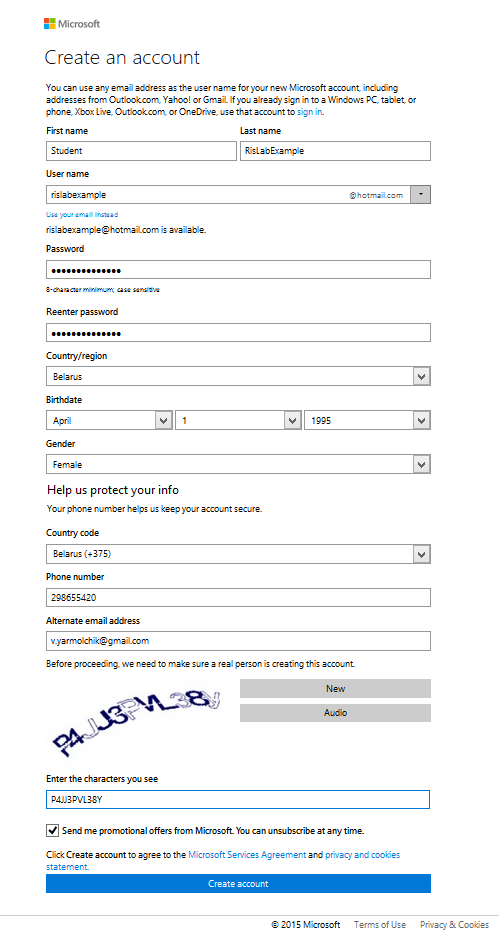
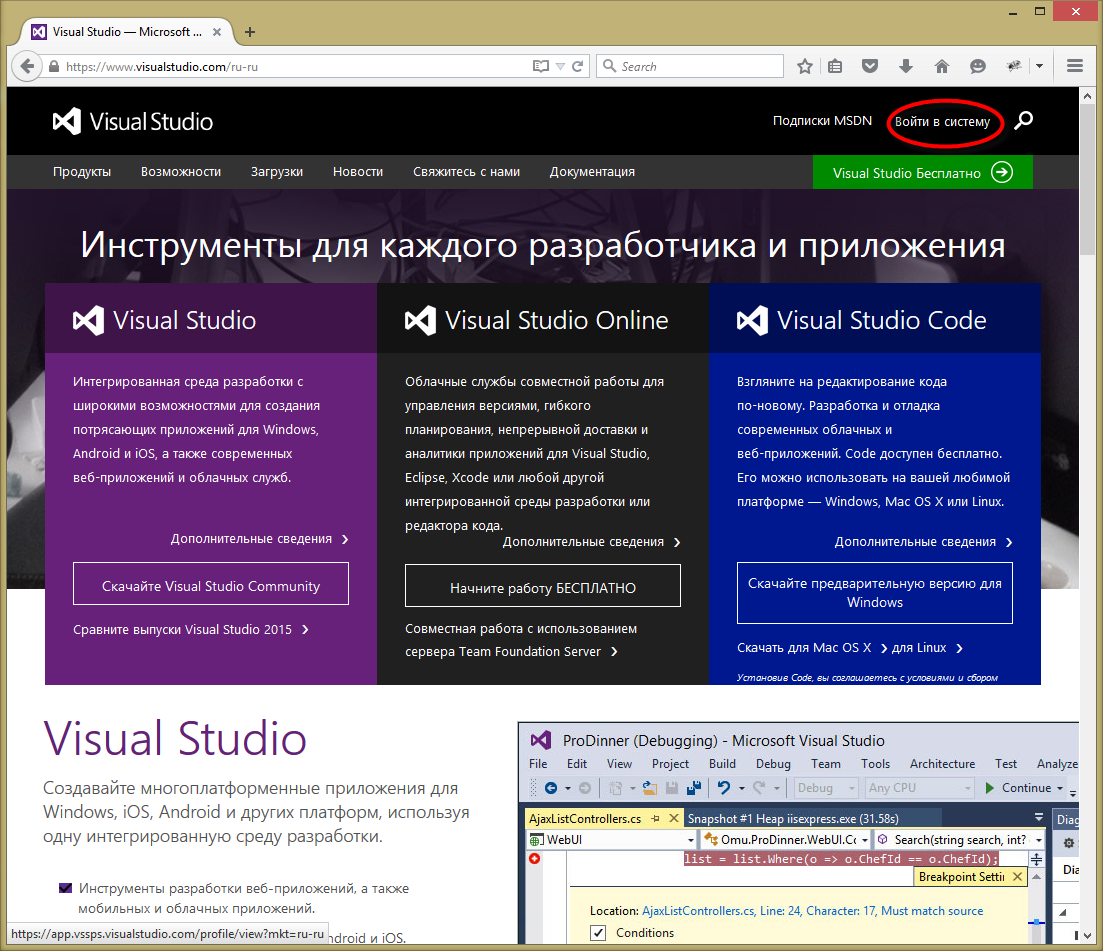
Лабораторная №1

Создание аккаунта и проекта в TFS, добавление участников, настройка TFS в Visual Studio.

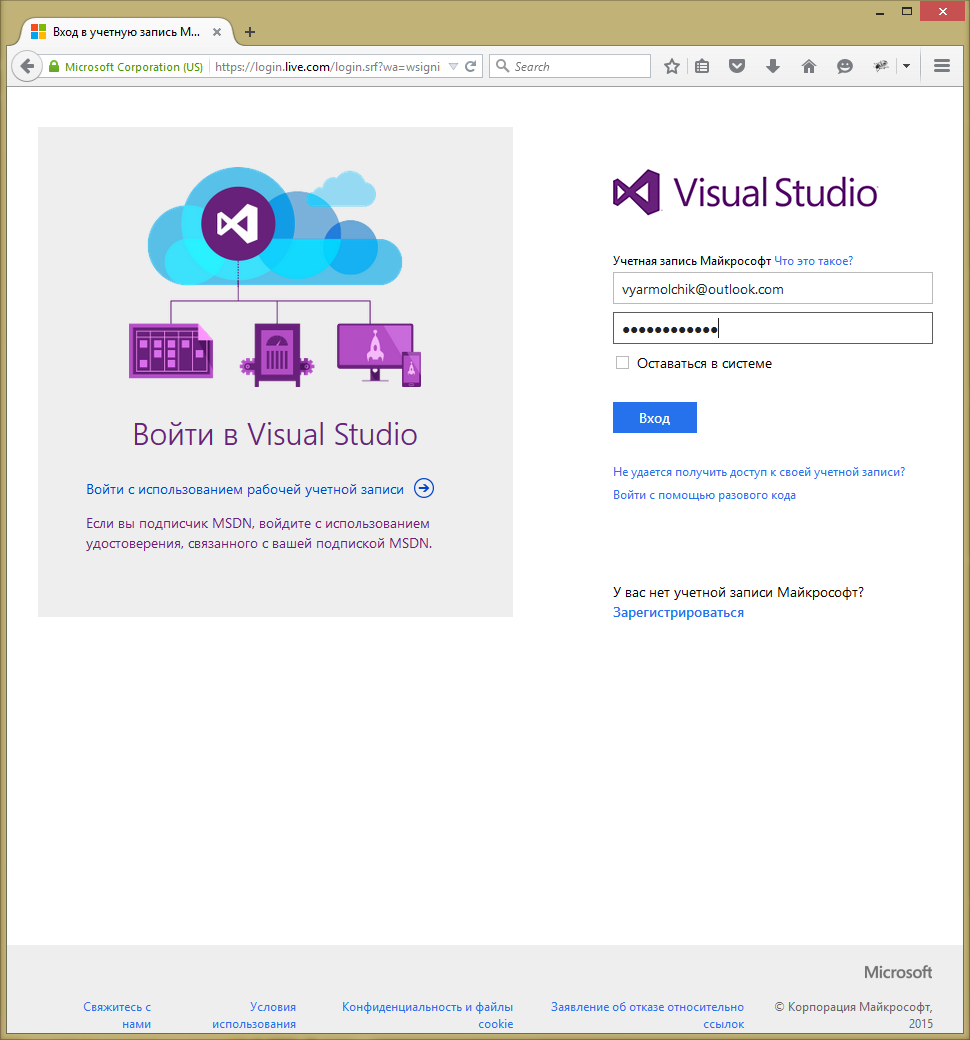
Для начала каждому нужно [создать аккаунт Microsoft](https://signup.live.com/signup). На рисунке ниже представлено форма регистрации новой учетной записи.



Потом идем на [сайт Visual Studio](https://www.visualstudio.com/ru-ru) и жмем Войти в систему.

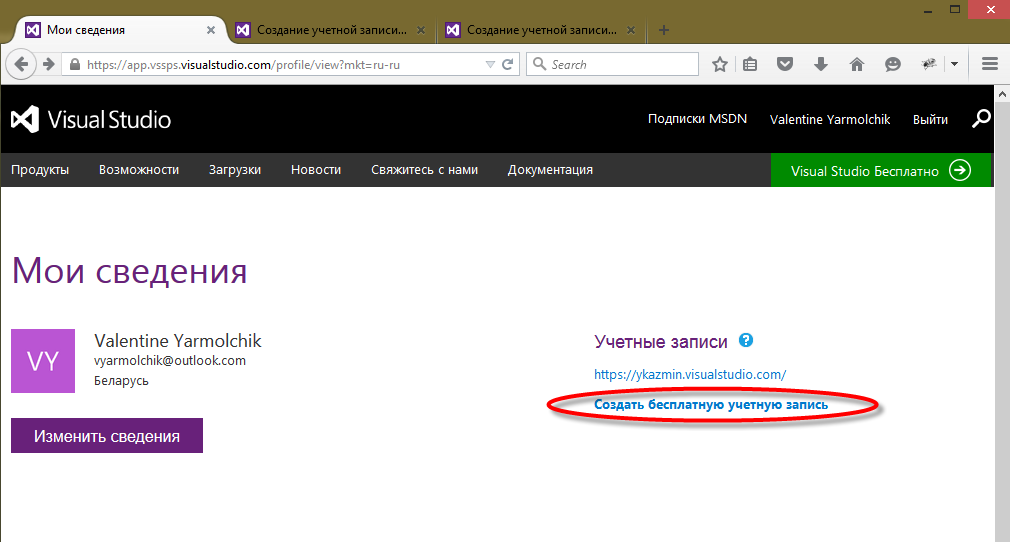


Вводим данные своей учетной записи Microsoft.

