562.613 APPLIED DATA STRUCTURES

Lecture 04

Dr. M. G. Abbas Malik

muhammad.malik@manukau.ac.nz

Lecture 04

- Indexer

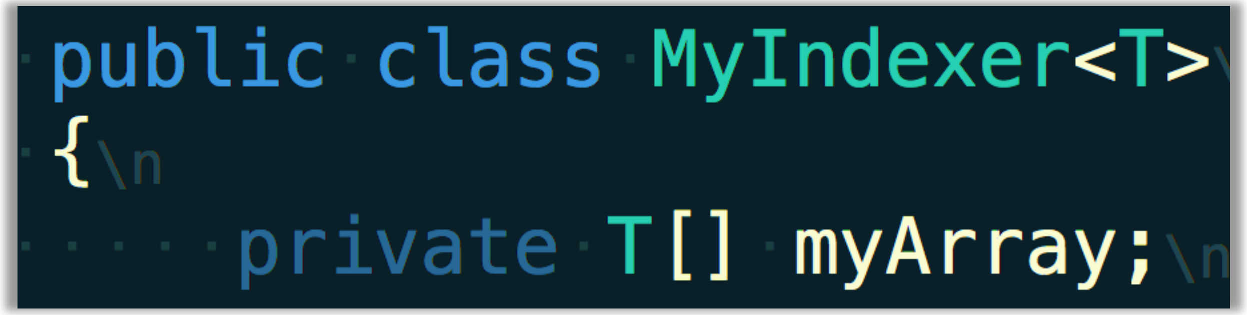
Generic Collections

- List<T>
- Stack<T>
- Queue<T>
- SortedList<T>
- HashSet<T>
- Dictionary

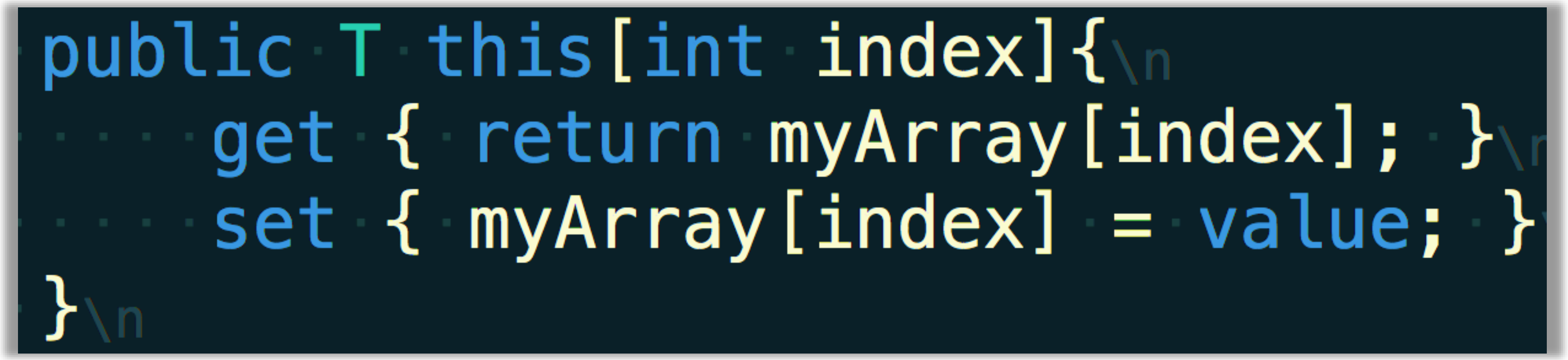
Indexer

- Allow instances of a **class** or **struct** to be **indexed** just like arrays
- Indexed value can be set or retrieved without explicitly specifying a type or instance member
- Same as **property** except that it defined with **this** keyword with **square bracket** and **parameters**

