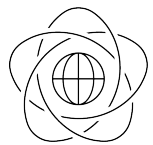
Programming 3 Advanced

Collections, Delegates and Lambda expressions, Extension methods, LINQ

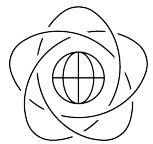
Tomasz Herman

Faculty of Mathematics and Information Science
Warsaw University of Technology

Lecture 5, 4 listopada 2024



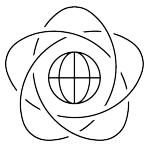
Outline



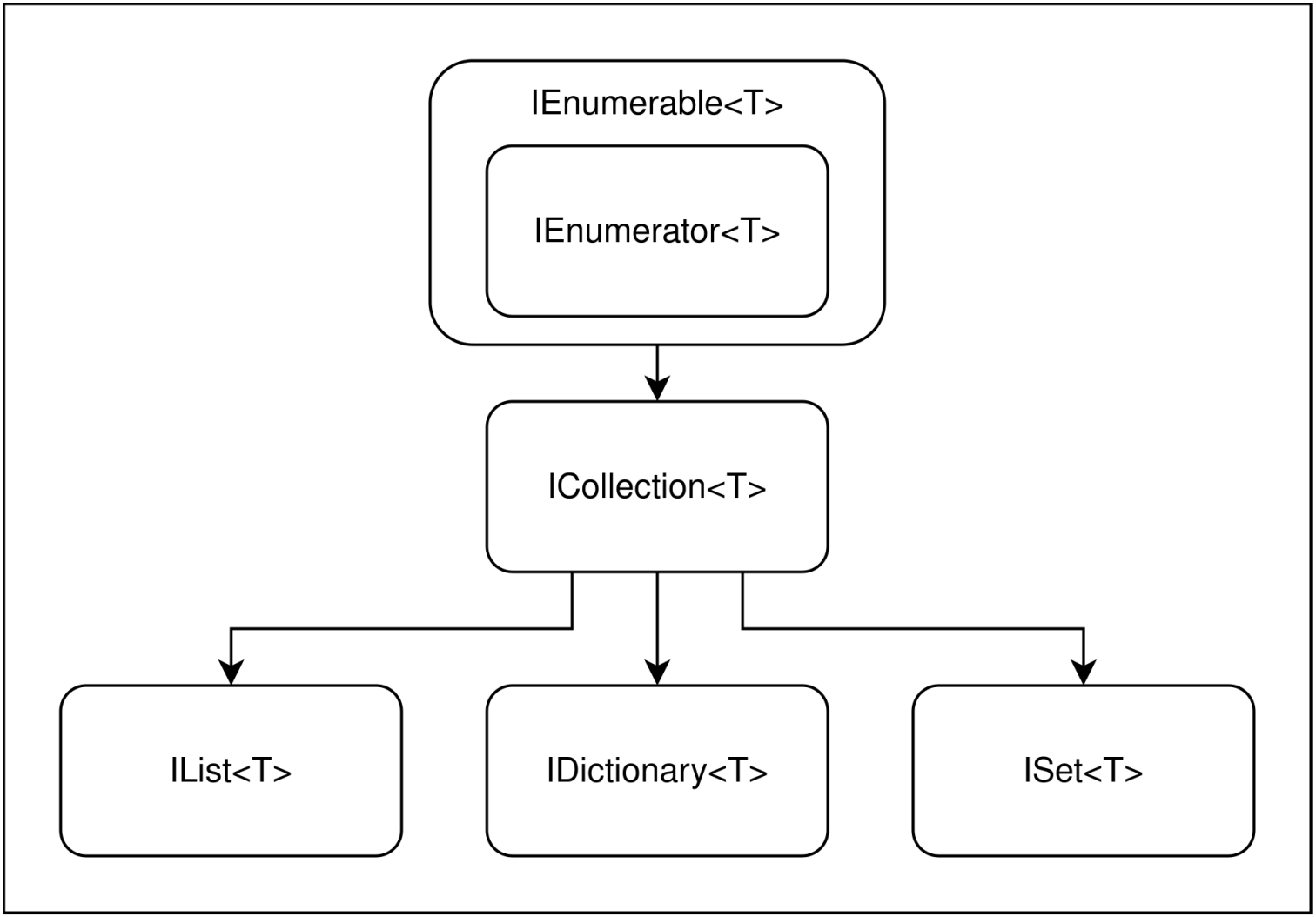
Collections Namespaces

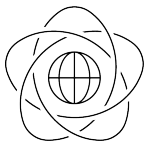
Collection Namespaces:

- `System.Collections` - Nongeneric collection classes and interfaces
- `System.Collections.Generic` - Generic collection classes and interfaces
- `System.Collections.Concurrent` - Thread-safe collections
- `System.Collections.Immutable` - Immutable (read-only) collections