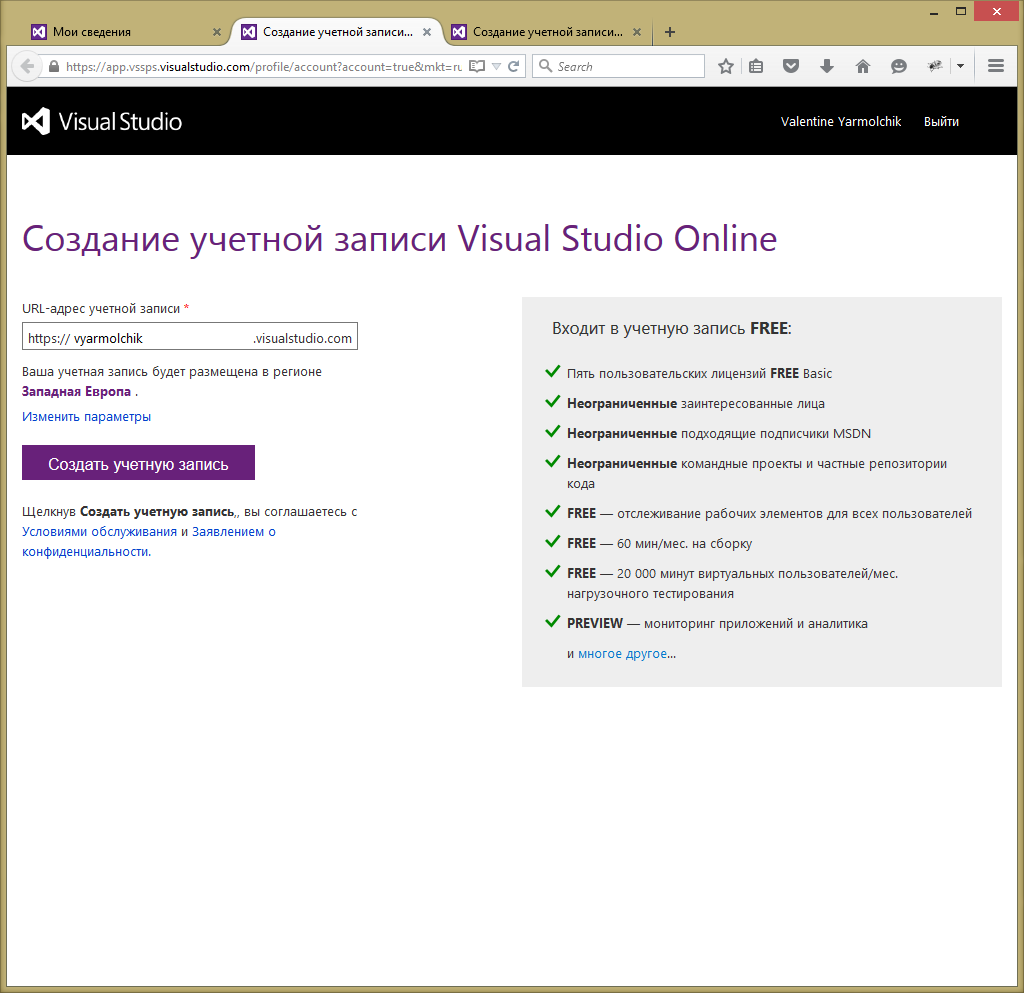


# Инструкция для тех, кто добавляет проект и участников

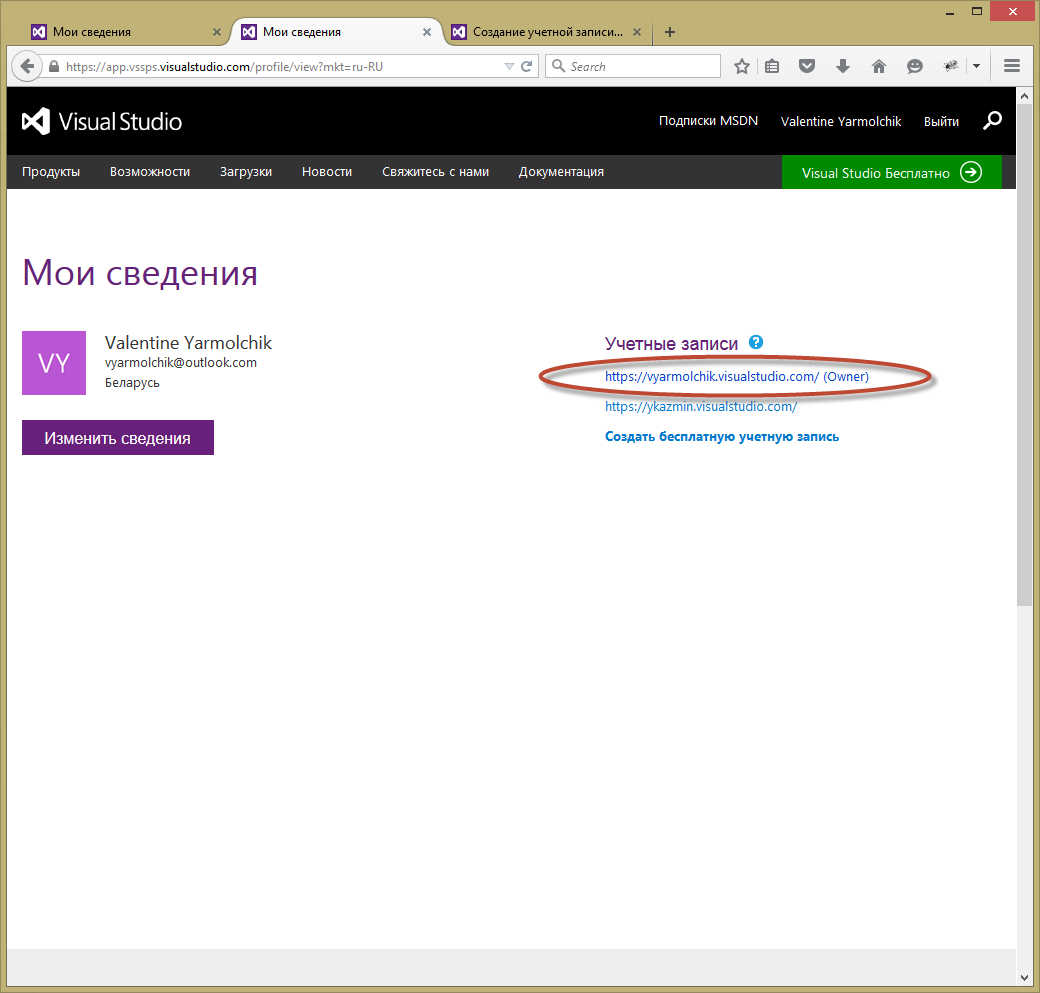
После входа выбираем «Создать бесплатную учетную запись»



