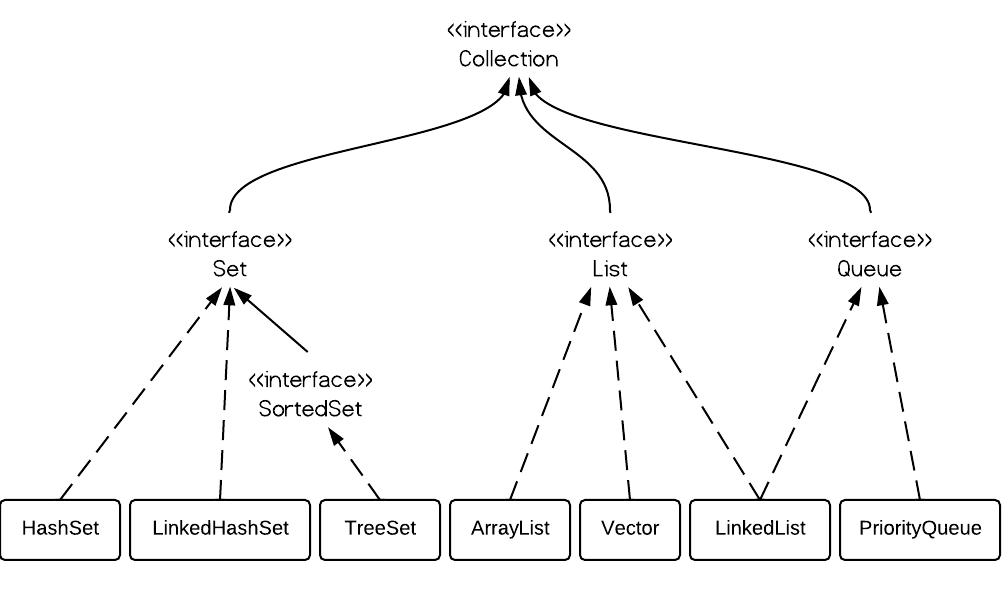
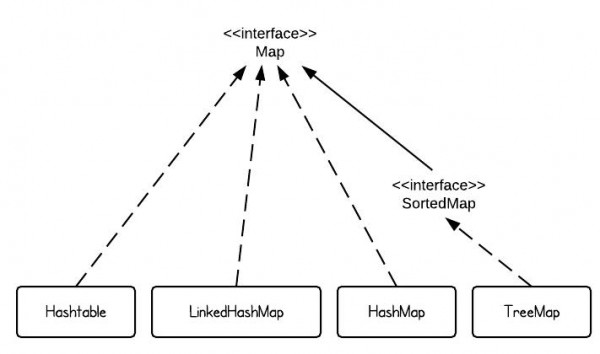
**C:\Users\StayCalm\Desktop\collection-hierarchy.pngCollection Hierarchy**

****

**Map Interface Hierarchy:**

****

**Map**

* HashMap
* LinkedHashMap
* HashTable

**SortedMap**

* TreeMap

**More Interfaces:**

1. NavigableMap
2. NavigableSet
3. Iterator
4. ListIterator
5. Enumeration
6. Comparator
7. Comparable

**More Classes:**

1. Properties
2. Collections
3. Arrays
4. ConcurrentHashMap
5. CopyOnWriteArrayList
6. CopyOnWriteArraySet
7. WeakHasMap

**What is Collection?**

Collection framework is most used topic in java development. JDK 10 has also added few more methods.

A collection allows a group of object to be treated as a single unit. These objects can be stored, retrieved and manipulate as elements of collections.

Collection is used to perform some operation in the object. Collection has lots of predefined optimized methods that given us to perform our operations.

It also improve the performance, suppose we got some data from database and now again need to get new data by some kind of filtration like ordering, then we don’t need to go to database again, we can do searching sorting via collection.

