**various interfaces used in the Collection framework**

**1. Iterable Interface**: collection framework's primary interface. it is extended by the collection interface. As a result, all interfaces and classes implement this interface by default. This interface's main purpose is to provide an iterator for the collections. this interface only has one abstract method, the iterator.

**2. Collection Interface**: This interface covers all of the basic methods that every collection has, such as adding data to the collection, removing data from the collection, clearing data

**3. List Interface**: List interface extends the Collection interface, and it is an ordered collection of objects. It contains duplicate elements. It also allows random access of elements.

**4. Queue Interface**: it follows the FIFO (First In First Out).

**5. Deque Interface**: It differs slightly from the queue data structure.  Deque, also known as a double-ended queue, is a data structure in which elements can be added and removed from both ends.

**6. Set Interface**: A set is an unordered group of objects in which duplicate values cannot be kept.

**7. Sorted Set Interface**: The sorted set interface is an extension of the set interface that is used to manage sorted data. TreeSet is the class that implements this interface.

### differences between array and collection

* Arrays are always of fixed size, i.e., a user can not increase or decrease the length of the array according to their requirement or at runtime, but In Collection, size can be changed dynamically as per need.
* Arrays can only store homogeneous or similar type objects, but in Collection, heterogeneous objects can be stored.
* Arrays cannot provide the ?ready-made? methods for user requirements as sorting, searching, etc. but Collection includes readymade methods to use.

### difference between ArrayList and Vector?

|  |  |  |
| --- | --- | --- |
| **No.** | **ArrayList** | **Vector** |
| 1) | ArrayList is not synchronized. | Vector is synchronized. |
| 2) | ArrayList is not a legacy class. | Vector is a legacy class. |
| 3) | ArrayList increases its size by 50% of the array size. | Vector increases its size by doubling the array size. |
| 4) | ArrayList is not ?thread-safe? as it is not synchronized. | Vector list is ?thread-safe? as it?s every method is synchronized. |

### difference between ArrayList and LinkedList?

|  |  |  |
| --- | --- | --- |
| **No.** | **ArrayList** | **LinkedList** |
| 1) | ArrayList uses a dynamic array. | LinkedList uses a doubly linked list. |
| 2) | ArrayList is not efficient for manipulation because too much is required. | LinkedList is efficient for manipulation. |
| 3) | ArrayList is better to store and fetch data. | LinkedList is better to manipulate data. |
| 4) | ArrayList provides random access. | LinkedList does not provide random access. |
| 5) | ArrayList takes less memory overhead as it stores only object | LinkedList takes more memory overhead, as it stores the object as well as the address of that object. |

### difference between Iterator and ListIterator?

|  |  |  |
| --- | --- | --- |
| **No.** | **Iterator** | **ListIterator** |
| 1) | The Iterator traverses the elements in the forward direction only. | ListIterator traverses the elements in backward and forward directions both. |
| 2) | The Iterator can be used in List, Set, and Queue. | ListIterator can be used in List only. |
| 3) | The Iterator can only perform remove operation while traversing the collection. | ListIterator can perform ?add,? ?remove,? and ?set? operation while traversing the collection. |

### What is the difference between Iterator and Enumeration?

|  |  |  |
| --- | --- | --- |
| **No.** | **Iterator** | **Enumeration** |
| 1) | The Iterator can traverse legacy and non-legacy elements. | Enumeration can traverse only legacy elements. |
| 2) | The Iterator is fail-fast. | Enumeration is not fail-fast. |
| 3) | The Iterator is slower than Enumeration. | Enumeration is faster than Iterator. |
| 4) | The Iterator can perform remove operation while traversing the collection. | The Enumeration can perform only traverse operation on the collection. |

### difference between List and Set?

* The List can contain duplicate elements whereas Set includes unique items.
* The List is an ordered collection which maintains the insertion order whereas Set is an unordered collection which does not preserve the insertion order.
* The List interface can allow n number of null values whereas Set interface only allows a single null value.

difference between HashSet and TreeSet?

* HashSet maintains no order whereas TreeSet maintains ascending order.
* HashSet impended by hash table whereas TreeSet implemented by a Tree structure.
* HashSet performs faster than TreeSet.
* HashSet is backed by HashMap whereas TreeSet is backed by TreeMap.

### difference between Set and Map?

* Set contains values only whereas Map contains key and values both.
* Set contains unique values whereas Map can contain unique Keys with duplicate values.
* Set holds a single number of null value whereas Map can include a single null key with n number of null values.

### difference between HashMap and Hashtable?

|  |  |  |
| --- | --- | --- |
| **No.** | **HashMap** | **Hashtable** |
| 1) | HashMap is not synchronized. | Hashtable is synchronized. |
| 2) | HashMap can contain one null key and multiple null values. | Hashtable cannot contain any null key or null value. |
| 3) | HashMap is not ?thread-safe,? so it is useful for non-threaded applications. | Hashtable is thread-safe, and it can be shared between various threads. |
| 4) | 4) HashMap inherits the AbstractMap class | Hashtable inherits the Dictionary class. |

### difference between Collection and Collections?

* The Collection is an interface whereas Collections is a class.
* The Collection interface provides the standard functionality of data structure to List, Set, and Queue. However, Collections class is to sort and synchronize the collection elements.
* The Collection interface provides the methods that can be used for data structure whereas Collections class provides the static methods which can be used for various operation on a collection.