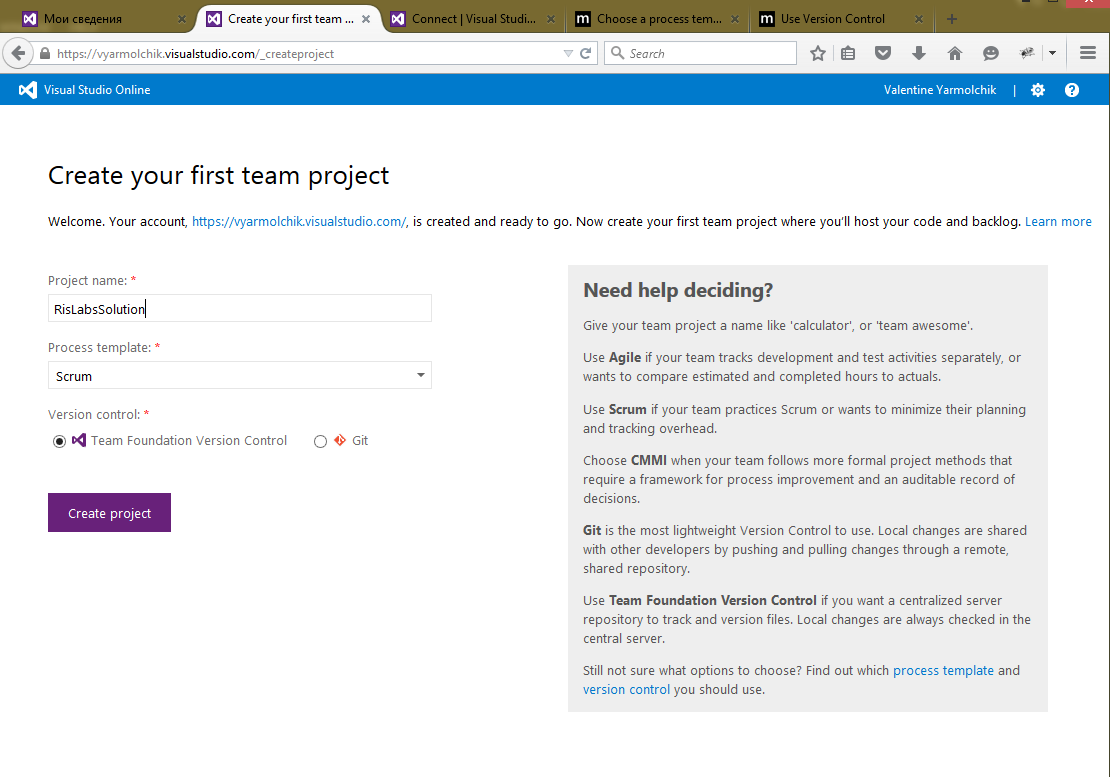
Придумываем название для своей учетной записи в Visual Studio и нажимаем Создать учетную запись.



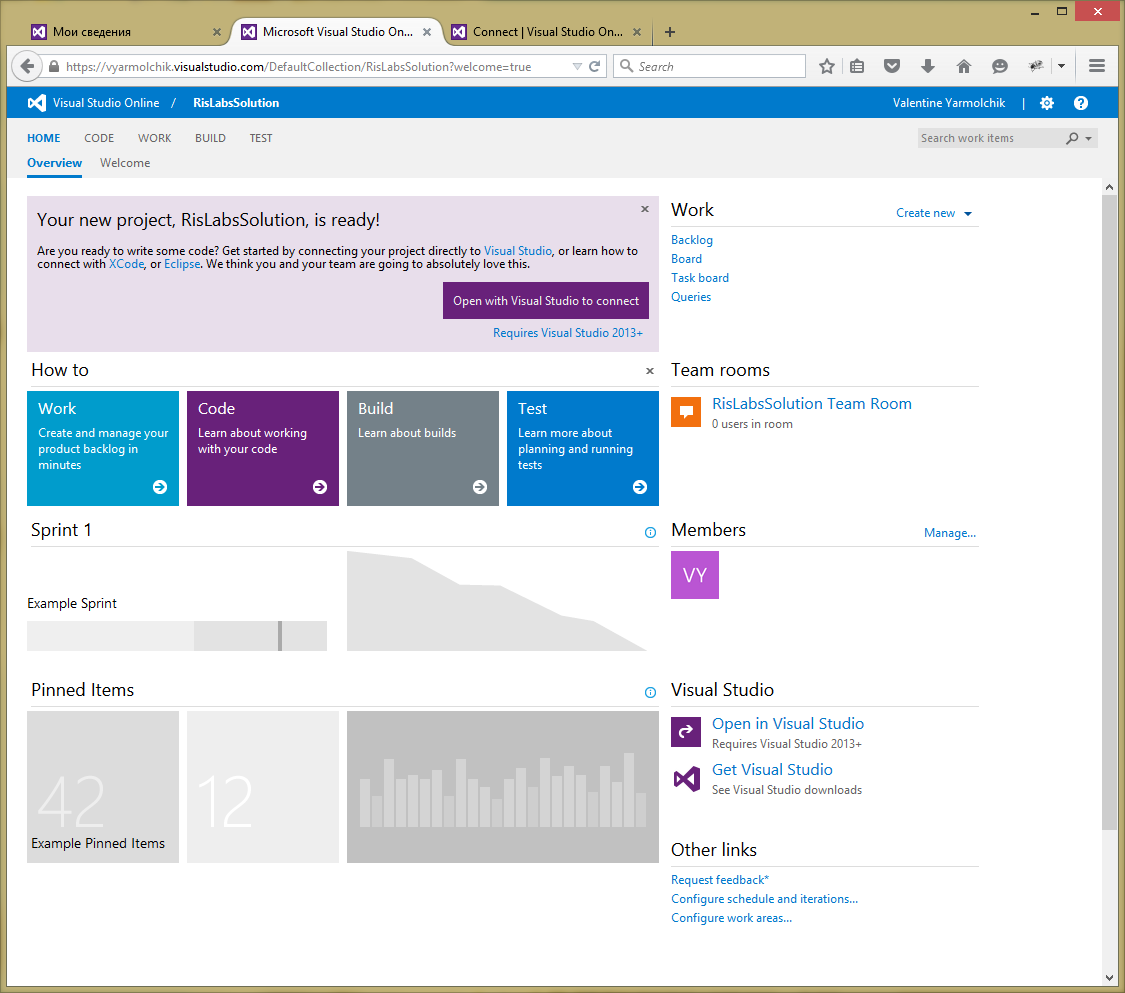
В списке учетных записей выбираем свою новоиспеченную.