**Benefits of Java Collections Framework**

Java Collections framework have following benefits:

* One common task on multiple objects. Suppose we have to serialize 50 objects then we have to call **writeObject()** 50 times but with the help of **Collection** we can call **writeObject()** only once and serialize the collection object.
* Detached Record set.
* Return any number of values from single function.
* Passing any number of argument on one function.
* Pooling
* Duplicacy Removing
* Sorting
* Searching
* Updation
* Serialization
* Cloning
* **Reduced Development Effort** – It comes with almost all common types of collections and useful methods to iterate and manipulate the data. So we can concentrate more on business logic rather than designing our collection APIs.
* **Increased Quality** – Using core collection classes that are well tested increases our program quality rather than using any home developed data structure.
* **Reusability and Interoperability**
* **Reduce effort** – to learn any new API if we use core collection API classes.

Collection is a first framework. It has created standard to perform operations. If we take example of photo frame then we can notice that frame designer never fix, what person photo should place there. He just make standard frame of size x and y. In Java we make standard via abstraction and collection does the same, it’s tasks varies user-to-user.

**#.** Some collections allow duplicate element and other do not. Some are ordered and others unordered.

**#.** All general purpose collection implementation classes should provide two standard constructors.

1. clasName()
2. className(Collection c): Suppose we have old collection and we are creating new collection and wants to give all old collection elements to new collection, then we can create collection with second constructor.

**#.** This collection frame is provided in **java.util** package and comprises of **two** main parts.

1. The core interfaces that allow collection to be manipulated independently of their implementation.
2. A small set of implementation that is concrete classes that are specific implementation of the core interfaces.

**#.** Differences on the basis of interfaces. It mean when to use which interfaces.

|  |  |  |
| --- | --- | --- |
| **Interfaces** | **Description** | **Concrete Classes** |
| **Collection** | A basic interface that defines the normal operation that allow a collection of object to be maintained or handled as a single unit. |  |
| **Set**: Unique elements and no order. | A set of unique elements. | Hash Set  LinkedHashSet |
| **SortedSet:** Unique elements but in sorted order. | A set in which elements are stored in some sorted order. | Tree Set |
| **List**: Duplicate elements but in insertion order. | A sequence of elements that not need to be unique. | ArrayList  VectorList  LinkedList |
| **Map**: Unique Keys, wants to store data with unique key so duplicate data will recognize. | A basic interface that defines operation for maintaining mappings of key to values. | HashMap  LinkedHashMap  HashTable |
| **SortedMap**: Unique keys but in sorted order. | This interface maintain their mapping sorted in key order. | Treemap |

**Methods of Collection Interface:**

There are many methods declared in the Collection interface. They are as follows:

|  |  |
| --- | --- |
| **Method** | **Description** |
| public boolean add(Object element) | If elements successfully added it returns **true** else **false**. This method is of collection so all child **List** and **Set** will have this method. |
| public boolean addAll(Collection c) | It is used to insert the specified collection elements in the invoking collection. To add one collection values to other collection. |
| public boolean remove(Object element) | It is used to delete an element from this collection. |
| public boolean removeAll(Collection c) | It is used to delete all the elements of specified collection from the invoking collection. |
| public boolean retainAll(Collection c) | It is used to delete all the elements of invoking collection except the specified collection. |
| public int size() | Return the total number of elements in the collection. |
| public void clear() | Removes the total no of element from the collection. |
| public boolean contains(Object element) | It is used to search an element. Check if any object is in collection |
| public boolean containsAll(Collection c) | It is used to search the specified collection in this collection. |
| public Iterator iterator() | Returns an iterator. Retrieve element one by one from any collection. |
| public Object[] toArray() | Converts collection into array. Returns all elements in the form of array. |
| public boolean isEmpty() | Checks if collection is empty. |
| public boolean equals(Object element) | Matches two collection. |
| public int hashCode() | Returns the hashcode number for collection. |

**Iterator Interface**

A collection framework provides an Iterator which allows sequential access to the elements of a collection. An iterator can be obtained by calling the following method of the collection interface.

* **public Iterator iterator()**
* **public boolean hasNext():** Returns true if there are more elements otherwise returns false.
* **public Object next():** Moves the iterator to the next element and returns the current element. Throws **NoSuchElementException,** if there is not a next element.
* **Public Object remove():** Removes the element that was returned by last to the next() invoking this method result in an **IllegalStateException,** if the next() method has not yet be called.   
  (It only removes those elements that came from next() method).

**#.** Collection classes like ArrayList, Vector creates nested classes named “**Itr**” that implements **Iterator** interface. “**Itr**” class have methods like hasNext(), next()and etc. Collection classes have iterator() method that returns the object of Iterator. Same happens with ListIterator interface.

**ListIterator Interface**

ListIterator interface extends **Iterator** interface. This interface is a child of Iterator interface which allows programmer to traverse a list in inner direction and make modification to the underline list.

* public boolean hasNext()
* public boolean hasPrevious()
* public Object next()
* public Object previous()
* public int nextIndex(): It returns the index of next element in the list or size of list if there are no more elements.

**Modification methods of ListIterator:**

* **public boolean add(Object o):** Difference between Iterator add() and ListIterator add() is that **it inserts the new Object immediately** before the element which would be returned by the next() method.
* **public Object remove()**
* **public void set(Object o):** Replace the last elements in the list retrieved by a next() or previous() operation.

**Enumeration Interface (From 1.2)**

The Enumeration interface defines the method by which we can obtain (one at time) the elements in a collection of an object. The functionality of this interface is duplicated by the iterator interface.

Enumeration’s method public boolean hashMoreElemen() is same as Iterator’s hasNext().

public Object nextElement() is same as Iterator’s next() Method.

**Difference between Iterator and Enumeration:**

|  |  |
| --- | --- |
| **Enumeration** | **Iterator** |
| It is a legacy interface which does not come under the collection framework. | Whereas Iterator interface present in the collection interface. |
| All the legacy classes like Stack, Vector can use Enumeration. | The classes under the collection framework like ArrayList, LinkedList can use Iterator interface. |
| Enumeration an average performs above 50% faster than Iterator for sequential access of the collection elements.  Suppose two threads T1, T2 are sharing a common collection elements. If T1 updated the data then T2 won’t get the updated data because it does not check for any updation. | While Iteration process there are checks where the elements are undergoing, updating. So this is the reason for slowness.  But Iteration checks for every updation that’s why it is to slow. |
| Enumeration does not have remove() method. | Iterator has remove() method. |
| Enumeration is not Fail-Fast. | Iterator is Fail-Fast. |

**What is the meaning of Fail-Fast?**

Suppose two threads T1, T2 are sharing a common collection elements. While Iteration process if data updated and T1 sleeps then it fails and throw **ConcurrentModificationException.**

**List Interface**

Can put duplicate value also follows insertion orders.

**#.** All Collection methods came to List. List class has also overloaded many number of methods.

An **Ordered Collection** also known as sequence elements can be inserted or access by their position in the List using a 0 based index. A List may contain duplicate values, the position of an element can be changed as element are inserted or deleted from the List.

**Methods:**

* public void **add** (Integer index, Object go
* public boolean **addOn** (Integer index Collection c)
* public Object **get** (Integer index)
* public Object **remove** (Integer index)
* public Object **set** (Integer index, Object go)
* ListIterator **listIterator**()

**IQ. Why Set class does not have listIterator() method but List Class have?**

* **Collection**
  + **List**
    - ListIterator listIterator() method.
  + **Set**
    - Does not have listIterator() method.

Because Set class does not maintain orders. Elements come randomly so how we will know if we are going forward or backward so no sense to give this method in Set Class.

