Advanced Programming C#

Lecture 11

dr inż. Małgorzata Janik

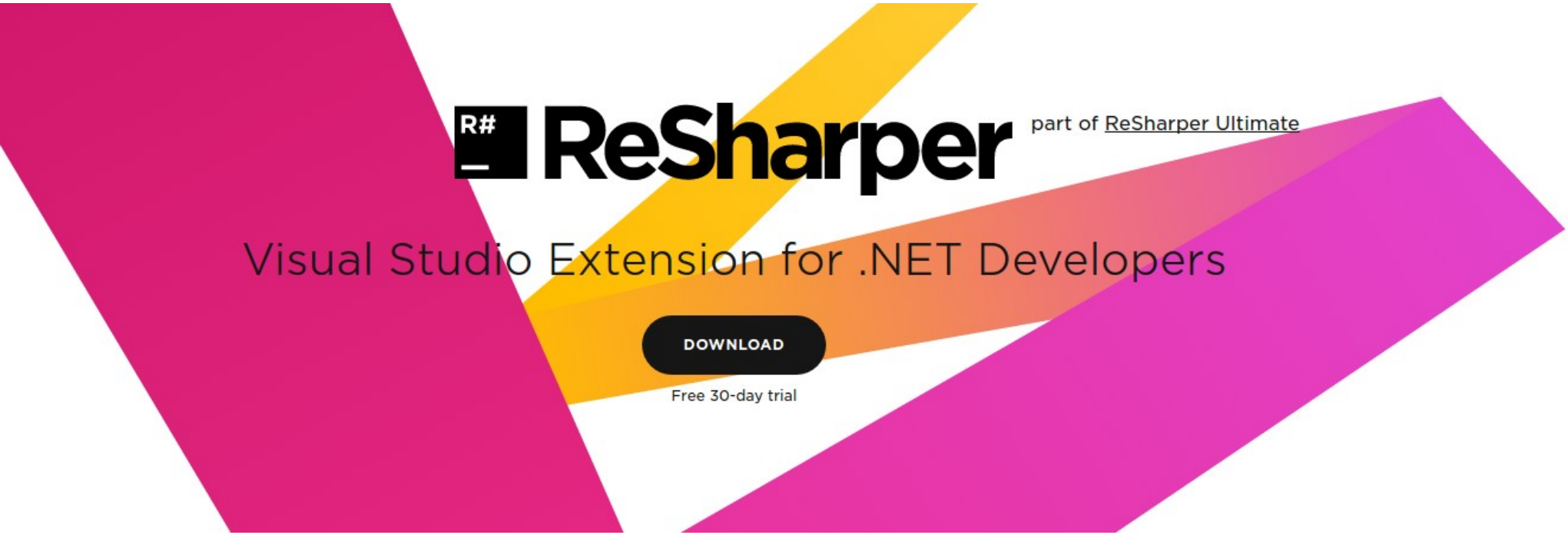
malgorzata.janik@pw.edu.pl

Project part II

- Date: **9.01.2020**
- Prepare the presentation that describes the project advancement – max 5-6 min.
  - Clear list of accomplished tasks
  - Screenshots of application and/or code examples
  - Working prototype expected (½ project should be finished)
  - To-do list
- Send presentations to: malgorzata.janik@pw.edu.pl, 2h before classes.

ReSharper

Try it for your projects!



http://www.jetbrains.com/resharper/

ReSharper

How ReSharper helps Visual Studio users

Analyze code quality

On-the-fly code quality analysis is available in C#, VB.NET, XAML, ASP.NET, ASP.NET MVC, JavaScript, TypeScript, CSS, HTML, and XML. You'll know right away if your code needs to be improved.

Eliminate errors and code smells

Not only does ReSharper warn you when there's a problem in your code but it provides hundreds of quick-fixes to solve problems automatically. In most cases, you can select the best quick-fix from a variety of options.

Safely change the code base

Automated solution-wide code refactorings help safely change your code base. Whether you need to revitalize legacy code or put your project structure in order, you can lean on ReSharper.

Instantly traverse the entire solution

You can instantly navigate and search in the whole solution. Jump to any file, type, or type member, or navigate from a specific symbol to its usages, base and derived symbols, or implementations.