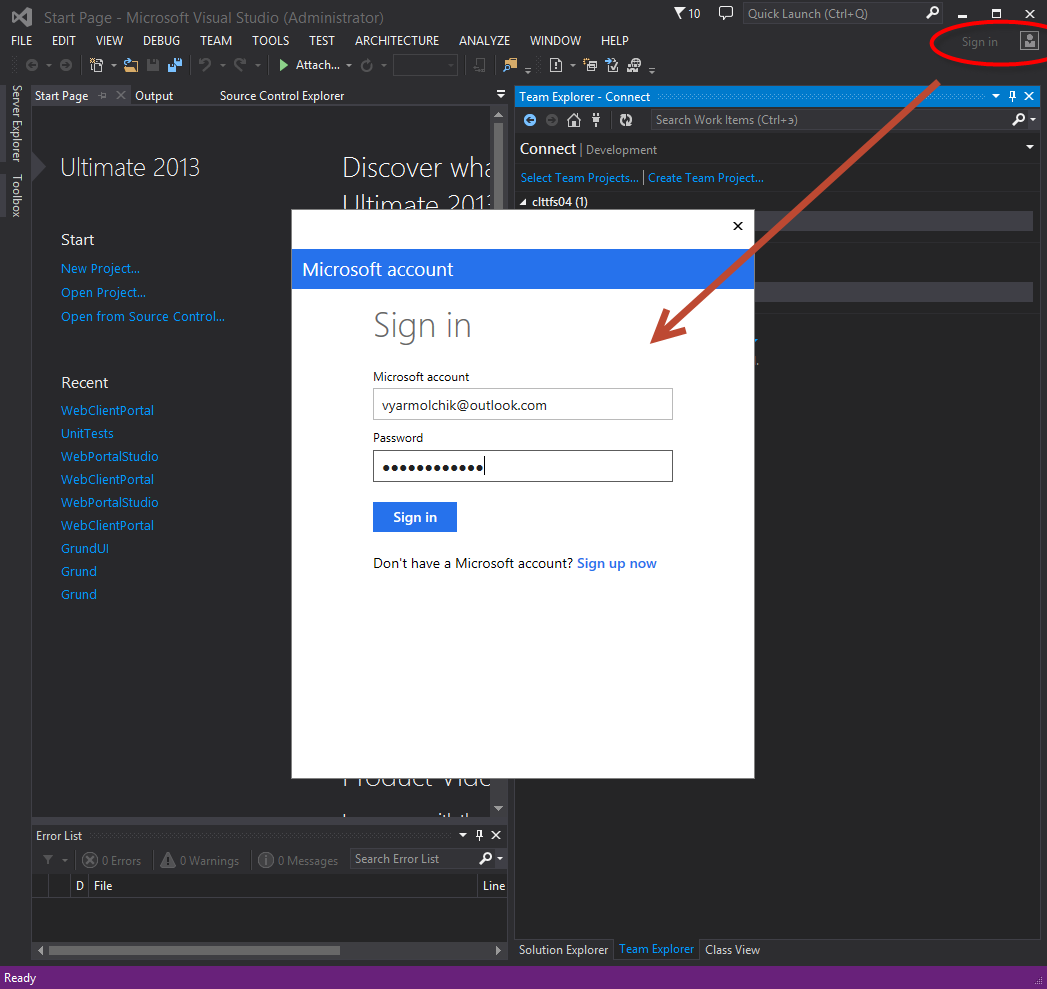
Теперь нужно добавить проект. Придумываем название, выбираем Process Template и Version Control, кликаем на кнопку создания проекта «Create project».



