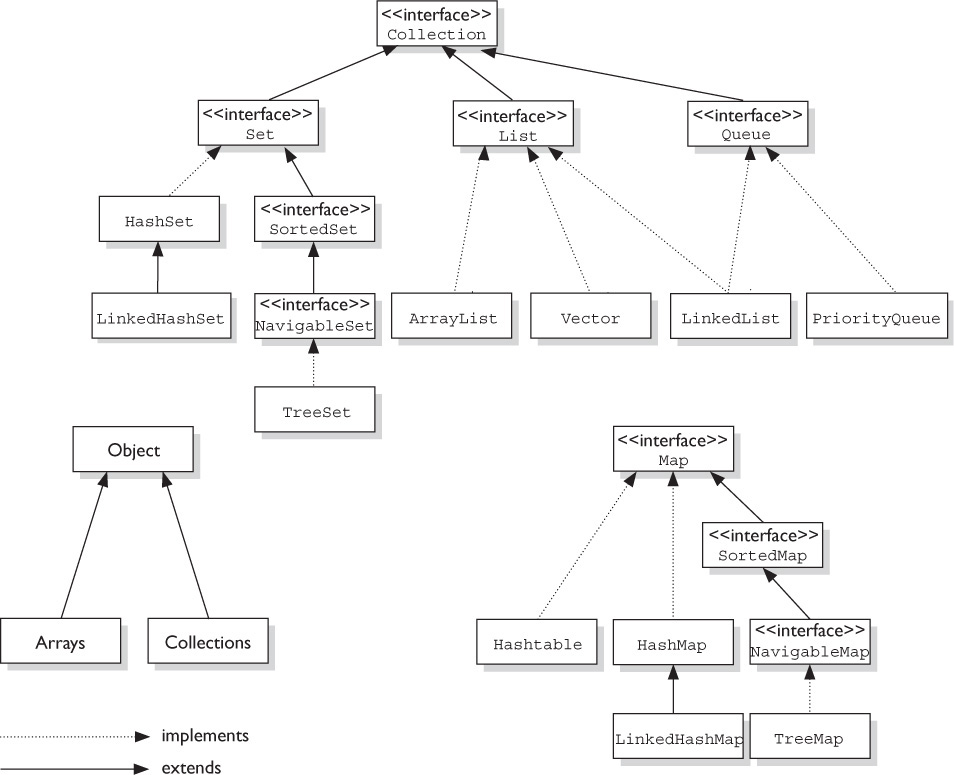
**Collections:**

**9 Major interfaces:**

* Collection
* Set
* Map
* List
* Queue
* NavigableSet
* NavigableMap
* SortableSet
* SortableMap

**Implementations:**



**Note: HashMap, Hashtable,LinkedHashMap,TreeMap are not subclasses of Collection**

**List Implementations:**

* ArrayList
* LinkedList
* Vector (Obsolete)
* Stack (Obsolete)
* CopyOnWriteArrayList

**Set Implementations:**

* HashSet
* LinkedHashSet
* TreeSet
* CopyOnWriteArraySet
* ConcurrentSkipListSet

**Map Implementations:**

* HashTable (Obsolete)
* HashMap
* LinkedHashMap
* TreeMap
* Properties
* IdentityHashMap
* EnumMap
* WeakHashMap
* ConcurrentHashMap
* ConcurrentSkipListMap

**Queue Implementations:**

* PriorityQueue
* ArrayDeque
* DelayQueue
* SynchronousQueue
* LinkedTransferQueue
* PriorityBlockingQueue
* LinkedBlockingDeque
* ArrayBlockingQueue
* ConcurrentLinkedQueue