Enjoy code editing helpers

Multiple code editing helpers include extended IntelliSense, hundreds of instant code transformations, auto-importing namespaces, rearranging code and displaying documentation.

Comply to coding standards

Code style and formatting functionality with fine-grained, language-specific settings will help you get rid of unused code and create a common coding standard for your team.


WYDZIAŁ FIZYKI
POLITECHNIKA WARSZAWSKA
POLITECHNIKA WARSZAWSKA
WARSAW UNIVERSITY OF TECHNOLOGY

Arrays

Arrays

- Declaring an array, the square brackets ([]) must come after the type, not the identifier
  - `int[] table; // not int table[];`
- Size of the array is not part of its type
  - `int[] numbers; // declare numbers as an int array of any size`
  - `numbers = new int[10];  // numbers is a 10-element array`
  - `numbers = new int[20];  // now it's a 20-element array`

https://msdn.microsoft.com/en-us/library/aa288453(v=vs.71).aspx

Arrays declarations

- Single-dimensional arrays:
  - `int[] numbers;`
- Multidimensional arrays:
  - `string[,] names;`
- Array-of-arrays (jagged):
  - `byte[][] scores;`
- Declaring them (as shown above) does not actually create the arrays. In C#, arrays are objects and must be instantiated.

https://msdn.microsoft.com/en-us/library/aa288453(v=vs.71).aspx

Creation of arrays

- Single-dimensional arrays:
  - `int[] numbers = new int[5];`
- Multidimensional arrays:
  - `string[,] names = new string[5,4];`
- Array-of-arrays (jagged):

```
byte[][] scores = new byte[5][];
for (int x = 0; x < scores.Length; x++)
{
   scores[x] = new byte[4];
}
```

https://msdn.microsoft.com/en-us/library/aa288453(v=vs.71).aspx

Arrays Examples

- You can have a three-dimensional rectangular array:
  - **`int[,,] buttons = new int[4,5,3];`**
- You can even mix rectangular and jagged arrays. For example, the following code declares a single-dimensional array of three-dimensional arrays of two-dimensional arrays of type int:
  - **`int[][,,][,] numbers;`**

https://msdn.microsoft.com/en-us/library/aa288453(v=vs.71).aspx

Initializing Arrays

- Single-Dimensional Array
  - int[] numbers = new int[5] {1, 2, 3, 4, 5};
  - string[] names = new string[3] {"Matt", "Joanne", "Robert"};
- You can omit the size of the array, like this:
  - int[] numbers = new int[] {1, 2, 3, 4, 5};
  - string[] names = new string[] {"Matt", "Joanne", "Robert"};
- You can also omit the new operator if an initializer is provided, like this:
  - int[] numbers = {1, 2, 3, 4, 5};
  - string[] names = {"Matt", "Joanne", "Robert"};

https://msdn.microsoft.com/en-us/library/aa288453(v=vs.71).aspx

Initializing Arrays

- Multidimensional Array
  - `int[,] numbers = new int[3, 2] { {1, 2}, {3, 4}, {5, 6} };`
  - `string[,] siblings = new string[2, 2] { {"Mike","Amy"}, {"Mary","Albert"} };`
- You can omit the size of the array, like this:
  - `int[,] numbers = new int[,] { {1, 2}, {3, 4}, {5, 6} };`
  - `string[,] siblings = new string[,] { {"Mike","Amy"}, {"Mary","Albert"} };`
- You can also omit the new operator if an initializer is provided, like this:
  - `int[,] numbers = { {1, 2}, {3, 4}, {5, 6} };`
  - `string[,] siblings = { {"Mike", "Amy"}, {"Mary", "Albert"}`

https://msdn.microsoft.com/en-us/library/aa288453(v=vs.71).aspx

Initializing Arrays

- Jagged Array (Array-of-Arrays)
  - int[][] numbers = new int[2][] { new int[] {2,3,4}, new int[] {5,6,7,8,9} };
- You can omit the size of the array, like this:
  - iint[][] numbers = new int[][] { new int[] {2,3,4}, new int[] {5,6,7,8,9} };
- You can also omit the new operator if an initializer is provided, like this:
  - int[][] numbers = { new int[] {2,3,4}, new int[] {5,6,7,8,9} };

https://msdn.microsoft.com/en-us/library/aa288453(v=vs.71).aspx

Accessing Array Members

- Accessing array members is straightforward and similar to how you access array members in C/C++.
  - int[] numbers = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0};
  - numbers[4] = 5;
- Multidimensional array and assigns 5 to the member located at [1, 1]:
  - int[,] numbers = { {1, 2}, {3, 4}, {5, 6}, {7, 8}, {9, 10} };
  - numbers[1, 1] = 5;
- Single-dimension jagged array:
  - int[][] numbers = new int[][] { new int[] {1, 2},
    new int[] {3, 4, 5}};
  - numbers[1][1] = 667;

https://msdn.microsoft.com/en-us/library/aa288453(v=vs.71).aspx

Arrays are Objects

- Length:
  - int[] numbers = {1, 2, 3, 4, 5};
  - int LengthOfNumbers = numbers.Length;
- foreach on Arrays

```
int[,] numbers = new int[3, 2] {{9, 99}, {3, 33}, {5, 55}};
foreach(int i in numbers)
{
   Console.Write("{0} ", i);
}
```

  - The output of this example is: 9 99 3 33 5 55

https://msdn.microsoft.com/en-us/library/aa288453(v=vs.71).aspx


WYDZIAŁ FIZYKI
POLITECHNIKA WARSZAWSKA
POLITECHNIKA WARSZAWSKA
WARSAW UNIVERSITY OF TECHNOLOGY

LINQ

LINQ

- **LINQ** – Language Integrated Query (pronounced „link”)
- .NET Framework component that adds native data querying capabilities to .NET languages
- LINQ extends the language by the addition of query expressions, which are similar to SQL statements
  - can be used to conveniently extract and process data from arrays, enumerable classes, XML documents, relational databases, and third-party data sources.
- LINQ was released as a major part of .NET Framework 3.5 on November 19, 2007.

LINQ Query

All LINQ query operations consist of three distinct actions:

- Obtain the data source.
- Create the query.
- Execute the query.

```
class IntroToLINQ
 {
   static void Main()
   {
     // The Three Parts of a LINQ Query:
     //  1. Data source.
     int[] numbers = new int[7] { 0, 1, 2, 3, 4, 5, 6 };

     // 2. Query creation.
     // numQuery is an IEnumerable<int>
     var numQuery =
       from num in numbers
       where (num % 2) == 0
       select num;

     // 3. Query execution.
     foreach (int num in numQuery)
     {
       Console.Write("{0,1} ", num);
     }
   }
 }
```

In LINQ the execution of the query is distinct from the query itself; in other words you have not retrieved any data just by creating a query variable.

LINQ Data Sources

- The basic rule is very simple: a LINQ data source is any object that supports the generic **IEnumerable<T>** interface, or an interface that inherits from it (e.g. **IQueryable<T>**).
  - Collections: Arrays & Lists

```
string[] words = {"hello", "wonderful", "LINQ", "beautiful"};
```

  - XML documents

```
// using System.Xml.Linq;
XElement contacts = XElement.Load(@"c:\myContactList.xml");
```

  - Databases

```
Northwnd db = new Northwnd(@"c:\northwnd.mdf");
// Query for customers in London.
IQueryable<Customer> custQuery =
from cust in db.Customers  where cust.City == "London"  select cust;
```

More about IEnumerable interface:
https://msdn.microsoft.com/en-us/library/9eekhta0.aspx

Type returned from LINQ query

- From previous example „**var**” type was used: we allow for the compiler to decide on the type
- Actual type of selecting several int numbers from the table would be collection **`IEnumerable<int>`**
- Similarly, for such query:
  - `var queryAllCustomers = from cust in customers select cust;`

  actual type would be **IEnumerable<Customer>**

  - If we use queries such as **Count** or **Max** the result will be of type „int”.

Query Execution

- **Deferred Execution** – query variable itself only stores the query commands. The actual execution of the query is deferred until you **iterate over the query variable in a foreach statement**.
- **Forcing Immediate Execution** – Queries that perform aggregation functions over a range of source elements must first iterate over those elements. Examples of such queries are Count, Max, Average, and First.
  - These execute without an explicit foreach statement because the query itself must use foreach in order to return a result. Note also that these types of queries return a single value, not an IEnumerable collection.
  - To force immediate execution of any query and cache its results, you can call the **ToList()** or **ToArray()** methods.

