IENUMRABLE

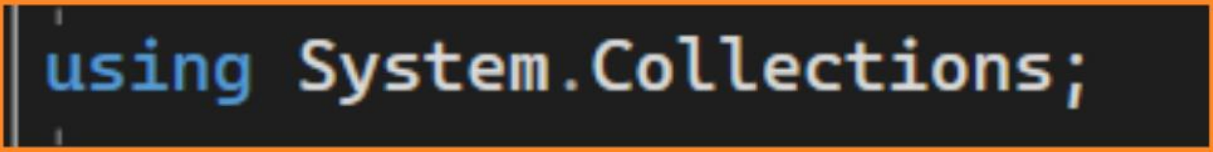
IN C# .NET

Mohamed Fadel | .Net Core Developer

What is IEnumerable?

- In C#, IEnumerable is an interface that defines a standard way for classes to represent a sequence of objects that can be iterated over

WHAT IS THE NAME OF THE NAMESPACE THAT INCLUDE IENUMERABLE?



```
using System.Collections;
```

Mohamed Fadel | .Net Core Developer

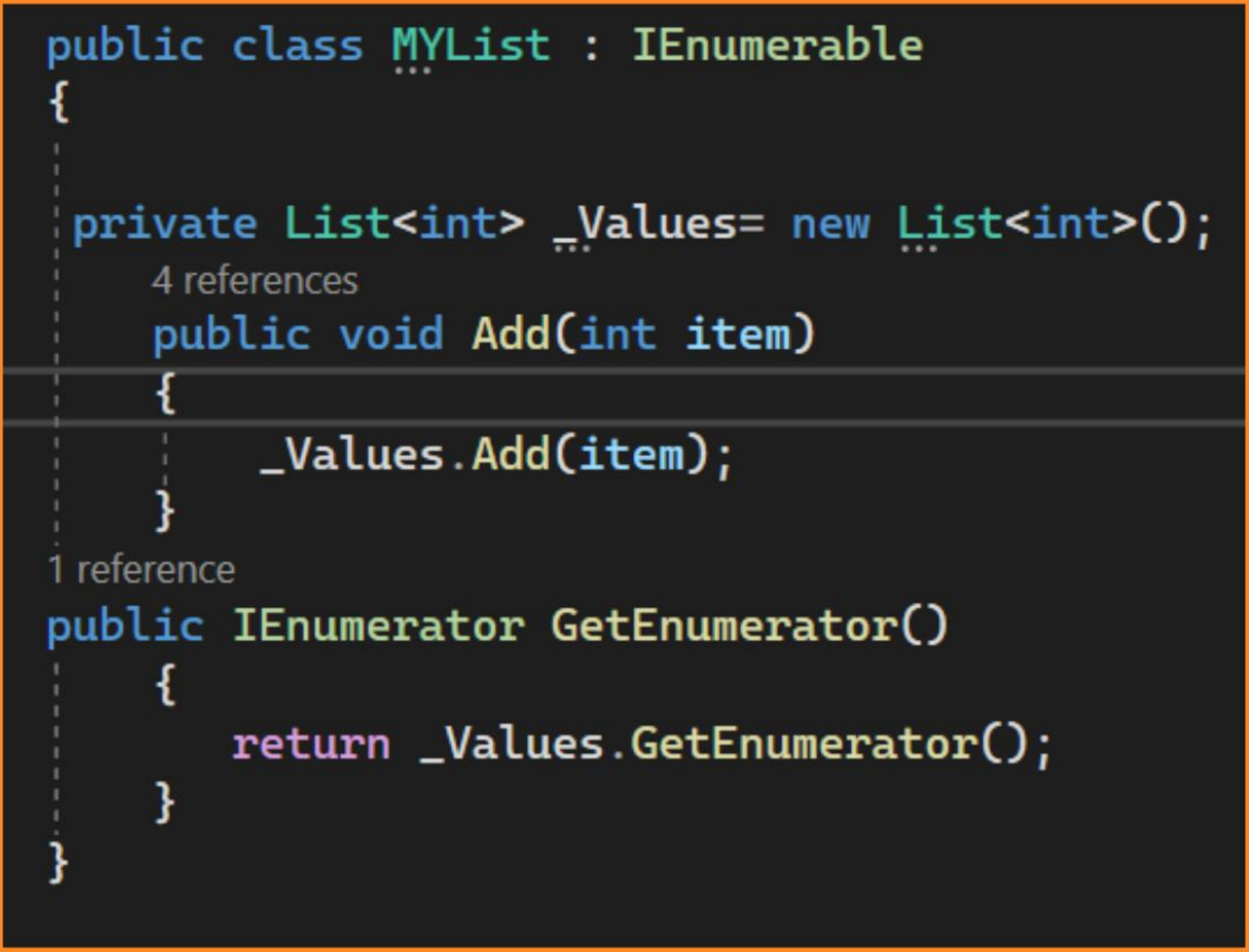
IEnumerable contains a GetEnumerator method that returns an IEnumerator interface.

You may now be wondering what is Methods inside IEnumerator interface ?

METHODS IN IENUMERATAOR INTERFACE :

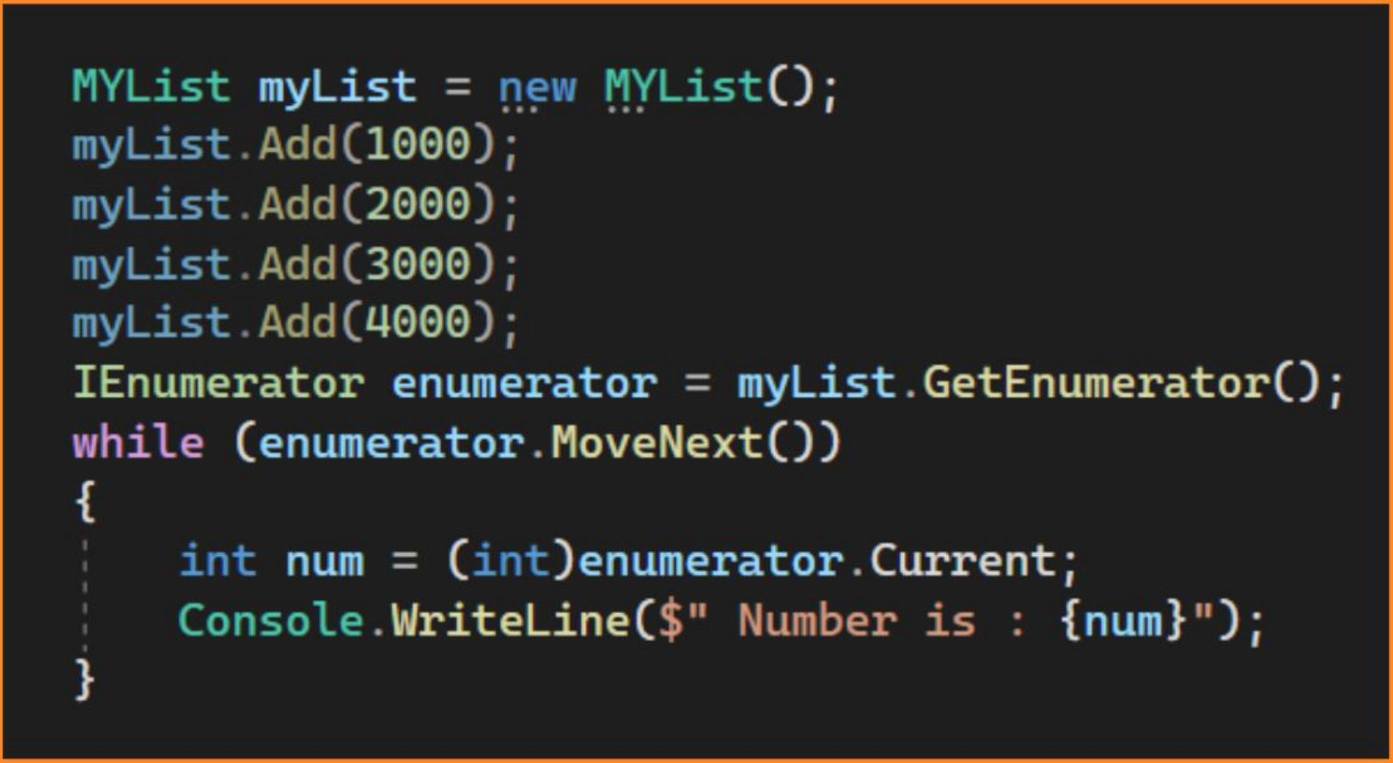
1. **Current Property : get current elements from the collection**
2. **MoveNext() :Sets the enumerator to the next element of the collection**
3. **Reset() : Sets the enumerator to its initial position, which is before the first element in the collection.**

