Collections in Selenium

Table of content

Collections in Java/selenium

Let me put it in a fundamental way :
Collections is nothing but a backpack, guess what the things you can store in a backpack are.

Probably you must have guessed everything that we can store in a backpack, but let list a few of them : 1. Laptop, Mouse, HeadPhones, earPhones, Mouse (computer's and animal too), Book, Notepad, pen, pencil you can write so on.

Almost you can store everything; you cannot store an elephant though. Within the backpack's size, you can store all the things.

Let's discuss Java Collection : Collection is a framework that manipulates only one data type, which is the god data type of Java.

I mean data type of Object class, I hope you know that Object class is the topmost class in java.

> Collection framework mostly used along with a group of Objects rather than Objects.

Description as per Books:

Collections in java is a framework that provides an architecture to store and manipulate the group of objects Collections in java is a framework that provides an architecture to store and manipulate the group of objects

I am not going to discuss everything present in the collection, but the only collection that we use frequently and which are :

List

Set

Map (map is not a collection but mostly used for collections)

List in Java

List is an ordered collection and can contain duplicate elements. You can access any element from its index.

List is more like an array with dynamic length; List is one of the most used Collection types.

> In my language : List is nothing but a library, what do you think about 'How the books are stored in the Library'
>
> In any Library in the world, they will store it in the racks which have some ISBN or some unique number.
>
> If you ask a Librarian to bring a book which is in '12371' rack, now the librarian will go to that rack and fetches the book.
>
> Probably if you ask he may not do, but he will fetch if someone from higher official asks.
>
> The library can hold several number of the same book, isn't it ?, because it is stored in a rack which has unique numbers.

Similar to the library, the list collection also stores the objects based on the index number, and it can also store duplicate objects as the access is going to happen through the index.

List is an interface, and it just provides what should be present if somebody implements the interface, so the list interface will not have any implementations.

ArrayList and LinkedList classes provide implementation to the List interface methods, the methods present in the ArrayList and LinkedList are the same with little changes.

Methods present in List :

There are few frequently used methods in java list, and I would be discussing them:

add(Object o):

This method adds an element/value into the list based on the index. Here the index is calculated automatically based on the number of existing elements.

> As said earlier, the Collection accepts only Object class type, so if you store any other type, those will be upcasted to Object class because Object class is top most class.
>
> You can add a list inside a list as well.

In the below code, String value will be stored as Object type only, and the value will be stored at the last position in the list.

```
obj.add("karthiq")
```

add(int index, Object o):

This add method adds the given value and accepts two parameters; one the index where you want to add the value, two Object O is the value to be added.

> When you add a value at a given position; then the element which existing at that position will be moved to the next position, so every element in the list.

You will face IndexOutOfBoundsException exception if you try to add value to the index which does not exist in the list.

```
obj.add(2, boolean)
```

remove:

Remove method is an overloaded method. There are two remove() methods in List.

First, remove method accepts an index of int type and removes an object at the given position., Second remove() method accepts the value of object type and removes the value by searching the total list one by one till it finds the matching element.

> remove() method which works based on an index is much faster than the remove method which works on the element values.
>
> We have unique indexes, so it is easy to remove an item from an index but removing an element by value involves in comparing all the object from the beginning of the List.

After removing an element, the element after that index will move one step closer to the beginning of the list.

```
obj.remove(int index) // faster
obj.remove(Object o)  // slower
```

set(int index, Object o):

set method updates the element value at a given index; the set method does not return any value.

```
obj.set(4, "karthiQ")
// sets element at index 4 to "karthiQ"
```

int indexOf(Object o):

indexOf method retrieves the index of the first matching object o. If the element is not found in the list, then this method returns the value -1.

indexOf method returns int values

```
obj.indexOf("karthiQ")
```

Object get(int index):

get fetches the value at a given position, returns Object value.

```
obj.get(2); // retrieves the value at position 2
```

int size():

size() method works based on the number of elements in the list.

> There is a counter (variable) inside the list, which will increase by one when we add an element to the list and decrease when we remove an element from the list.