Collections Hierarchy



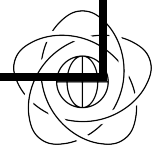


ICollection Interface

ICollection

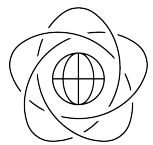
ICollection represents a collection of objects that can be enumerated, counted, and manipulated.

```
1  public interface ICollection<T> : IEnumerable<T>
2  {
3      int Count { get; }
4      bool Contains (T item);
5      void CopyTo (T[] array, int arrayIndex);
6      bool IsReadOnly { get; }
7      void Add(T item);
8      bool Remove (T item);
9      void Clear();
10 }
```



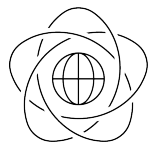
IList Interface

```
1  public interface IList<T> : ICollection<T>
2  {
3      T this [int index] { get; set; }
4      int IndexOf (T item);
5      void Insert (int index, T item);
6      void RemoveAt (int index);
7  }
```



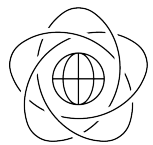
IList Implementations

- List<T>
  - dynamically sized array
  - provides Sort, BinarySearch, Find and some other methods on top of the IList implementation



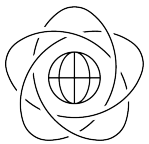
IDictionary interface

```
1  public interface IDictionary<TKey, TValue> :
2      ICollection<KeyValuePair<TKey, TValue>>
3  {
4      bool ContainsKey (TKey key);
5      bool TryGetValue (TKey key, out TValue value);
6      void Add(TKey key, TValue value);
7      bool Remove(TKey key);
8      TValue this [TKey key] { get; set; }
9      ICollection <TKey> Keys { get; }
10     ICollection <TValue> Values { get; }
11 }
```



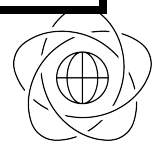
IDictionary Implementations

- Dictionary<T>
  - hash table implementation
  - TKey type should properly override GetHashCode
  - $O(1)$ item retrieval time
- SortedDictionary<T>
  - red/black tree implementation
  - $O(\log n)$ item retrieval time
- SortedList<T>
  - dynamically sized array implementation
  - slower than SortedDictionary
  - low overhead per item
  - $O(n)$ item retrieval time



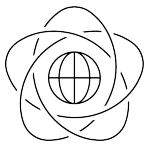
ISet Interface

```
1  public interface ISet<T> : ICollection<T>
2  {
3      bool Add(T item);
4      void UnionWith(IEnumerable<T> other);
5      void IntersectWith(IEnumerable<T> other);
6      void ExceptWith(IEnumerable<T> other);
7      void SymmetricExceptWith(IEnumerable<T> other);
8      bool IsSubsetOf(IEnumerable<T> other);
9      bool IsSupersetOf(IEnumerable<T> other);
10     bool IsProperSupersetOf(IEnumerable<T> other);
11     bool IsProperSubsetOf(IEnumerable<T> other);
12     bool Overlaps(IEnumerable<T> other);
13     bool SetEquals(IEnumerable<T> other);
14 }
```



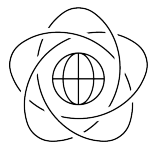
ISet Implementations

- HashSet<T>
  - hash table implementation
  - TKey type should properly override GetHashCode
  - $O(1)$ item lookup
- SortedSet<T>
  - red/black tree implementation
  - $O(\log n)$ item lookup



Other collections implementing ICollection

- LinkedList<T>
  - generic doubly linked list
- Queue<T>
  - dynamically sized array implementation
- Stack<T>
  - dynamically sized array implementation
- BitArray
  - compacted bool values
  - uses 1 bit per boolean



Delegates

Delegate

A delegate is an object that knows how to call a method.

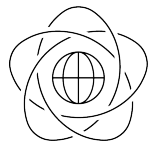
```
1  // Create delegate instance:
2  Function func = Square;
3  // Invoke delegate:
4  double result = func(3.0);
5  Console.WriteLine(result); // 9
6
7  double Square(double x) => x * x;
8  // Delegate type declaration:
9  delegate double Function(double x);
```

Documentation

https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/

Generic delegates

```
1 GenericFunction<double> func = Square;
2 double result = func(3);
3 Console.WriteLine(result);
4
5 // Delegate type declaration:
6 delegate T GenericFunction<T>(T x);
```

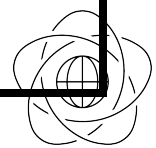


Built-in delegates

Defined in System namespace:

```
1  delegate TResult Func <out TResult>();
2  delegate TResult Func <in T, out TResult>(T arg);
3  delegate TResult Func <in T1, in T2, out TResult>
4      (T1 arg1, T2 arg2);
5  // ... and so on, up to T16
6  delegate void Action();
7  delegate void Action <in T>(T arg);
8  delegate void Action <in T1, in T2>(T1 arg1, T2 arg2);
9  // ... and so on, up to T16
```

```
1  Func<double, double> func = Square;
2  double result = func(3);
3  Action<double> write = Console.WriteLine;
4  write(result);
```



Multicast delegates

```
1  Action<string> writeLog = null!;
2  // += works when delegate is null,
3  // it assigns new value instead
4  writeLog += Console.WriteLine;
5  writeLog += WriteLogToFile;
6  writeLog("DEBUG: This is a test log");
7
8  private static void WriteLogToFile(string log)
9  {
10     File.WriteAllText("test.log", log);
11 }
```

Warning

If a multicast delegate has a non-void return type, the caller receives the return value from the last method to be invoked. The preceding methods are still called, but their return values are discarded.

Lambda expressions

Lambda expression

A lambda expression is an unnamed method written in place of a delegate instance.

Lambda expression has the following syntax:

```
1 (parameters) => expression-or-statement-block
```

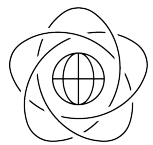
Documentation

`https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/lambda-expressions`

Lambda expressions

examples

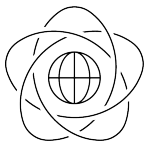
```
1  // Can skip parenthesis if lambda takes only 1 parameter:
2  Func<double, double> square = x => x * x;
3  Func<string> greet = () => "Hello, world";
4  Func<char, int, string> repeat = (c, i) => new string(c, i);
5  Action<string> write = str => Console.Write(str);
```



Lambda expressions

Explicit parameter and return types

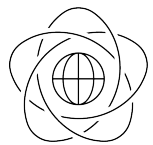
```
1  Func<double, double> square = double (double x) => x * x;
2  Func<string> greet = string () => "Hello, world";
3  Func<char, int, string> repeat =
4      string (char c, int i) => new string(c, i);
5  Action<string> write =
6      void (string str) => Console.Write(str);
```



Lambda expressions

Default parameters (C# 12)

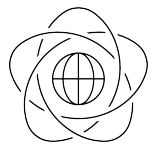
```
1 var write = (string str = "hello") => Console.Write(str);
2 write();
3 write("world");
4 Console.WriteLine();
```



Lambda expressions

Capturing variables

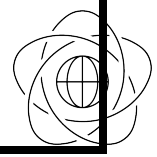
```
1 Action[] actions = new Action[3];
2 for (int i = 0; i < 3; i++)
3     actions [i] = () => Console.Write(i);
4 foreach (Action action in actions) action();
```



Extension Methods

```
1  namespace ExtensionMethods;
2  // Extension methods must be defined inside a static class:
3  public static class MyExtensions
4  {
5  // Are marked with prepending this to the first parameter:
6      public static int WordCount(this string str)
7      {
8          return str.Split(' ',
9              StringSplitOptions.RemoveEmptyEntries).Length;
10     }
11 }
```

```
1  using ExtensionMethods;
2
3  string greeting = "Hello Extension Methods";
4  int i = MyExtensions.WordCount(greeting);
5  int j = greeting.WordCount();
```



System.Linq.Enumerable class

LINQ = Language Integrated Query

The majority of the methods in this class are defined as extension methods that extend IEnumerable<T>.

```
1  string[] names = { "Tom", "Dick", "Harry" };
2  IEnumerable<string> filteredNames = names
3      .Where(n => n.Length >= 4);
4  foreach (string name in filteredNames)
5      Console.WriteLine(name);
```

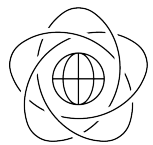
Documentation

https://learn.microsoft.com/en-us/dotnet/api/system.linq.enumerable?view=net-8.0

LINQ

Fluent syntax

```
1  string[] names = { "Tom", "Dick", "Harry", "Mary", "Jay" };
2  IEnumerable<string> query = names
3      .Where(n => n.Contains("a"))
4      .OrderBy(n => n.Length)
5      .Select(n => n.ToUpper());
```



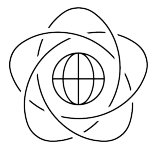
LINQ

Deffered execution

```
1  string[] names = { "Tom", "Dick", "Harry", "Mary", "Jay" };
2  IEnumerable<string> query = names
3      .Where(n => n.Contains("a"))
4      .OrderBy(n => n.Length)
5      .Select(n => n.ToUpper());
6  // Query is not executed, unless enumerated:
7  foreach (var name in query)
8  {
9      Console.WriteLine(name);
10 }
```

Some operators can cause the query to execute immediately

- Aggregation operators: (Max, Average, Count, First, etc.)
- Conversion operations: (ToList, ToArray, ToDictionary, etc.)



LINQ

Internal implementation

```
1  public static IEnumerable<TSource> Where<TSource>
2      (this IEnumerable<TSource> source,
3       Func<TSource,bool> predicate)
4  {
5      foreach (TSource element in source)
6          if (predicate(element))
7              return element;
8  }
```