Query Execution

- **Deferred Execution**

```
foreach (int num in numQuery)
{
    Console.Write("{0,1} ", num);
}
```

- **Forcing Immediate Execution**

```
var evenNumQuery =
    (from num in numbers
    where (num % 2) == 0
    select num).Count();
```

```
List<int> numQuery2 =
(from num in numbers
where (num % 2) == 0
select num).ToList();

var numQuery3 =
  (from num in numbers
  where (num % 2) == 0
  select num).ToArray();
```

LINQ Query

All LINQ query operations consist of three distinct actions:

- Obtain the data source.
- Create the query.
- Execute the query.

```
class IntroToLINQ
 {
   static void Main()
   {
     // The Three Parts of a LINQ Query:
     //  1. Data source.
     int[] numbers = new int[7] { 0, 1, 2, 3, 4, 5,
6 };

     // 2. Query creation.
     // numQuery is an IEnumerable<int>
     var numQuery =
       from num in numbers
       where (num % 2) == 0
       select num;

     // 3. Query execution.
     foreach (int num in numQuery)
     {
       Console.Write("{0,1} ", num);
     }
   }
 }
```

You can use any name you choose for the query as well as for the iterator.

Good practice is to use a name that is easily readable by anybody.

any name

any name

In LINQ the execution of the query is distinct from the query itself; in other words you have not retrieved any data just by creating a query variable.


WYDZIAŁ FIZYKI
POLITECHNIKA WARSZAWSKA
POLITECHNIKA WARSZAWSKA
WARSAW UNIVERSITY OF TECHNOLOGY

Task

Prerequisites

- Download School.cs class from:
  - http://www.if.pw.edu.pl/~majanik/data/Csharp/School.cs
- Create a Console Application
- In the beginning of the program create single School object. Use in as your collection.
- Complete next 8 tasks. Each task should be proceeded by

```
//****** Task [put number] *******
```

comment

Task 0: arrays

- Create 1 dimenstional array for 10 integers. Fill it with random numbers 0-99.
- Print array on the screen.
- Write a LINQ query that will count number of all elements in the array.
- Write a LINQ query that will count number of **even** elements in the array.
- Write a LINQ query that will return list of **odd** elements in the array.
- Print results of all the queries.
- Create 2D array (10 x 5). Fill in with random numbers. Print it on the screen.

Filtering

- The most common query operation is to apply a filter in the form of a Boolean expression
- The filter causes the query to return only those elements for which the expression is true
  - Example:

```
var queryEnglandCustomers =
    from cust in customers
    where cust.Country == "England"
    select cust;
```

- You can combine **multiple Boolean conditions** in the **where clause** in order to further refine a query. The following code adds a condition so that the query returns those pupiles whose first mark was over 4 and whose last mark was less than 3.5.

```
where pupil.Marks[0] > 4 && pupil.Marks[3] < 3.5
```

- Task 1: write a LINQ query to extract all students in the school from Seattle or Warsaw. Print them on the screen. Write second query, that will count number of such students. Print the number on the screen.

Sorting

- Often it is convenient to sort the returned data.
- The **orderby clause** will cause the elements in the returned sequence to be sorted according to the default comparer
  - Example:

```
var queryEnglandCustomersOrder =
        from cust in customers
        where cust.City == "England"
        orderby cust.Name ascending
//descending
        select cust;
```

- Task 2: write a LINQ query to extract all students in the school from Seattle or Warsaw who are sorted by their last name. Print them on the screen.

