List and dictionaries

Agenda

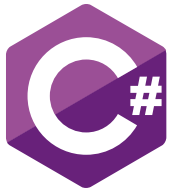
Lists

Dictionaries

Dictionaries under the hood

IEqualityComparer interface

Lists




To Do List
1.
2.
3.
4.
5.

List of objects

```
List<T>
```

List

- Index based access collection
- Provides some features that are provided by arrays
- It allows to add or remove elements
- Dynamic number of elements

Index

Each element has an index

An element can be accessed with its index

List class

The `List` class represents a list

Implements the `ICollection` and `IList`

List declaration and instantiation

```
var list = new List<string>();
```

List declaration and initialization

```
var list = new List<string>(){
    "Mercury",
    "Venus",
    "Earth",
    "Mars",
    "Jupiter",
    "Saturn",
    "Uranus",
    "Neptune"
};
```

List declaration and initialization

```
List<string> list = new() {
    "Mercury",
    "Venus",
    "Earth",
    "Mars",
    "Jupiter",
    "Saturn",
    "Uranus",
    "Neptune"
};
```

Add/Remove elements

`Add(T)` adds an element at the end

`Insert(int,T)` insert an element anywhere

`Remove(T)` remove an element

`RemoveAt(int)` removes an element at the index specified

`RemoveAll()` removes all the elements from the list

Add elements

```
var list = new List<string>();
list.Add("Mercury");
list.Add("Venus");
list.Add("Earth");
list.Add("Mars");
list.Add("Jupiter");
list.Add("Saturn");
list.Add("Uranus");
list.Add("Neptune");
```

Retrieve an element

```
string first = list[0];
```

Other members

`Count` number of elements

`Capacity` number of elements in the internal data structure that contains the elements can hold

`Contains(T)` indicates if the list contains an element

`IndexOf(T)` gets the index of an element

`Reverse()` reverses the order of the elements

`Sort()` sorts the elements

`ToArray()` creates a new array and copies the elements in it

Under the hood

Internally, a list contains a data structure that contains the elements

The structure size is fix and has a length equal to its capacity

If there is not enough, a new array with double capacity is created

List performance

Inserting/Removing elements is slow

When inserting an element, all the elements after the insertion must be moved

List performance

Add → O(1) or O(n)

Contains → O(n)

Insert → O(n)

Remove → O(n)

RemoveAt → O(n)

[] → O(1)

Arrays/Lists

List are more convenient to use and dynamic

Arrays are more performant and fixed size

For better performance, use arrays over lists if you can use an array

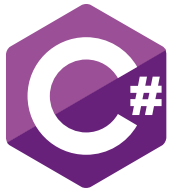
Demo

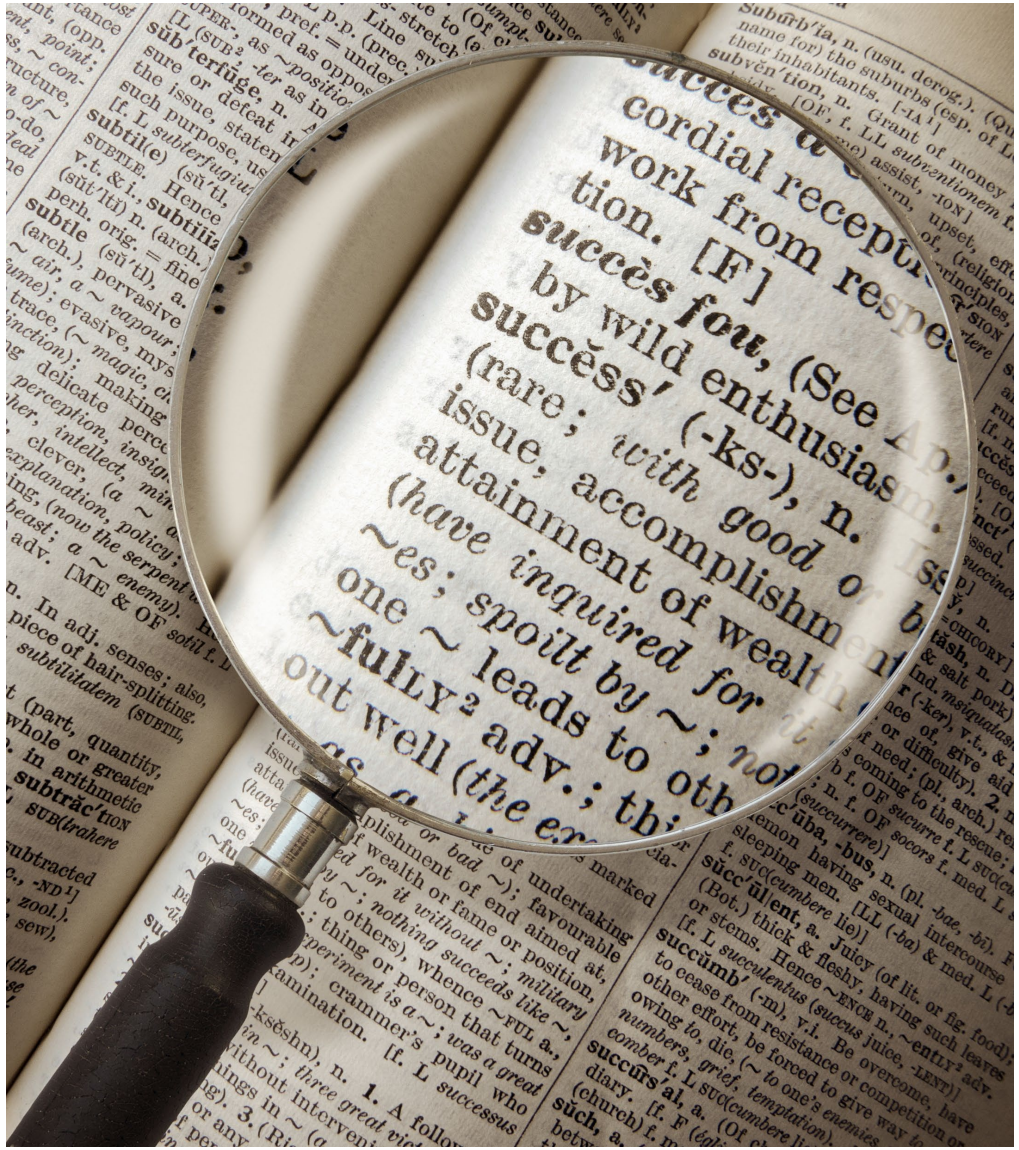
Declare/Instantiate/initialize lists

Use list methods

Sort lists using IComparer implementation

Dictionary




cordial recepti
work from respe
tion. [F]
succès fou, (See
by wild enthusias
succĕss′ (-ks-), n.
(rare; with good o
issue, accomplishm
attainment of wealt
(have inquired for
~es; spoilt by ~; not
one ~ leads to oth
~fully² adv.; thi
out well (the ex

Dictionary

Strongly typed collection of key/value pairs

Collection of key/value pairs

```
Dictionary<TKey,TValue>
```

Key/Value

Each object has a unique key, can be of any type unlike arrays and dictionaries

The actual object is stored in the value

A key can’t be null

Dictionary class

Represent a collection of key/value pairs

Implements `IEnumerable<KeyValuePair<Tkey,Tvalue>>`

Implements `ICollection` and `IDictionary`

Dictionary declaration and instantiation

```
var dictionary = new Dictionary<int,Movie>();
```

Dictionary declaration and initialization

```
var dictionary = new Dictionary<int,Movie>
{
    {1, new Movie{Id = 1, Title = "Title 1"}},
    {2, new Movie{Id = 2, Title = "Title 2"}},
    {3, new Movie{Id = 3, Title = "Title 3"}}
};
```

Add/Remove elements

`Add(TKey,TValue)` adds an element

`Remove(TKey)` remove an element by its key

`TryAdd(TKey,TValue)` attempts to add an element

Add elements

```
dictionary.Add(4, new Movie{Id = 4, Title = "Title 4"});
```

Retrieve an element

```
var first = dictionary[1];
```

Order

Dictionaries are not ordered

You can’t rely on the order of a foreach

Enumerate over a dictionary

```
foreach (KeyValuePair<int,Movie> keyValuePair in dictionary)
{
    Console.WriteLine(keyValuePair.Value.Title);
}
```

Other members

`Count` number of elements

`Capacity` number of elements in the internal data structure that contains the elements can hold

`ContainsKey(TKey)` Determines if contains the specified key

`ContainsValue(TValue)` Determines if contains the specified value

`TryGetValue(TKey,TValue)` Gets the value associated with the specified key

Other members

`Keys` collection of contained keys

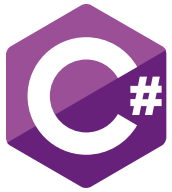
`Values` collection of contained Values

Demo

Declare/Instantiate/initialize dictionaries

Use dictionary methods

Dictionaries under the hood



Capacity

The internal structure size is fix and has a length equal to its capacity

If the size is reached, the capacity is doubled

Hash table

The internal data structure that contains the elements is a hash table

A hash table is a key/value pair mapping structure

A hash table made of buckets

Key hash

When an element is inserted, a hash of the key is calculated (`GetHashCode` method)

The key hash is used to store an element in a particular bucket

Key hash

The key hash is also used to look up for an element

Dictionary performance

Adding, removing and retrieving an element is fast

Dictionary performance

Add → O(1) or O(n)

Remove → O(1)

[] → O(1)

Dictionaries are great for non-integer-based indexing, retrieving, adding or removing elements

Demo

Using IEqualityComparer interface with dictionaries

Summary

The List collection is an indexed based collection

The dictionary collection is a collection of key value pairs

Lists are performant for adding and accessing elements

Dictionaries are performant for accessing, adding and removing elements