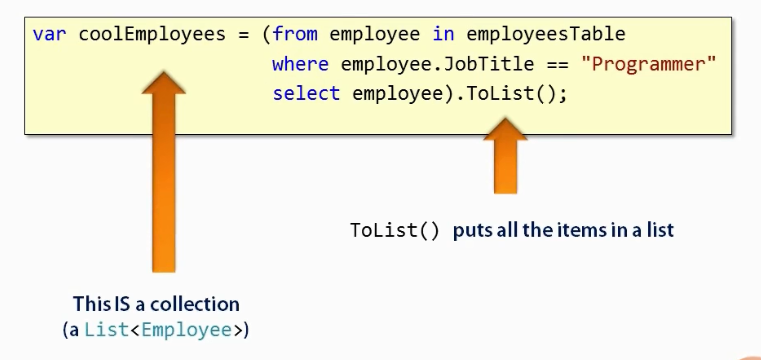
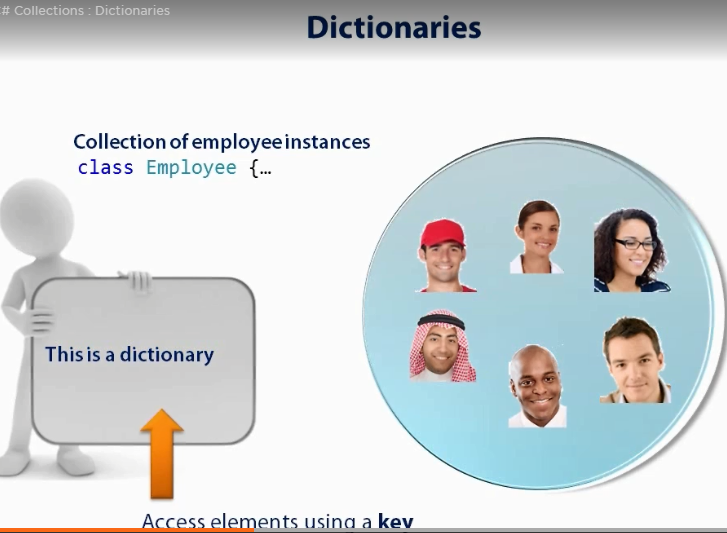
A collection is a group of data.



collection classes are : List, Dictionies and sets

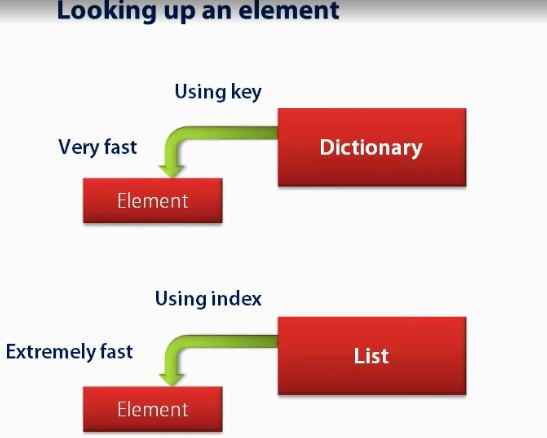




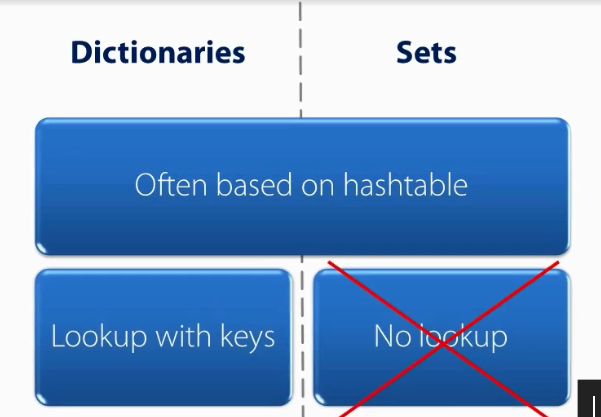
Dictionaries

Staff working at an organization. It works like we will need the information of particular employee using their name.

