C# Collections

Enumerators

Simon Robinson
http://TechieSimon.com
@TechieSimon

hardcore developer training

Stack<T>

ObservableCollection<T>

SortedList<TKey, TValue>

LinkedList<T>

List<T>

Dictionary<TKey, TValue>

SortedSet<T>

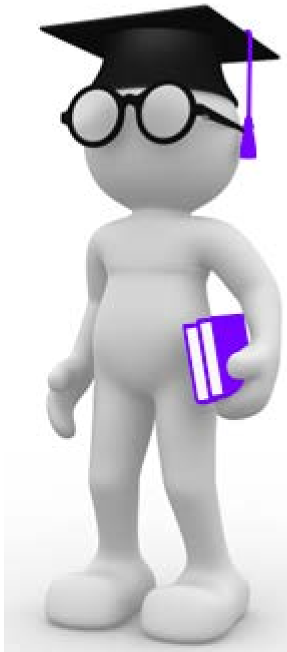
Collection<T>

foreach
works on all
of these and
more!

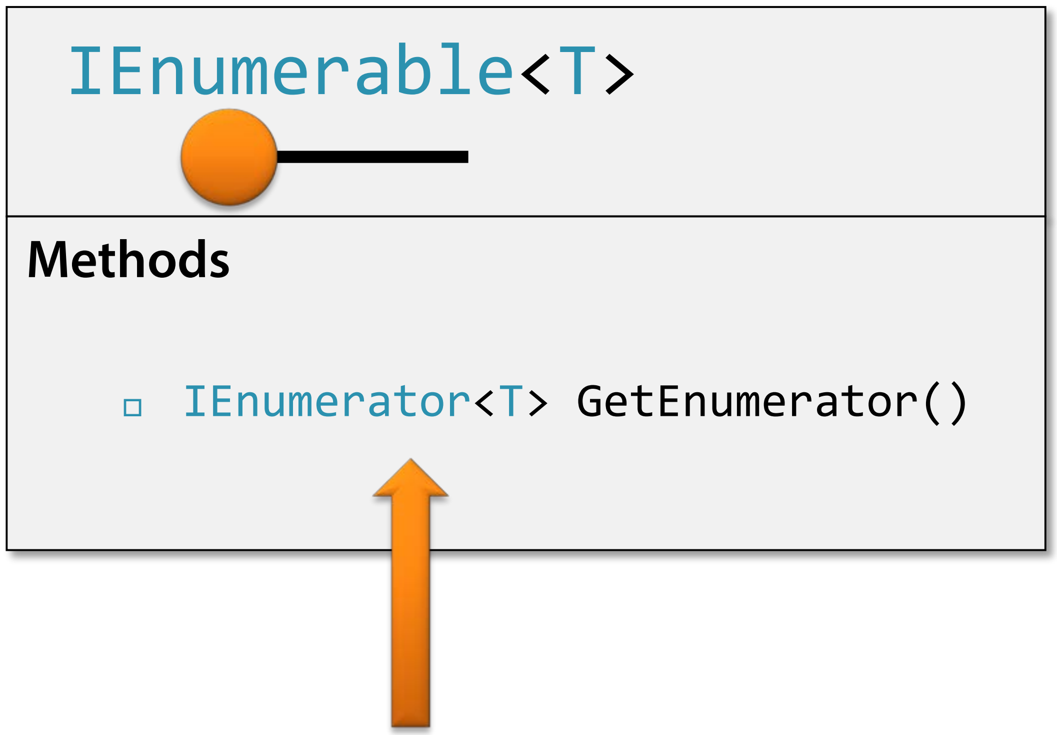
How?

Module Overview

- **Iterating under the hood**
  - `IEnumerable<T>` and `IEnumerator<T>`
- **The `foreach` loop**
  - How it works using enumerators
- **Enumerating collections that change**
- **Writing your own enumerators**
- **Enumerator covariance**



Enumerating a Collection



IEnumerable<T>

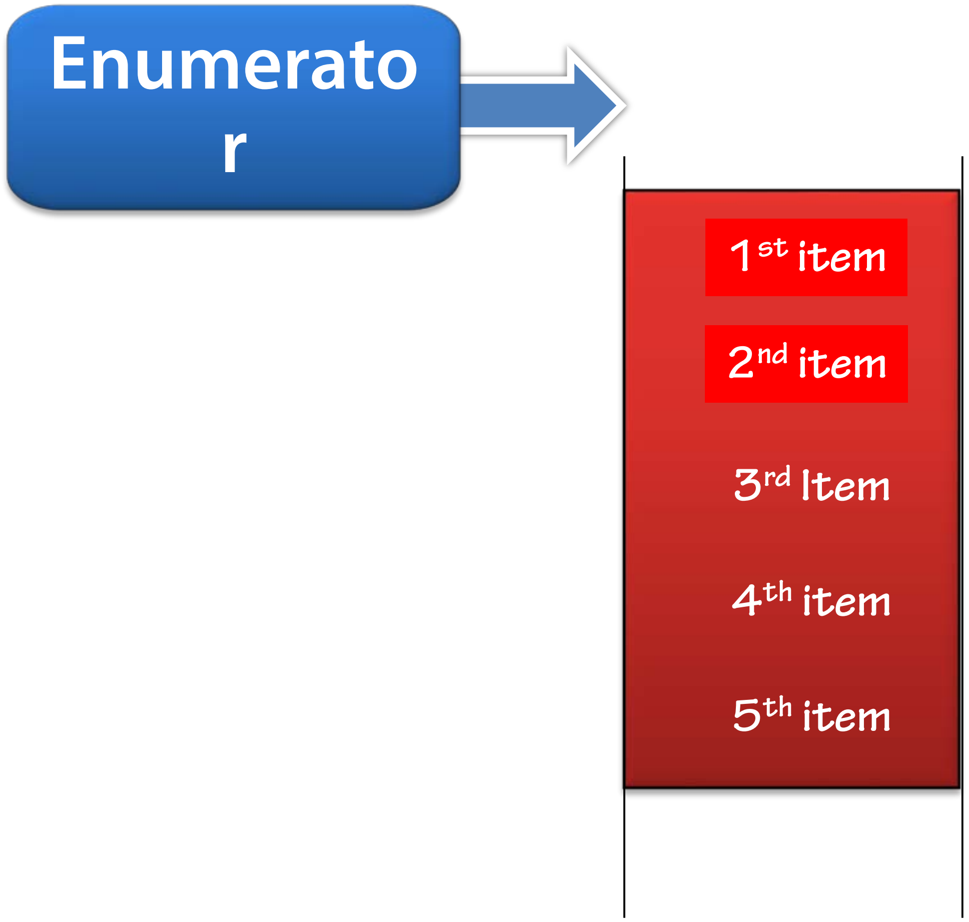
Methods

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

Use this method to get an enumerator to enumerate a collection

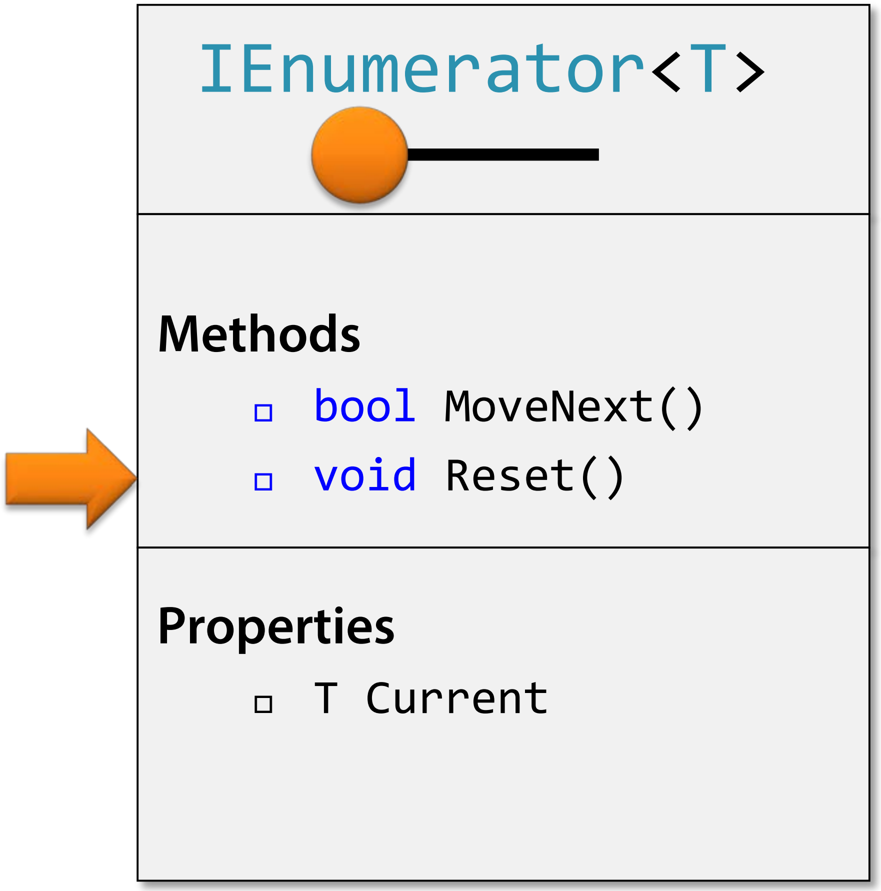
Enumerating a Collection

```
bool MoveNext();
```



True if there is at least one more item in the collection

Code Demo



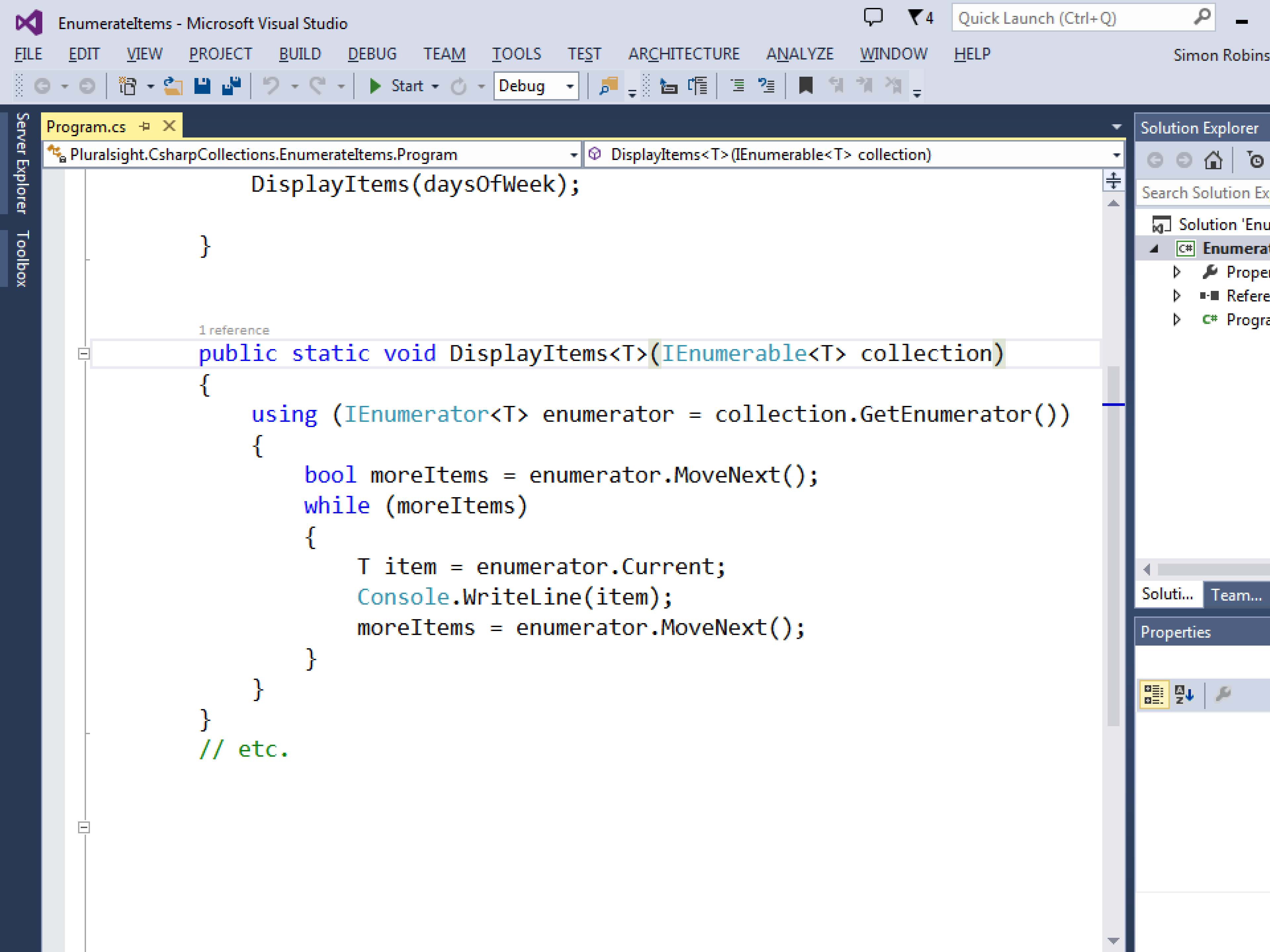
IEnumerator<T>

Methods

```
▫ bool MoveNext()
▫ void Reset()
```

Properties

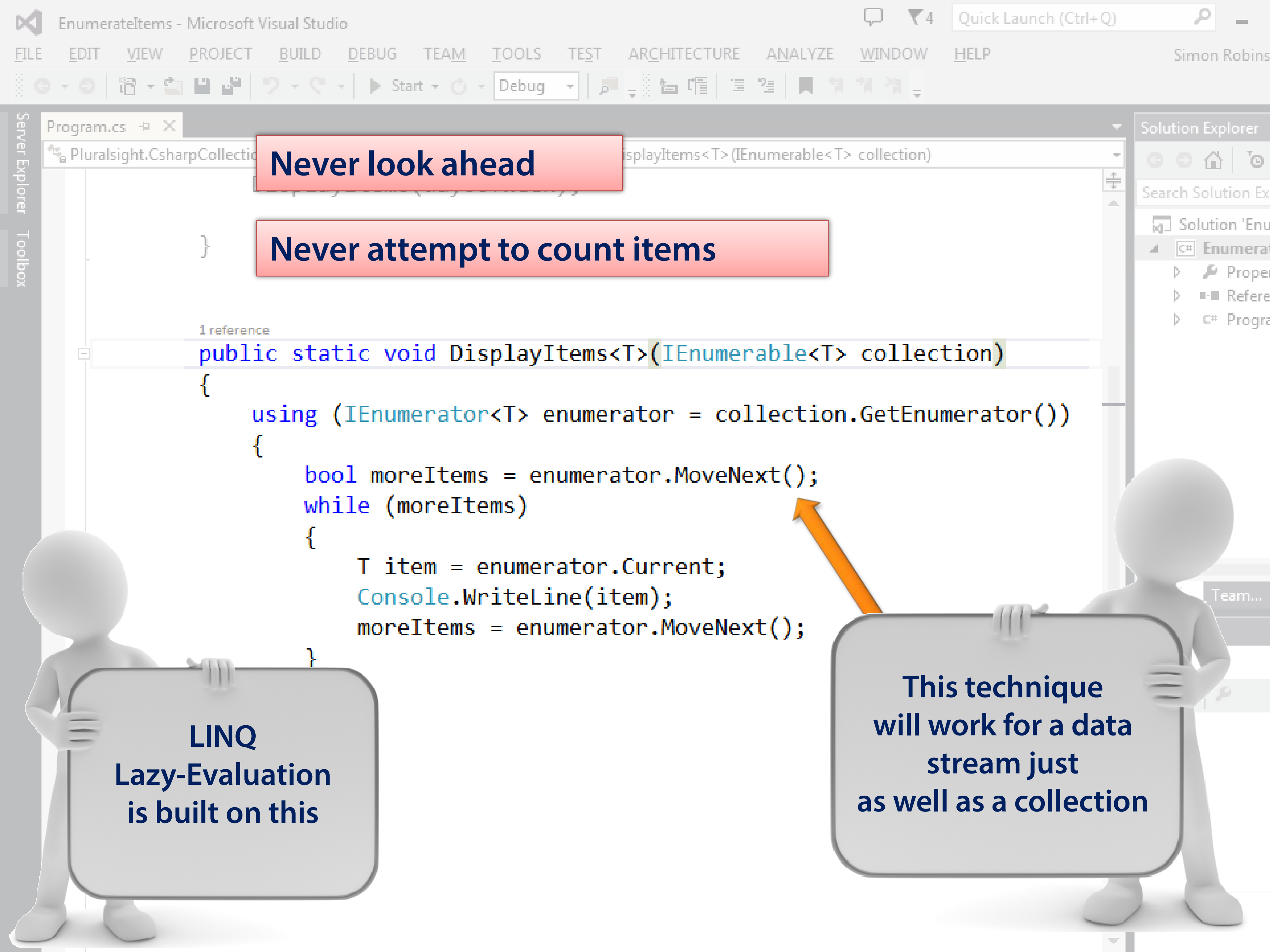
```
▫ T Current
```


EnumerateItems - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP
Simon Robins
Start
Debug
Program.cs
Pluralsight.CsharpCollections.EnumerateItems.Program
DisplayItems<T>(IEnumerable<T> collection)
Server Explorer
Toolbox
Solution Explorer
Search Solution Ex
Solution 'Enu
Enumerat
Proper
Refere
Progra
Soluti...
Team...
Properties

```
            DisplayItems(daysOfWeek);

        }


        1 reference
        public static void DisplayItems<T>(IEnumerable<T> collection)
        {
            using (IEnumerator<T> enumerator = collection.GetEnumerator())
            {
                bool moreItems = enumerator.MoveNext();
                while (moreItems)
                {
                    T item = enumerator.Current;
                    Console.WriteLine(item);
                    moreItems = enumerator.MoveNext();
                }
            }
        }
        // etc.
```


EnumerateItems - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP
Simon Robins
Start
Debug
Program.cs
Pluralsight.CsharpCollecti
isplayItems<T>(IEnumerable<T> collection)
Solution Explorer
Search Solution Ex
Solution 'Enu
Enumera
Prope
Refere
Progra
Server Explorer
Toolbox
Team...

