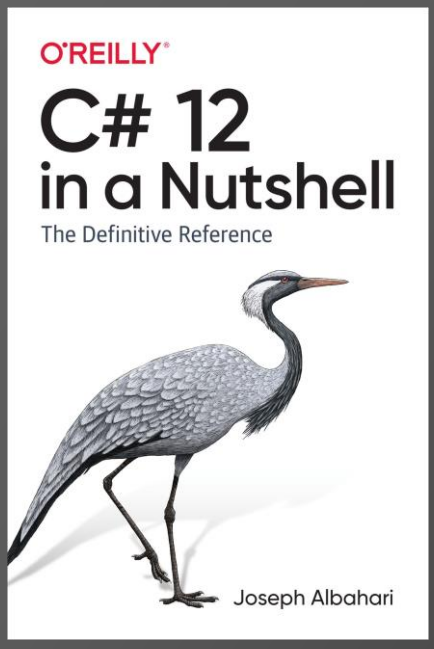
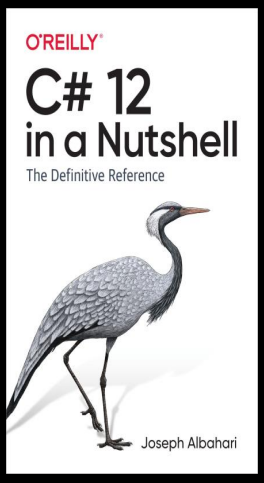
Chapter 7
DICTIONARIES



- Dictionaries
- IDictionary<TKey,TValue>
- IDictionary
- Dictionary<TKey,TValue> and Hashtable
- OrderedDictionary
- ListDictionary and HybridDictionary
- Sorted Dictionaries

Dictionaries



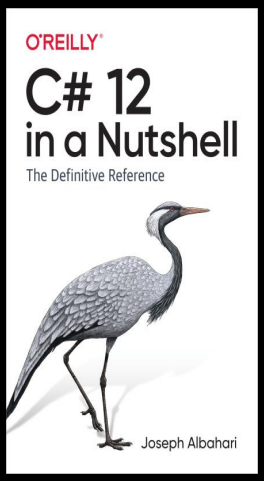
دیکشنری یک کالکشن هست که هر عنصرش یک زوج key/value هستش.

Dictionary بیشتر برای جستجوها و sorted list ها مورد استفاده قرار میگیره.

دات‌نت برای دیکشنری‌ها یک پروتکل استاندارد با استفاده از اینترفیس‌های IDictionary و IDictionary<key,value> و همچنین مجموعه‌ای از کلاس‌هایی که این اینترفیس‌ها رو پیاده‌سازی کردند، تعریف کرده. این کلاس‌ها از همدیگر در موارد زیر متفاوت‌اند:

- آیا آیتم‌ها در یک توالی مرتب ذخیره میشند؟
- آیا آیتم‌ها به وسیله ایندکس و همچنین با استفاده از key در دسترس هستند یا خیر؟
- آیا generic یا nongeneric است؟
- آیا می‌توان آیتم‌ها را با استفاده از کلید از یک دیکشنری بزرگ بازیابی کرد؟ (با سرعت بالا یا پایین)

Dictionaries



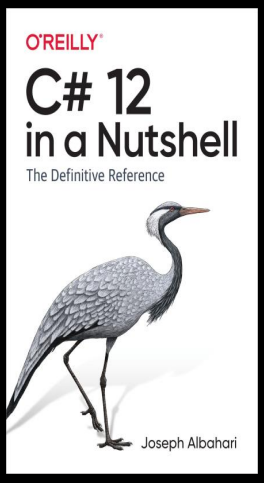
در عکس زیر میتونید تفاوت‌ها و شباهت‌های این کلاس‌ها رو ببینید.
(زمان performance براساس میلی‌ثانیه است و براساس انجام ۵۰،۰۰۰ عملیات روی دیکشنری با کلید از جنس int و مقادیر روی یک PC ۱.۵ گیگاهرتز است).

Type	Internal structure	Retrieve by index?	Memory overhead (avg. bytes per item)	Speed: random insertion	Speed: sequential insertion	Speed: retrieval by key
Unsorted						
`Dictionary <K,V >`	Hashtable	No	22	30	30	20
`Hashtable`	Hashtable	No	38	50	50	30
`ListDictionary`	Linked list	No	36	50,000	50,000	50,000
`OrderedDictiona ry`	Hashtable + array	Yes	59	70	70	40
Sorted						
`SortedDictionar y <K,V>`	Red/black tree	No	20	130	100	120
`SortedList <K,V >`	2xArray	Yes	2	3,300	30	40
`SortedList`	2xArray	Yes	27	4,500	100	180

پیچیدگی زمانی دریافت یک عنصر براساس key اینجوریه:

- O(1): Hashtables, Dictionary, OrderedDictionary
- O(log n): SortedDictionary, SortedList
- O(n): ListDictionary (and nondictionary types such as List<T>)

IDictionary<TKey,TValue>

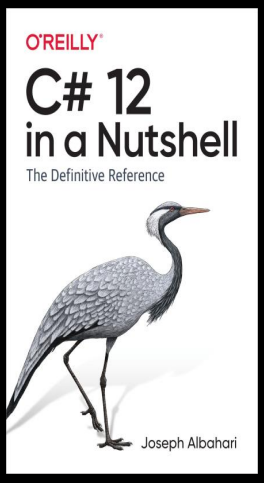


IDictionary<TKey,TValue> یک پروتکل استاندارد برای همه کالکشن‌های بر پایه key/value تعریف میکنه.

```
namespace System.Collections.Generic
{
    0 references | 0 changes | 0 authors, 0 changes
    public interface IDictionary<TKey, TValue> :
        ICollection<KeyValuePair<TKey, TValue>>, IEnumerable<KeyValuePair<TKey, TValue>>, IEnumerable
    {

        0 references | 0 changes | 0 authors, 0 changes
        TValue this[TKey key] { get; set; }
        0 references | 0 changes | 0 authors, 0 changes
        ICollection<TKey> Keys { get; }
        0 references | 0 changes | 0 authors, 0 changes
        ICollection<TValue> Values { get; }
        0 references | 0 changes | 0 authors, 0 changes
        void Add(TKey key, TValue value);
        0 references | 0 changes | 0 authors, 0 changes
        bool ContainsKey(TKey key);
        0 references | 0 changes | 0 authors, 0 changes
        bool Remove(TKey key);
        0 references | 0 changes | 0 authors, 0 changes
        bool TryGetValue(TKey key, [MaybeNullWhen(false)] out TValue value);
    }
}
```

اینترفیس IReadOnlyDictionary<TKey,TValue> هم برای تعریف دیکشنری با اعضای فقط خواندنی وجود دارد.

IDictionary<TKey,TValue>



برای اضافه کردن یک عنصر به دیکشنری میتونیم هم از متد Add استفاده کنیم و هم از indexer. با این روش اگر آیتم قبلا در دیکشنری نباشد به آن اضافه میشه و اگر وجود داشته باشه مقدارش آپدیت میشه.

در همه پیاده‌سازی‌های دیکشنری اجازه اضافه کردن کلید تکراری رو نداریم و اگر اینکار رو انجام بدیم به خطا میخوریم:

```
IDictionary<int, string> dict = new Dictionary<int, string>();
dict.Add(1, "value1");
dict[2] = "value 2";
dict[2] = "value2";
dict.Add(2, "new value2");


#endregion
```

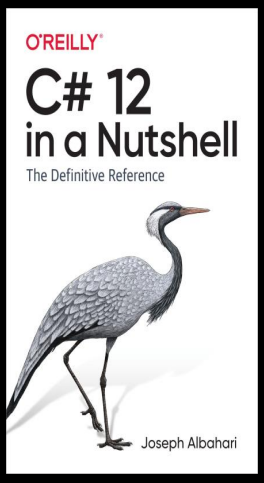
Exception Unhandled

System.ArgumentException: 'An item with the same key has already been added. Key: 2'

Show Call Stack | View Details | Copy Details | Start Live Share session

Exception Settings

IDictionary<TKey,TValue>

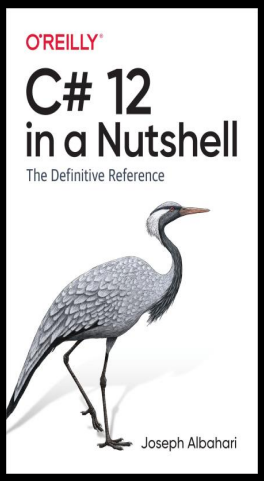


برای خواندن یک عنصر میتونیم هم از متد TryGetValue استفاده کنیم و هم از indexer. اگر کلید وجود نداشته باشه indexer خطا میده درصورتیکه متد TryGetValue مقدار false برمیگردونه.

برای چک کردن اینکه یک کلید در دیکشنری وجود داره یا نه میتونید از متد ContainsKey استفاده کنید اگرچه این باعث میشه تا هزینه دوبار خواندن دیتا رو متحمل بشید.

```
var item1 = dict[1];
if (dict.ContainsKey(3))
{
    var item3 = dict[3];
}

var hasValue2 = dict.TryGetValue(2, out var item2);
```

IDictionary<TKey,TValue>

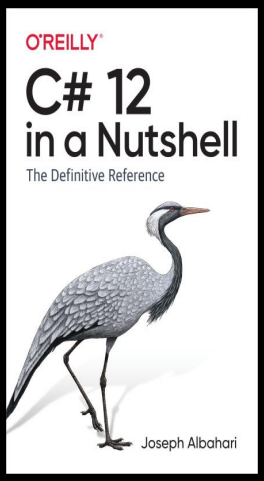


برای پیمایش یک دیکشنری میتونید مستقیم این کار رو انجام بدید و در نتیجه در هر پیمایش یک ساختار KeyValuePair خواهیم داشت:

```
public readonly struct KeyValuePair<TKey, TValue>
{
    private readonly TKey key; // Do not rename (binary serialization)
    private readonly TValue value; // Do not rename (binary serialization)

    public KeyValuePair(TKey key, TValue value)...

    public TKey Key => key;

    public TValue Value => value;

    public override string ToString()...
    public void Deconstruct(out TKey key, out TValue value)...
}
```

همینطور اگه خواستید میتونید با استفاده از پراپرتی‌های keys/values روی کلید یا مقدار پیمایش کنید:

```
foreach (var item in dict)
{
    Console.WriteLine($"Key: {item.Key}, Value: {item.Value}");
}
```



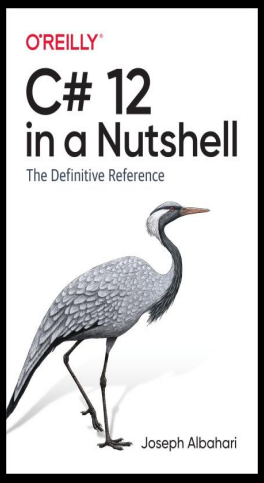
IDictionary

IDictionary کاملا شبیه IDictionary<TKey,TValue> هستش با دو تفاوت:

- اگر با indexer بخواهیم یک کلید رو پیدا کنیم که وجود نداره مقدار null برمیگردونه. (به جای خطا دادن)
- برای بررسی وجود داشتن یک کلید متد contains رو معرفی میکنه(به جای متد ContainsKey)

```
public interface IDictionary : ICollection, IEnumerable
{
    object? this[object key] ...
    bool IsFixedSize { get; }
    bool IsReadOnly { get; }
    ICollection Keys { get; }
    ICollection Values { get; }

    void Add(object key, object? value);
    void Clear();
    bool Contains(object key);
    IDictionaryEnumerator GetEnumerator();
    void Remove(object key);
}
```

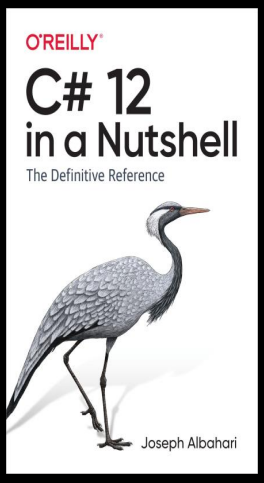
IDictionary



برای پیمایش یک IDictionary میشه از foreach استفاده کرد و هریک از آیتمها یک آبجکت از نوع DictionaryEntry هستند.

```
IDictionary sampleDictionary = new Hashtable();
sampleDictionary.Add(1, "value1");
sampleDictionary.Add(2, "value2");
foreach (DictionaryEntry item in sampleDictionary)
{
    Console.WriteLine($"Key: {item.Key}, Value: {item.Value}");
}
```

Dictionary<TKey,TValue> and Hashtable

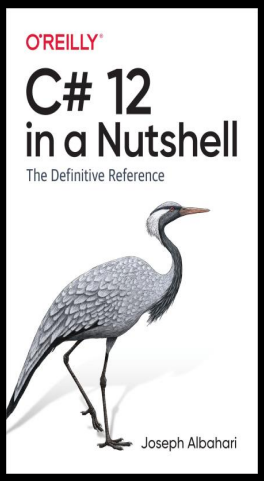


کلاس جنریک Dictionary یکی از پرکاربردترین کالکشن‌ها (در کنار List<T>) هست. این کلاس از یک ساختار دیتای hashtable برای نگهداری key و value استفاده میکنه که سرعت بالایی داره.

نسخه غیرجنریک Dictionary<TKey,TValue> کلاسی است با نام Hashtable. و کلاس غیرجنریکی به اسم Dictionary وجود نداره، برای همین برای راحتی متن، ازین به بعد هرجا از کلمه Dictionary استفاده میشه منظور همون کلاس Dictionary<TKey,TValue> هست.

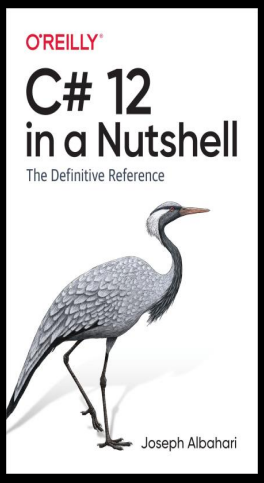
کلاس Dictionary هر دو اینترفیس جنریک و غیرجنریک IDictionary رو پیاده‌سازی کرده، در ادامه میتونید یه مثال از نحوه استفاده ش ببینید:

Dictionary<TKey,TValue> and Hashtable




```
var d = new Dictionary<string, int>();

d.Add("One", 1);
d["Two"] = 2;     // adds to dictionary because "two" isn't already present
d["Two"] = 22;    // updates dictionary because "two" is now present
d["Three"] = 3;

Console.WriteLine(d["Two"]);                // Prints "22"
Console.WriteLine(d.ContainsKey("One"));    // true (fast operation)
Console.WriteLine(d.ContainsValue(3));      // true (slow operation)
int val = 0;
if (!d.TryGetValue("onE", out val))
    Console.WriteLine("No val");              // "No val" (case sensitive)

// Three different ways to enumerate the dictionary:

foreach (KeyValuePair<string, int> kv in d)          //  One; 1
    Console.WriteLine(kv.Key + "; " + kv.Value);      //  Two; 22
                                                      //  Three; 3

foreach (string s in d.Keys) Console.Write(s);      // OneTwoThree
Console.WriteLine();
foreach (int i in d.Values) Console.Write(i);       // 1223
```

OrderedDictionary



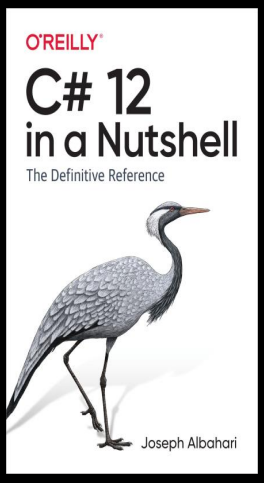
OrderedDictionary یک پیاده سازی غیرجنریک از اینترفیس IDictionary هست که اجازه میده آیتم‌ها به همون ترتیب اضافه شدن به دیکشنری ذخیره بشن و به همین خاطر میتونیم هم از طریق key و هم از طریق index به آیتم‌ها دسترسی داشته باشیم.

```
OrderedDictionary orderedDictionary = new OrderedDictionary();
orderedDictionary.Add("1", "value1");
orderedDictionary.Add("3", "value3");
orderedDictionary.Add("2", "value2");

Console.WriteLine(orderedDictionary[3]);
```

این کلاس در اصل ترکیبی از کلاسهای Hashtable و ArrayaList هستش و همه عملکردهای این دو کلاس رو میتونه انجام بده.

