# Iterators and Comparators in C#

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://about.softuni.bg/

## Questions

sli.do

#csharp-advanced

# Table of Contents

# Iterators in C#

## `IEnumerable<T>` and `IEnumerator<T>`

## Enumerable Collections and "foreach"

- In C# **enumerable collections** and types can be traversed through the "**foreach**" loop:

```
List<int> nums = new List<int>() { 10, 20, 30 };
// Lists in .NET are enumerable → "foreach"
is available
foreach (int num in nums)
  Console.WriteLine(num);
```

- Internally, **foreach** works though **iterators**:
  - The collection should implement **IEnumerable<T>**

# IEnumerable<T>

- IEnumerable<T> == the **root interface** for .NET types, which support **iteration** over elements
  - Defines a single method **GetEnumerator()**, which returns an **IEnumerator<T>**
  - **IEnumerator<T>** allows passing through the elements
- Types, which implement **IEnumerable<T>** can be used in a **foreach** loop traversals

```
IEnumerable<int> nums = new int[] {10, 20, 30};
foreach (int num in nums)
  Console.WriteLine(num);
```

## IEnumerable<T>: Definition

```
public interface IEnumerable<T> : IEnumerable
{
  IEnumerator<T> GetEnumerator();
}

// Non-generic version
// (compatible with the legacy .NET 1.1)
public interface IEnumerable
{
  IEnumerator GetEnumerator();
}
```

# IEnumerator<T>

- IEnumerator<T> implements a **sequential**, **forward-only iteration** over a collection
  - Current – returns the current element of the enumerator
  - MoveNext() – goes to the next element of the collection
  - Reset() – goes to the initial (start) position

```
public interface IEnumerator
{
  object Current { get; }
  bool MoveNext();
  void Reset();
}
public interface IEnumerator<T>
  : IEnumerator
{
  T Current { get; }
  bool MoveNext();
  void Reset();
}
```

# The "params" Keyword in C#

- Methods can take a **variable number** of arguments:

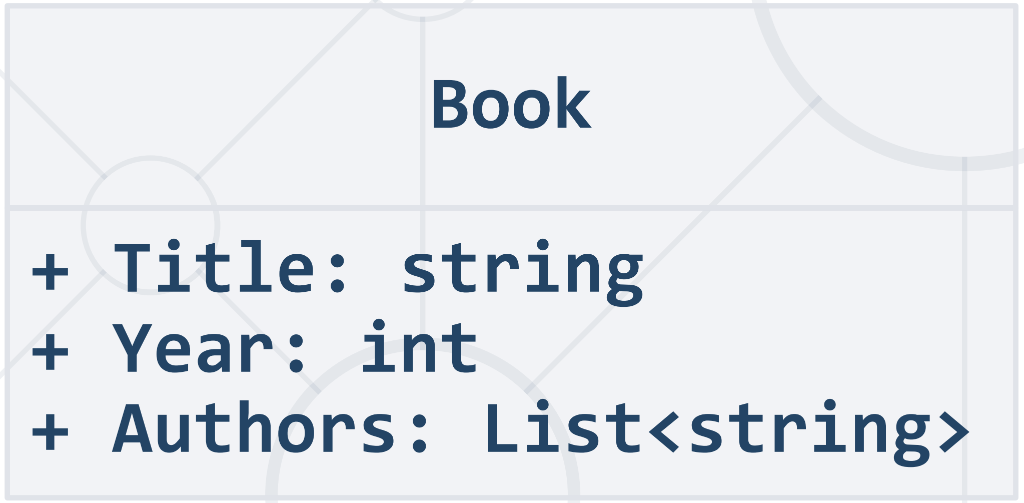
```
PrintNames("Steve", "Teddy");
PrintNames("Peter", "Sam", "Jay", "Chriss");

void PrintNames(params string[] names)
{
  foreach(var name in names)
    Console.WriteLine(name);
}
```