The size() method returns the counter, so we would have the number of elements present in the list as output.

size() method returns int value.

```
obj.size(); // total number of elements in the list
```

boolean contains(Object o):

contains() method checks whether the given element is present in the list or not. If the element is present then contains() method results in true otherwise false

```
obj.contains("karthiQ");
```

clear():

clear() method is used for removing all the elements of the list in one go, this will not delete the list but elements in the list.

```
obj.clear();
```

Complete program for the list operations.

```
package cherchertech;
import java.util.ArrayList;
import java.util.List;
public class OverLoading {

   public static void main(String[] args) {


      List al  = new ArrayList();
```

```
        al.add("chercher tech");
        al.add(true);
        al.add(10);
        al.add(new ArrayList());
        System.out.println("value at index 2 before Adding value at 2 : " +al.get(2));
        al.add(2, 20);
        System.out.println("value at index 2 after Adding value at 2 : " +al.get(2));

        al.remove("chercher tech");  // based on object
        System.out.println("All values in list : " +al);
        al.remove(1); // based on index
        System.out.println("All values in list : " +al);

        al.set(1, "eee");
        System.out.println("Value at index 1 :" +al.get(1));

        System.out.println("index of avengers (none present): "+ al.indexOf("Avenger"));
        System.out.println("get the value at index 0 " +al.get(0));

        System.out.println("Number of elements present in the list : "+al.size());
        System.out.println("does list contains 'eee' : "+al.contains("eee"));

        al.clear();
        System.out.println("Elements present in list after clearing the list :" +al );
    }
}
```

Set in Java

The Set interface extends the Collection interface. It will make sure that an instance of Set contains no duplicate elements.

The subclasses class implements hashcode and equals methods to make sure the uniqueness of objects.

Three concrete classes of Set are HashSet, LinkedHashSet, and TreeSet

> In my Way of telling Set : Set is nothing but a sack, which can contain any objects inside the sack
>
> If you place two identical pebbles inside a sack will you able to find which one you dropped first into the sack ?, heck No.
>
> Because sack doesnot maintain any order, so when you pick the pebble it will lead to the confusion that

> which one you want to access(intention), so due to this issue Set type of collection in java will not allow you put duplicates in the set.

Set stores the values in random order and accessing the elements happens through element Value rather than index.

HashSet, LinkedHashSet, TreeSet are the classes that implement the Set interface.

Methods Present in the Set :

We will discuss the most used methods in the Set type of collection in java.

add(setElement) :

add() method adds an element to the set if it is not already present.

addAll(fromList) :

addAll() method Adds all of the elements in the specified list to the set if they are not already present.

addAll(fromSet) :

Adds all of the elements in the specified set to the set that calls the method if they are not already present.

clear() :

clear() method removes all of the elements present in the set.

clone() :

clone() creates a duplicate copy of the set.

contains(setElement) :

contains() method returns true if the set contains the specified element otherwise false.

containsAll(listToCompare)

containsAll method returns true if the set contains all of the elements in the specified list. The list must be of the same type as the set that calls the method.

containsAll(setToCompare) :

Returns true if the set contains all of the elements in the specified set. The specified set must be of the same type as the original set that calls the method.

equals(set2) :

Compares this set with the specified set and returns true if both sets are equal; otherwise, returns false.

isEmpty() :

isEmpty() method returns true if the set has zero elements, other wise false.

remove(setElement)

Removes the specified element from the set if it is present.

removeAll(listOfElementsToRemove) :

Removes the elements in the specified list from the set if they are present.

removeAll(setOfElementsToRemove)

Removes the elements in the specified set from the original set if they are present.

retainAll(listOfElementsToRetain) :

Retains only the elements in this set that are contained in the specified list.

retainAll(setOfElementsToRetain) :

Retains only the elements in the original set that are contained in the specified set.

size() :

Returns the number of elements in the set

When to use List and Set

We cannot use list and set interchangeably unless it is very required, let's see when to use list and when to use set.

If You want a Collection that does not store duplicate values, then we use a Set based collection.