این کلاس از زمان .Net 2 معرفی شده و هنوز هم نسخه جنریکی براش ارائه نشده.

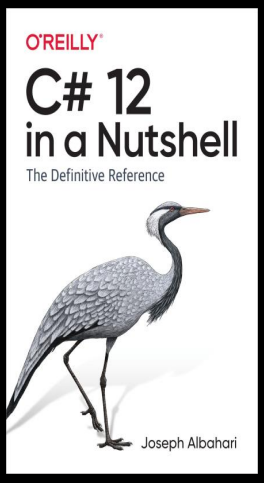
ListDictionary and HybridDictionary



ListDictionary از یک linked likst برای نگهداری اطلاعات استفاده میکنه و آیتمها رو به همون شکل ابتدایی که به دیکشنری اضافه شدند نگه میداره. این کلاس برای حجم بالای دیتا به شدت کنده و تنها برای داده‌های کمتر از ده عدد میتونه مفید باشه.

HybridDictionary هم همون ListDictionary هستش که به صورت خودکار به hashtable تبدیل میشه و اومده که مشکلات قبلی رو حل کنه ولی خب خیلی موفق نبوده.

(نظر خودم: خلاصه اینکه تا حالا نمیدونستیم چیه چیزی رو از دست ندادیم ازین به بعد هم کاری باهاش نداشته باشیم بهتره)



Sorted Dictionaries

BCL در دات‌نت دو ساختار برامون فراهم کرده که آیتم‌هاش همیشه براساس key مرتب میشند:

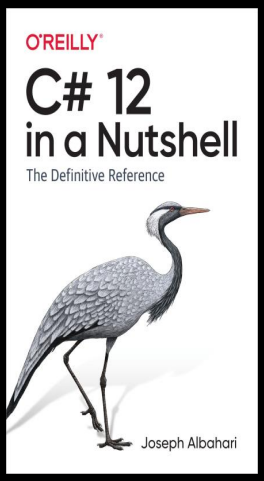
SortedDictionary<TKey,TValue>
SortedList<TKey,TValue>

SortedDictionary از ساختار درختی red/black استفاده میکنه که یک ساختار دیتای بسیار قوی برای خوندن یا درج یک آیتم در دیکشنری است.

SortedList از یک زوج آرایه داخلی برای نگهداری آیتمهاش استفاده میکنه و ازونجایی که برای اضافه شدن هر عنصر جدید نیازه که آیتم‌ها شیفت پیدا کنند، سرعت درج آیتم در اون از SortedDictionary کمتره.

امکان پیمایش هم روی هر دونوع وجود داره و اینکه مثل دیکشنری‌های دیگه، امکان اضافه کردن آیتم تکراری وجود نداره.

Sorted Dictionaries



اینم یه مثال ساده و کلی از دونوع معرفی شده در اسلاید قبلی:

```
SortedDictionary<string,string> sortedDict = new SortedDictionary<string,string>();
sortedDict.Add("Ali", "value1");
sortedDict.Add("Negar", "value1");
sortedDict.Add("Omid", "value1");

// MethodInfo is in the System.Reflection namespace

var sorted = new SortedList<string, MethodInfo>();

foreach (MethodInfo m in typeof(object).GetMethods())
    sorted[m.Name] = m;

foreach (string name in sorted.Keys)
    Console.WriteLine(name);
/*
Equals
GetHashCode
GetType
ReferenceEquals
ToString
 */
foreach (MethodInfo m in sorted.Values)
    Console.WriteLine(m.Name + " returns a " + m.ReturnType);
/*
Equals returns a System.Boolean
GetHashCode returns a System.Int32
GetType returns a System.Type
ReferenceEquals returns a System.Boolean
ToString returns a System.String
 */
```