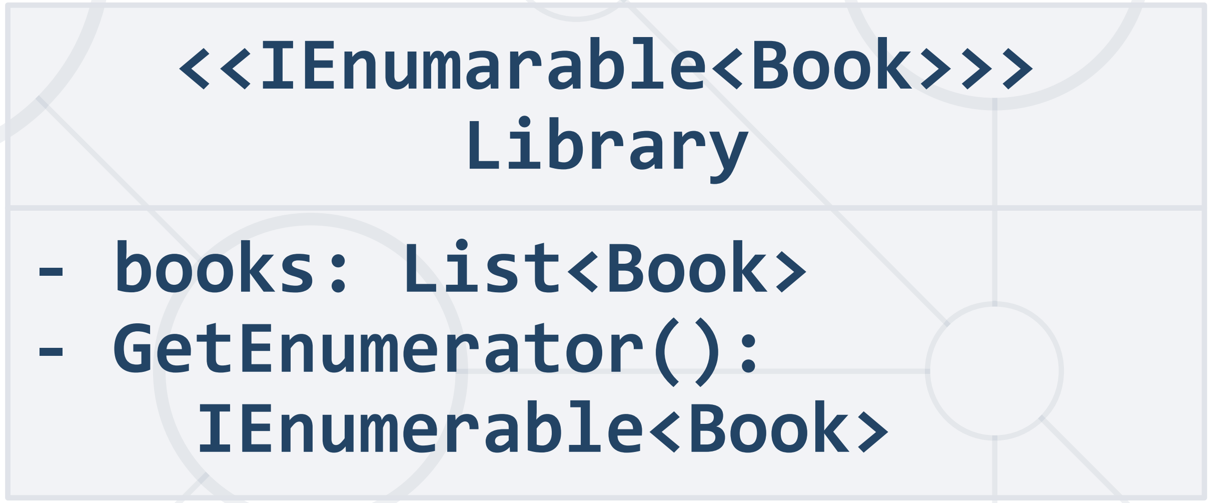
- Only **one** **params** declaration per method; should be put **last**

# Problem: Library Iterator

- Create a class **Library** to store a **collection of books** and implement the **IEnumerable<Book>** interface



```
Book
+ Title: string
+ Year: int
+ Authors: List<string>
```



```
<<IEnumarable<Book>>>
Library
- books: List<Book>
- GetEnumerator():
    IEnumerable<Book>
```

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1489#0

## Problem: Library Iterator

- Inside the **`Library`** class create nested class **`LibraryIterator`**, which implements **`IEnumerator<Book>`**

```
<<IEnumerator<Book>>>
   LibraryIterator
-currentIndex: int
-books: List<Book>
+Current: Book
+Reset(): void
+MoveNext(): bool
+Dispose(): void
```

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1489#1

## Solution: Library Iterator

```
public class Book {
  public Book(string title, int year, params string[] authors) {
    this.Title = title;
    this.Year = year;
    this.Authors = authors.ToList();
  }
  public string Title { get; private set; }
  public int Year { get; private set; }
  public List<string> Authors { get; private set; }
}
```

# Solution: Library Iterator

```
public class Library : IEnumerable<Book> {
  private List<Book> books;
  public Library(params Book[] books) {
      this.books = new List<Book>(books);
  }
  public IEnumerator<Book> GetEnumerator() {
      return new LibraryIterator(this.books);
  }
  IEnumerator IEnumerable.GetEnumerator()
     => this.GetEnumerator();
}
```

## Solution: Library Iterator

```
private class LibraryIterator : IEnumerator<Book> {
  private readonly List<Book> books;
  private int currentIndex;
  public LibraryIterator(IEnumerable<Book> books) {
    this.books = books;
    this.Reset();
  }
  public void Dispose() {}
  public bool MoveNext() =>
    ++this.currentIndex < this.books.Count;
  public void Reset() => this.currentIndex = -1;
  public Book Current => this.books[this.currentIndex];
  object IEnumerator.Current => this.Current;
}
```

## Yield Return

- The "`yield return`" statement simplifies **`IEnumerator<T>`** implementations:

```
private readonly List<Book> books;
public IEnumerator<Book> GetEnumerator()
{
  for (int i = 0; i < this.books.Count; i++)
    yield return this.books[i];
}
```

