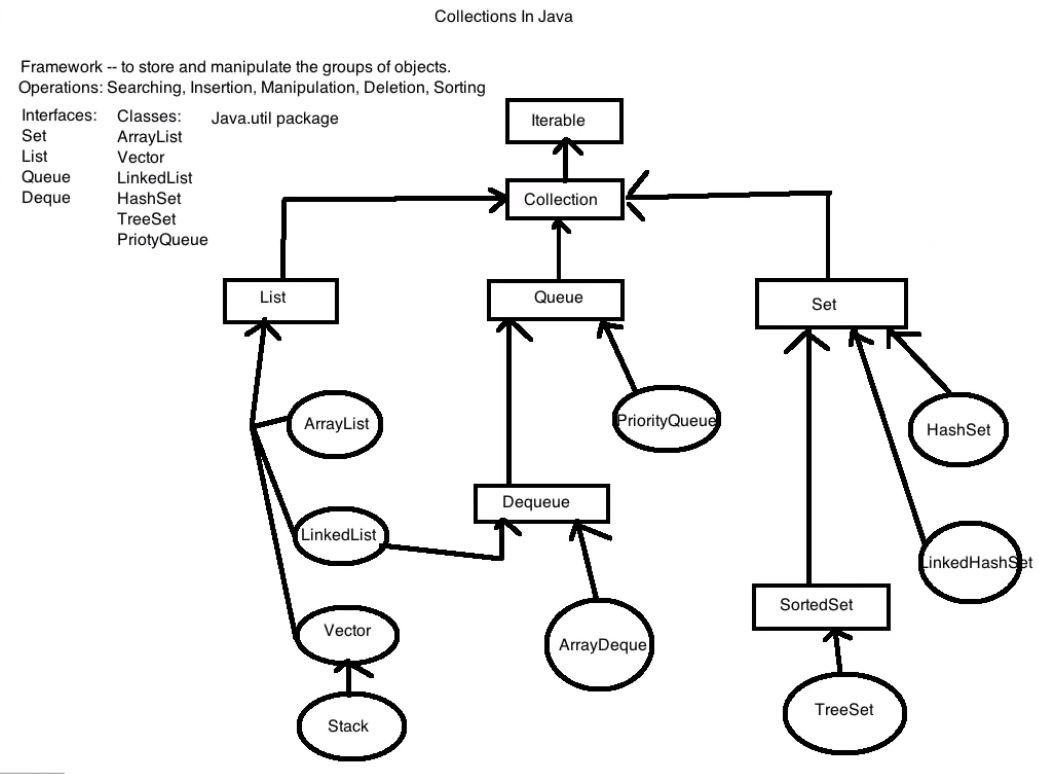
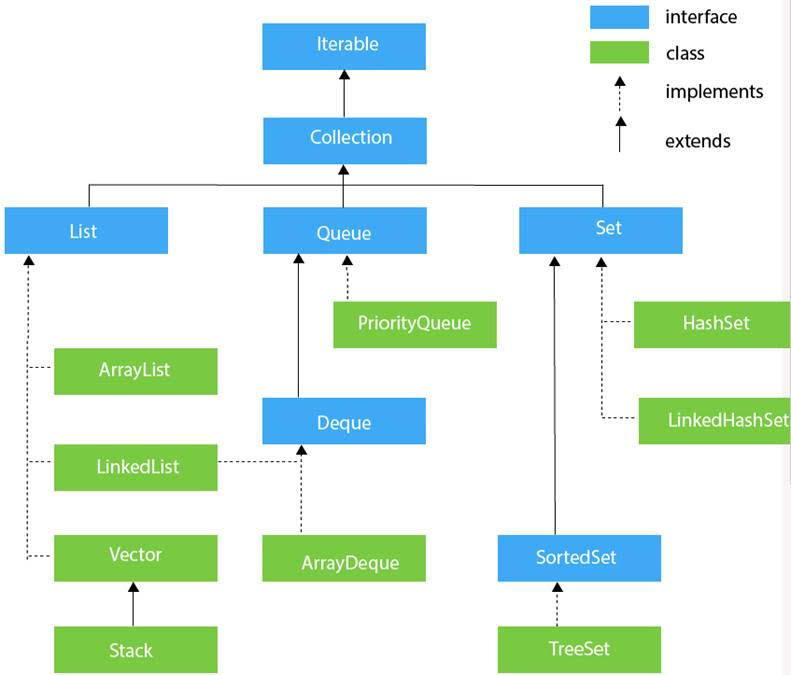
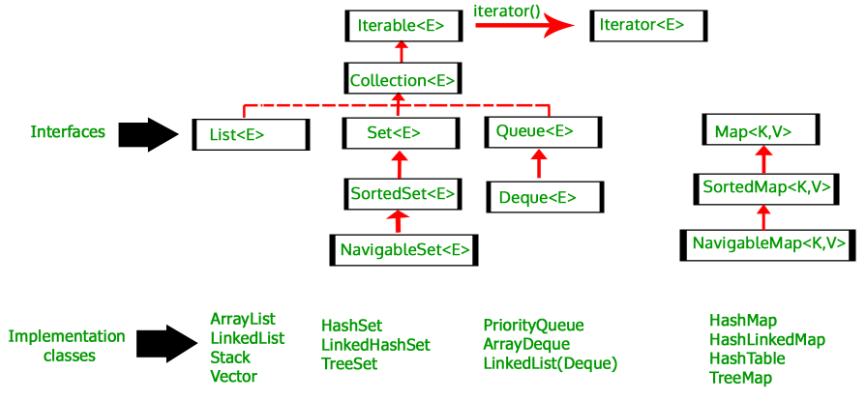
**Collection**