If You want to frequently access elements operations based on an index value, then we use a List based collection. E.g., ArrayList.

If You want to maintain the insertion order of elements in a collection, then we use a List based collection.

For fast search operation based on a key, value pair, we use a HashMap based collection.

If You want to maintain the elements in sorted order, then we use a TreeSet based collection

Map in Java

Maps are not part of the collection but built based on the collection concepts.

Map is nothing but a key value pair mappings; every in the map has a value, and every value has a key.

Map keys follow Set interface and allow only unique values; Map Values are following List interface and allows duplicates

It is similar to the attendance system, the roll number is Keys, and names are Values. Roll Number is unique, but the names could be duplicates.

Methods present in Maps :

Methods present in Maps :

We will discuss only important methods present in Maps.

clear():

It removes all the key and value pairs from the specified Map.

clone():

It returns a copy of all the mappings of a map and used for cloning them into another map.

containsKey(Object key):

It is a boolean function that returns true or false based on whether the specified key is found on the map.

containsValue(Object Value):

Similar to the containsKey() method, however it looks for the specified value instead of a key.

get(Object key):

It returns the value for the specified key.

isEmpty():

It checks whether the map is empty. If there are no key-value mapping present in the map, then this function returns true else false.

keySet():

It returns the Set of the keys fetched from the map.

put(Key k, Value v):

Inserts key value mapping into the map. Used in the above example.

size():

Returns the size of the map – Number of key-value mappings.

values():

It returns a collection of values of map.

remove(Object key):

It removes the key-value pair for the specified key. Used in the above example.

putAll(Map m):

Copies all the elements of a map to another specified map.

Generics in java

In my Way : Collection is like road, any vehicle can go on the road. Consider that it is peak time and you have heavy traffic(2, 4, 6 wheelers) on the road, now you feel like you want to make all the vehicles to into two-wheelers(I'm assuming that you do not have superpowers), is it possible ? No. If you force to convert, you may have issue* in converting.

Consider that there is a road which allows only two-wheelers, now it has traffic, will you able to convert them into two-wheelers ?, Yes, because we allowed only two-wheelers on the road.

Coming to the Generic, Generic is nothing but the road which accepts everything, Generic provides a way to make it accept only one kind of value.

In the above examples, you might have noticed that few Data types are inside < > (angular braces); that is nothing but the generic.

The below code shows that the ArrayList accepts only the type of String values into the List.

```
Common syntax:
List< Generic > = new ArrayList();
```

```
ArrayList <String> products = new ArrayList();
```

When we did not mention any generic type, then the collection will accept everything, but if we mention some type, then the collection will accept only that type of data.

Why do we need Generic :

Consider in a scenario you want to store boolean values in a list without specifying the generic type, so you got 8 values as boolean and one value as float, and other values as int.

Do you think, we can convert int to boolan, or float to boolean ?, No, you cannot convert.

This is what happens when you donot specify any generic type in the collection, and this throws ClassCastException in java.

But if we make a list to accept only boolean values while accepting if it identifies the int value, then it will not only reject but also gives you an error during compile time itself.

Generics in selenium :

Have you ever tried to work with the findElements() method in selenium, if yes then what is the return type of the findElements method.

findElements in selenium returns List<WebElement>, so now got to know that with/without knowing you have worked with generics.

```
List<WebElement> elements = driver.findElements(By.tagName("button"));
```

I hope you have worked on handling multiple windows, when we have multiple windows we would be using getWindlowHandles()

getWindlowHandles() returns Set<String>, and this is also generic, I hope you got a bit of idea on generics.

Advantages of Generics in Collections :

Generics add re-usability by enabling you to define a class or interface without specifying fixed-parameter data types.

In practical application, this feature enables the creation of some very flexible code. For example, you need to write an interface that takes both integer and string values for the same parameter. How do you handle it?

You might write a single interface with weak data typing, but this creates its own issues and is against best practices.

You could write two versions of the interface. This allows strong data types but creates additional code to be maintained.

Generics allow you to combine the best of both by enabling a single definition of the interface that maintains strong typing.

