Collection Interfaces

Simon Robinson
http://TechieSimon.com
@TechieSimon

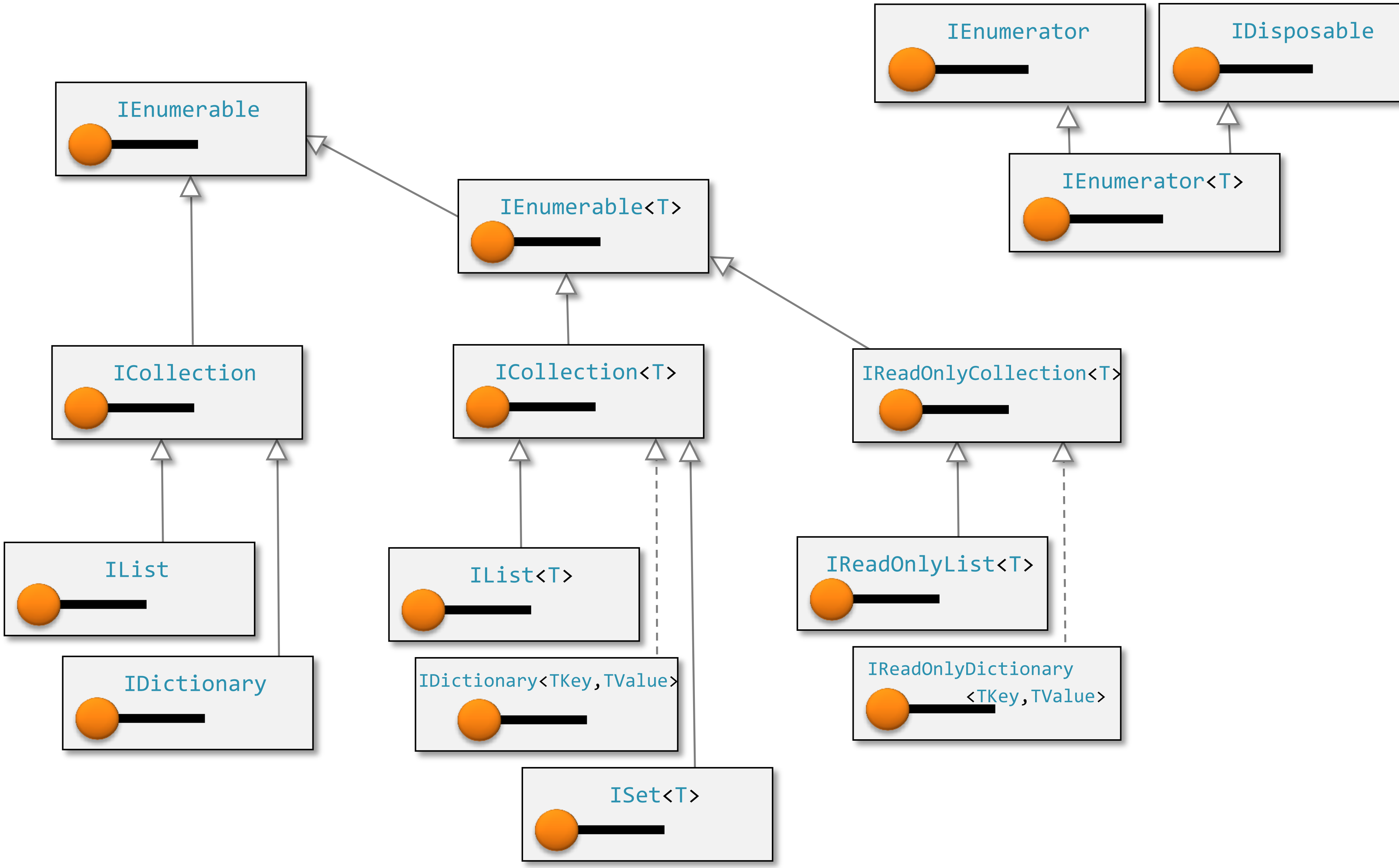
Benefits of learning the interfaces…

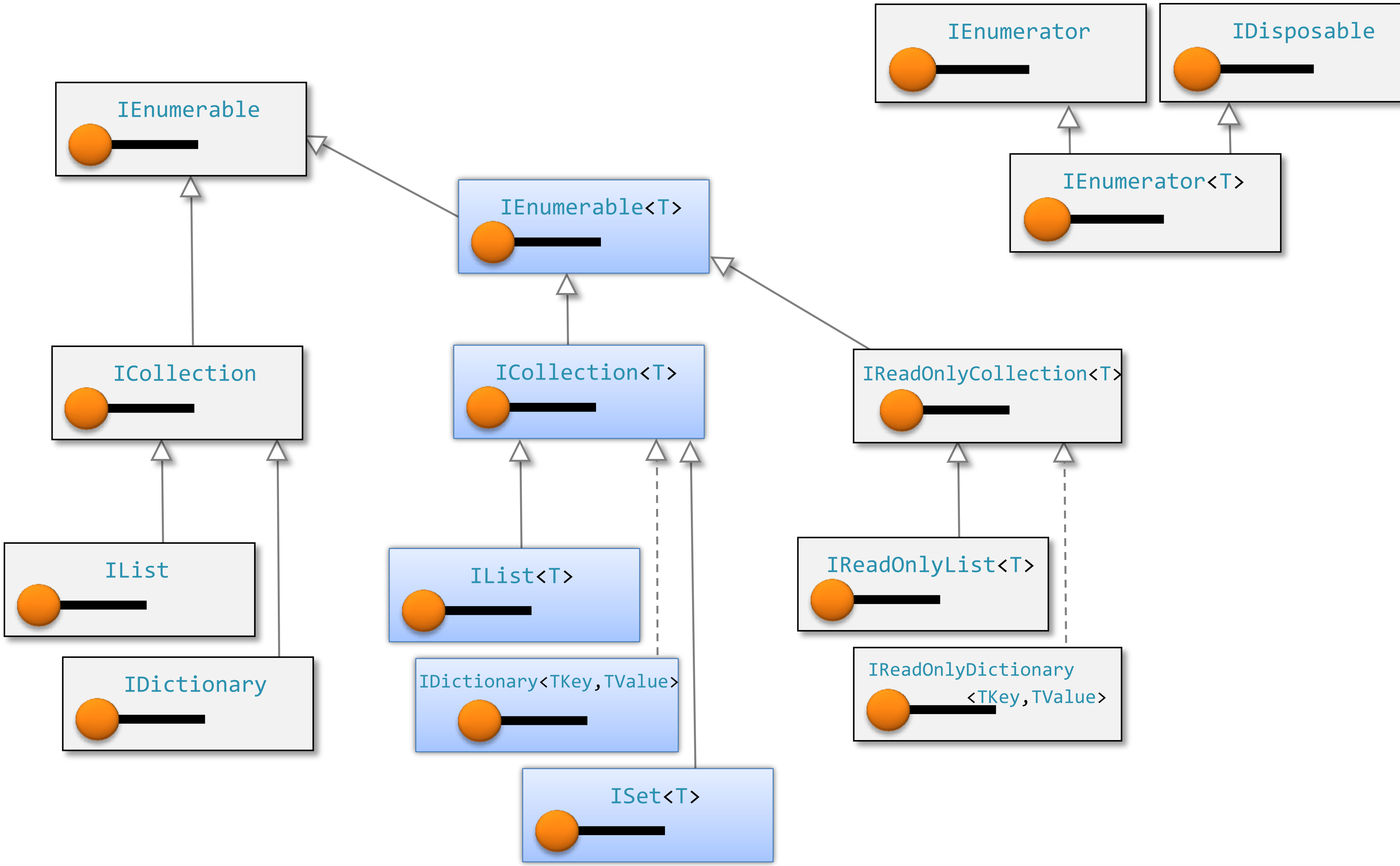
Create loosely coupled code ✓

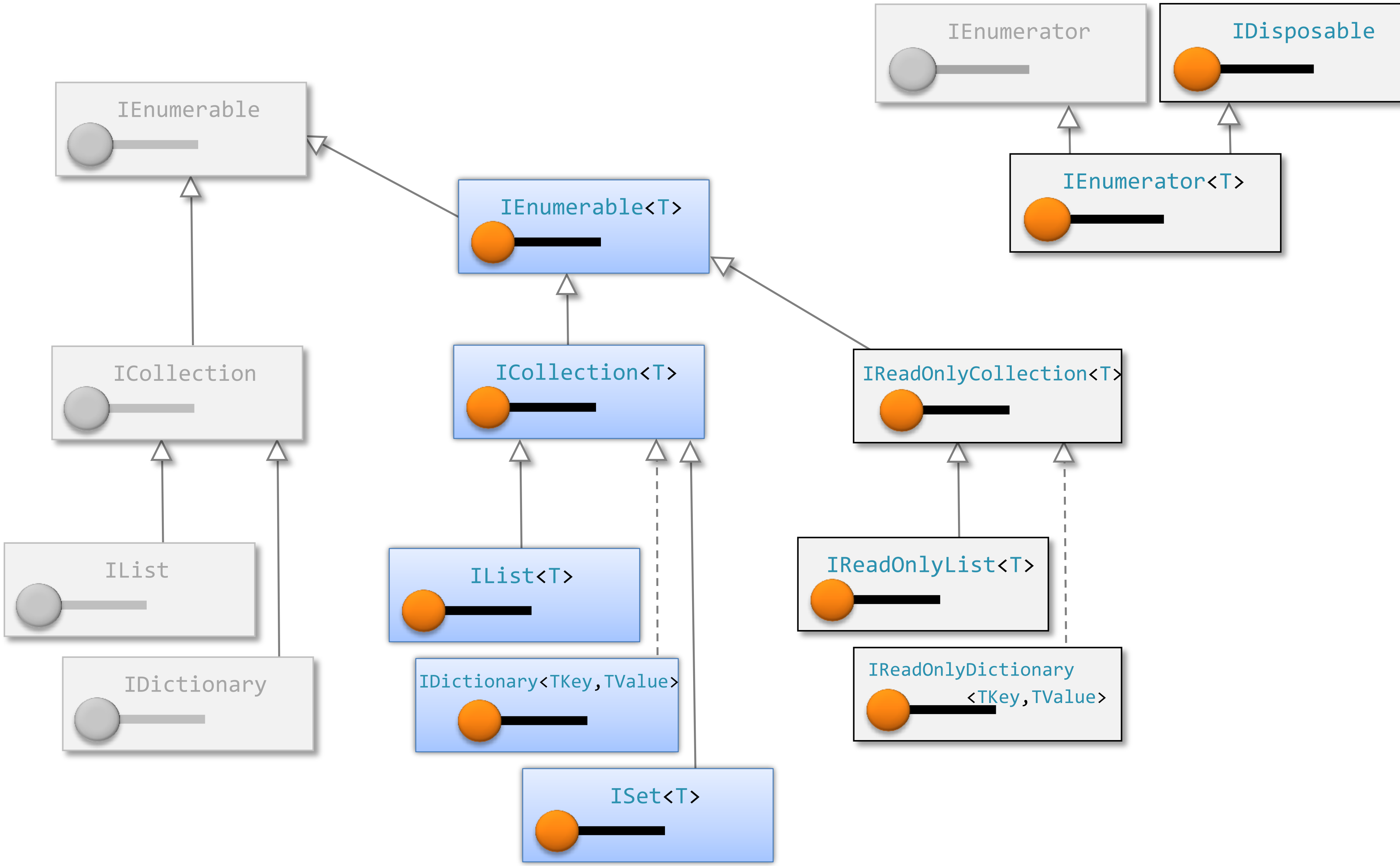
Understand MS collections better ✓

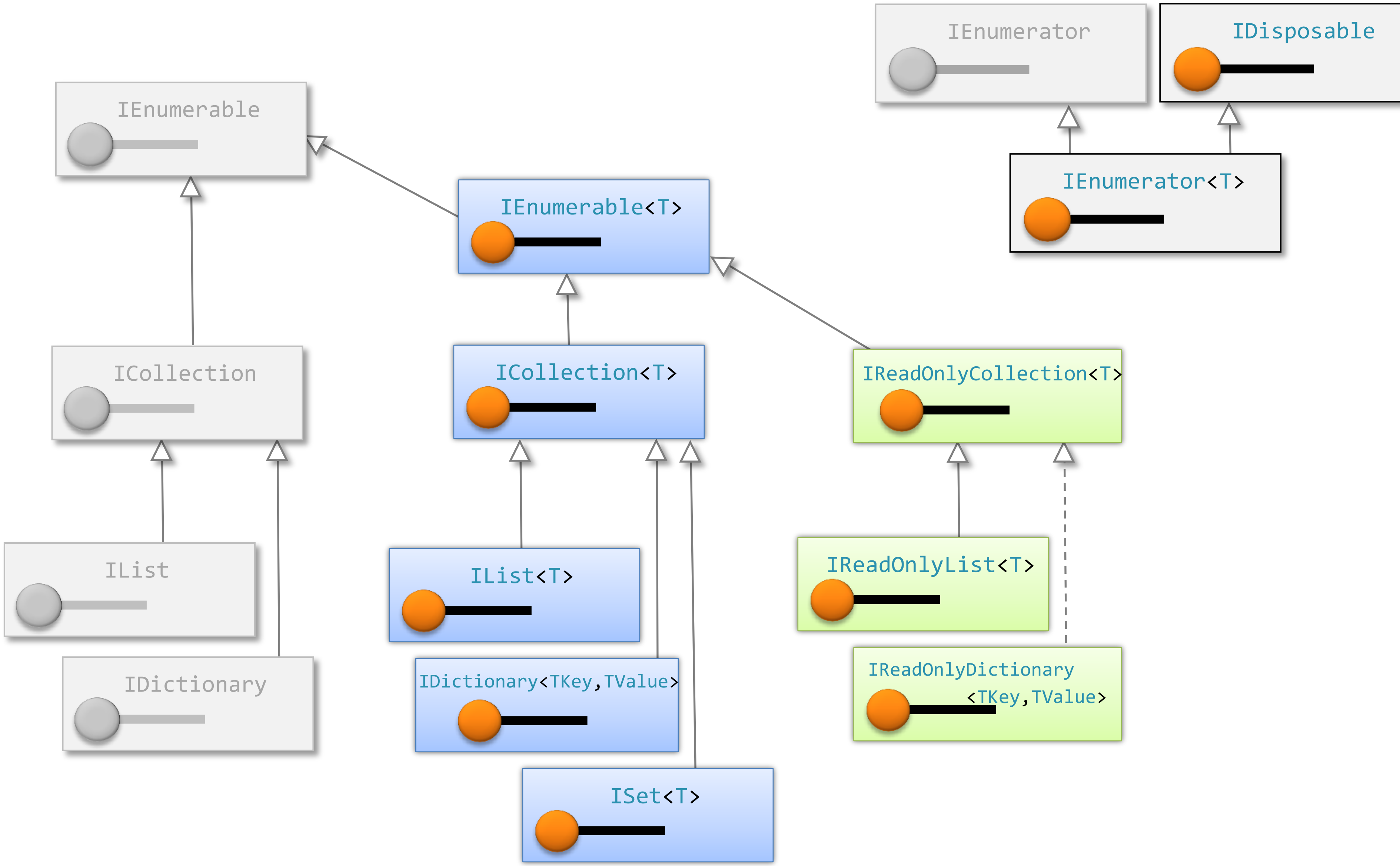
Module Overview

- **Interface hierarchy**
  - Core generic interfaces
  - Read-only interfaces
  - Older non-generic interfaces
- **Specific intefaces**
  - `IEnumerable<T>`
  - `ICollection<T>`
  - `IList<T>`
  - `IDictionary<TKey, TValue>`
  - `ISet<T>`
- **Explicit implementation**

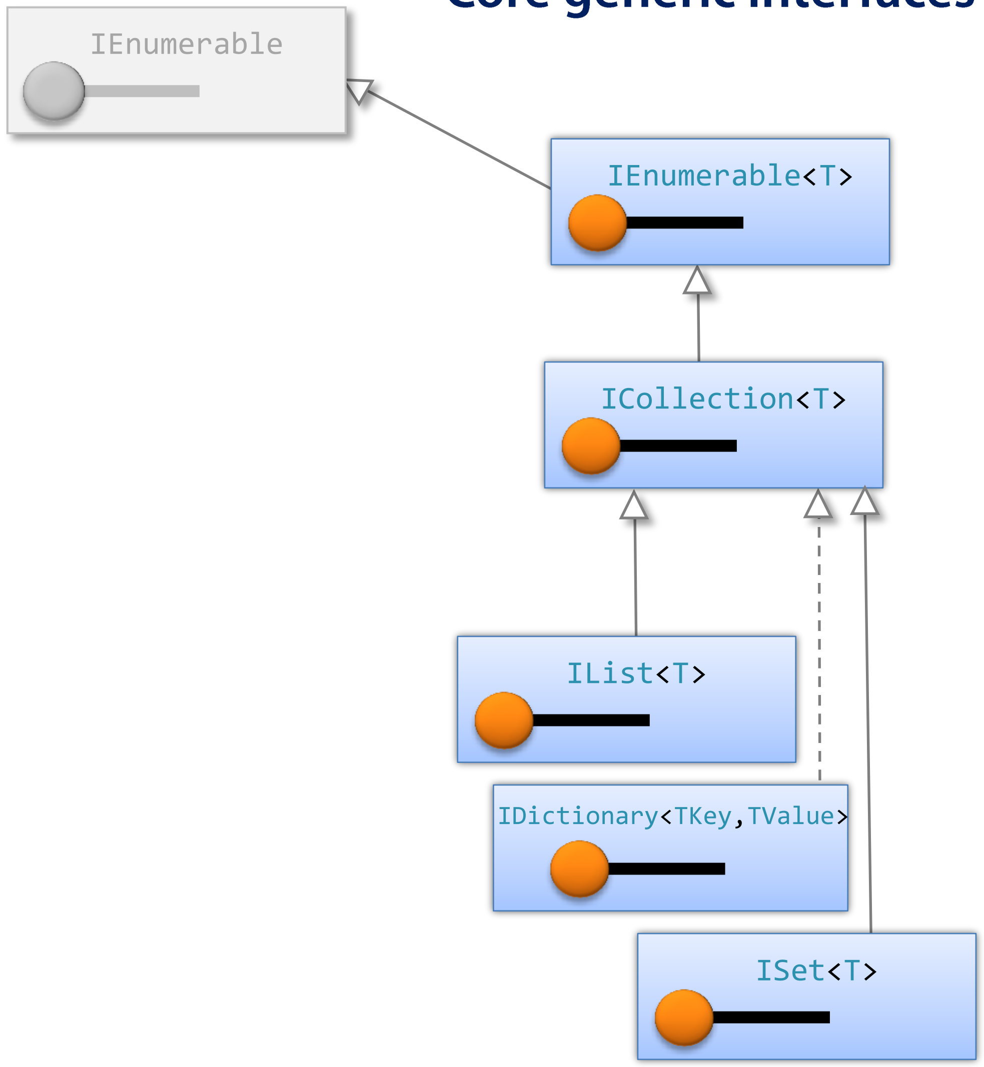

IEnumerator
IDisposable
IEnumerable
IEnumerable<T>
IEnumerator<T>
ICollection
ICollection<T>
IReadOnlyCollection<T>
IList
IList<T>
IReadOnlyList<T>
IDictionary
IDictionary<TKey,TValue>
IReadOnlyDictionary <TKey,TValue>
ISet<T>

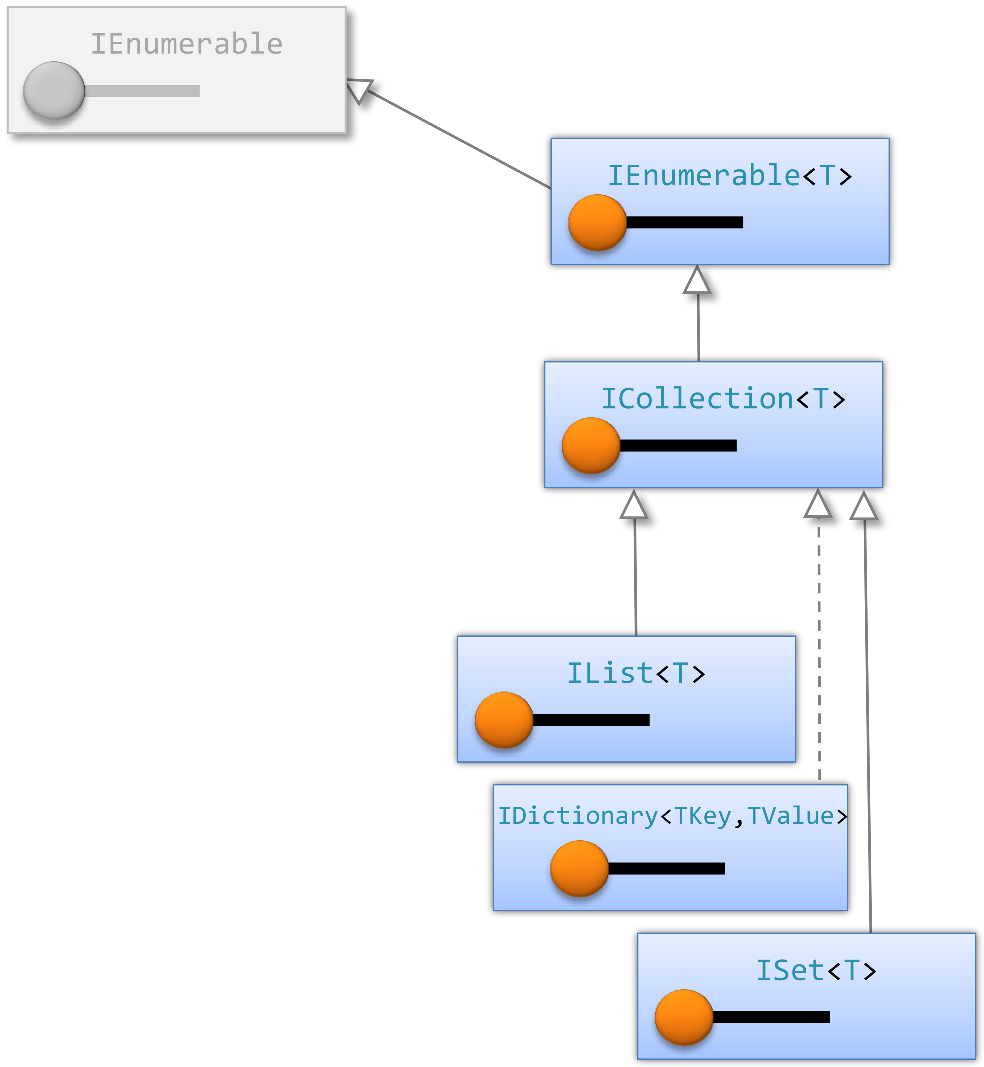

IEnumerator
IDisposable
IEnumerable
IEnumerable<T>
IEnumerator<T>
ICollection
ICollection<T>
IReadOnlyCollection<T>
IList
IList<T>
IReadOnlyList<T>
IDictionary
IDictionary<TKey,TValue>
IReadOnlyDictionary
<TKey,TValue>
ISet<T>


IEnumerator
IDisposable
IEnumerable
IEnumerable<T>
IEnumerator<T>
ICollection
ICollection<T>
IReadOnlyCollection<T>
IList
IList<T>
IReadOnlyList<T>
IDictionary
IDictionary<TKey,TValue>
IReadOnlyDictionary<TKey,TValue>
ISet<T>

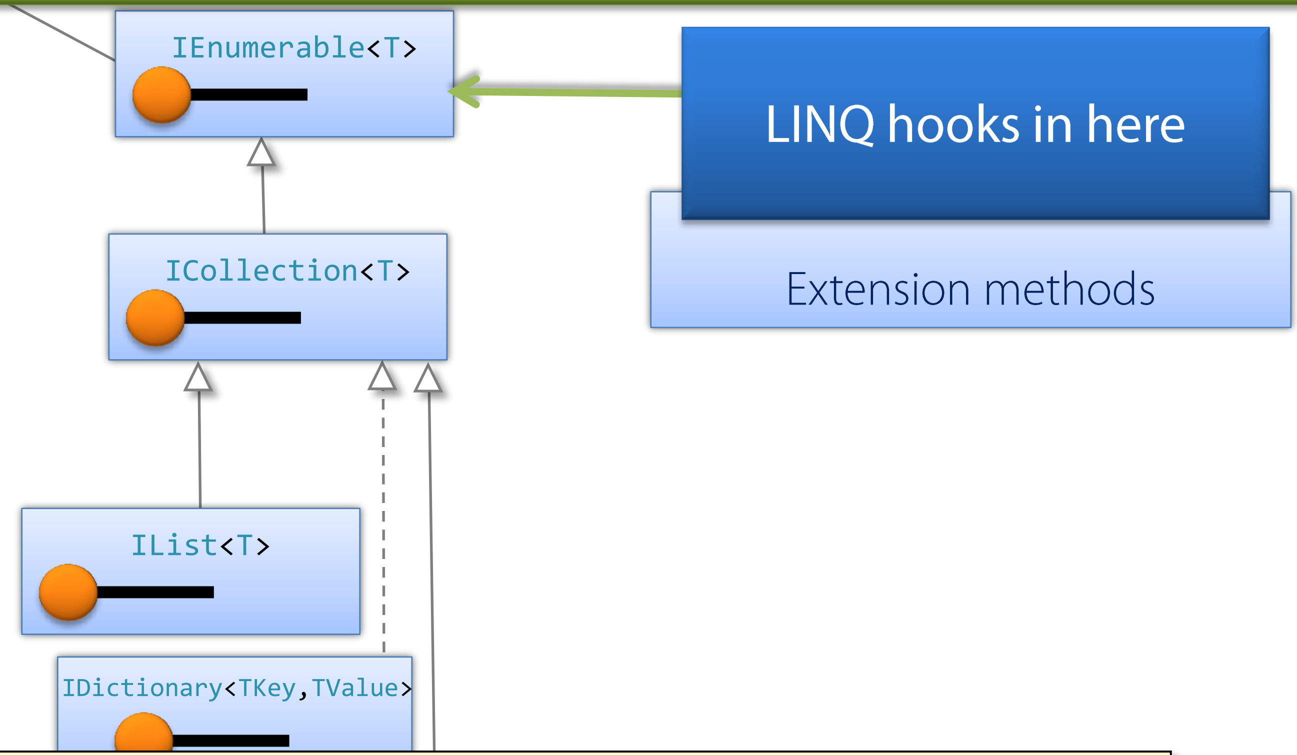

IEnumerator
IDisposable
IEnumerable
IEnumerable<T>
IEnumerator<T>
ICollection
ICollection<T>
IReadOnlyCollection<T>
IList
IList<T>
IReadOnlyList<T>
IDictionary
IDictionary<TKey,TValue>
IReadOnlyDictionary
<TKey,TValue>
ISet<T>

Core generic interfaces




IEnumerable
IEnumerable<T>
ICollection<T>
IList<T>
IDictionary<TKey,TValue>
ISet<T>

IEnumerable<T>:
"You can iterate my elements"



```
var bigControlNames = from control in controls
                      where control.Width > 400
                      select control.Name;
```

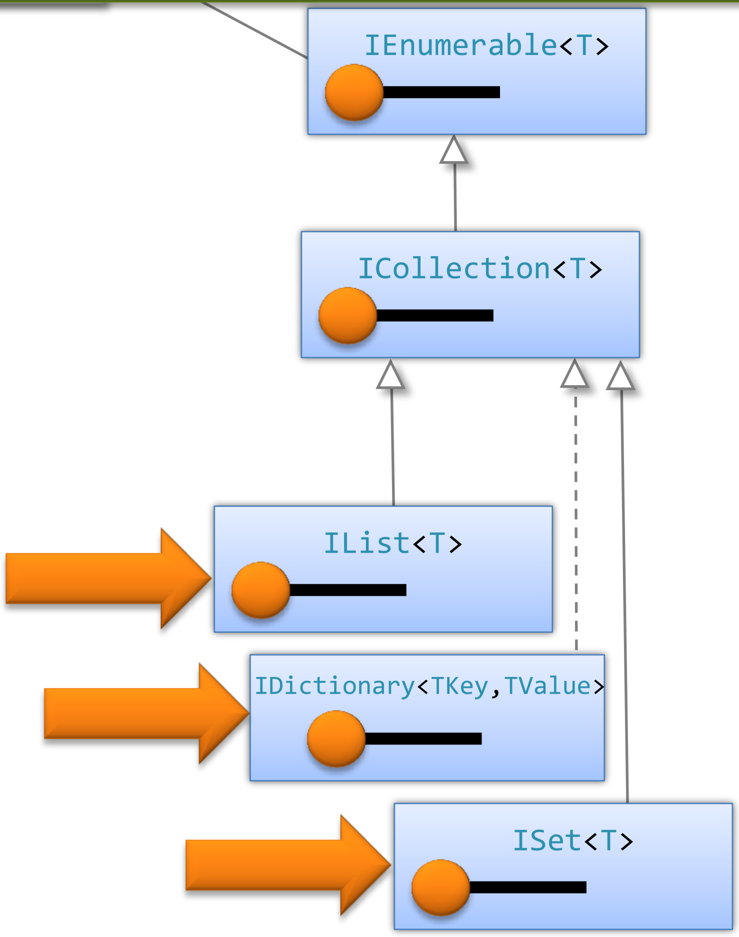
ICollection vs IEnumerable

SQL
Too many to fit in memory at once?
Data records
This is enumerable but not a collection!

ICollection<T>:

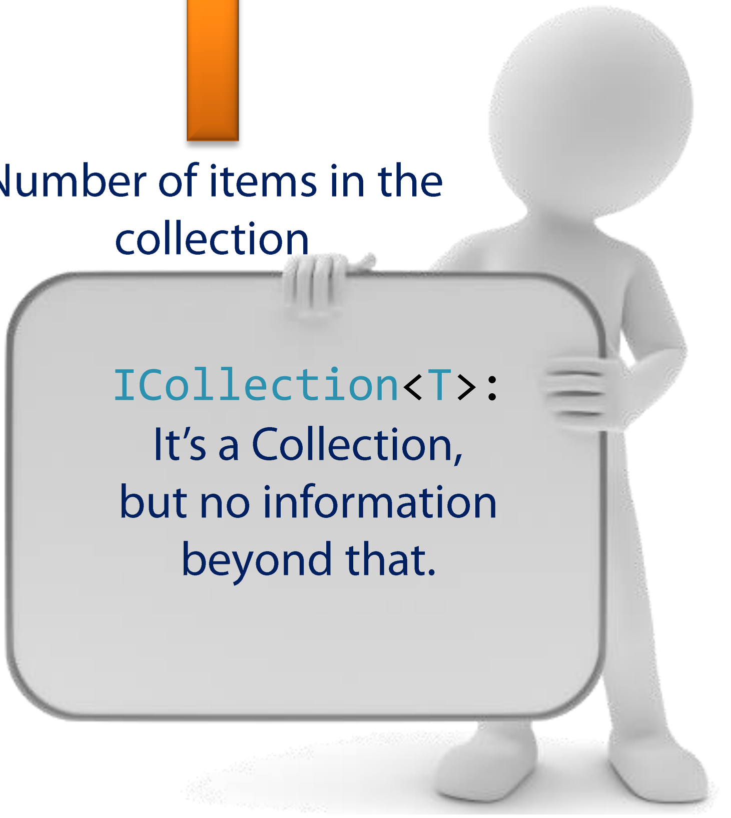
“I know how many elements I have”

