

Collection Interfaces

Simon Robinson
<http://TechieSimon.com>
@TechieSimon



pluralsight 
hardcore developer training

Benefits of learning the interfaces...

Create loosely coupled code



Understand MS collections better



Module Overview



Interface hierarchy

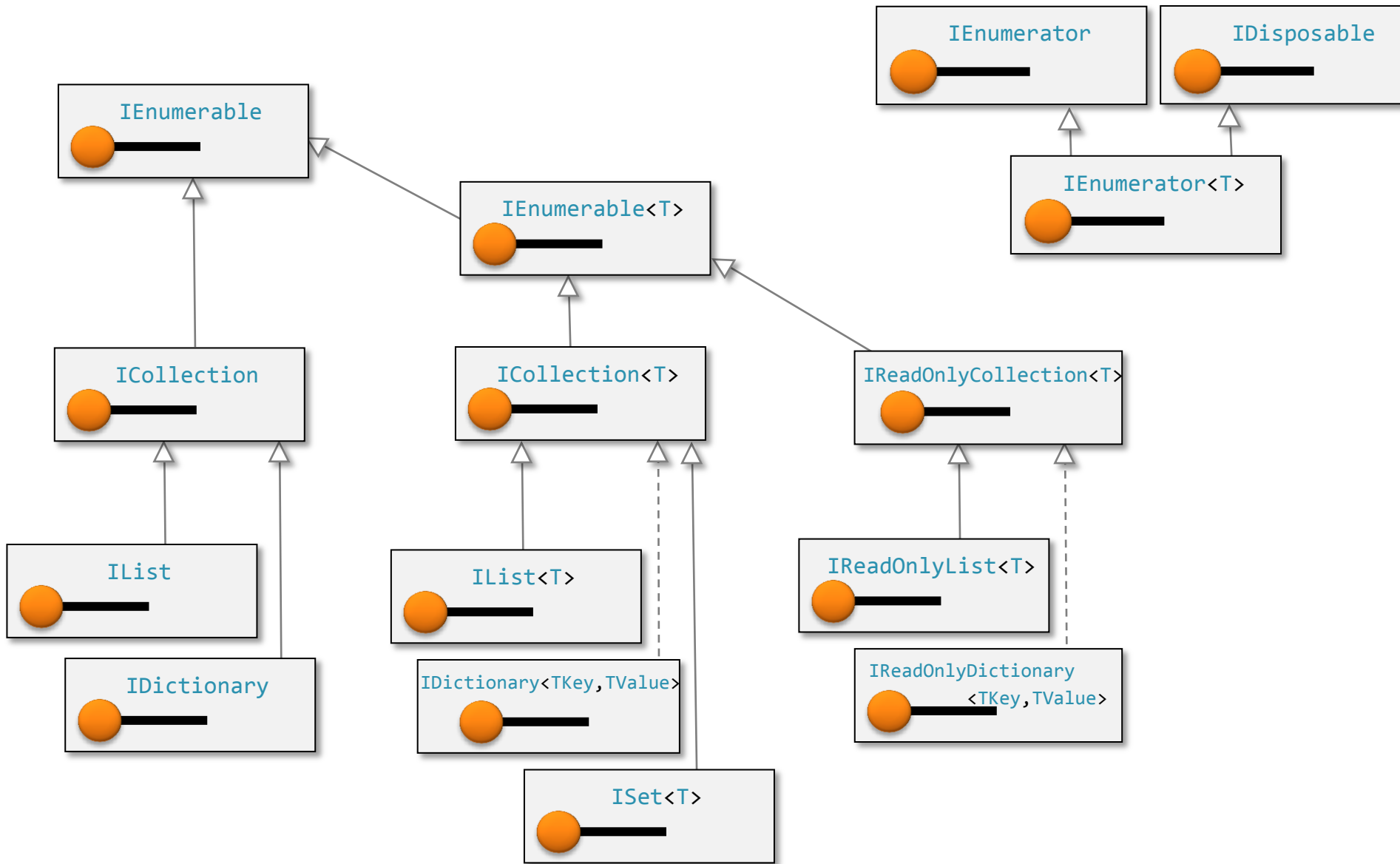
- Core generic interfaces
- Read-only interfaces
- Older non-generic interfaces

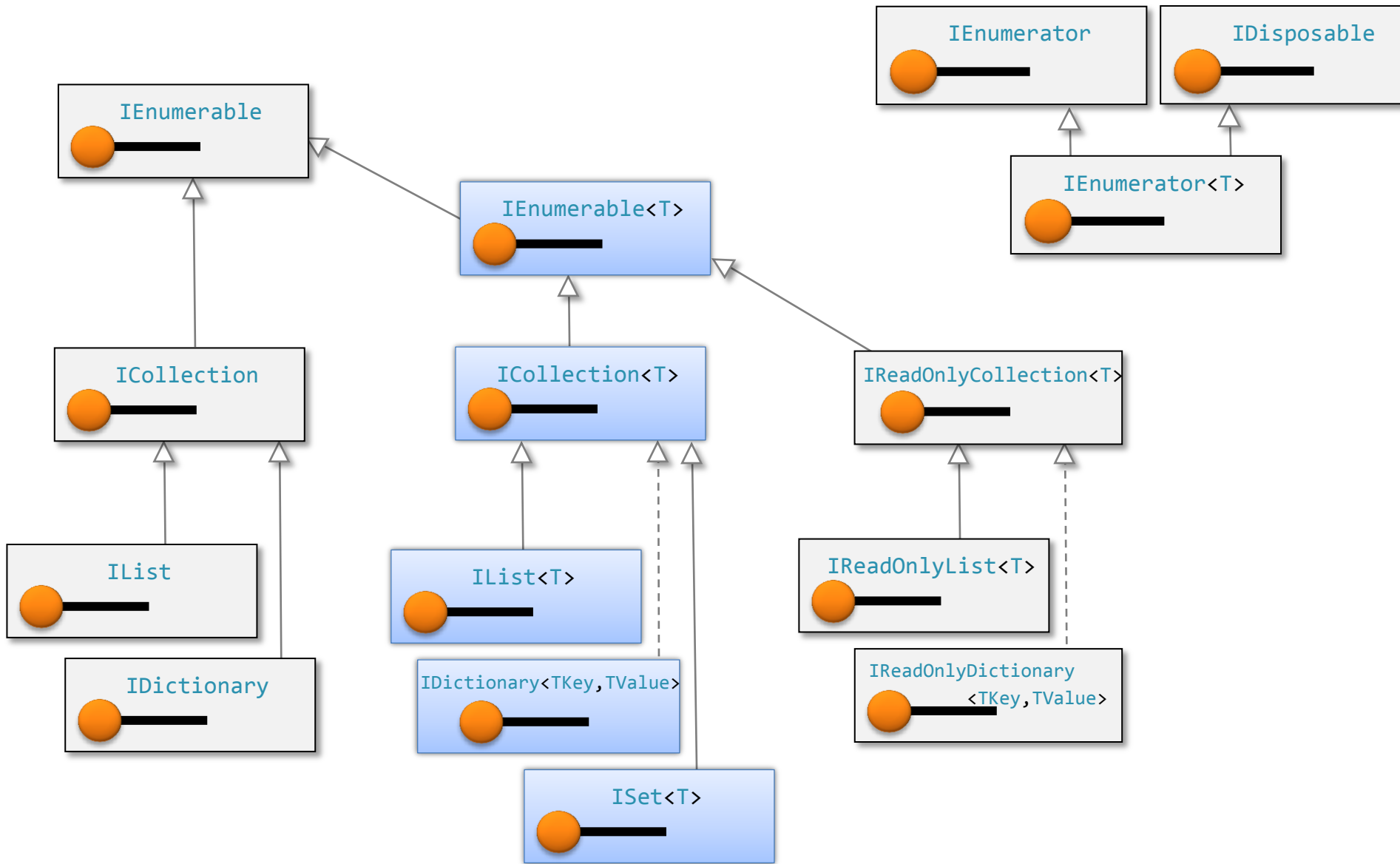
■ Specific interfaces

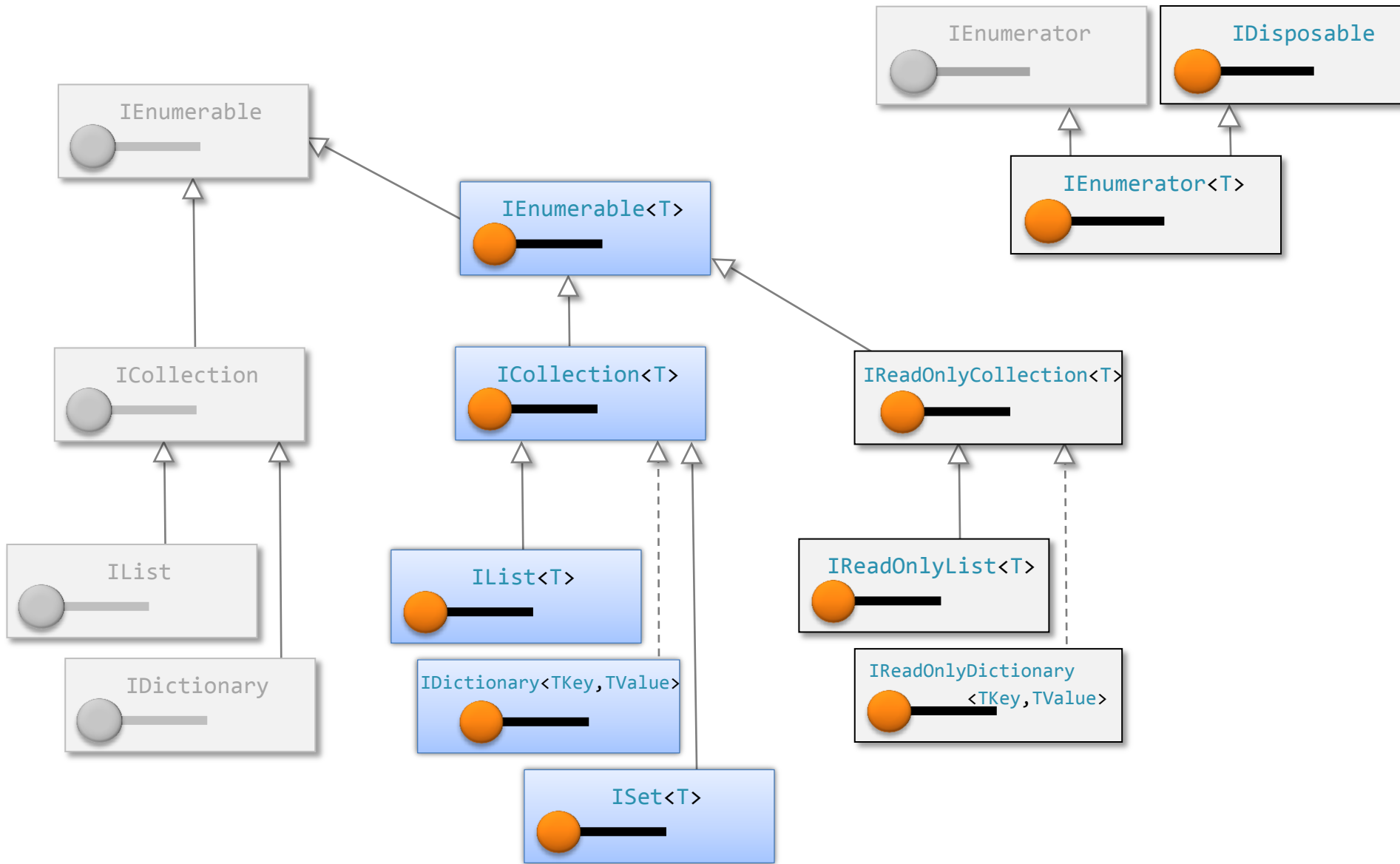
- `IEnumerable<T>`
- `ICollection<T>`
- `IList<T>`
- `IDictionary<TKey, TValue>`
- `ISet<T>`

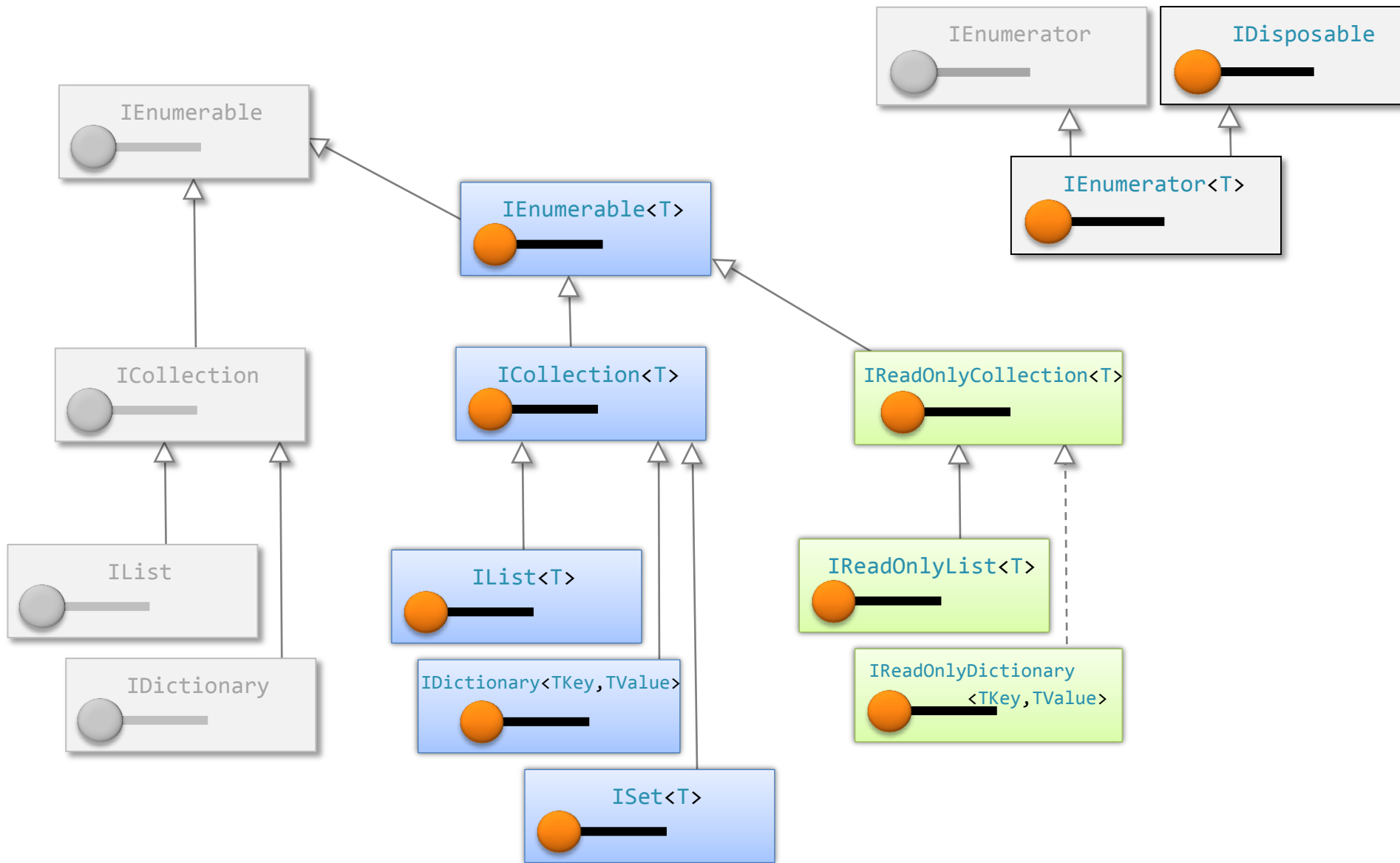
■ Explicit implementation



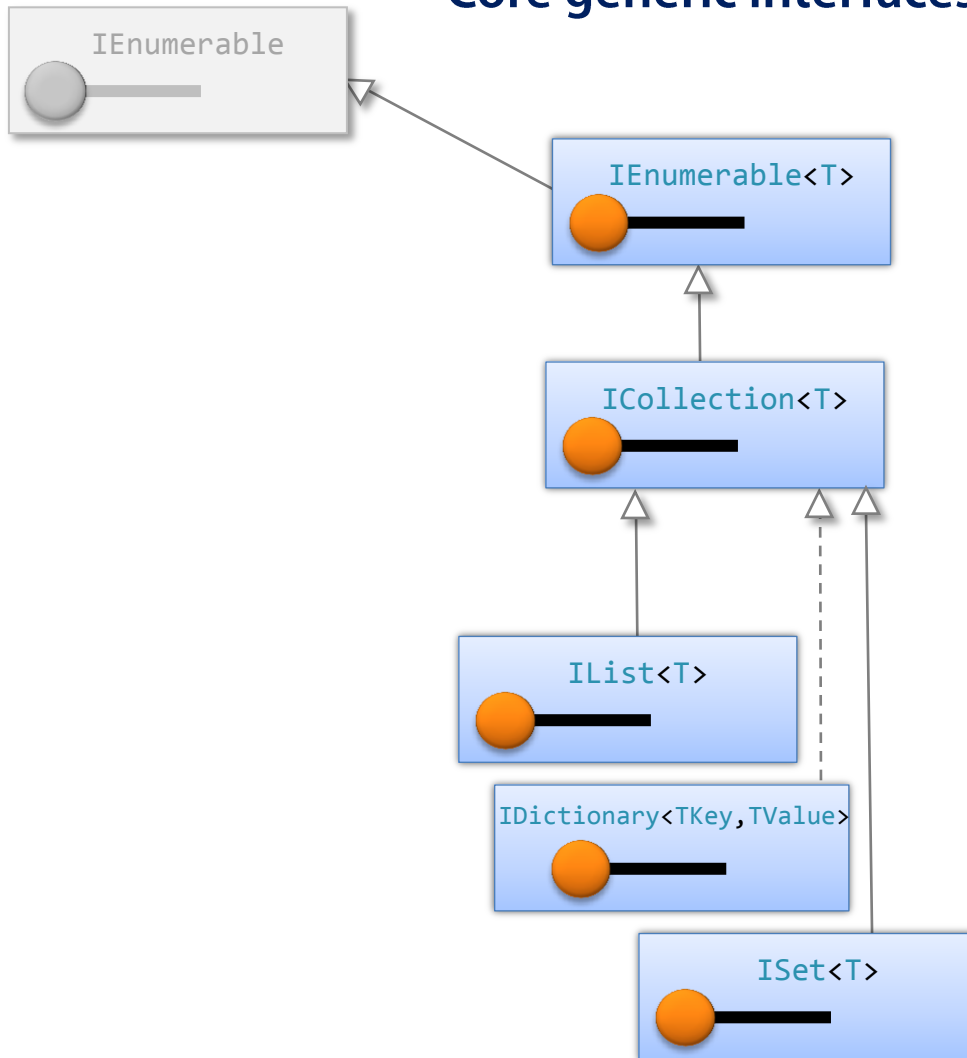


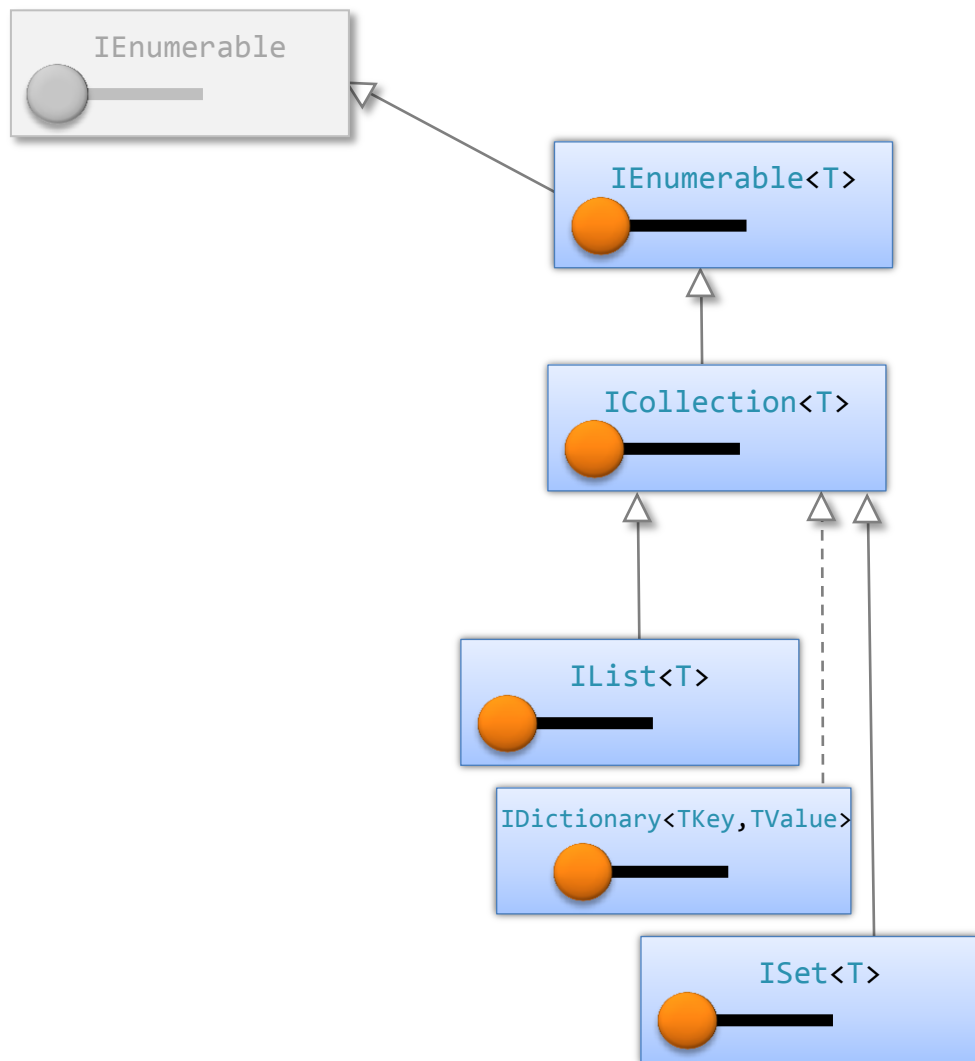






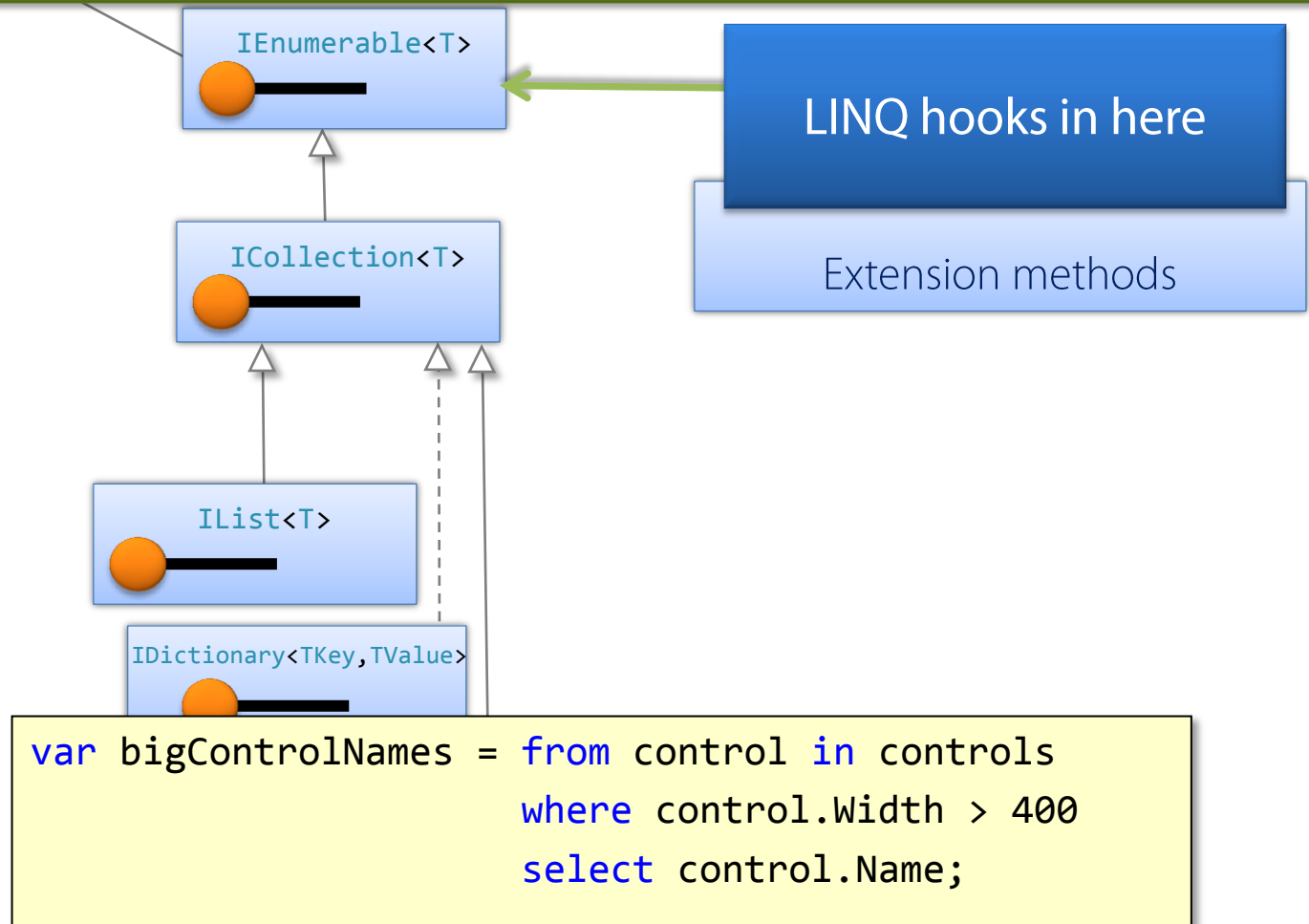
Core generic interfaces





IEnumerable<T>:

"You can iterate my elements"



ICollection vs IEnumerable



Too many to fit in
memory at once?

Data records



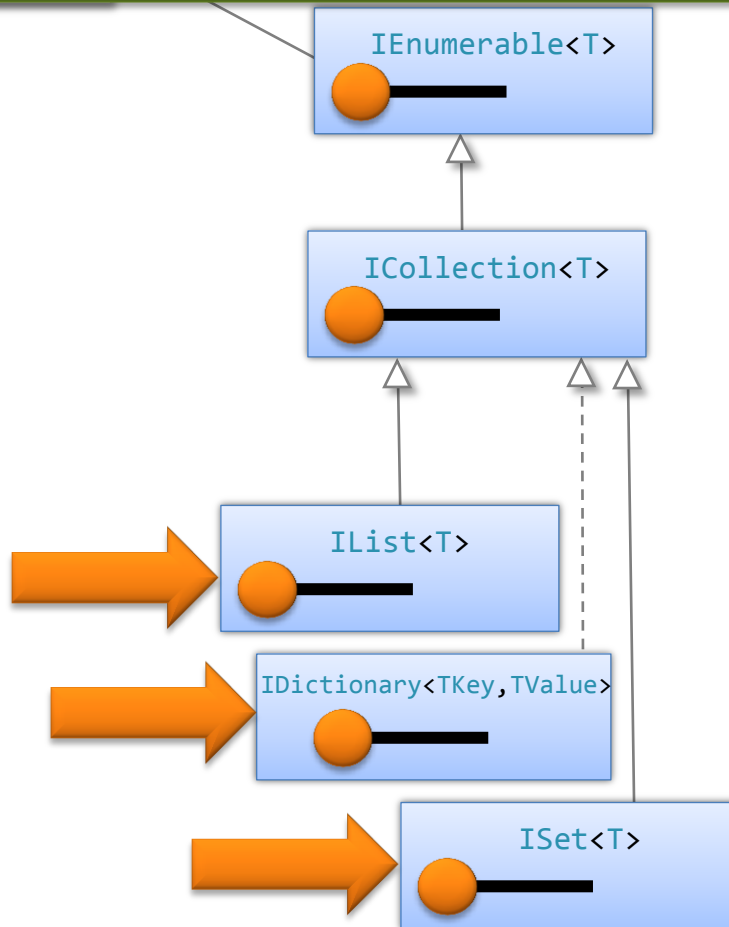
**This is
enumerable
but
not a collection!**



ICollection<T>:

"I know how many elements I have"

"You can modify my contents"