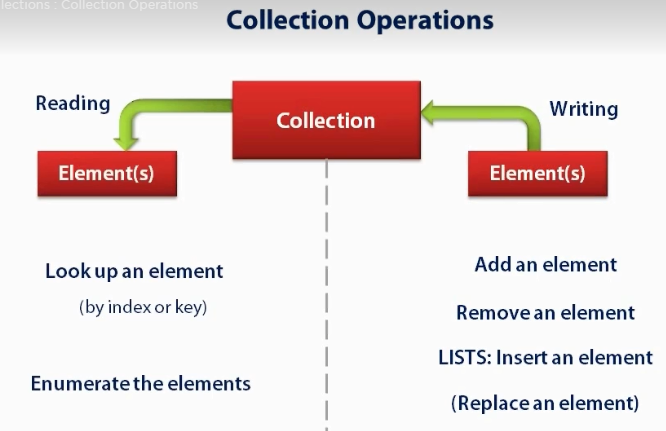


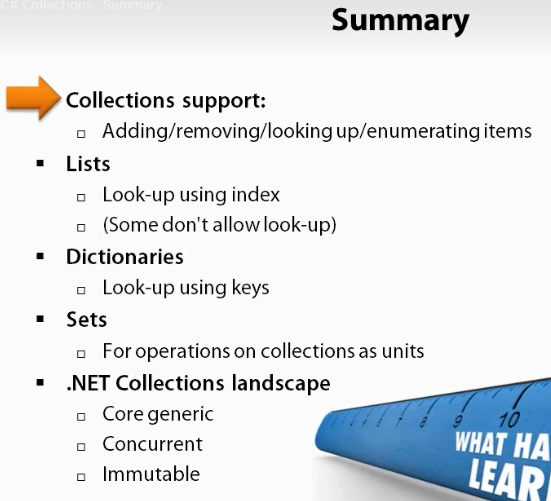


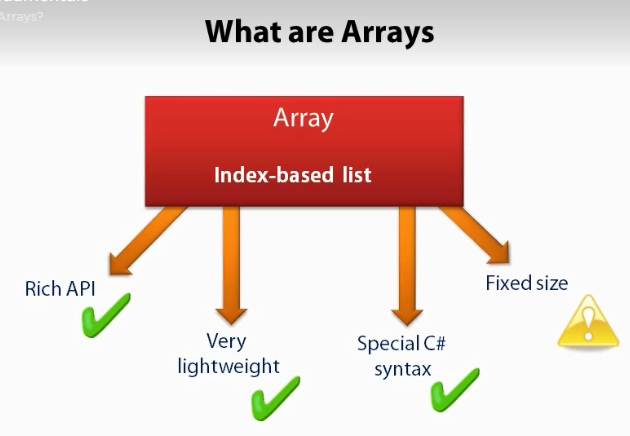


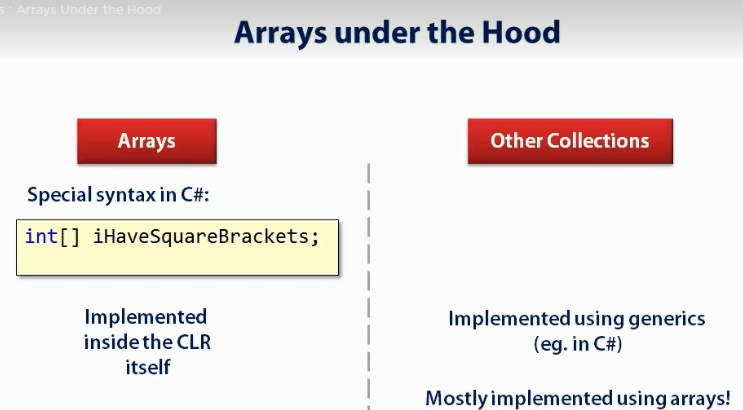


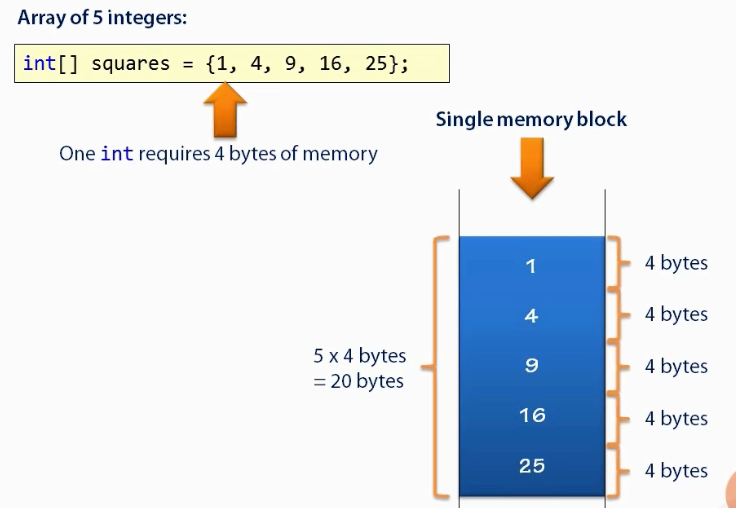
Sets enumerate all the data.