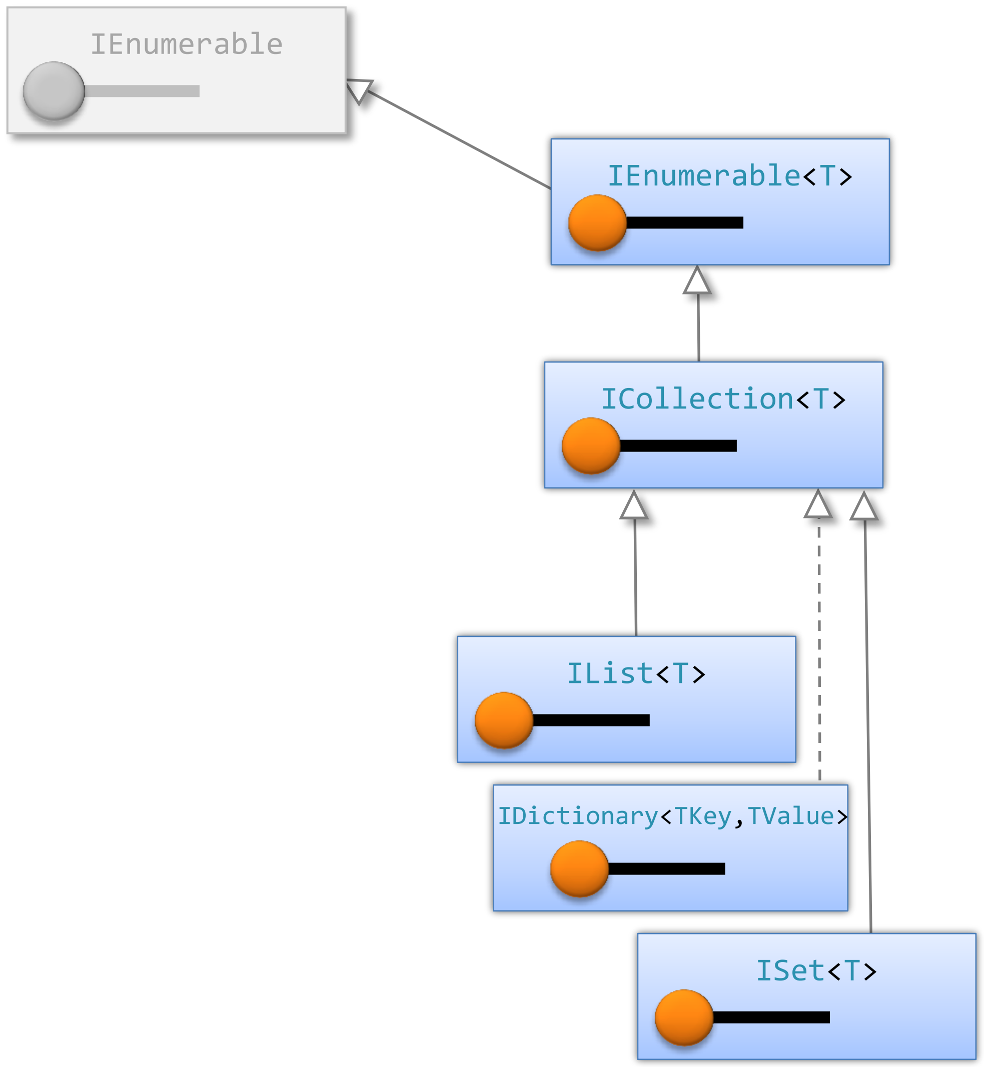
“You can modify my contents”

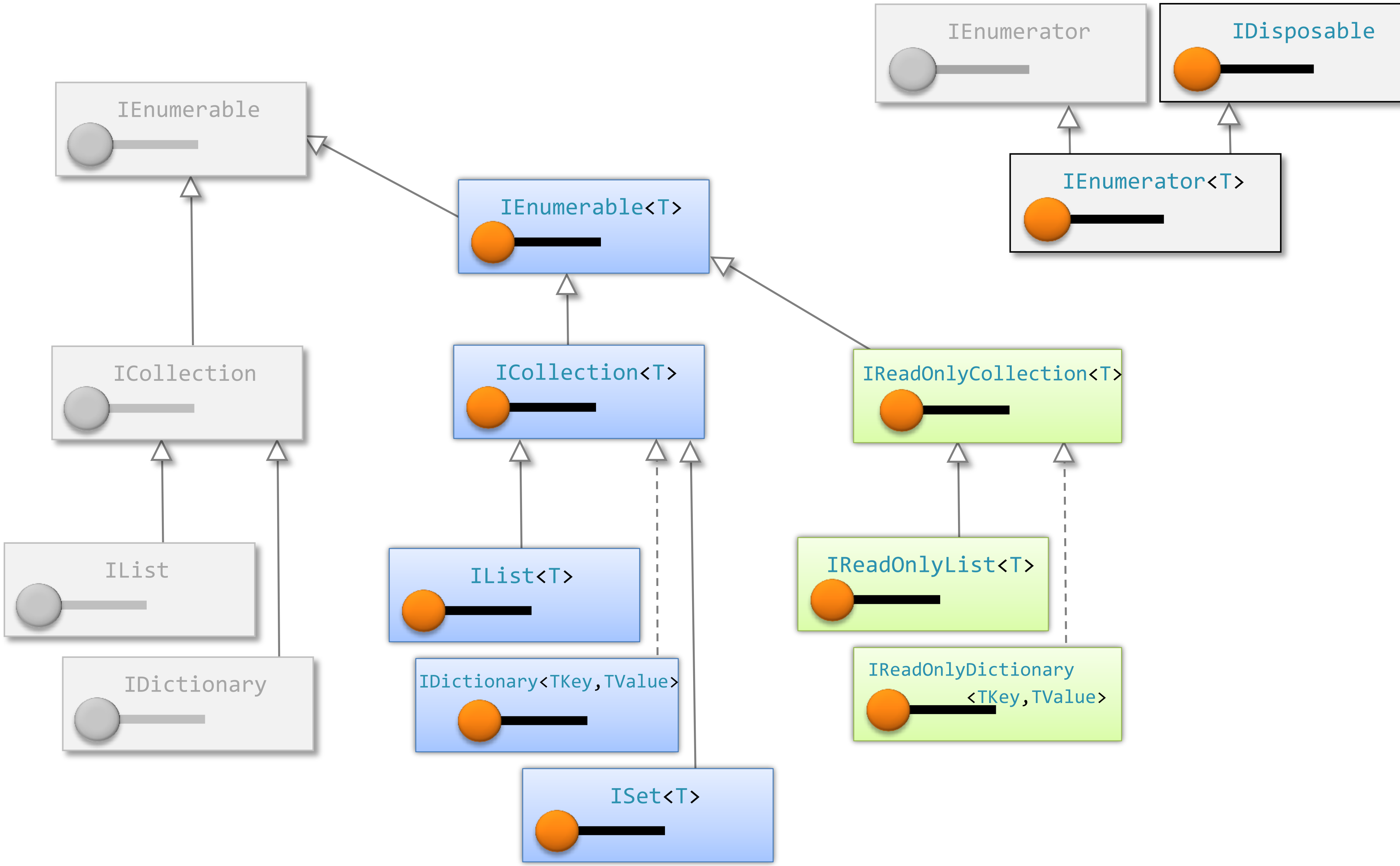


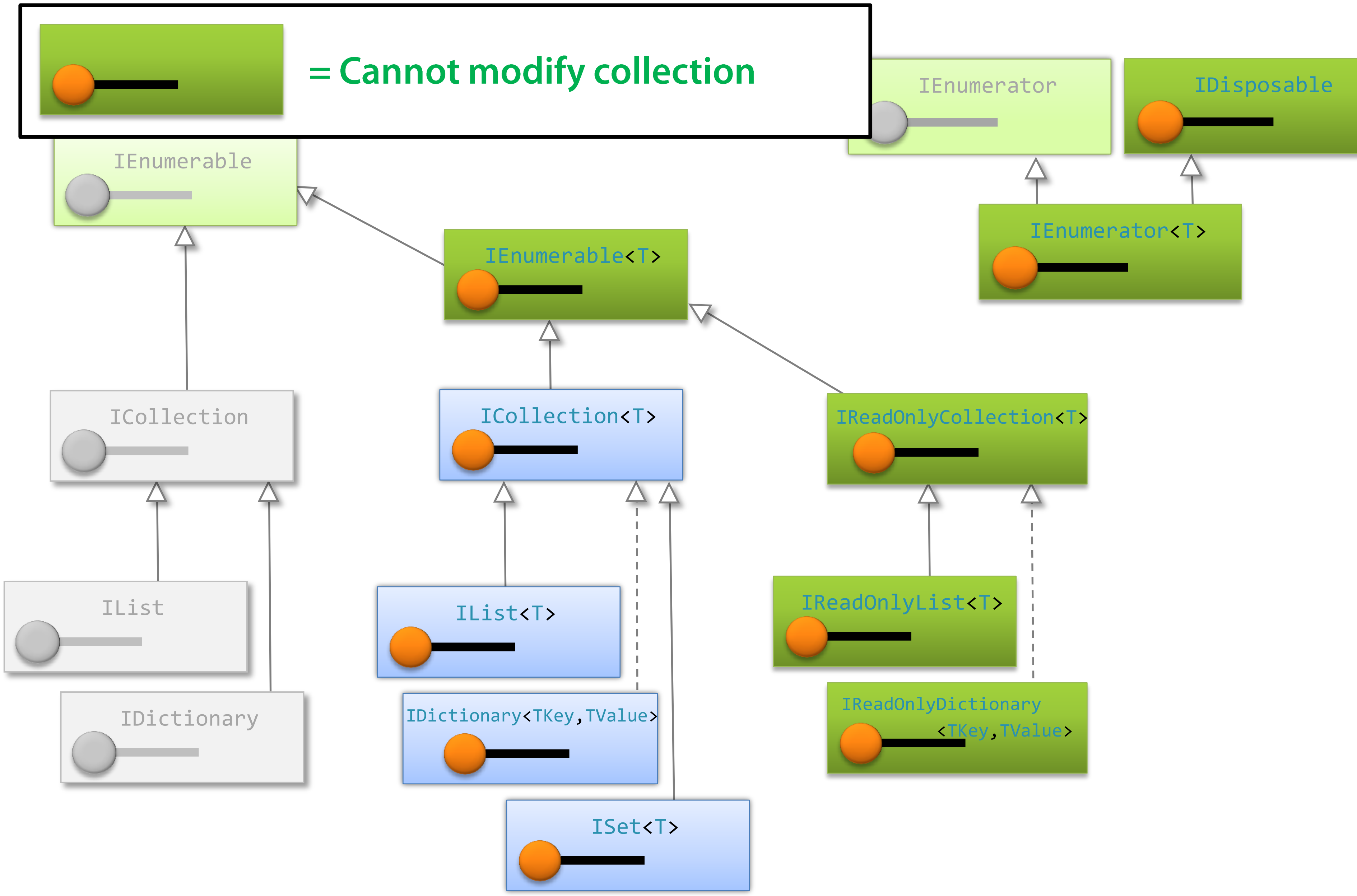
```
int ICollection<T>.Count
```

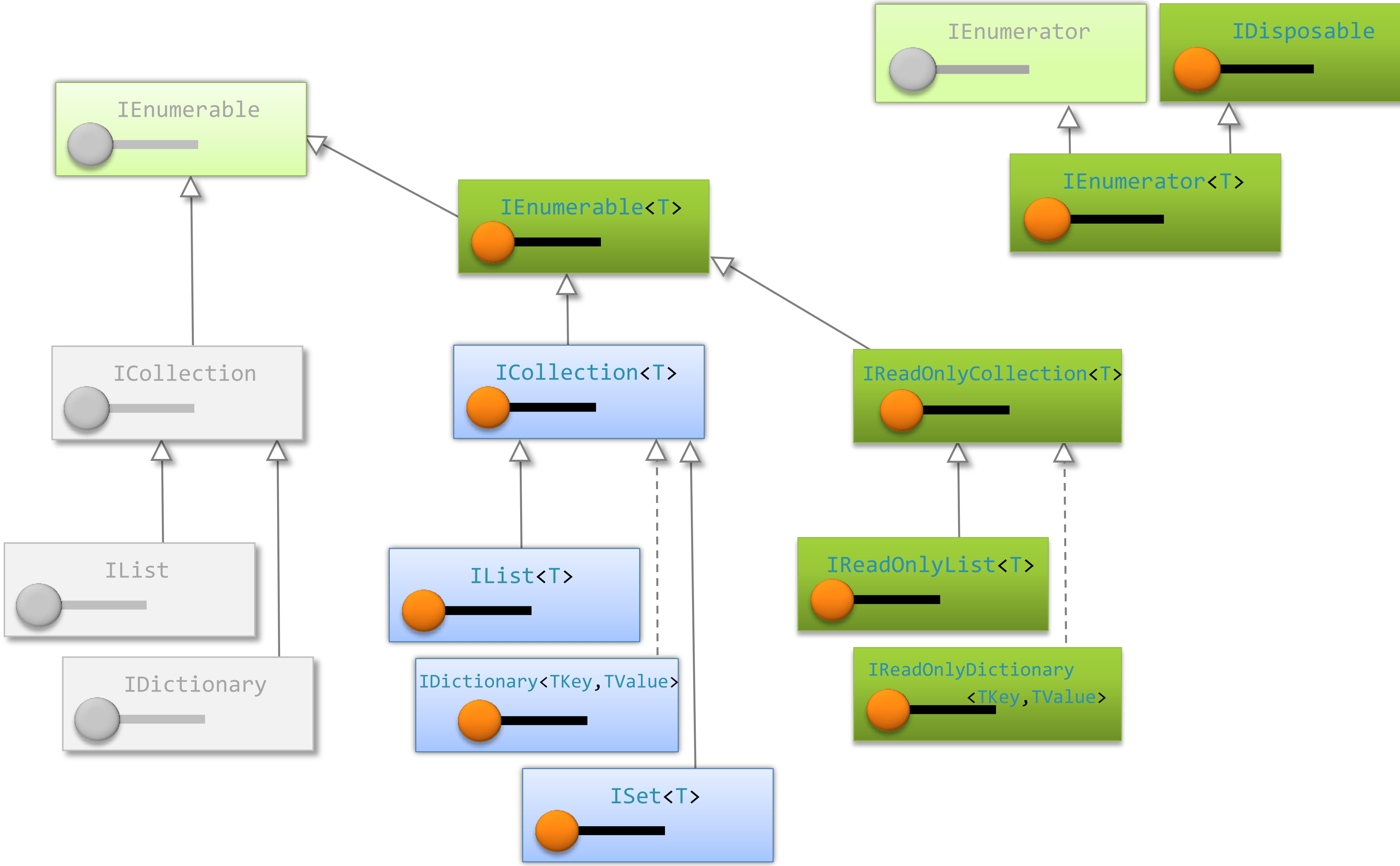
Number of items in the collection



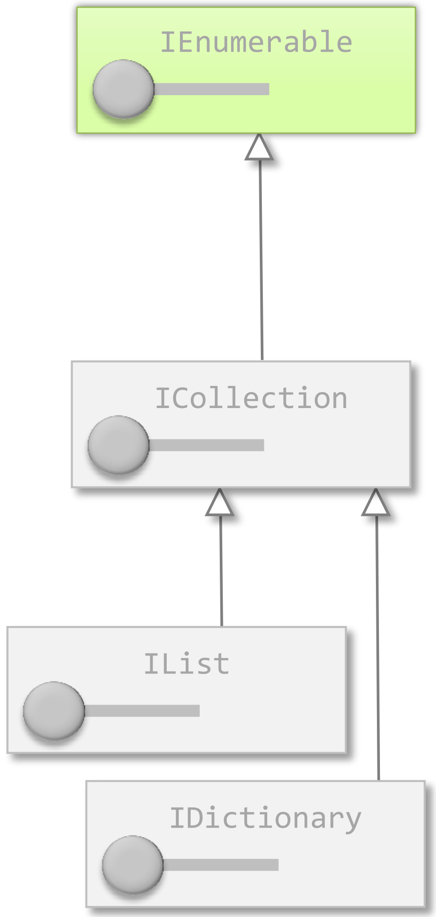

IEnumerable
IEnumerable<T>
ICollection<T>
IList<T>
IDictionary<TKey,TValue>
ISet<T>

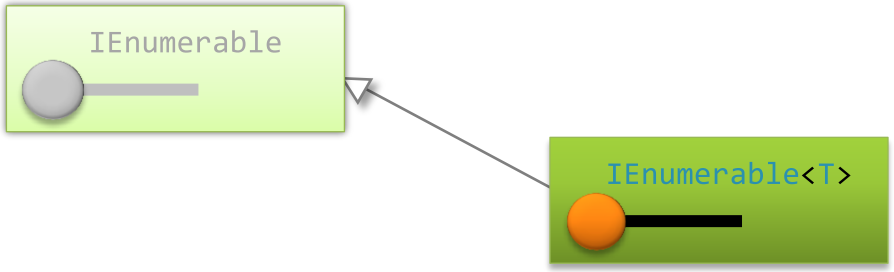

IEnumerator
IDisposable
IEnumerable
IEnumerable<T>
IEnumerator<T>
ICollection
ICollection<T>
IReadOnlyCollection<T>
IList
IList<T>
IReadOnlyList<T>
IDictionary
IDictionary<TKey,TValue>
IReadOnlyDictionary <TKey,TValue>
ISet<T>


= Cannot modify collection
IEnumerator
IDisposable
IEnumerable
IEnumerable<T>
IEnumerator<T>
ICollection
ICollection<T>
IReadOnlyCollection<T>
IList
IList<T>
IReadOnlyList<T>
IDictionary
IDictionary<TKey,TValue>
IReadOnlyDictionary <TKey,TValue>
ISet<T>


IEnumerator
IDisposable
IEnumerable
IEnumerator<T>
IEnumerable<T>
ICollection
ICollection<T>
IReadOnlyCollection<T>
IList
IList<T>
IReadOnlyList<T>
IDictionary
IDictionary<TKey,TValue>
IReadOnlyDictionary
<TKey,TValue>
ISet<T>