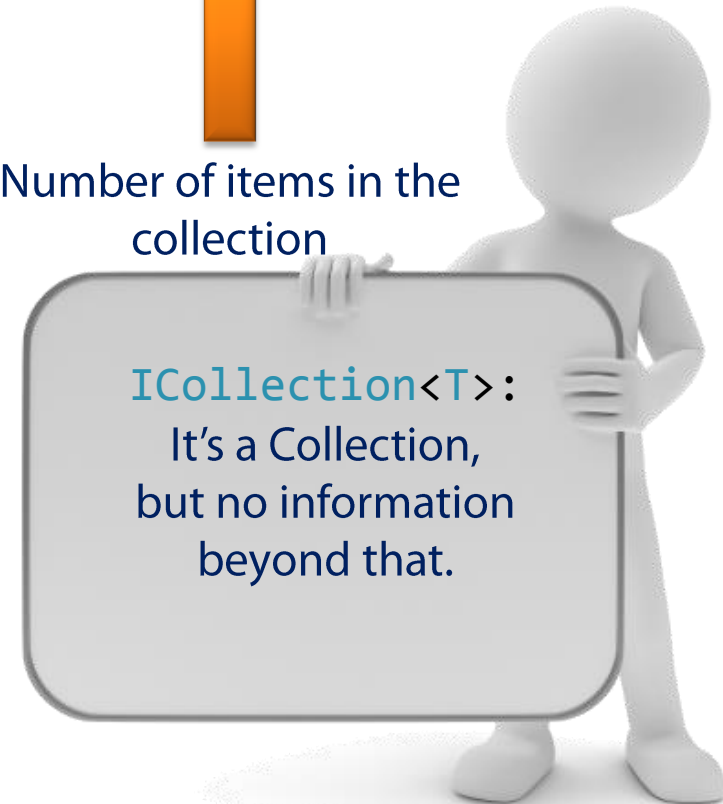


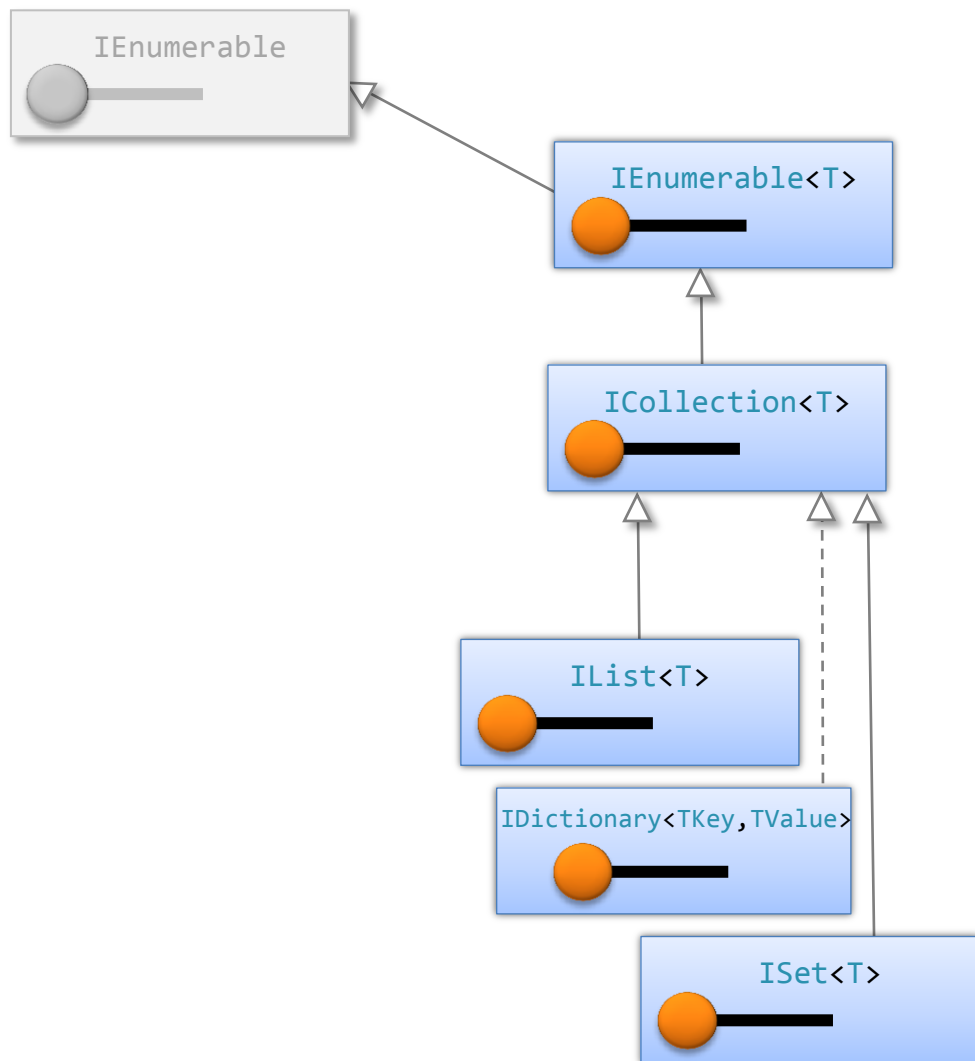
```
int ICollection<T>.Count
```



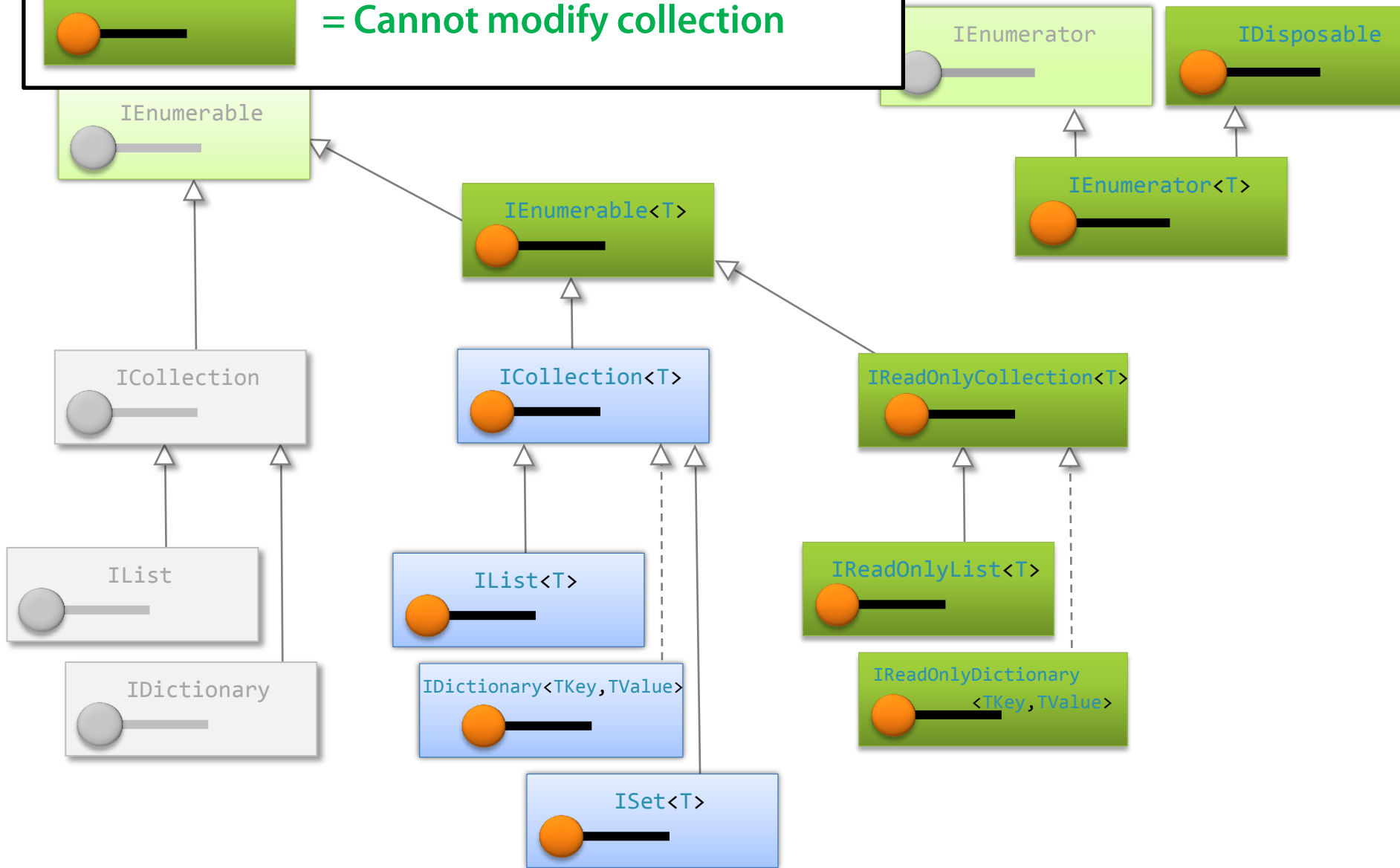
Number of items in the collection

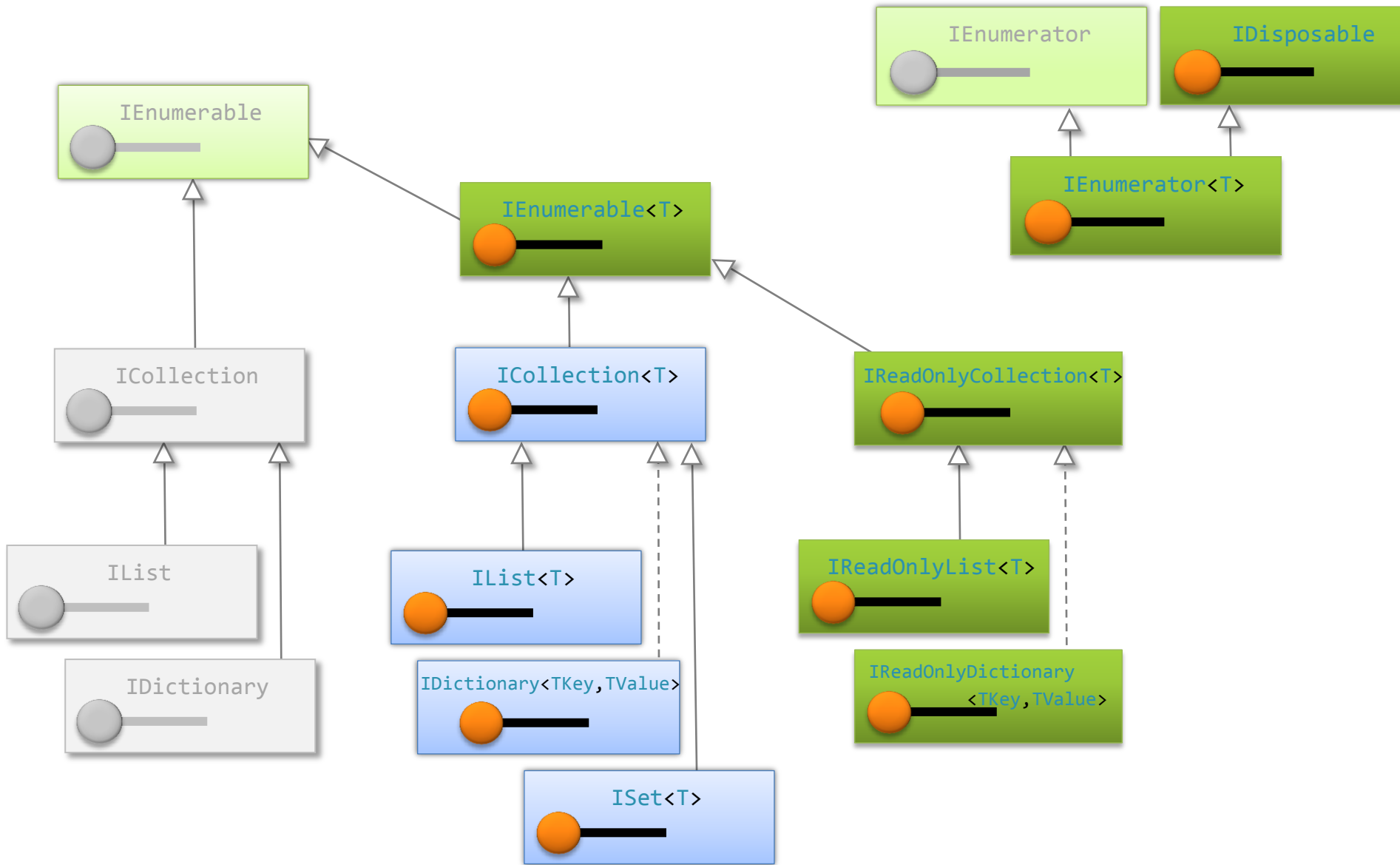
`ICollection<T>:`
It's a Collection,
but no information