- Returns **one element** upon **each** loop cycle

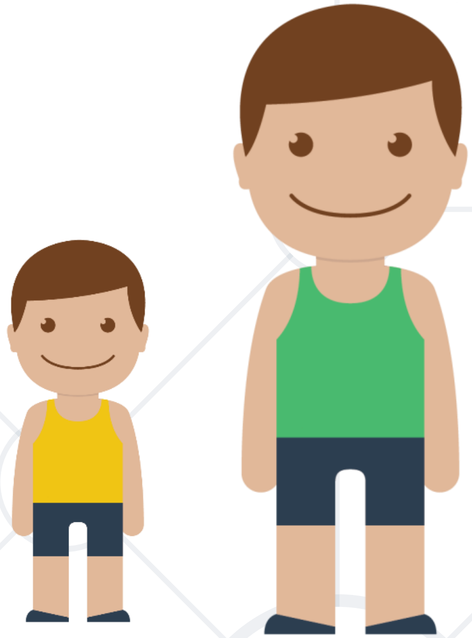
# Comparators

## IComparable<T> and IComparer<T>

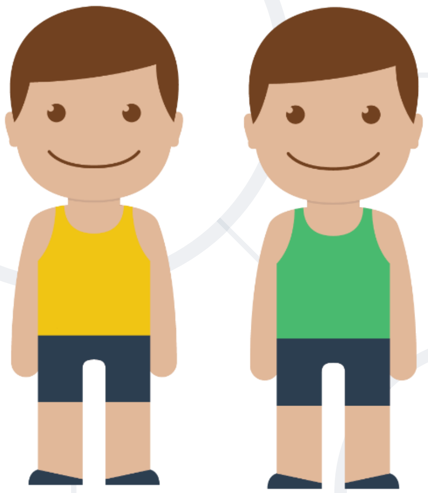
# IComparable<T>

- Reads out as "**I am Comparable**"
- Provides a method of **comparing two objects** of a particular type - **`CompareTo()`**
- Sets a **default sort order** for the particular object type
- **Affects** the original class

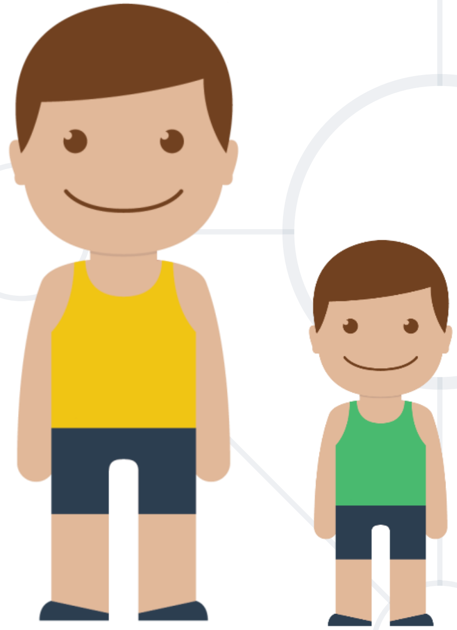
# CompareTo(T) Method Returns



< 0



= 0



> 0

## IComparable<T>: Example

```
class Point : IComparable<Point>
{
  public int X { get; set; }
  public int Y { get; set; }

  public int CompareTo(Point otherPoint)
  {
    if (this.X != otherPoint.X)
      return (this.X - otherPoint.X);
    if (this.Y != otherPoint.Y)
      return (this.Y - otherPoint.Y);
    return 0;
  }
}
```

# Problem: Comparable Book

- Implement the `IComparable<Book>` interface in the existing class Book
  - First sort them in **ascending chronological** order (by year)
  - If two books are published in the **same year**, sort them **alphabetically**
- Override the `ToString()` method in your Book class, so it returns a string in the format:
  - `"{title} - {year}"`
- Change your `Library` class so that **it stores the books** in the **correct** order

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1489#2

## Solution: Comparable Book

```
public class Book : IComparable<Book>
{
    public int CompareTo(Book other)
    {
        int result = this.Year.CompareTo(other.Year);
        if (result == 0)
        {
            result = this.Title.CompareTo(other.Title);
        }
        return result;
    }
}
```

## IComparer<T>

- Reads out as "**I'm a comparer**" or "**I compare**"
- Provides a way to **customize** the **sort order** of a **collection**
- Defines a **method** that a type implements to **compare two objects**
- Does not **affect** original class (it's a **separate** class)

# IComparer<T> - Example

```
class Cat
{
    public string Name { get; set; }
}
```

```
class CatComparer : IComparer<Cat>
{
  public int Compare(Cat x, Cat y)
  {
    return x.Name.CompareTo(y.Name);
  }
}
```

```
IComparer<Cat> comparer = new CatComparer();
var catsByName = new SortedSet(comparer);
```

## Problem: Book Comparer

- Create a class **`BookComparator`**, which implements the **`IComparer<Book>`** interface
- **`BookComparator`** must **compare two** books by:
  - Book title - **alphabetical order**
  - Year of publishing a book - from **the newest to the oldest**
- Modify your **`Library`** class once again to implement the new sorting

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1489#3

# Solution: Book Comparer

```
public class BookComparator : IComparer<Book>
{
  public int Compare(Book x, Book y)
  {
    int result = x.Title.CompareTo(y.Title);
    if (result == 0)
    {
      result = y.Year.CompareTo(x.Year);
    }
    return result;
  }
}
```

## Summary

- Iterators in C#
  - `IEnumerable<T>`
  - `IEnumerator<T>`
  - `yield return`
- Params: variable number of arguments
- Comparators in C#
  - `IComparable<T>`
  - `IComparer<T>`

# Questions?

Software University

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Finance Academy

# SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SOFTWARE GROUP

DXC TECHNOLOGY

createX

Postbank

Решения за твоето утре

SUPER HOSTING .BG

DRAFT KINGS

VIVACOM

INDEAVR

Serving the high achievers

AMBITIONED

PHAR VISION

## Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

## License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg