# Arrays in C#

آرایه مجموعه یا collections از یک نوع داده است که مکانی از حافظه تحت یک نام آن ها را نگه داری می کند.

آرایه نوعی ساختار داده ترتیبی است sequential data structure که مجموعه ای داده های یک نوع را در خود نگه داری می کند.

**How does using [] this work in real memory?**

**int[] empno = { 1, 2, 3, 4, 5 };**

زمانی که از علامت [] استفاده می کنید به کامپایلر می گویید که نوع داده ی که در حال استفاده از آن هستید یک آرایه است و بلوکی از حافظه به آن تخصیص داده شود.

**Types of Arrays in C#:**

1. **Single dimensional array تک بعدی**
2. **Multi-dimensional array چند بعدی**

در آرایه تک بعدی دیتا به صورت row یا ردیف در کنار هم قرار می گیرند در حالی که در آرایه های چند بعدی داده های به صورت ردیف و ستون rows and columns در کنار هم قرار می گیرند.

آرایه های چند بعدی 2 نوع هستند

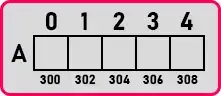
1. **Jagged array**: Whose rows and columns are not equal دندانه دار ردیف و ستون نا برابر
2. **Rectangular array**: Whose rows and columns are equal مستطیلی ردیف و ستون ها برابر

**نکته:** برای دسترسی به آرایه می توان از index استفاده کرده و همیشه اولین خانه دارای مقدار برابر است با صفر 0 و تعداد داده های آرایه نیز برابر است با تعداد خانه ها -1

ساختن آرایه با 5 خانه

**Example: int[] A = new int[5];**

فرض کنید خانه حافظه به شکل زیر است از 300 شروع شده و آرایه ما از نوع int است که هر کدام 2byte فضا می گیرد پس حافظه به شکل زیر خواهد بود.



در این مثال خانه a[0] برابر است با 300 و 3001 , a[1] برابر است با 302 و 303

Indexer

در آرایه در واقع یک pointer است که برای دسترسی به داده آرایه در یک خانه مشخص استفاده می شود و از 0 تا n-1 ادامه دارد.

Array is fast?

به دلیل این که مقادیر در حافظه به صورت پیوسته یا پشت سر هم قراردارند دسترسی به آن ها سریع است.

**2 نوع آرایه در سی شارپ داریم طول ثابت و متغییر در طول متغییر سایز آرایه به صورت خودکار با افزایش ایتم های جدید اضافه می شود.**

**نکته: array is reference type and heap memory is allocate**

**چرا می توان روی آرایه foreach زد؟**

به دلیل پیاده سازی رابط IEnumerable می توانیم بر روی عناصر آرایه پیمایش کنیم

Int[] arr = {1,23,4,5,6} روش initialize کردن آرایه.

Foreach

به صورت مخصوص برای دسترسی به عناصر آرایه ها ایجاد شده است و از عنصر اول تا آخر بدون نیاز به شرط پیمایش می کند.

Foreach vs for

For نیاز به دسترسی به index دارد ولی foreach مستقیما به داده دسترسی دارد

نکته : for برای دسترسی و assign کردن کاربرد دارد ولی foreach فقط برای دسترسی به داده.

نکته: دسترسی به عنصری از آرایه که خارج از محدوده است سبب exception می شود

**Array base class is implementing IList interface**

# 2D Arrays in C#

آرایه های که در سی شارپ که عناصر آن در ردیف و ستون rows & column ذخیره می شوند دو بعدی نام دارند.

انوع آرایه 2 بعدی

Rectangular در این نوع آرایه ها تعداد ردیف ها و ستون ها با هم برابر است

Jagged در این نوع آرایه ها تعداد ردیف ها و ستون ها با هم برابر نیست.

در این نوع آرایه ها به هر عنصر از آرایه با 2 index دسترسی می توان داشت المان ها در این نوع آرایه ها به صورت matrix ذخیره می شوند.

در این نوع آرایه ها index اول به ردیف row و index دوم به column یا ستون اشاره می کند.

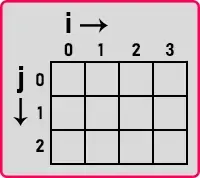
**int[,] matrix = new int[3,3];**



برای دسترسی به خانه ی اول matrix[0][0] به این روش عمل می کنیم.

**How 2D Array is Created and Accessed in C#?**

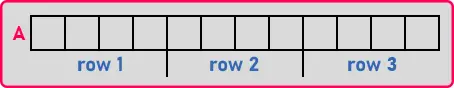
**int[,] A = new int[3,4];**



**برای دسترسی و چاپ عنصر آرایه از روش زیر استفاده می کنیم.**

**Console.WriteLine(A[1,2]);**

نحوه ذخیره سازی و دسترسی به عناصر آرایه 2 بعدی در حافظه Ram به این صورت است که در ردیف های پشت سر هم در حافظه داده ها ذخیره می شوند و تقسیم بندی منطقی ردیف ها انجام می شود



نحوه تعریف یک آرایه با 2 ردیف داده و 3 ستون

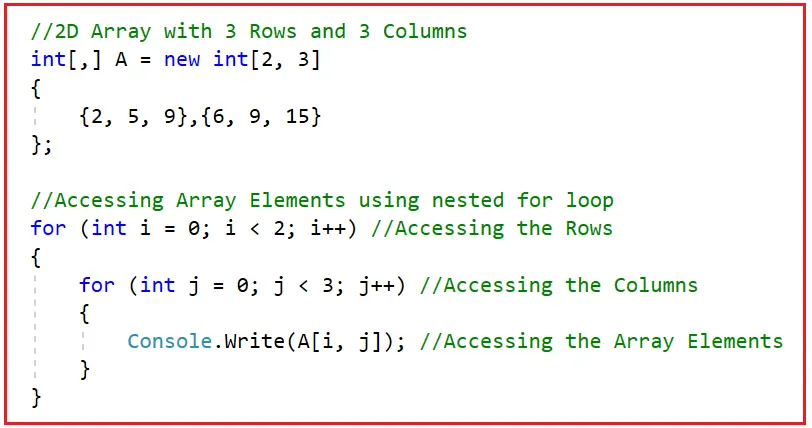
**int[,] A = new int[2,3]**  
**{**  
**{2, 5, 9},{6, 9, 15}**  
**};**

**OR**

**int[,] A = {{2, 5, 9},{6, 9, 15}};**

**Accessing the Elements of the 2D array in C#:**

برای دسترسی به عناصر آرایه های 2 بعدی به 2 حلقه تو در تو نیاز داریم یکی برای دسترسی به عناصر row و دیگری برای دسترسی به عناصر columns به مثال زیر توجه کنید.



**نکته** برای دسترسی به تعداد ردیف ها GetLength(0) و برای دسترسی به تعداد ستون ها GetLenght(1) استفاده می شود.

مثال پرکردن عناصر آرایه:

//Assigning values to the 2D array by using nested for loop

//arr.GetLength(0): Returns the size of the Row

//arr.GetLength(0): Returns the size of the Column

**for** **(int** i = 0; i **<** RectangleArray.GetLength**(**0**)**; i++**)**

**{**

**for** **(int** j = 0; j **<** RectangleArray.GetLength**(**1**)**; j++**)**

**{**

a += 5;

RectangleArray**[**i, j**]** = a;

**}**

**}**

**Jagged Array in C#:**

نوع دیگری از آرایه های 2 بعدی هستند که شباهتی بسیاری به آرایه های 2 بعدی مستطیلی دارد با این تفاوت که در این نوع آرایه ها می توان هر ردیف داده دارای تعداد ستون داده متفاوتی باشد مثلا ردیف اول دارای 4 ستون داده و ردیف دوم دارای 2 ستون داده باشد

به این نوع آرایه، آرایه ی از آرایه ها نیز می گویند به این دلیل که هر ردیف خود یک آرایه تک بعدی است.

در تعریف این نوع آرایه ها فقط کافی است تعداد row را مشخص کنید. و سپس برای هر row می توانید تعداد ستون ها را مشخص کنید.

**int [][] arr = new int[4][];**

**arr[0] = new int[5];** // we want five columns in the first row  
**arr[1] = new int[6];** // we want six columns in the first row  
**arr[2] = new int[4];** // we want four columns in the first row  
**arr[3] = new int[5];** // we want five columns in the first row

در ادامه نحوه دسترسی به عناصر آرایه را بررسی می کنیم.

//Printing the values of Jagged array using nested for loop

//It will print the default values as we are not assigning any

//values to the array

//GetLength(0): Returns the Size of the Rows (4)

Console.WriteLine**(**"Printing the Default Values of Jagged Array:"**)**;

**for** **(int** i = 0; i **<** arr.GetLength**(**0**)**; i++**)**

**{**

//arr[i].Length: Returns the Length of Each Row

**for** **(int** j = 0; j **<** arr**[**i**]**.Length; j++**)**

**{**

Console.Write**(**arr**[**i**][**j**]** + " "**)**;

**}**

**}**

# Advantages and Disadvantages of Arrays in C#

مزایا Advantages

ذخیره کردن چندین مقدار از یک نوع تحت عنوان یک نام

برای پیاده سازی سایر ساختار های داده مانند linked list graphs stacks queues کاربرد دارد

آرایه های 2 بعدی برای پیاده سازی ماتریکس کاربرد دارند.

از نوع strongly type هستند.

چون از نوع strongly هستند مزایای دارند :

Performance اپلیکیشن ما بهتر می شود چون boxing , unboxing رخ نمی دهد و خطای RunTime رخ نمی دهد چون در زمان compile متوجه خطا ها می شویم.

**نکته : loosely type collection مثل ArrayList نوع های کالکشنی هستند که می توان در آن ها هر نوع داده ی را نگه داری کرد و در زمان کامپایل خطا نمی دهد.**

**معایت Disadvantages**

Fix size به معنی این است که زمانی که ارایه ایجاد می شود دیگر قابلیت increase or decrease برای آن آرایه وجود ندارد

اگر حافظه ی بیشتر از نیاز خود اختصاص دهیم waste می شود حافظه و اگر کمتر از نیاز تخصیص دهیم دچار مشکل می شویم

هیچ وقت نمی توانیم که مقداری را از middle of array اضافه کنیم یا همچنین نمی توان از وسط آرایه حذف کرد.

# Collections in C#

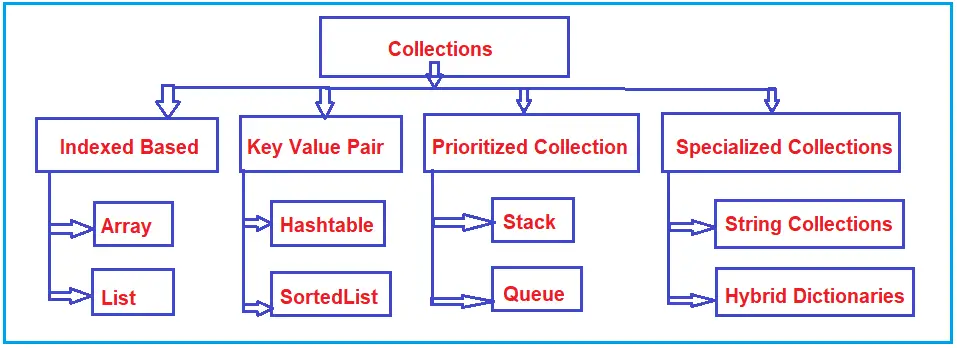
کالکشن ها شبیه آرایه هستند و راهی منعطف برای کار کردن با گروهی از اشیا هستند.

گروهی از رکورد ها هستند که می توانند تحت یک واحد منطقی استفاده شوند.

Categories of Collections

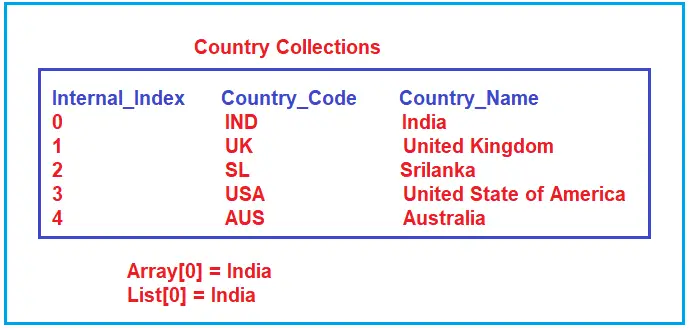
4 نوع کلی هستند که عبارتند از

Indexed – kay value pair – prioritized collection – specialized collections



**Indexed Base Collections:**

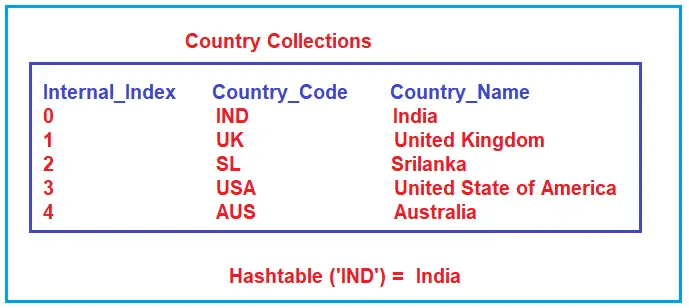
کالکشنی مانند Array or List را در نظر بگیرید این نوع ها زمانی که المانی به آن ها اضافه می شود به صورت خودکار توسط فریم ورک برای آن عنصر یک index اختصاص داده می شود که از طریق این index اطلاعات ما قابل بازیابی است.



**Key-Value Pair Collections**

از انواع آن ها HashTable – Dictionary – SortedList هستند

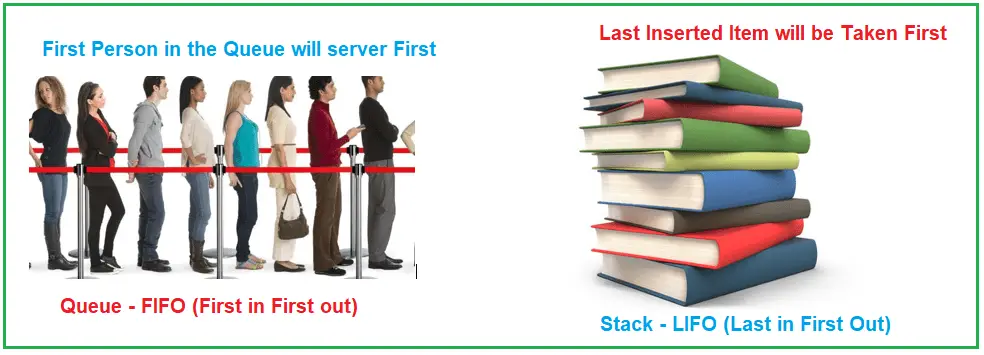
در پروژه ها بزرگ معمولا برای identify کردن رکورد ها از index استفاده نمی کنیم بلکه از یک key استفاده می شود که در ادامه مثال آن را مشاهده خواهید کرد. برای بازیابی رکورد ها با استفاده از کلید این ها مناسب هستند.



Prioritized Collections:

مجموعه های اولویت دار.

کمک می کند که به عناصر به ترتیب خاصی دسترسی داشت. Stack , queue از دسته بندی کالکشن های اولیت دار است مثلا برای پیاده سازی FIFO می توان از Queue استفاده کرد و برای پیاده سازی LIFO می توان از نوع Stack استفاده کرد.



**Specialized Collections:**

The Specialized Collections are specifically designed for a specific purpose. For example, a Hybrid Dictionary starts as a list and then becomes a hashtable.

**What are Array and Their disadvantages in C#?**

به طور مثال در آرایه ها لازم است که شما ابتدا size آرایه را مشخص کنید و در صورتی که در طول اجرای برنامه نیاز به تغییر size آرایه باشد این امکان فراهم نیست نیاز است به ایجاد یک آرایه به صورت دستی و کپی کردم مقادیر به آن است.

عدم امکان increase or decrease کردن سایز آرایه و همچنین افزودن یا کاستن مقدار از middle of array.

**What is a Collection in C#?**

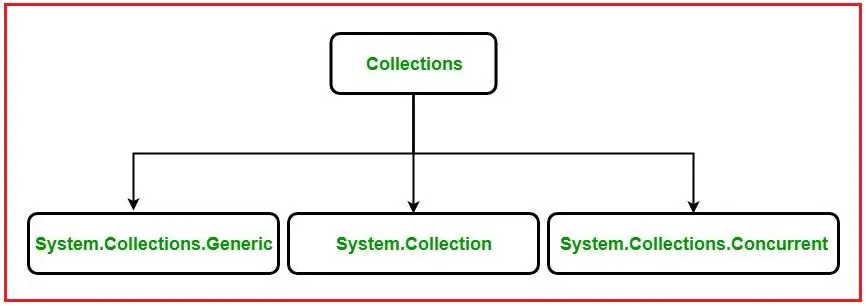
کلاس های در فضای نام system.Collections هستند که دارای ظرفیت و عملکرد بهتر نسبت به آرایه های سنتی هستند. Collections در سی شارپ reusable هستند و کارآمدی بهتر دارند.

به زبان ساده کالکشن ها در سی شارپ برابر dynamic array است.

قابلیت افزایش سایز به صورت پویا

درج کردم مقدار جدید در middle of array و همچنین حذف مقدار. قابلیت مرتب سازی جستجو بروزرسانی و حذف و .. را دارند.

**Types of Collections in C#**



Non generic برای مدیرت هر نوع داده ی کاربرد دارد. بر روی object کار می کند.

1. [**ArrayList**](https://dotnettutorials.net/lesson/arraylist-collection-csharp/)**:** It Implements the System.Collections.IList interface using an array whose size is dynamically increased as required.
2. [**Stack**](https://dotnettutorials.net/lesson/stack-collection-csharp/)**:** It represents a simple last-in-first-out (LIFO) non-generic collection of objects.
3. [**Queue**](https://dotnettutorials.net/lesson/queue-collection-class-csharp/)**:** It represents a first-in, first-out collection of objects.
4. [**Hashtable**](https://dotnettutorials.net/lesson/hashtable-csharp/)**:** It represents a collection of key/value pairs that are organized based on the hash code of the key.
5. [**SortedList**](https://dotnettutorials.net/lesson/sortedlist-collection-class-in-csharp/)**:** It represents a collection of key/value pairs that are sorted by the keys and are accessible by key and by index.

**Generic Collections Classes in C#:**

این نوع به صورت type safe هستند یعنی فقط مواردی ار که با نوع مجموعه compatible است را می توان در آن ها ذخیره کرد.

1. [**List<T>**](https://dotnettutorials.net/lesson/list-collection-csharp/)**:** It represents a strongly typed list of objects that can be accessed by index. Provides methods to search, sort, and manipulate lists.
2. [**Stack<T>**](https://dotnettutorials.net/lesson/generic-stack-csharp/)**:** It represents a variable size last-in-first-out (LIFO) collection of instances of the same specified type.
3. [**Queue<T>**](https://dotnettutorials.net/lesson/generic-queue-collection-class-csharp/)**:** It represents a first-in, first-out collection of objects.
4. [**HashSet<T>**](https://dotnettutorials.net/lesson/generic-hashset-collection-class-in-csharp/)**:** It represents a set of values. It removes duplicate elements from the collection.
5. [**Dictionary<TKey, TValue>**](https://dotnettutorials.net/lesson/dictionary-generic-collection-csharp/)**:** It represents a collection of keys and values.
6. [**SortedList<TKey, TValue>**](https://dotnettutorials.net/lesson/generic-sortedlist-collection-class-in-csharp/)**:** It represents a collection of key/value pairs that are sorted by key based on the associated System.Collections.Generic.IComparer implementation.
7. [**SortedSet<T>**](https://dotnettutorials.net/lesson/generic-sortedset-collection-class-in-csharp/)**:** It represents a collection of objects that are maintained in sorted order.
8. [**SortedDictionary<TKey, TValue>**](https://dotnettutorials.net/lesson/generic-sorteddictionary-collection-class-in-csharp/)**:** It represents a collection of key/value pairs that are sorted on the key.
9. [**LinkedList<T>**](https://dotnettutorials.net/lesson/generic-linkedlist-collection-class-in-csharp/)**:** It represents a doubly linked list.

**Concurrent Collection Classes in C#:**

این نوع ها safe collection هستند برای کار با thread ها زمانی که چندین thread به صورت همزمان به collection دسترسی دارند فضای نام Concurrent کلاس های را برای عملیات Thread – safe فراهم می کند که مشکلی برای دسترسی به ایتم های کالکشن ایجاد نشود.

1. [**BlockingCollection<T>**](https://dotnettutorials.net/lesson/blockingcollection-class-in-csharp/)**:** It provides blocking and bounding capabilities for thread-safe collections that implement System.Collections.Concurrent.IProducerConsumerCollection.
2. [**ConcurrentBag<T>**](https://dotnettutorials.net/lesson/concurrentbag-collection-class-in-csharp/)**:** It represents a thread-safe, unordered collection of objects.
3. [**ConcurrentStack<T>**](https://dotnettutorials.net/lesson/concurrentstack-collection-class-csharp/)**:** It represents a thread-safe last-in-first-out (LIFO) collection.
4. [**ConcurrentQueue<T>**](https://dotnettutorials.net/lesson/concurrentqueue-collection-class-in-csharp/)**:** It represents a thread-safe first-in-first-out (FIFO) collection.
5. [**ConcurrentDictionary<TKey, TValue>**](https://dotnettutorials.net/lesson/concurrentdictionary-collection-class-in-csharp/)**:** It represents a thread-safe collection of key/value pairs that can be accessed by multiple threads concurrently.

# ArrayList in C#

از نوع کالکشن های non-generic است مانند یک Array است اما امکاناتی مانند dynamic resize , adding deleting از میانه کالکشن را فراهم می کند.

می تواند برای افزودن unknow data استفاده شود زمانی که type , size داده را نمی دانیم کاربرد دارد.

ویژگی مهم آن افرایش سایز و استفاده از نوع داده یکسان و غیر یکسان در این نوع کالکشن ها است.

ظرفیت arrayList برابر است با تعداد عناصر موجود در آن. قابلیت عنصر تکراری دارد و می تواند مقادیر duplicate را نیز نگه داری کند.

**How to Create an ArrayList in C#?**

1. **ArrayList():** The method is used to initialize a new instance of the ArrayList class that is empty and has the default initial capacity.
2. **ArrayList(ICollection c):** The method is used to initialize a new instance of the ArrayList class that contains elements copied from the specified collection and that have the same initial capacity as the number of elements copied. The parameter c specifies the Collection whose elements are copied to the new list.
3. **ArrayList(int capacity):** The method is used to initialize a new instance of the ArrayList class that is empty and has the specified initial capacity. The parameter capacity specifies the number of elements that the new list can initially store.

نکته: در یک ArrayList امکان اضافه کردن نوع های داده مختلف وجود دارد. قابلیت object initializer نیز دارد

var arrayList2 = new ArrayList**()**

**{**

102, "Smith", "Smith", **true**, 15.6

**}**;

**How to Access an ArrayList in C#?**

اگر definition کلاس ArrayList را نگاه کنید می بینید که رابط IList را پیاده سازی کرده است به همین دلیل با استفاده از **indexer به عناصر آن دسترسی داریم نکته IList**

**نکته:** زمانی که داده ی در ArrayList ذخیره می شود به صورت خدکار به object تغییر پیدا می کند برای بازیابی داده یا باید از نوع var استفاده کرد یا از Explicit cast استفاده کنید.

//Accessing individual elements from ArrayList using Indexer

**int** firstElement = **(int)**arrayList1**[**0**]**; //returns 101

**string** secondElement = **(string)**arrayList1**[**1**]**; //returns "James"

//int secondElement = (int) arrayList1[1]; //Error: cannot cover string to int

Console.WriteLine**(**$"First Element: {firstElement}, Second Element: {secondElement}"**)**;

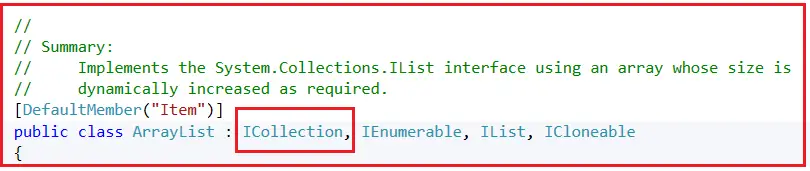
//Using var keyword without explicit casting

var firsItem = arrayList1**[**0**]**; //returns 101

var secondItem = arrayList1**[**1**]**; //returns "James"

//var fifthElement = arrayList1[5]; //Error: Index out of range

نکته : اگر به definition کلاس arrayList رفته باشید می بینید که این کلاس رابط ICollection را پیاده سازی کرده است این interface قابلیت iteration کردن بین عناصر آرایه را به ما می دهد به همین دلیل foreach را پشتیابنی می کند.



Add vs Insert

برای اضافه کردن یک ایتم به انتهای لیست می توان از Add استفاده کرد و همچنین برای افزودن یک ایتم به position به خصوصی می توان از insert استفاده کرد.

//Insert "First Element" at First Position i.e. Index 0

arrayList.Insert**(**0, "First Element"**)**;

InsertRange

برای افزودن مقادیر یک Collection به یک ArrayList به صورت جمعی می توان از InsertRange استفاده کرد دارای پارامتر index برای مشخص کردن این که از چه موقعیتی داده ها اضافه شوند به ArrayList

arrayList1.InsertRange**(**2, arrayList2**)**;

**How to Remove Elements from ArrayList in C#?**

1. **Remove(object? obj):** This method is used to remove the first occurrence of a specific object from the System.Collections.ArrayList. The parameter obj specifies the Object to remove from the ArrayList. The value can be null.
2. **RemoveAt(int index):** This method is used to remove the element at the specified index of the ArrayList. The parameter index specifies the index position of the element to remove.
3. **RemoveRange(int index, int count):** This method is used to remove a range of elements from the ArrayList. The parameter index specifies the starting index position of the range of elements to remove and the parameter count specifies the number of elements to remove.

##### **How to Remove all the elements from the ArrayList in C#?**

برای پاک کردن تمامی عناصر می توان از متد Clear استفاده کرد ضمن در نظر گرفتن این نکته که **این متد ظرفیت ArrayList را کاهش نمی دهد.**

**int** totalItems = arrayList.Count;

Console.WriteLine**(string**.Format**(**$"Total Items: {totalItems}, Capacity: {arrayList.Capacity}"**))**;

//Remove all items from the Array list

arrayList.Clear**()**;

totalItems = arrayList.Count;

Console.WriteLine**(string**.Format**(**$"Total Items After Clear(): {totalItems}, Capacity: {arrayList.Capacity}"**))**;

Console.ReadKey**()**;



**How do we Check whether an Element exists in ArrayList or not in C#?**

1. **bool Contains(object? item):** This method is used to determine whether an element is in the ArrayList. The parameter item specifies the Object to locate in the ArrayList. The value can be null. It returns true if the item is found in the ArrayList; otherwise, false.

**Note:**It is not recommended to use the non-generic collection class ArrayList in C# due to performance issues i.e. boxing and unboxing as it is operating on the object data type. So, instead of using ArrayList, it is recommended to use the generic collection List<object> to store heterogeneous objects. To store data of the same data type, use Generic List<T>.

**How to Clone the Non-Generic ArrayList Collection in C#?**

If you want to clone the Non-Generic ArrayList collection in C#, then you need to use the following Clone() method provided by the ArrayList Collection Class.

1. **Clone():** This method is used to create and return a shallow copy of the ArrayList.

**How to copy an ArrayList to an existing array in C#?**

1. **CopyTo(Array array):** This method is used to copy the entire ArrayList to a compatible one-dimensional Array, starting at the beginning of the target array. The parameter array specifies the one-dimensional Array that is the destination of the elements copied from ArrayList. The Array must have zero-based indexing. If the parameter array is null, then it will throw ArgumentNullException.
2. **CopyTo(Array array, int arrayIndex):** This method is used to copy the entire ArrayList to a compatible one-dimensional Array, starting at the specified index of the target array. Here, the parameter array specifies the one-dimensional array that is the destination of the elements copied from the ArrayList. The Array must have zero-based indexing. The arrayIndex parameter specifies the zero-based index in the array at which copying begins. If the parameter array is null, then it will throw ArgumentNullException. If the parameter arrayIndex is less than zero, then it will throw ArgumentOutOfRangeException.
3. **CopyTo(int index, Array array, int arrayIndex, int count)**: This method is used to copy a range of elements from the System.Collections.ArrayList to a compatible one-dimensional Array, starting at the specified index of the target array. The index parameter specifies the zero-based index in the source System.Collections.ArrayList at which copying begins. The array parameter specifies the one-dimensional Array that is the destination of the elements copied from ArrayList. The Array must have zero-based indexing. The arrayIndex parameter specifies the zero-based index in the array at which copying begins. The count parameter specifies the number of elements to copy. If the parameter array is null, then it will throw ArgumentNullException. If the parameter index is less than zero, arrayIndex is less than zero, or the count is less than zero, then it will throw ArgumentOutOfRangeException.

**How to Sort the Elements of an ArrayList Collection in C#?**

If you want to sort the elements of the ArrayList in C#, then you can use the following Sort() method of the ArrayList class.

1. **Sort():** This method is used to sort the elements in the entire System.Collections.ArrayList.
2. **Sort(IComparer? comparer):** This method is used to sort the elements in the entire ArrayList using the specified comparer.
3. **Sort(int index, int count, IComparer? comparer):** This method is used to sort the elements in a range of elements in ArrayList using the specified comparer.

**What is the difference between an Array and an Array List in C#?**

The ArrayList collection in C# is very much similar to the Arrays data type. The major difference between them is the dynamic nature of the non-generic collection ArrayList. For arrays, we need to define the size i.e. the number of elements that the array can hold at the time of array declaration. But in the case of the ArrayList collection in C#, this does not need to be done beforehand. Elements can be added or removed from the Array List collection at any point in time.

This is one of the frequently asked interview questions in C#. So let us discuss the difference between an array and an ArrayList.

**Array:**

1. Fixed Length
2. Cannot insert it into the middle
3. Cannot delete from middle
4. It is type-safe, so we can store only similar types of data based on the data type.
5. Boxing and Unboxing are not required.

**ArrayList:**

1. Variable Length
2. Can insert an element into the middle of the collection
3. Can delete elements from the middle of the collection
4. It is not type-safe, so we can store any type of data.
5. Boxing and Unboxing are required as it is operated on the object data type.

# Hashtable in C#

از نوع non-generic است برای حل مشکلاتی است که با Array , ArraList داریم

**What are the Problems with Array and ArrayList Collection in C#?**

در Array , ArrayList دسترسی به عناصر با استفاده از Index است که از 0 تا تعداد کل – 1 متغییر است که برای ما سخت است که به ترتیب index تمامی اعضا را بدانیم.

ArrayList al = new ArrayList**()**;

al.Add**(**1001**)**; //EId - Index Position = 0

al.Add**(**"James"**)**; //Name - Index Position = 1

al.Add**(**"Manager"**)**; //Job - Index Position = 2

//I want to print the Location, Index position is 4

Console.WriteLine**(**"Location : " + al**[**4**])**;

//I want to print the Email ID, Index position is 6

Console.WriteLine**(**"Emaild ID : " + al**[**6**])**;

اگر تعداد زیاد باشد به خاطر سپردن index ها سخت است.

**راه حل استفاده از Key و Hashtable**

این نوع non generic collection برای ذخیره سازی داده ها در فرمت **key-value pair** کاربرد دارد.

**داده ها بر اساس کلید هش شده سازمان دهی می شوند.**

**نکته : کلید می تواند هر نوع داده ی باشد.**

این نوع برای هر کلید یک کد هش محاسبه می کند که برای جست جوی هر المان در مجموعه سبب افرایش سرعت می شود.

نکته : این collection رابط IDictionary را پیاده سازی کرده است

نکته: هر **کلید** باید unique باشد و null نباشد و می تواند هر نوع داده ی باشد.

نکته: مقادیر می تواندد تکراری و null باشد.

نکته: ظرفیت Hashtable بر اساس تعداد المان های که می تواند نگه دارد است.

نکته : به دلیل این که non-generic است می تواند هر نوع داده و نوع داده یکسانی را در خود نگه دارد.

**How Actually the Hashtable works in C#?**

زمانی که به Hashtable یک المان اضافه می کنیم از هر نوعی که باشد int string ,… سپس کلید مربوط به داده را فارغ ار نوع آن تبدیل به کد hash از نوع int تبدیل می کند به همین دلیل پیدا کردن آن سریع است زمانی که تبدیل کلید تمام شود سپس عملیات افزودن انجام می شود.

نکته : Hashtable performance is less than ArrayList because of key Conversion

**Differences between ArrayList and Hashtable in C#:**

1. **Lookup**: ArrayList can be only looked up via the index number which is generated internally. Hashtable can be looked up by a custom-defined key.
2. **Performance**: ArrayList is faster than hashtable because of extra tasks performed in hashtables i.e. hashing.
3. **Scenario**: If you want a key lookup use hashtable. If you just want to add and browser through a collection then use ArrayList.

**var cities = new Hashtable(){**  
**{“UK”, “London, Manchester, Birmingham”},**  
**{“USA”, “Chicago, New York, Washington”},**  
**{“India”, “Mumbai, Delhi, BBSR”}**  
**};**

**نکته : اگر کلید مربوطه در collection یافت نشد سبب بروز Exception می شود.**

**How to access**

با استفاده از حلقه بر روی keys یا با استفاده از DictionaryEntry به دلیل این که عناصر به صورت DictionaryEntry ذخیره می شوند.

**foreach (object obj in hashtable.Keys)**  
**{**  
**Console.WriteLine(obj + ” : ” + hashtable[obj]);**  
**}**

**foreach (DictionaryEntry item in hashtable)**  
**{**  
**Console.WriteLine($”Key: {item.Key}, Value: {item.Value}”);**  
**}**

**How to Check the Availability of a key/value Pair in a Hashtable in C#?**

بررسی وجود یا عدم وجود یک Key/value می توان از روش های زیر استفاده کرد.

1. **Contains(object key):** The Contains(object key) method of the Hashtable is used to check whether the Hashtable contains a specific key. The parameter key to locating in the Hashtable object. It returns true if the Hashtable contains an element with the specified key; otherwise, false. If the key is null, then it will throw System.ArgumentNullException.
2. **ContainsKey(object key):** The ContainsKey(object key) method of the Hashtable is used to check if a given key is present in the Hashtable or not. The parameter key to locating in the Hashtable object. If the given key is present in the collection then it will return true else it will return false. If the key is null, then it will throw System.ArgumentNullException.
3. **ContainsValue(object value):** The ContainsValue(object value) Method of the Hashtable class is used to check if a value is present in the Hashtable or not. The parameter value to locate in the hashtable object. If the given value is present in the collection then it will return true else it will return false.

**How to Remove Elements from a Non-Generic Hashtable Collection in C#?**

برای پاک کردن یک object بر اساس کلید آن می توانید از از متد زیر استفاده کنید.

1. **Remove(object key):** This method is used to remove the element with the specified key from the Hashtable. Here, the parameter key specifies the element to remove. It throws the KeyNotfoundException if the specified key is not found in the Hashtable, so check for an existing key using the ContainsKey() method before removing it.

برای پاک کردن تمامی عناصر می توان از متد **Clear** استفاده کرد.

**How to Assign Values to a Hashtable with Indexer in C#?**

برای اضافه کردن مقدار با استفاده از indexer می توان ابتدا کلید و سپس مقدار را مشخص کرد.

Hashtable hashtable = new Hashtable**()**;

hashtable**[**1**]** = "One";

hashtable**[**5**]** = "Five";

hashtable**[**30**]** = "Thirty";

**How to Update a Hashtable in C# using Indexer?**

Hashtable employee = new Hashtable

**{**

**{** "EId", 1001 **}**,

**{** "Name", "James" **}**,

**{** "Salary", 3500 **}**,

**{** "Location", "Mumbai" **}**,

**{** "EmailID", "a@a.com" **}**

**}**;

//Updating the Name and Salary

employee**[**"Name"**]** = "Smith"; //Update value of Name key

employee**[**"Salary"**]** = 5000; //Update value of Salary key

**How to Clone a Non-Generic Hashtable Collection in C#?**

با استفاده از متد Clone می توان یک کپی از collection ایجاد کرد و کپی آن را دریافت کرد.

//Creating a clone Hashtable using Clone method

Hashtable cloneHashtable = **(**Hashtable**)**hashtable.Clone**()**;

**How to Copy a Hashtable to an Existing Array in C#?**

برای کپی یک Hashtable به یک ارایه ی از قبل وجود دارد می توان از متد CopyTo() استفاده کرد

عملیات کپی با پیچیدگی زمان O(n) است که وابسته به تعداد زمان افزایش می یابد.

1. **CopyTo(Array array, int arrayIndex):** The CopyTo method of the Non-Generic Hashtable Collection Class is used to copy hashtable elements to a one-dimensional Array object, starting at the specified index in the array. Here, the parameter array specifies the one-dimensional Array object that is the destination of the DictionaryEntry objects copied from the hashtable. The Array must have zero-based indexing. The arrayIndex parameter specifies the zero-based index in the array at which copying begins. If the parameter array is null, then it will throw ArgumentNullException. If the parameter arrayIndex is less than zero, then it will throw ArgumentOutOfRangeException.

نکته

1. To copy only the keys in the Hashtable, use Hashtable.Keys.CopyTo.
2. To copy only the values in the Hashtable, use Hashtable.Values.CopyTo.

مثال

//Copying the Hashtable to an object array

DictionaryEntry**[]** myArray = new DictionaryEntry**[**hashtable.Count**]**;

hashtable.CopyTo**(**myArray, 0**)**;

Console.WriteLine**(**"\nHashtable Copy Array Elements:"**)**;

**foreach** **(**DictionaryEntry item in myArray**)**

**{**

Console.WriteLine**(**$"Key: {item.Key}, Value: {item.Value}"**)**;

**}**

Object**[]** myObjArrayKey = new Object**[**hashtable.Count**]**;

Object**[]** myObjArrayValue = new Object**[**hashtable.Count**]**;

Console.WriteLine**(**"\nCopyTo Method to Copy Keys:"**)**;

hashtable.Keys.CopyTo**(**myObjArrayKey, 0**)**;

**foreach** **(**var key in myObjArrayKey**)**

**{**

Console.WriteLine**(**$"{key} "**)**;

**}**

Console.WriteLine**(**"\nCopyTo Method to Copy Values:"**)**;

hashtable.Values.CopyTo**(**myObjArrayValue, 0**)**;

**foreach** **(**var key in myObjArrayValue**)**

**{**

Console.WriteLine**(**$"{key} "**)**;

**}**

**Non-Generic Hashtable Collection Class Properties in C#**

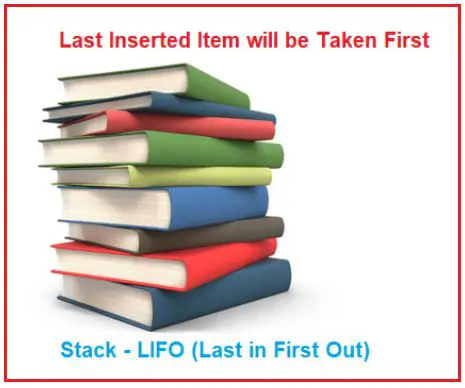
1. **IsFixedSize**: Gets a value indicating whether the System.Collections.Hashtable has a fixed size. It returns true if the Hashtable object has a fixed size; otherwise, false. The default is false.
2. **Keys**: Gets a System.Collections.ICollection containing the keys in the Hashtable. It returns a System.Collections.ICollection containing the keys in the Hashtable.
3. **IsSynchronized**: Gets a value indicating whether access to the Hashtable is synchronized (thread-safe). It returns true if access to the Hashtable is synchronized (thread-safe); otherwise, false. The default is false.
4. **IsReadOnly**: Gets a value indicating whether the Hashtable is read-only. It returns true if the Hashtable object is read-only; otherwise, false. The default is false.
5. **Count**: Gets the number of key/value pairs contained in the Hashtable. It returns the number of key/value pairs contained in the System.Collections.Hashtable.
6. **Values**: Gets a System.Collections.ICollection containing the values in the Hashtable. It returns a System.Collections.ICollection containing the values in the Hashtable.
7. **SyncRoot**: Gets an object that can be used to synchronize access to the Hashtable. It returns an object that can be used to synchronize access to the Hashtable.
8. **comparer**: Gets or sets the System.Collections.IComparer to use for the Hashtable. It returns the System.Collections.IComparer to use for the Hashtable.
9. **hcp**: Gets or sets the object that can dispense hash codes. It returns the object that can dispense hash codes.
10. **EqualityComparer**: Gets the System.Collections.IEqualityComparer to use for the Hashtable. It returns the System.Collections.IEqualityComparer to use for the Hashtable.

# **Non-Generic Stack Collection**

در این نوع از کالکشن روش Last in – First out استفاده می شود برای زمانی کاربرد دارد که ما نیاز به سیستم آخرین عضوی که وارد شده به عنوان اولین عضو خارج شود.

##### **What is Stack in C# and how does it work?**

یک کلاس غیر generic است. که به صورت LIFO کار می کند. اضافه کردن ایتم را psuh و خارج کردن ایتم از کالکشن را pop می گویند.



نکته : stack به صورت generic و هم به صورت non-generic پیاده سازی شده است.

##### **Methods, Properties, and Constructor of Stack Class in C#:**

برای هر دو حالت generic , non-generic در پیاده سازی هر دو رابط های IEnumerable. Icollection , ICloneable

Ctor

1. **Stack():** It is used to initialize a new instance of the Stack class that is empty and has the default initial capacity.
2. **Stack(ICollection col):** It is used to initialize a new instance of the non-generic Stack class that contains elements copied from the specified collection and has the same initial capacity as the number of elements copied. Here, the parameters col specifies the System.Collections.ICollection to copy elements from.
3. **Stack(int initialCapacity):** It is used to initialize a new instance of the System.Collections.Stack class that is empty and has the specified initial capacity or the default initial capacity, whichever is greater. Here, the parameter initialCapacity specifies the initial number of elements that the Stack can contain.

Stack stack = new Stack**()**;

//Adding item to the stack using the push method

stack.Push**(**10**)**;

stack.Push**(**"Hello"**)**;

stack.Push**(**3.14f**)**;

stack.Push**(true)**;

stack.Push**(**67.8**)**;

stack.Push**(**'A'**)**;

//Printing the stack items using foreach loop

**foreach** **(object** item in stack**)**

**{**

Console.WriteLine**(**item**)**;

**}**

##### **How to Remove Elements from a Non-Generic Stack Collection in C#?**

1. **Pop():** This method is used to remove and return the object at the top of the Stack. It returns the Object (element) removed from the top of the Stack.
2. **Clear():** This method is used to remove all objects from the Stack.

##### **How to get the topmost element of a Stack in C#?**

1. **Pop():** This method is used to remove and return the object at the top of the Stack. It returns the Object (element) removed from the top of the Stack. If there is no object (or element) present in the stack and if you are trying to remove an item or object from the stack using the pop() method then it will throw an exception i.e. **System.InvalidOperationException**
2. **Peek():** The peek() method is used to return the object from the top of the Stack without removing it. If there is no object (or element) present in the stack and if you are trying to return an item (object) from the stack using the peek() method then it will throw an exception i.e. **System.InvalidOperationException**

##### **How to check whether an element exists or not in the stack in C#?**

1. **Contains(object obj):** This method is used to determine whether an element is in the Stack. Here, the parameter obj specifies the object or element to locate in the Stack. The value can be null. It returns true if obj is found in the Stack; otherwise, false.

نکته : مرتبه زمانی بررسی وجود یا عدم وجود یک object در stack دارای هزینه مرتبه زمانی O(n) است یعنی با توجه به تعداد زمان بیشتری می برد.

##### **How to Clone the Non-Generic Stack Collection in C#?**

1. **Clone():** This method is used to create and return a shallow copy of a stack object.

shallowCopy کپی که در آن یک object جدید ایجاد می شود و برای عناصر value-type آن ها را مستقیما کپی می کند و برای مقادیری که reference type هستند فقط reference را کپی می کند.

##### **How to copy a stack to an existing array in C#?**

1. **CopyTo(Array array, int index)**: This method is used to copy the Stack elements to an existing one-dimensional Array, starting at the specified array index. Here, the parameter array specifies the one-dimensional array that is the destination of the elements copied from the stack. The Array must have zero-based indexing. The index parameter specifies the zero-based index in the array at which copying begins. If the parameter array is null, then it will throw ArgumentNullException. If the parameter index is less than zero, then it will throw ArgumentOutOfRangeException.
2. **object[]** stackCopy = new **object[**5**]**;
3. stack.CopyTo**(**stackCopy, 0**)**;

##### **Properties of Non-Generic Stack Collection Class in C#**

1. **Count**: It returns the number of elements contained in the Stack.
2. **IsSynchronized**: Gets a value indicating whether access to the Stack is synchronized (thread-safe). It returns true if access to the Stack is synchronized (thread-safe); otherwise, false. The default is false.
3. **SyncRoot**: Gets an object that can be used to synchronize access to the Stack. It returns an object that can be used to synchronize access to the Stack.

نکته : ظرفیت stack با افزودن عنصار اضافه می شود

نکته : اگر count از ظرفیت Stack کمتر باشد عملیات push دارای مرتبه زمانی O(1) بوده و اگر ظرفیت برای اضافه شدن عنصر موجود نباشد افزودن عنصر باعث افزایش ظرفیت شده و در این حالت عملیات دارای مرتبه زمانی O(n) می شود و عملیات pop در هر زمانی دارای مرتبه زمانی O(1) است.

# Non-Generic Queue in C#

ساختار عملکردی این نوع کالکشن به صورت first in – first out است به این معنی که اولین المانی که وارد کالکشن شود به عنوان اولین المان می تواند از صف خارج شود.

**What is a Queue and How Does the Queue Collection Work in C#?**

به صورت FIFO کار میکند به این معنی که اولین چیزی که اضافه می شود اولین چیزی است که حذف می شود.

عملیات افزودن enqueuing

عملیات حذف dequeuing



Mehtods

متد Enqueue برای افزودن عنصر به انتهای صف

متد Dequeue برای حذف قدیمی ترین عنصر صف

Peek برای دریافت قدیمی ترین عنصر بدون حذف از صف

ظرفیت برابر تعداد عناصر اضافه شده است و با افزودن عنصر ظرفیت به صورت خودکار اضافه می شود.

هم به صورت generic , non-generic است و امکان null و duplicate value نیز دارد.

Interfaces

Icollection – Ienumerable – Icloneable

**How to create a Queue in C#?**

1. **Queue():** It is used to initialize a new instance of the Queue class that is empty and has the default initial capacity, and uses the default growth factor.
2. **Queue(ICollection col):** It is used to initialize a new instance of the Queue class that contains elements copied from the specified collection and has the same initial capacity as the number of elements copied and uses the default growth factor. Here, the parameters col specifies the System.Collections.ICollection to copy elements from.
3. **Queue(int capacity):** It is used to initialize a new instance of the Queue class that is empty, has the specified initial capacity, and uses the default growth factor. Here, the parameter capacity specifies the initial number of elements that the Queue can contain.
4. **Queue(int capacity, float growFactor):** It is used to initialize a new instance of the Queue class that is empty, has the specified initial capacity, and uses the specified growth factor. Here, the parameter capacity specifies the initial number of elements that the Queue can contain and the parameter growFactor specifies the factor by which the capacity of the Queue is expanded.

**growFactor**

**مشخص کننده میزان رشد ظرفیت زمانی که به فضای بیشتری نیاز داریم. مشخص می کند با چه سرعتی ظرفیت رشد کند.**

* Default value is 2.0 (meaning capacity doubles when needed)
* Must be between 1.0 and 10.0
* Minimum capacity increase is always 4 slots, regardless of the growth factor

**How to Remove Elements from the Queue Collection in C#?**

1. **Dequeue():** This method is used to remove and return the object at the beginning of the Queue. It returns the object that is removed from the beginning of the Queue.
2. **Clear():** This method is used to remove all objects from the queue.

How to get first element of the queue

1. **Dequeue():** The Dequeue() method of the Queue class is used to Remove and return the object from the beginning of the Queue. If there is no object (or element) present in the Queue and if we are trying to remove an item or object from the Queue using the pop() method then it will throw an exception i.e. **System.InvalidOperationException**
2. **Peek():** The peek() method of the Queue class is used to return the oldest object i.e. the object present at the start of the Queue without removing it. If there is no object (or element) present in the Queue and if we are trying to return an item (object) from the Queue using the peek() method then it will throw an exception i.e. **System.InvalidOperationException**

##### **How to Check Whether an Element Exists or not in the Queue Collection in C#?**

1. **Contains(object obj):** This method is used to determine whether an element is in the Queue. Here, the parameter obj specifies the object or element to locate in the Queue. The value can be null. It returns true if obj is found in the queue; otherwise, false.

**How to Clone the Non-Generic Queue Collection in C#?**

1. **Clone():** This method is used to create and return a shallow copy of a Queue object.

Queue cloneQueue = **(**Queue**)**queue.Clone**()**;

**How to copy a queue to an existing array in C#?**

1. **CopyTo(Array array, int index):** The CopyTo method of the Non-Generic Queue Collection Class in C# is used to copy the System.Collections.Queue elements to an existing one-dimensional System.Array, starting at the specified array index. Here, the parameter array specifies the one-dimensional Array that is the destination of the elements copied from Queue. The Array must have zero-based indexing. The index parameter specifies the zero-based index in the array at which copying begins. If the parameter array is null, then it will throw ArgumentNullException. If the parameter index is less than zero, then it will throw ArgumentOutOfRangeException.
2. **object[]** queueCopy = new **object[**5**]**;
3. queue.CopyTo**(**queueCopy, 0**)**;

# Non-Generic SortedList in C#

از نوع non-generic است و به صورت key/value می باشد یک لیست مرتب که با index و key قابل دسترسی است.

به صورت پیش فرض مرتب سازی به صورت **ascending یا صعودی است.**

**Properties of Non-Generic SortedList Class in C#**

رابط های IEnumerable , ICollection , IDictionary , ICloneable را پیاده سازی

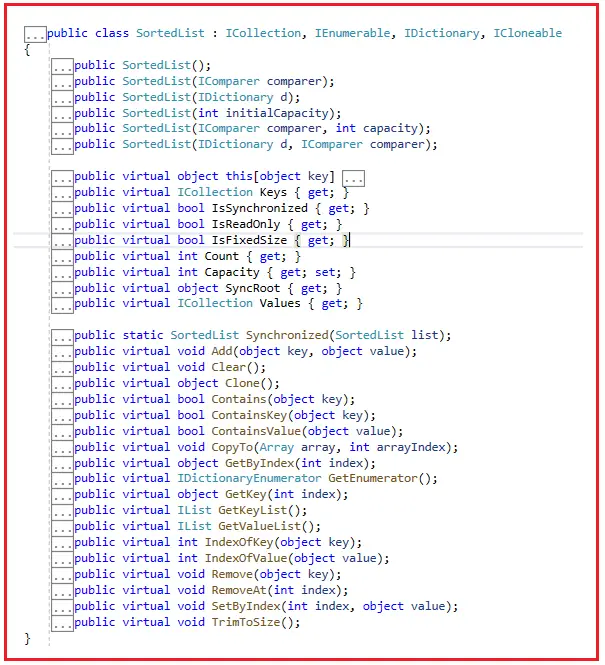
قابلیت دسترسی به عنصر با index و کلید

از 2 آرایه برای ذخیره سازی داده ها استفاده می کند یک آرایه برای نگه داری کلید و دیگری و دیگری برای نگه داری مقدار

نکته : کلید نمی تواند دارای مقدار null باشد.

ظرفیت بر اساس تعداد عناصری است که نگه می دارد. می توان مقادیر یک نوع یا چندین نوع را در خود نگه داری کند

نکته : کلید نمی تواند از نوع مختلف باشد.



1. **SortedList():** It initializes a new instance of the System.Collections.SortedList class that is empty, has the default initial capacity and is sorted according to the IComparable interface implemented by each key added to the System.Collections.SortedList object.
2. **SortedList(IComparer comparer):** It initializes a new instance of the System.Collections.SortedList class that is empty, has the default initial capacity and is sorted according to the specified IComparer interface. The Parameter comparer specifies the System.Collections.IComparer implementation to use when comparing keys. -or- null to use the System.IComparable implementation of each key.
3. **SortedList(IDictionary d):** IT initializes a new instance of the System.Collections.SortedList class contains elements copied from the specified dictionary, has the same initial capacity as the number of elements copied, and is sorted according to the System.IComparable interface implemented by each key. The Parameter d specifies the System.Collections.IDictionary implementation to copy to a new System.Collections.SortedList object.
4. **SortedList(int initialCapacity):** It initializes a new instance of the System.Collections.SortedList class that is empty, has the specified initial capacity, and is sorted according to the System.IComparable interface implemented by each key added to the System.Collections.SortedList object. The Parameter initialCapacity specifies the initial number of elements that the System.Collections.SortedList object can contain.
5. **SortedList(IComparer comparer, int capacity):** It initializes a new instance of the System.Collections.SortedList class that is empty, has the specified initial capacity, and is sorted according to the specified System.Collections.IComparer interface. The Parameter comparer specifies the System.Collections.IComparer implementation to use when comparing keys. -or- null to use the System.IComparable implementation of each key. The Parameter capacity specifies the initial number of elements that the System.Collections.SortedList object can contain.
6. **SortedList(IDictionary d, IComparer comparer):** It initializes a new instance of the System.Collections.SortedList class contains elements copied from the specified dictionary, has the same initial capacity as the number of elements copied, and is sorted according to the specified System.Collections.IComparer interface. The Parameter d specifies the System.Collections.IDictionary implementation to copy to a new System.Collections.SortedList object. The Parameter comparer specifies the System.Collections.IComparer implementation to use when comparing keys. -or- null to use the System.IComparable implementation of each key.

**How to Add Elements into a SortedList in C#?**  
**SortedList sortedList = new SortedList();**  
**sortedList.Add(1, “One”);**  
**sortedList.Add(3, “Three”);**

You can also store a key/value pair in the SortedList using Collection Initializer as follows.  
**SortedList sortedList = new SortedList**  
**{**  
**{ 1, “One” },**  
**{ 3, “Three” }**  
**};**

**How to Remove Elements from a SortedList Collection in C#?**

1. **Remove(object key):** This method is used to remove the element with the specified key from a System.Collections.SortedList object. The parameter key specifies the element to remove.
2. **RemoveAt(int index):** This method is used to remove the element at the specified index of a System.Collections.SortedList object. The parameter index specifies the element to remove. It is 0 based Index.
3. **Clear()**: This method is used to remove all elements from a System.Collections.SortedList object.

**How to check the availability of key/value pairs in a SortedList in C#?**

1. **Contains(object key):** This method is used to determine whether the SortedList object contains a specific key. The parameter key to locate in the SortedList object. It returns true if the SortedList object contains an element with the specified key; otherwise, false. If the key is null, then it will throw System.ArgumentNullException.
2. **ContainsKey(object key):** This method is used to determine whether a SortedList object contains a specific key. The parameter key to locate in the SortedList object. It returns true if the SortedList object contains an element with the specified key; otherwise, false. If the key is null, then it will throw System.ArgumentNullException.
3. **ContainsValue(object value):** This method is used to determine whether a System.Collections.SortedList object contains a specific value. The parameter value to locate in the SortedList object. The value can be null. It returns true if the SortedList object contains an element with the specified value; otherwise, false.

##### **How to Clone the Non-Generic SortedList in C#?**

1. **Clone():** This method is used to create and return a shallow copy of a SortedList object.

Example

SortedList cloneSortedList = **(**SortedList**)**sortedList.Clone**()**;

**CopyTo(Array array, int arrayIndex):** The CopyTo method of the Non-Generic SortedList Collection Class in C# is used to copy SortedList elements to a one-dimensional Array object, starting at the specified index in the array. Here, the parameter array specifies the one-dimensional Array object that is the destination of the DictionaryEntry objects copied from SortedList. The Array must have zero-based indexing. The arrayIndex parameter specifies the zero-based index in the array at which copying begins. If the parameter array is null, then it will throw ArgumentNullException. If the parameter arrayIndex is less than zero, then it will throw ArgumentOutOfRangeException.

**کپی داده ها به همان ترتیبی که ذخیره شده اند اتفاق می افتد و پیچیدگی زمانی برابر است با O(n) که n برابر Count است**

**امکان کپی کردن صرفا key یا value نیز وجود دارد.**

1. **To copy only the keys in the SortedList, use SortedList.Keys.CopyTo.**
2. **To copy only the values in the SortedList, use SortedList.Values.CopyTo.**

##### **When to Use Non-Generic SortedList Collection in C#?**

برای مواقعی که نیاز به تغییر سریع داده ها بر اساس key/value به ترتیب منظم کاربرد دارد.

**نکته: به دلیل این که با هر بار اضافه کردن مقدار به کالکشن یا حذف مقدار بر اساس ماهیت این نوع لازم است که همیشه مرتب باشد پس عملیات مرتب سازی انجام می پذیرد. که با افزایش تعداد عناصر cost بالا می رود.**

باید زمانی از SortedList استفاده کنیم که کالکشن کوچکی نیاز داریم که باید تمام مدت مرتب باشد

مواقعی که دیتا زیاد است باید از مواردی مانند Dictionay , HastSet , List استفاده کنیم

# Advantages and Disadvantages of Non-Generic Collection in C#

* کلاس های non-generic افزایش ظرفیت خودکار دارند زمانی که به آن ها ایتم اضافه می شود و این از مزایای آن ها است. ArrayList, Stack, Queue, SortedList, and Hashtable,…

**مثال**

آرایه ی از نوع non-generic با ظرفیت 3 ایجاد می کنیم سپس 4 ایتم به آن اضافه می کنیم.

ArrayList Numbers = new ArrayList**(**3**)**;

Numbers.Add**(**100**)**;

Numbers.Add**(**200**)**;

Numbers.Add**(**300**)**;

Numbers.Add**(**400**)**;

بدون این که خطای run time , compile time داشته باشیم.

* امکان اضافه یا حذف ایتم از میانه ی کالکشن که در آرایه ها امکان پذیر نیست اما در collection امکان پذیره.

معایب

به دلیل این که loosely type هستند امکان برخورد با خطای runtime وجود دارد. Loosely type به معنی استفاده از نوع object است است به عنوان value .

نه تنها به دلیل این که loosely type سبب exception می شود بلکه به دلیل این که عملیات boxing , unboxing برای object رخ می دهد بر روی performance تاثیر گذار است.

یعنی مقادیر 100 و 200 را به کالکشن می دهیم از نوع int هستند و این نوع نیز value type هست اما نوع object از نوع از نوع reference type است پس بنابراین باید مقادیر به reference type تبدیل شوند و عملیات boxing رخ دهد.

همچنین برای retrieve کردن عناصر لازم است که از نوع object تبدیل به نوع int رخ دهند که عملیات unboxing است.

پس بنابراین باید از کالشکن های بزرگ از این نوع پرهیز شود چون عملیات boxing , unboxing یک عمل هزینه بر است.

# Generic Collections in C#

Generic collection ها برای از بین بردن معایب مربوط به non-generic ها کاربرد دارد در این نوع collection ها باید نوع داده ای که می خواهیم ذخیره کنیم را مشخص کنیم.

Strongly type safe بودن از مزایای این نوع از کالکشن ها است. و فقط یک نوع داده را می توان به کالکشن اضافه کرد.

1. [**List<T>**](https://dotnettutorials.net/lesson/list-collection-csharp/)**:** It represents a strongly typed list of objects that can be accessed by index. Provides methods to search, sort, and manipulate lists. It grows automatically as you add elements to it.
2. [**Dictionary<TKey, Tvalue>**](https://dotnettutorials.net/lesson/dictionary-generic-collection-csharp/)**:** It represents a collection of keys and values.
3. [**SortedSet<T>:**](https://dotnettutorials.net/lesson/generic-sortedset-collection-class-in-csharp/)It represents a collection of objects that are maintained in sorted order.
4. [**SortedDictionary<TKey, TValue>**](https://dotnettutorials.net/lesson/generic-sorteddictionary-collection-class-in-csharp/)**:** It represents a collection of key/value pairs that are sorted on the key.
5. [**LinkedList<T>**](https://dotnettutorials.net/lesson/generic-linkedlist-collection-class-in-csharp/)**:** It represents a doubly linked list.

از مزایای آن ها می توان به عملکرد مناسب به دلیل عدم نیاز به boxing , unboxing می توان اشاره کرد. و همچنین از بروز خطا های run time به دلیل نوع داده جلوگیری می کند.

**List<T> Class in C#**

می تواند مقدار تکراری داشته باشد

**HashSet<T> Class in C#**

فقط مقادیر unique می گیرد و امکان ذخیره سازی مقادیر تکراری را ندارد. می توانید مقدار تکراری اضافه کنید اما در نهایت خود کالکشن مقدار تکراری را حذف می کند

**SortedSet<T> Class in C#:**

عدم امکان ذخیره سازی مقادیر تکراری و مناسب نگه داری مقادیر unique

**SortedDictionary<TKey, TValue> Class in C#**

کلید باید غیر تکراری باشد. و ترتیب ascending یا صعودی است.

**LinkedList<T> Class in C#:**

اجازه ی افزودن و حذف عنصر با سرعت بالا را دارد. می تواند مقادیر تکراری نیز داشته باشد.

این امکان را به ما می دهد که قبل یا بعد از آخرین index عناصری را اضافه کنیم.

قابلیت AddFirst , AddLast را نیز دارد

linkedList.AddLast**(**"Three"**)**;

linkedList.AddLast**(**"Four"**)**;

linkedList.AddFirst**(**"Five"**)**;

# **Generic Constraints in C#**

Constraint به این معنا است که می توان محدودیت های برای محدود کردن type که می تواند استفاده شود کاربرد دارد.

به شما خطای compile-time می دهد در صورتی که بخواهید نوعی غیر از نوع مشخص شده توسط constraint به generic الحاق کنید.

می توانید یک یا چند constraint را بر روی یک generic با استفاده از کلمه کلیدی where اعمال کنید.

اگر می خواهید که generic شما فقط یک نوع خاص را دریافت کند باید از constraint ها استفاده کنید و نوع قابل قبول را مشخص کنید. و اگر نوع درست را استفاده نکنید باعث خطای compile time می شود.

##### **Types of Generic Constraints in C#**

1. **where T: struct** => The type argument must be non-nullable value types such as primitive data types int, double, char, bool, float, etc. The struct constraint can’t be combined with the unmanaged constraint.
2. **where T: class** => The type argument must be a reference type. This constraint can be applied to any class (non-nullable), interface, delegate, or array type in C#.
3. **where T: new()** => The type argument must be a reference type that has a public parameterless (default) constructor.
4. **where T: <base class name>** => The type of argument must be or derive from the specified base class.
5. **where T: <interface name>** => The type argument must be or implement the specified interface. Also, multiple interface constraints can be specified.
6. **where T: U** => The type argument supplied for must be or derive from the argument supplied for U. In a nullable context, if U is a non-nullable reference type, T must be a non-nullable reference type. If U is a nullable reference type, T may be either nullable or non-nullable.

T:Class به این معنی است که فقط مقادیر reference type قابل پذیرش هستند .

 In C#, the string is a reference type.  
**GenericClass<string> stringClass = new GenericClass<string>();**

**X**

The following statement will give you a compile-time error as int is a value type, not a reference type.  
**GenericClass<int> intClass = new GenericClass<int>();**

Struct به این معنی است که مقادیر Value type قابل پذیرش هستند.

create an instance of GenericClass by passing value-type arguments as follows.  
**GenericClass<int> intClass = new GenericClass<int>();**

**X**

The following statement will give you a compile-time error as the string is a reference type, not a value type.  
**GenericClass<string> stringClass = new GenericClass<string>();**

#### **where T: BaseClass Generic Constraint in C#**

به این معنی است که می توان یک کلاس مشخص را به عنوان constraint مشخص کرد و تمامی کلاس های که از آن کلاس مشخص ارث برده اند نیز آن استفاده کنند

**public** **class** GenericClass**<**T**>** **where** T : BaseEmployee

**{**

**public** T Message;

**public** **void** GenericMethod**(**T Param1, T Param2**)**

**{**

Console.WriteLine**(**$"Message: {Message}"**)**;

Console.WriteLine**(**$"Param1: {Param1}"**)**;

Console.WriteLine**(**$"Param2: {Param2}"**)**;

**}**

**}**

**public** **class** BaseEmployee

**{**

**}**

**public** **class** Employee : BaseEmployee

**{**

**public** **string** Name **{** **get**; **set**; **}**

**}**

**public** **class** Customer

**{**

**public** **string** Name **{** **get**; **set**; **}**

**}**

 It works fine because Employee is derived from the BaseEmployee class.  
**GenericClass<Employee> employee = new GenericClass<Employee>();**

The following statement will give you a compile-time error as the Customer class is not derived from the BaseEmployee type.  
**GenericClass<Customer> customer = new GenericClass<Customer>();**

Interface constraint

می توان به عنوان محدودیت نوع interface را نیز مشخص کرد که در این حالت فقط کلاسی می تواند در generic باشد که فرآیند ساخت نیز مشارکت داشته اند.

**public** **class** GenericClass**<**T**>** **where** T : IEmployee

**{**

**public** T Message;

**public** **void** GenericMethod**(**T Param1, T Param2**)**

**{**

Console.WriteLine**(**$"Message: {Message}"**)**;

Console.WriteLine**(**$"Param1: {Param1}"**)**;

Console.WriteLine**(**$"Param2: {Param2}"**)**;

**}**

**}**

let us create an instance of GenericClass bypassing Employee type arguments as follows. It works fine because the Employee class implements the IEmployee interface.  
**GenericClass<Employee> employee = new GenericClass<Employee>();**

The following statement will give you a compile-time error as the Customer class does not implement the IEmployee interface.  
**GenericClass<Customer> customer = new GenericClass<Customer>();**

#### **where T: U Generic Constraint in C#**

Here, the type argument supplied must be or derive from the argument supplied for U. In a nullable context, if U is a non-nullable reference type, T must be a non-nullable reference type. If U is a nullable reference type, T may be either nullable or non-nullable. So, in this constraint, there are two Type Arguments i.e. T and U. U can be an interface, abstract class, or simple class. T must inherit or implements the U class. For a better understanding, please have a look at the below code.

**using** *System;*

**namespace** *GenericsDemo*

**{**

**public** **class** GenericClass**<**T, U**>** **where** T : U

**{**

**public** T Message;

**public** **void** GenericMethod**(**T Param1, U Param2**)**

**{**

Console.WriteLine**(**$"Message: {Message}"**)**;

Console.WriteLine**(**$"Param1: {Param1}"**)**;

Console.WriteLine**(**$"Param2: {Param2}"**)**;

**}**

**}**

**}**

As you can see in the above code, here we have used **where T: U** constraint which allows the type (T) that must inherit or implements the U class. Now, let us create one interface and two more classes as follows.

**namespace** *GenericsDemo*

**{**

**public** **interface** IEmployee

**{**

**}**

**public** **class** Employee : IEmployee

**{**

**public** **string** Name **{** **get**; **set**; **}**

**}**

**public** **class** Customer

**{**

**public** **string** Name **{** **get**; **set**; **}**

**}**

**}**

As you can see in the above code, the Employee class implements the IEmployee interface. On the other hand, the Customer class is not implementing the IEmployee interface. Now, Let’s create an instance of Genericclass bypassing Employee and IEmployee as type arguments for T and U as follows. It works fine because the Employee class implements the IEmployee interface.

**GenericClass<Employee, IEmployee> employeeGenericClass = new GenericClass<Employee, IEmployee>();**

The following statement will give you a compile-time error as the Customer class does not implement the IEmployee interface i.e. T does not implement U.  
**GenericClass<Customer, IEmployee> customerGenericClass = new GenericClass<Customer, IEmployee>();**

#### **Multiple Generic Constraints in C#:**

In C# generics, it is also possible to apply multiple constraints on generic classes based on our requirements. Let us understand this with an example. In the below example, we are creating the generic class with two constraints. The first constraint specifies that the T parameter must be a reference type whereas the second constraint specifies that the X parameter must be a value type.

**public** **class** GenericClass**<**T, X**>** **where** T: **class** **where** X: struct

**{**

**public** T Message;

**public** **void** GenericMethod**(**T Param1, X Param2**)**

**{**

Console.WriteLine**(**$"Message: {Message}"**)**;

Console.WriteLine**(**$"Param1: {Param1}"**)**;

Console.WriteLine**(**$"Param2: {Param2}"**)**;

**}**

**}**

**class** Program

**{**

**static** **void** Main**()**

**{**

GenericClass**<string**, **int>** multipleGenericConstraints = new GenericClass**<string**, **int>()**;

multipleGenericConstraints.Message = "Good Morning";

multipleGenericConstraints.GenericMethod**(**"Anurag", 100**)**;

Console.ReadKey**()**;

**}**

**}**

# Generic List Collection in C#

لیستی از object ها را ارائه می دهد که به صورت strongly type است به دلیل generic بودن و قابل دسترسی با integer indexer که از مقدار 0 شروع می شود.

نکته : به صورت خودکار با افزودن داده ظرفیت لیست اضافه می شود.

نکته : به دلیل پیاده سازی IList interface می توان از indexer برای دسترسی به اعضای کاکلشن استفاده کرد.

**How to Insert Elements at a Specific Position**

برای درج مقدار در specific position در لیست می توان از متد های زیر استفاده کرد.

**Insert(int index, T item):**

**InsertRange(int index, IEnumerable<T> collection)**

**How to Remove Elements from a Generic List Collection in C#?**

1. **Remove(T item):** This method is used to remove the first occurrence of a specific object from the Generic List. Here, the parameter item specifies the object to remove from the Generic List. It returns true if the item is successfully removed; otherwise, false. This method also returns false if the item was not found in the Generic List.
2. **RemoveAll(Predicate<T> match):** This method is used to remove all the elements that match the conditions defined by the specified predicate. Here, the parameter match specifies the predicate delegate that defines the conditions of the elements to remove. It returns the number of elements removed from the Generic List. If the parameter match is null, then it will throw ArgumentNullException.
3. **RemoveAt(int index):** This method is used to remove the element at the specified index of the Generic List. Here, the parameter index is the zero-based index of the element to remove. If the index is less than 0 or the index is equal to or greater than Generic List Count, then it will throw ArgumentOutOfRangeException.
4. **RemoveRange(int index, int count):** This method is used to remove a range of elements from the Generic List. Here, the parameter index is the zero-based starting index of the range of elements to remove and the parameter count is the number of elements to remove. If the index is less than 0 or the count is less than 0, then it will throw ArgumentOutOfRangeException. If the index and count do not denote a valid range of elements in the Generic List, then it will throw ArgumentException.
5. **Clear():** This method is used to remove all elements from the Generic List.

##### **How to Copy an Array to a List in C#?**

// Create new array with 3 elements.

**string[]** array = new **string[]** **{** "INDIA", "USA", "UK" **}**;

// Copy the array to a List.

List**<string>** copiedList = new List**<string>(**array**)**;

**How to Find Element in a Generic List Collection in C#?**

1. **Find():** The Find() method is used to find the first element from a list based on a condition that is specified by a lambda expression.
2. **FindLast():** The FindLast() method is used to search for an element that matches the conditions specified by a predicate. If it found any elements with that specified condition then it returns the Last matching element from the list.
3. **FindAll():** The FindAll() method is used to retrieve all the elements from a list that matches the conditions specified by a predicate.
4. **FindIndex():** The FindIndex() method is used to return the index position of the first element that matches the conditions specified by a predicate. The point that you need to remember is the index here in generic collections is zero-based. This method returns -1 if an element that matches the specified conditions is not found. There are 2 other overloaded versions of this method is available, one of the overload versions allows us to specify the range of elements to search within the list.
5. **FindLastIndex():** The FindLastIndex() Method searches for an element in the list that matches the condition specified by the lambda expression and then returns the index of the last occurrence of the item within the list. There are 2 other overloaded versions of this method is available, one of the overload versions allows us to specify the range of elements to search within the list.
6. **Exists():** The Exists() method is used to check or determine whether an item exists or not in a list based on a condition. If the item exists then it will return true else it will return false.
7. **Contains():** The Contains() method is used to determine whether the specified item exists or not in the list. If the specified item exists then it will return true else return false.

**Generic List Class important methods in C#:**

1. **TrueForAll():** This method returns true or false depending on whether every element in the list matches the conditions defined by the specified predicate.
2. **AsReadOnly():** This method returns a read-only wrapper for the current collection. Use this method, if you don’t want the client to modify the collection i.e. add or remove any elements from the collection. The ReadOnlyCollection will not have methods to add or remove items from the collection. We can only read items from this collection.
3. **TrimExcess():** This method sets the capacity to the actual number of elements in the List if that number is less than a threshold value.
4. Console.WriteLine**(**$"\nList Capacity Before invoking TrimExcess: {listEmployees.Capacity}"**)**;
5. // Invoke TrimExcess() to set the capacity to the actual number of elements in the List
6. listEmployees.TrimExcess**()**;
7. Console.WriteLine**(**$"\nList Capacity After invoking TrimExcess: {listEmployees.Capacity} "**)**;

**How to Sort a List of Simple Types in C#?**

Then we just need to invoke the Sort() method on numbersList collection as shown below  
**numbersList.Sort();**

If you want the data to be retrieved in descending order, then use the Reverse() method on the list instance as shown below.  
**numbersList.Reverse();**

**نکته این دو متد بر روی داده های از نوع Complex قابل استفاده نیست و InvalidOperationException می دهد.**

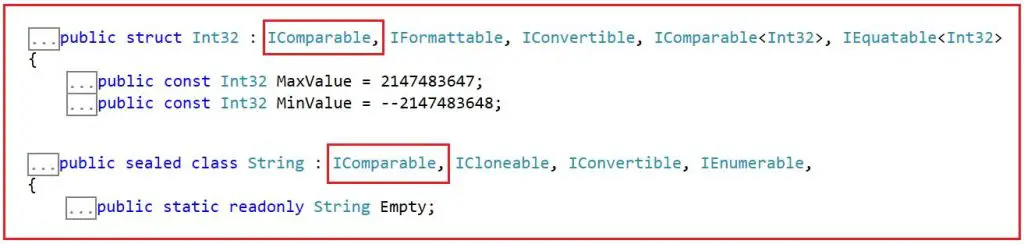
**مثلا بر اساس کدام property باید sort کند در این حالت باید IComparable را پیاده سازی کنیم که بعدا شرح داده می شود.**

**اما چطور در int , dobuble , string کار میکند sort ؟**

**به دلیل این که رابط Icomparable را این نوع ها پیاده سازی کرده اند.**

**لیست مقدار null برای Reference type ها را قبول می کند. مقادیر تکراری نیز می پذیرد.**

**حداکثر 2 میلیارد object را می پذیرد. در سیستم 64 بیتی**

****

# **How to Sort a List of Complex Type in C#**

راه های مختلف مرتب سازی در لیست ها

1. **Sort():** This method is used to sort the elements in the entire Generic List using the default comparer.
2. **Sort(IComparer<T> comparer):** This method is used to sort the elements in the entire Generic List using the specified comparer.
3. **Sort(Comparison<T> comparison):** This method is used to sort the elements in the entire Generic List using the specified System.Comparison.
4. **Sort(int index, int count, IComparer<T> comparer):** This method is used to sort the elements in a range of elements in a Generic List using the specified comparer

برای نوع های داده ساده مثل int کار ما بسیار ساده است کافی است از متد sort استفاده کنیم. یا نوع داده string

Reverse() برای زمانی که بخواهیم به صورت descending مرتب سازی انجام شود.

برای نوع های complex اگر sort استفاده شود سبب بروز خطای runtime ex می شود.

**نکته: دلیل بروز خطا این است که .net نمی داند که با استفاده از کدام یک از property ها عمل sort را انجام دهد.**

**راه حل :**

پیاده سازی رابط Icomparable مواردی که قابل sort است این رابط را پیاده سازی کرده اند.

بعد از پیاده سازی این رابط لازم است متد CompareTo را پیاده سازی کرد برای complex type که مقدار int برمیگرداند.

1. **Return value greater than ZERO** – The current instance is greater than the object being compared with.
2. **Return value less than ZERO** – The current instance is less than the object being compared with.
3. **The Return value is ZERO** – The current instance is equal to the object being compared with.

مثال

**public** **class** Employee : IComparable**<**Employee**>**

**{**

**public** **int** ID **{** **get**; **set**; **}**

**public** **string** Name **{** **get**; **set**; **}**

**public** **string** Gender **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

**public** **int** CompareTo**(**Employee obj**)**

**{**

**if** **(**this.Salary **>** obj.Salary**)**

**{**

**return** 1;

**}**

**else** **if** **(**this.Salary **<** obj.Salary**)**

**{**

**return** -1;

**}**

**else**

**{**

**return** 0;

**}**

**}**

**}**

اگر بخواهید برای نوع داده string پیاده سازی را انجام دهید از روش زیر استفاده کنید.

**public** **class** SortByName : IComparer**<**Employee**>**

**{**

**public** **int** Compare**(**Employee x, Employee y**)**

**{**

**return** x.Name.CompareTo**(**y.Name**)**;

**}**

**}**

SortByName sortByName = new SortByName**()**;

listEmployees.Sort**(**sortByName**)**;

Console.WriteLine**(**"\nEmployees After Sorting"**)**;

**foreach** **(**Employee employee in listEmployees**)**

**{**

Console.WriteLine**(**"ID = {0}, Name = {1}, Gender = {2}, Salary = {3}",

employee.ID, employee.Name, employee.Gender, employee.Salary**)**;

روش دیگری که برای مرتب سازی وجود دارد استفاده از Linq است

listEmployees = listEmployees.OrderBy**(**x =**>** x.Name**)**.ToList**()**;

# **Comparison Delegate in C# with Examples**

این دلیگیت متدی را ارائه می دهد که 2 object را با یکدیگر مقایسه می کند که از یک نوع هستند. که پارامتر x شی اول و y شی دوم و t نوع آنها است.

یک int علامت دار signed برمیگرداند که نشان دهنده ارتباط بین 2 object است.

1. Return value greater than ZERO – x is greater than y.
2. Return value less than ZERO – x is less than y
3. The Return value is ZERO – x equals y

//Create an instance of the Comparison Delegate

Comparison**<**Employee**>** employeeComparer = new Comparison**<**Employee**>(**CompareEmployees**)**;

//Passing Comparison Delegate as an argument to the Sort method

listEmployees.Sort**(**employeeComparer**)**;

Console.WriteLine**(**"\nEmployees After Sorting"**)**;

**foreach** **(**Employee employee in listEmployees**)**

**{**

Console.WriteLine**(**$"ID = {employee.ID}, Name = {employee.Name}, Gender = {employee.Gender}, Salary = {employee.Salary}"**)**;

**}**

تعریف متدی که بهdelegate دادیم.

**private** **static** **int** CompareEmployees**(**Employee e1, Employee e2**)**

**{**

//Sorting the Employees Based on Name

**return** e1.Name.CompareTo**(**e2.Name**)**;

**}**

**public** **class** Employee

**{**

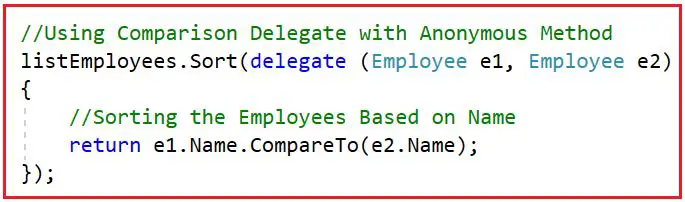
**public** **int** ID **{** **get**; **set**; **}**

**public** **string** Name **{** **get**; **set**; **}**

**public** **string** Gender **{** **get**; **set**; **}**

**public** **int** Salary **{** **get**; **set**; **}**

##### **Comparison Delegate with Anonymous Method in C#:**



//Using Comparison Delegate with Anonymous Method

listEmployees.Sort**(delegate** **(**Employee e1, Employee e2**)**

**{**

//Sorting the Employees Based on Name

**return** e1.Name.CompareTo**(**e2.Name**)**;

**})**;

##### **Comparison Delegate with Lambda Expression in C#**

//Using Lambda Expression

listEmployees.Sort**((**e1, e2**)** =**>** **{** **return** e1.Name.CompareTo**(**e2.Name**)**; **})**;

# Dictionary Collection Class in C#

یک ساختار داده ای است که مجموعه ای key , value ها را به صورت pair یا جفت ارائه می دهد.

Dictionary بسیار شبیه به ساختار Hashtable است. با این تفاوت که زمانی که object از این جنس ایجاد می کنیم نیاز است که type را برای key و Value مشخص کنیم.

فضای dictionary به صورت dynamic افزایش می یابد.

نکته: کلید در دیکشنری نمی تواند null باشد در صورتی که type آن reference باشد.

نکته: تمامی کلید ها باید unique باشد و کلید تکراری سبب بروز Ex می شود.

ظرفیت برابر است با تعداد عناصری که در آن نگه داری می شود.

Overload های مربوط به ctor

1. **Dictionary():** It initializes a new instance of the Generic Dictionary class that is empty, has the default initial capacity and uses the default equality comparer for the key type.
2. **Dictionary(IDictionary<TKey, TValue> dictionary)**: It initializes a new instance of the Generic Dictionary class that contains elements copied from the specified System.Collections.Generic.IDictionary and uses the default equality comparer for the key type.
3. **Dictionary(IEnumerable<KeyValuePair<TKey, TValue>> collection):** It initializes a new instance of the Generic Dictionary class that contains elements copied from the specified System.Collections.Generic.IDictionary and uses the default equality comparer for the key type.
4. **Dictionary(IEqualityComparer<TKey>? comparer):** It initializes a new instance of the Generic Dictionary class that is empty, has the default initial capacity, and uses the specified System.Collections.Generic.IEqualityComparer.
5. **Dictionary(int capacity):** It initializes a new instance of the Generic Dictionary class that is empty, has the specified initial capacity and uses the default equality comparer for the key type.
6. **Dictionary(IDictionary<TKey, TValue> dictionary, IEqualityComparer<TKey>? comparer):** It initializes a new instance of the Generic Dictionary class that contains elements copied from the specified System.Collections.Generic.IDictionary and uses the specified System.Collections.Generic.IEqualityCompare.
7. **Dictionary(int capacity, IEqualityComparer<TKey>? comparer):** It initializes a new instance of the Generic Dictionary class that is empty, has the specified initial capacity, and uses the specified System.Collections.Generic.IEqualityComparer.
8. **Dictionary(SerializationInfo info, StreamingContext context):** It initializes a new instance of the Generic Dictionary class with serialized data.

**Using Key to access Dictionary<TKey, TValue> Collection in C#:**

**dictionaryCountries[“UK”]**

**Using for loop to Access Dictionary<TKey, TValue> Collection in C#:**

**for (int i = 0; i < dictionaryCountries.Count; i++)**  
**{**  
**string key = dictionaryCountries.Keys.ElementAt(i);**  
**string value = dictionaryCountries[key];**  
**}**

چگونه بررسی کنیم که یک key یا value موجود است؟

1. **ContainsKey(TKey key):** The ContainsKey(TKey key) method of the Dictionary is used to check if the given key is present in the Dictionary or not. The parameter key to locating in the Dictionary object. If the given key is present in the collection, then it will return true else it will return false. If the key is null, then it will throw System.ArgumentNullException.
2. **ContainsValue(TValue value):** The ContainsValue(TValue value) Method of the Dictionary class is used to check if the given value is present in the Dictionary or not. The parameter value to locate in the Dictionary object. If the given value is present in the collection, then it will return true else it will return false.

نحوه پاک کردن یک المان؟

1. **Remove(TKey key):** This method is used to remove the element with the specified key from the Dictionary collection. Here, the parameter key specifies the element to remove. It throws KeyNotfoundException if the specified key is not found in the Dictionary, so check for an existing key using the ContainsKey() method before removing it.
2. **Clear():** This method is used to remove all elements i.e. all the keys and values from the Dictionary object.

چگونه به صورت efficient عناصر را پیمایش کنیم ؟

با استفاده از متد های AsParallel و ForAll

**dictionaryCountries.AsParallel()**

**.ForAll(entry => Console.WriteLine(entry.Key + " : " + entry.Value));**

نحوه تخصیص با استفاده از indexer

**dictionary[key] = value;**

**نکته : نحوه بروزرسانی مقدار نیز مشابه روش قبل است.**

**نکته:** می توان برای Value از complexType ها نیز استفاده کرد.

**TryGetValue() method of Dictionary Class in C#?**

متد مهمی است که کلید و مقدار را می گیرد و در صورتی که برای کلید مقدار موجود باشد. در out آن مقدار را می دهد و مقدار true را برمیگرداند. یک راه حل امن برای بررسی وجود یک مقدار در کالکشن است که سبب عدم مواجه با ex می شود.

Student std105;

**if** **(**dictionaryStudents.TryGetValue**(**105, out std105**))**

**{**

Console.WriteLine**(**"\nStudent with Key = 102 is found in the dictionary"**)**;

Console.WriteLine**(**$"ID: {std105.ID}, Name: {std105.Name}, Branch: {std105.Branch}"**)**;

**}**

**else**

**{**

Console.WriteLine**(**"\nStudent with Key = 105 is not found in the dictionary"**)**;

**}**

**How to Convert an Array to a Dictionary in C#?**

با استفاده از متد ToDictionary می توان یک آرایه را به collection تبدیل کرد. کلید و مقدار میگیرد و تمام.

Student**[]** arrayStudents = new Student**[**3**]**;

arrayStudents**[**0**]** = new Student**()** **{** ID = 101, Name = "Anurag", Branch = "CSE" **}**;

arrayStudents**[**1**]** = new Student**()** **{** ID = 102, Name = "Mohanty", Branch = "CSE" **}**;

arrayStudents**[**2**]** = new Student**()** **{** ID = 103, Name = "Sambit", Branch = "ETC" **}**;

Dictionary**<int**, Student**>** dictionaryStudents = arrayStudents.ToDictionary**(**std =**>** std.ID, std =**>** std**)**;

**How to get all the keys and Values of a Dictionary in C#?**

**foreach** **(int** key in dictionaryStudents.Keys**)**

**{**

Console.WriteLine**(**key + " "**)**;

**}**

**foreach** **(int** key in dictionaryStudents.Keys**)**

**{**

var student = dictionaryStudents**[**key**]**;

Console.WriteLine**(**$"Key: {key}, ID: {student.ID}, Name: {student.Name}, Branch: {student.Branch}"**)**;

**}**

**نکته:** باززیابی مقدار با استفده از کلید بسیار سریع است. پیچیدگی زمانی O(1) نزدیک است. به دلیل این که این کالکشن از پیاده سازی HashTable استفاده کرده است. سرعت بازیابی بسته به سرعت الگوریتم هش است.

سریع ترین را پیدا کردن مقدار در این نوع استفاده از key است. و حتما باید unique باشد.

# Conversion Between Array List and Dictionary in C#

برای تبدیل انواع کالکشن ها به یک دیگر می تواندی از متد های extention مربوط به آن نوع کالکشن استفاده کنید.

1. **Convert an array to a List – Use ToList() method**
2. **Convert a list to an array – Use ToArray() method**
3. **Convert a List to a Dictionary – Use ToDictionary() method**
4. **Convert an array to a Dictionary – Use ToDictionary() method**
5. **Convert a Dictionary to an array – Use ToArray() method on the Values Property of the dictionary object**
6. **Convert a Dictionary to a List – Use the ToList() method on the Values Property of the dictionary object**

# List vs Dictionary in C#

هر دو دارای Random access data structures هستند.

Dictionary بر پایه hash table است به این معنی که از hash lookup استفاده می کند که یک الگوریتم کارآمد برای look up است. از الگوریتم hash برای جستجو استفاده می کند.

اما List در جستجو نیاز به برسی عنصر به عنصر المان ها است تا زمانی که به نتیجه برسد.

Details

دیکشنری از الگوریتم hash برای جست و جو استفاده می کند. ابتدا مقدار hash را برای کلید محاسبه می کند و در یک bucket مقادیر را ذخیره می کند و هر عنصر موجود در bucket نیاز به بررسی equality برای پیدا کردن مقدار دارد.

List در جستجو برای مقادیری که در نواحی ابتدای list هستند از دیکشنری سریع تر است. و زمانی که حجم داده زیاد باشد سرعت جستجوی کمی خواهد داشت.

O(n) برای لیست و دیکشنری O(1) پیچیدگی زمانی جستجو را دارد.

دیکشنری نیازمند unique key است.

# Generic Stack Collection Class in C#

به صورت Lifo کار میکند و هم حالت generic دارد هم حالت non generic که بهتر است از generic به دلیل type safe بودن و عدم رخ داد boxing – unboxing که دارای هزینه است استفاده شود.

کاربرد : به طور مثال زمانی که بخواهیم در یک لیست کتاب ها جدید ترین کتابی که لیست اضافه شده را اول برداریم بهر است از این روش استفاده کنیم چون آخرین عصنر وارد شده اولین عنصری است که خارج می شود.

آخرین عنصر اضافه شده اولین عنصری است که حذف می شود.

Push اضافه کردن عنصر به stack

Pop حذف عنصر از stack

ظرفیت stack با افرایش تعداد عناصر افرایش می یابد. همچنین قابلیت دریافت مقدار null را نیز به عنوان یک داده مجاز reference type نیز دارا است.

نکته مهم

تنها collection های قابلیت initializer را دارا هستند که متد ADD در آن ها موجود است به دلیل این که stack فاقد متد Add است پس نمی توان collection initializer را انجام داد سبب compile time error می شود.

بلکه قابلیت push , pop را دارا است.

متد های مهم

Pop

Push

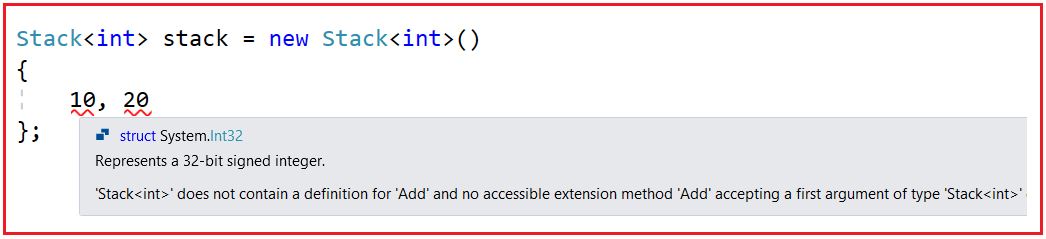
Peek

Clear

Contains

CopyTo

امکان ایجاد stack از complex datatype نیز وجود دارد.



در حالت های non generic به دلیل امکان بروز خطای type mismatch وجود دارد به همین دلیل بهتر است از این نوع که type safe است استفاده کنیم.

Stack برای مواقعی که می خواهیم داده موقت بعد از بازیابی از حافظه حذف شود کاربرد دارد.

مقادیر تکرای در Stack قابل پذیرش هستند.

# Generic HashSet Collection Class in C#

برای ذخیره کردن عناصری که unique هستند کاربرد دارد و قابلیت duplicate element را ندارد.

Performance مربوط به hashSet ها از لیست بسیار بهتر است.

**Constractor overloads**

1. **public HashSet():** It initializes a new instance of the System.Collections.Generic.HashSet class that is empty and uses the default equality comparer for the set type.
2. **public HashSet(IEnumerable<T> collection):** It initializes a new instance of the System.Collections.Generic.HashSet class that uses the default equality comparer for the set type, contains elements copied from the specified collection and has sufficient capacity to accommodate the number of elements copied.
3. **public HashSet(IEqualityComparer<T> comparer):** It initializes a new instance of the System.Collections.Generic.HashSet class that is empty and uses the specified equality comparer for the set type.
4. **public HashSet(int capacity):** It initializes a new instance of the System.Collections.Generic.HashSet class that is empty, but has reserved space for capacity items and uses the default equality comparer for the set type.
5. **public HashSet(IEnumerable<T> collection, IEqualityComparer<T> comparer):** It initializes a new instance of the System.Collections.Generic.HashSet class that uses the specified equality comparer for the set type, contains elements copied from the specified collection and has sufficient capacity to accommodate the number of elements copied.
6. **public HashSet(int capacity, IEqualityComparer<T> comparer):** It initializes a new instance of the System.Collections.Generic.HashSet class that uses the specified equality comparer for the set type, and has sufficient capacity to accommodate capacity elements.
7. **protected HashSet(SerializationInfo info, StreamingContext context):** It initializes a new instance of the System.Collections.Generic.HashSet class with serialized data.

نکته: اگر به HashSet مقادیر duplicate اضافه کنید مقدار را ignore می کند و اضافه نمی کند و به این صورت مقادیر را maintain می کند و همچنین در صورت اضافه کردن مقادیر تکراری exception نمی دهد.

HashSet**<string>** hashSetCountries = new HashSet**<string>()**;

//Adding Elements to HashSet using Add Method

hashSetCountries.Add**(**"INDIA"**)**;

hashSetCountries.Add**(**"USA"**)**;

قابلیت collection initializer را نیز دارا است به دلیل وجود متد Add.

**How to Remove Elements from a Generic HashSet<T> Collection in C#?**

1. **Remove(T item):** This method is used to remove the specified element from a HashSet object. Here, the parameter item specifies the element to remove. It returns true if the element is successfully found and removed; otherwise, false. This method returns false if the item is not found in the System.Collections.Generic.HashSet object.
2. **RemoveWhere(Predicate<T> match):** This method is used to remove all elements that match the conditions defined by the specified predicate from a HashSet collection. It returns the number of elements that were removed from the HashSet collection. Here, the parameter match specifies the Predicate delegate that defines the conditions of the elements to remove.
3. **Clear():** This method is used to remove all elements from a HashSet object.

//Remove Element from HashSet Using RemoveWhere() method where element length is > 3

**int** NumberOfElementRemoved = hashSetCountries.RemoveWhere**(**x =**>** x.Length **>** 3**)**;

**Set Operations on Generic HashSet<T> Collection Class in C#**

**UnionWith(IEnumerable<T> other):**

برای ایجاد تغییر در current hashSet کاربرد دارد به این صورت که می تواند شامل تمامی مقادیری که در خود و مجموعه مشخص شده یا هر دو است را شامل باشد. در صورت null بودن بروز exception می کند.

مقادیری که در هر دو باشند. بدون تکرار اجتماع می کند.

**IntersectWith(IEnumerable<T> other):**

برای تغییر در hashSet جاری به گونه ای که فقط شامل عناصر موجود در آن شی و مجموعه مشخص شده باشد.

اشتراک گیری مقادیری که در هر دو موجود باشند را می دهد

**ExceptWith(IEnumerable<T> other):**

برای حذف تمام عناصر مشخص شده از کالکشن جاری کاربرد دارد.

مقادیری که در کالکشن A وجود دارند که در b نیز هستند را از a حذف می کند تکراری های a حذف می شوند.

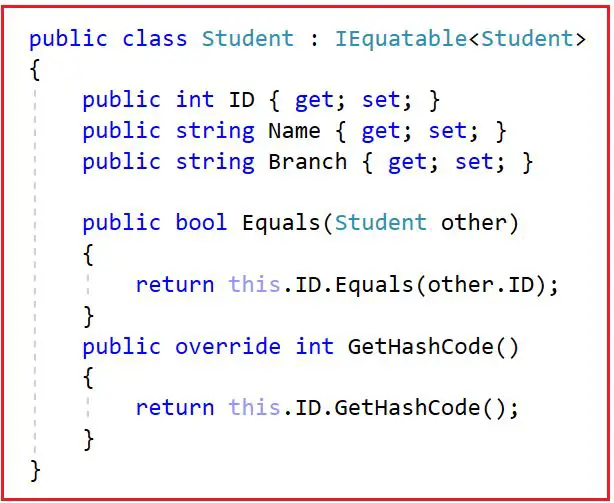
**SymmetricExceptWith(IEnumerable<T> other)**

برای تغییر در collection جاری به صورتی که فقط شامل عناصر موجود در آن شی یا مجموعه مشخص شده باشد.

مقادیر غیر مشترک را به ما برمیگرداند.

پیچیدگی زمانی o(n) برای دسترسی به عناصر در آرایه و در hashSet برابر O(1) که نشان دهنده سرعت خوب برای جستجو است.

نکته : برای استفاده از hashSet ها با ComplexDataType ها همان طور که گفته شده این نوع کالکشن برای نگاه داری مقادیری که یکتا هستند کاربرد دارد اما در نوع های پیچیده برای جلوگیری از مقادیر تکراری لازم است که یک interface به نام IEquatable<> را برای آن نوع خاص پیاده سازی کرده تا سبب بروز ذخیره سازی مقادیر تکراری نشویم.