.NET 1.x interfaces

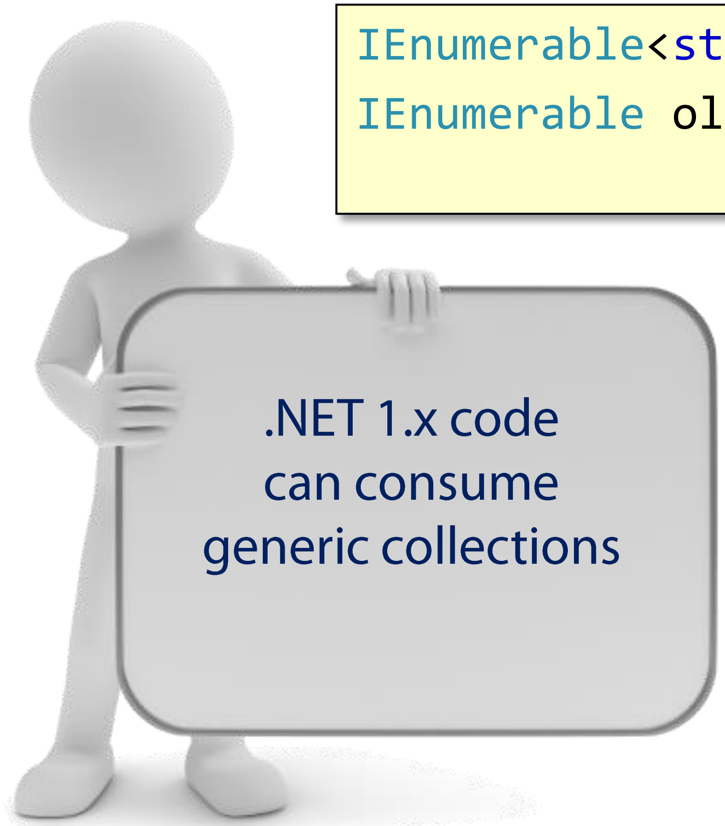


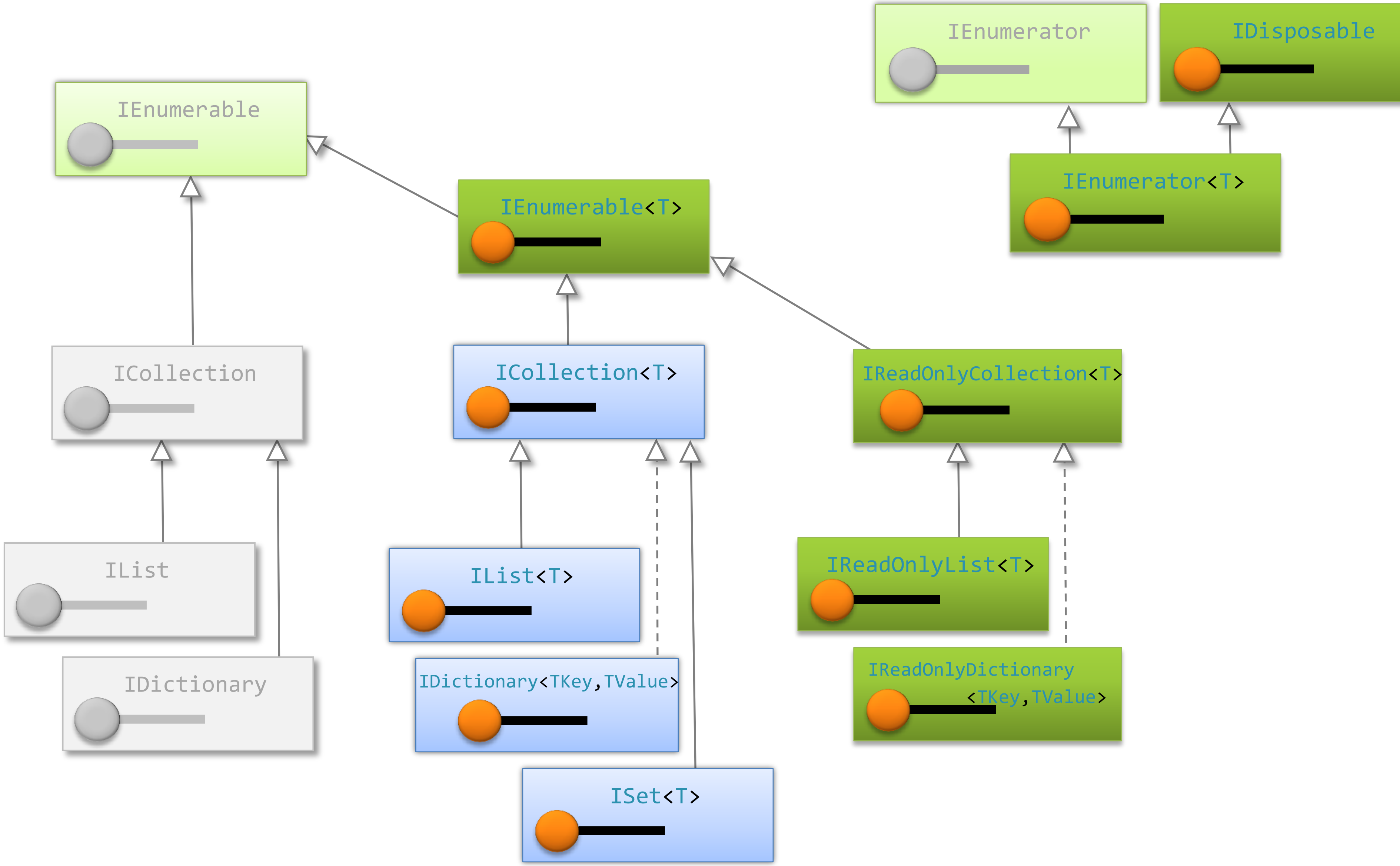


IEnumerable<T>
derives from
IEnumerable

Hence this is valid…

```
IEnumerable<string> genericEnum = // initialize
IEnumerable old = genericEnum
```

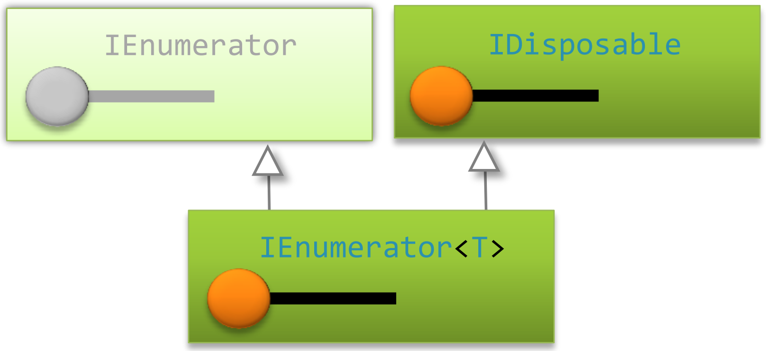



IEnumerator
IDisposable
IEnumerable
IEnumerator<T>
IEnumerable<T>
ICollection
ICollection<T>
IReadOnlyCollection<T>
IList
IList<T>
IReadOnlyList<T>
IDictionary
IDictionary<TKey,TValue>
IReadOnlyDictionary
<TKey,TValue>
ISet<T>

IEnumerable<T>

```
IEnumerator<T> GetEnumerator()
```

Returns an enumerator
– the thing that does the enumerating



`IEnumerable<T>`

`IEnumerator<T> GetEnumerator()`

Returns an enumerator
– the thing that does the enumerating

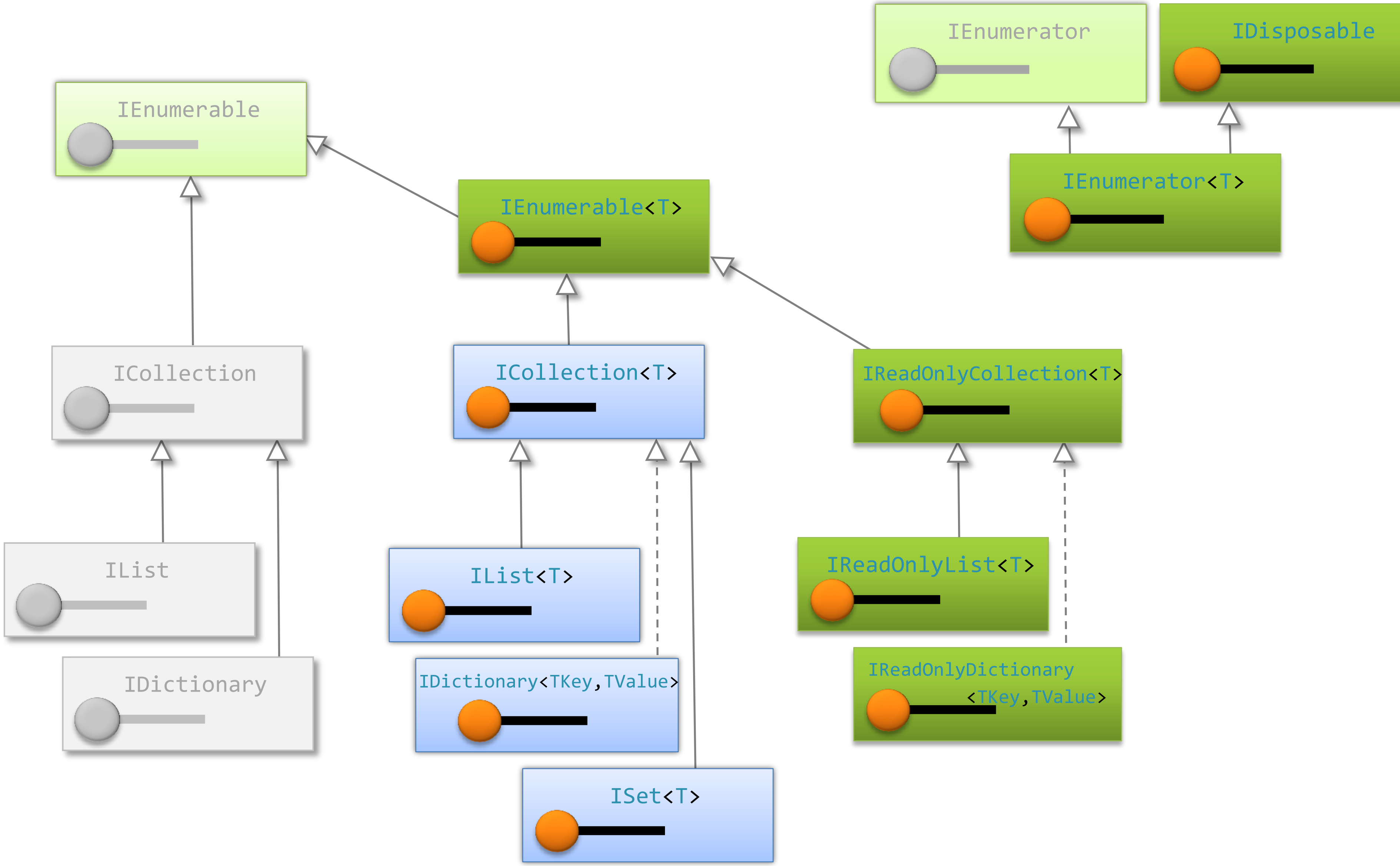
Collection

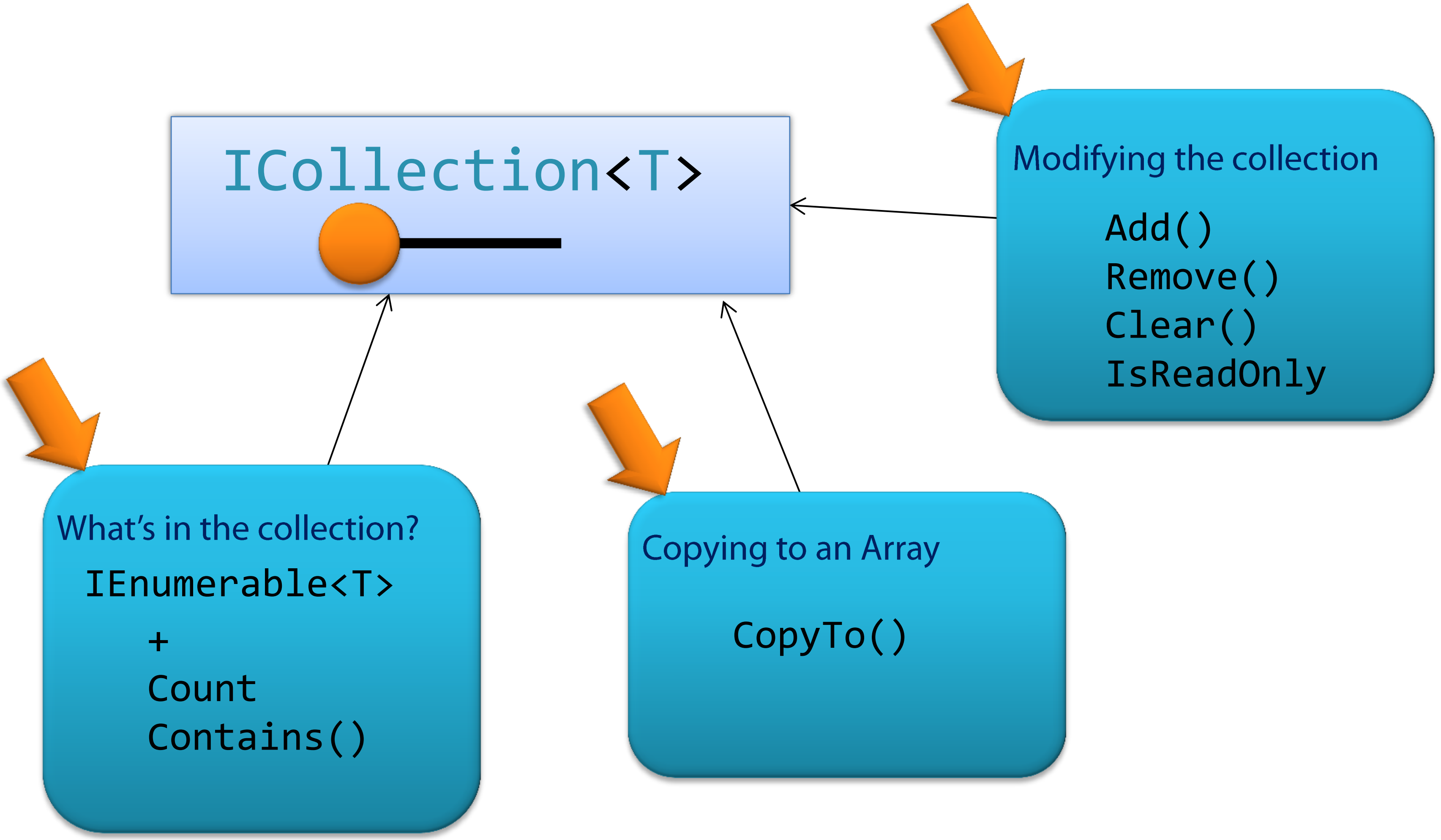
(Enumerable)