****

* Size of array is fixed, you can’t expand it nor shrink it.
  + int a[] = new int[];
* Due to this limitation, Collection are introduced.
* Collection means group of elements.
* Collection provides dynamic size of array where you can add elements, expand or shrink the size of elements.
* Collection is an interface.
* Collection interface encapsulates number of other interfaces like Set, List etc
* We can not create object of interface. So we need a class for its implementation.
* Collection interface has general methods which are require for any collection object like add, remove, contains, isEmpty etc.

****

****

**List**

* Collection don’t work with indexing. So, to use indexing we need to use List.
* If user wants a collection where duplicate values are allowed & insertion order to be preserved then he should use List.
* List is an interface which extends Collection.
* List came in 1.2 version.
* ArrayList and LinkedList are implementations of List interface.
* We can create object of ArrayList even to work with List (like Collection).
* In List you can have duplicate values.
* To sort the List we use sort method from Collections class.

List<Integer> values = **new** ArrayList<Integer>();

Collections.*sort*(values);

**ArrayList**

* ArrayList is a class which implements Collection indirectly. ArrayList implements List (which is interface) which extends collection.
  + Collection values = new ArrayList();
* ArrayList came in 1.2 version.
* In above example type of values is object (as noting is explicitly specified). So we can add integer, string, object of a class anything in values.
* To define the type of Collection, we use generics.
* The concept of mentioning what type of data you have in your Collection is called ‘generics’.
* Generics were introduced in JDK 1.5
* In generics we can define both primitive and non primitive datatypes.
* To fetch values of Collection, we can use either ‘iterator’ (which is an interface) or ‘for loop’.

ArrayList<Integer> values = **new** ArrayList<Integer>();

values.add(5);

values.add(9);

values.remove(5);

System.out.println(values.size()) // will return the size of ArrayList

System.out.println(value.get(2)) // fetching the value using index

ar1.addAll(ar2) // ar2 will get added to ar1

ar1.removeAll(ar2) // removes all the values of ar2 from ar1

ar1.retainAll(ar2) // Only common values from both ArrayList will remain in ar1

Properties of ArrayList 🡪

* It can contain duplicate values.
* It maintains insertion value.
* It is not synchronized.
* It allows random access to fetch the element because it stores the value on the basis of index.

Fetching the value using for loop,

**for** (**int** i=0; i<ar.size(); i++) {

System.***out***.println(ar.get(i));

}

**Converting a string to ArrayList**

String str1 = "I am doing well";

String[] arStr1 = str1.split(" ");

List<String> ar = **new** ArrayList<String>();

ar = Arrays.asList(arStr1);

**Methods of List / Set Interface 🡪**

* add()
* clear()
* contains()
* isEmpty()
* iterator()
* remove()
* size()

**Iterator**

* Allows user to visit (traverse) the elements in the collection one by one i.e. it allows to retrieve the elements in forward direction but not in backward direction.
* ‘Iterator’ is an interface which belongs to collection framework
* Iterator can be available in all the iterator classes.
* Iterator interface contains three methods,
  + boolean hasNext()
  + next()
  + remove()
* In ArrayList mostly iterator is used where ArrayList’s datatype is user defined like class objects.