Indexer

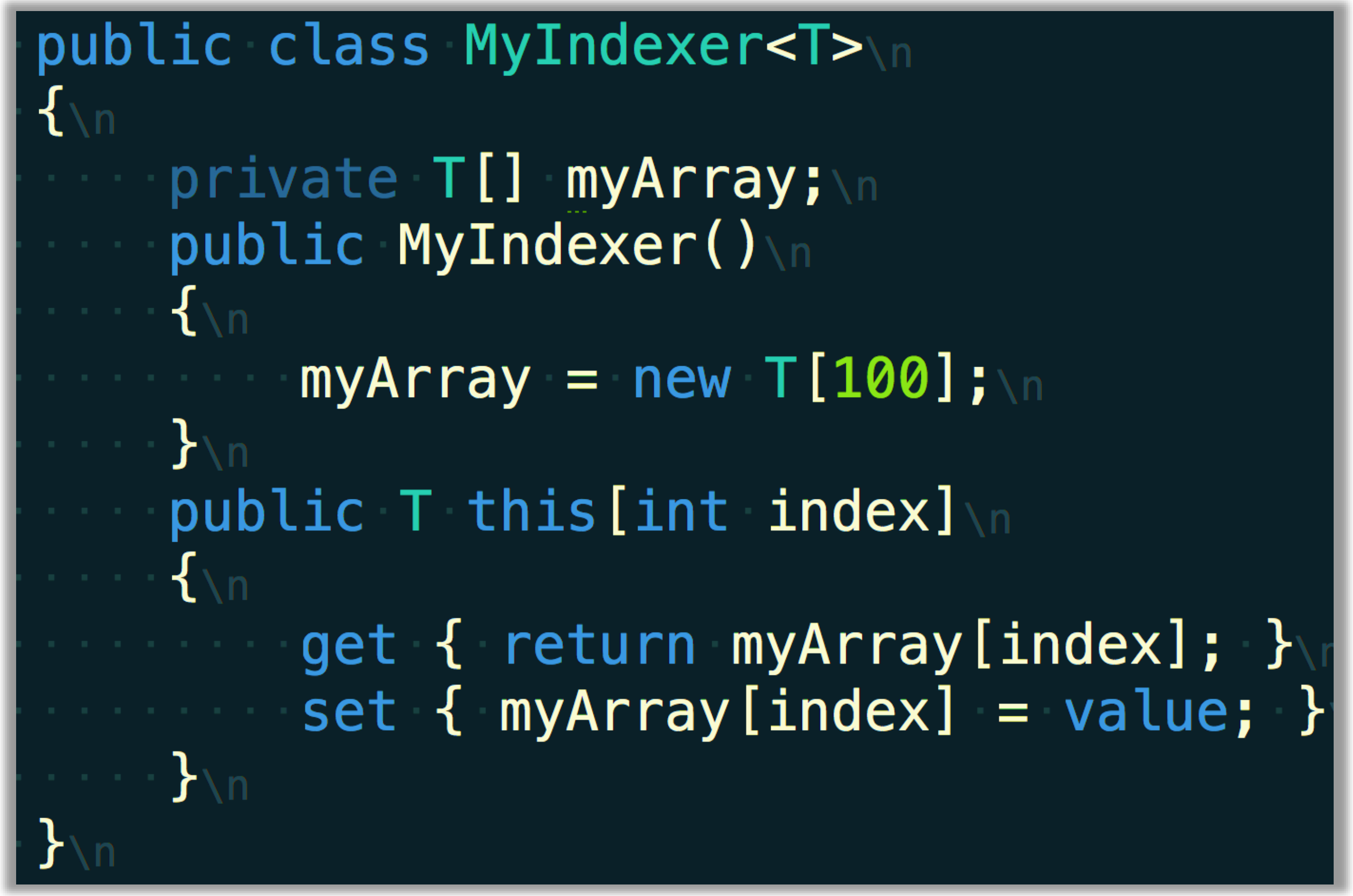


```
public class MyIndexer<T>
{
    private T[] myArray;
```



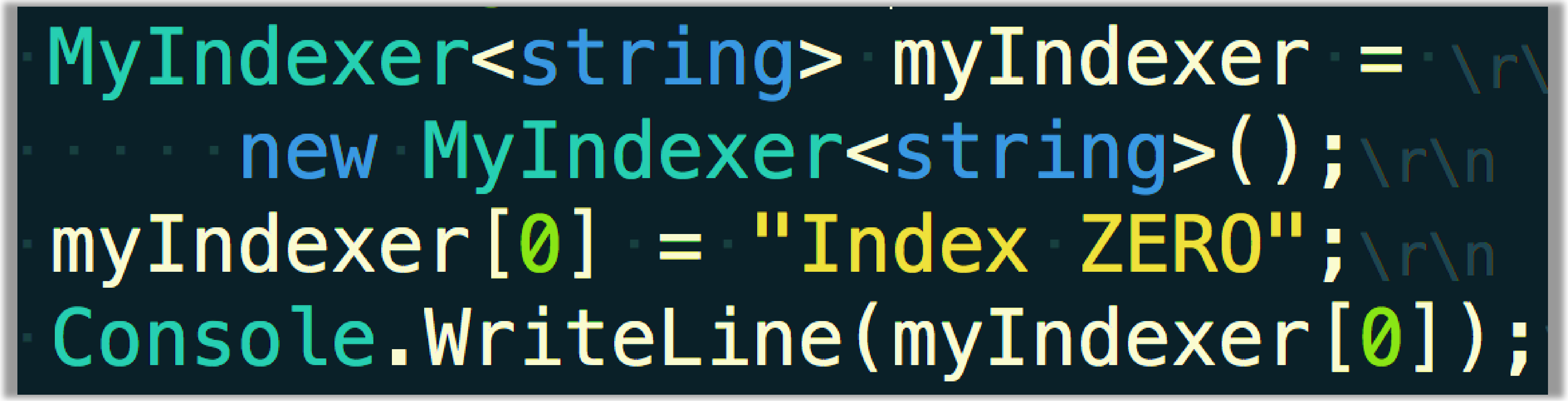
```
public T this[int index]{
    get { return myArray[index]; }
    set { myArray[index] = value; }
}
```

Indexer



```
public class MyIndexer<T>
{
    private T[] myArray;
    public MyIndexer()
    {
        myArray = new T[100];
    }
    public T this[int index]
    {
        get { return myArray[index]; }
        set { myArray[index] = value; }
    }
}
```

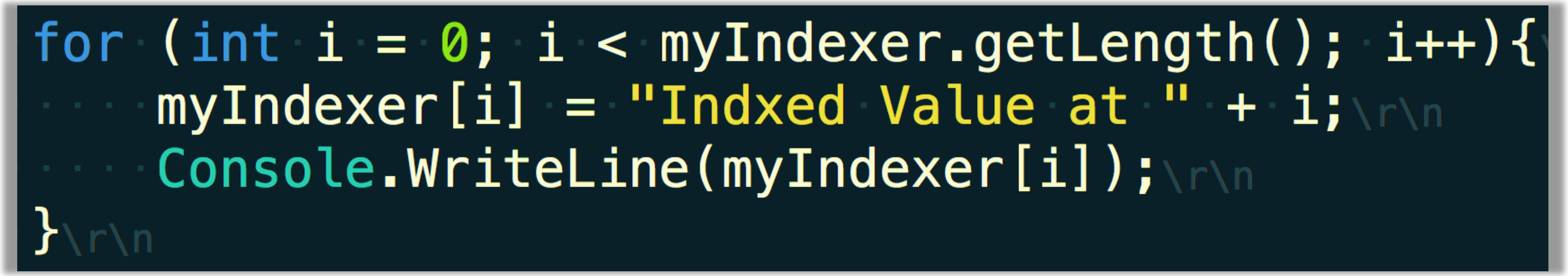
Indexer



```
MyIndexer<string> myIndexer =
    new MyIndexer<string>();
myIndexer[0] = "Index ZERO";
Console.WriteLine(myIndexer[0]);
```

Indexer

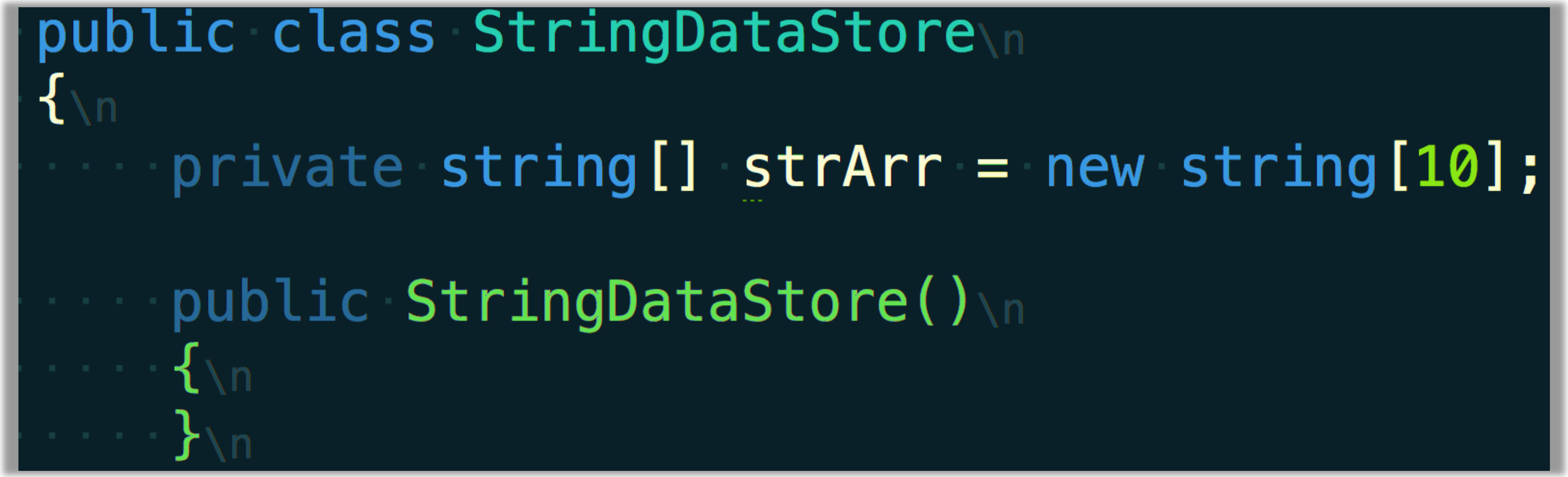
- Making Loops on the Indexer to Access all the elements in the class



```
for (int i = 0; i < myIndexer.getLength(); i++){
    myIndexer[i] = "Indxed Value at " + i;
    Console.WriteLine(myIndexer[i]);
}
```