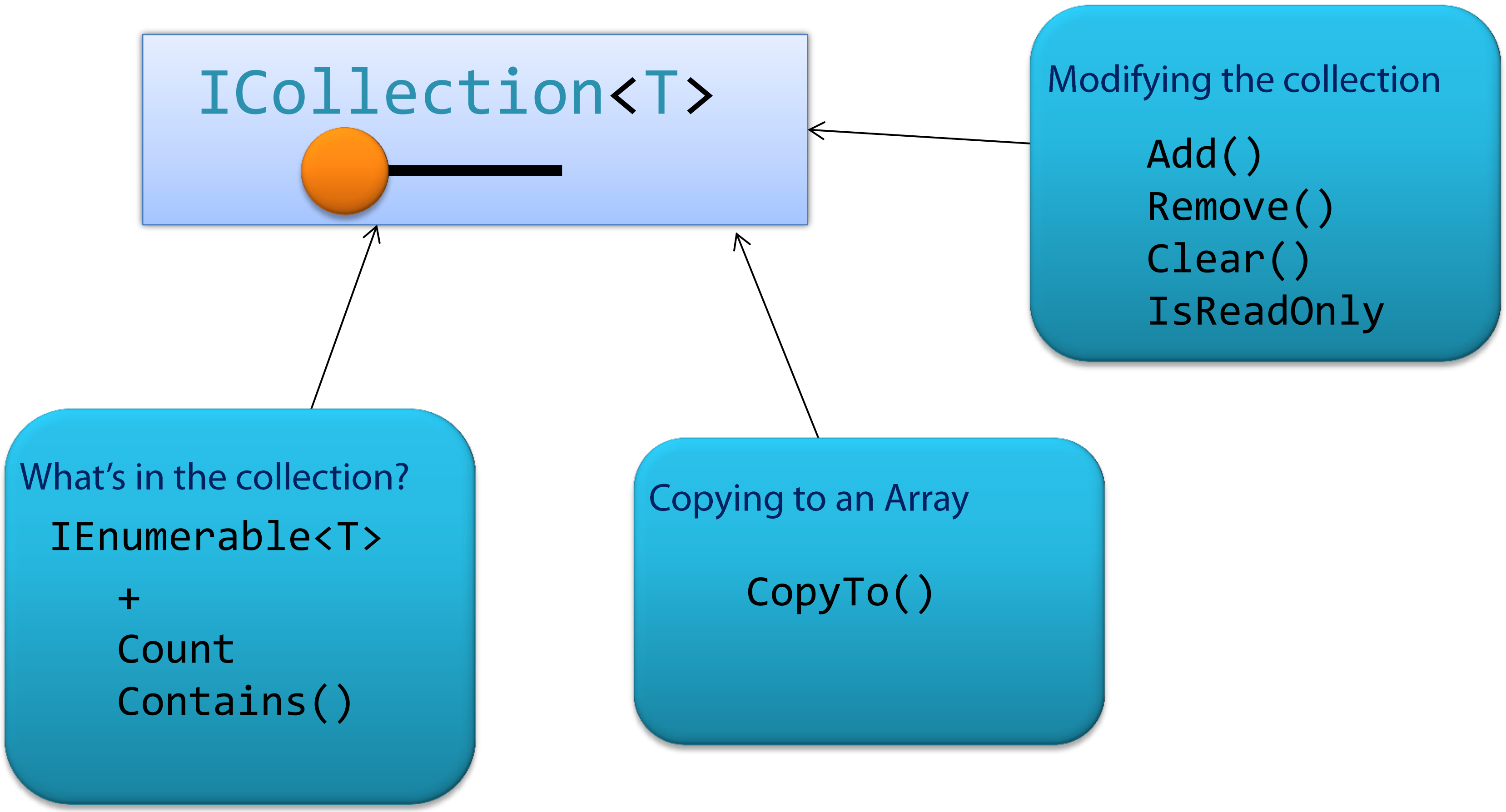
Enumerator

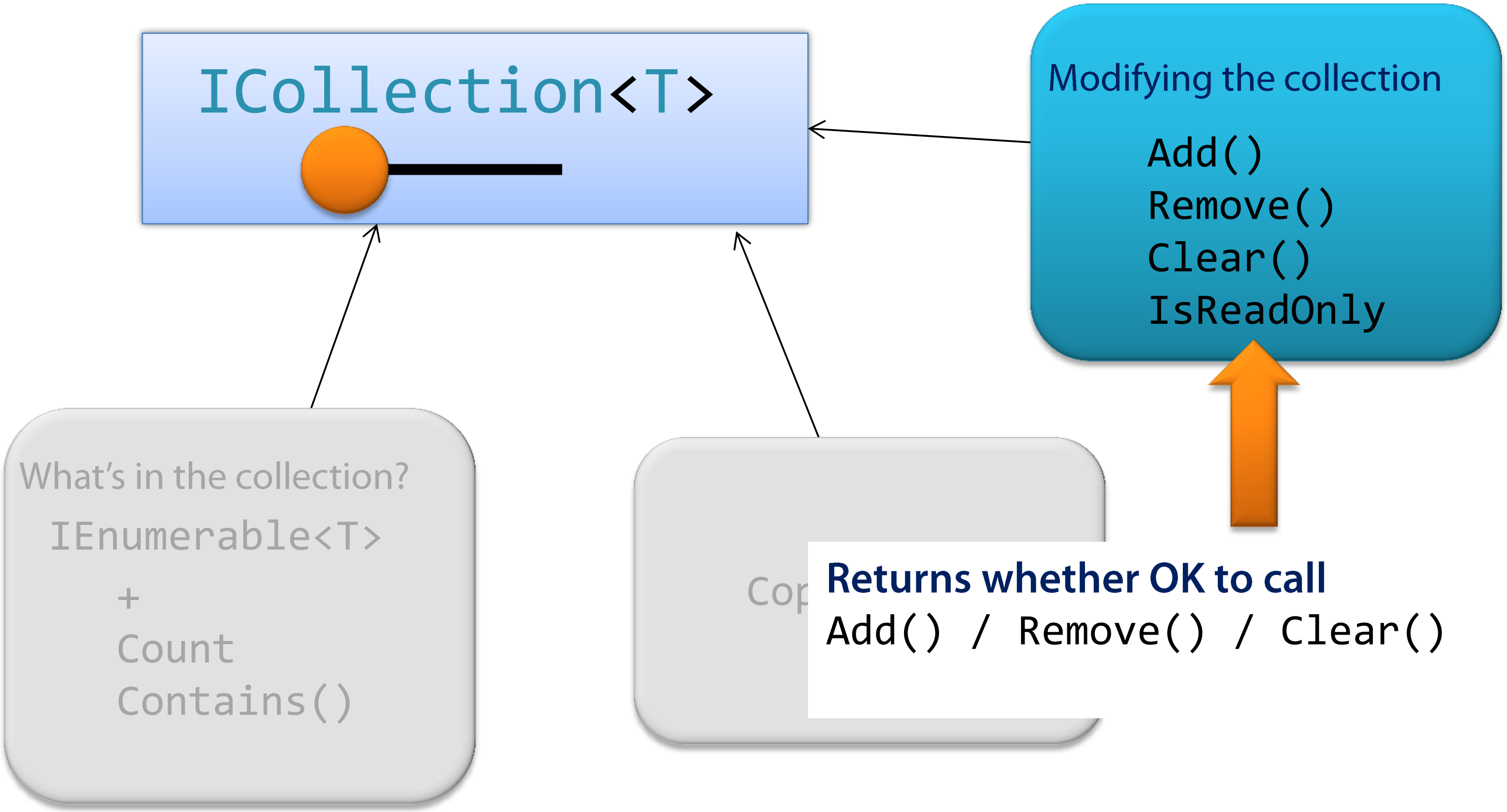
foreach
does all this
for you

Code Demo


IEnumerator
IDisposable
IEnumerable
IEnumerator<T>
IEnumerable<T>
ICollection
ICollection<T>
IReadOnlyCollection<T>
IList
IList<T>
IReadOnlyList<T>
IDictionary
IDictionary<TKey,TValue>
IReadOnlyDictionary
<TKey,TValue>
ISet<T>

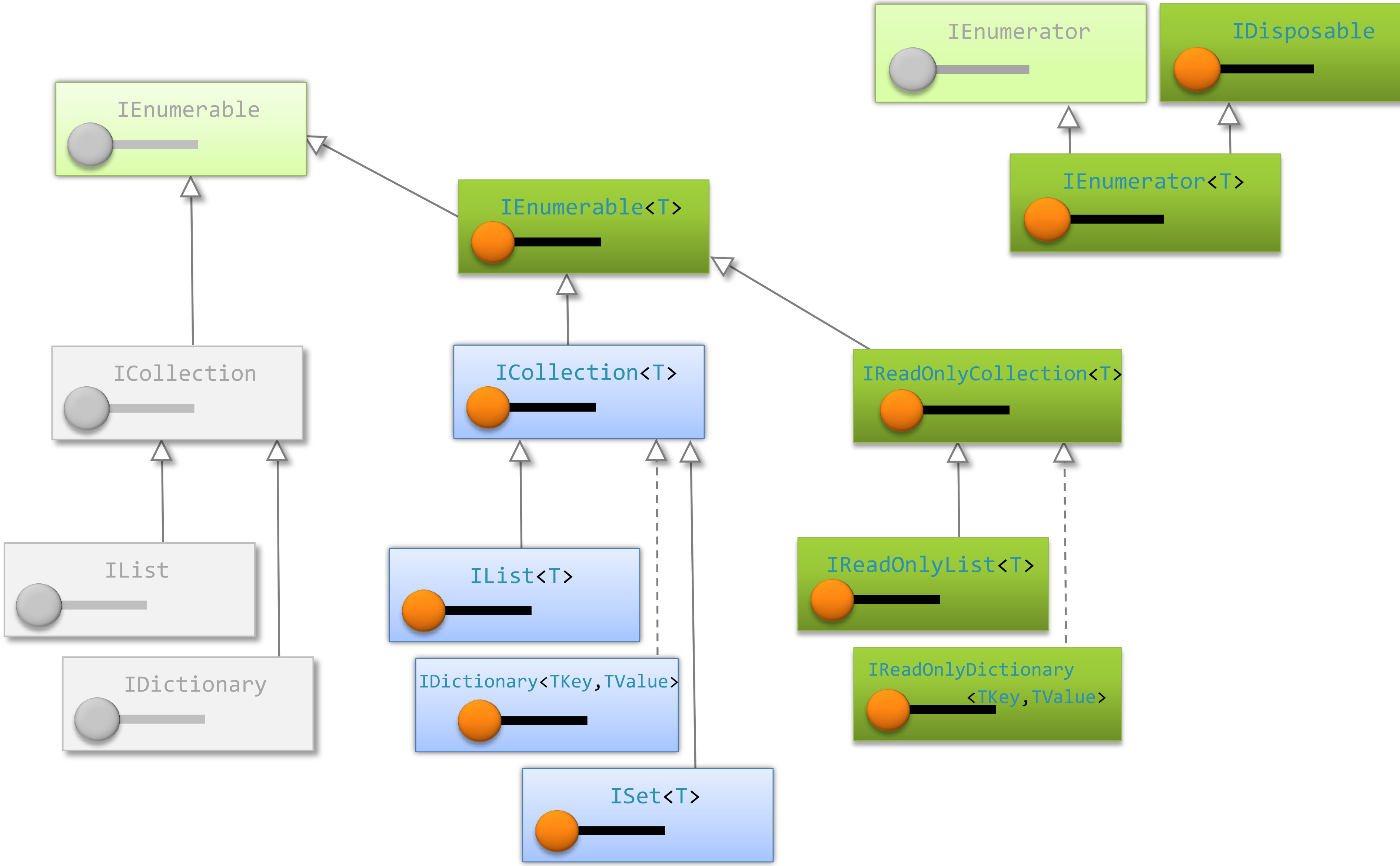

ICollection<T>
Modifying the collection
Add()
Remove()
Clear()
IsReadOnly
What's in the collection?
IEnumerable<T>
+
Count
Contains()
Copying to an Array
CopyTo()

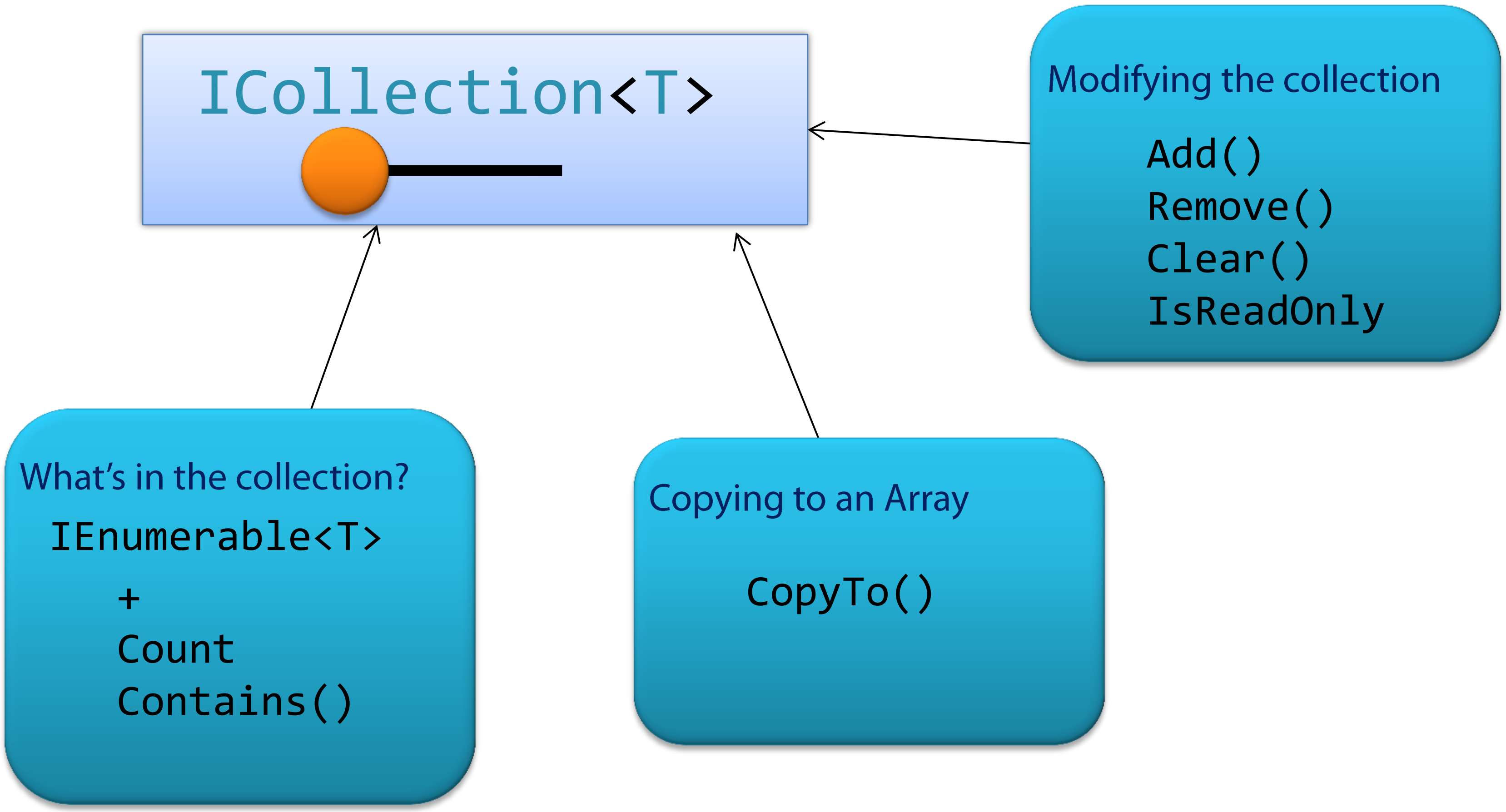

ICollection<T>
Modifying the collection
Add()
Remove()
Clear()
IsReadOnly
What's in the collection?
IEnumerable<T>
+
Count
Contains()
Copying to an Array
CopyTo()

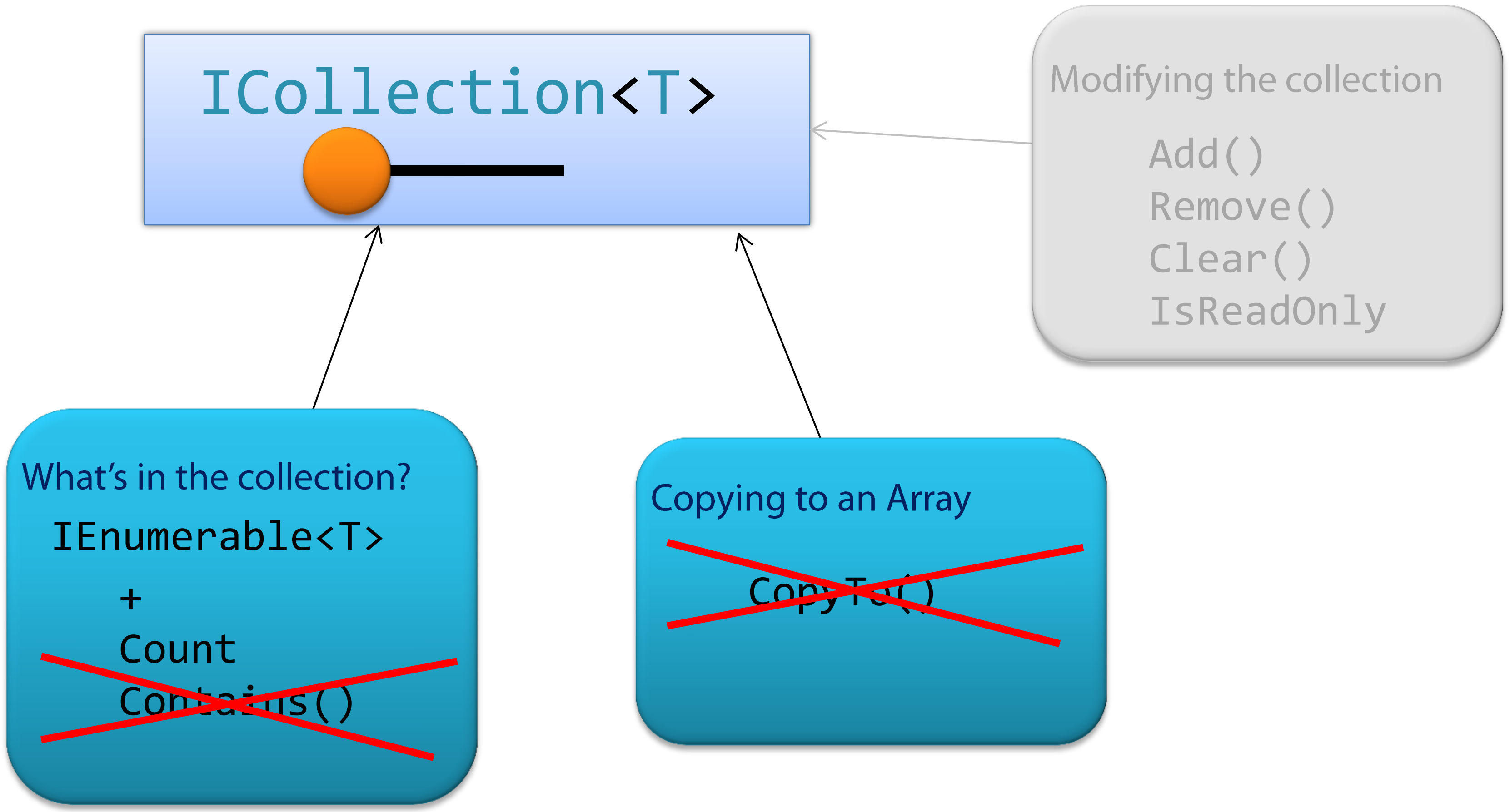


Returns whether OK to call

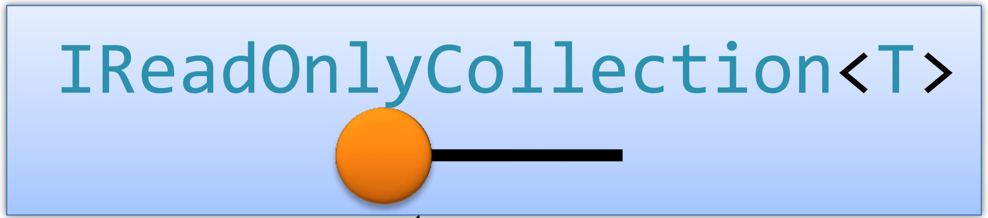
`Add() / Remove() / Clear()`


IEnumerator
IDisposable
IEnumerable
IEnumerable<T>
IEnumerator<T>
ICollection
ICollection<T>
IReadOnlyCollection<T>
IList
IList<T>
IReadOnlyList<T>
IDictionary
IDictionary<TKey,TValue>
IReadOnlyDictionary<TKey,TValue>
ISet<T>