<http://www.karambelkar.info/2012/06/java-collections-uml-class-diagrams.html>

**List (Index)**

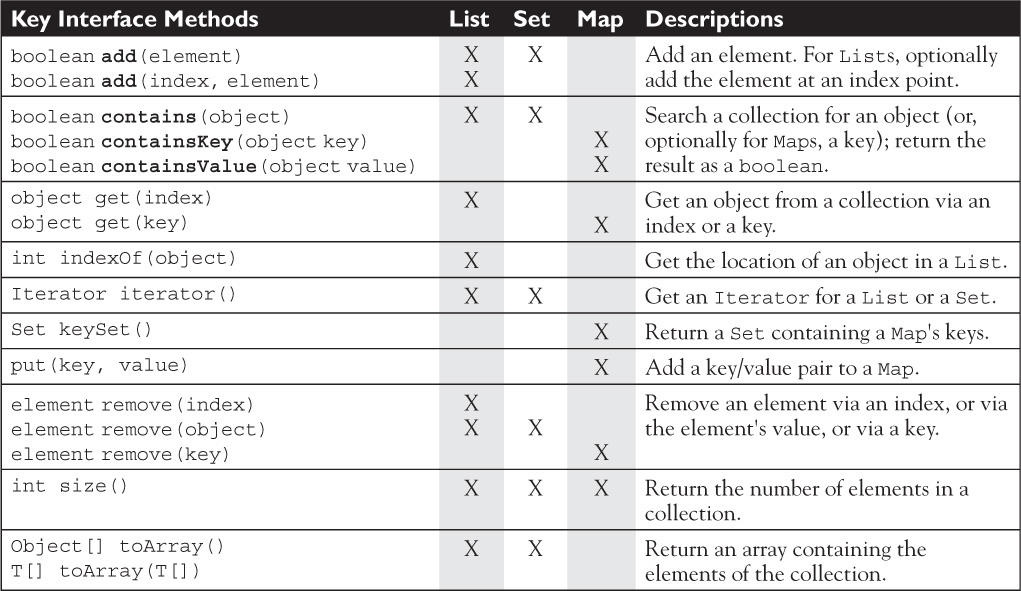
* ArrayList
  + Growable array
  + Ordered, Not sorted
  + Fast iteration, random access, not many inserts/deletes
  + Sort and ArrayList using Collections.sort()
* Vector
  + Same as ArrayList, synchronized methods
  + Vector and Hashtable – part of Java 1.2 original collections
  + Obsolete, if thread safety is needed, use methods from Collections
    - Use ArrayList but
      * Use Collections.synchronizedList() method on a Arraylist to get a synchronized version of the list
      * Use thread-safe variant – CopyOnWriteArrayList
* LinkedList
  + Ordered, doubly linked
  + Implements both list and queue
  + Fast insertions,deletions; slower iterations
  + Add/remove elements from beginning or end. Implement a stack / queue
* Stack
  + LIFO
  + Deque is more preferred
* CopyOnWriteArrayList
  + Concurrent package (Fail safe iterator)

**Set (Uniqueness)**

* HashSet
  + Unsorted, unordered
  + No duplicates, order is not a preference
  + Performance depends on hashCode implementation
* LinkedHashSet
  + Ordered, unsorted
  + Order in which they are inserted
* TreeSet
  + Sorted, ordered (natural order)
  + Custom order by using Comparator

**Map (Unique Identifiers, Key-value pairs)**

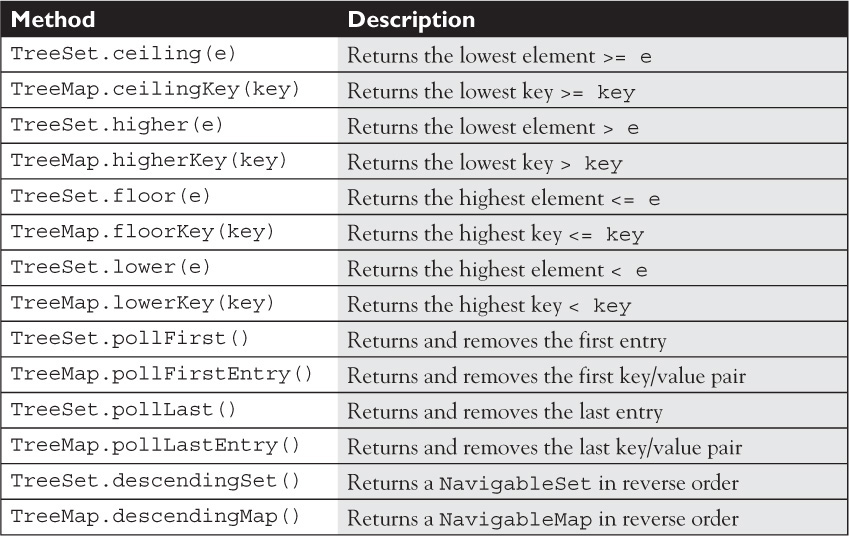
* HashMap
  + Unsorted, unordered
  + Performance based on hashCode implementation
  + Allows one null key and multiple null values
* Hashtable
  + Synchronized version of HashMap
  + No null key or value
  + We could use Collections.syncronizedMap or ConcurrentHashMap
* LinkedHashMap
  + Ordered HashMap; insertion order
  + Faster iteration; slower inserts/deletes
* TreeMap
  + Sorted, ordered (Natural order)
  + Custom order by Comparator
  + One drawback is that all elements added into the set needs to be mutually comparable (example, cannot add 3 strings and one Integer into a TreeSet since they are not comparable and are of different types)



