Telerik Academy

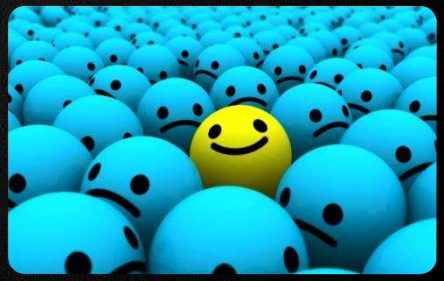
Extension Methods, Lambda Expressions and LINQ

Extension Methods, Anonymous Types, Delegates, Lambda Expressions, LINQ, Dynamic

C# OOP

Telerik Software Academy

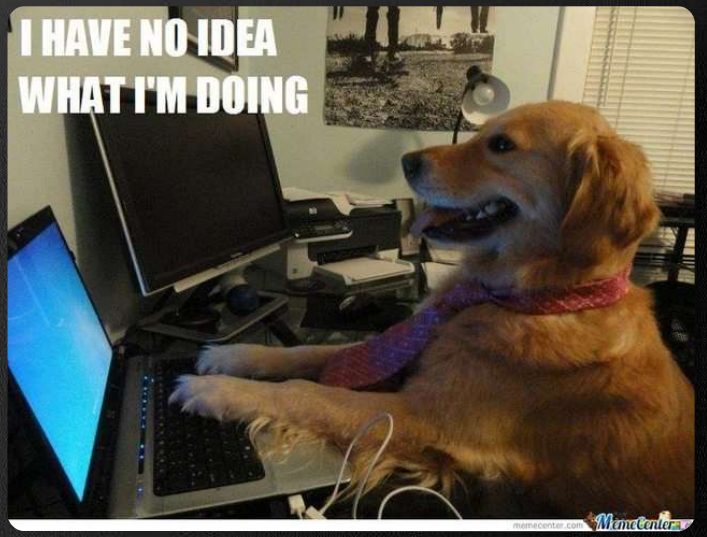
https://telerikacademy.com



Follow us

Telerik Academy

Table of Contents



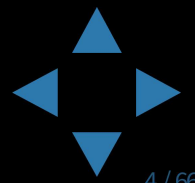
Telerik Academy

Extension Methods

Telerik Academy

Extension Methods

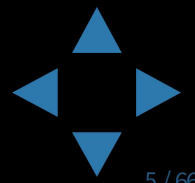
- Once a type is defined and compiled into an assembly its definition is, more or less, final
  - The only way to update, remove or add new members is to recode and recompile the code
- Extension methods allow existing compiled types to gain new functionality
  - Without recompilation
  - Without touching the original assembly



Telerik Academy

Defining Extension Methods

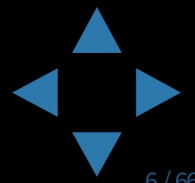
- Extension methods
  - Defined in a static class
  - Defined as `static`
  - Use `this` keyword before its first argument to specify the class to be extended
- Extension methods are "attached" to the extended class
  - Can also be called statically through the defining static class



Telerik Academy

Extension Methods – *Examples*

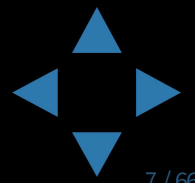
```
public static class Extensions
{
  public static int WordCount(this string str)
  {
    return str.Split(new char[] { ' ', '.', '?' },
    StringSplitOptions.RemoveEmptyEntries).Length;
  }
}
...
static void Main()
{
  string s = "Hello Extension Methods";
  int i = s.WordCount();
  Console.WriteLine(i);
}
```



Telerik Academy

Extension Methods – *Examples*

```
public static void IncreaseWith(
    this IList list, int amount)
{
    for (int i = 0; i < list.Count; i++)
    {
        list[i] += amount;
    }
}
// ...
static void Main()
{
    List ints =
        new List { 1, 2, 3, 4, 5 };
    ints.IncreaseWith(5); // 6, 7, 8, 9, 10
}
```

Telerik Academy

Extension Methods

Demo

Telerik Academy

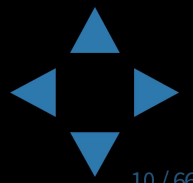
Anonymous Types

Telerik Academy

Anonymous Types

- Anonymous types
  - Encapsulate a set of read-only properties and their value into a single object
  - No need to explicitly define a type first
- To define an anonymous type
  - Use of the new `var` keyword in conjunction with the object initialization syntax

```
var point = new { X = 3, Y = 5 };
```

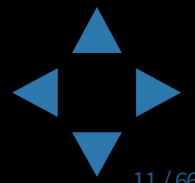


Telerik Academy

Anonymous Types – *Example*

```
// Use an anonymous type representing a car
var myCar =
  new { Color = "Red", Brand = "BMW", Speed = 180 };
Console.WriteLine("My car is a {0} {1}.",
  myCar.Color, myCar.Brand);
```

- At compile time, the C# compiler will autogenerate an uniquely named class
- The class name is not visible from C#
  - Using implicit typing (var keyword) is mandatory

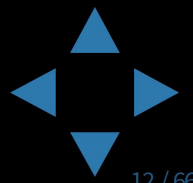


Telerik Academy

Anonymous Types – Properties

- Anonymous types are reference types directly derived from `System.Object`
- Have overridden version of `Equals()`, `GetHashCode()`, and `ToString()`
  - Do not have == and != operators overloaded

```
var p = new { X = 3, Y = 5 };
var q = new { X = 3, Y = 5 };
Console.WriteLine(p == q); // false
Console.WriteLine(p.Equals(q)); // true
```

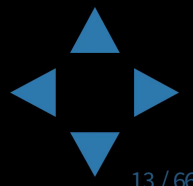



Telerik Academy

Arrays of Anonymous Types

- You can define and use arrays of anonymous types through the following syntax:

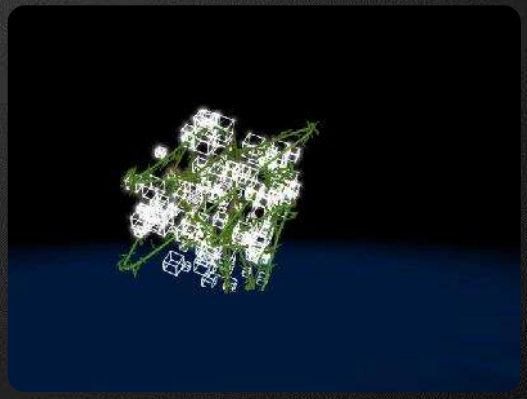
```
var arr = new[]
{
    new { X = 3, Y = 5 },
    new { X = 1, Y = 2 },
    new { X = 0, Y = 7 }
};

foreach (var item in arr)
{
    Console.WriteLine("({0}, {1})", item.X, item.Y);
}
```



Telerik Academy

Anonymous Types

Demo



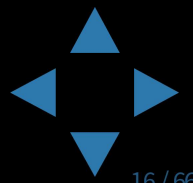
Telerik Academy

Delegates in .NET Framework

Telerik Academy

What are Delegates?

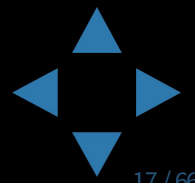
- Delegates are special .NET types that hold a method reference
- Describe the signature of given method
  - Number and types of the parameters
  - The return type
- Their "values" are methods
  - These methods match their signature (parameters and return types)
  - Delegates are reference types



Telerik Academy

What are Delegates?

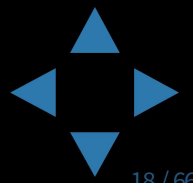
- Delegates are roughly similar to function pointers in C and C++
  - Strongly-typed pointer (reference) to a method
  - Pointer (address) to a callback function
- Can point to static and instance methods
- Can point to a sequence of multiple methods
  - Known as multicast delegates
- Used to perform callback invocations
- Implement the "publish-subscribe" model



Telerik Academy

Delegates – *Example*

```
// Declaration of a delegate
public delegate void SimpleDelegate(string param);
public class Delegates_Example_
{
  static void TestMethod(string param)
  {
    Console.WriteLine("I was called by a delegate.");
    Console.WriteLine("I got parameter: {0}.", param);
  }
  static void Main()
  {
    // Instantiate the delegate
    SimpleDelegate d = new SimpleDelegate(TestMethod);
    // Invocation of the method, pointed by delegate
    d("test");
  }
}
```

Telerik Academy

Simple Delegate

Demo

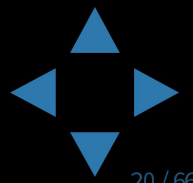
Telerik Academy Generic and Multicast Delegates

- A delegate can be generic:

```
public delegate void SomeDelegate<T>(T item);
```

- Using a generic delegate:

```
public static void Notify(int i) { … }
SomeDelegate d = new SomeDelegate(Notify);
```

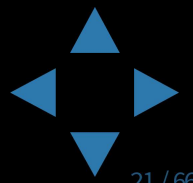
Telerik Academy Generic and Multicast Delegates

- The above can simplified as follows:

```
SomeDelegate d = Notify;
```

- Delegates are multicast (can hold multiple methods), assigned through the += operator:

```
d += Notify;
```

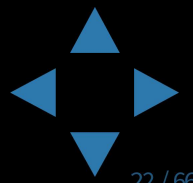



Telerik Academy

Anonymous Methods

- Anonymous methods are methods without name
  - Can take parameters and return values
  - Declared through the `delegate` keyword

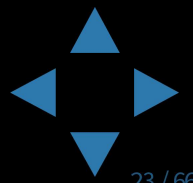
```
class SomeClass
{
  delegate void SomeDelegate(string str);
  static void Main()
  {
    SomeDelegate d = delegate(string str)
    {
      MessageBox.Show(str);
    };
    d("Hello");
  }
}
```

Telerik Academy

Multicast Delegates – *Example*

```
delegate int StringDelegate<T>(T value);
public class MultiDelegates
{
  static int PrintString(string str)
  {
    Console.WriteLine("Str: {0}", str);
    return 1;
  }
  int PrintStringLength(string value)
  {
    Console.WriteLine("Length: {0}", value.Length);
    return 2;
  }
  public static void Main()
  {
    StringDelegate d = MultiDelegates.PrintString;
    d += new MultiDelegates().PrintStringLength;
    int result = d("some string value");
    Console.WriteLine("Returned result: {0}", result);
  }
}
```

Telerik Academy

Predefined Delegates

- Predefined delegates in .NET:
  - `Action<T1,T2,T3>` - generic predefined void delegate with parameters of types T1, T2 and T3
  - `Func<T1,T2,TResult>` - generic predefined delegate with return value of type TResult
  - Both have quite a lot of overloads

```
Func<string, int> predefinedIntParse = int.Parse;
int number = predefinedIntParse("50");

Action<object> predefinedAction = Console.WriteLine;
predefinedAction(1000);
```


Telerik Academy

Multicast Generic Delegate

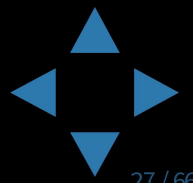
Demo

Telerik Academy

Events

Events

- A message sent by an object to signal the occurrence of an action
- Enable a class or object to notify other classes or objects when something of interest occurs
  - Publisher/event sender – the class that sends/raises the event
    - Doesn't know which object/method will handle the event
  - Subscribers – the classes that receive/handle the event
- In the .NET events are based on the `EventHandler` delegate and the `EventArgs` base class

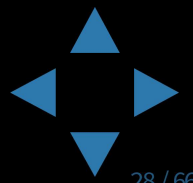


Telerik Academy

Define Events

- Use `event` keyword
- Specify type of delegate for the event – `EventHandler`
- Add a `protected virtual` method
  - Name the method `On[EventName]`

```
class Counter {
  public event EventHandler ThresholdReached;
  protected virtual void OnThresholdReached(EventArgs e) {
    if (this.ThresholdReached != null) {
        this.ThresholdReached(this, e);
    }
  }
  // provide remaining implementation for the class
}
```

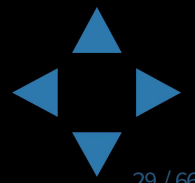



Telerik Academy

Event Data

- Data that is associated with an event can be provided through an event data class
- `EventArgs` class is the base type for all event data classes
  - Also used when an event does not have any data associated with it
  - Naming of the data class – `[Name]EventArgs`

```
public class ThresholdReachedEventArgs : EventArgs
{
  public int Threshold { get; set; }
  public DateTime TimeReached { get; set; }
}
```

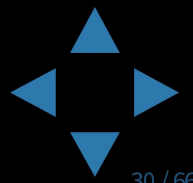


Telerik Academy

Event Handlers

- To respond to an event, you define an event handler method
  - Must match the signature of the delegate

```
class Program {
  static void Main() {
    Counter counter = new Counter();
    counter.ThresholdReached += CounterThresholdReached;
    // provide remaining implementation for the class
  }
  static void CounterThresholdReached(
    object sender, EventArgs e) {
    Console.WriteLine("The threshold was reached.");
  }
}
```



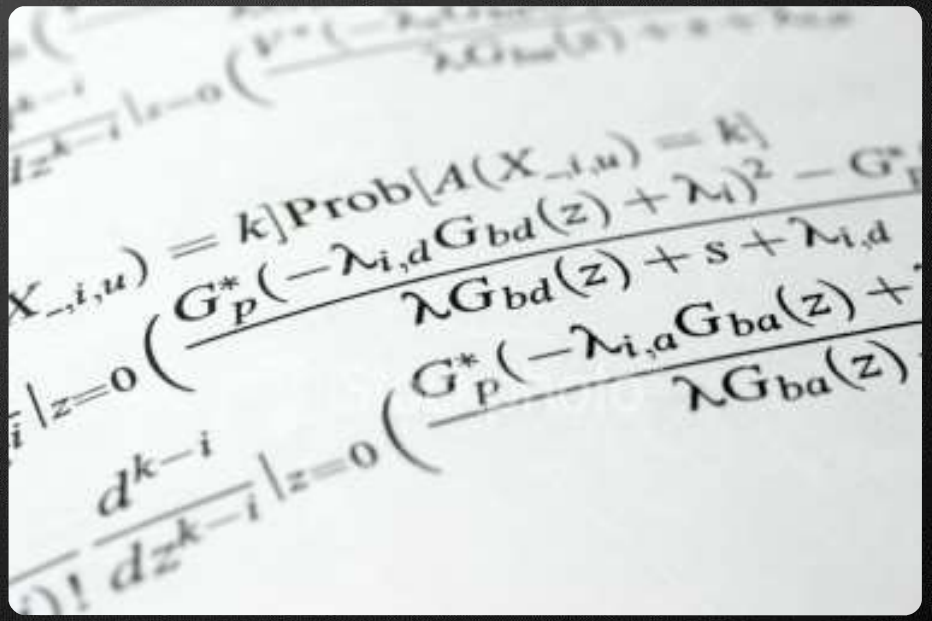
Telerik Academy

Events

Demo

Telerik Academy

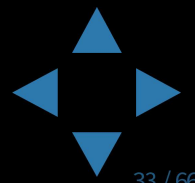
Lambda Expressions



Telerik Academy

Lambda Expressions

- A lambda expression is an anonymous function containing expressions and statements
  - Used to create delegates or expression tree types
- Lambda expressions
  - Use the lambda operator =>
    - Read as "goes to"
  - The left side specifies the input parameters
  - The right side holds the expression or statement
- Link: Lambda notation vs delegate keyword



Telerik Academy

Lambda Expressions – *Examples*

- Usually used with collection extension methods like `FindAll()` and `RemoveAll()`

```
List list = new List() { 1, 2, 3, 4 };
List evenNumbers =
  list.FindAll(x => (x % 2) == 0);
foreach (var num in evenNumbers)
{
    Console.Write("{0} ", num);
}
Console.WriteLine();
// 2 4

list.RemoveAll(x => x > 3); // 1 2 3
```

Telerik Academy

Sorting with Lambda Expression

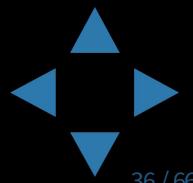
```
var pets = new Pet[]
{
  new Pet { Name="Sharo", Age=8 },
  new Pet { Name="Rex", Age=4 },
  new Pet { Name="Strela", Age=1 },
  new Pet { Name="Bora", Age=3 }
};
var sortedPets = pets.OrderBy(pet => pet.Age);
foreach (Pet pet in sortedPets)
{
  Console.WriteLine("{0} -> {1}",
    pet.Name, pet.Age);
}
```

Telerik Academy

Lambda Code Expressions

- Lambda code expressions:

```
List list = new List()
    { 20, 1, 4, 8, 9, 44 };

// Process each argument with code statements
List evenNumbers = list.FindAll((i) =>
    {
        Console.WriteLine("value of i is: {0}", i);
        return (i % 2) == 0;
    });

Console.WriteLine("Here are your even numbers:");
foreach (int even in evenNumbers)
    Console.Write("{0}\t", even);
```

Telerik Academy

Delegates HoldingLambda Functions

- Lambda functions can be stored in variables of type `delegate`
  - Delegates are typed references to functions
- Standard function delegates in .NET:
  - `Func<TResult>, Func<T,TResult>, Func<T1,T2,TResult>, …`

```
Func<bool> boolFunc = () => true;
Func<int, bool> intFunc = (x) => x < 10;
if (boolFunc() && intFunc(5))
  Console.WriteLine("5 < 10");
```

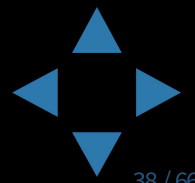
Telerik Academy

Predicates

- Predicates are predefined delegates with the following signature

```
public delegate bool Predicate &lt; T &gt;(T obj)
```

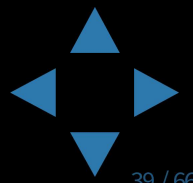
- Define a way to check if an object meets some Boolean criteria
- Similar to `Func<T,bool>`
  - Used by many methods of `Array` and `List<T>` to search for an element
- For example `List<T>.FindAll(…)` retrieves all elements meeting the criteria



Telerik Academy

Predicates – *Example*

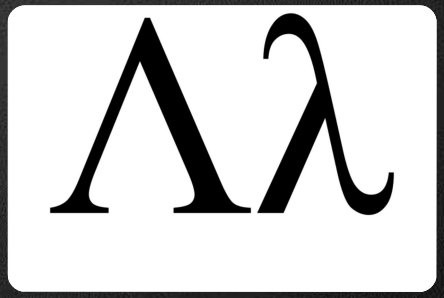
```
List towns = new List()
{
  "Sofia", "Plovdiv", "Varna", "Sopot", "Silistra"
};

List townsWithS =
  towns.FindAll(delegate(string town)
  {
    return town.StartsWith("S");
  });

// A short form of the above (with lambda expression)
List townsWithS =
  towns.FindAll((town) => town.StartsWith("S"));

foreach (string town in townsWithS)
{
  Console.WriteLine(town);
}
```

Telerik Academy

Lambda Expressions

Demo

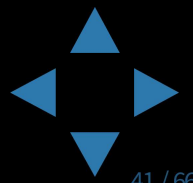


Telerik Academy

Action<T> and Func<T>

- `Action<T>` - void delegate with parameter T
- `Func<T, Result>` - result delegate returning Result

```
Action<int> act = (number) =>
{
   Console.WrileLine(number);}
act(10); // logs 10

Func<string, int, string> greet = (name, age) =>
{
   return "Name: " + name + "Age: " + age;
}
Console.WriteLine(greet("Ivaylo", 10));
```

Telerik Academy

Action<T> and Func<T>

Demo

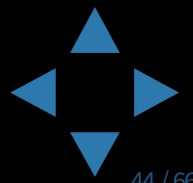
Telerik Academy

LINQ and Query Keywords

Telerik Academy

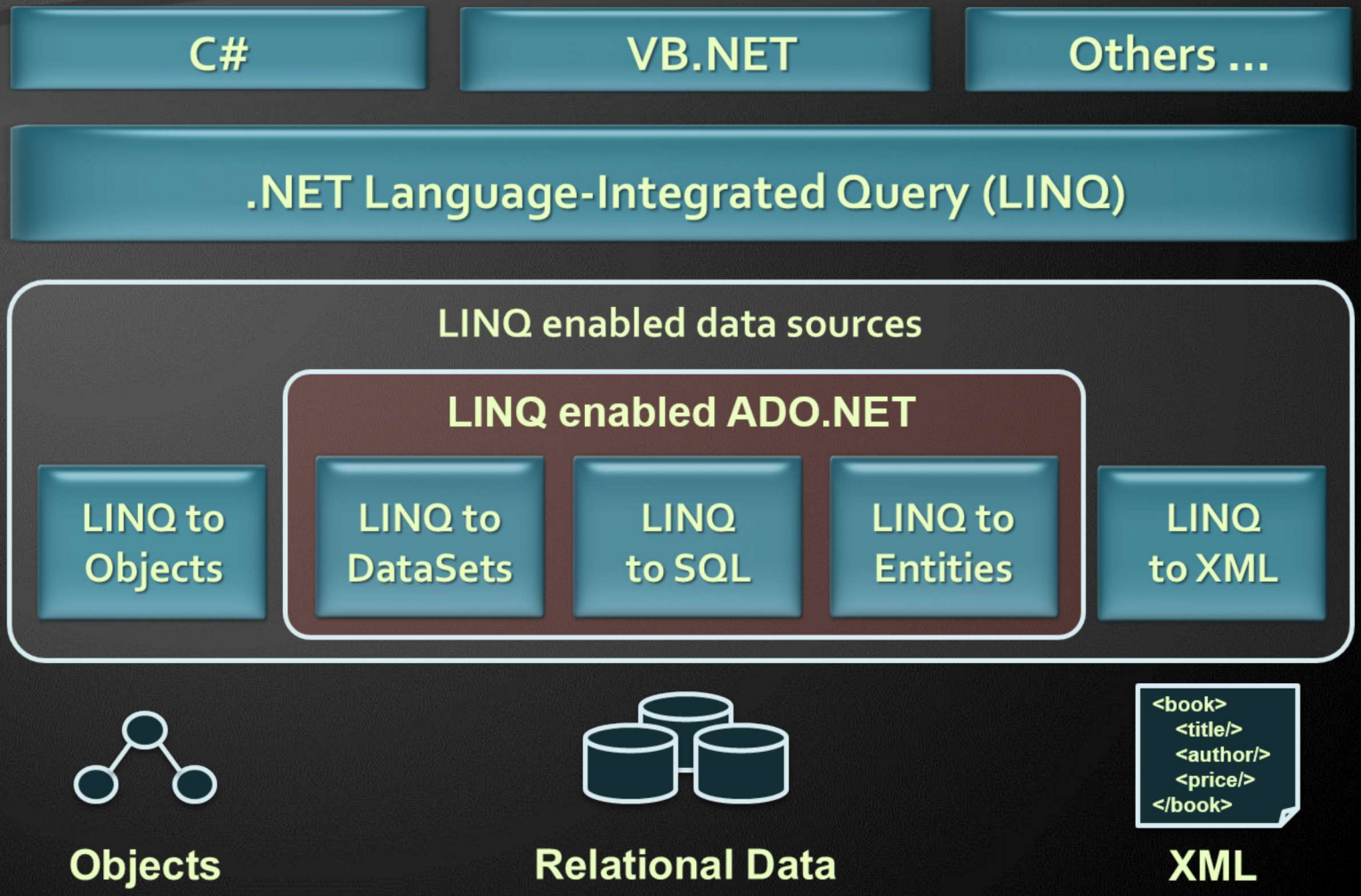
LINQ Building Blocks

- `LINQ` is a set of extensions to .NET Framework
  - Encompasses language-integrated query, set, and transform operations
  - Consistent manner to obtain and manipulate "data" in the broad sense of the term
- `Query expressions` can be defined directly within the C# programming language
  - Used to interact with numerous data types
  - Converted to expression trees at compile time and evaluated at runtime



Telerik Academy

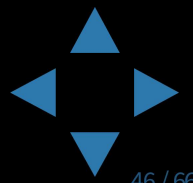
LINQ to *



Telerik Academy

LINQ and Query Keywords

- Language Integrated Query (LINQ) query keywords
  - `from` – specifies data source and range variable
  - `where` – filters source elements
  - `select` – specifies the type and shape that the elements in the returned sequence
  - `group` – groups query results according to a specified key value
  - `orderby` – sorts query results in ascending or descending order

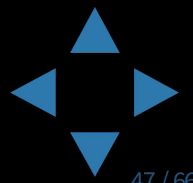


Telerik Academy

Query Keywords *Examples*

- `select`, `from` and `where` clauses:

```
int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

var querySmallNums =
    from num in numbers
    where num < 5
    select num;

foreach (var num in querySmallNums)
{
   Console.Write(num.ToString() + " ");
}
// The result is 4 1 3 2 0
```

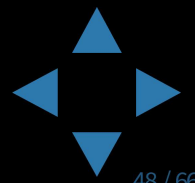


Telerik Academy

Query Keywords *Examples*

- Nested queries:

```
string[] towns =
  { "Sofia", "Varna", "Pleven", "Ruse", "Bourgas" };

var townPairs =
  from t1 in towns
    from t2 in towns
    select new { T1 = t1, T2 = t2 };

foreach (var townPair in townPairs)
{
  Console.WriteLine("({0}, {1})",
    townPair.T1, townPair.T2);
}
```

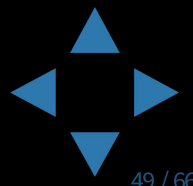



Telerik Academy

Query Keywords *Examples*

- Sorting with orderby:

```
string[] fruits =
  { "cherry", "apple", "blueberry", "banana" };

// Sort in ascending sort
var fruitsAscending =
    from fruit in fruits
    orderby fruit
    select fruit;

foreach (string fruit in fruitsAscending)
{
   Console.WriteLine(fruit);
}
```



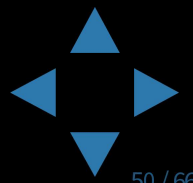
Telerik Academy

Standard Query Operators

Example

```
string[] games = {"Morrowind", "BioShock", "Half Life",
  "The Darkness", "Daxter", "System Shock 2"};

// Build a query expression using extension methods
// granted to the Array via the Enumerable type

var subset = games
            .Where(game => game.Length > 6)
            .OrderBy(game => game)
            .Select(game => game);

foreach (var game in subset)
   Console.WriteLine(game);
```

```
var subset =
  from g in games
  where g.Length > 6
  orderby g
  select g;
```

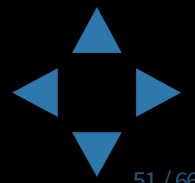



Telerik Academy

Counting the Occurrences of a Word in a String – *Example*

```
string text = "Historically, the world of data ...";
// …
string searchTerm = "data";
string[] source = text.Split(
   new char[] { '.', '?', '!', ' ', ';', ':', ',' },
   StringSplitOptions.RemoveEmptyEntries);

// Use ToLower() to match both "data" and "Data"
var matchQuery =
   from word in source
   where word.ToLower() == searchTerm.ToLower()
   select word;
int wordCount =
   matchQuery.Count();
```

```
int wordCount = source.Select(
  w => w.toLower() ==
  searchTerm.ToLower()).Count();
```

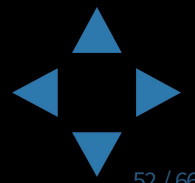
Telerik Academy

Querying Arrays

- Any kind of arrays can be used with LINQ
  - Can be even an untyped array of objects
  - Queries can be applied to arrays of custom objects
  - *Example*:

```
Book[] books = {
   new Book { Title="LINQ in Action" },
   new Book { Title="LINQ for Fun" },
   new Book { Title="Extreme LINQ" } };
var titles = books
   .Where(book => book.Title.Contains("Action"))
   .Select(book => book.Title);
```

```
var titles =
   from b in books
   where b.Title.Contains("Action")
   select b.Title;
```

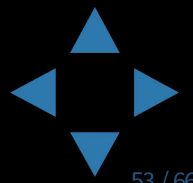



Telerik Academy

Querying Generic Lists

- The previous example can be adapted to work with a generic list
  - `List<T>`, `LinkedList<T>`, `Queue<T>`, `Stack<T>`, `HashSet<T>`, etc.

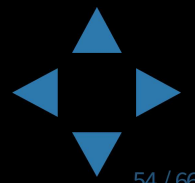
```
List<Book> books = new List<Book>() {
   new Book { Title="LINQ in Action" },
   new Book { Title="LINQ for Fun" },
   new Book { Title="Extreme LINQ" } };
var titles = books
   .Where(book => book.Title.Contains("Action"))
   .Select(book => book.Title);
```

Telerik Academy

Querying Strings

- Although `System.String` may not be perceived as a collection at first sight
  - It actually is a collection, because it implements `IEnumerable<char>`
- String objects can be queried with LINQ to Objects, like any other collection

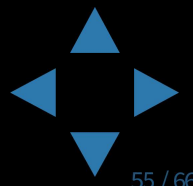


Telerik Academy

Querying Strings

```
var count = "Non-letter characters in this string: 8"
  .Where(c => !Char.IsLetter(c))
  .Count();
Console.WriteLine(count);
// The result is: 8
```

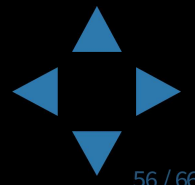
```
var count =
  (from c in "Non-letter…"
  where !Char.IsLetter(c)
  select c).Count();
```



Telerik Academy

Operations

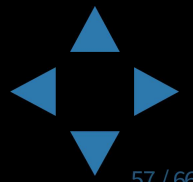
- `Where()`
  - Searches by given condition
- `First()/FirstOrDefault()`
  - Gets the first matched element
- `Last()/LastOrDefault()`
  - Gets the last matched element
- `Select()/Cast()`
  - Makes projection (conversion) to another type
- `OrderBy()/ThenBy()/OrderByDescending()`
  - Orders a collection



Telerik Academy

Operations

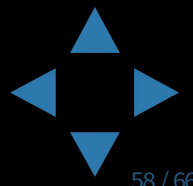
- `Any()`
  - Checks if any element matches a condition
- `All()`
  - Checks if all element matches a condition
- `ToArray()/ToList()/AsEnumerable()`
  - Converts the collection type
- `Reverse()`
  - Reverses a collection



Telerik Academy

Aggregation Methods

- `Average()`
  - Calculates the average value of a collection
- `Count()`
  - Counts the elements in a collection
- `Max()`
  - Determines the maximum value in a collection
- `Sum()`
  - Sums the values in a collection



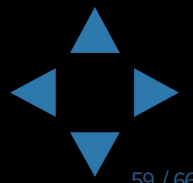
Telerik Academy

Aggregation Methods – *Examples*

- Count(<condition>)

```
double[] temperatures =
   { 28.0, 19.5, 32.3, 33.6, 26.5, 29.7 };
int highTempCount = temperatures.Count(p => p > 30);
Console.WriteLine(highTempCount);
// The result is: 2
```

```
var highTempCount =
  (from p in temperatures
   where p > 30
   select p).Count();
```

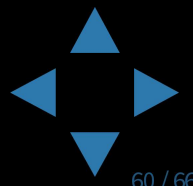
Telerik Academy

Aggregation Methods – *Examples*

- Max()

```
double[] temperatures =
   { 28.0, 19.5, 32.3, 33.6, 26.5, 29.7 };
double maxTemp = temperatures.Max();
Console.WriteLine(maxTemp);
// The result is: 33.6
```

```
var maxTemp =
  (from p in temperatures
   select p).Max();
```



Telerik Academy

LINQ Query Keywords

Demo

Telerik Academy

Dynamic Type

Telerik Academy

Dynamic Type

- The Dynamic type is
  - Defined with the `dynamic` keyword
  - Can hold everything (different from object)
  - Evaluated at runtime
  - *Example*:

```
dynamic dyn = 5;
dyn = "Some text";
dyn = new Student();
dyn.Name = "Ivan";
dyn = new[] { 5, 8, 10 };
```


Telerik Academy

Dynamic Type

Demo