Indexer

- Override Indexer

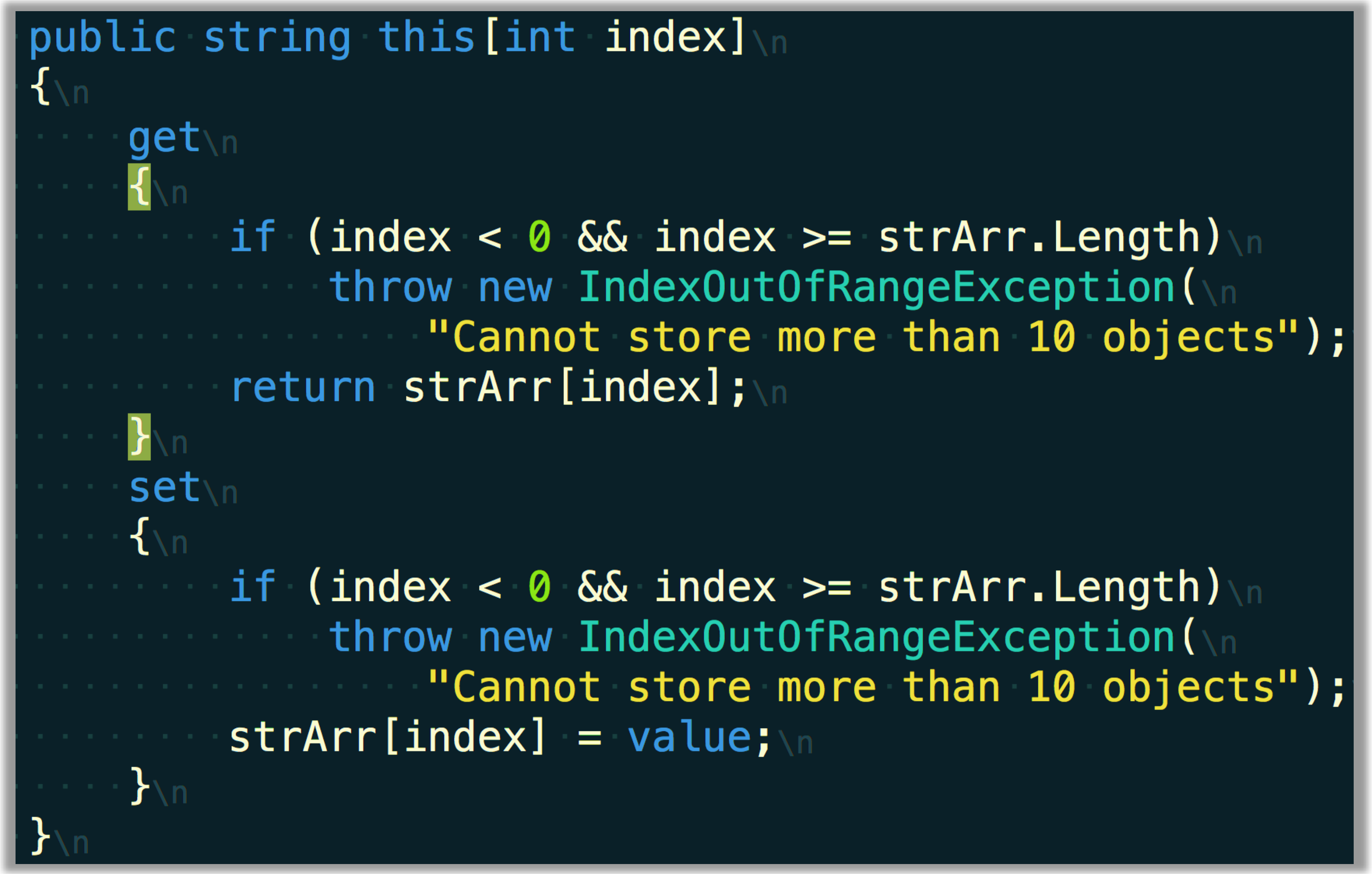


```
public class StringDataStore
{
    private string[] strArr = new string[10];

    public StringDataStore()
    {
    }
```