**Getting an Enumerator that Iterates through HashSet<T> Collection in C#:**

**O(1)**

HashSet**<string>** hashSetCountries1 = new HashSet**<string>()**;

//Adding Elements to HashSet using Add Method

hashSetCountries1.Add**(**"IND"**)**;

hashSetCountries1.Add**(**"USA"**)**;

hashSetCountries1.Add**(**"UK"**)**;

hashSetCountries1.Add**(**"NZ"**)**;

hashSetCountries1.Add**(**"BAN"**)**;

**HashSet<string>.Enumerator em = hashSetCountries1.GetEnumerator();**

**while** **(**em.MoveNext**())**

**{**

**string** val = em.Current;

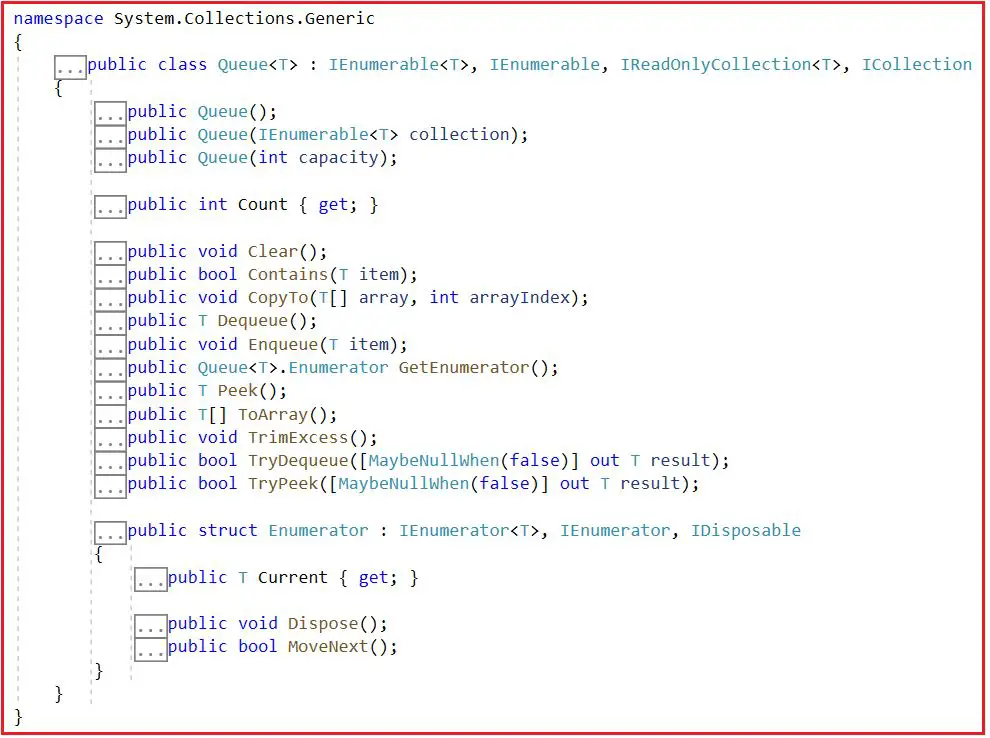
Console.WriteLine**(**val**)**;

**}**

# Generic Queue Collection Class in C#

نوع کالکشن صف یا queue که به صورت first in first out کار می کند و پیشنهاد می شود که از حالت generic به دلیل عدم نیاز به boxing و unboxing استفاده شود

اولین ایتمی که به لیست اضافه می شود برابر است با اولین ایتمی که از لیست خارج می شود.



**How to Add Elements**

1. **Enqueue(T item):** The Enqueue(T item) method is used to add an element at the end of the queue. Here, the parameter item specifies the element to add to the queue. The value can be null for a reference type i.e. when T is a reference type.

به دلیل این که متد Add ندارد نمی توان initializer کرد.

**How to Remove Elements**

1. **Dequeue():** This method is used to remove and return the object at the beginning of the Generic Queue. It returns the Object (element) removed from the beginning of the Generic Queue. If the Queue is empty, then it will throw InvalidOperationException.
2. **Clear():** This method is used to remove all objects from the Generic Queue.

**How to Get the First Element**

1. **Dequeue():** The Dequeue() method of the Queue class is used to Remove and return the object from the beginning of the Queue. That means it returns the object that is removed from the beginning of the Generic Queue. If the queue is empty, it will throw InvalidOperationException
2. **Peek():** The peek() method of the Queue class is used to return the object at the beginning of the Queue without removing it. That means it returns the object from the beginning of the Queue. If the queue is empty, then it will throw InvalidOperationException.

##### **How to check whether an Element Exists or Not**

1. **Contains(T item):** The Contains(T item) method is used to determine whether an element exists in the generic queue or not. It returns true if the item is found in the generic queue; otherwise, false. Here, the parameter item specifies the element to locate in the queue. The value can be null for a reference type.

**How to Copy a Generic Queue**

1. **CopyTo(T[] array, int arrayIndex):** This method is used to copy the Generic Queue Collection Elements to an existing one-dimensional Array, starting at the specified array index. Here, the parameter array specifies the one-dimensional array that is the destination of the elements copied from the generic queue. The Array must have zero-based indexing. The arrayIndex parameter specifies the zero-based index in the array at which copying begins. If the parameter array is null, then it will throw ArgumentNullException. If the parameter index is less than zero, then it will throw ArgumentOutOfRangeException. If the number of elements in the source Generic Queue is greater than the available space from arrayIndex to the end of the destination array, then it will throw ArgumentException.

**Note:** To add items to the end of the queue, use the **Enqueue()** method. Similarly, to remove an item that is present at the beginning of the queue, use **Dequeue()** method. A foreach loop will iterate thru all the items in the queue. The peek method will return the item from the beginning of the queue but will not remove that item from the queue. To check if an item exists in the queue or not, use the **Contains()** method.

# Foreach Loop in C#

# برای پیشروی در بین ایتم های کالکشن به صورت یک به یک کاربرد دارد.

نوع های داده ای که رابط IEnumerable را پیاده سازی کرده اند قابلیت Foreach را نیز دارا هستند.

تا زمانی که در کالکشن element وجود داشته باشد عملیات loop را تکرار می کند.

کلمه کلیدی IN در foreach برای iterate کردن کاربرد دارد که هر ایتم را انتخاب می کند و در variable می ریزد و تا زمانی که به انتهای کالکشن برسد این روند ادامه دارد.

**Implement IEnumerable Interface in C#**

**class** Shop : IEnumerable

**{**

**private** Customer**[]** CustomerArray = new Customer**[**4**]**;

**public** Shop**()**

**{**

CustomerArray**[**0**]** = new Customer**()** **{** Id = 1, Name = "James" **}**;

CustomerArray**[**1**]** = new Customer**()** **{** Id = 2, Name = "Smith" **}**;

**}**

//Here implementation for the GetEnumerator method.

**public** IEnumerator GetEnumerator**()**

**{**

**return** CustomerArray.GetEnumerator**()**;

**}**

**}**

Shop objShop = new Shop**()**;

**foreach** **(**Customer cust in objShop**)**

**{**

Console.WriteLine**(**cust.Name**)**;**}**

**Limitations of foreach loop**

**امکان تغییر خود کالکشن در حلقه foreach وجود ندارد.**

**Index ایتم ها در foreach وجود ندارد و از آن بی اطلاع است.**

**Foreach فقط برای پیمایش رو به جلو کاربرد دارد و حالت معکوس با For پیاده سازی شود.**

# Generic SortedList Collection Class in C#

کالکشنی از جنس key/value است که بر اساس key و بر پایه پیاده سازی IComparer مرتب سازی را انجام می دهد. به صورت پیش فرض به ترتیب صعودی. برای نوع های داده primitive مثل int از کوچک به بزرگ و برای string به صورت alphabetically یا بر اساس حروف الفبا مرتب سازی می کند.

بر اساس key or index می توان به سادگی ایتم را یافت.

امکان ذخیره سازی value تکراری فراهم است اما key نباید duplicate شود.



نکته : می توانید با روش های forach و index و for به عناصر کالکشن دسترسی داشته باشید.

نکته : می توانید به ترتیب دلخواه خود به کالکشن عناصر را اضافه کنید اما بر اساس key و به صورت ascending آن ها را مرتب می کند.

**How to Remove Elements from a Generic SortedList Collection in C#?**

1. **Remove(TKey key):** This method is used to remove the element with the specified key from the System.Collections.Generic.SortedList. The parameter key specifies the element to remove. It returns true if the element is successfully removed; otherwise, false. This method also returns false if the key was not found in the original Generic SortedList.
2. **RemoveAt(int index):** This method is used to remove the element at the specified index of a Generic SortedList. The parameter index specifies the element to remove. It is 0 based Index.
3. **Clear():** This method is used to remove all elements from a Generic SortedList Collection.

**How to check the Availability of key/value Pairs**

1. **ContainsKey(TKey key):** This method is used to determine whether the Generic SortedList collection contains a specific key. The parameter key to locating in the Generic SortedList object. It returns true if the Generic SortedList collection contains an element with the specified key; otherwise, false. If the key is null, then it will throw System.ArgumentNullException.
2. **ContainsValue(TValue value):** This method is used to determine whether a Generic SortedList contains a specific value. The parameter value to locate in the Generic SortedList collection. The value can be null for reference types. It returns true if the Generic SortedList Collection contains an element with the specified value; otherwise, false.

**What are the differences between SortedList<TKey, TValue> and SortedDictionary< > in C#?**

sortedList مصرف حافظه ی کمتری دارد

sortedDictionary درج و حذف سریع تری برای داده های مرتب نشده دارد

اگر عناصر به صورت یکباره از داده های مرتب شده پر شود sorted List سریع تر است

sortedList قابلیت پشتیبانی از Index را دارد.

SortedList قابلیت دریافت مقدار null و duplicate برای key را ندارد.

# Generic SortedSet Collection Class in C#

عناصر موجود در کالکشن را به ترتیب asc مرتب سازی می کند و همچنین مقدار duplicate نمی پذیرد.

کاربرد : مواقعی که میخواهید عناصر فقط unique را به ترتیب Asc مرتب سازی کنید و در کالکشن ذخیره کنید.

افرایش ظرفیت به صورت dynamic دارد.

همچنین این نوع کالکشن ارائه دهنده تعداد زیادی از mathematical set operation است.

Corts

1. **SortedSet():** It initializes a new instance of the Generic SortedSet class.
2. **SortedSet(IComparer<T> comparer):** It initializes a new instance of the Generic SortedSet class that uses a specified comparer
3. **SortedSet(IEnumerable<T> collection):** It initializes a new instance of the Generic SortedSet class that contains elements copied from a specified enumerable collection.
4. **SortedSet(IEnumerable<T> collection, IComparer<T> comparer):** It initializes a new instance of the Generic SortedSet class that contains elements copied from a specified enumerable collection and that uses a specified comparer.
5. **SortedSet(SerializationInfo info, StreamingContext context):** It initializes a new instance of the Generic SortedSet class that contains serialized data. The parameter info specifies the object that contains the information that is required to serialize the Generic SortedSet object and the context parameter specifies the structure that contains the source and destination of the serialized stream associated with the Generic SortedSet object.

اگر مقدار تکرای به کالکشن اضافه شود یکی را نگه داری می کند و به صورت خودکار عملیات مرت سازی را انجام می دهد.

**Using Enumerator to Loop through SortedSet<T> Collection in C#:**

SortedSet**<int>** sortedSetNumbers = new SortedSet**<int>**

**{**

10,

5,

**}**;

//Accessing the SortedSet Elements using Enumerator

Console.WriteLine**(**"SortedSet Elements"**)**;

SortedSet**<int>**.Enumerator em = sortedSetNumbers.GetEnumerator**()**;

**while** **(**em.MoveNext**())**

**{**

**int** val = em.Current;

Console.WriteLine**(**val**)**;

**}**

**How to Remove Elements**

1. **Remove(T item):** This method is used to remove the specified element from a SortedSet object. Here, the parameter item specifies the element to remove. It returns true if the element is successfully found and removed; otherwise, false. This method returns false if the item is not found in the Generic SortedSe collection.
2. **RemoveWhere(Predicate<T> match):** This method is used to remove all elements that match the conditions defined by the specified predicate from a SortedSet collection. It returns the number of elements that were removed from the SortedSet collection. Here, the parameter match specifies the Predicate delegate that defines the conditions of the elements to remove.
3. **Clear():** This method is used to remove all elements from a SortedSet object.

SortedSet**<string>** sortedSetCountries = new SortedSet**<string>()**

**{**

"BANGLADESH",

"NEPAL"

**}**;

sortedSetCountries.Remove**(**"Bangladesh"**)**;

sortedSetCountries.RemoveWhere**(**x =**>** x.Length **>** 3**)**;

**How to Check the Availability**

1. **Contains(T item):** This method is used to determine whether a SortedSet object contains the specified element. The parameter item specifies the element to locate in the SortedSet object. It returns true if the SortedSet object contains the specified element; otherwise, false.

Console.WriteLine**(**"Is INDIA Key Exists : " + sortedSetCountries.Contains**(**"INDIA"**))**;

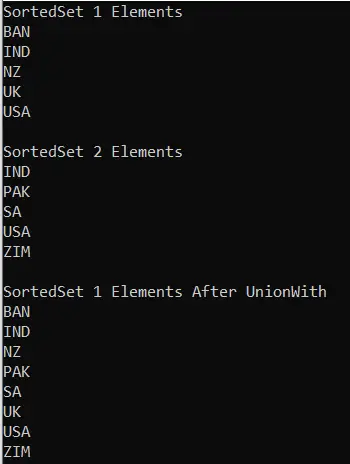
**Set Operations on Generic SortedSet<T>**

1. **UnionWith(IEnumerable<T> other):** This method is used to modify the current SortedSet object to contain all elements that are present in itself, the specified collection, or both. Here, the parameter other specifies the collection to compare to the current SortedSet object. If the parameter other is null, then we will get ArgumentNullException.
2. **IntersectWith(IEnumerable<T> other):** This method is used to modify the current SortedSet object to contain only elements that are present in that object and in the specified collection. Here, the parameter other specifies the collection to compare to the current SortedSet object. If the parameter other is null, then we will get ArgumentNullException.
3. **ExceptWith(IEnumerable<T> other):** This method is used to remove all elements in the specified collection from the current SortedSet object. Here, the parameter other specifies the collection of items to remove from the SortedSet object. If the parameter other is null, then we will get ArgumentNullException.
4. **SymmetricExceptWith(IEnumerable<T> other):** This method is used to modify the current SortedSet object to contain only elements that are present either in that object or in the specified collection, but not both. Here, the parameter other specifies the collection to compare to the current SortedSet object. If the parameter other is null, then it will throw ArgumentNullException.

**UnionWith**

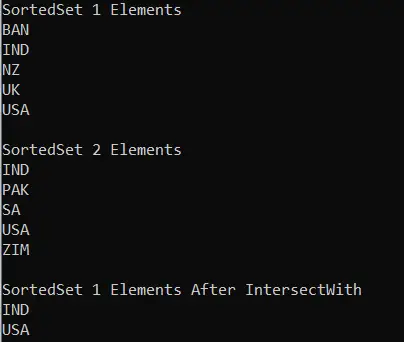
برای اجتماع 2 کالکشن کاربرد دارد به صورتی که تمام مقادیری که در خود یا کالشکن دیگر یا هر دو هست را نگه داری می کند.

مقدار هر دو را نگه می دارد با حذف مقادیر تکرای که فقط یک بار باشند.



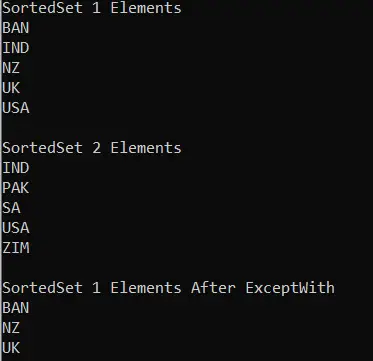
**IntersectWith**

برای اشتراک گیری بین 2 مجموعه کاربرد دارد.



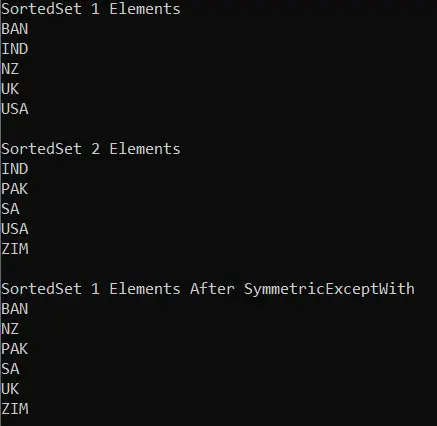
**ExceptWith**

تمامی المان های از کالکشن اول را حفظ می کند که در دومی وجود ندارند.



**SymmetricExceptWith**

عناصر یونیک بین هر دو کالکشن را نگه می دارد.



**Generic SortedSet Collection with Complex Type in C#:**

اکنون نحوه کارکرد sortedList با complex datatype ها را بررسی می کنیم.

نکته برای این که کالکشن متوجه شود که کدام property نوع complex باید مبنا قرار گیرد به این منظور نیاز به پیاده سازی رابط IComparer است کد زیر دارای خطای run time است.

SortedSet**<**Student**>** sortedSetStudents = new SortedSet**<**Student**>()**

**{**

new Student**(){** ID = 101, Name ="Anurag", Branch="CSE"**}**,

new Student**(){** ID = 101, Name ="Any Value", Branch="Any Value"**}**,

new Student**(){** ID = 102, Name ="Mohanty", Branch="CSE"**}**,

**}**;

راه حل در ادامه پیاده سازی رابط و نوشتن مجدد کد بالا بدون خطا

**public** **class** Student : IComparable**<**Student**>**

**{**

**public** **int** ID **{** **get**; **set**; **}**

**public** **string** Name **{** **get**; **set**; **}**

**public** **string** Branch **{** **get**; **set**; **}**

**public** **int** CompareTo**(**Student other**)**

**{**

**if** **(**this.ID **>** other.ID**)**

**{**

**return** 1;

**}**

**else** **if** **(**this.ID **<** other.ID**)**

**{**

**return** -1;

**}**

**else**

**{**

**return** 0;

**}**

**}**

**}**

نکته : build in datatype یا Primitive ها به دلیل پیاده سازی این رابط از قبل خطا نمی دهند.

**How to copy a list to a SortedSet in C#?**

متد کپی ندارد بلکه لازم است از یکی از overload های مربوط به Ctor استفاده کرد برای ایجاد کالکشن جدید با مقادیر کالکشن موجود

SortedSet**<string>** sortedSetCountries = new SortedSet**<string>(**listCountries**)**;

**Generic SortedSet Collection Class Properties in C#**

1. **Min**: Returns the minimum value in the set
2. **Max**: Returns the maximum value in the set
3. **Count**: Returns the number of elements in the SortedSet.
4. **Comparer**: Returns the comparer that is used to order the values in the Generic SortedSet.

**sortedSetTips**

نکته : یک ترتیب مرتب شده را بدون تاثیر در عملکرد برای عملیات های حذف و درج نگه داری میکند.

نکته: تغییر مقادیر ممکن است منجر به بروز رفتار پیش بینی نشده می شود.

# Generic SortedDictionary Collection Class in C#

کالکشن key/value بیس است که sort بر اساس key انجام می شود. با کمک کلید جستجو و حذف از کالکشن بسیار راحت است.

مقدار حافظه به صورت خودکار افزایش می باید و همچنین مقدار key نمی تواند null باشد بروز exception اما value در حالت های refereceType می تواند null باشد.

Cort

1. **SortedDictionary():** It initializes a new instance of the Generic SortedDictionary class that is empty and uses the default System.Collections.Generic.IComparer implementation for the key type.
2. **SortedDictionary(IComparer<TKey> comparer):** It initializes a new instance of the Generic SortedDictionary class that is empty and uses the specified System.Collections.Generic.IComparer implementation to compare keys. The parameter comparer specifies the System.Collections.Generic.IComparer implementation to use when comparing keys, or null to use the default System.Collections.Generic.Comparer for the type of the key.
3. **SortedDictionary(IDictionary<TKey, TValue> dictionary):** It initializes a new instance of the Generic SortedDictionary class that contains elements copied from the specified System.Collections.Generic.IDictionary and uses the default System.Collections.Generic.IComparer implementation for the key type. The parameter comparer specifies the System.Collections.Generic.IDictionary whose elements are copied to the new Generic SortedDictionary. If the dictionary is null, then it will throw ArgumentNullException. If the dictionary contains one or more duplicate keys, then it will throw ArgumentException.
4. **SortedDictionary(IDictionary<TKey, TValue> dictionary, IComparer<TKey> comparer):** It initializes a new instance of the Generic SortedDictionary class that contains elements copied from the specified System.Collections.Generic.IDictionary and uses the specified System.Collections.Generic.IComparer implementation to compare keys. The parameter comparer specifies the System.Collections.Generic.IDictionary whose elements are copied to the new Generic SortedDictionary. The parameter comparer specifies the System.Collections.Generic.IComparer implementation to use when comparing keys, or null to use the default System.Collections.Generic.Comparer for the type of the key. If the dictionary is null, then it will throw ArgumentNullException. If the dictionary contains one or more duplicate keys, then it will throw ArgumentException.

به صورت خودکار عملیات مرتب سازی صعودی را انجام می دهد.

دسترسی با استفاده از indexer foreach for

**How to check the Availability of key/value**

1. **ContainsKey(TKey key):** This method is used to determine whether the Generic SortedDictionary collection contains a specific key. The parameter key to locating in the Generic SortedDictionary object. It returns true if the Generic SortedDictionary collection contains an element with the specified key; otherwise, false. If the key is null, then it will throw ArgumentNullException.
2. **ContainsValue(TValue value):** This method is used to determine whether a Generic SortedDictionary contains a specific value. The parameter value to locate in the Generic SortedDictionary collection. The value can be null for reference types. It returns true if the Generic SortedDictionary Collection contains an element with the specified value; otherwise, false.

##### **How to Assign Values to a SortedDictionary<TKey, TValue> with Indexer in C#?**

**genericSortedDictionary[key] = value;**

بروزرسانی مقادیر کالکشن :

SortedDictionary**<string**, **string>** genericSortedDictionary = new SortedDictionary**<string**, **string>**

**{**

**{** “Ind”, “India” **}**,

**{** “USA”, “United State of America” **}**

**}**;

genericSortedDictionary**[**“SA”**]** = “South Africa”;

genericSortedDictionary**[**“SL”**]** = “Srilanka”;

genericSortedDictionary**[**“ENG”**]** = “England”;

**Using Enumerator to Iterates through SortedDictionary**

// To get an IDictionaryEnumerator for the SortedDictionary

IDictionaryEnumerator myEnumerator = genericSortedDictionary.GetEnumerator**()**;

// If MoveNext passes the end of the collection, the enumerator is positioned

// after the last element in the collection and MoveNext returns false.

**while** **(**myEnumerator.MoveNext**())**

**{**

Console.WriteLine**(**$"Key: {myEnumerator.Key} and Value: {myEnumerator.Value}"**)**;

**}**

متد مهم TryGetValue()

بررسی می کند یک مقدار وجود دارد یا نه اگر وجود داشت آن را در پارامتر دوم خود می دهد بر اساس کلید ارسالی:

genericSortedDictionary.TryGetValue**(**102, out std102**)**

**Differences Between SortedDictionary<TKey, TValue> and SortedList<TKey, TValue> in C#?**

هر دو یک درخت جستجوی دو دویی یا binary search tree با پیچیدگی زمانی بازیابی O(log N) هستند و تفاوت در استفاده از حافظه و سرعت درج و حذف است.

sortedList حافظه ی کمتری از sortedDitionary استفاده می کند

sortedDictionary برای insert , remove کردن دیتای غیر مرتب سریع تر است O(Log n ) در مقابل O(N) برای sortedList

اگر collection در یک لحظه مقدار دهی شود sortedList از SortedDictionay سریع تر است

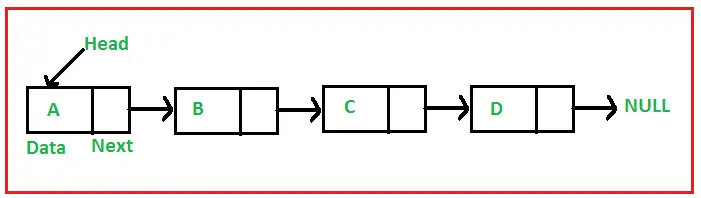
# Generic LinkedList Collection Class in C#

یک ساختار ذخیره سازی خطی است که برای ذخیره سازی داده ها به صورت غیر پیوسته استفاده می شود.

عناصر در linkedList به یک دیگر متصل هستند با استفاده از pointer ها. به زبان دیگر شامل node های است که هر node شامل data feild و reference یا لینکی به next node یا داده بعدی است.

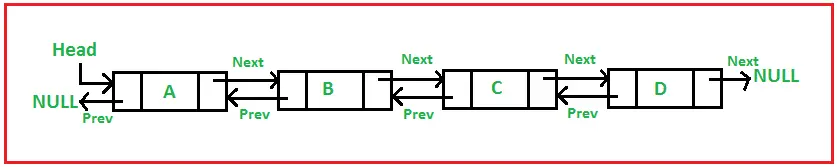
هر node دارای دو قسمت است.

1. **Data−** Each node of a linked list can store data.
2. **Address −** Each node of a linked list contains an address to the next node, called “Next”



نکته : LinkedList به صورت **double-ended** است به این معنی که هر node به forward , backward نود خود نیز همزمان اشاره می کند dll یا doubly linked list نیز با آن ها می گویند.

نکته: قابلیت insert و remove سریع را نیز دارا است. مقدار فضای آن با توجه به برنامه شما به صورت dynamic افزایش می یابد.



نکته : insert , remove در linkedList دارای پیچیدگی زمانی O(1) است.

نکته: اگر list به صورت empty باشد property های first, last آن برابر null است.

نکته: مقادیر duplicate را نیز می پذیرد.

##### **How to Add Elements into a Generic LinkedList<T>**

1. **AddAfter(LinkedListNode<T> node, LinkedListNode<T> newNode):** This is used to add the specified new node after the specified existing node in the Generic LinkedList.
2. **AddAfter(LinkedListNode<T> node, T value):** This is used to add a new node containing the specified value after the specified existing node in the Generic LinkedList.
3. **AddBefore(LinkedListNode<T> node, LinkedListNode<T> newNode):** This method is used to add the specified new node before the specified existing node in the Generic LinkedList.
4. **AddBefore(LinkedListNode<T> node, T value):** This method is used to add a new node containing the specified value before the specified existing node in the Generic LinkedList.
5. **AddFirst(LinkedListNode<T> node):** This is used to add the specified new node at the start of the Generic LinkedList.
6. **AddFirst(T value):** This is used to add a new node containing the specified value at the start of the Generic LinkedList.
7. **AddLast(LinkedListNode<T> node):** This is used to add the specified new node at the end of the Generic LinkedList.
8. **LinkedListNode<T> AddLast(T value):** This is used to add a new node containing the specified value at the end of the Generic LinkedList.

معمولا برای افزودن ایتم به linkedList از متد addLast استفاده می کنیم.

نکته: به دلیل این که متد Add را ندارد پس collection Initializer امکان پذیر نیست.

نکته: این نوع فاقد indexer است.

##### **How to Remove Elements**

1. **Remove(LinkedListNode<T> node):** The Remove(LinkedListNode<T> node) method is used to remove the specified node from the Generic LinkedList.
2. **Remove(T value):** The Remove(T value) method is used to remove the first occurrence of the specified value from the Generic LinkedList.
3. **RemoveFirst():** The RemoveFirst() method is used to remove the node at the start of the Generic Linked List.
4. **RemoveLast():** The RemoveLast() method is used to remove the node at the end of the Generic Linked List.
5. **Clear():** The Clear() method is used to remove all nodes from the Generic LinkedList.

##### **How to check the Availability of elements**

1. **Contains(T value):** This method is used to determine whether a value is in the Generic LinkedList. Here, the parameter value specifies the value to locate in the Generic LinkedList. The value can be null for reference types. It returns true if the value is found in the Generic LinkedList; otherwise, false.

##### **How to Add a Node After a Given Node of a Linked List in C#?**

LinkedList**<string>** linkedList = new LinkedList**<string>()**;

linkedList.AddLast**(**"India"**)**;

LinkedListNode**<string>** USANode = linkedList.AddLast**(**"USA"**)**;

linkedList.AddLast**(**"Srilanka"**)**;

Console.WriteLine**(**"LinkedList Elements"**)**;

**foreach** **(**var item in linkedList**)**

**{**

Console.WriteLine**(**item**)**;

**}**

Console.WriteLine**(**"\nAfter Adding a Node After USA Node"**)**;

linkedList.AddAfter**(**USANode, "UK"**)**;

**foreach** **(**var item in linkedList**)**

**{**

Console.WriteLine**(**item**)**;

**}**

برای کارکردن با متد AddBefore نیز از روش بالا استفاده می کنیم.

نکات: بر اساس نیاز حافظه افزایش می یابد. ساختار داده stack , queue نیز با این نوع به راحتی پیاده سازی میشود. زمان دسترسی سریعی دارد و حذف اضافه نیز سریع انجام می شود.

# **Concurrent Collection in C#**

تا به الان تمامی کالکشن های که با آن ها آشنا شدیم type safe یا not type safe بوده اند اما هیچ کدام thread safe نیستند.

فرض کنید یک dictionary داریم که یک یا چند thread به صورت همزمان عملیات add , remove را می خواهند بر روی آن collection پیاده سازی کنند.

سناریو:

یک فیلد از نوع dictionary داریم و 2 متد برای افزودن به دیکشنری یکی را توسط thread1 دومی را توسط thread2 به صورت concurrent صدا می زنیم.

**static** Dictionary**<int**, **string>** dictionary = new Dictionary**<int**, **string>()**;

**static** **void** Main**(string[]** args**)**

**{**

Thread t1 = new Thread**(**Method1**)**;

Thread t2 = new Thread**(**Method2**)**;

t1.Start**()**;

t2.Start**()**;

Console.ReadKey**()**;

**}**

**public** **static** **void** Method1**()**

**{**

**for** **(int** i = 0; i **<** 10; i++**)**

**{**

dictionary.Add**(**i, "Added By Method1 " + i**)**;

Thread.Sleep**(**100**)**;

**}**

**}**

**public** **static** **void** Method2**()**

**{**

**for** **(int** i = 0; i **<** 10; i++**)**

**{**

dictionary.Add**(**i, "Added By Method2 " + i**)**;

Thread.Sleep**(**100**)**;

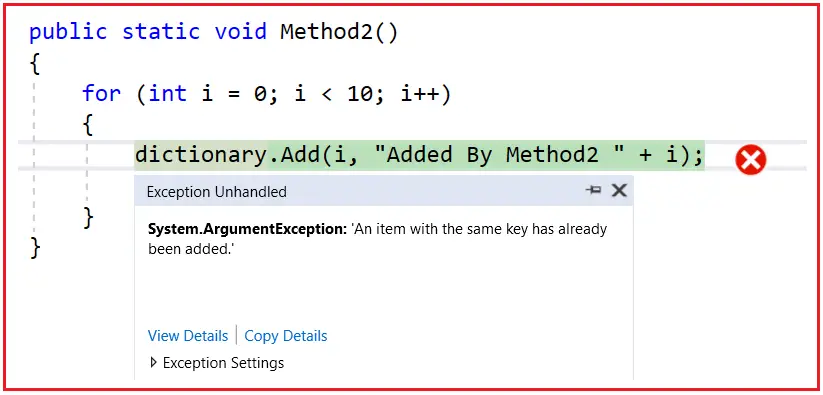
**}**

**}**

**}**

در ابتدا با خطای ArgumentException مواجه می شویم.

علت: کلید مربوط به dictionary باید به صورت unique باشد اما از for استفاده کردیم باعث شده کلید یکسان داشته باشیم.



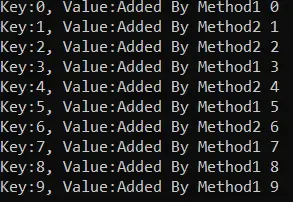
به عنوان توسعه دهنده می توانیم از lock استفاده کنیم اما به دلیل performance issue پیشنهاد نمی شود در این مواقع است که concurrent collection به کار می آید.

* تغییر زیر را در کد قبلی می دهیم و مجددا اجرا می کنیم.

**static** ConcurrentDictionary**<int**, **string>** dictionary = new ConcurrentDictionary**<int**, **string>()**;

dictionary.TryAdd**(**i, "Added By Method1 " + i**)**;

بدون هیج خطای اجرا می شود خروجی:



1. **AddOrUpdate**: Adds a new entry if does not exist else updates the existing one
2. **GetOrAdd**: Retrieves an item if exists, else first adds it then retrieve it
3. **TryAdd**, **TrygetValue**, **TryUpdate**, **TryRemove**: Try to do the specified operation like Add/Get/Update/Remove. It returns true if performing the operation and false if failing to do the operation.

# ConcurrentDictionary Collection Class in C#

ارائه دهنده thread safe key/value collection است. که می تواند توسط multithread به صورت concurrent در دسترس باشد.

نکته : به صورت internally مدیریت locking را انجام می دهد و متد های update add remove را به صورت threadSafe ارائه می دهد.

نکات:

کلید باید یکتا باشد – مقادیر اگر reference type باشد اشکالی ندارد null باشد. ظرفیت با توجه به تعداد عناصر افزایش می یابد.

**Ctors**

1. **ConcurrentDictionary():** It initializes a new instance of the ConcurrentDictionary class that is empty, has the default concurrency level, has the default initial capacity, and uses the default comparer for the key type.
2. **ConcurrentDictionary(IEnumerable<KeyValuePair<TKey, TValue>> collection):** It initializes a new instance of the ConcurrentDictionary class that contains elements copied from the specified System.Collections.Generic.IEnumerable has the default concurrency level, has the default initial capacity, and uses the default comparer for the key type.
3. **ConcurrentDictionary(IEqualityComparer<TKey> comparer):** It initializes a new instance of the ConcurrentDictionary class that is empty, has the default concurrency level and capacity, and uses the specified System.Collections.Generic.IEqualityComparer.
4. **ConcurrentDictionary(int concurrencyLevel, int capacity):** It initializes a new instance of the ConcurrentDictionary class that is empty, has the specified concurrency level and capacity, and uses the default comparer for the key type.
5. **ConcurrentDictionary(IEnumerable<KeyValuePair<TKey, TValue>> collection, IEqualityComparer<TKey> comparer):** It initializes a new instance of the ConcurrentDictionary class that contains elements copied from the specified System.Collections.IEnumerable has the default concurrency level, has the default initial capacity, and uses the specified System.Collections.Generic.IEqualityComparer.
6. **ConcurrentDictionary(int concurrencyLevel, IEnumerable<KeyValuePair<TKey, TValue>> collection, IEqualityComparer<TKey> comparer):** It initializes a new instance of the ConcurrentDictionary class that contains elements copied from the specified System.Collections.IEnumerable, and uses the specified System.Collections.Generic.IEqualityComparer.
7. **ConcurrentDictionary(int concurrencyLevel, int capacity, IEqualityComparer<TKey> comparer):** It initializes a new instance of the ConcurrentDictionary class that is empty, has the specified concurrency level, has the specified initial capacity, and uses the specified System.Collections.Generic.IEqualityComparer.

### **ConcurrentDictionary Collection Class Important Methods in C#**

**TryUpdate(TKey key, TValue newValue, TValue comparisonValue)**

مقدار کلید – مقدار جدید – مقداری که کلید در حال حاظر دارد. اگر موفق باشد true برمیگرداند و مقدار جدید را جایگزین می کند.

##### **AddOrUpdate Methods of ConcurrentDictionary**

اگر کلید وجو داشته باشد آن را بروزرسانی می کند و اگر وجود نداشته باشد مقدار جدید را اضافه می کند.

در مثال زیر هم بروزرسانی و هم افزودن را میبینید.

**static** **void** Main**(string[]** args**)**

**{**

ConcurrentDictionary**<string**, **string>** dictionaryCountries = new ConcurrentDictionary**<string**, **string>()**;

dictionaryCountries.TryAdd**(**"UK", "United Kingdom"**)**;

dictionaryCountries.TryAdd**(**"USA", "United State of America"**)**;

dictionaryCountries.TryAdd**(**"IND", "India"**)**;

//All ConcurrentDictionary Elements

Console.WriteLine**(**"All ConcurrentDictionary Elements"**)**;

**foreach** **(**KeyValuePair**<string**, **string>** KVP in dictionaryCountries**)**

**{**

Console.WriteLine**(**$"Key:{KVP.Key}, Value: {KVP.Value}"**)**;

**}**

dictionaryCountries.AddOrUpdate**(**"UK", "Kingdom United", **(**k, v**)** =**>** "United Kingdom Updated"**)**;

dictionaryCountries.AddOrUpdate**(**"SL", "Srilanka", **(**k, v**)** =**>** "Srilanka Updated"**)**;

Console.WriteLine**(**"\nAll ConcurrentDictionary Elements After AddOrUpdate Method"**)**;

**foreach** **(**KeyValuePair**<string**, **string>** KVP in dictionaryCountries**)**

**{**

Console.WriteLine**(**$"Key:{KVP.Key}, Value: {KVP.Value}"**)**;

**}**

Console.ReadKey**()**;

**}**

##### **GetOrAdd Methods of ConcurrentDictionary**

عملکرد مشابه متد قبلی دارد با این تفاوت که اگر مقدار به ازای کلید وجود داشته باشد آن را برمیگرداند

// Get UK or add it with value of United Kingdom.

**string** Result1 = dictionaryCountries.GetOrAdd**(**"UK", "United Kingdom"**)**;

Console.WriteLine**(**$"Key:UK, Value: {Result1}"**)**;

// Get SL or add it with value Srilanka.

**string** Result2 = dictionaryCountries.GetOrAdd**(**"SL", "Srilanka"**)**;

Console.WriteLine**(**$"Key:SL, Value: {Result2}"**)**;

Console.WriteLine**(**"\nConcurrentDictionary Elements After GetOrAdd Method"**)**;

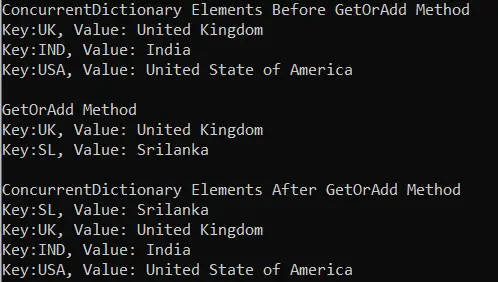
**foreach** **(**KeyValuePair**<string**, **string>** KVP in dictionaryCountries**)**

**{**

Console.WriteLine**(**$"Key:{KVP.Key}, Value: {KVP.Value}"**)**;

**}**

نتیجه نهایی



**TryGetValue(TKey key, out TValue value):**

**مقدار وجود داشته باشد true به علاوه مقدار به ازای کلید و اگر وجود نداشته باشد false برمیگرداند.**

# ConcurrentQueue Collection Class in C#

برای پیاده سازی ساختار داده (data structure) first in first out به صورت thread safe است.در محیط های multi thread کاربرد دارد.

##### **Generic Queue Example with Multi-Thread in C#**

به قطعه کد زیر توجه کنید:

**public** **static** **void** TestQueue**()**

**{**

var MobileOrders = new Queue**<string>()**;

Task t1 = Task.Run**(()** =**>** GetOrders**(**"Pranaya", MobileOrders**))**;

Task t2 = Task.Run**(()** =**>** GetOrders**(**"Anurag", MobileOrders**))**;

Task.WaitAll**(**t1, t2**)**; //Wait till both the task completed

**foreach** **(**var mobileOrder in MobileOrders**)**

**{**

Console.WriteLine**(**$"Order: {mobileOrder}"**)**;

**}**

**}**

**private** **static** **void** GetOrders**(string** custName, Queue**<string>** MobileOrders**)**

**{**

**for** **(int** i = 0; i **<** 100; i++**)**

**{**

//Thread.Sleep(100);

**string** order = **string**.Format**(**$"{custName} Needs {i+3} Mobiles"**)**;

MobileOrders.Enqueue**(**order**)**;

**}**

**}**

قطعه کد بالا به exception مواجه می شود به دلیل این که Queue و متد enqueue برای کارکردن با بیش از thread طراحی نشده است.

راه حل :**explicit lock**

**برای متد enqueue که** قابلیت چند نخی ندارد قفل می گذاریم.

**private** **static** **void** GetOrders**(string** custName, Queue**<string>** MobileOrders**)**

**{**

**for** **(int** i = 0; i **<** 100; i++**)**

**{**

//Thread.Sleep(100);

**string** order = **string**.Format**(**$"{custName} Needs {i+3} Mobiles"**)**;

**lock** **(**lockObj**)**

**{**

MobileOrders.Enqueue**(**order**)**;

**}**

**}**

**}**

##### **ConcurrentQueue with more than one Thread in C#**

به صورت خودکار پیاده سازی thread safe را انجام داده است و مناسب محیط های multi thread است.

تنها تغییر لازم به شرح زیر است برای کد ها قبلی

var MobileOrders = new ConcurrentQueue**<string>()**;

##### **How to Remove Elements from ConcurrentQueue<T>**

1. **TryDequeue(out T result):** This method tries to remove and return the object at the beginning of the concurrent queue. Here, the parameter result contains the object removed if the operation was successful. If no object was available to be removed, the value is unspecified. This method returns true if an element was removed and returned from the beginning of the ConcurrentQueue successfully; otherwise, false.

سایر متد ها نیز مشابه queue است که لازم به ذکر مجدد در این قسمت نمی باشد.

# ConcurrentStack Collection Class in C#

برای ارائه ی ساختار داده یا DATA STRUCTURE به صورت LAST in first out کاربرد دارد در محیط های multi thread به صورت thread safe

تنها تفاوتی که با stack عادی دارد thread safe بودن concurrent stack است. اما stack را نیز میتوان به صورت دستی thread safe کرد با استفاده از explicit lock.

Example

**public** **static** **void** TestStack**()**

**{**

Stack**<string>** MobileOrders = new Stack**<string>()**;

Task t1 = Task.Run**(()** =**>** GetOrders**(**"Pranaya", MobileOrders**))**;

Task t2 = Task.Run**(()** =**>** GetOrders**(**"Anurag", MobileOrders**))**;

Task.WaitAll**(**t1, t2**)**; //Wait till both the task completed

**foreach** **(**var mobileOrder in MobileOrders**)**

**{**

Console.WriteLine**(**$"Order Placed: {mobileOrder}"**)**;

**}**

**}**

**private** **static** **void** GetOrders**(string** custName, Stack**<string>** MobileOrders**)**

**{**

**for** **(int** i = 0; i **<** 3; i++**)**

**{**

Thread.Sleep**(**100**)**;

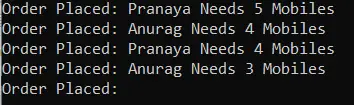
**string** order = **string**.Format**(**$"{custName} Needs {i + 3} Mobiles"**)**;

MobileOrders.Push**(**order**)**;

**}**

**}**

قطعه کد بالا که به صورت multithread پیاده سازی شده است فاقد هرگونه خطا است و به درستی اجرا می شود اما نکته ی منفی که دارد این است که نتیجه ی نهایی پیش بینی ناپذیر است.



به دلیل این که متد push برای حالت های multi thread بهینه نشده است نیاز به استفاده از lock است برای دریافت نتیجه درست.

**private** **static** **void** GetOrders**(string** custName, Stack**<string>** MobileOrders**)**

**{**

**for** **(int** i = 0; i **<** 3; i++**)**

**{**

Thread.Sleep**(**100**)**;

**string** order = **string**.Format**(**$"{custName} Needs {i + 3} Mobiles"**)**;

**lock** **(**lockObject**)**

**{**

MobileOrders.Push**(**order**)**;

**}**

**}**

**}**

راه حل مناسبی نیست به دلیل این که ممکن است جایی فراموش شود قفل گذاشته شود.

solution

ConcurrentStack**<string>** MobileOrders = new ConcurrentStack**<string>()**;

##### **How to Add Elements into a ConcurrentStack<T>**

1. **Push(T item):** This method is used to insert an object at the top of the ConcurrentStack.
2. **PushRange(T[] items):** This method is used to insert multiple objects at the top of the ConcurrentStack atomically.
3. **PushRange(T[] items, int startIndex, int count):** This method is used to insert multiple objects at the top of the ConcurrentStack atomically. Here, the parameter items specify the objects to push onto the ConcurrentStack. The parameter startIndex specifies the zero-based offset in items at which to begin inserting elements onto the top of the ConcurrentStack. And, the parameter count specifies the number of elements to be inserted onto the top of the ConcurrentStack.

##### **How to Remove Elements**

1. **TryPop(out T result):** This method attempts to pop and return the object at the top of the ConcurrentStack. Here, the output parameter result will contain the object removed if the operation was successful. If no object was available to be removed, the value is unspecified. It returns true if an element was removed and returned from the top of the ConcurrentStack successfully; otherwise, false.
2. **TryPopRange(T[] items):** This method attempts to pop and return multiple objects from the top of the ConcurrentStack atomically. The parameter items specify the Array to which objects popped from the top of the ConcurrentStack will be added. It returns the number of objects successfully popped from the top of the ConcurrentStack and inserted into items.
3. **TryPopRange(T[] items, int startIndex, int count):** This method attempts to pop and return multiple objects from the top of the ConcurrentStack atomically. Here, the parameter items specify the Array to which objects popped from the top of the ConcurrentStack will be added. The parameter startIndex specifies the zero-based offset in items at which to begin inserting elements from the top of the System.Collections.Concurrent.ConcurrentStack. And the parameter count specifies the number of elements to be popped from the top of the ConcurrentStack and inserted into items. It returns the number of objects successfully popped from the top of the stack and inserted into items.

##### **How to Get the Top Element from the ConcurrentStack**

1. **TryPop(out T result):** This method attempts to pop and return the object at the top of the ConcurrentStack. Here, the output parameter result will contain the object removed if the operation was successful. If no object was available to be removed, the value is unspecified. It returns true if an element was removed and returned from the top of the ConcurrentStack successfully; otherwise, false.
2. **TryPeek(out T result):** This method attempts to return an object from the top of the ConcurrentStack without removing it. Here, the parameter result contains an object from the top of the ConcurrentStack or an unspecified value if the operation failed. It returns true if an object was returned successfully; otherwise, false.
3. **CopyTo(T[] array, int index):** This method is used to copy the ConcurrentStack Elements to an existing one-dimensional Array, starting at the specified array index. Here, the parameter array specifies the one-dimensional array that is the destination of the elements copied from the ConcurrentStack. The Array must have zero-based indexing. The index parameter specifies the zero-based index in the array at which copying begins.

تفاوت ها

قابلیت افزودن multiple item که در حالت stack به صورت single است و همچنین قابلیت popRange دارد که در حالت stack به صورت pop است قابلیت tryPop , TryPopRange

# **ConcurrentBag Collection Class in C#**

قابلیت ذخیره کردن object ها در فرم نامرتب و همچنین قابلیت ذخیره کردن duplicate object را نیز دارا است. کارایی این نوع بسیار شبیه به List است با این تفاوت ک به صورت threadSafe است

اگر به صورت multithread فرآیند و مقدار دهی یک لیست را انجام دهیم با فراخوانی یک متد توسط 2 ترد نتیجه حاصله به این صورت است که به ترتیب فراخوانی لیست پر نمی شود و هر بار به ترتیب متفاوت پر می شود.

راه حل ها

Lock , concurrentBag<>

##### **How to Remove Elements**

**TryTake(out T result):** This method attempts to remove and return an object from the ConcurrentBag collection. When this method returns, the result contains the object removed from the ConcurrentBag or the default value of T if the bag is empty. It returns true if an object was removed successfully; otherwise, false.

**TryPeek(out T result):** This method attempt to return an object from the ConcurrentBag without removing it. When this method returns, the parameter result contains an object from the ConcurrentBag or the default value of T if the operation failed. It returns true if an object was returned successfully; otherwise, false.

##### **ConcurrentBag with Producer/Consumer Example in C#:**

For example, let’s say we have two threads Thread1 and Thread2. Thread1 added four elements such as 10,20,30,40 to the ConcurrentBag collection. Then Thread2 added three elements as 50,60,70 to the same ConcurrentBag collection. Once both threads added the elements into the collection, Thread1 starts retrieving the data. As Thread1 added 10,20,30,40 elements to the collection, so these elements get preferences over 50,60,70. Once Thread1 retrieves all the four elements which are added by Thread1 then Thread1 goes to retrieve Thread2 inserted elements such as 50,60,70. For a better understanding, please have a look at the below example.

static ConcurrentBag<int> concurrentBag = new ConcurrentBag<int>();

static AutoResetEvent autoEvent1 = new AutoResetEvent(false);

static void Main(string[] args)

{

Task thread1 = Task.Factory.StartNew(() => AddThread1Elements());

Task thread2 = Task.Factory.StartNew(() => AddThread2Elements());

Task.WaitAll(thread1, thread2);

Console.WriteLine("End of the Main Method");

Console.ReadKey();

}

public static void AddThread1Elements()

{

int[] array = { 10, 20, 30, 40 };

for (int i = 0; i < array.Length; i++)

{

concurrentBag.Add(array[i]);

}

//wait for second thread to add its items

autoEvent1.WaitOne();

while (concurrentBag.IsEmpty == false)

{

if (concurrentBag.TryTake(out int item))

{

Console.WriteLine($"Thread1 Reads: {item}");

}

}

}

public static void AddThread2Elements()

{

int[] array = { 50, 60, 70 };

for (int i = 0; i < array.Length; i++)

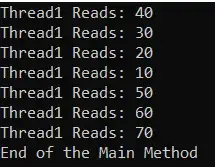
{

concurrentBag.Add(array[i]);

}

autoEvent1.Set();

}



# BlockingCollection in C#

یک پیاده سازی از الگوی producer-consumer است و ویژگی های bonding , blocking را به آن اضافه می کند. و تنها کالکشنی است که bonding , blocking را پشتیبانی می کند. همجنین thread safe نیز است به این معنی که چندین thread می توان عملیات افزودن یا حذف را روی این نوع انجام دهند.

**Producer-consumer pattern**

در این سناریو دو thread داریم یکی producer و دیگیری consumer تولید کننده و مصرف کننده. و نکته مهم این است که هر دو نخ کالکشن را بین خود share می کنند و داده های آن استفاده می کنند. و در این سناریو ها blockingCollection ها مناسب هستند. Producer در این سناریو عملیات generate data را انجام می دهد و consumer استفاده از داده ها.

نکته: می توانیم برای producer مقدار maximum تعریف داده نیز تعریف کنیم. Consumer مقدار را از کالکشن حذف می کند.

2 ویژگی مهم که سبب پیاده سازی این الگو شده:

1. **Bounding:** Bounding means as we already discussed we can set the maximum number of objects that we can store in the collection. When a producer thread reaches the BlockingCollection maximum limit, it is blocked to add new objects. In the blocked stage, the producer thread goes into sleep mode. It will unblock as soon as the consumer thread removes objects from the collection.
2. **Blocking:** Blocking means as we already discussed when the BlockingCollection is empty, the consumer thread is blocked until the producer thread adds new objects to the collections.

زمانی که maximum می رسیم به صورت خودکار پروپرتی isCompleted به true ست می شود و از افزودن دیتای بیشتر جلوگیری می کند.

**Ctors**

1. **BlockingCollection():** It initializes a new instance of the BlockingCollection class without an upper bound.
2. **BlockingCollection(int boundedCapacity):** It initializes a new instance of the BlockingCollection class with the specified upper bound. The boundedCapacity parameter specifies the bounded size of the collection. It will throw ArgumentOutOfRangeException if the bounded capacity is not a positive value.
3. **BlockingCollection(IProducerConsumerCollection<T> collection):** It initializes a new instance of the BlockingCollection class without an upper bound and uses the provided IProducerConsumerCollection as its underlying data store. Here, the parameter collection specifies the collection to use as the underlying data store. It will throw ArgumentNullException if the collection argument is null.
4. **BlockingCollection(IProducerConsumerCollection<T> collection, int boundedCapacity):** It initializes a new instance of the BlockingCollection class with the specified upper bound and uses the provided IProducerConsumerCollection as its underlying data store. Here, the boundedCapacity parameter specifies the bounded size of the collection. The parameter collection specifies the collection to use as the underlying data store. It will throw ArgumentNullException if the collection argument is null. It will throw ArgumentOutOfRangeException if the bounded capacity is not a positive value.

**نکته مهم:**

به صورت پیش فرض blockingCollection از concurrentQueue استفاده می کند. اما امکان پذیر است که از کالکشن های concurrent دیگری مانند stack , bag نیز استفاده کرد. اما نکته مهم اینجا است که فقط کالکشن های را می توانیم پاس بدهیم که IProducerConsueerCollection<> را پیاده سازی کرده اند.

##### **How to Add Elements**

**TryAdd(T item):** This method tries to add the specified item to the BlockingCollection. The parameter item to be added to the collection. It returns true if the item could be added; otherwise, false. If the item is a duplicate, and the underlying collection does not accept duplicate items, then an InvalidOperationException is thrown.

متد Add را نیز پشتیبانی می کند.

##### **How to Remove Elements**

1. **Take():** This method is used to remove an item from the BlockingCollection. It returns the item removed from the collection. The Take method is blocked when the collection is empty. It’ll unblock automatically when an item is added by another thread.
2. **TryTake(out T item):** This method tries to remove an item from the BlockingCollection. It will store the removed item in the output item parameter. It returns true if an item could be removed; otherwise, false.
3. **TryTake(out T item, TimeSpan timeout):** This method tries to remove an item from the BlockingCollection in the specified time period. The parameter timeout specifies an object that represents the number of milliseconds to wait or an object that represents -1 milliseconds to wait indefinitely. It returns true if an item could be removed from the collection within the specified time; otherwise, false. If the collection is empty then this method will wait for the time specified in the timeout parameter. If the new item is not added within the timeout value, then it returns false.
4. **TryTake(out T item, int millisecondsTimeout):** This method tries to remove an item from the System.Collections.Concurrent.BlockingCollection in the specified time period. The parameter millisecondsTimeout specifies the number of milliseconds to wait or System.Threading.Timeout.Infinite (-1) to wait indefinitely. It returns true if an item could be removed from the collection within the specified time; otherwise, false. If the collection is empty then this method will wait for the time specified in the timeout parameter. If the new item is not added within the timeout value, then it returns false.

##### **CompleteAdding method and IsCompleted Property**

Producer thread فراخوانی متد CompleteAdding را انجام می دهد و سبب این می شود که به صورت internally مقدار پروپرتی IsAddingComplete برابر True شود. پروپرتی IsComplete توسط consumer استفده می شود و زمانی true برمیگرداند که مقدار IsAddingComleted برابر true باشد و blocingCollection **برابر empty باشد. یعنی زمانی که isComplete** **برابر true است هیج نخی ایتم جدید نمی تواند اضافه کند.**

1. **CompleteAdding():** The CompleteAdding method Marks the BlockingCollection instances as not accepting any more additions.
2. **IsAddingCompleted { get; }:** This property returns true if the BlockingCollection has been marked as complete for adding else it will return false.
3. **IsCompleted { get; }:** This property returns true if the BlockingCollection has been marked as complete for adding and is empty else it will return false.

**مثال**

static void Main()

{

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

//Thread 1 (Producer Thread) Adding Item to blockingCollection

Task producerThread = Task.Factory.StartNew(() =>

{

for (int i = 0; i < 10; ++i)

{

blockingCollection.Add(i);

}

//Mark blockingCollection will not accept any more additions

blockingCollection.CompleteAdding();

});

//Thread 2 (Consumer Thread) Removing Item from blockingCollection and Printing on the Console

Task consumerThread = Task.Factory.StartNew(() =>

{

//Loop will continue as long as IsCompleted returns false

while (!blockingCollection.IsCompleted)

{

int item = blockingCollection.Take();

Console.Write($"{item} ");

}

});

Task.WaitAll(producerThread, consumerThread);

Console.ReadKey();

##### **BlockingCollection in the Foreach loop:**

**IEnumerable<T> GetConsumingEnumerable():** This method returns IEnumerable<T> so that we can use that method in the foreach loop. This method returns items as soon as items are available in the collection. The GetConsumingEnumerable() method has a blocking feature. It will block the foreach loop when the collection is empty. A foreach loop ends when the producer thread calls the CompleteAdding method.

foreach (int item in blockingCollection.GetConsumingEnumerable())

{

Console.Write($"{item} ");

}

##### **Working with Multiple Producers and Consumers using BlockingCollection in C#**

1. **AddToAny(BlockingCollection<T>[] collections, T item):** This method is used to add the specified item to any one of the BlockingCollection instances. The parameter collections specify the array of collections and the parameter item specifies the item to be added to one of the collections. It returns the index of the collection in the collections array to which the item was added.
2. **TryAddToAny(BlockingCollection<T>[] collections, T item):** This method tries to add the specified item to any one of the specified BlockingCollection instances. The parameter collections specify the array of collections and the parameter item specifies the item to be added to one of the collections. It returns the index of the collection in the collections array to which the item was added, or -1 if the item could not be added.
3. **TakeFromAny(BlockingCollection<T>[] collections, out T item):** This method takes an item from any one of the specified BlockingCollection instances. The parameter collections specify the array of collections and the parameter item specifies the item removed from one of the collections. It returns the index of the collection in the collections array from which the item was removed.
4. **TryTakeFromAny(BlockingCollection<T>[] collections, out T item):** This method tries to remove an item from any one of the specified BlockingCollection instances. The parameter collections specify the array of collections and the parameter item specifies the item removed from one of the collections. It returns the index of the collection in the collections array from which the item was removed, or -1 if an item could not be removed.

static void Main()

{

BlockingCollection<int>[] producers = new BlockingCollection<int>[3];

producers[0] = new BlockingCollection<int>(boundedCapacity: 10);

producers[1] = new BlockingCollection<int>(boundedCapacity: 10);

producers[2] = new BlockingCollection<int>(boundedCapacity: 10);

Task t1 = Task.Factory.StartNew(() =>

{

for (int i = 1; i <= 10; ++i)

{

producers[0].Add(i);

Thread.Sleep(100);

}

producers[0].CompleteAdding();

});

Task t2 = Task.Factory.StartNew(() =>

{

for (int i = 11; i <= 20; ++i)

{

producers[1].Add(i);

Thread.Sleep(150);

}

producers[1].CompleteAdding();

});

Task t3 = Task.Factory.StartNew(() =>

{

for (int i = 21; i <= 30; ++i)

{

producers[2].Add(i);

Thread.Sleep(250);

}

producers[2].CompleteAdding();

});

while (!producers[0].IsCompleted || !producers[1].IsCompleted || !producers[2].IsCompleted)

{

BlockingCollection<int>.TryTakeFromAny(producers, out int item, TimeSpan.FromSeconds(1));

if (item != default(int))

{

Console.Write($"{ item} ");

}

}

Console.ReadKey();

}

}

1,11,21,2,12,3,22,13,4,5,14,23,6,15,7,8,16,24,9,17,10,25,18,19,26,20,27,28,29,30