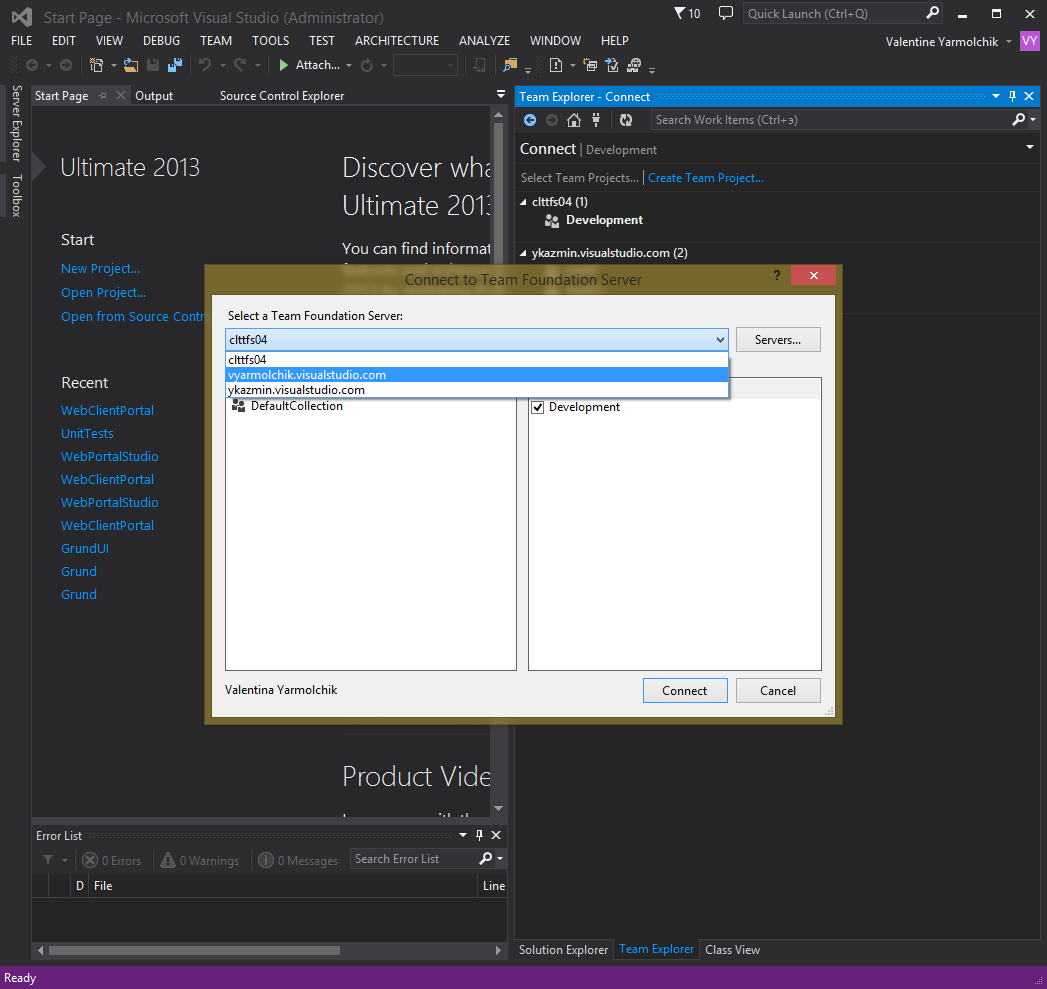
Видим вот такую страничку и переходим к настройке Visual Studio...



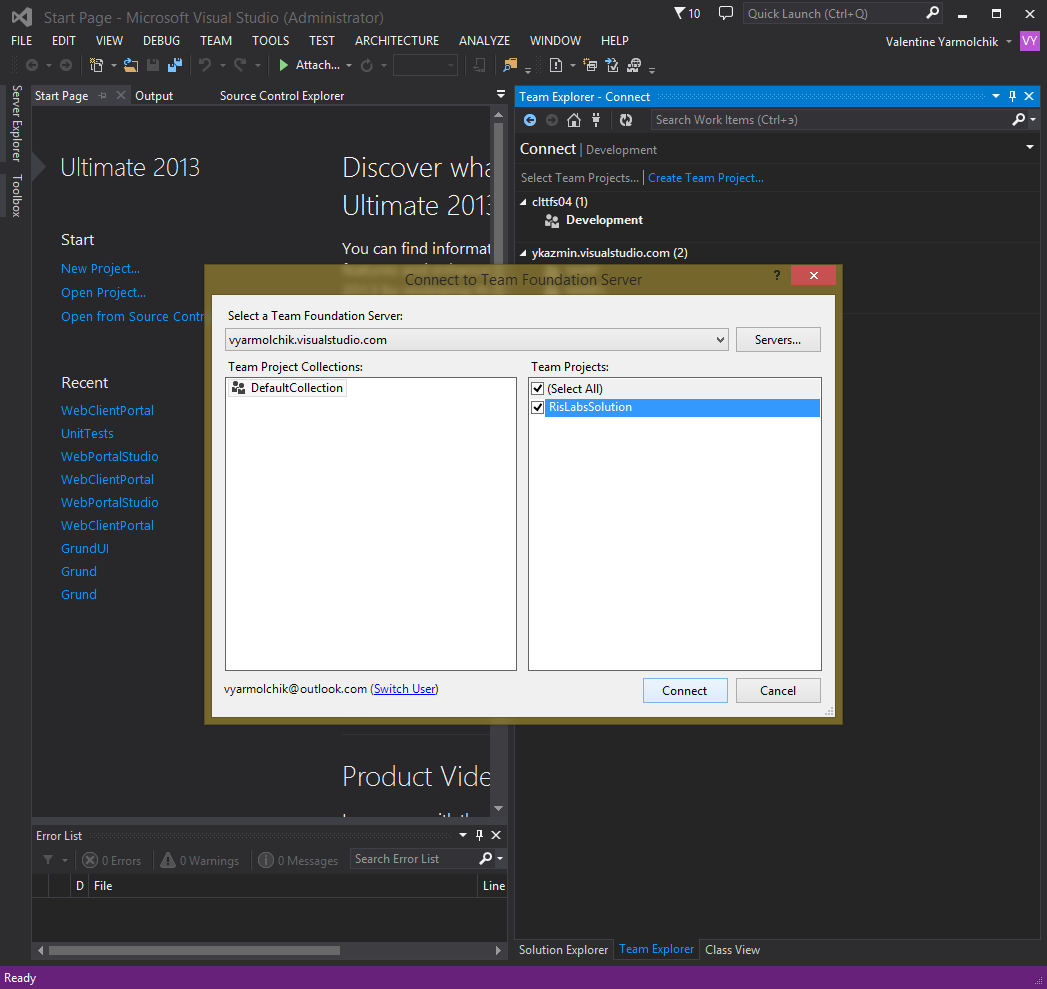
В VS нужно выполнить вход, используя данные своей учетной записи Microsoft.