beyond that.



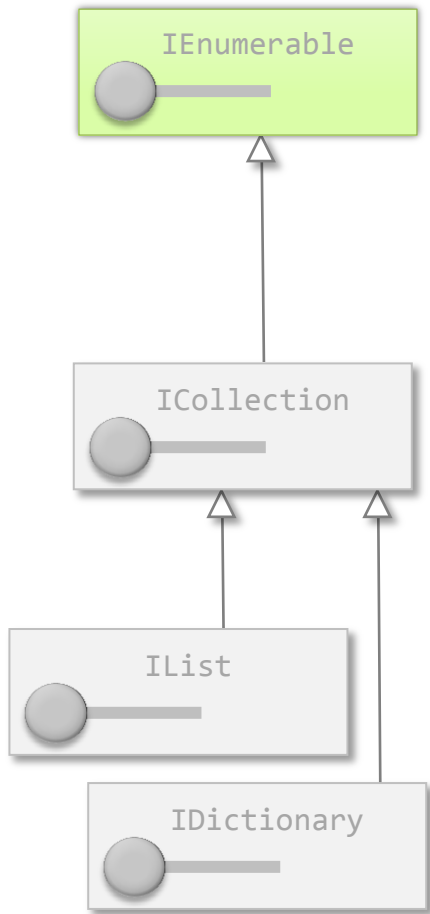


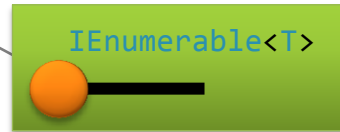
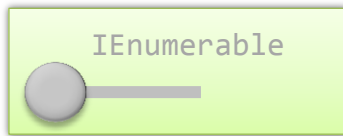
 = Cannot modify collection





.NET 1.x interfaces



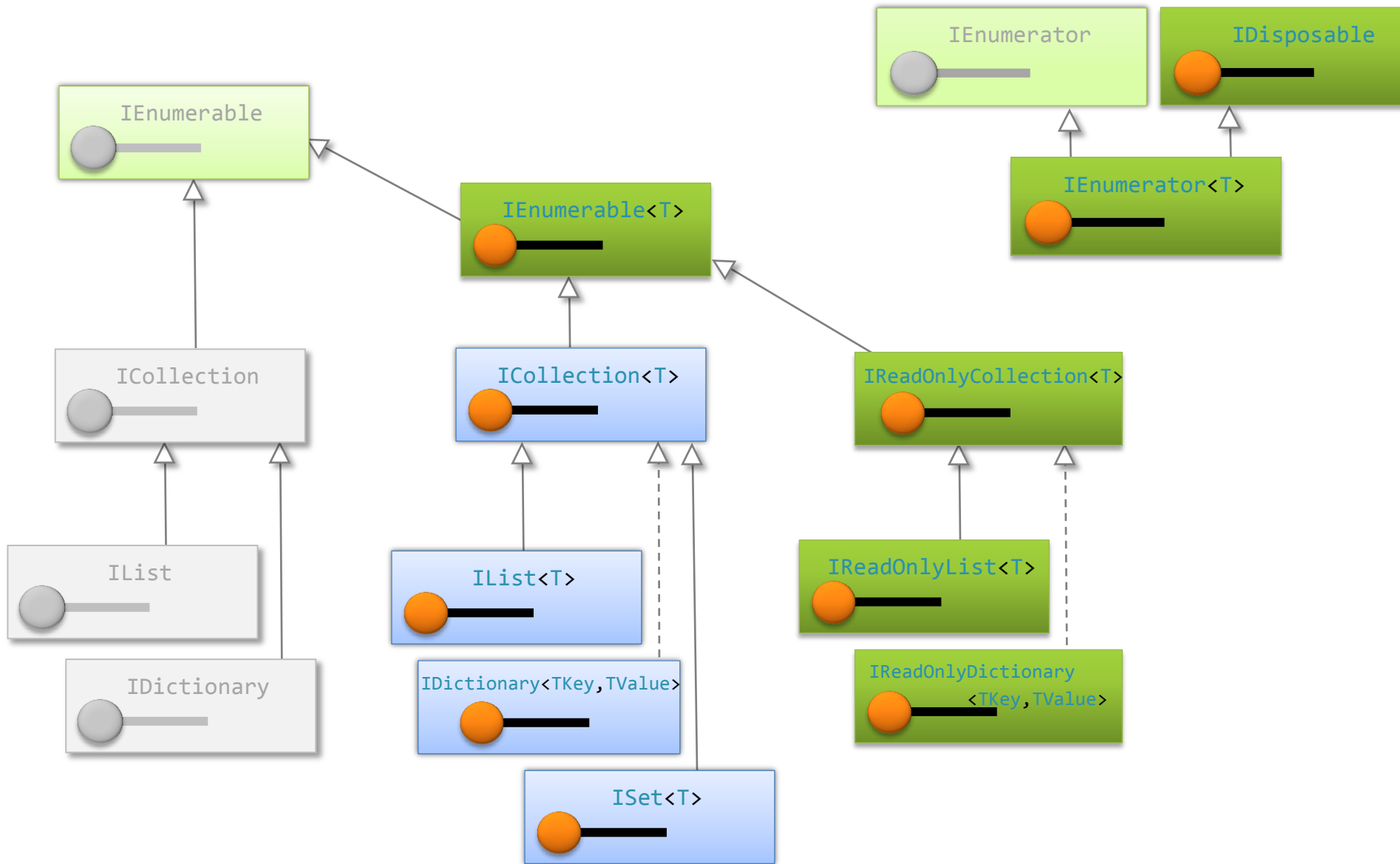


`IEnumerable<T>`
derives from
`IEnumerable`

Hence this is valid...