**public** **class** MyClass {

**public** **static** **void** main(String[] args) {

Emp E1 = **new** Emp("VJ", 7, "QA"); // Emp class is created in same package

Emp E2 = **new** Emp("Swanand", 10, "Dev");

Emp E3 = **new** Emp("Akshay", 8.5, "Support");

ArrayList<Emp> ar1 = **new** ArrayList<Emp>();

ar1.add(E1);

ar1.add(E2);

ar1.add(E3);

Iterator<Emp> It1= ar1.iterator();

**while** (It1.hasNext()) {

Emp person = It1.next();

System.***out***.println(person.name);

System.***out***.println(person.sal);

System.***out***.println(person.dep);

}

}

}

**List Iterator**

ListIterator<Integer> li1 = a1.listIterator();

|  |  |
| --- | --- |
| **Iterator** | **Listiterator** |
| Iterator is used to traverse list, set and map | Listiterator used to traverse list |
| Iterator traverses only in forward direction | Listiterator traverses in both forward and backward direction |
| While traversing collection using iterator only remove operation can be performed. | While traversing collection using Listiterator add, remove and replace operations can be performed. |
| We can not get index of object when we traverse using iterator | We can get index of object when we traverse using Listiterator using previousIndex(E e) and nextIndex(E e) method |
| We can not get current position of iterator | Listiterator can obtain iterators current position in list |
| We have only 3 methods in iterator  hasNext()  next()  remove() | We have 9 methods in iterator  hasNext()  hasPrevious()  next()  previous()  add(E e)  remove()  nextIndex()  previousIndex()  set() |
|  |  |
|  |  |

**LinkedList**

* Arraylist follows the concept of dynamic array & LinkedList follows the concept of Double linked list.
* LinkedList is very helpful when you want to insert a value in between. In this case LinkedList is much faster than the ArrayList.
* While fetching the values using index (searching), LinkedList becomes slow compared to ArrayList.
* LinkedList came in 1.2 version.
* Linked List is a Data Structure like other data Array, Stacks and Queue.
* Linked List has three major parts Head, Nodes and Tail.
* Every node in Linked List is interconnected with next node using address of the next node.
* Unlike arrays, Linked list do not work with completely reserved sequential collection of memory, instead they choose random block from the heap and store data into them
* Once the use of Linked List is done we can actually dump the memory blocks back into memory heap which makes them reusable.
* Each block has memory address of next node and space for data to be stored. So the last node will replace the memory address with Null value which indicates the end of Linkedlist.
* Using the interconnection between the nodes, linked list behaves like an array.
* Types of Linked list 🡪
  + Singly Linked List
  + Doubly Linked List
  + Circular Linked List
* Singly linked list consist of two fields, address of next node and data assigned to it.
* We can traverse in one single direction in Singly linked list.
* Doubly linked list consist of three fields, address of previous node, data and address of next node. This makes the doubly linked list traverse in both the directions.
* Circular linked list is singly linked list with final address block points to the address of Head node.

**Features of Linked list 🡪**

* Linked list implements Que and Deque interface. So it can also be used as Que, deque or a stack.
* It can contains all the elements including null and duplicate.
* It maintains the insertion order of elements.
* Linked lists are Synchronized.
* Linked list do not implement random access interface. So we can access elements in sequential order only.
* We can use list iterator to iterate the elements through list.

LinkedList<String> ll = **new** LinkedList<String>();

ll.add("VJ");

ll.addFirst("Me"); // Adds as First element

ll.addLast("Do"); // Adds as Last element

System.***out***.println(ll); // Returns the complete list

ll.set(1, "Introduced"); // Replaces element at specified index

System.***out***.println(ll.get(1)); // Returns element of specified index

ll.removeFirst(); // Removes First element

ll.removeLast(); // Removes Last element

ll.remove(1); // Removes element at specified index

**Vector**

* Vector is a type of List. It implements List and almost same as ArrayList.
* Vector increases size by 100% of the existing capacity (10,20,40,…).
* Vector & Stack came in 1.0 version. So these are known as ‘Legacy Classes’.
* ArrayList increases memory by 50% (so saves memory compared to vector).
* ArrayList is fast and vector is slow. So always use ArrayList.
  + vector v = new Vector();

**Set**

* If user wants to store a group of individuals where duplicates are not allowed & no need to preserve insertion order, then he should use Set.
* **Set supports only unique value.**
* **Set can’t be accessed using index.**
* HashSet do not maintain the insertion order.
* LinkedHashSet maintains the insertion order.
* Set & HashSet came in 1.2 version while LinkedHashSet came in 1.4 version.
* To store values in ascending order, use TreeSet class object (instead of HashSet).