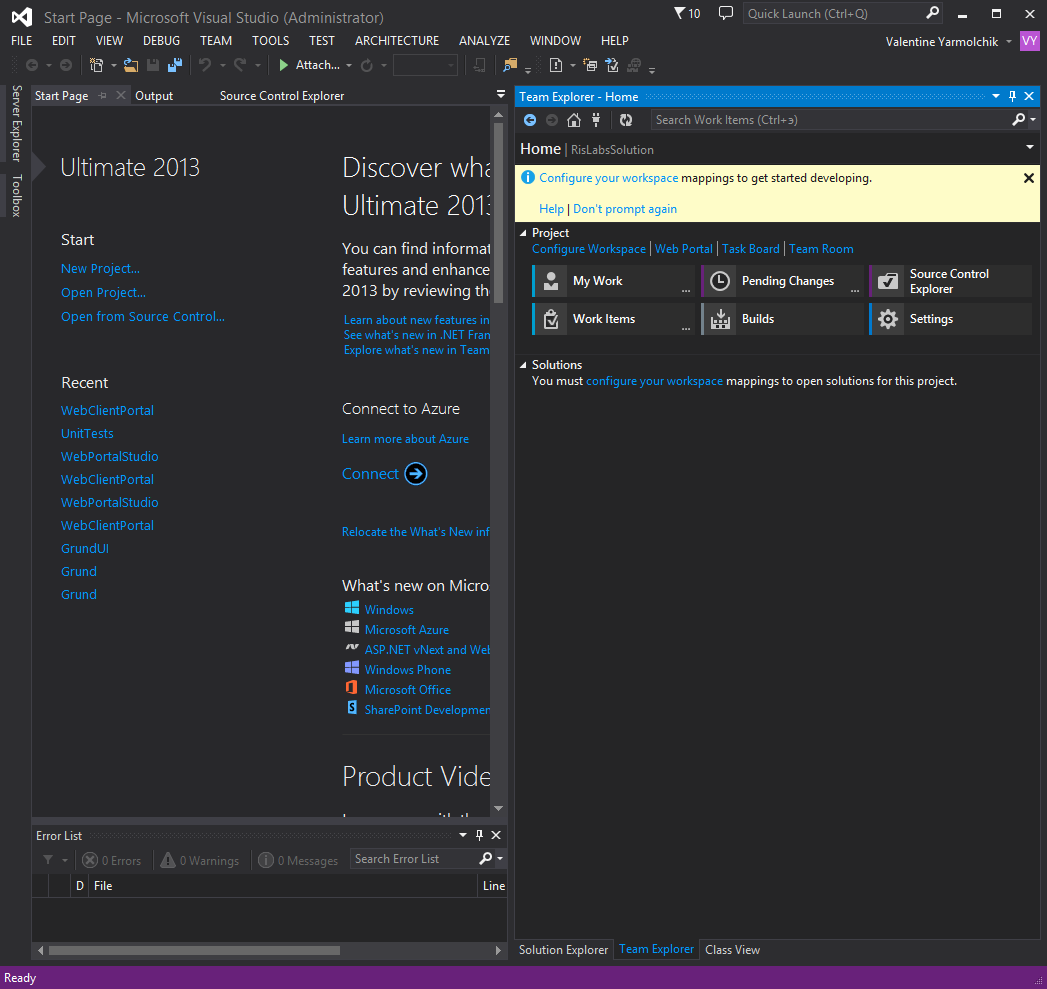
Далее ищем вкладку Team Explorer, нажимаем на ссылку Select Team Projects и в выпадающем списке выбираем нашу учетную запись TFS.