Never look ahead

Never attempt to count items

```
    }

1 reference
public static void DisplayItems<T>(IEnumerable<T> collection)
{
    using (IEnumerator<T> enumerator = collection.GetEnumerator())
    {
        bool moreItems = enumerator.MoveNext();
        while (moreItems)
        {
            T item = enumerator.Current;
            Console.WriteLine(item);
            moreItems = enumerator.MoveNext();
        }
```

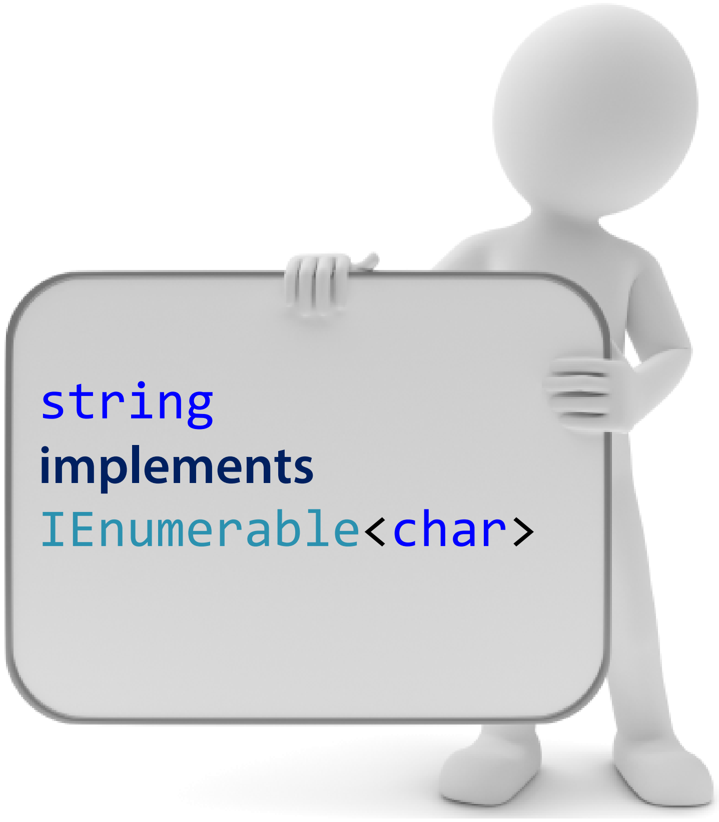
LINQ Lazy-Evaluation is built on this

This technique will work for a data stream just as well as a collection

`string`

```
"Hello, World!"
```

This is just a list of characters



```
string
```
implements
```
IEnumerable<char>
```

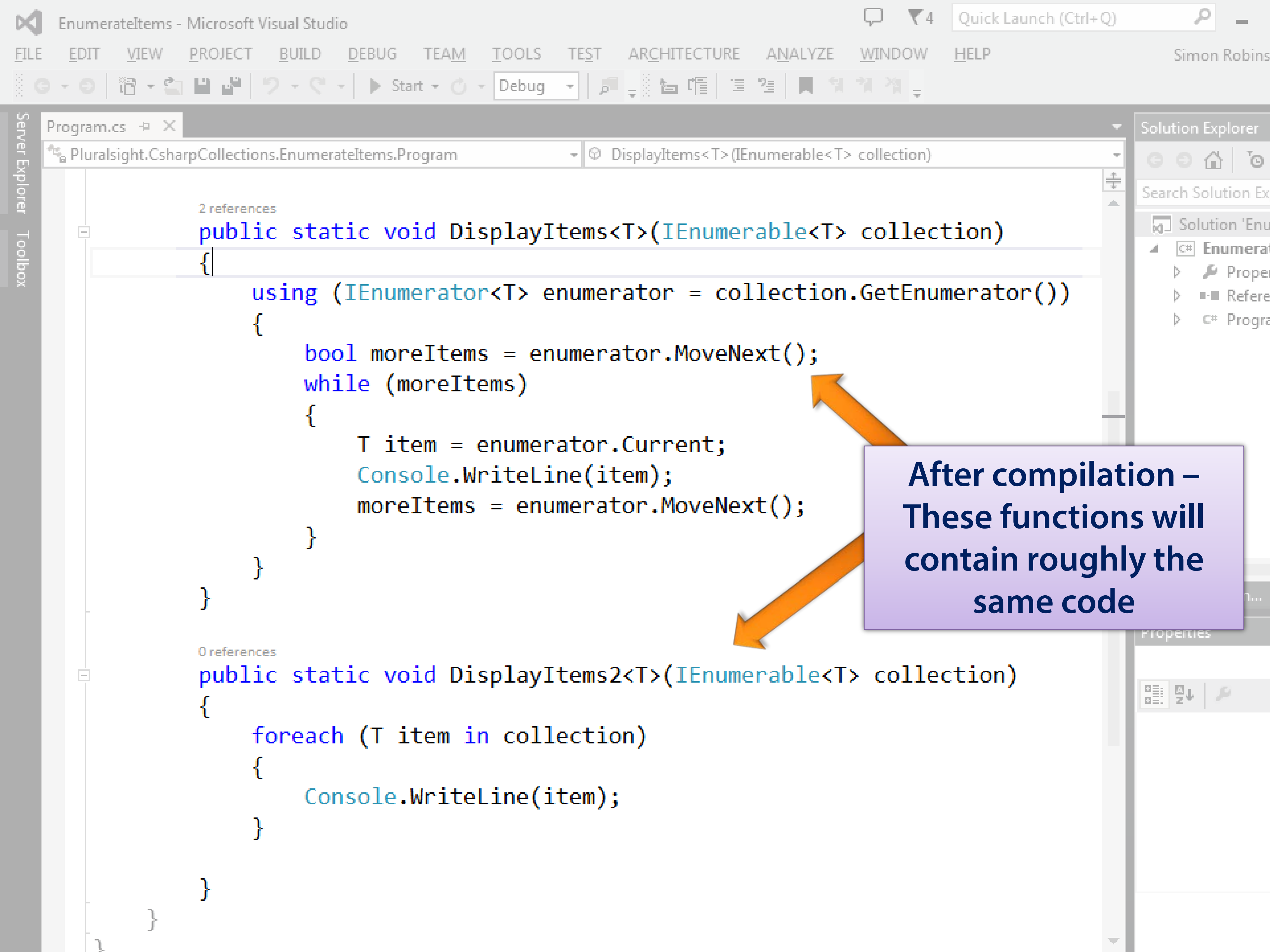
foreach Loops

When the compiler sees this…

```
foreach (T item in collection)
{
   // do something
}
```

…it replaces it with something equivalent to this…

```
using (IEnumerator<T> enumerator collection.GetEnumerator())
{
    bool moreItems = enumerator.MoveNext();
    while (moreItems)
    {
        T item = enumerator.Current;
        // do something
        moreItems = enumerator.MoveNext();
    }
}
```


EnumerateItems - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP
Quick Launch (Ctrl+Q)
Simon Robins
Start
Debug
Program.cs
Pluralsight.CsharpCollections.EnumerateItems.Program
DisplayItems<T>(IEnumerable<T> collection)
Solution Explorer
Search Solution Ex
Solution 'Enu
Enumera
Prope
Refere
Progra
Properties
Server Explorer
Toolbox

```
        2 references
        public static void DisplayItems<T>(IEnumerable<T> collection)
        {
            using (IEnumerator<T> enumerator = collection.GetEnumerator())
            {
                bool moreItems = enumerator.MoveNext();
                while (moreItems)
                {
                    T item = enumerator.Current;
                    Console.WriteLine(item);
                    moreItems = enumerator.MoveNext();
                }
            }
        }

        0 references
        public static void DisplayItems2<T>(IEnumerable<T> collection)
        {
            foreach (T item in collection)
            {
                Console.WriteLine(item);
            }

        }
    }
}
```

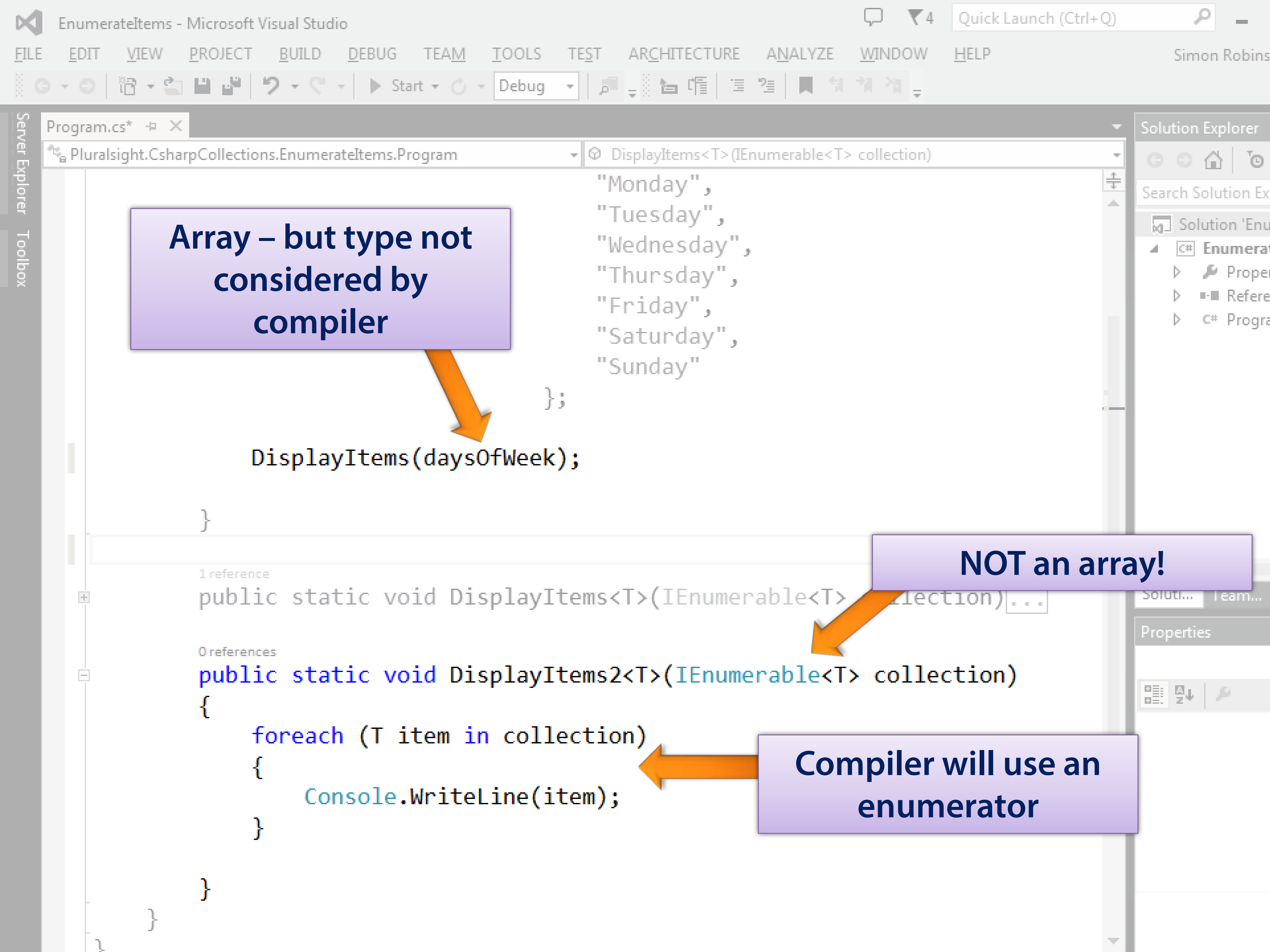
After compilation – These functions will contain roughly the same code

foreach Loops

BUT….

```
foreach (T item in collection)
{
   // etc.
}
```

If this is an array type
- the compiler will use a for loop
- NOT an Enumerator


EnumerateItems - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP
Start
Debug
Program.cs*
Pluralsight.CsharpCollections.EnumerateItems.Program
DisplayItems<T>(IEnumerable<T> collection)
Solution Explorer
Properties

```
                "Monday",
                "Tuesday",
                "Wednesday",
                "Thursday",
                "Friday",
                "Saturday",
                "Sunday"
            };

        DisplayItems(daysOfWeek);

    }

    1 reference
    public static void DisplayItems<T>(IEnumerable<T> collection)...

    0 references
    public static void DisplayItems2<T>(IEnumerable<T> collection)
    {
        foreach (T item in collection)
        {
            Console.WriteLine(item);
        }

    }
  }
}
```

Array – but type not considered by compiler

NOT an array!

Compiler will use an enumerator

$$\begin{tikzcd}\text{Collection} \arrow[r, shift left] & \text{Enumerator} \arrow[l, shift left]\end{tikzcd}$$

Why can't a collection enumerate its own elements?
Why is the enumerator a different object?

```
foreach (string item in daysOfWeek)
{
```

Imagine if these variables refer to the same collection…

```
foreach (string item in theDays)
{
```

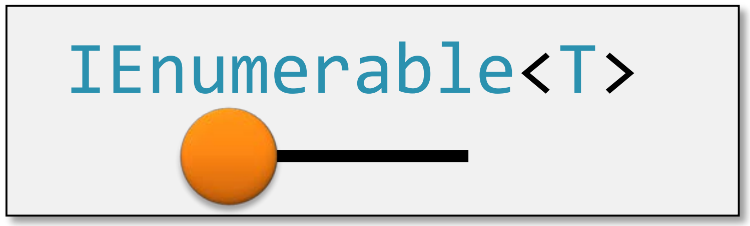
… and the loops ran at the same time…

```
foreach (string item in daysOfWeek)
{
```

These loops must run independently

```
foreach (string item in theDays)
{
```

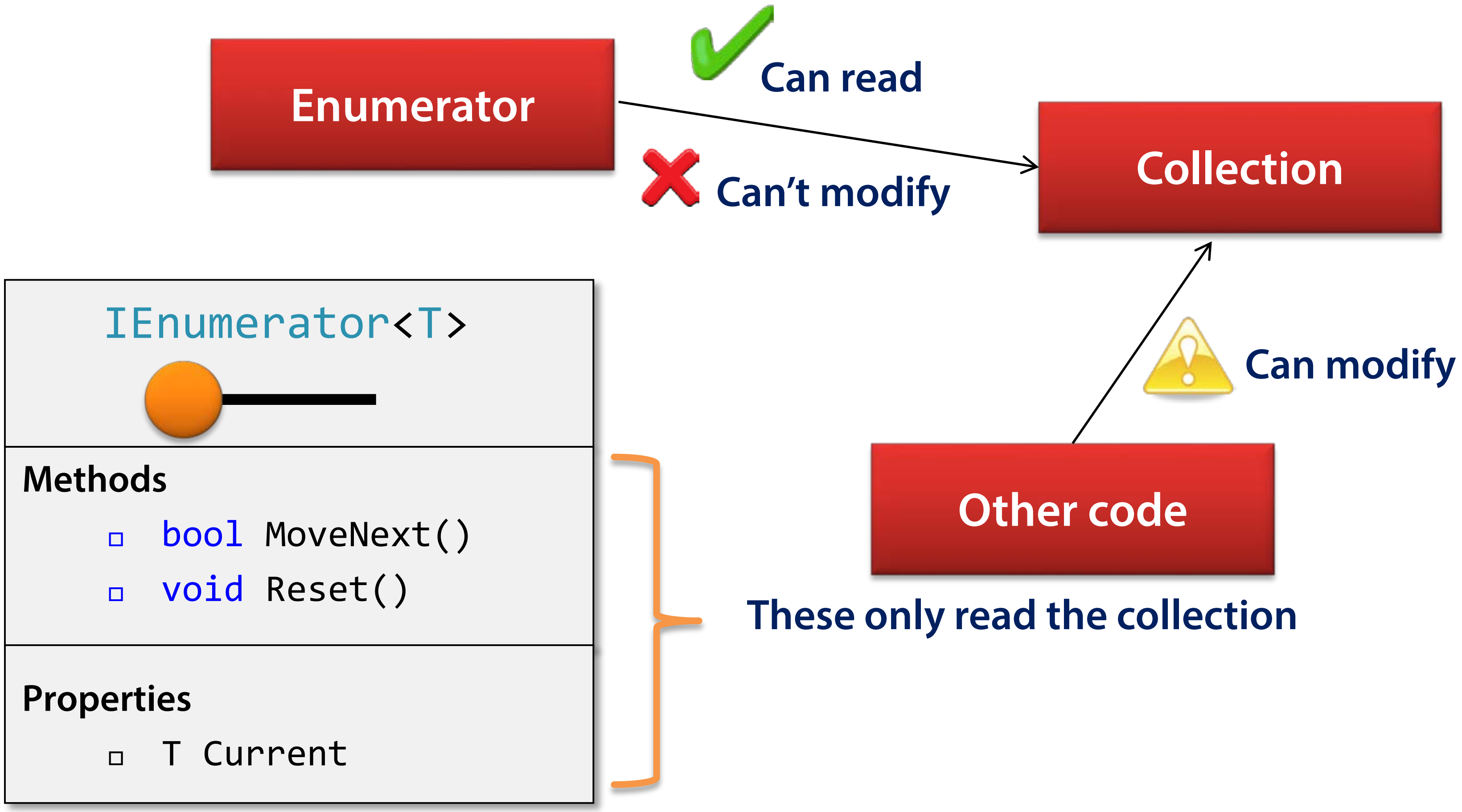
Each must have its own independent enumerator



```
T GetEnumerator()
```

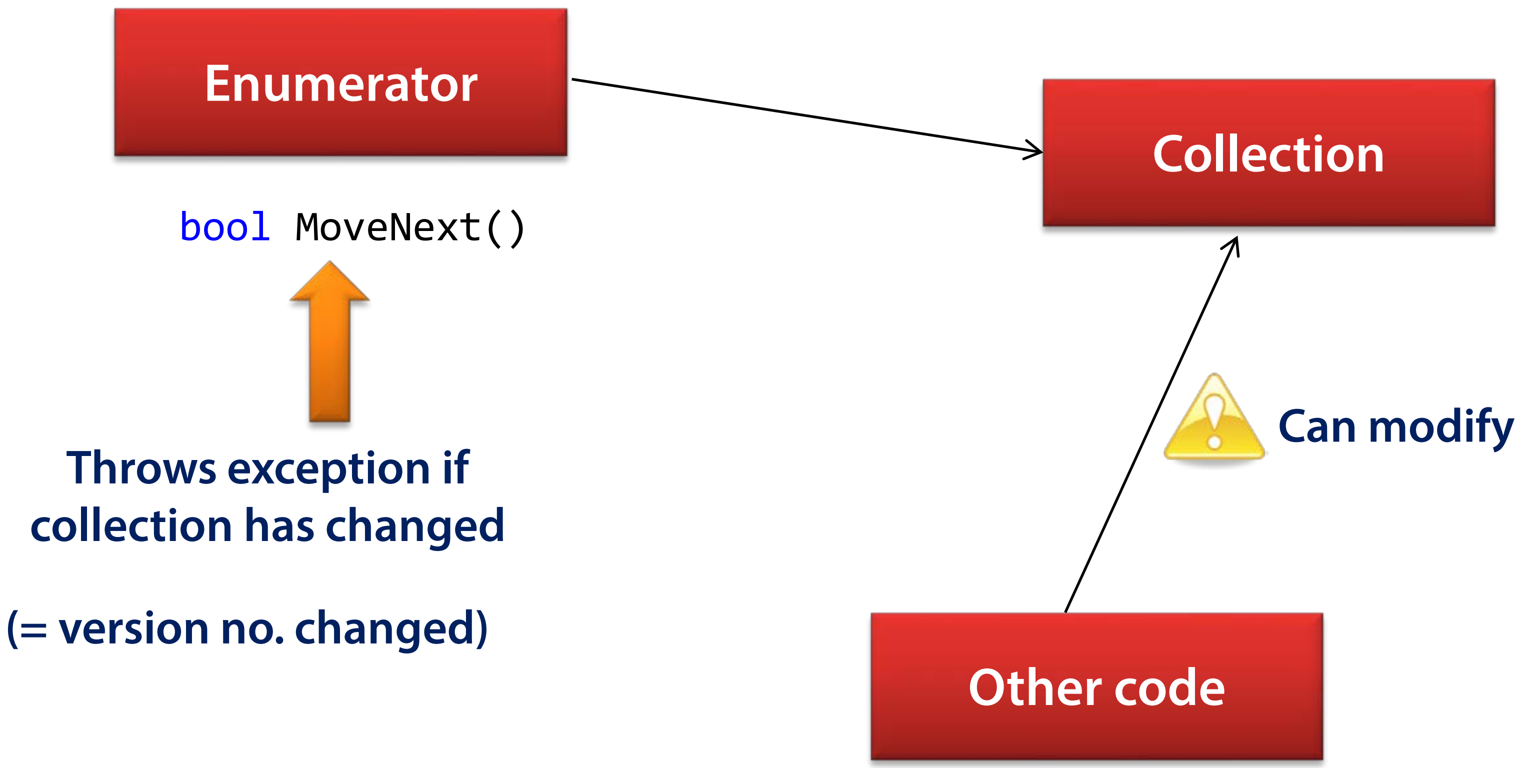
Each call must return a fresh enumerator – no caching allowed!

Enumerating a Collection that Changes



Code Demo

Enumerating a Collection that Changes

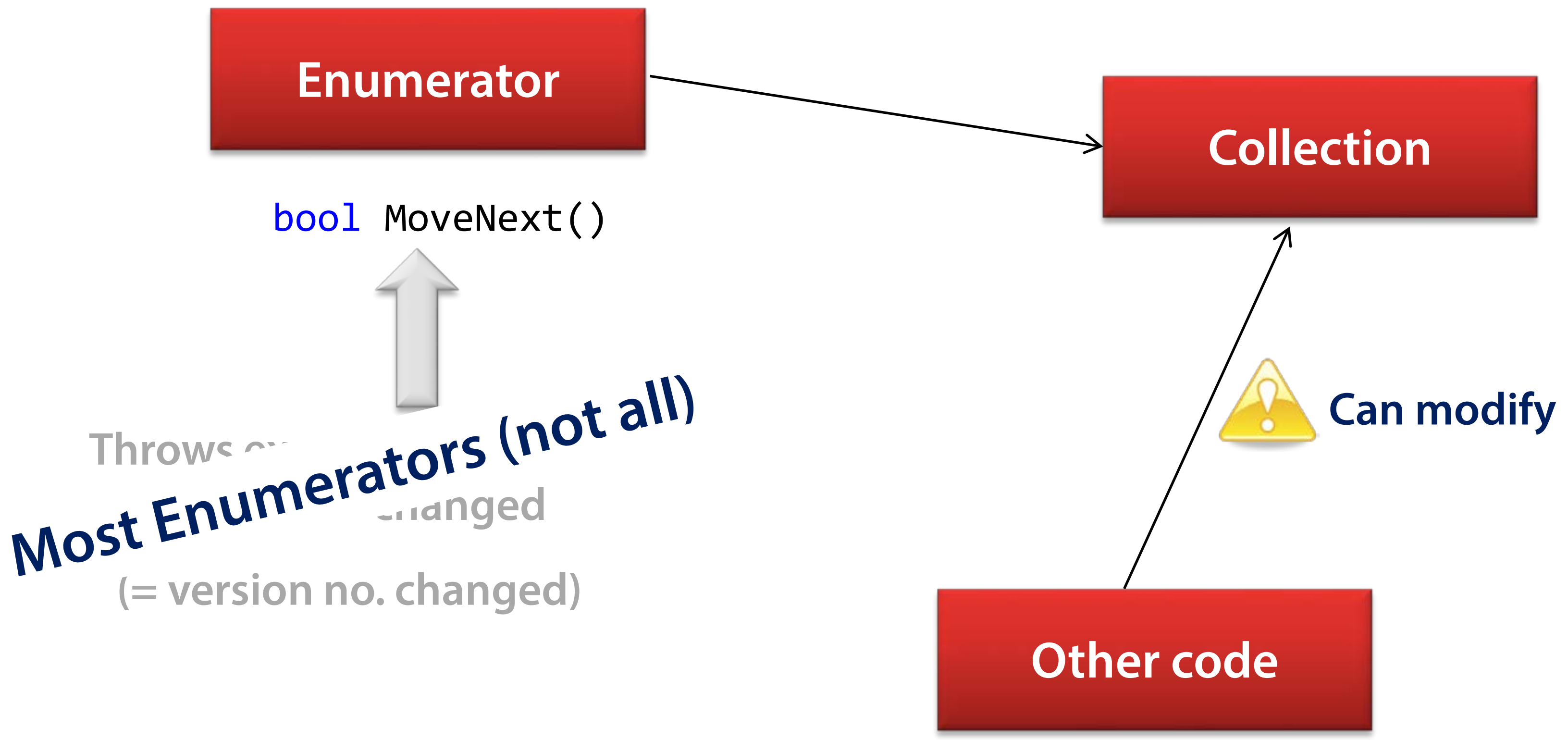


```
bool MoveNext()
```

Throws exception if collection has changed

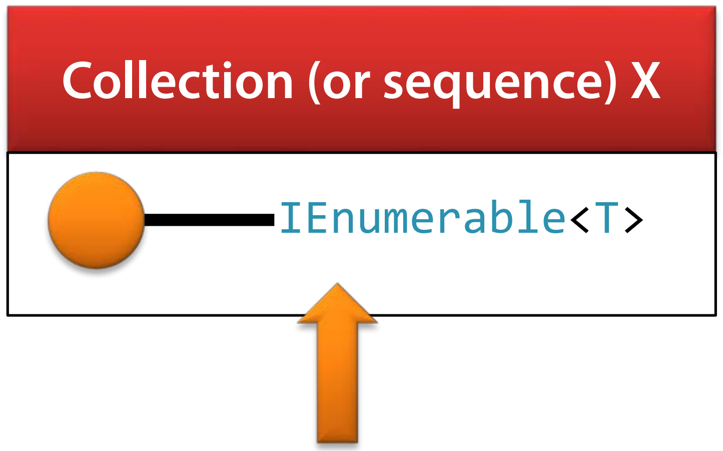
(= version no. changed)

Enumerating a Collection that Changes



Code Demo

Implementing `IEnumerable<T>`



How do you implement this?

1. Write an enumerator for X

2. Have X. GetEnumerator() return a new enumerator instance

Implementing `IEnumerable<T>`

The C# compiler can do all this for you!

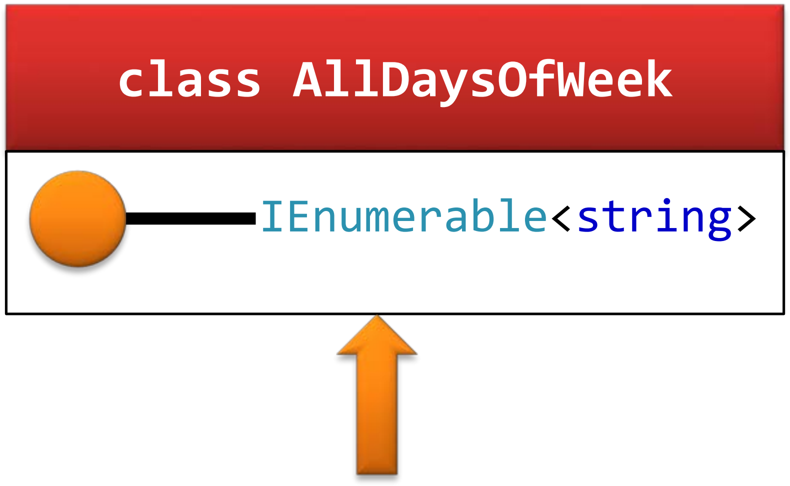
- You just tell the compiler the values to be returned

1. Write an enumerator for X

2. Have X. GetEnumerator() return a new enumerator instance

Implementing `IEnumerable<T>`

Example:



Enumerating returns
`"Monday"`, `"Tuesday"`, etc.

Summary - Enumerators

- **`MoveNext()` and `Current` used to iterate a collection**
  - `foreach` loop hides details of this
- **Enumerator must be separate from collection**
  - This architecture allows multiple clients
- **Exception if collection modified while enumerating**
- **`yield return` makes it easy to write enumerators**
  - Compiler will do most of the work
- **Enumerator interfaces are covariant**