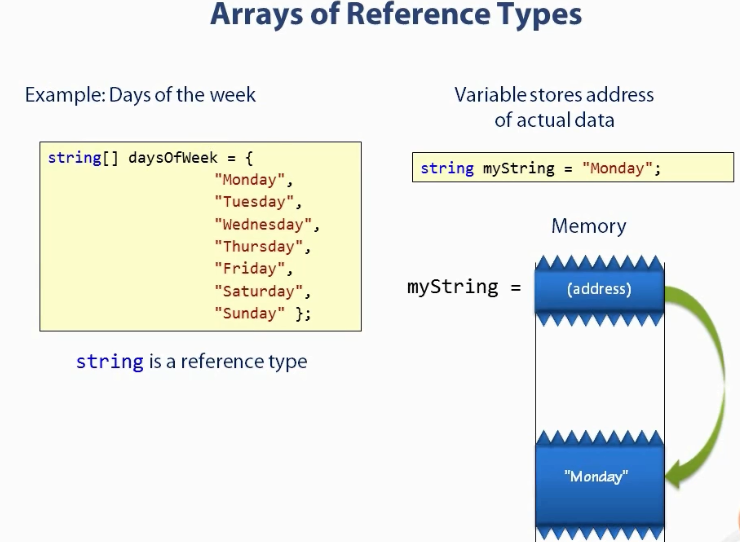


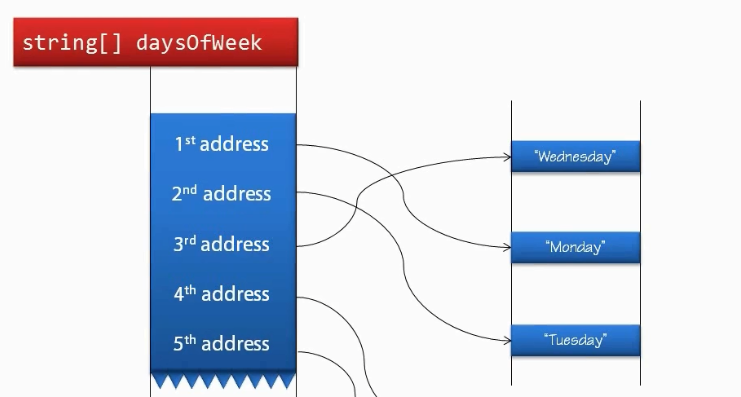


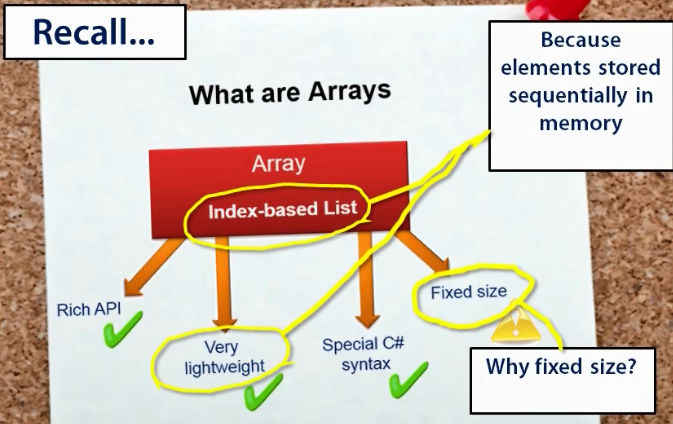








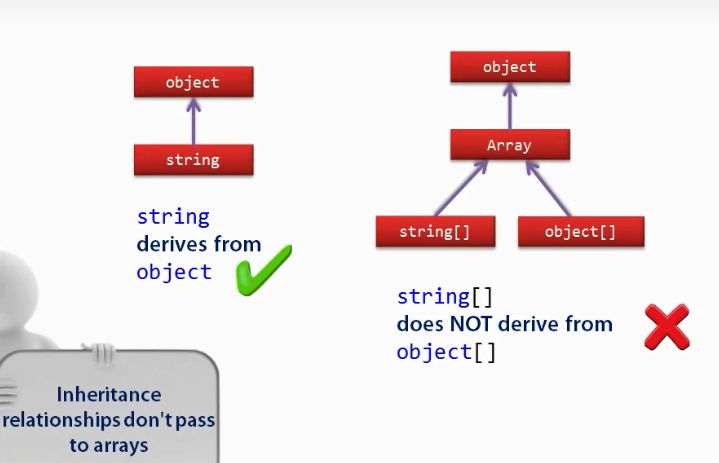


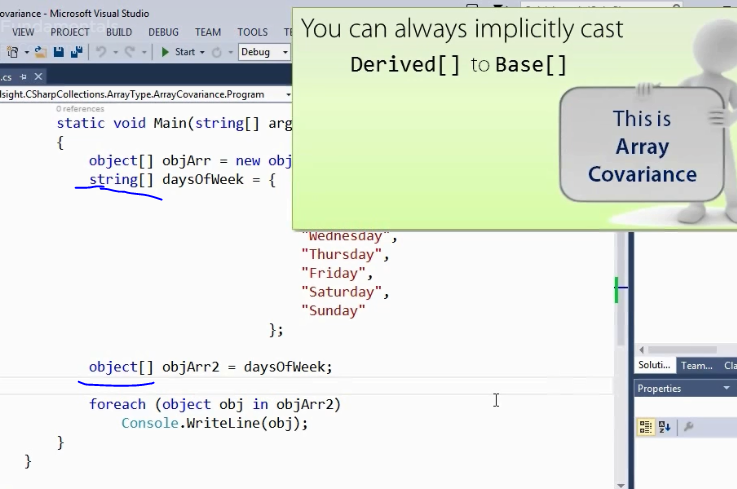


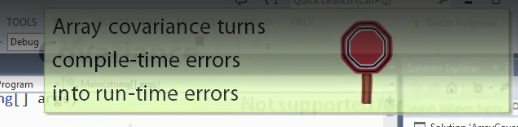
The memory immediately following the array is allocated to some other completely unrelated variable and space is not available to extend our array.

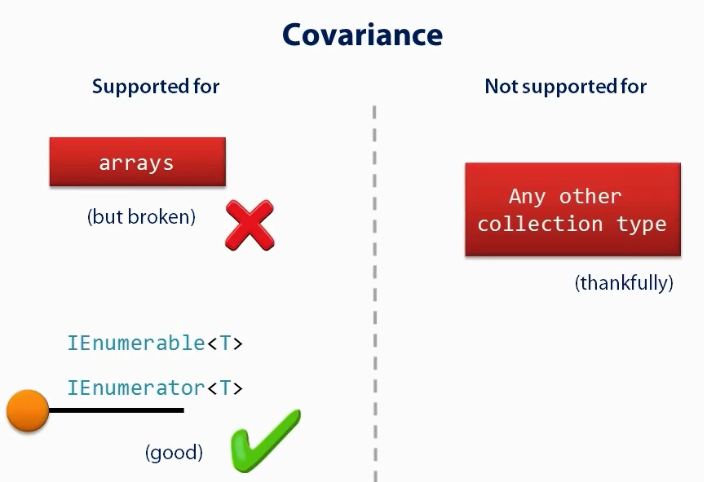


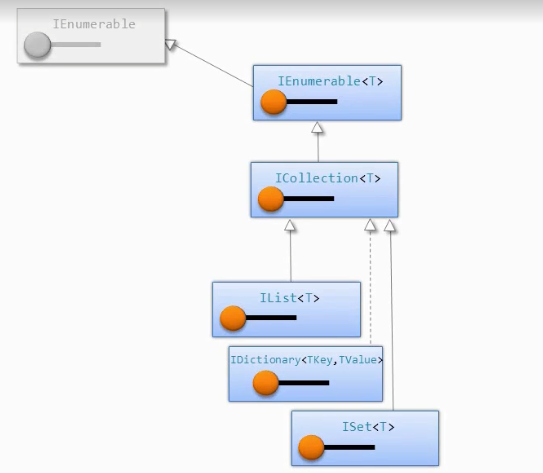










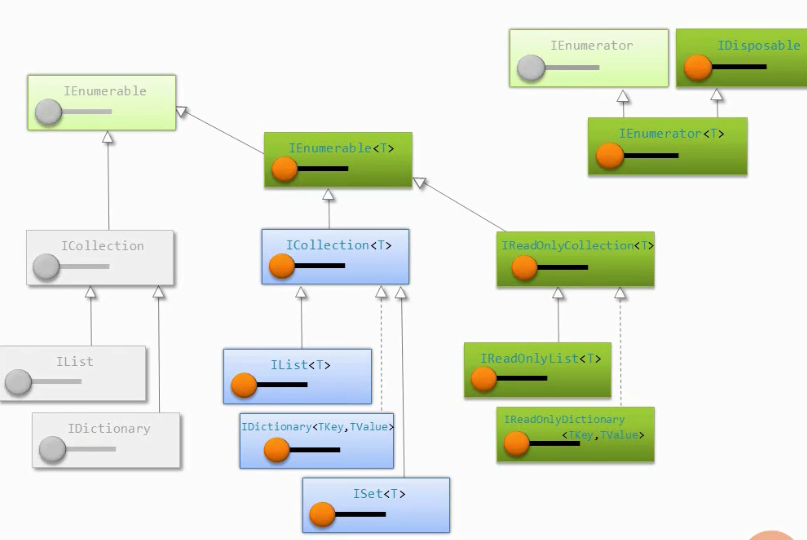


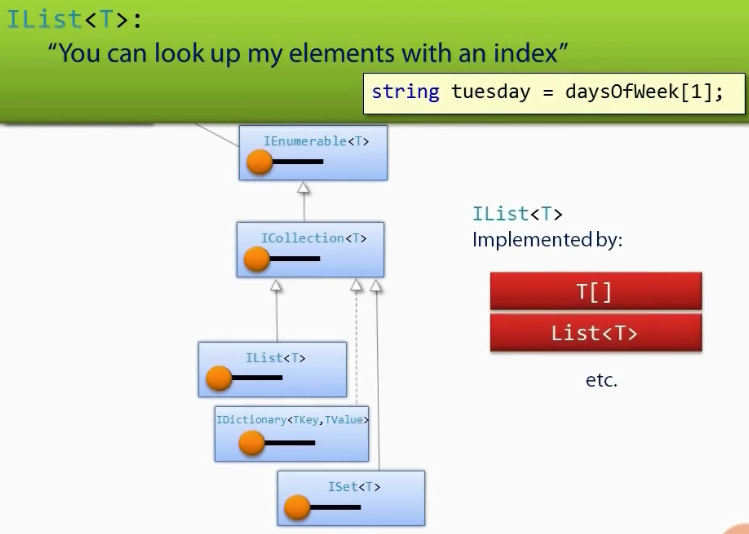


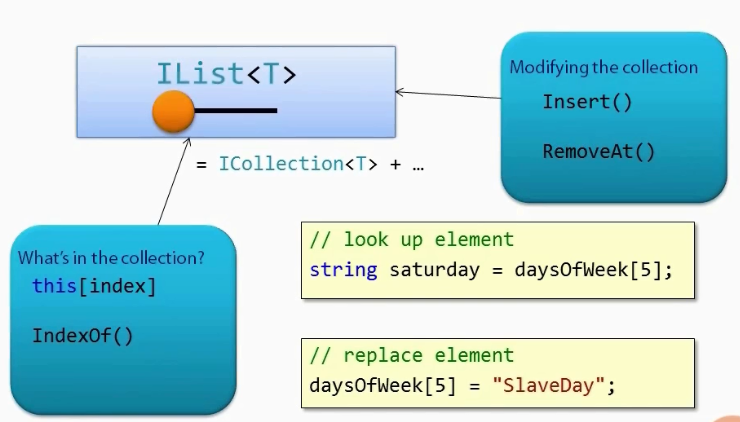
IEnumerable is a generic interface, it contains items that we can iterate. It does not know how many items are there.

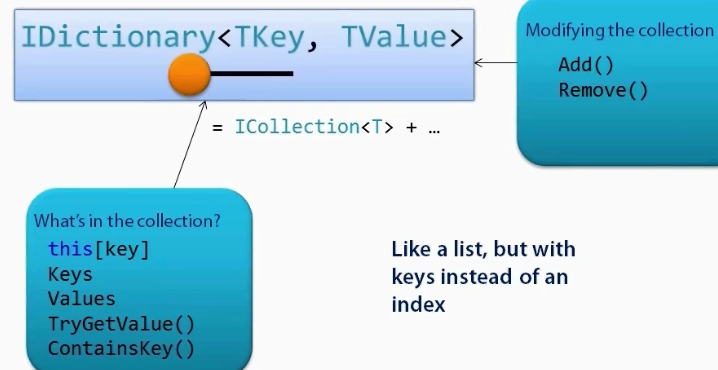


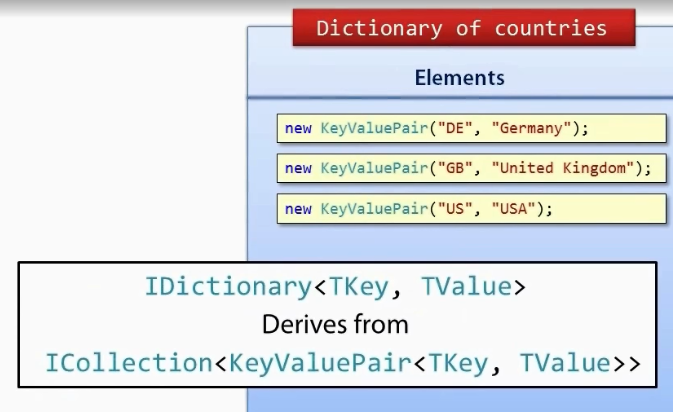
It’s a collection only and it will not let us know the type of collection.

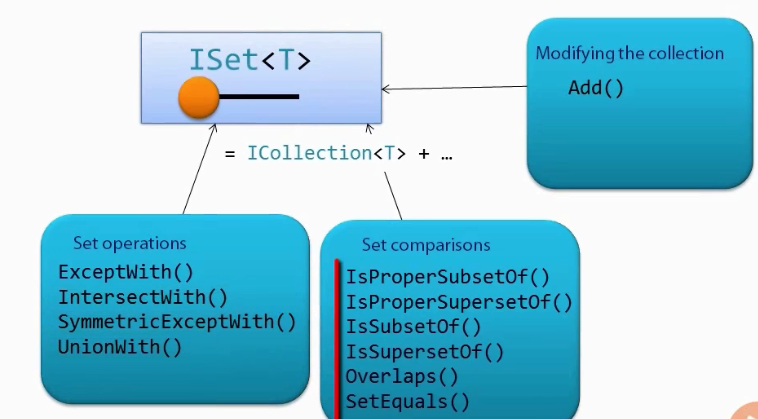


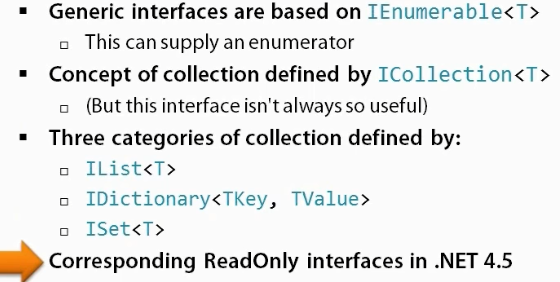






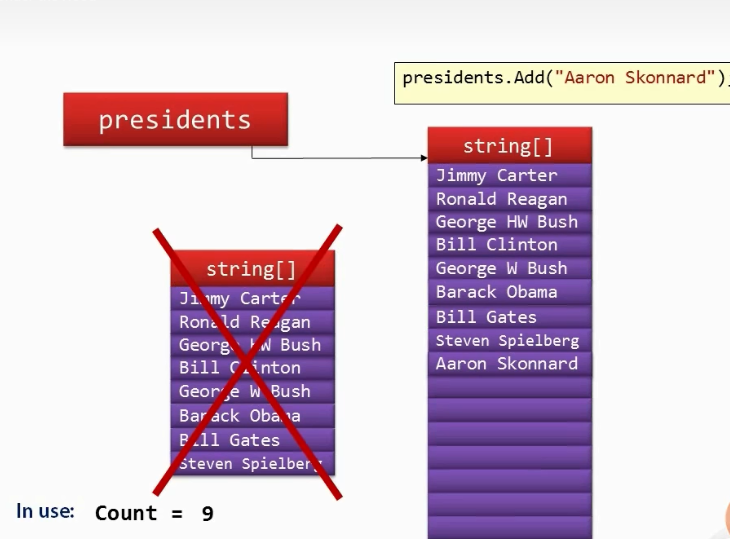






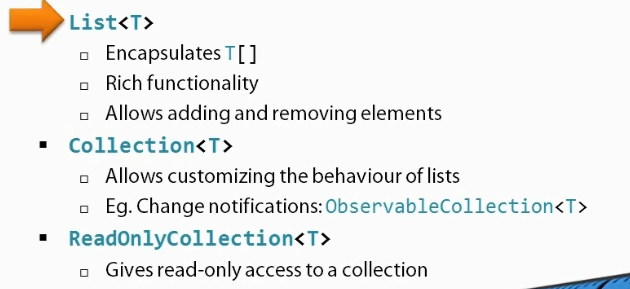
What we get is a reference to a newly created list of string instance. Like all collection type List<T> is reference type inside that there is reference to a string of array which contains the actual data.

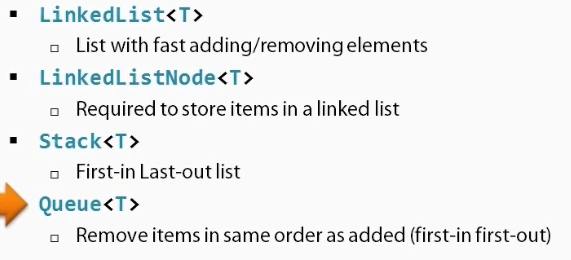
Removing the element near the beginning of list is expensive as we have to move a lot of element.



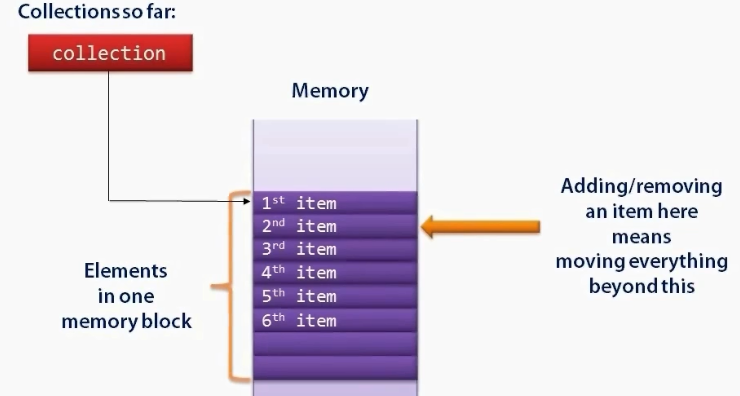
Lists are similar to array i.e. index based but only difference is we can add and remove an element form a list.

It works like: if the list capacity gets full then it discard the array and create new array which is double of the size of old array and all the element are added to the new array. In this way we can add new elements if there is no space left in the present list.

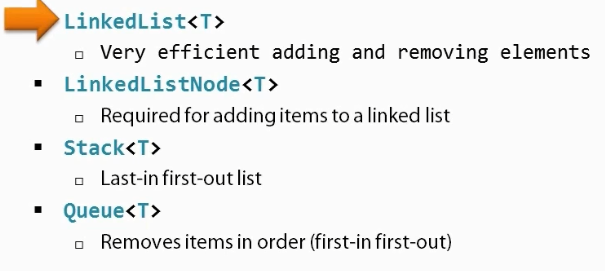


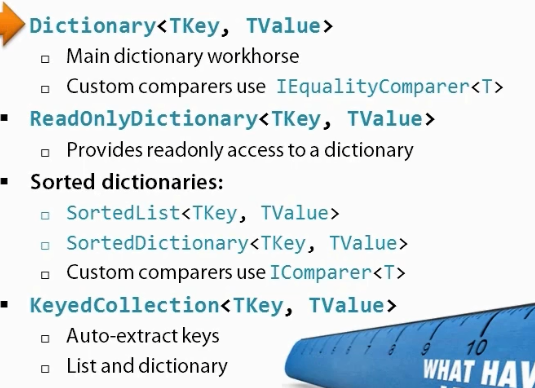






First element will point to the second element and it can tell us where the second element is.







List and Linked List:

Well, List<T> is basically backed by an array which is usually bigger than the current number of items. The elements are put in an array, and a new array is created when the old one runs out of space. This is fast for access by index, but slow at removing or inserting elements within the list or at the start. Adding/removing entries at the end of the list is reasonably cheap.

LinkedList<T> is a doubly-linked list - each node knows its previous entry and its next one. This is fast for inserting after/before a particular node (or the head/tail), but slow at access by index.

LinkedList<T> will usually take more memory than List<T> because it needs space for all those next/previous references - and the data will probably have less locality of reference, as each node is a separate object. On the other hand, a List<T> can have a backing array which is much larger than its current needs.