### difference between Comparable and Comparator?

|  |  |  |
| --- | --- | --- |
| **No.** | **Comparable** | **Comparator** |
| 1) | Comparable provides only one sort of sequence. | The Comparator provides multiple sorts of sequences. |
| 2) | It provides one method named compareTo(). | It provides one method named compare(). |
| 3) | It is found in java.lang package. | It is located in java.util package. |
| 4) | If we implement the Comparable interface, The actual class is modified. | The actual class is not changed. |

### BlockingQueue?

BlockingQueue is an interface which extends the Queue interface. It provides concurrency in the operations like retrieval, insertion, deletion. While retrieval of any element, it waits for the queue to be non-empty. While storing the elements, it waits for the available space. BlockingQueue cannot contain null elements, and implementation of BlockingQueue is thread-safe.

### generic collection

* If we use the generic class, we don't need typecasting.
* It is type-safe and checked at compile time

### default size of load factor in hashing based collection

The default size of load factor is **0.75**. The default capacity is computed as initial capacity \* load factor. For example, 16 \* 0.75 = 12. So, 12 is the default capacity of Map.

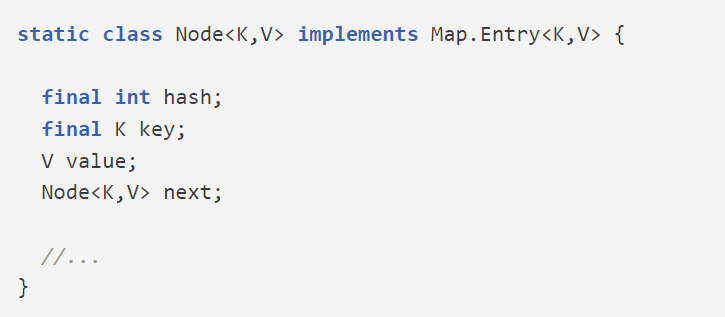
### What do you understand by fail-fast?

The Iterator in java which immediately throws ConcurrentmodificationException, if any structural modification occurs in, is called as a Fail-fast iterator

### Internal Working of HashMap in Java

it works on the hashing mechanism

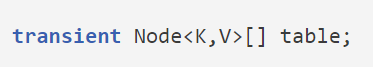
Internally, the key-value pairs are stored as an instance of Map.entry instances represented by Node. Each instance of Node class contains the supplied, key, value, the hash of the key object, and a reference to the next Node, if any, as in the LinkedList



The generated hash of the Key object is also stored to avoid calculating hash every time during comparisons, to improve the overall performance.

## Internal Implementation of HashMap

 It internally maintains an array, also called a **“bucket array”**. The size of the bucket array is determined by the initial capacity of the HashMap, the default is 16.



Each index position in the array is a bucket that can hold multiple Node objects using a [*LinkedList*](https://howtodoinjava.com/java/collections/java-linkedlist-class/).

it is possible that multiple keys may produce the hash that maps them into a single bucket. This is why, the *Map* entries are stored as [LinkedList](https://howtodoinjava.com/java/collections/java-linkedlist-class/).

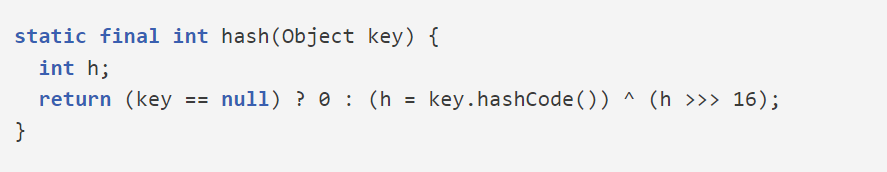
But when entries in a single bucket reach a threshold (*TREEIFY\_THRESHOLD*, default value 8) then *Map* converts the bucket’s internal structure from the linked list to a *RedBlackTree* ([JEP 180](https://openjdk.org/jeps/180)). All *Entry* instances are converted to *TreeNode* instances.

## How Hashing is used to Locate Buckets?

Hashing, is **a way to assign a unique code for any object after applying a formula**/algorithm to its properties. A true hash function **should return the same hash code every time the function is applied to the same or equal objects**. In other words, two equal objects must consistently produce the same hash code.

In Java, all objects inherit a default implementation of [hashCode()](https://howtodoinjava.com/java/basics/java-hashcode-equals-methods/) function defined in Object class. It produces the hash code by typically converting the internal address of the object into an integer, thus producing different hash codes for all different objects.

Map class internally applies another round of hashing function on the key’s hashcode() to make them reasonably distributed.



## HashMap.put() Operation

he put() API, internally, first calculates the initial hash using the key.hashcode() method and then calculates the final hash using the hash() method, which is ultimately used to compute an index in the array or bucket location.

## HashMap.get() Operation

Similar to the put() API, the logic to find the bucket location is similar to the get() API. Once the bucket is located using the final hash value, the first node at the index location is checked.