Iterator** | Fail-safe iterators allow modifications of a collection while iterating over it.  These iterators don’t throw any Exception if a collection is modified while iterating over it.  They use copy of original collection to traverse over the elements of the collection.  These iterators require extra memory for cloning of collection. Ex :, CopyOnWriteArrayList  in case of ConcurrentHashMap, it does not operate on a separate copy although it is not fail-fast. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

GENERAL QUESTIONS:

|  |  |
| --- | --- |
| How to make a collection thread safe? | Collections.synchronizedList(list);  Collections.synchronizedSet(set);  Collections.synchronizedMap(map); |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |