# Collections in C#

There 3 types if collections in C#

1. System.Collections classes
2. System.Collections.Generics classes
3. System.Collections.Concurrent classes

## Collections in System.Collections namespace :

**ArrayList**: it implements the System.Collections.IList interface using an array whose size is dynamically increased and this class is index based it takes the value as object.

Example:

ArrayList arrayList = new ArrayList();

arrayList.Add(123);

arrayList.Add("Ahmad");

arrayList.Add('2');

Console.WriteLine(arrayList[1]);

**Stack**: this data structure takes value as object and it represents the last-in-first-out (LIFO) and it is a non-generics.

Example:

Stack stack = new Stack();

stack.Push(123);

stack.Push('a');

stack.Push("Ahmad");

**Queue**: this data structure take value as object, it represents the first-in-first-out.

Example:

Queue queue = new Queue();

queue.Enqueue("Ahmad");

queue.Enqueue('S');

queue.Enqueue(123);

Console.WriteLine(queue.Peek());

**Hashtable**: it represents a key/value pairs that are organized based on the hash code of the key.

Example:

Hashtable hashtable = new Hashtable();

hashtable.Add(1, 2);

hashtable.Add(2, "Ahmad");

hashtable.Add(3, 'A');

Console.WriteLine(hashtable[2]);

**SortedList**: it represents a key/value pairs that are sorted by the keys and are accessible by the key and by index.

Example:

SortedList sortedList = new SortedList();

sortedList.Add(1, "Ahmad");

# Collections in System.Collections.Generics namespace:

**List<T>**: it represents a strongly typed list of objects that can be accessed by index, this provides methods to sort, search remove and add items.

Example:

List<string> vs = new List<string>();

vs.Add("Ahmad");

vs.Add("Sayeed Ahmad");

for (int i = 0; i < vs.Count(); i++)

{

Console.WriteLine(vs[i]);

}

**Stack<T>**: it is the same as **System.Collections.Stack** but it is restricted with types.

Example:

Stack<String> vs = new Stack<string>();

vs.Push("Ahmad");

vs.Push("Hamid");

Console.WriteLine(vs.Peek());

**Queue<T>**: it is the same as **System.Collections.Queue** but it is restricted with types.

Example:

Queue<String> vs = new Queue<string>();

vs.Enqueue("Ahmad");

vs.Enqueue("Mohammad");

vs.Enqueue("Sayeed Ahmad");

vs.Enqueue("Hamid");

Console.WriteLine(vs.Peek());

**HashSet<T>**: it takes a set of specified values and removes the duplicated values.

Example:

HashSet<String> vs = new HashSet<string>();

vs.Add("Ahmad");

vs.Add("Hamid");

vs.Add("Ahmad");

foreach (string item in vs)

{

Console.WriteLine(item);

}

**Dictionary<TKey, TValue>**: it represents a collection of key and value pairs with restricted types. This cannot let duplicated keys.

Example:

Dictionary<int, string> pairs = new Dictionary<int, string>();

pairs.Add(1, "Ahmad");

Console.WriteLine(pairs[1]);

**SortedList<TKey, TValue>**: it represents a Collection if a key/value pairs that are sorted based on implementation of **System.Collections.Generics.IComparer** interface.

Example:

SortedList<int, String> keys = new SortedList<int, string>();

keys.Add(1, "Ahmad");

keys.Add(3, "Mohammad");

keys.Add(2, "Khan");

keys.Add(5, "Hamid");

keys.Add(4, "Jan");

keys.Add(6, "Ubaid");

foreach (var item in keys)

{

Console.WriteLine(item.Key + " " + item.Value);

}

**SortedSet<T>**: this is a set of objects that are maintained in a sorted order.

Example:

SortedSet<int> vs = new SortedSet<int>();

vs.Add(1);

vs.Add(2);

vs.Add(6);

vs.Add(5);

vs.Add(3);

foreach (var item in vs)

{

Console.WriteLine(item);

}

**SortedDictionary<TKey, TValue>**: it represents a collection of a key value pairs.

Example:

SortedDictionary<int, string> pairs = new SortedDictionary<int, string>();

pairs.Add(1, "Ahmad");

pairs.Add(3, "Mohammad");

pairs.Add(2, "Khan");

pairs.Add(5, "Hamid");

pairs.Add(4, "Jan");

pairs.Add(6, "Ubaid");

foreach (var item in pairs)

{

Console.WriteLine(item.Key + " " + item.Value);

}

**LinkedList<T>**: it represents a collection that contain nodes that linked to each other. And a **LinkedListNode<T>** is just a node of the list.

Example:

LinkedList<int> vs = new LinkedList<int>();

LinkedListNode<int> ll = vs.AddFirst(1);

LinkedListNode<int> llb = vs.AddAfter(ll, 5);

vs.AddBefore(llb, 2);

vs.AddLast(20);

foreach (var item in vs)

{

Console.WriteLine(item);

}

## Collections in System.Collections.Concurrent namespace:

It came in .NET Framework Version 4 and onwards. It provides various threads-safe collection classes that are used in place of the corresponding types in the **System.Collections** and **System.Collections.Generic** namespaces, when multiple threads are accessing the collection simultaneously. The **System.Collections.Concurrent** namespace provides classes for thread-safe operations. Now multiple threads will not create problems for accessing the collection items.

**BlockingCollection<T>:** It provides blocking and bounding capabilities for thread-safe collections that implement **System.Collections.Concurrent.IProducerConsumerCollection**.

Example:

BlockingCollection<int> vs = new BlockingCollection<int>();

vs.Add(1);

vs.Add(2);

vs.Add(5);

vs.Add(3);

foreach (var item in vs)

{

Console.WriteLine(item);

}

**ConcurrentBag<T>**: It represents a thread-safe, unordered collection of objects.

Example:

ConcurrentBag<int> vs = new ConcurrentBag<int>();

vs.Add(1);

vs.Add(2);

vs.Add(4);

vs.Add(3);

vs.Add(7);

vs.Add(8);

vs.Add(6);

vs.Add(5);

foreach (var item in vs)

{

Console.WriteLine(item);

}

**ConcurrentStack<T>:**It represents a thread-safe last-in-first-out (LIFO) collection.

Example:

ConcurrentStack<int> vs = new ConcurrentStack<int>();

vs.Push(1);

vs.PushRange(new int[] { 2, 3, 4, 5, 6, 7});

**ConcurrentQueue<T>**: It represents a thread-safe first-in-first-out (FIFO) collection.

Example:

ConcurrentQueue<int> vs = new ConcurrentQueue<int>();

vs.Enqueue(1);

vs.Enqueue(3);

vs.Enqueue(4);

vs.Enqueue(5);

vs.Enqueue(6);

foreach (var item in vs)

{

Console.WriteLine(item);

}

**ConcurrentDictionary<TKey, TValue>:** It represents a thread-safe collection of key/value pairs that can be accessed by multiple threads concurrently.

Example:

ConcurrentDictionary<string, string> myConcuDict = new ConcurrentDictionary<string, string>();

myConcuDict.TryAdd("1", "A");

myConcuDict.TryAdd("2", "B");

myConcuDict.TryAdd("3", "C");

foreach (var item in myConcuDict)

{

Console.WriteLine(item.Key + " " + item.Value);

}