Set<Integer> values = **new** HashSet<Integer>();

Set<Integer> values = new TreeSet<Integer>();

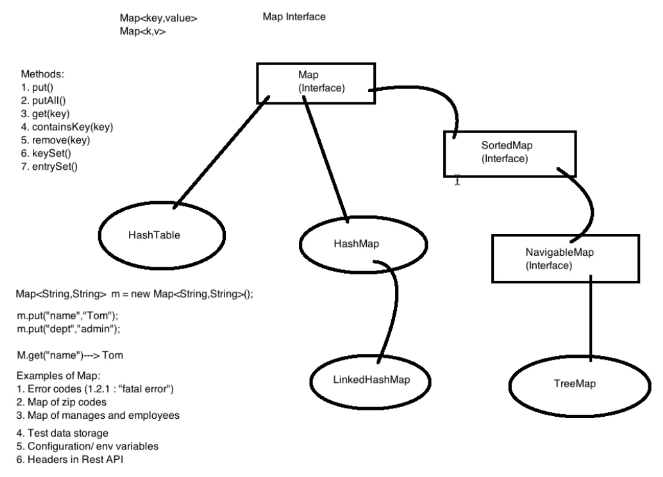
values.add(5);

**Sorted Set**

* If user wants to represent a group of objects where duplicates are not allowed but all objects should be inserted according to some sorting order then we go for Sorted Set.
* Sorted Set came in 1.2 version.

**Map**

* Map interface, which is also a part of java collection framework, doesn’t inherit from Collection interface.
* To achieve key value pair we use map.
* Map interface is implemented using HashMap, Hashtable, & LinkedHashMap classes.
* Hashtable is synchronized and HashMap is not synchronized. So HashMap is faster and mostly used.



**HashMap**

* HashMap is a class which implements Map interface.
* It does not maintain the order in which data was created as it does not store the value in terms of index.
* It may have one null key and multiple null values.
* HashMap is non synchronized (non thread safe) means multiple threads can access it simultaneously. So performance increases.
* If any thread modifies the HashMap (edit some value OR delete any key value pair) and other thread tries to access it simultaneously then ‘Fail Fast’ condition will occur & ‘Concurrent Modification Exception’ will occur.
* To add any key value pair, here we use ‘put’ method (unlike in List & Set we use ‘add’ method).
* To print the value of specific pair, use ‘get()’ method and pass key associated to it.
* In map, we can repeat values but we can’t repeat keys. If we put pair with same key and different value, then it overrides the earlier value.
* To get all the keys in a map use keyset() method.

Map<String, String> map = **new** HashMap<>();

map.put("MyName", "VJ"); // add it in map

map.remove("MyName"); // removes key value pair from map

Set<String> keys = map.keySet();

* putAll() is used to copy all elements from one HashMap to another HashMap.

new\_hash\_map.putAll(hash\_map)

* map.get(“key1”) is used to get its associated value. If user passes a key which is not present in hashmap then it returns null (error do not occur).
* To convert HashMap to set,

System.***out***.println(studentData.entrySet());

Output is 🡪 [UserName1=Password1, UserName2=Password2, UserName3=Password3]

* To print all key values from HashMap

**for** (Entry<String, String> entry : studentData.entrySet()) {

String key = entry.getKey();

String value = entry.getValue();

System.***out***.println(key);

System.***out***.println(value);

}

Hashmap vs Hashtable

1. HashMap is non synchronized. It is not-thread safe and can’t be shared between many threads without proper synchronization code whereas Hashtable is synchronized. It is thread-safe and can be shared with many threads.

2. HashMap allows one null key and multiple null values whereas Hashtable doesn’t allow any null key or value.

3. HashMap is generally preferred over HashTable if thread synchronization is not needed

4. HashTable stores the value based on HashCode of the key object.

Q) Where do we use Collections in Selenium ?

1. List -

- When we use driver.findElements method, it returns List (which is one of the interface of Collection family).

List<WebElement> rows = driver.findElements(By.*xpath*("//table[@class='dataTable']/tbody/tr"));

2. Set -

- When we user driver.getWindowHandles method, it returns Set (which is also one of the interface of Collection family).

Set<String> allWindows = driver.getWindowHandles();

3. HashMap -

- We can use HashMap in the scenario where we have to check access of multiple people with different roles to specific data.

- We create a HashMap with key as Role (like Owner, Contributor, Reader etc) and Value as username and password separated by some special character like ‘-’.

- We can pass the login credentials with key of role name and then splitting the value using ‘-’.

HashMap<String, String> mp = new HashMap<String, String>();

mp.put("owner", "9420289001-rahul004");

we1.sendKeys(mp.get("contributor").split("-")[0]);

we2.sendKeys(mp.get("contributor").split("-")[1]);

Q.) What is the difference between Collection and Collections ?

- Collection is a root level interface of the Java Collection Framework. List, Set and Queue are main sub interfaces of this interface.

- Collections is an utility class in java.util package. It consists of several utility methods which are used to operate on objects of type Collection, for example: Collections.sort(anyCollection);