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

A 3D white figure stands on the left, holding a large grey rectangular sign with rounded corners. The sign contains the text: `.NET 1.x code
can consume
generic collections`



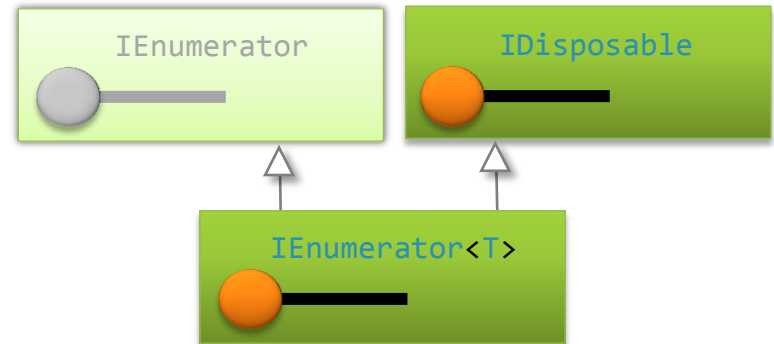
`IEnumerator<T>`



`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

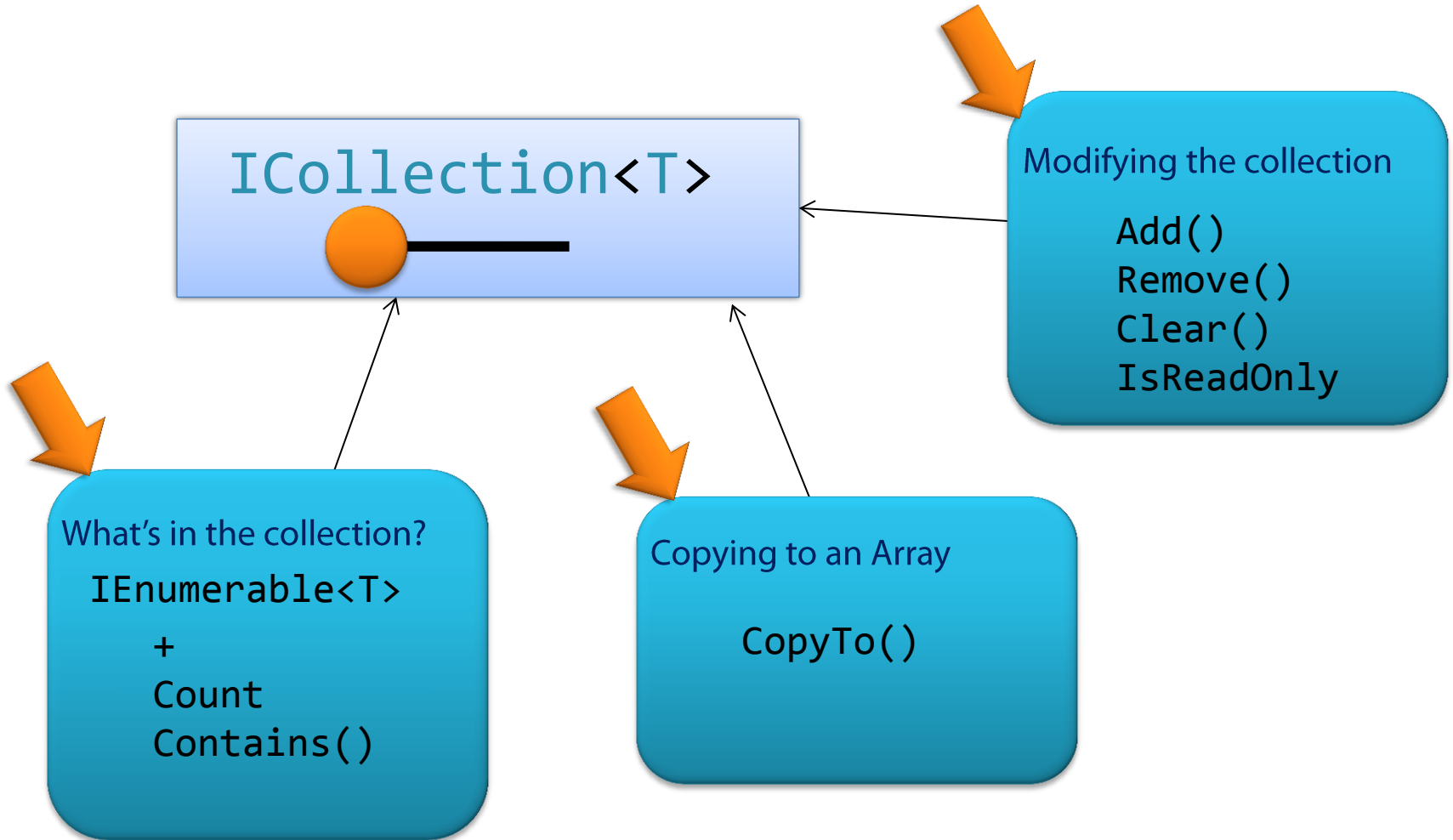
Collection

(Enumerable)

Enumerator

foreach
does all this
for you

Code Demo



`ICollection<T>`



```
graph TD; A[What's in the collection? IEnumerable<T> + Count Contains()] --> B[ICollection<T>]; C[Copying to an Array CopyTo()] --> B; D[Modifying the collection Add() Remove() Clear() IReadOnly] --> B;
```



Modifying the collection

`Add()`
`Remove()`
`Clear()`
`ReadOnly`

What's in the collection?

`IEnumerable<T>`

+

`Count`

`Contains()`

Copying to an Array

`CopyTo()`

ICollection<T>



The diagram illustrates the components of the **ICollection<T>** interface. At the top center is a light blue rectangular box labeled **ICollection<T>**. Below this box is an orange circle followed by a horizontal line. To the right of the box is a blue rounded rectangle titled "Modifying the collection" which lists the methods **Add()**, **Remove()**, **Clear()**, and the property **IsReadOnly**. An arrow points from this blue box to the **ICollection<T>** box. Below the **ICollection<T>** box are two light gray rounded rectangles. The left one is titled "What's in the collection?" and lists **IEnumerable<T>**, **+ Count**, and **Contains()**. An arrow points from this box to the **ICollection<T>** box. The right one is titled "Copy" and contains a white box with the text "Returns whether OK to call Add() / Remove() / Clear()". An orange arrow points from this text box up to the **IsReadOnly** property in the blue box.

Modifying the collection

Add()
Remove()
Clear()
IsReadOnly

What's in the collection?

IEnumerable<T>

+ Count
Contains()

Copy
Returns whether OK to call
Add() / Remove() / Clear()

`ICollection<T>`

```
graph TD; A[What's in the collection? IEnumerable<T> + Count Contains()] --> B[ICollection<T>]; C[Modifying the collection Add() Remove() Clear() IsReadOnly] --> B; D[Copying to an Array CopyTo()] --> B;
```

Modifying the collection

`Add()`
`Remove()`
`Clear()`
`IsReadOnly`

What's in the collection?

`IEnumerable<T>`

+

`Count`

`Contains()`

Copying to an Array

`CopyTo()`

ICollection<T>



The diagram illustrates the components of the **ICollection<T>** interface. At the top center is a light blue rectangular box labeled **ICollection<T>**. Below the text in this box is an orange circle followed by a horizontal black line. To the right of the main box is a light gray rounded rectangle titled "Modifying the collection" which lists the methods **Add()**, **Remove()**, **Clear()**, and **IsReadOnly**. Below the main box are two blue rounded rectangles. The one on the left is titled "What's in the collection?" and lists **IEnumerable<T>**, **+ Count**, and **Contains()**. The one on the right is titled "Copying to an Array" and lists **CopyTo()**. Arrows point from the bottom of the main box to each of these two blue boxes. Additionally, a red 'X' is drawn over the **Contains()** method in the left blue box and the **CopyTo()** method in the right blue box.

Modifying the collection

Add()
Remove()
Clear()
IsReadOnly

What's in the collection?

IEnumerable<T>

+

Count

Contains()

Copying to an Array

CopyTo()

`ICollection<T>`



What's in the collection?

`IEnumerator<T>`

+

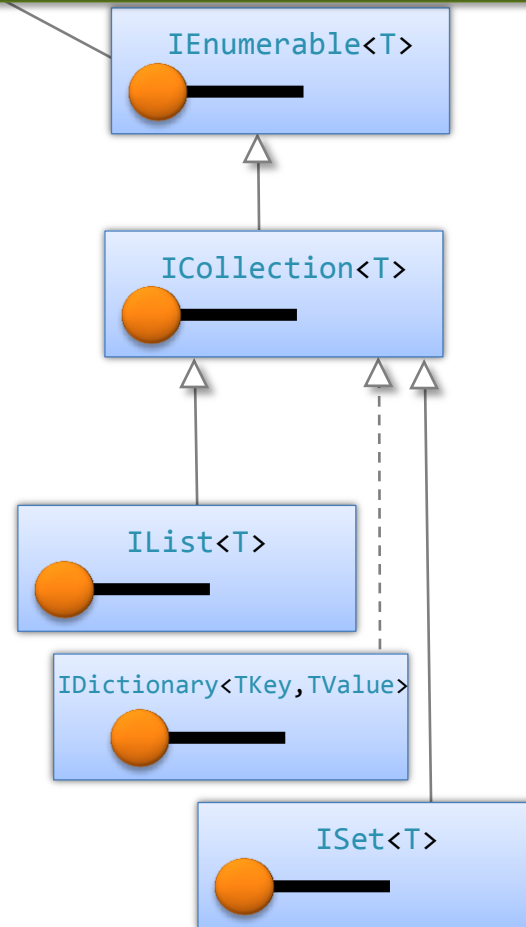
Count

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.

`IList<T>`

A light blue rectangular box containing the text 'IList<T>'. Below the text is an orange circle with a black horizontal line extending to the right. An arrow points from the bottom-left box to the orange circle, and another arrow points from the top-right box to the right side of the box.

`= ICollection<T> + ...`

Modifying the collection

`Insert()`

`RemoveAt()`

What's in the collection?