Grouping

- The **group clause** enables you to group your results based on a key that you specify.
  - Example:

```
// queryCustomersByCountry is an IEnumerable<IGrouping<string, Customer>>
var queryCustomersByCountry =
    from cust in customers
    group cust by cust.Country;

// customerGroup is an IGrouping<string, Customer>
foreach (var customerGroup in queryCustomersByCountry)
{
    Console.WriteLine(customerGroup.Key);
    foreach (Customer customer in customerGroup)
    {
        Console.WriteLine("    {0}", customer.Name);
    }
}
```

- Task 3: write a LINQ query that will write the names of the students (first & last) grouped by the city

Joining multiple inputs into one output

- **.Concat()**
- You can use a LINQ query to create an output sequence that contains elements from more than one input sequence. The following example shows how to combine two in-memory data structures, but the same principles can be applied to combine data from XML or SQL or DataSet sources.
- Task 4: write a LINQ query that will return the collection all of students & teachers (last names only) that live in Warsaw.

```
class Pet
{
  public string Name { get; set; }
  public int Age { get; set; }
}

static Pet[] GetCats()
{
  Pet[] cats = { new Pet { Name="Barley", Age=8
},
          new Pet { Name="Boots", Age=4 } };
  return cats;
}

static Pet[] GetDogs()
{
  Pet[] dogs = { new Pet { Name="Bounder",
Age=3 },
          new Pet { Name="Snoopy", Age=14 }};
  return dogs;
}

public static void ConcatEx1()
{
  Pet[] cats = GetCats();
  Pet[] dogs = GetDogs();

  IEnumerable<string> query =
      (from cat in cats select cat.Name)
      .Concat(from dog in dogs select dog.Name);

  foreach (string name in query)
  {
    Console.WriteLine(name);
  }
}
```

```
// This code produces the following output:
// Barley
// Boots
// Bounder
// Snoopy
```

Selecting subset of each Source Element

- There are 2 primary ways to select a subset of each element in the source sequence:
  - To select just one member of the source element, use the **dot** operation.
    - var query = from cust in Customers select **cust.City**;
  - To create elements that contain more than one property from the source element, you can use an object initializer with either a named object or an anonymous type.
    - var query = from cust in Customer

      select new {Name = cust.Name, City = cust.City};
  - Task 5: write a LINQ query that returns all students & teachers from Warsaw and **creates anonymus type** that includes both first & last names.

Lambda syntax

- Instead of
  - `from person in Data.People select person.LastName;`

  we can write:
  - `Data.People.Select(p => p.LastName);`
- Lambda syntax is more concise, however some operations (i.e. multiple table joins) is very complicated.
- There are a number of LINQ operations that only exist within the Lambda syntax: Single(), First(), Count() etc.

```
SomeDataContext dc = new SomeDataContext();

var queue = from q in dc.SomeTable
      where q.SomeDate <= DateTime.Now && q.Locked != true
      orderby (q.Priority, q.TimeCreated)
      select q;
```

```
var queue2 = dc.SomeTable
       .Where( q => q.SomeDate <= DateTime.Now && q.Locked !=
true )
       .OrderBy(q => q.Priority)
       .ThenBy(q => q.TimeCreated);
```

- Task 6: rewrite Task 1 & Task 4 with lambda syntax.

Transforming in-Memory Objects into XML

- Task 7: try implementing example below to see how to create XML file from collection using LINQ.

```
// Create the query.
            var studentsToXML = new XElement("Root",
                from student in students
                let x = String.Format("{0},{1},{2},{3}", student.Scores[0],
                        student.Scores[1], student.Scores[2], student.Scores[3])
                select new XElement("student",
                           new XElement("First", student.First),
                           new XElement("Last", student.Last),
                           new XElement("Scores", x)
                        ) // end "student"
                    ); // end "Root"

            // Execute the query.
            Console.WriteLine(studentsToXML);
```

Additional

- LINQ Quiz:
  - http://www.albahari.com/nutshell/linqquiz.aspx
- LINQ Myths:
  - http://www.albahari.com/nutshell/10linqmyths.aspx

References

- Getting Started with LINQ in C#

  https://msdn.microsoft.com/en-US/library/bb397933.aspx

  - Introduction to LINQ Queries (C#)
  - LINQ and Generic Types (C#)
  - Basic LINQ Query Operations
  - Data Transformations with LINQ (C#)
  - Type Relationships in LINQ Query Operations
  - Query Syntax and Method Syntax in LINQ
  - C# Features That Support LINQ
  - Walkthrough: Writing Queries in C#
  - Step-by-step instructions for creating a C# LINQ project, adding a simple data source, and performing some basic query operations.

- C#: ZROZUMIEĆ LINQ

  http://itcraftsman.pl/zrozumiec-linq/

THE END

dr inż. Małgorzata Janik

malgorzata.janik@pw.edu.pl