**Queue (List of To-dos)**

* PriorityQueue
  + Basic queue functionality can be achieved using LinkedList
  + Priority in, priority out queue
  + Natural order or Comparator
  + Sorted by to-do order
  + Priority 1 is highest
  + Peek(), poll(), offer() methods
    - Offer adds elements to the queue
    - Peek returns the highest priority item but does not remove from queue
    - Poll returns the highest priority item and removes it from the queue

**Searching in TreeSet and TreeMap**



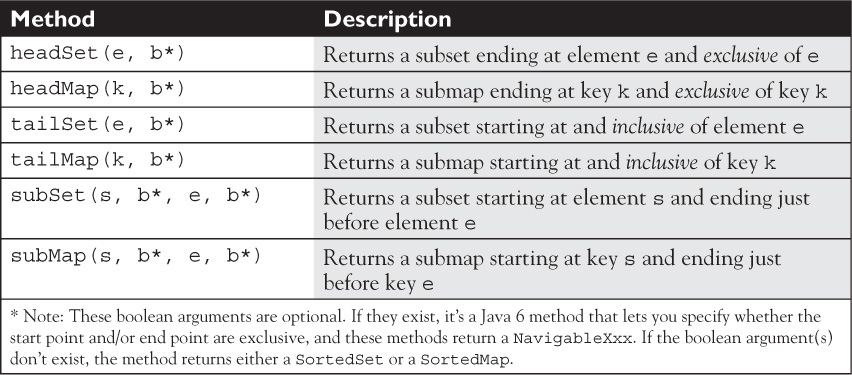
**Polling**

Although the idea of polling isn’t new to Java 6 (as you’ll see in a minute, PriorityQueue had a poll() method before Java 6), it is new to TreeSet and TreeMap. The idea of polling is that we want both to retrieve and remove an element from either the beginning or the end of a collection. In the case of TreeSet, pollFirst() returns and removes the first entry in the set, and pollLast() returns and removes the last. Similarly, TreeMap now provides pollFirstEntry() and pollLastEntry() to retrieve and remove key/value pairs.

**Descending Order**

Also new to Java 6 for TreeSet and TreeMap are methods that return a collection in the reverse order of the collection on which the method was invoked. The important methods for the exam are TreeSet.descendingSet() and TreeMap.descendingMap().

As with the navigation-oriented methods we just discussed, we can see a lot of parallels between the TreeSet and the TreeMap methods. The headSet()/headMap() methods create a subset that starts at the beginning of the original collection and ends at the point specified by the method’s argument. The tailSet()/tailMap() methods create a subset that starts at the point specified by the method’s argument and goes to the end of the original collection. Finally, the subSet()/subMap() methods allow you to specify both the start and end points for the subset collection you’re creating.



**Fail-fast vs Fail-safe iterators**

What is Concurrent Modification ?

When one or more thread is iterating over the collection, in between, one thread changes the structure of the collection (either adding the element to the collection or by deleting the element in the collection or by updating the value at particular position in the collection) is known as Concurrent Modification

Difference between Fail Fast iterator and Fail Safe iterator

Fail fast Iterator

Fail fast iterator while iterating through the collection , instantly throws Concurrent Modification Exception if there is structural modification of the collection . Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

Fail-fast iterator can throw ConcurrentModificationException in two scenarios :

1. Single Threaded Environment

After the creation of the iterator , structure is modified at any time by any method other than iterator's own remove method.