`this[index]`

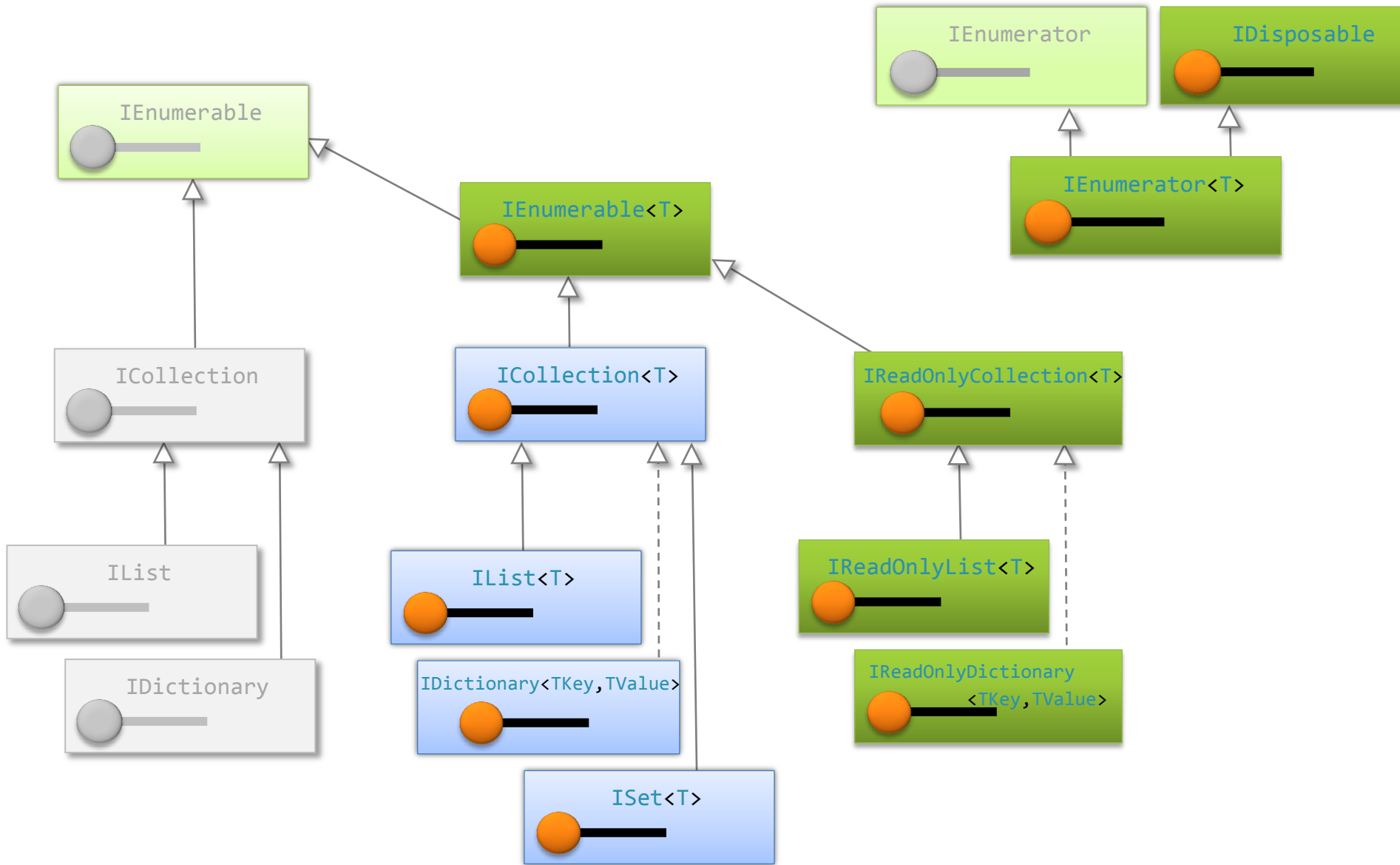
`IndexOf()`

`// look up element`

`string saturday = daysOfWeek[5];`

`// replace element`

`daysOfWeek[5] = "SlaveDay";`



`ReadOnlyList<T>`



`= IReadOnlyCollection<T> + ...`

What's in the collection?

`this[index]`

(Read only)

`// look up element`

`string saturday = daysOfWeek[5];`

`ReadOnlyList<T>`



=

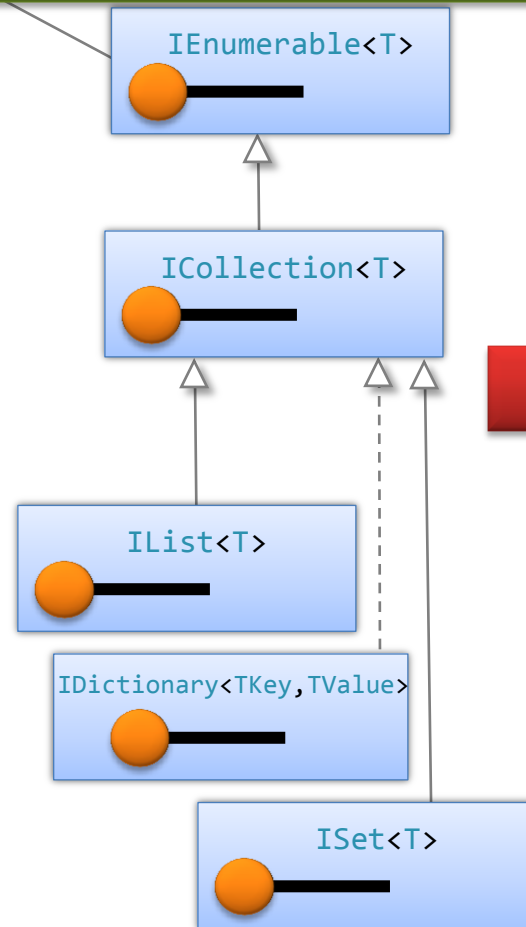
`IEnumerable<T>`

- + 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>`



The diagram illustrates the components of the `IDictionary<TKey, TValue>` interface. At the top center is a light blue rectangular box containing the text `IDictionary<TKey, TValue>`. Below this text, there is an orange circle followed by a horizontal line. An arrow points from the orange circle down to a blue rounded rectangle at the bottom left. Another arrow points from the horizontal line to a blue rounded rectangle at the top right. Below the orange circle and line, the text `= ICollection<T> + ...` is displayed.

`= ICollection<T> + ...`

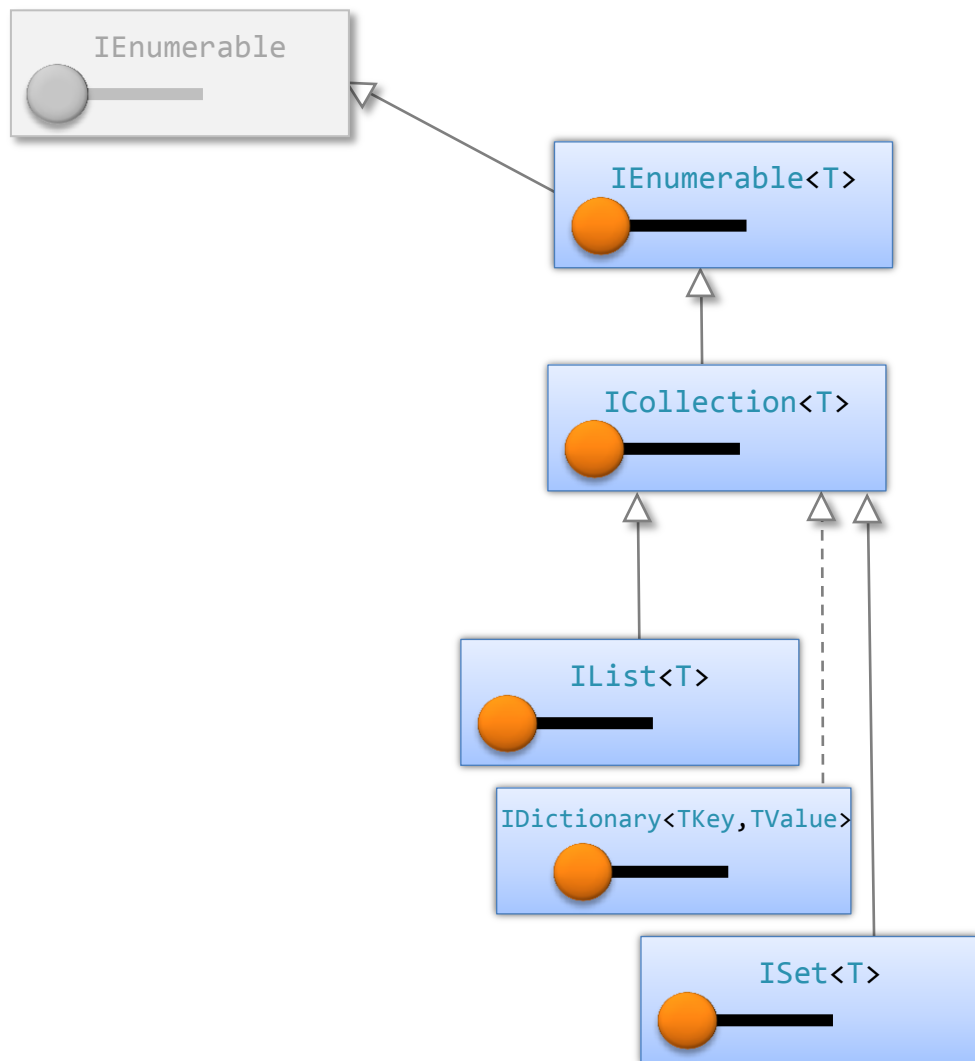
Modifying the collection

`Add()`
`Remove()`

What's in the collection?