Indexer

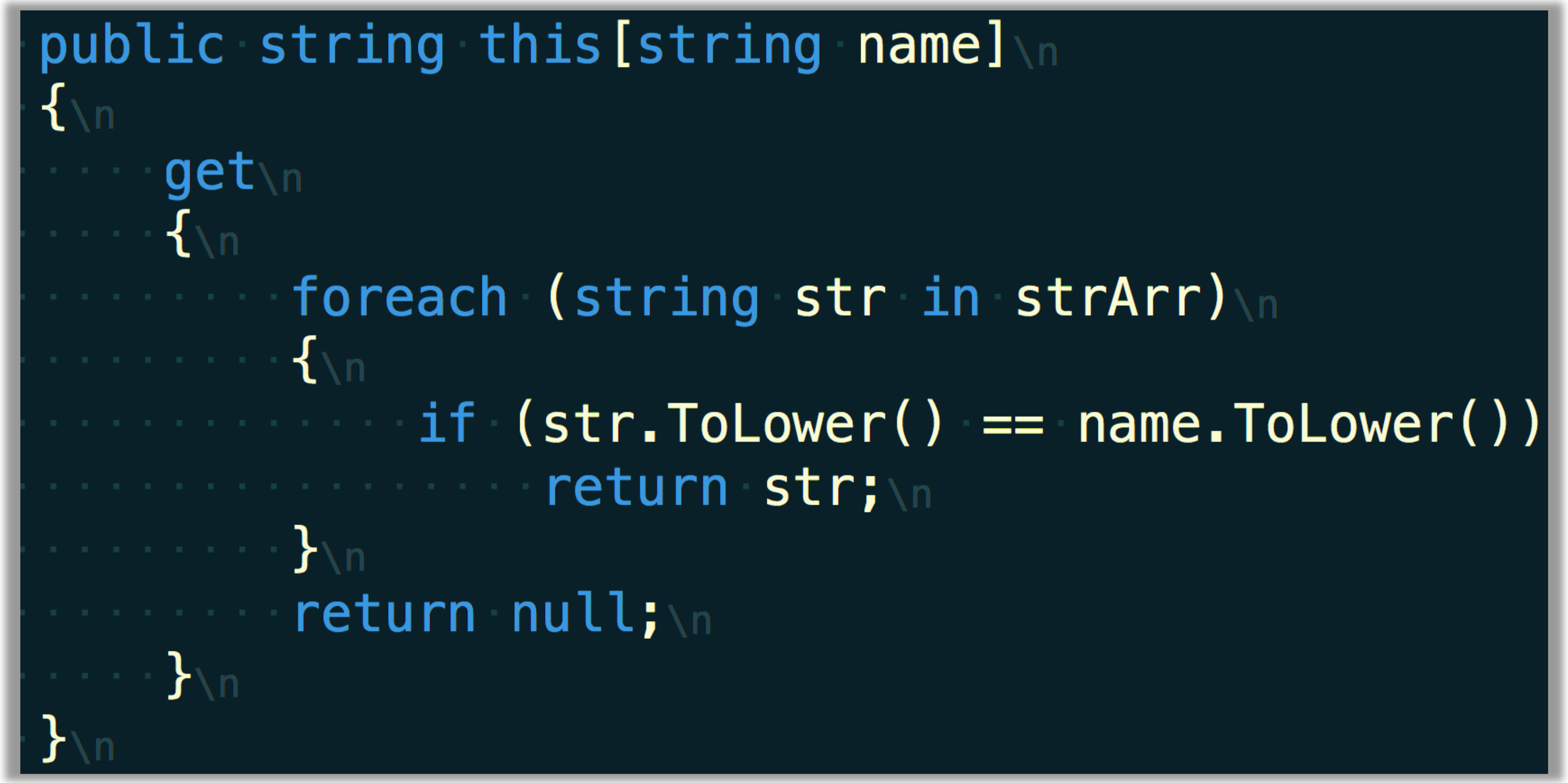
- 1st Indexer



```
public string this[int index]
{
    get
    {
        if (index < 0 && index >= strArr.Length)
            throw new IndexOutOfRangeException(
                "Cannot store more than 10 objects");
        return strArr[index];
    }
    set
    {
        if (index < 0 && index >= strArr.Length)
            throw new IndexOutOfRangeException(
                "Cannot store more than 10 objects");
        strArr[index] = value;
    }
}
```

Indexer

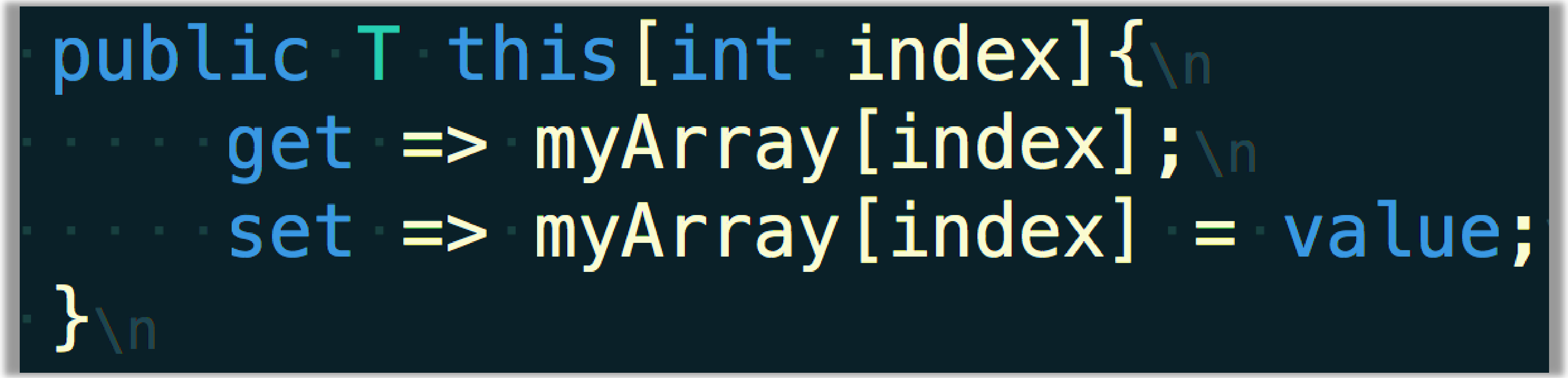
- 2nd Indexer



```
public string this[string name]
{
    get
    {
        foreach (string str in strArr)
        {
            if (str.ToLower() == name.ToLower())
                return str;
        }
        return null;
    }
}
```

Expression-Bodies

- => Lambda Arrow
- Started in C# 6 - only with Methods
- From C# 7, we can use them with properties

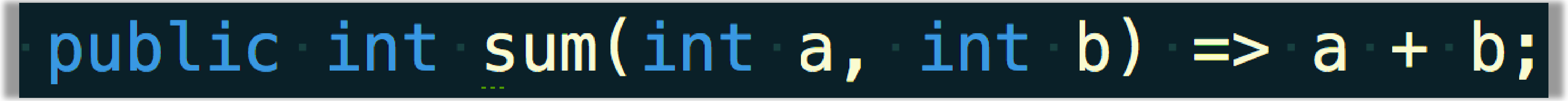


```
public T this[int index]{
    get => myArray[index];
    set => myArray[index] = value;
}
```

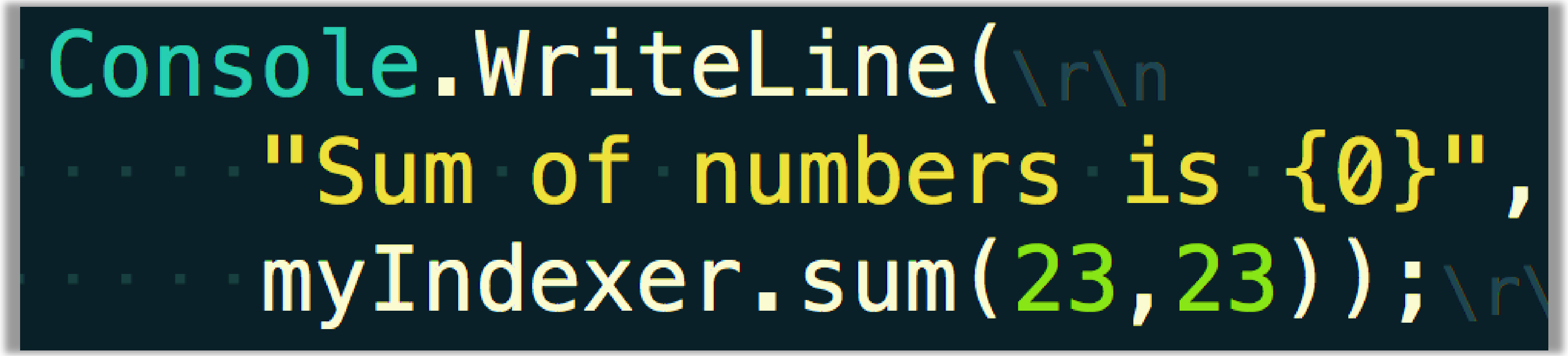
Expression-Bodies

Let us code

- => Lambda Arrow



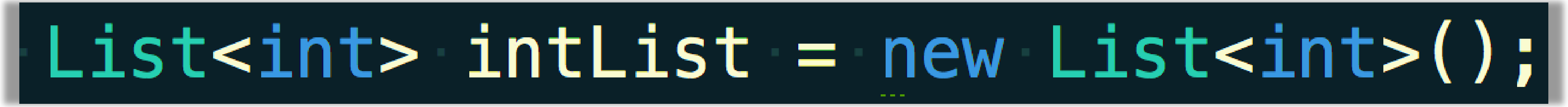
```
public int sum(int a, int b) => a + b;
```



```
Console.WriteLine(\r\n
    "Sum of numbers is {0}",
    myIndexer.sum(23,23));\r\
```

List<T> - Generic Collection

- Same as ArrayList except that **List<T>** is Generic
- Resizes automatically as it grows
- Accessed by index

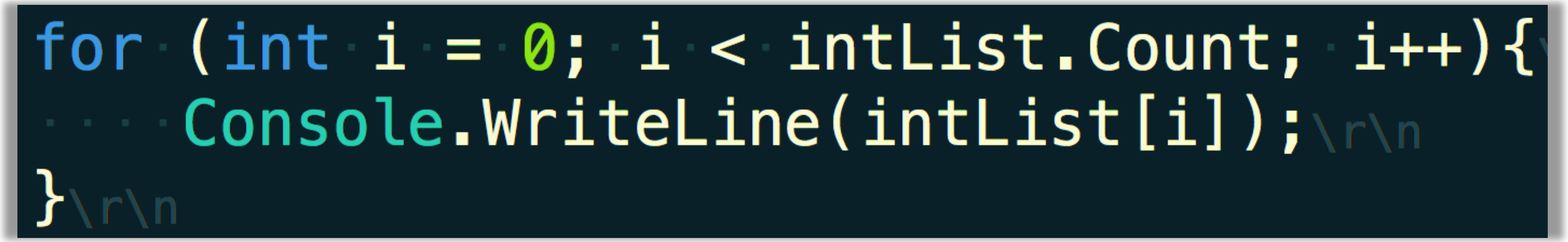


```
List<int> intList = new List<int>();
```

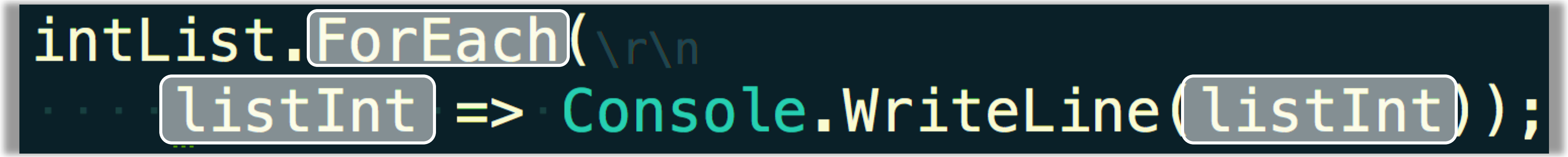
List<T> - Generic Collection

Property Methods	Usage
Items	Gets or sets the element at the specified index
Count	Returns the total number of elements exists in the List<T>
Add	Adds an element at the end of a List<T>
Clear	Removes all the elements from a List<T>
Contains	Checks whether the specified element exists or not in a List<T>
Insert	Inserts an element at the specified index in a List<T>
Remove	Removes the first occurrence of the specified element
RemoteAt	Removes the element at the specified index

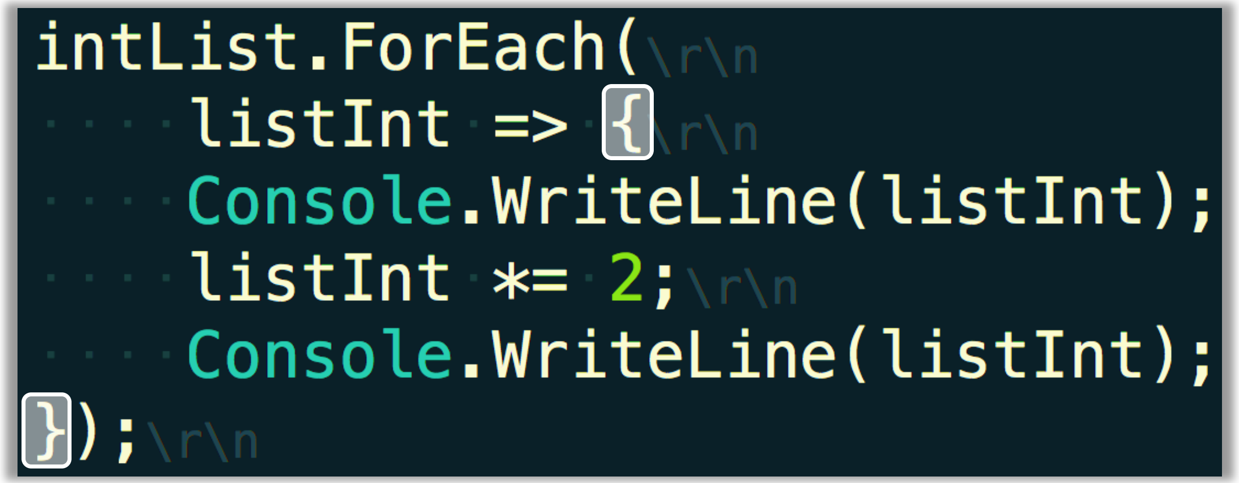
List<T> - Access List Elements



```
for (int i = 0; i < intList.Count; i++){
    Console.WriteLine(intList[i]);
}
```

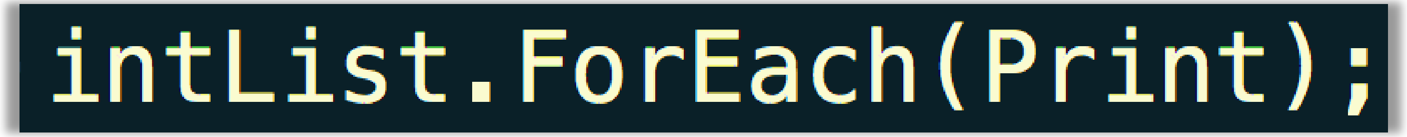


```
intList.ForEach(
    listInt => Console.WriteLine(listInt));
```

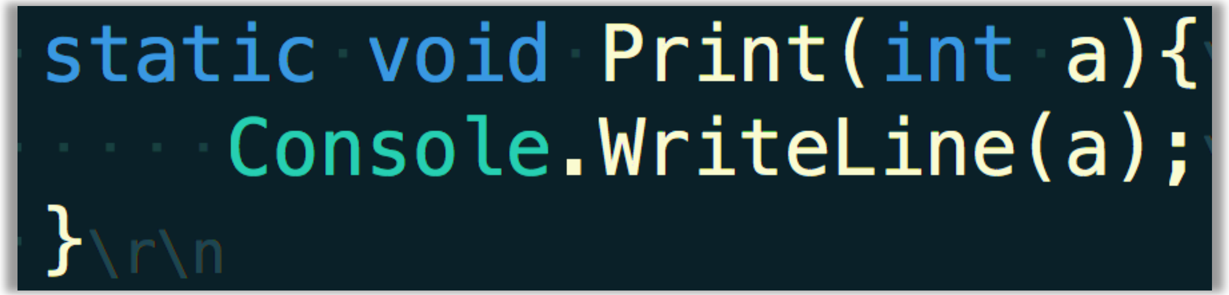


```
intList.ForEach(
    listInt => {
    Console.WriteLine(listInt);
    listInt *= 2;
    Console.WriteLine(listInt);
});
```