1. Multiple Threaded Environment

If one thread is modifying the structure of the collection while other thread is iterating over it .

According to Oracle docs , the fail-fast behavior of an iterator cannot be guaranteed as it is, generally speaking, impossible to make any hard guarantees in the presence of unsynchronized concurrent modification. Fail-fast iterators throw ConcurrentModificationException on a best-effort basis. Therefore, it would be wrong to write a program that depended on this exception for its correctness: the fail-fast behavior of iterators should be used only to detect bugs.

Interviewer : How Fail Fast Iterator come to know that the internal structure is modified ?

Iterator read internal data structure (object array) directly . The internal data structure(i.e object array) should not be modified while iterating through the collection. To ensure this it maintains an internal flag "mods" .Iterator checks the "mods" flag whenever it gets the next value (using hasNext() method and next() method). Value of mods flag changes whenever there is an structural modification. Thus indicating iterator to throw ConcurrentModificationException.

Fail Safe Iterator :

Fail Safe Iterator makes copy of the internal data structure (object array) and iterates over the copied data structure.Any structural modification done to the iterator affects the copied data structure. So , original data structure remains structurally unchanged .Hence , no ConcurrentModificationException throws by the fail safe iterator.

Two issues associated with Fail Safe Iterator are :

1. Overhead of maintaining the copied data structure i.e memory.

2. Fail safe iterator does not guarantee that the data being read is the data currently in the original data structure.

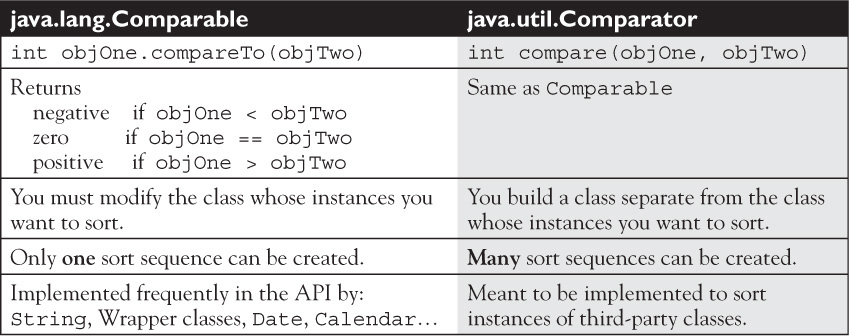
According to Oracle docs , fail safe iterator is ordinarily too costly, but may be more efficient than alternatives when traversal operations vastly outnumber mutations, and is useful when you cannot or don’t want to synchronize traversals, yet need to preclude interference among concurrent threads. The "snapshot" style iterator method uses a reference to the state of the array at the point that the iterator was created. This array never changes during the lifetime of the iterator, so interference is impossible and the iterator is guaranteed not to throw ConcurrentModificationException.The iterator will not reflect additions, removals, or changes to the list since the iterator was created. Element-changing operations on iterators themselves (remove(), set(), and add()) are not supported. These methods throw UnsupportedOperationException.

Difference between Fail Fast Iterator and Fail Safe Iterator

|  |  |  |
| --- | --- | --- |
|  | **Fail Fast Iterator** | **Fail Safe Iterator** |
| Throw ConcurrentModification Exception | Yes | No |
| Clone object | No | Yes |
| Memory Overhead | No | Yes |
| Examples | HashMap,Vector,ArrayList,HashSet | CopyOnWriteArrayList, ConcurrentHashMap |

**Concurrency**

**Comparable, Comparator**



**Collections class and its methods**

Collections.sort()

Collections.reverseOrder()

Collections.binarySearch() //need to sort before search; if we use comparator to sort, we need to use comparator to search as well

**Collection class and its methods**

**Arrays class and its methods**

Arrays.sort()

Arrays.asList() // List.toArray(). Set.toArray() as converse

Arrays.binarySearch() //need to sort before search; if we use comparator to sort, we need to use comparator to search as well