The desired data type (string or int) is defined at the time of use. This also can help with code optimization.

This flexible yet strong data typing allows you to avoid heavy usage of casting and other processing-intensive method calls.

Iterator in Java

Often you will want to traverse through the elements of a collection, say to display each element.

The better way is to employ an iterator, which is an object that implements either the Iterator or the ListIterator interface.

Iterator enables you to traverse through the elements, add or remove elements. This is unidirectional.
ListIterator is Bi-directional in traversing.

The iterator cursor maintains its position moved by the previous operation unless disturbed.

So when we move the cursor to end in the last operation, then it stays there only unless we change it; in the below example it is the cursor.

```
import java.util.ArrayList;
import java.util.Iterator;
public class SampleIterator {
   public static void main(String args[]){
```

```
        ArrayList products = new ArrayList();
        products.add("Chercher tech");
        products.add("Google");
        products.add("Bing");

        Iterator it = products.iterator();

        while(it.hasNext()) {
          String obj = it.next();
          System.out.println("Value : " +obj);
        }
      }
}
```

List iterator :

An iterator for lists that allows one to traverse the list in either direction, modify the list during iteration, and obtain the iterator's current position in the list.

The iterator position of a ListIterator is never placed at an element; rather, it is placed between two elements in a list. Because of this List, Iterator has the capability to make it traverse in both directions, i.e., forward and backward.

> List iterator provides methods to traverse forward using **next()** and traverses backward using **previous()**

List iterator is only Applicable to List and does not apply for Set.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.ListIterator;
import org.testng.annotations.Test;
public class SampleListIterator {
   @Test
   public void test(){
      ArrayList fruits = new ArrayList();
       fruits.add("Chercher tech");
       fruits.add("Google");
       fruits.add("Bing");

       ListIterator itList = fruits.listIterator();
       while(itList.hasNext()) {

         String obj = itList.next();
```

```
            System.out.println("Value : " +obj);
        }

        System.out.println("*********Reverse the Iteration*********");
        while(itList.hasPrevious()) {
            String objPrev = itList.previous();
            System.out.println("Value : " +objPrev);
        }
    }
}
```

Output of the List Iterator:

```
Value : Chercher tech
Value : Google
Value : Bing
*********Reverse the Iteration*********
Value : Bing
Value : Google
Value : Chercher tech
```

Difference between Iterator and ListIterator :

ListIterator can be used to traverse only a List. But Iterator can be used to traverse List, Set, and Queue.

Iterator traverses the elements in one direction only. ListIterator can traverse the elements in two directions i.e. backward as well as forward directions.

Iterator cannot provide us index of an element in the Data Structure. ListIterator provides us methods like nextIndex() and previousIndex() to get the index of an element during traversal.

Iterator does not allow us to add an element to collection while traversing it. It throws ConcurrentModificationException. ListIterator allows users to add an element at any point of time while traversing a list.

Differences between Collection and Collections

Below are the few difference between the Collection(framework), Collections class.

Collection is an interface in Java. But Collections is a class in Java.

Collection is a base interface. Collections is a utility class in Java.

Collection defines methods that are used for data structures that contain the objects.

Collections defines the methods that are used for operations like access, find etc. on a Collection.

Collection Interface has only abstract methods but Collections consists of only static methods that are used to operate on objects of type Collection.

Arrays vs Collection

There are so many differences but I am highlighting whatever I know, so there is a chance that more than these differences to be present between them.

1. Arrays are fixed in size but Collections are dynamic in size.

With respect to memory arrays are not good to use but with respect to memory Collections are better to use.

With respect to performance; it is better to use arrays but with respect to performance collection are not good to use.

Arrays can hold only homogeneous elements but collections can hold both homogeneous and heterogeneous elements.

Arrays don't have readymade methods but collections have readymade data structures and methods, which makes collections more user-friendly.

Arrays can hold both primitives and wrapper objects but collections can hold only objects.

ClassCasException occurs in Collection but not in Arrays.

We have iterators for going through the collection but we do not have them for arrays.

Java returns Arrays rather than Collection in most of the operations due to performance issues.