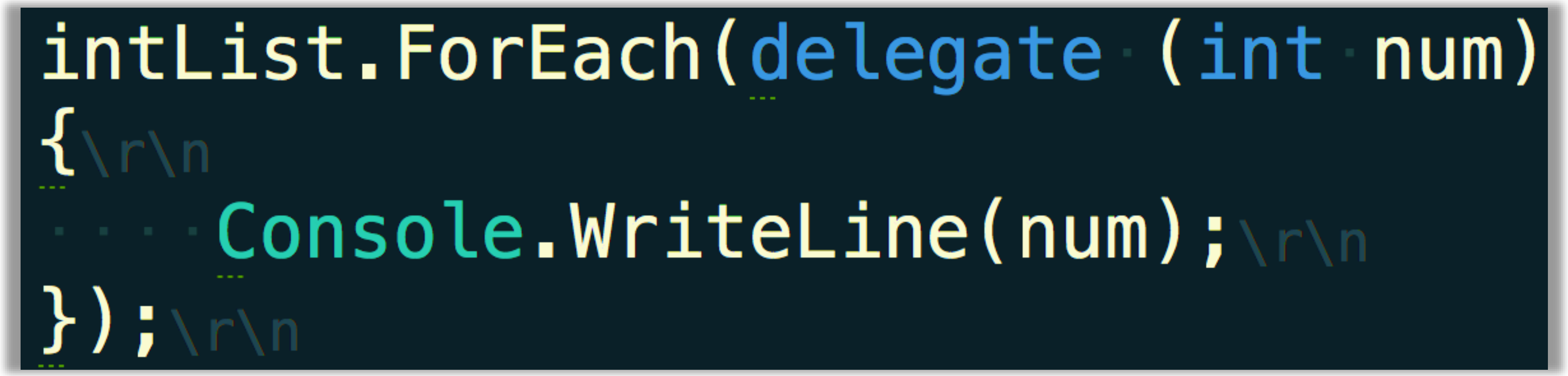
List<T> - Access List Elements



```
intList.ForEach(Print);
```



```
static void Print(int a){
    Console.WriteLine(a);
}\r\n
```



```
intList.ForEach(delegate (int num)
{\r\n
    Console.WriteLine(num);\r\n
});\r\n
```

Let us code

SortedList<Tkey, Tvalue>

- SortedList - covered in previous lecture
- Stores **key-value pairs** in the **ascending order** of key by default
- **Key** cannot be null
- **Value** can be null

SortedList<Tkey, Tvalue>

Property	Description
Capacity	Gets or sets the number of elements that the list contain
Count	Gets the number of elements actually in the list
Keys	Get list of keys of SortedList<TKey,TValue>
Values	Get list of values in SortedList<TKey,TValue>

SortedList<Tkey, Tvalue> Let us code

Method	Description
Add	Add one element
Remove	Remove element with specific key
RemoveAt	Remove element at a specified index
ContainsKey	Check whether specified key exist in SortedList<Tkey, Tvalue>
ContainsValue	Check whether specified key exist in SortedList<Tkey, Tvalue>
Clear	Clear all elements
IndexOfkey	Return the index of specified key
IndexOfValue	Return the index of specified value

Dictionary<Tkey, Tvalue>

- Stores **key-value pairs**
- **Keys** cannot be **duplicate** or null
- **Values** can be **duplicated** or set as null

Dictionary<Tkey, Tvalue> Let us code

Property	Description
Count	Gets the number of elements actually in Dictionary
Keys	Get list of keys of Dictionary<TKey,TValue>
Values	Get list of values in Dictionary<TKey,TValue>
Item	Gets or sets the element with the specified key in the Dictionary<TKey,TValue>.

SortedDictionary<TKey,TValue>

- A collection of key/value pairs that are sorted on the key

SortedList Vs. SortedDictionary

Collection	Indexed Lookup	Keyed Lookup	Value Lookup	Addition	Removal	Memory
SortedList	$O(1)$	$O(\log n)$	$O(n)$	$O(n)$	$O(n)$	Lesser
SortedDictionary	N.A.	$O(\log n)$	$O(n)$	$O(\log n)$	$O(\log n)$	Greater

Some Other Generic Collections

- Stack<T>
- Queue<T>

Reference and Reading Material

- Indexer: Link, Link
- Generics Collections: Link, Link, Link
- List<T>: Link, Link, Link
- SortedList<Tkey, TValue>: Link, Link
- Dictionary: Link, Link
- SortedDictionary: Link