- **LET US IMPLEMENT THE IENUMERABLE INTERFACE IN A CLASS AS:**



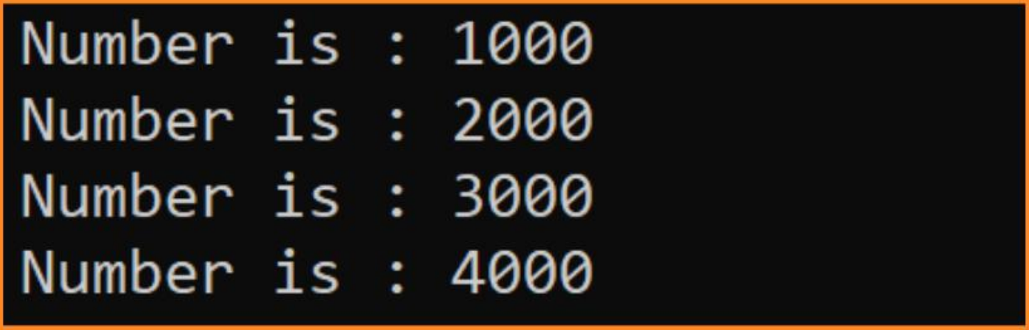
```
public class MYList : IEnumerable
{

 private List<int> _Values= new List<int>();
     4 references
     public void Add(int item)
     {
         _Values.Add(item);
     }
1 reference
public IEnumerator GetEnumerator()
     {
         return _Values.GetEnumerator();
     }
}
```

NOW USE THE MYLIST CLASS TO ITERATE FOR ITEMS



```csharp
MYList myList = new MYList();
myList.Add(1000);
myList.Add(2000);
myList.Add(3000);
myList.Add(4000);
IEnumerator enumerator = myList.GetEnumerator();
while (enumerator.MoveNext())
{
    int num = (int)enumerator.Current;
    Console.WriteLine($" Number is : {num}");
}
```

. **OUR OUTPUT WILL BY LIKE BELOW :**

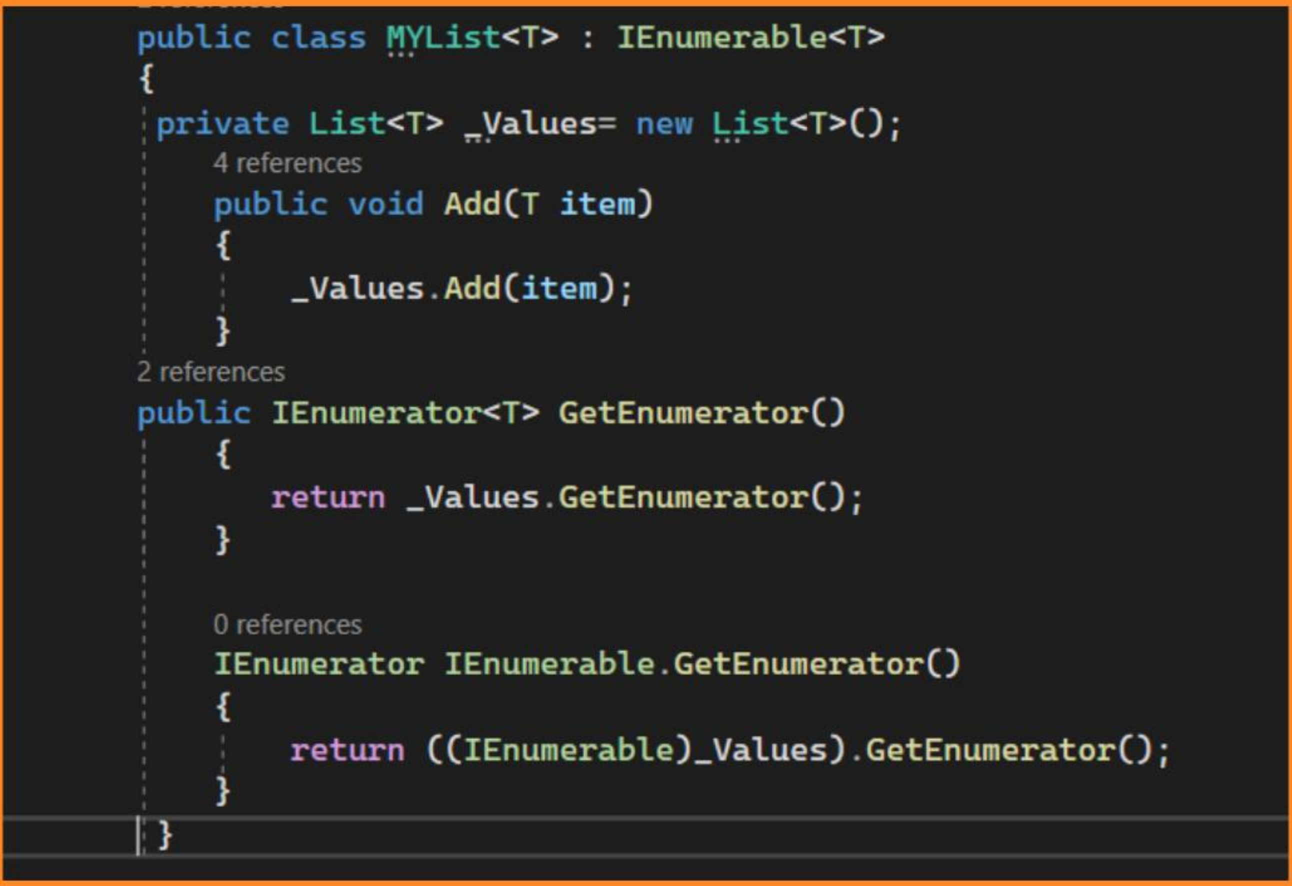


```
Number is : 1000
Number is : 2000
Number is : 3000
Number is : 4000
```

Mohamed Fadel | .Net Core Developer

IENUMRABLE<T>

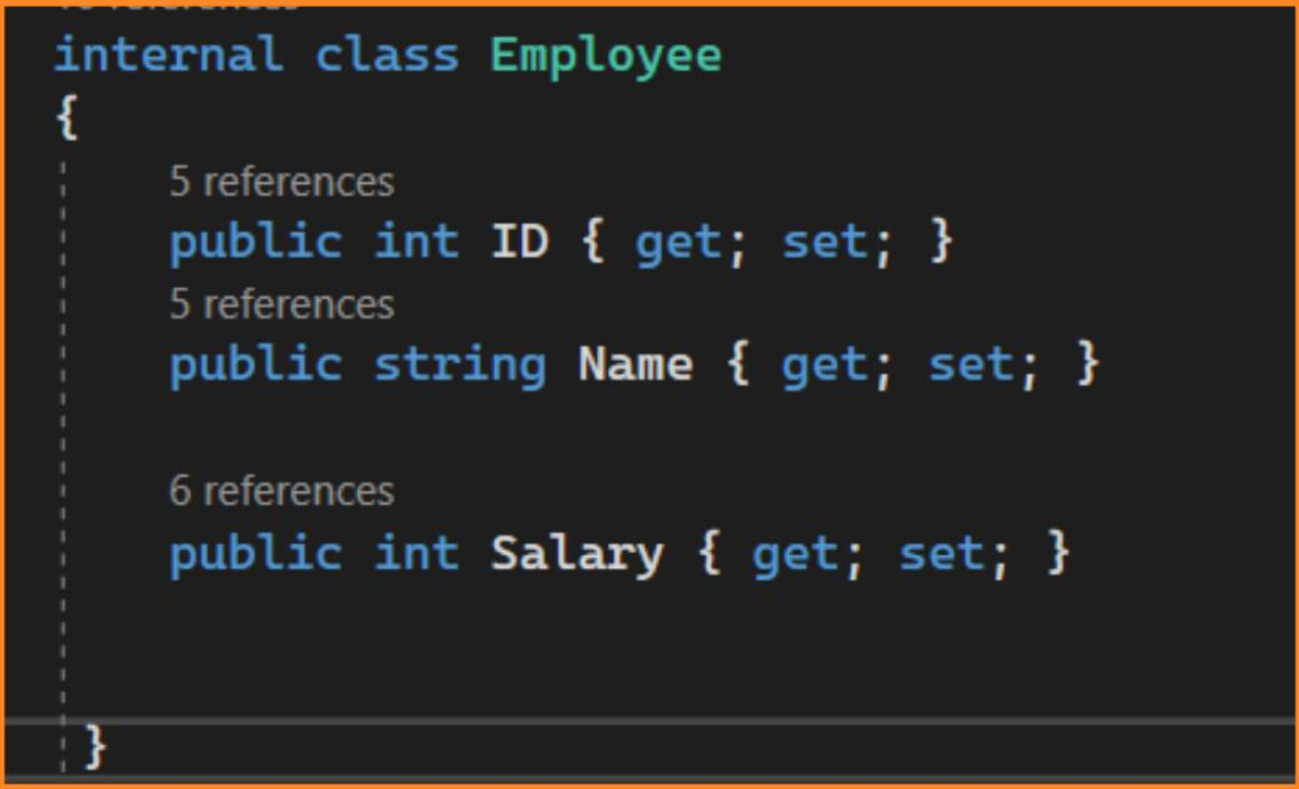
IEnumerable<T>: This generic interface extends IEnumerable and is part of the System.Collections.Generic namespace. It introduces the same GetEnumerator() method but is typed to the collection's element type T.



```
public class MYList<T> : IEnumerable<T>
{
 private List<T> _Values= new List<T>();
    4 references
    public void Add(T item)
    {
        _Values.Add(item);
    }
2 references
public IEnumerator<T> GetEnumerator()
    {
       return _Values.GetEnumerator();
    }

    0 references
    IEnumerator IEnumerable.GetEnumerator()
    {
        return ((IEnumerable)_Values).GetEnumerator();
    }
}
```

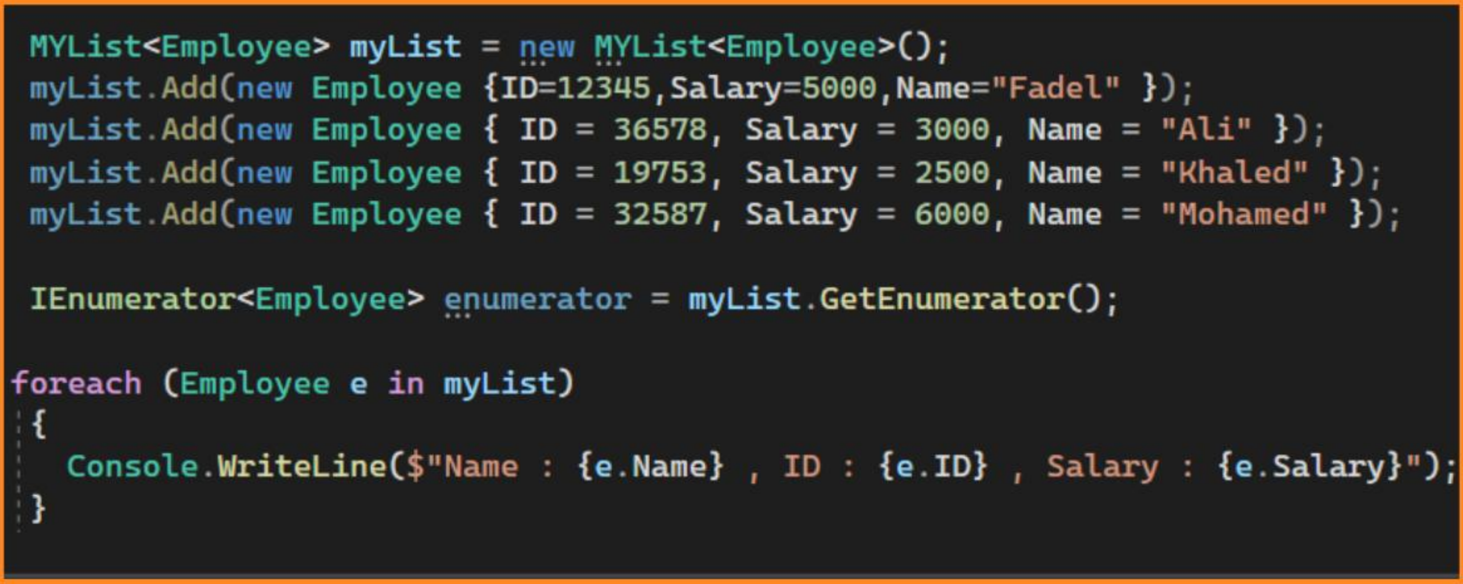
Mohamed Fadel | .Net Core Developer

WE WILL CREATE NEW CLASS WITH NAME EMPLOYEE TO USE ITS PROPERTIES



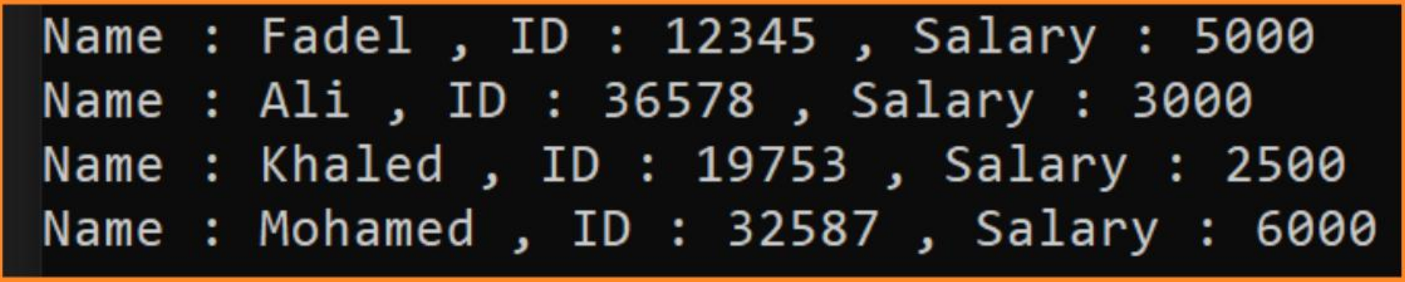
```
internal class Employee
{
        5 references
        public int ID { get; set; }
        5 references
        public string Name { get; set; }

        6 references
        public int Salary { get; set; }


}
```

NOW WE WILL ITERATE FOR THIS COLLECTION " EMPLOYEE "



```csharp
MYList<Employee> myList = new MYList<Employee>();
myList.Add(new Employee {ID=12345,Salary=5000,Name="Fadel" });
myList.Add(new Employee { ID = 36578, Salary = 3000, Name = "Ali" });
myList.Add(new Employee { ID = 19753, Salary = 2500, Name = "Khaled" });
myList.Add(new Employee { ID = 32587, Salary = 6000, Name = "Mohamed" });

IEnumerator<Employee> enumerator = myList.GetEnumerator();

foreach (Employee e in myList)
{
    Console.WriteLine($"Name : {e.Name} , ID : {e.ID} , Salary : {e.Salary}");
}
```

. **OUR OUTPUT WILL BY LIKE BELOW :**



```
Name : Fadel , ID : 12345 , Salary : 5000
Name : Ali , ID : 36578 , Salary : 3000
Name : Khaled , ID : 19753 , Salary : 2500
Name : Mohamed , ID : 32587 , Salary : 6000
```

I HOPE IT WAS HELPFUL

Mohamed Fadel | .Net Core Developer