ICollection<T>
Modifying the collection
Add()
Remove()
Clear()
IsReadOnly
What's in the collection?
IEnumerable<T>
+
Count
Contains()
Copying to an Array
CopyTo()

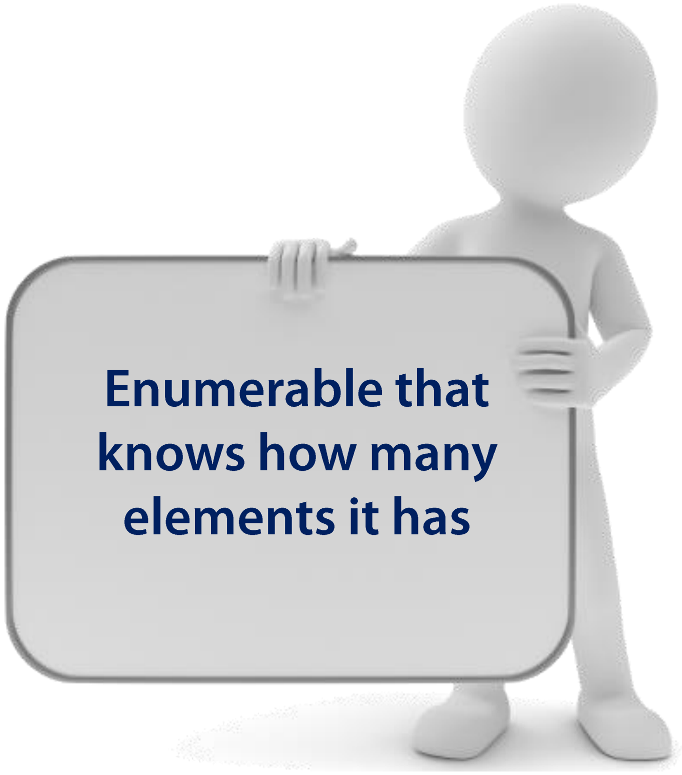

ICollection<T>
Modifying the collection
Add()
Remove()
Clear()
IsReadOnly
What's in the collection?
IEnumerable<T>
+
Count
~~Contains()~~
Copying to an Array
~~CopyTo()~~

`IReadOnlyCollection<T>`



What's in the collection?

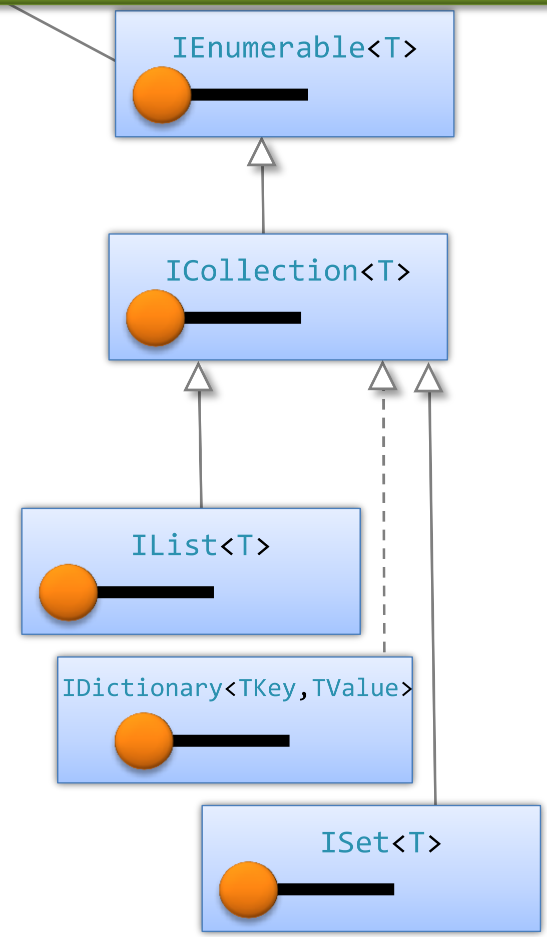
```
IEnumerable<T>
    +
    Count
```



Enumerable that knows how many elements it has

IList<T>:
“You can look up my elements with an index”

```
string tuesday = daysOfWeek[1];
```



IList<T>
Implemented by:

T[]

List<T>

etc.

= ICollection<T> + …

Modifying the collection

```
Insert()

RemoveAt()
```

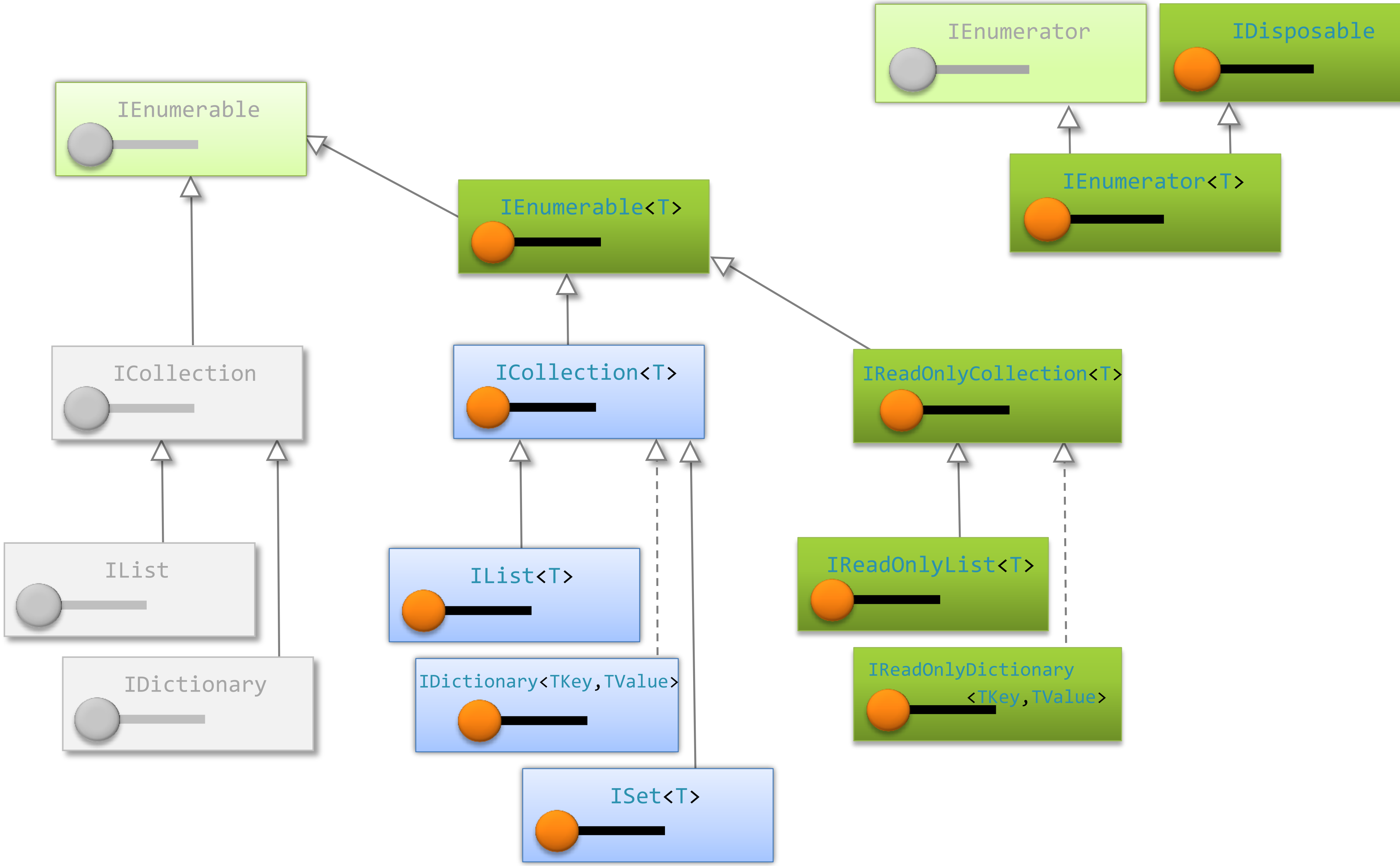
What's in the collection?

```
this[index]

IndexOf()
```

```
// look up element
string saturday = daysOfWeek[5];
```

```
// replace element
daysOfWeek[5] = "SlaveDay";
```


IEnumerator
IDisposable
IEnumerable
IEnumerator<T>
IEnumerable<T>
ICollection
ICollection<T>
IReadOnlyCollection<T>
IList
IList<T>
IReadOnlyList<T>
IDictionary
IDictionary<TKey,TValue>
IReadOnlyDictionary <TKey,TValue>
ISet<T>

IReadOnlyList<T>

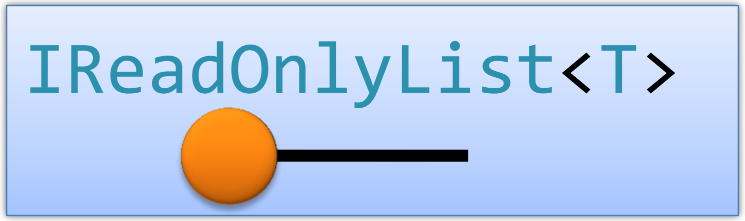
= IReadOnlyCollection<T> + …

What's in the collection?

`this[index]`

(Read only)

```
// look up element
string saturday = daysOfWeek[5];
```



=

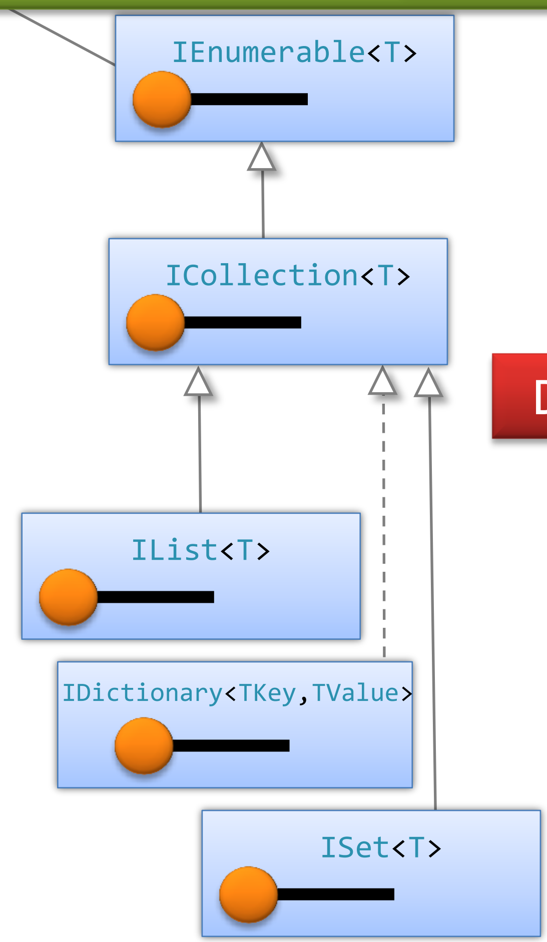
`IEnumerable<T>`

+ Knows how many elements there are

+ can look up element by index

`IDictionary<TKey, TValue>`:
“You can look up my elements with a key”

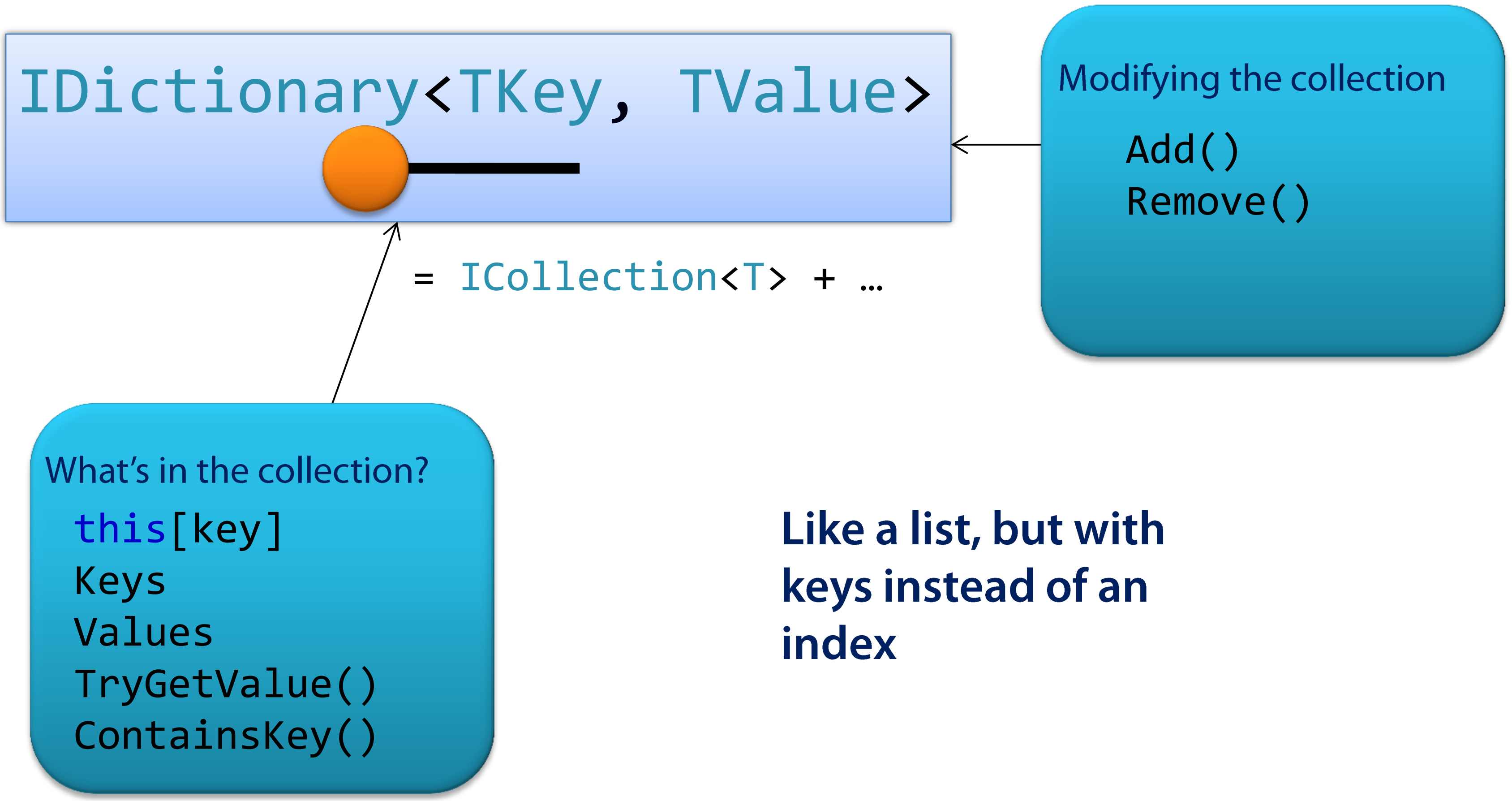
```
Country country = countries["UK"];
```