**ArrayList**

**IQ. How an object like ArrayList holds data?**

Collection in java is simply a data structure. One Type follow binary, then one follow Tree or Linked List and so on.

Vector and ArrayList both follows the same data structure to hold unlimited data. Vector and ArrayList is like brother with little differences. Vector came in 1.2 and ArrayList copied all works.

* There is no capacity method in ArrayList().
* ArrayList default size in 10 and it grows 50% of initial capacity when needed “**Initial capacity/2\*2**”
* Vector class **double** the Initial value when needed.

**#.** ArrayList, Vector classes are implemented using dynamically resizable arrays providing fast random access and list traversion very much like using an ordinary arrays.

**#.** ArrayList support dynamic arrays that can grow as needed that is ArrayList can dynamically increases or decreases in size.

**#.** ArrayList are created with an initial size when the size exceeded the collection is automatically enlarged.

If elements successfully added it returns **true** else **false**. This method is of collection so all child **List** and **Set** will have this method.

**#. Some important points about ArrayList class:**

1. ArrayList is not synchronized.
2. ArrayList supports dynamic array which can grow as needed.
3. Size of ArrayList can be dynamically increased (via ensuseCapasity() ) or decreased ( trimSize() ).
4. ArrayList are created with initial size of 10.
5. ArrayList can contain duplicate elements. ArrayList maintains insertion orders of the elements.
6. ArrayList is not synchronized collection hence it is not suitable to be used between multiple threads concurrently. If you want to be use ArrayList like data-structure in multi-threaded environment, then you need to either use new **copyOnWriteArrayList**() or use **Collections**.**synchronizedList**() to create a synchronized list.
7. Former is part of concurrent collection package and much more suitable than the second one, but only useful when there are readers and only few writers. Since a new copy of ArrayList is created every time a write happens. It can be overkill if used in a write heavy environment. Second option is strictly synchronized collection, much like **Vector** or **Hastable,** butit is not scalable because once number of thread increased drastically contention became a huge issue.
8. **CopyOnWriteArrayList**()is recommended for concurrent multi-threading environment as it is optimized for multiple concurrent read and creates copy for the write operation. This was added in JDK 1.5. It’s part of java.util.Concurrent package, along with ConcurrentHashmap.
9. When ArrayList gets full it creates another array and uses **System.arrayCopy**() to copy all elements from one array to another array. This is where insertion takes a lot of time.
10. Iterator and ListIterator of java ArrayList are fail-fast. It means if ArrayList is structurally modified at any time after the Iterator is created in any way except through their Iterator’s add() methods or remove() methods, the iterator will throw a **ConCurrentModificationException**. Thus in the case of concurrent modification, the Iterators fails quickly and clearly, that’s why it is called fail-fast.
11. ConcurrentModificationException is not guaranteed and only thrown at best effort.
12. If we are creating synchronized list it’s recommended to create while creating instance of underlying ArrayList to prevent accidental non-synchronized access to the list. It means don’t first create non-synchronized ArrayList then convert it on synchronized. We must create it synchronized at first place.
13. An application can increase the capacity of an ArrayList instance before adding a large number of elements using the ensureCapasity() method operation. This may reduce amount of incremental reallocation due to incremental filing of ArrayList.
14. The size(), isEmpty, set(), iterator() and iteratorList() operation run in constant time because ArrayList is based on Array but adding or removing on element is costly as compared to LinkedList.
15. ArrayList class is enhanced in JDK1.5 to support Generics which added extra type-safety on ArrayList. It is recommended to use generics version of ArrayList to ensure that your ArrayList contains only specified type of element and avoid any ClassCastException.
16. If we set ArrayList reference (variable) to null in Java. All the elements inside ArrayList becomes eligible to garbage collection in Java. Provided there are no more strong reference exists for those object.
17. Always use **isEmpty ()** method to check if ArrayList is empty or not, instead of using **size () == 0** check. Former one is much more readable.   
     if( listOfElements.isEmpty() ) { S.O.P(“Start Processing”); }  
     if( listOrders.size() ) { S.O.P(“Start Processing”); }

**Q. What if we make non-generic ArrayList.**

If we make non-generic ArrayList so we can put different-different type of object but while iterating we have to downcast from Object to Our class and here down-casting will be fail and it will throw ClassCastException.

**Difference between ArrayList and Vector Class?**

**Synchronization:** ArrayList is non-synchronized which means multiple threads can work on ArrayList at the same time. For e.g. if one thread is performing an add operation on ArrayList, there can be another thread performing remove operation on ArrayList at the same time in a multi-threaded environment.

While Vector is synchronized, this means if one thread is working on vector, no other thread can get a hold of it. Unlike ArrayList, only one thread can perform an operation on vector at a time.

**Resize:** Both ArrayList and Vector can grow and shrink dynamically to maintain the optimal use of storage. However the way they resize is different. ArrayList grow by half of its size when resize. In the other hand Vector doubles the size of itself by default when grows.

**Performance:** ArrayList gives better performance as it is non-synchronized. Vector operation ogives poor performance as they are thread-safe. The thread which works on Vector gets a lock on it which makes other thread wait till the lock is released.

**Fail-Fast:** If the collection (ArrayList, Vector etc.) gets structurally modified by any means, except add or remove methods of iterator, after creation of iterator then the iterator will throw **ConcurrentModificationException**. Structural modification refers to the addition or deletion of elements from the collection.

As per **Vector Javadoc** the Enumeration returned by Vector is not fail-fast. On the other side the iterator on ListIterator retuned by ArrayList are fail-fast.

**Q. Who belongs to Collection framework?**

The Vector was not the part of Collection framework. It has been included in collections later. It can be considered as **Legacy code.**  There is nothing about Vector which list collection cannot do. Therefore Vector should be avoided. If there is a need of thread-safe operation make ArrayList synchronized as use **copyOnWriteArrayList**() method which is a thread-safe variant of ArrayList.

**Size Increment:** In ArrayLIst we cannot define the increment size but Vector we can define the increment size.

**Similarities:** There are following similarities between those closes which are as follows.

* Both Vector and ArrayList use grow able array data structure.
* The Iterator and ListIterator returned by these classes (Vector and ArrayList) are fail-fast.
* They both are ordered collection classes as they maintain the elements insertion order.
* Vector & ArrayList both allows duplicate and null values. They both grow shrink automatically when overflow and deletion happens.

**Q. When to use ArrayList and when to use Vector?**

It totally depends on the requirement. If there is a need to perform thread-safe operation then Vector is your best as it ensures that only one thread access the collection at a time.

**Performance:** Synchronized operations consumes more time compared to non-synchronized ones so if there is no need for thread safe operation, ArrayList is better choice as performance will be improved because of the concurrent processed.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Collection** | **Ordering** | **Random Access** | **Key-Value** | **Duplicate Elements** | **Null Element** | **Thread Safety** |
| ArrayList | Yes | Yes | No | Yes | Yes | No |
| LinkedList | Yes | No | No | Yes | Yes | No |
| HashSet | No | No | No | No | Yes | No |
| TreeSet | Yes | No | No | No | No | No |
| HashMap | No | Yes | Yes | No | Yes | No |
| TreeMap | Yes | Yes | Yes | No | No | No |
| Vector | Yes | Yes | No | Yes | Yes | Yes |
| Hashtable | No | Yes | Yes | No | No | Yes |
| Properties | No | Yes | Yes | No | No | Yes |
| Stack | Yes | No | No | Yes | Yes | Yes |
| CopyOnWriteArrayList | Yes | Yes | No | Yes | Yes | Yes |
| ConcurrentHashMap | No | Yes | Yes | No | No | Yes |
| CopyOnWriteArraySet | No | No | No | No | Yes | Yes |