



# Лекция 10

## LINQ. WinForms vs WPF



# LINQ



- Аббревиатура LINQ (Language-Integrated Query) обозначает целый набор технологий, создающих и использующих возможности интеграции запросов непосредственно в язык C# вместо изучения различных языков запросов для каждого типа источников данных: баз данных (реляционных и нереляционных), XML-документов, различных веб-служб и т. д.
- LINQ to Objects: применяется для работы с массивами и коллекциями
- LINQ to Entities: используется при обращении к базам данных через технологию Entity Framework
- LINQ to Sql: технология доступа к данным в MS SQL Server
- LINQ to XML: применяется при работе с файлами XML
- LINQ to DataSet: применяется при работе с объектом DataSet
- Parallel LINQ (PLINQ): используется для выполнения параллельных запросов

```
string[] teams = {"Бавария", "Боруссия", "Реал Мадрид",  
"Манчестер Сити", "ПСЖ", "Барселона"};
```

```
var selectedTeams = new List<string>();
```

```
foreach(string s in teams){
```

```
    if (s.ToUpper().StartsWith("Б"))
```

```
        selectedTeams.Add(s);
```

```
}
```

```
selectedTeams.Sort();
```


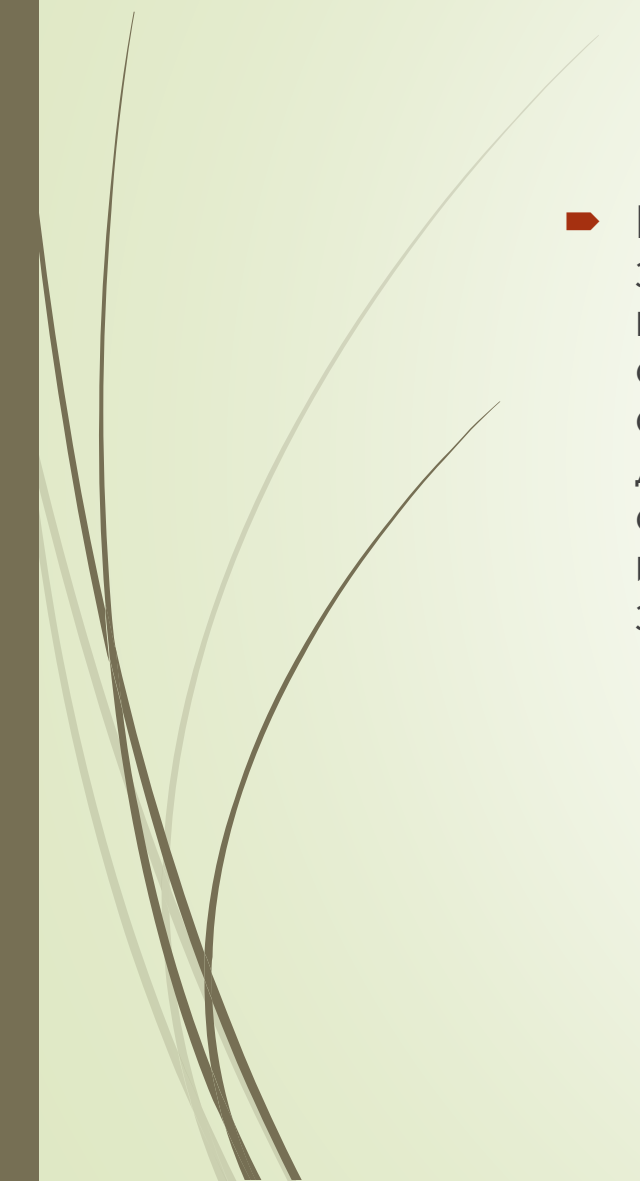
```
////////////////////////////////////
```

```
var selectedTeams = from t in teams
```

```
    where t.ToUpper().StartsWith("Б")
```

```
    orderby t select t;
```



- 
- 
- Выражение запроса должно начинаться предложением `from` и заканчиваться предложением `select` или `group`. Между первым предложением `from` и последним предложением `select` или `group` может содержаться одно или несколько необязательных предложений: `where`, `orderby`, `join`, `let` (определение локальной переменной) и даже дополнительных предложений `from`. Можно также использовать ключевое слово `into`, чтобы результат предложения `join` или `group` мог служить источником дополнительных предложений запроса в том же выражении запроса.

```
IEnumerable<string> queryFirstNames =  
    from name in names  
    let firstName = name.Split(' ')[0]  
    select firstName;
```



# Вложенный запросы

```
var queryGroupMax =  
    from student in students  
    group student by student.GradeLevel into studentGroup  
    select new  
    {  
        Level = studentGroup.Key,  
        HighestScore =  
            (from student2 in studentGroup  
             select student2.Scores.Average())  
            .Max()  
    };
```



# Join

- Соединение по простому ключу

```
var query = from person in people
            join pet in pets on person equals pet.Owner
            select new { OwnerName = person.FirstName, PetName = pet.Name };
```

- Соединение по составному ключу

```
IEnumerable<string> query = from employee in employees
                            join student in students
                                on new { employee.FirstName, employee.LastName }
                                equals new { student.FirstName, student.LastName }
                            select employee.FirstName + " " + employee.LastName;
```



# Обработка null-значений

```
var query =  
    from c in categories  
    where c != null  
    join p in products on c.ID equals  
        p?.CategoryID  
    select new { Category = c.Name, Name = p.Name };
```

```
var query =  
    from o in db.Orders  
    join e in db.Employees  
        on o.EmployeeID equals (int?)e.EmployeeID  
    select new { o.OrderID, e.FirstName };
```



# Left join

```
var query = from person in people
            join pet in pets on person equals pet.Owner into gj
            from subpet in gj.DefaultIfEmpty()
            select new { person.FirstName, PetName = subpet?.Name ?? String.Empty };
```





# Возвращение запроса из метода

```
IEnumerable<string> QueryMethod(ref int[] ints){  
    var intsToStrings = from i in ints  
        where i > 4  
        select i.ToString();  
    return intsToStrings;  
}  
...  
int[] nums = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };  
foreach (string s in app.QueryMethod(ref nums)){  
    Console.WriteLine(s);  
}
```

# LINQ to XML

```
<?xml version="1.0" encoding="utf-8"?>
  <phones>
    <phone name="iPhone 9">
      <company>Apple</company>
      <price>40000</price>
    </phone>
    <phone name="Samsung Galaxy">
      <company>Samsung</company>
      <price>33000</price>
    </phone>
  </phones>
```

```
var xdoc = XDocument.Load("phones.xml");
var items = from xe in xdoc.Element("phones").Elements("phone")
  where xe.Element("company").Value=="Samsung"
  select new Phone {
    Name = xe.Attribute("name").Value,
    Price = xe.Element("price").Value
  };
```



# PLINQ

- PLINQ или параллельный LINQ — это механизм параллельного выполнения выражений LINQ. Другими словами, обычное выражение LINQ можно легко распараллелить между любым количеством потоков. Это выполняется путем вызова метода `AsParallel`. Он реализован как метод расширения LINQ у массивов и коллекций. При вызове данного метода источник данных разделяется на части (если это возможно) и над каждой частью отдельно производятся операции

```
var factorials = from n in numbers.AsParallel()  
    select Factorial(n);
```

# AsOrdered

- Метод упорядочивает результат в соответствии с тем, как элементы располагаются в ИСХОДНОЙ ПОСЛЕДОВАТЕЛЬНОСТИ



```
var factorials = from n in numbers.AsParallel().AsOrdered()  
    where n > 0  
    select Factorial(n);
```

- Если в программе предстоят какие-нибудь манипуляции с полученным набором, однако упорядочивание больше не требуется, мы можем применить метод AsUnordered()

```
var factorials = from n in numbers.AsParallel().AsOrdered()  
    where n > 0  
    select Factorial(n);  
var query = from n in factorials.AsUnordered()  
    where n > 100  
    select n;
```

# Методы расширения LINQ

- Select: определяет проекцию выбранных значений
- Where: определяет фильтр выборки
- OrderBy: упорядочивает элементы по возрастанию
- OrderByDescending: упорядочивает элементы по убыванию
- ThenBy: задает дополнительные критерии для упорядочивания элементов возрастанию
- ThenByDescending: задает дополнительные критерии для упорядочивания элементов по убыванию
- Join: соединяет две коллекции по определенному признаку
- GroupBy: группирует элементы по ключу
- ToLookup: группирует элементы по ключу, при этом все элементы добавляются в словарь
- GroupJoin: выполняет одновременно соединение коллекций и группировку элементов по ключу
- Reverse: располагает элементы в обратном порядке
- All: определяет, все ли элементы коллекции удовлетворяют определенному условию
- Any: определяет, удовлетворяет хотя бы один элемент коллекции определенному условию
- Contains: определяет, содержит ли коллекция определенный элемент
- Distinct: удаляет дублирующиеся элементы из коллекции
- Except: возвращает разность двух коллекций, то есть те элементы, которые создаются только в одной коллекции
- Union: объединяет две однородные коллекции
- Intersect: возвращает пересечение двух коллекций, то есть те элементы, которые встречаются в обеих коллекциях

- 
- 
- Count: подсчитывает количество элементов коллекции, которые удовлетворяют определенному условию
  - Sum: подсчитывает сумму числовых значений в коллекции
  - Average: подсчитывает среднее значение числовых значений в коллекции
  - Agregate: подсчет итогового значения для всех элементов
  - Min: находит минимальное значение
  - Max: находит максимальное значение
  - Take: выбирает определенное количество элементов
  - Skip: пропускает определенное количество элементов
  - TakeWhile: возвращает цепочку элементов последовательности, до тех пор, пока условие истинно
  - SkipWhile: пропускает элементы в последовательности, пока они удовлетворяют заданному условию, и затем возвращает оставшиеся элементы
  - Concat: объединяет две коллекции
  - Zip: объединяет две коллекции в соответствии с определенным условием
  - First: выбирает первый элемент коллекции
  - FirstOrDefault: выбирает первый элемент коллекции или возвращает значение по умолчанию
  - Single: выбирает единственный элемент коллекции, если коллекция содержит больше или меньше одного элемента, то генерируется исключение
  - SingleOrDefault: выбирает первый элемент коллекции или возвращает значение по умолчанию
  - ElementAt: выбирает элемент последовательности по определенному индексу
  - ElementAtOrDefault: выбирает элемент коллекции по определенному индексу или возвращает значение по умолчанию, если индекс вне допустимого диапазона
  - Last: выбирает последний элемент коллекции
  - LastOrDefault: выбирает последний элемент коллекции или возвращает значение по умолчанию



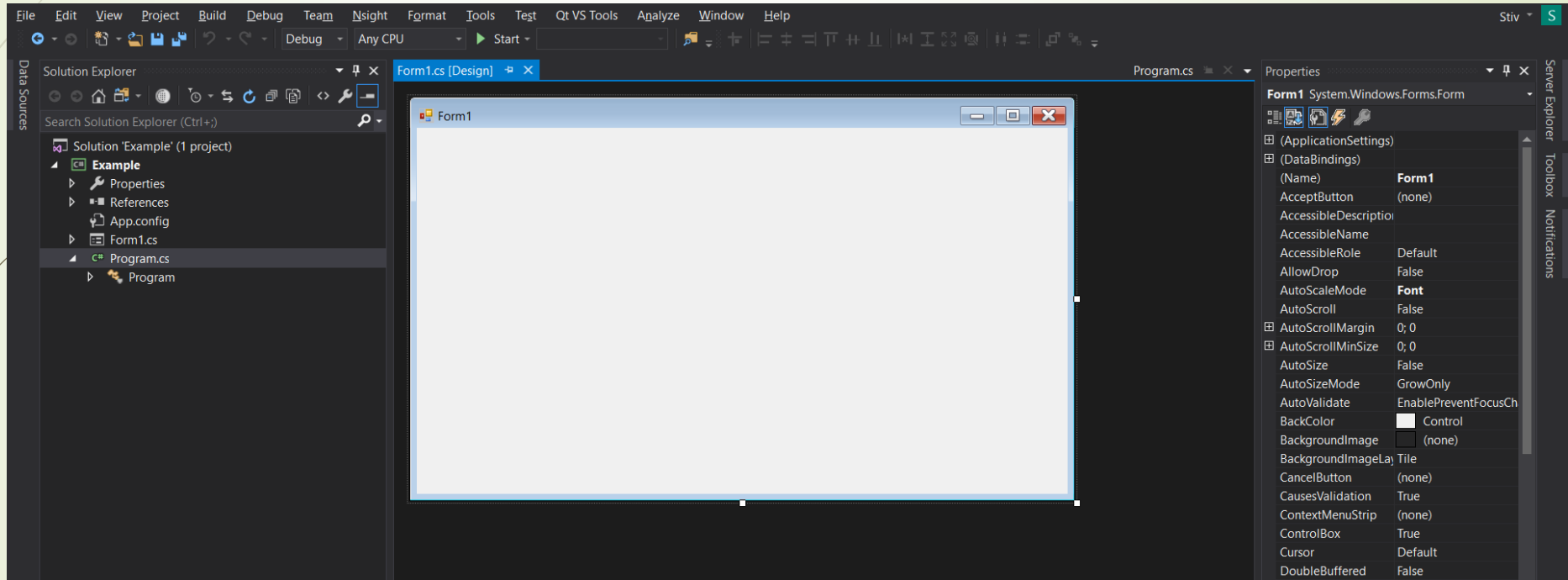


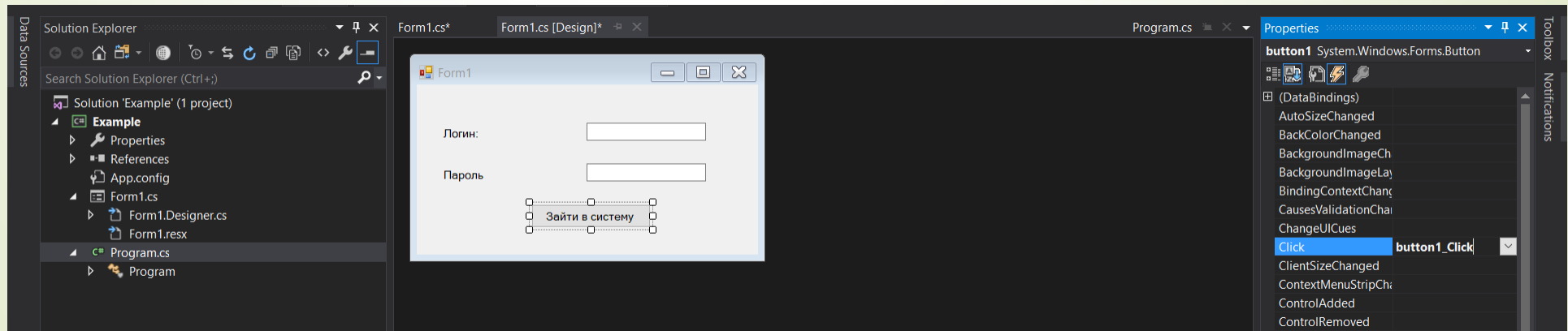
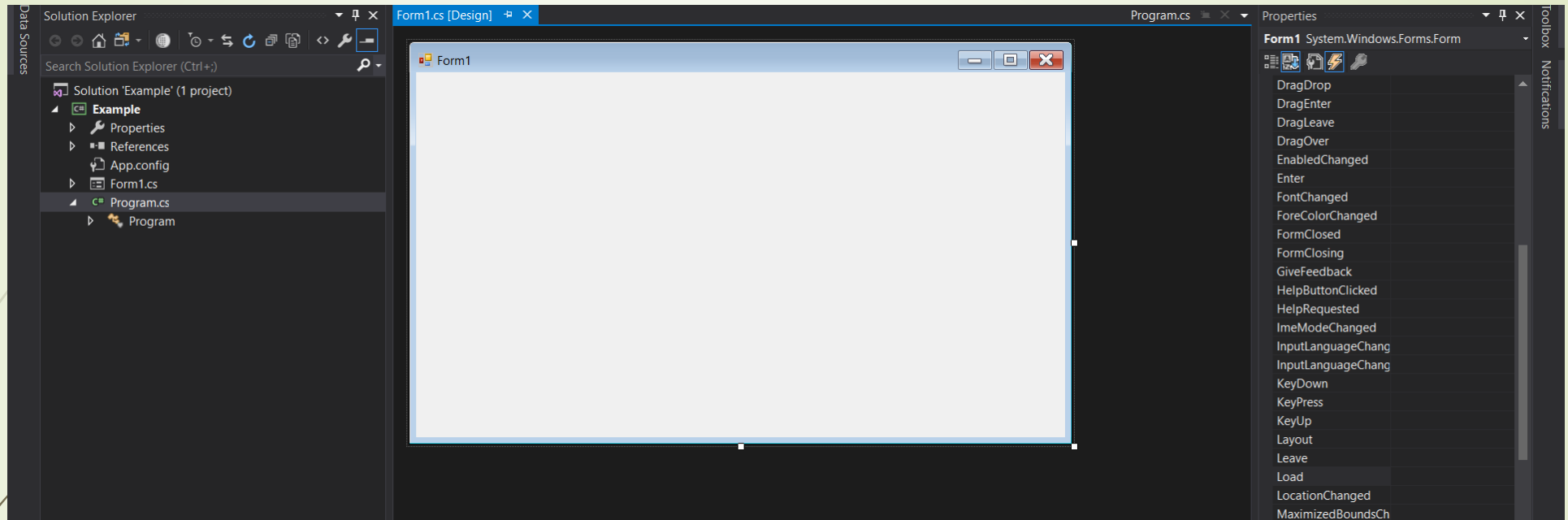
# Zip

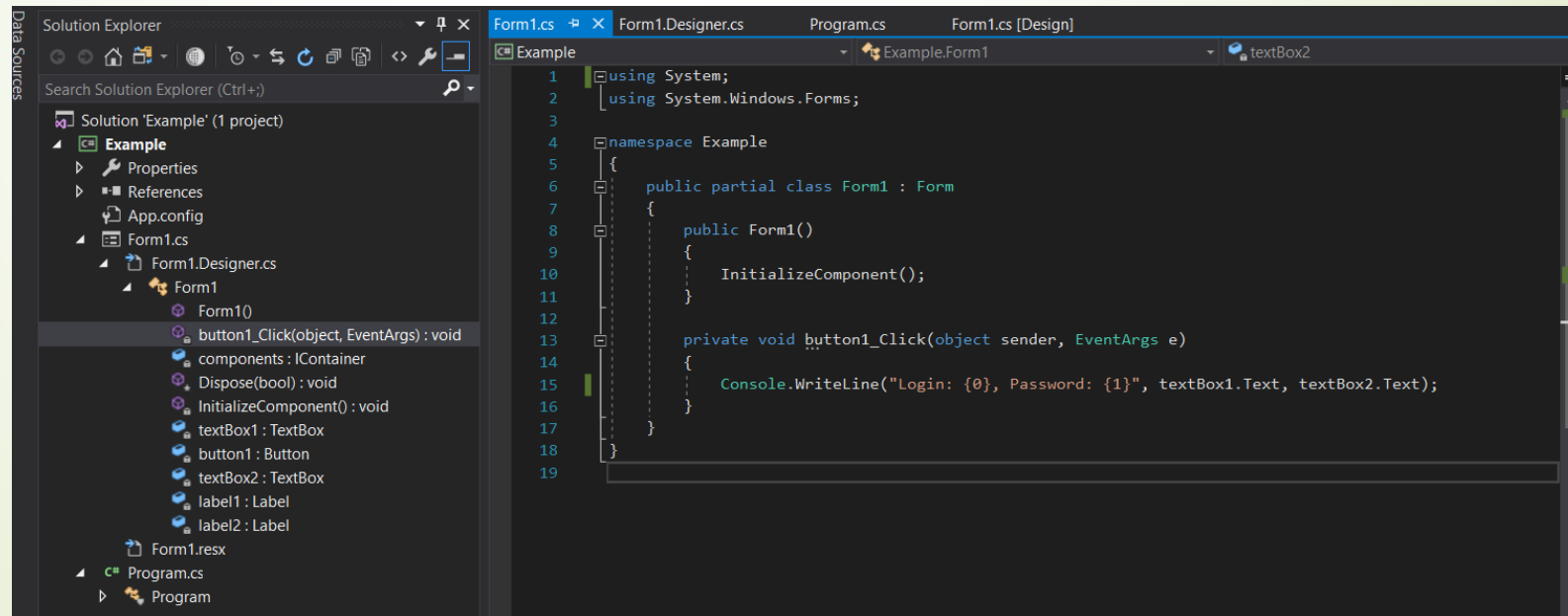
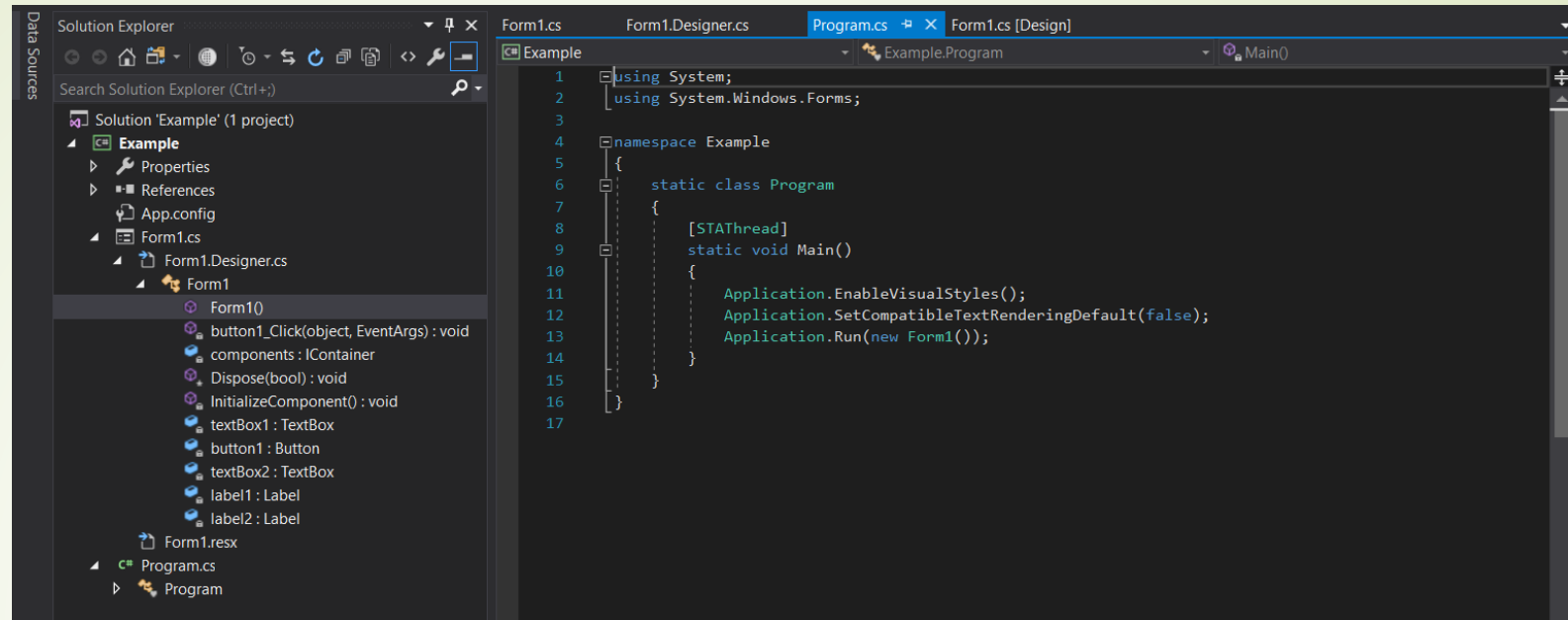
- Метод Zip позволяет объединять две последовательности таким образом, что первый элемент из первой последовательности объединяется с первым элементом из второй последовательности, второй элемент из первой последовательности соединяется со вторым элементом из второй последовательности и так далее

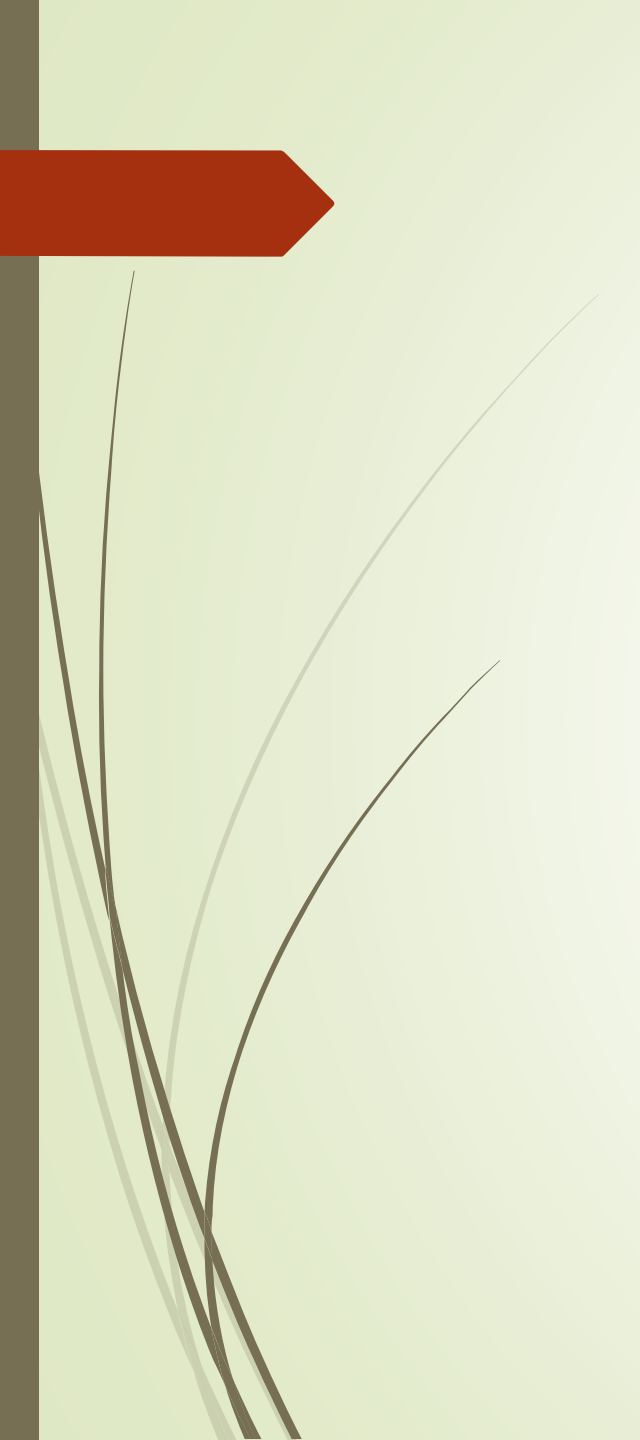
```
var result2 = players.Zip(teams,  
    (player, team) => new{  
        Name = player.Name,  
        Team = team.Name, Country = team.Country  
    });
```

# WinForms









Solution Explorer

Solution 'Example' (1 project)

- Example
  - Properties
  - References
  - App.config
  - Form1.cs
    - Form1.Designer.cs
      - Form1
        - Form10
        - button1\_Click(object, EventArgs) : void
        - components : IContainer
        - Dispose(bool) : void
        - InitializeComponent() : void
        - textBox1 : TextBox
        - button1 : Button
        - textBox2 : TextBox
        - label1 : Label
        - label2 : Label
  - Form1.resx
  - Program.cs
    - Program

Form1.cs

Form1.Designer.cs\*

Program.cs

Form1.cs [Design]

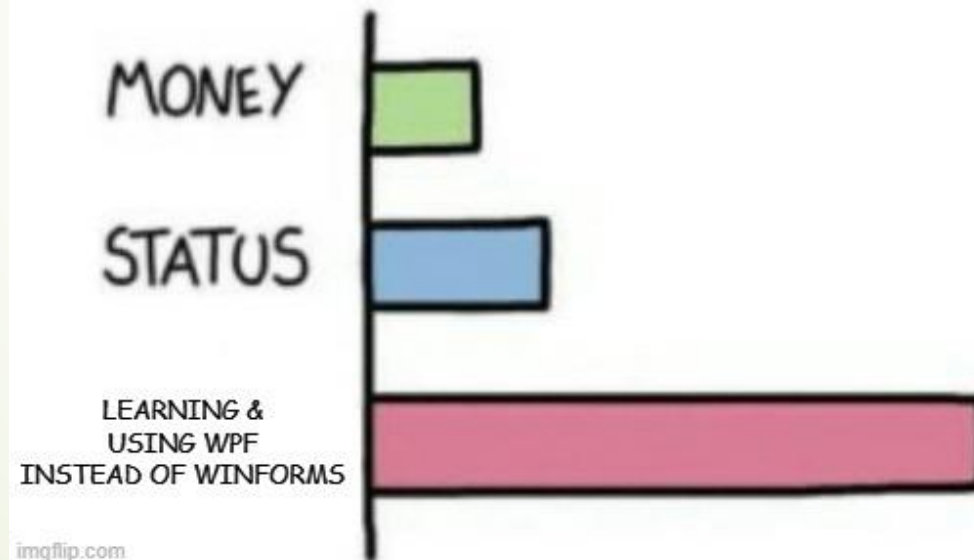
```
1 namespace Example
2 {
3     partial class Form1
4     {
5         private System.ComponentModel.IContainer components = null;
6
7         protected override void Dispose(bool disposing)
8         {
9             if (disposing && (components != null))
10             {
11                 components.Dispose();
12             }
13             base.Dispose(disposing);
14         }
15
16         #region Windows Form Designer generated code
17
18         private void InitializeComponent()
19         {
20             //
21             // button1
22             //
23             this.button1.Location = new System.Drawing.Point(141, 147);
24             this.button1.Name = "button1";
25             this.button1.Size = new System.Drawing.Size(147, 29);
26             this.button1.TabIndex = 1;
27             this.button1.Text = "Зайти в систему";
28             this.button1.UseVisualStyleBackColor = true;
29             this.button1.Click += new System.EventHandler(this.button1_Click);
```

Solution 'Example' (1 project)

- Example
  - Properties
  - References
  - App.config
  - Form1.cs
    - Form1.Designer.cs
      - Form1
        - Form10
        - button1\_Click(object, EventArgs) : void
        - components : IContainer
        - Dispose(bool) : void
        - InitializeComponent() : void
        - textBox1 : TextBox
        - button1 : Button
        - textBox2 : TextBox
        - label1 : Label
        - label2 : Label
    - Form1.resx
    - Program.cs
      - Program

```
30 //
31 // Form1
32 //
33 this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
34 this.AutoScaleMode = System.Windows.Forms.AutoScaleModeMode.Font;
35 this.ClientSize = new System.Drawing.Size(420, 209);
36 this.Controls.Add(this.label2);
37 this.Controls.Add(this.label1);
38 this.Controls.Add(this.textBox2);
39 this.Controls.Add(this.button1);
40 this.Controls.Add(this.textBox1);
41 this.Name = "Form1";
42 this.Text = "Form1";
43 this.ResumeLayout(false);
44 this.PerformLayout();
45
46 #endregion
47
48 private System.Windows.Forms.TextBox textBox1;
49 private System.Windows.Forms.Button button1;
50 private System.Windows.Forms.TextBox textBox2;
51 private System.Windows.Forms.Label label1;
52 private System.Windows.Forms.Label label2;
```

# WHAT GIVES PEOPLE FEELINGS OF POWER







# WPF (Windows Presentation Foundation)

- Независимость от разрешения экрана: поскольку в WPF все элементы измеряются в независимых от устройства единицах, приложения на WPF легко масштабируются под разные экраны с разным разрешением.
- Новые возможности, которых сложно было достичь в WinForms, например, создание трехмерных моделей, привязка данных, использование таких элементов, как стили, шаблоны, темы и др.
- Богатые возможности по созданию различных приложений: это и мультимедиа, и двухмерная и трехмерная графика, и богатый набор встроенных элементов управления, а также возможность самим создавать новые элементы, создание анимаций, привязка данных, стили, шаблоны, темы и многое другое
- Декларативное создание интерфейса
- Аппаратное ускорение графики - вне зависимости от того, работаете ли вы с 2D или 3D, графикой или текстом, все компоненты приложения транслируются в объекты, понятные Direct3D, и затем визуализируются с помощью процессора на видеокарте, что повышает производительность, делает графику более плавной.



# Grid



```
<Grid ShowGridLines="True">
  <Grid.RowDefinitions>
    <RowDefinition></RowDefinition>
    <RowDefinition></RowDefinition>
    <RowDefinition></RowDefinition>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition></ColumnDefinition>
    <ColumnDefinition></ColumnDefinition>
    <ColumnDefinition></ColumnDefinition>
  </Grid.ColumnDefinitions>
  <Button Grid.Column="0" Grid.Row="0" Content="Строка 0 Столбец 0" />
  <Button Grid.Column="0" Grid.Row="1" Content="Объединение трех столбцов" Grid.ColumnSpan="3" />
  <Button Grid.Column="2" Grid.Row="2" Content="Строка 2 Столбец 2" />
</Grid>
```