`IDictionary<TKey, TValue>`
Implemented by:

`Dictionary<TKey, TValue>`

etc.



```
IDictionary<TKey, TValue>
```

```
= ICollection<T> + …
```

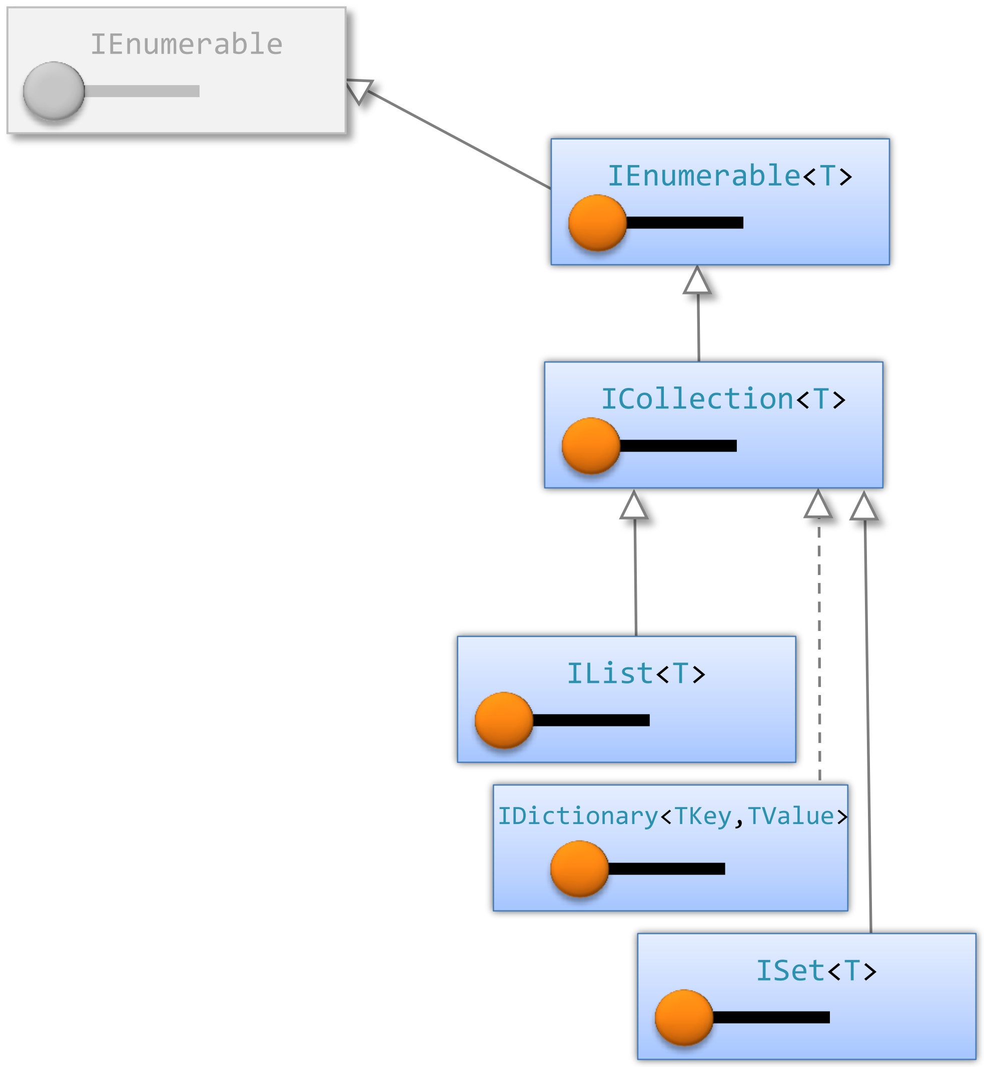
Modifying the collection

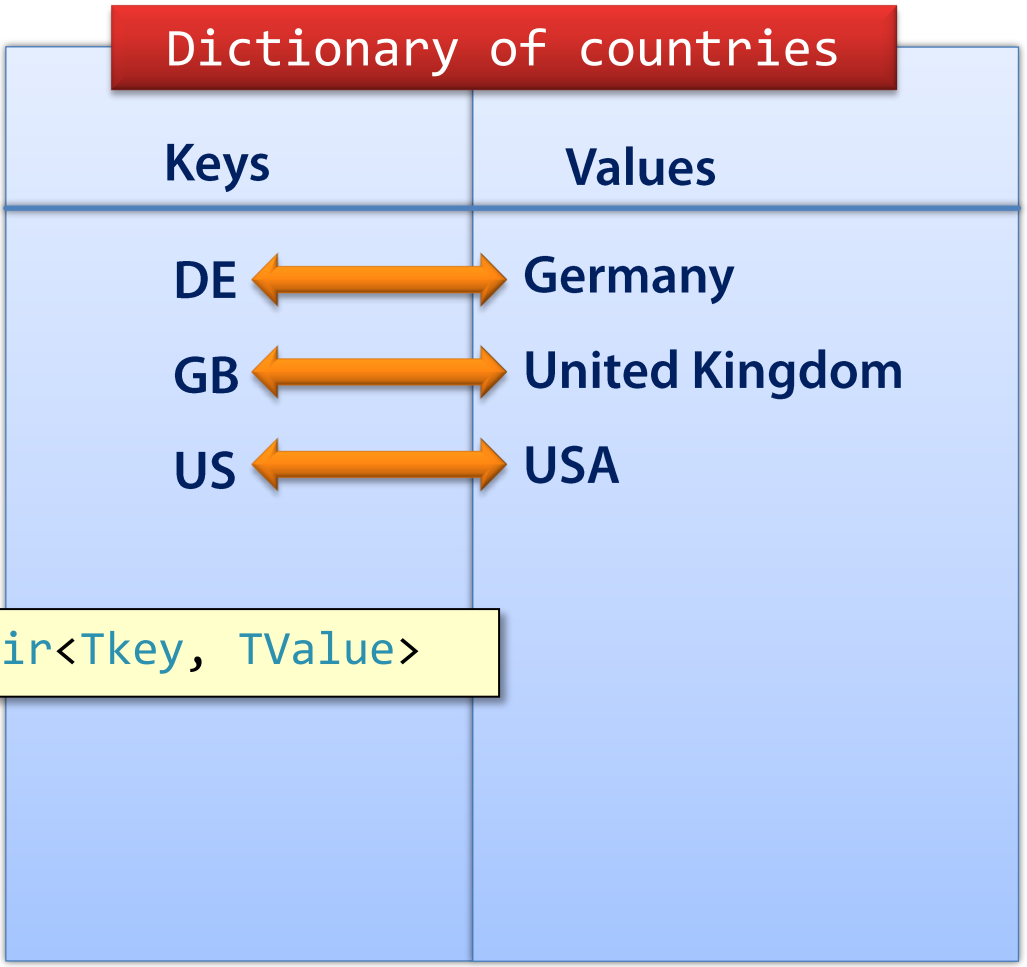
```
Add()
Remove()
```

What’s in the collection?

```
this[key]
Keys
Values
TryGetValue()
ContainsKey()
```

Like a list, but with keys instead of an index


IEnumerable
IEnumerable<T>
ICollection<T>
IList<T>
IDictionary<TKey,TValue>
ISet<T>



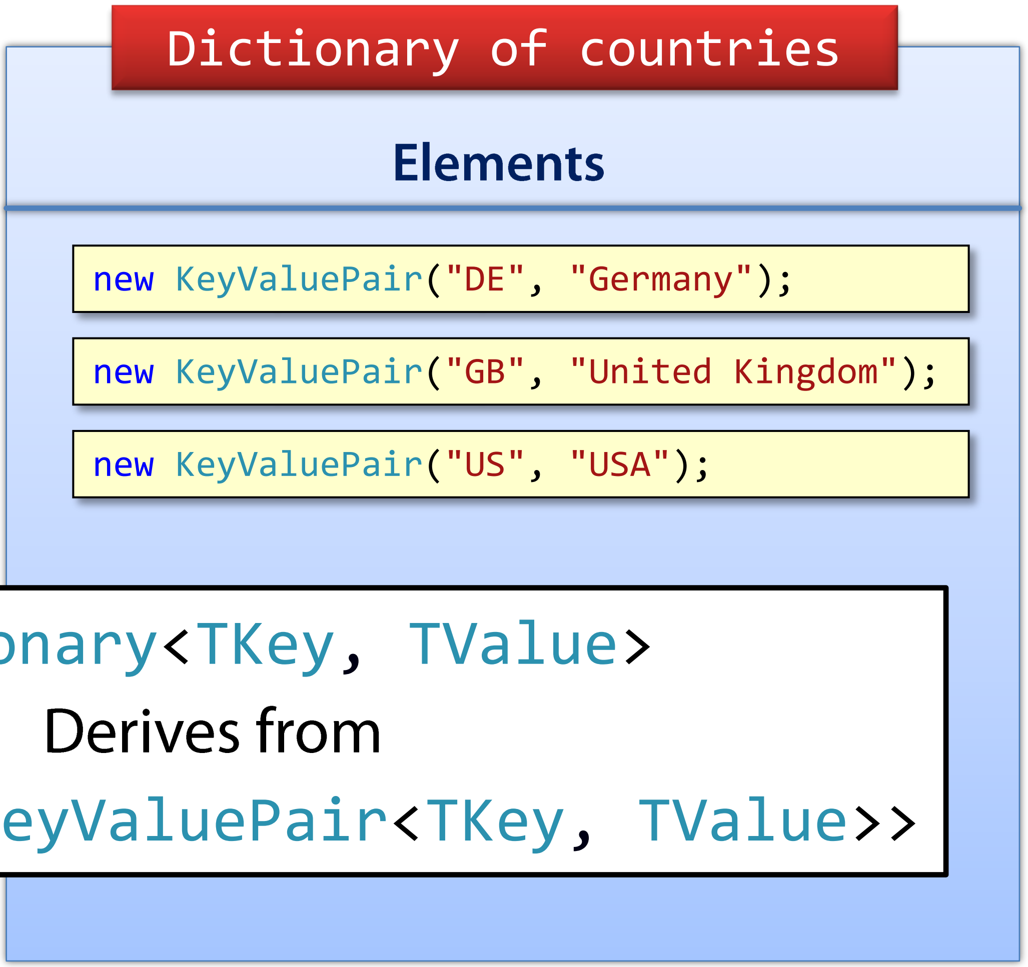
```
class KeyValuePair<Tkey, TValue>
```

Dictionary of countries

Elements

```
new KeyValuePair("DE", "Germany");
new KeyValuePair("GB", "United Kingdom");
new KeyValuePair("US", "USA");
```

```
class KeyValuePair<Tkey, TValue>
```

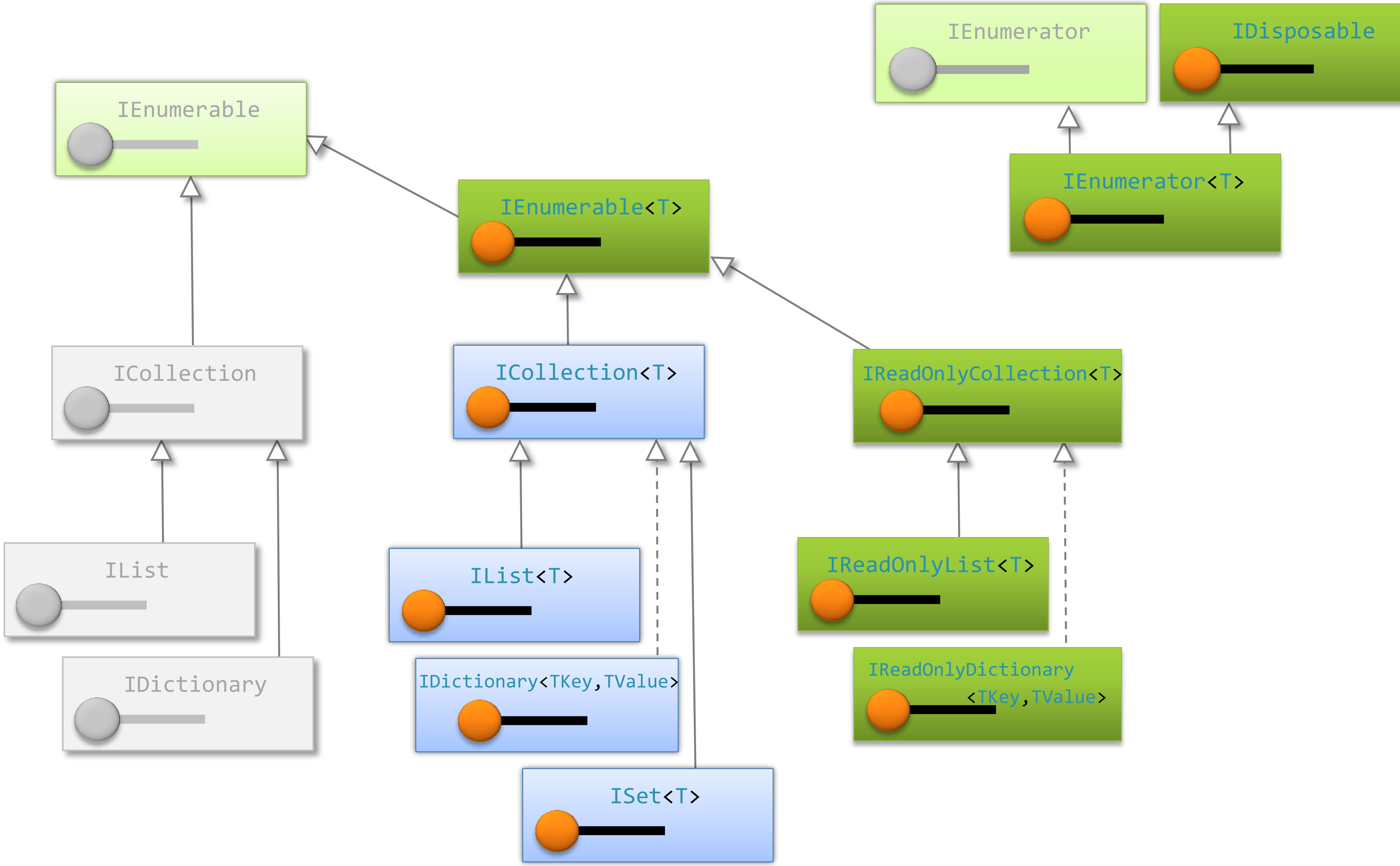


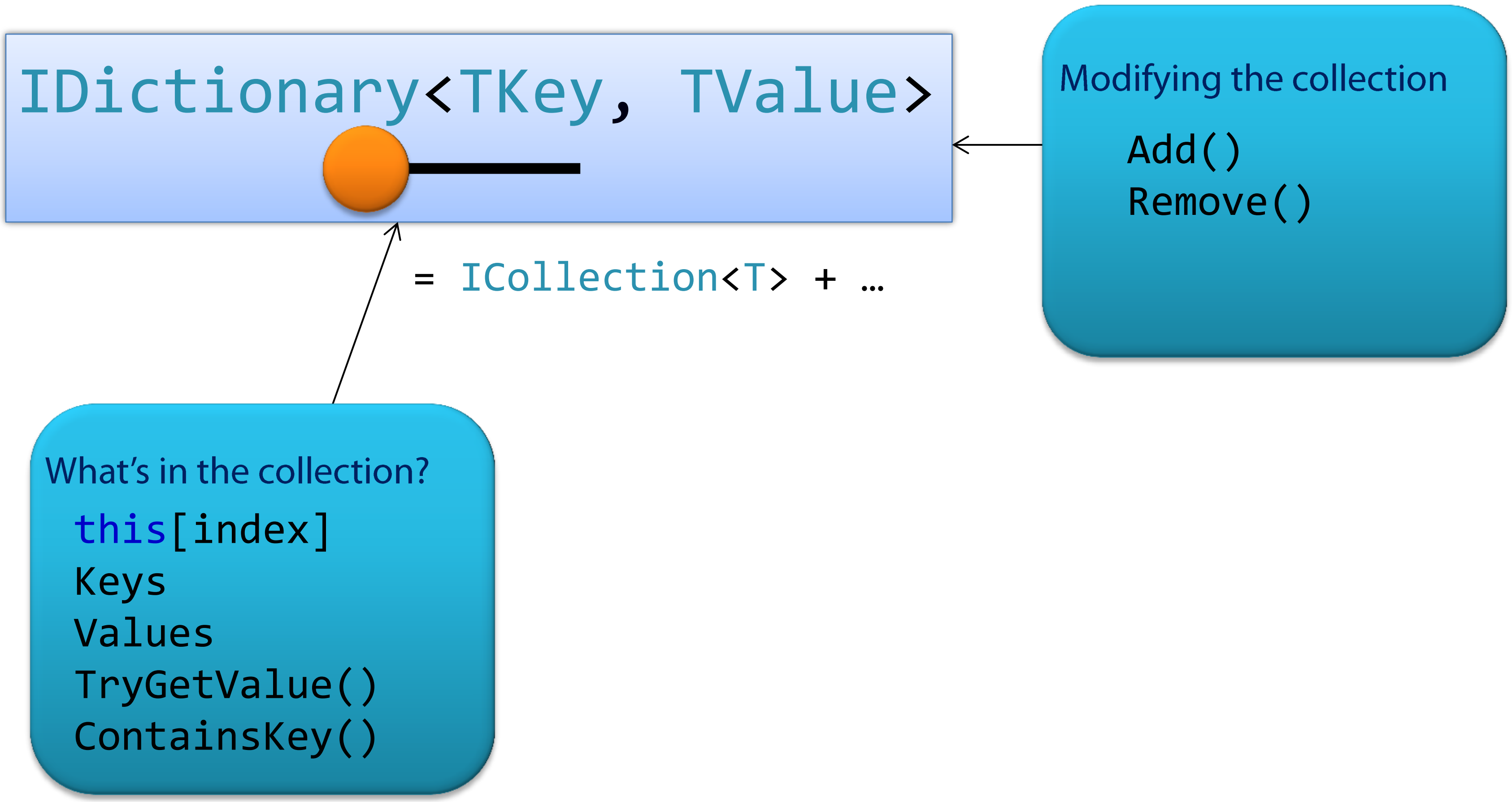
```
new KeyValuePair("DE", "Germany");
new KeyValuePair("GB", "United Kingdom");
new KeyValuePair("US", "USA");
```

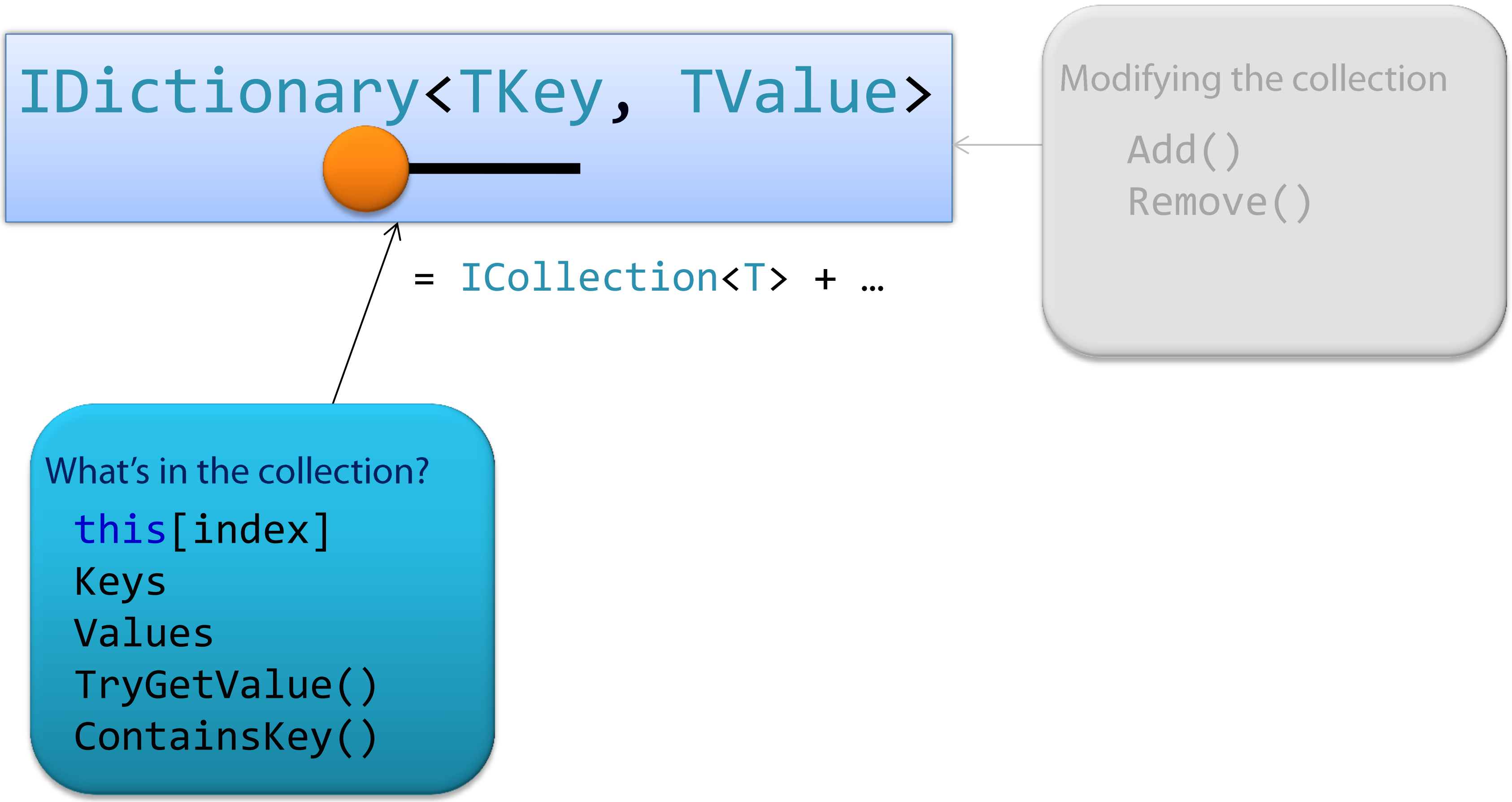
`IDictionary<TKey, TValue>`

Derives from

`ICollection<KeyValuePair<TKey, TValue>>`


IEnumerator
IDisposable
IEnumerable
IEnumerable<T>
IEnumerator<T>
ICollection
ICollection<T>
IReadOnlyCollection<T>
IList
IList<T>
IReadOnlyList<T>
IDictionary
IDictionary<TKey,TValue>
IReadOnlyDictionary<TKey,TValue>
ISet<T>


IDictionary<TKey, TValue>
= ICollection<T> + …
Modifying the collection
Add()
Remove()
What’s in the collection?
this[index]
Keys
Values
TryGetValue()
ContainsKey()



```
IDictionary<TKey, TValue>
```

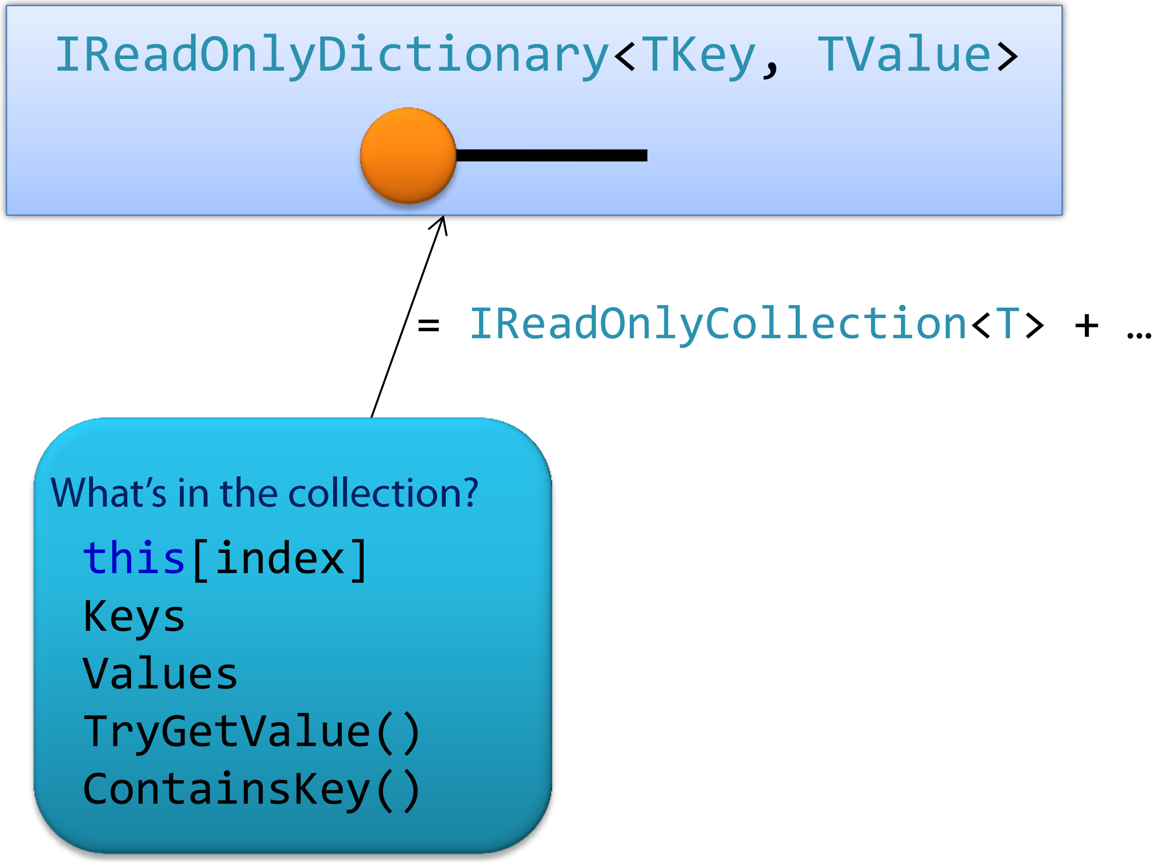
```
= ICollection<T> + …
```

Modifying the collection

```
Add()
Remove()
```

What's in the collection?

```
this[index]
Keys
Values
TryGetValue()
ContainsKey()
```



```
IReadOnlyDictionary<TKey, TValue>
```

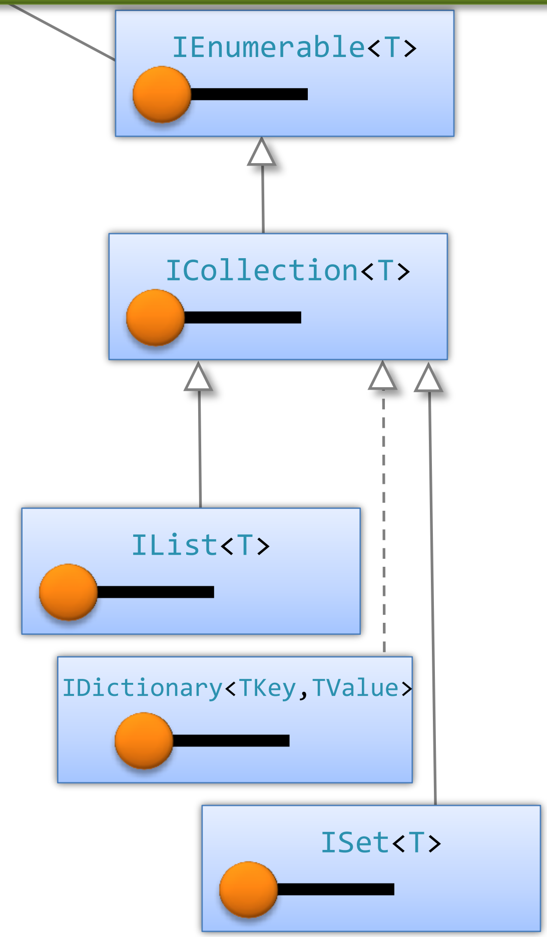
```
= IReadOnlyCollection<T> + …
```

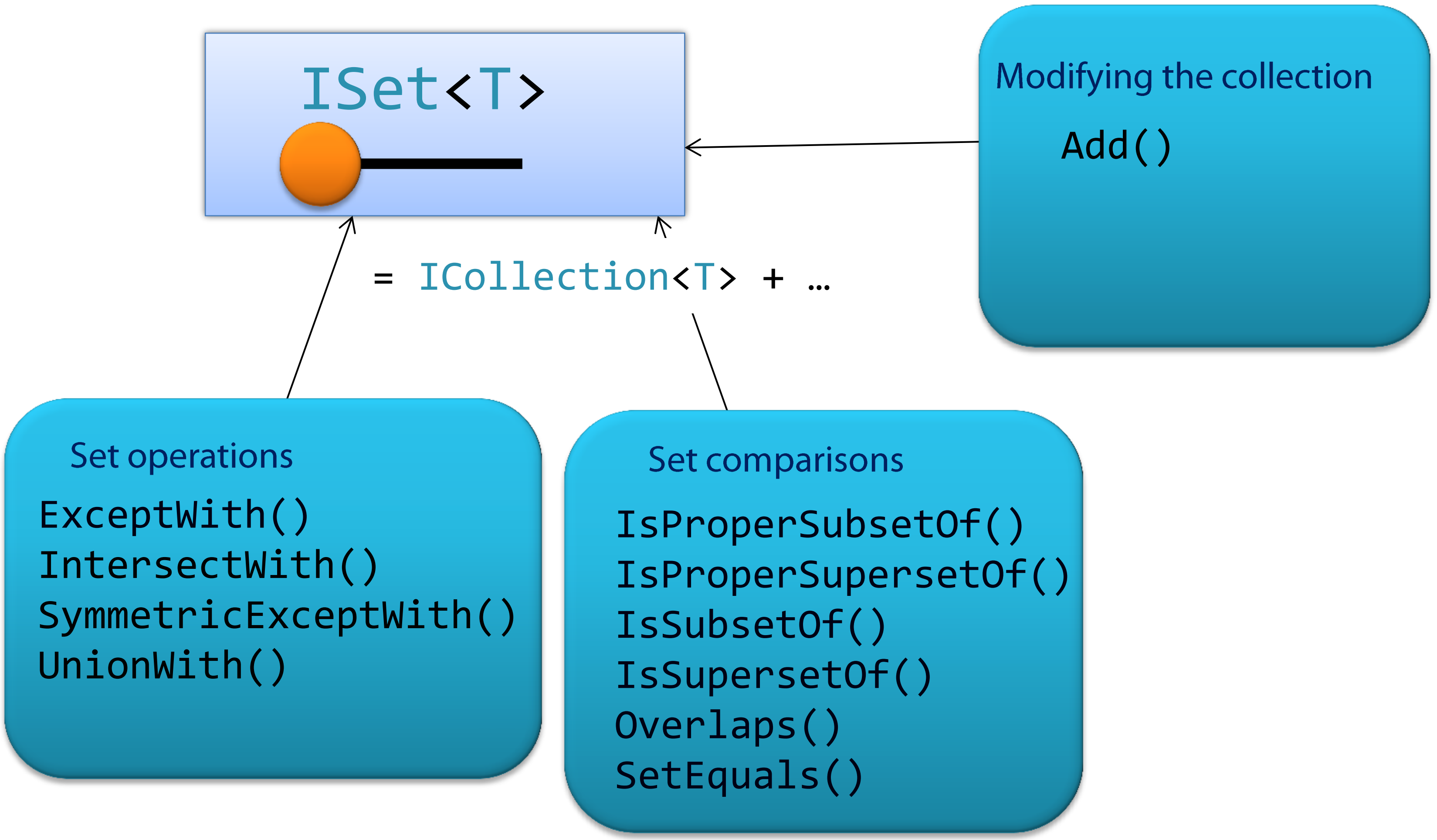
What’s in the collection?

```
this[index]
Keys
Values
TryGetValue()
ContainsKey()
```

ISet<T>:

"I can do set operations with other collections"




ISet<T>
= ICollection<T> + …
Modifying the collection
Add()
Set operations
ExceptWith()
IntersectWith()
SymmetricExceptWith()
UnionWith()
Set comparisons
IsProperSubsetOf()
IsProperSupersetOf()
IsSubsetOf()
IsSupersetOf()
Overlaps()
SetEquals()

Summary - Interfaces

- Generic interfaces are based on `IEnumerable<T>`
  - This can supply an enumerator
- Concept of collection defined by `ICollection<T>`
  - (But this interface isn't always so useful)
- Three categories of collection defined by:
  - `IList<T>`
  - `IDictionary<TKey, TValue>`
  - `ISet<T>`
- Corresponding ReadOnly interfaces in .NET 4.5