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

Like a list, but with
keys instead of an
index



Dictionary of countries

Keys

Values

DE

Germany

GB

United Kingdom

US

USA

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

Collection of
key-value pairs



Dictionary of countries

Elements

```
new KeyValuePair("DE", "Germany");
```

```
new KeyValuePair("GB", "United Kingdom");
```

```
new KeyValuePair("US", "USA");
```

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



Collection of
key-value pairs

Dictionary of countries

Elements

```
new KeyValuePair("DE", "Germany");
```

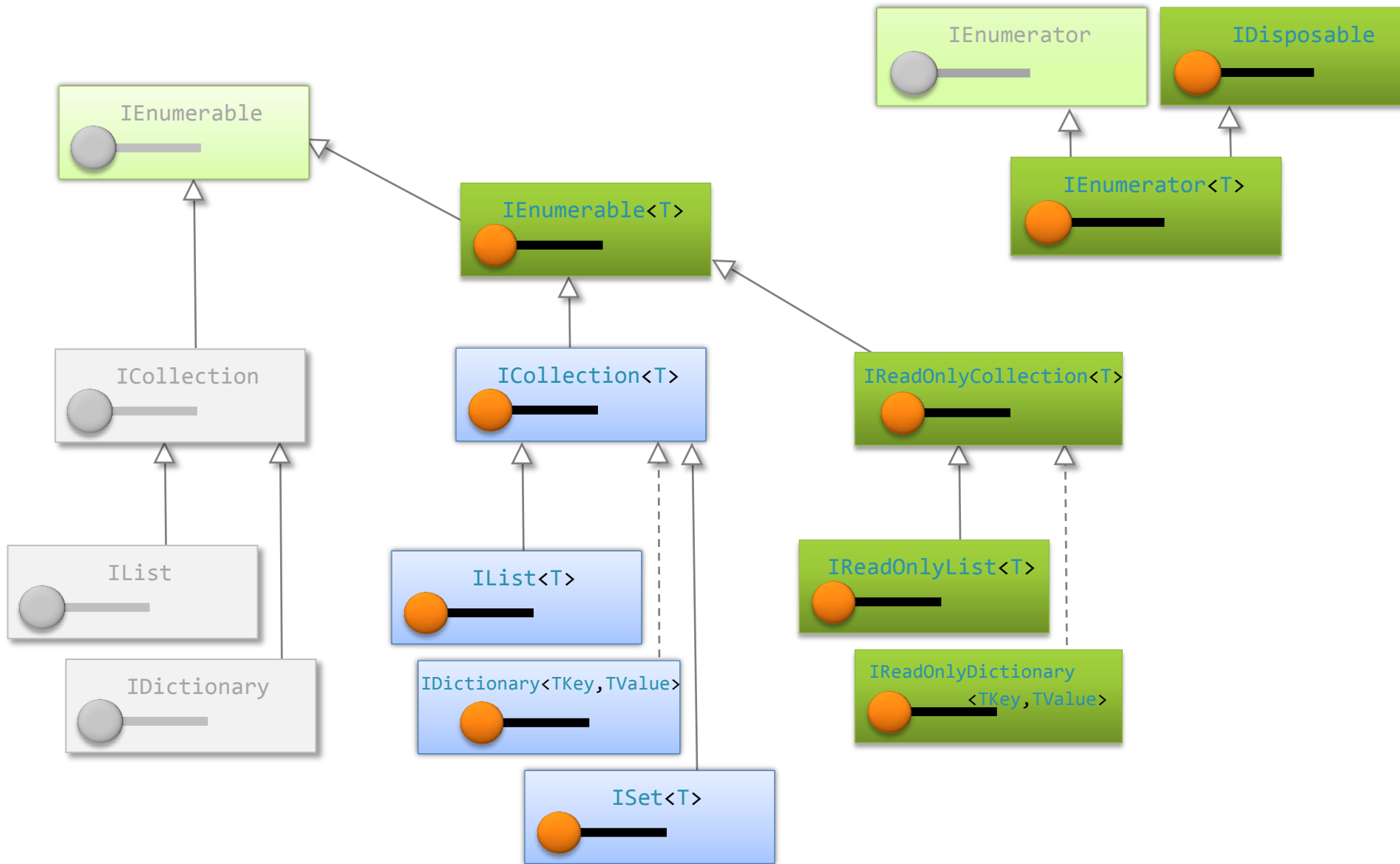
```
new KeyValuePair("GB", "United Kingdom");
```

```
new KeyValuePair("US", "USA");
```

`IDictionary<TKey, TValue>`

Derives from

`ICollection<KeyValuePair<TKey, TValue>>`



`IDictionary<TKey, TValue>`



The diagram illustrates the components of the `IDictionary<TKey, TValue>` interface. At the top center is a light blue rectangular box containing the interface name. Below the name, an orange circle is positioned on a horizontal line. An arrow points from the bottom-left box to this orange circle. To the right of the interface name, an arrow points from the top-right box to the same horizontal line. Below the interface name, the text `= ICollection<T> + ...` is displayed. The bottom-left box, titled 'What's in the collection?', lists `this[index]`, 'Keys', 'Values', `TryGetValue()`, and `ContainsKey()`. The top-right box, titled 'Modifying the collection', lists `Add()` and `Remove()`.

`= 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()`

`ReadOnlyDictionary<TKey, TValue>`



`= IReadOnlyCollection<T> + ...`

What's in the collection?

`this[index]`

Keys

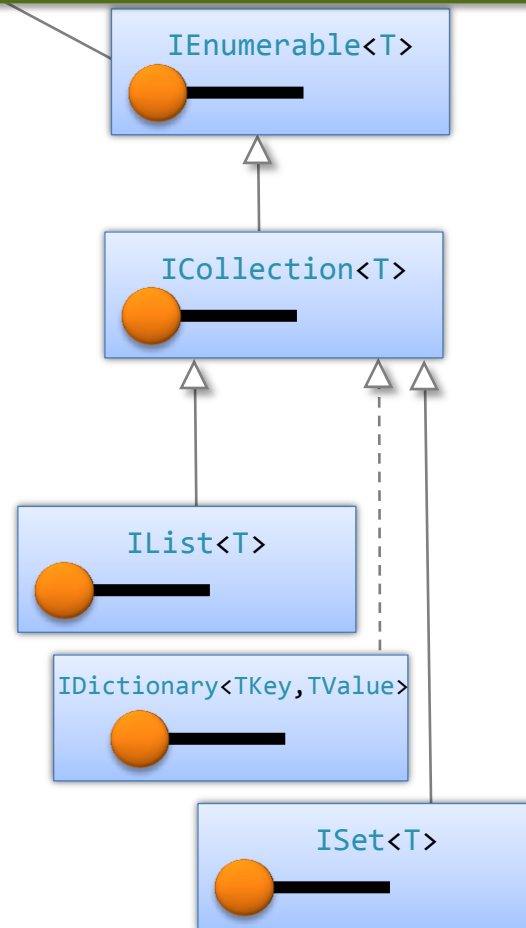
Values

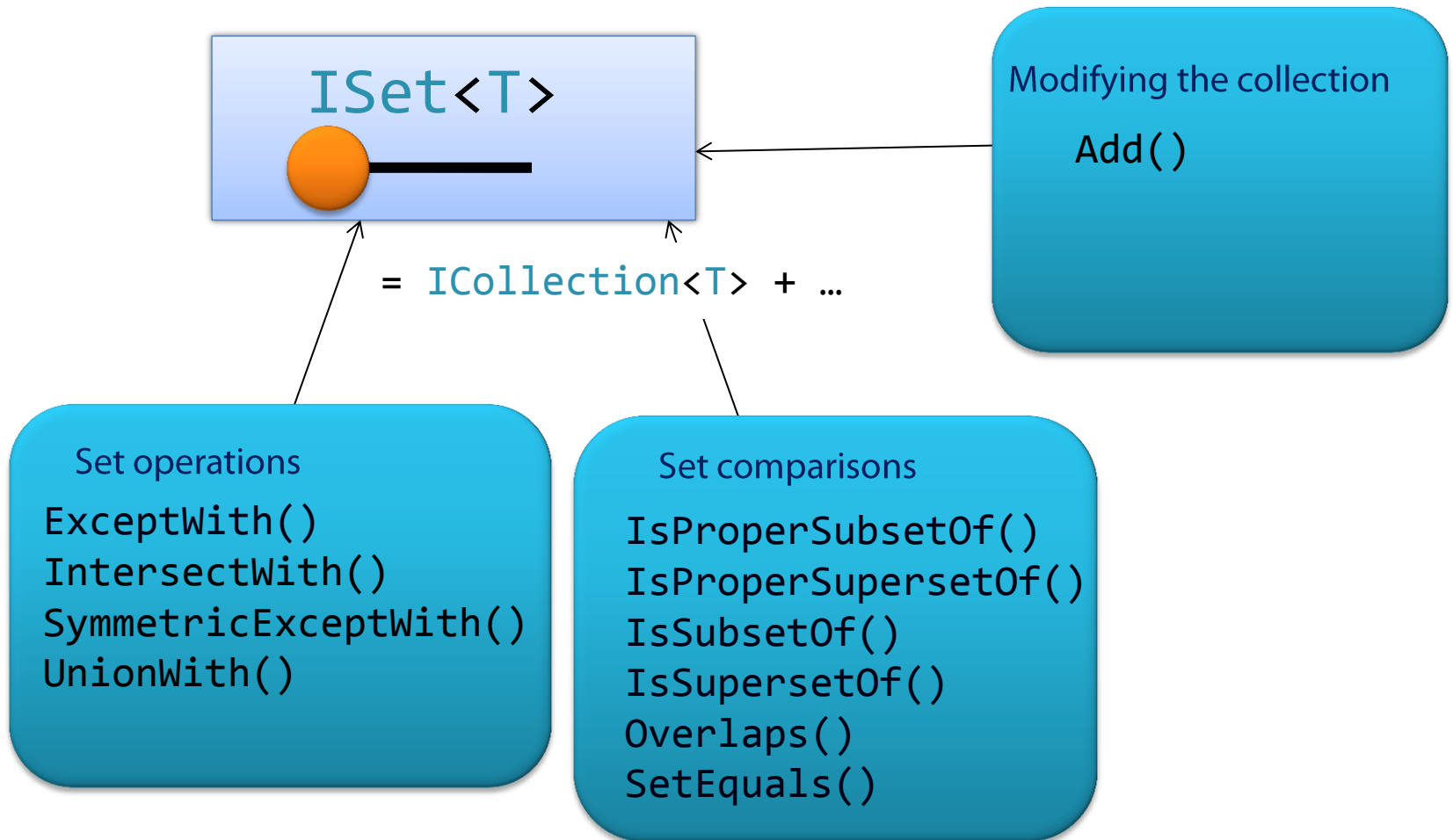
`TryGetValue()`

`ContainsKey()`

`ISet<T>:`

“I can do set operations with other collections”





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