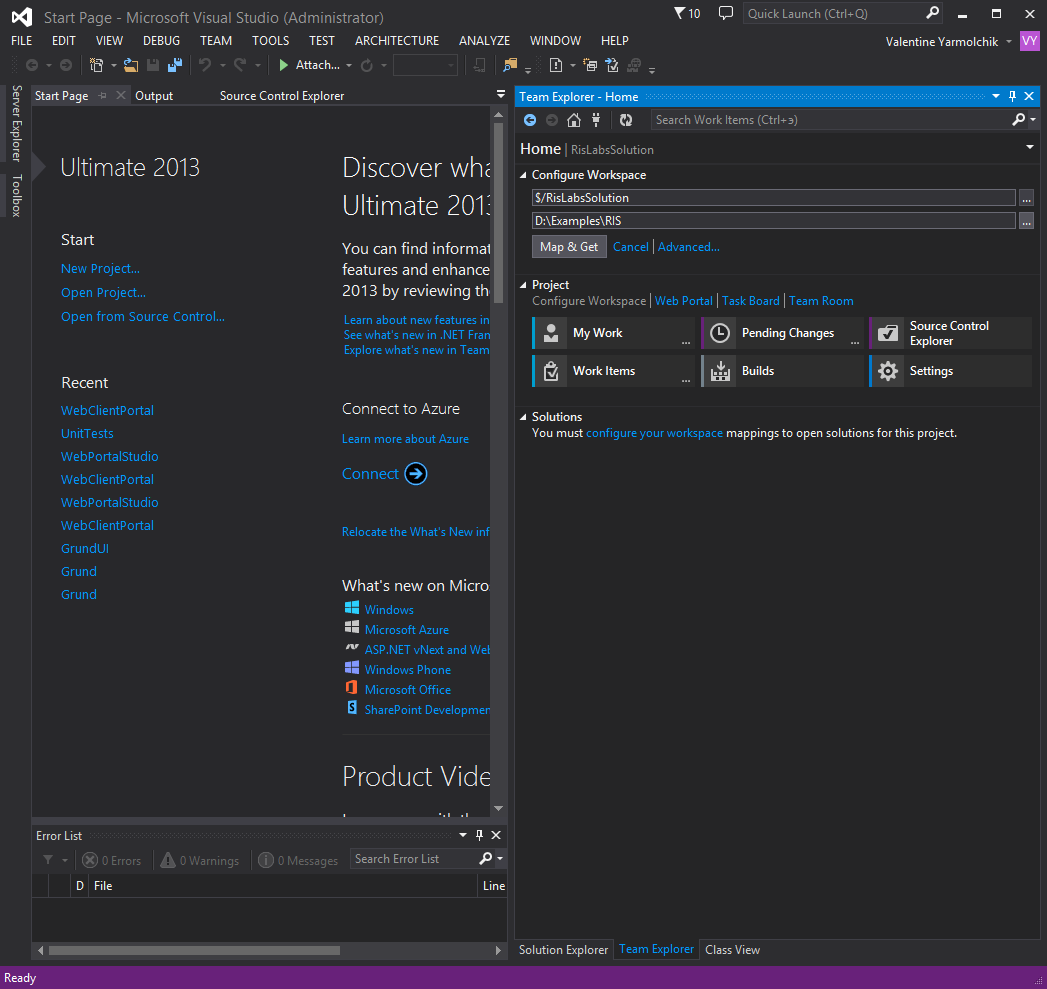
Выбираем наш проект:



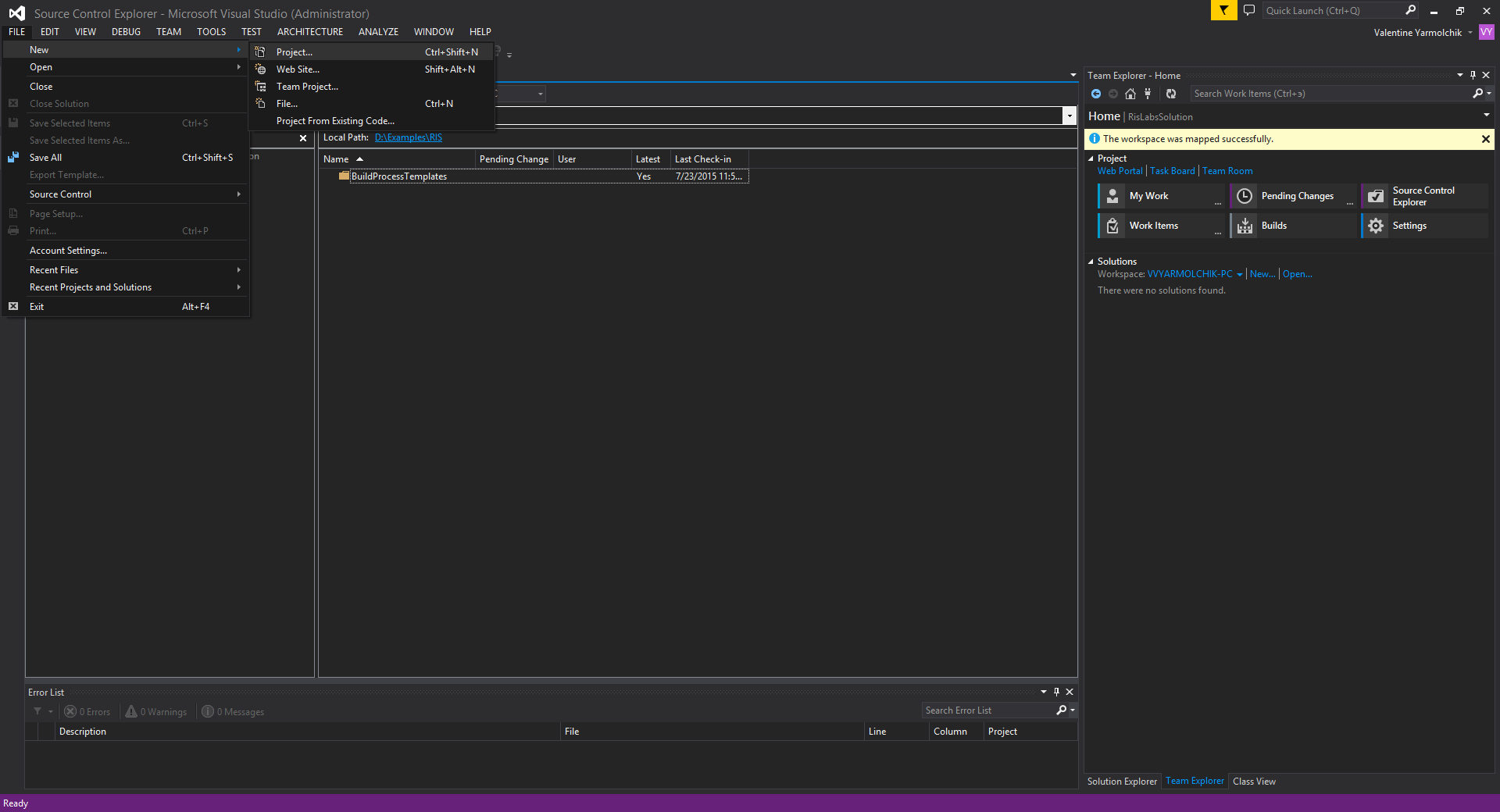
Выбираем ссылку Configure your workspace.



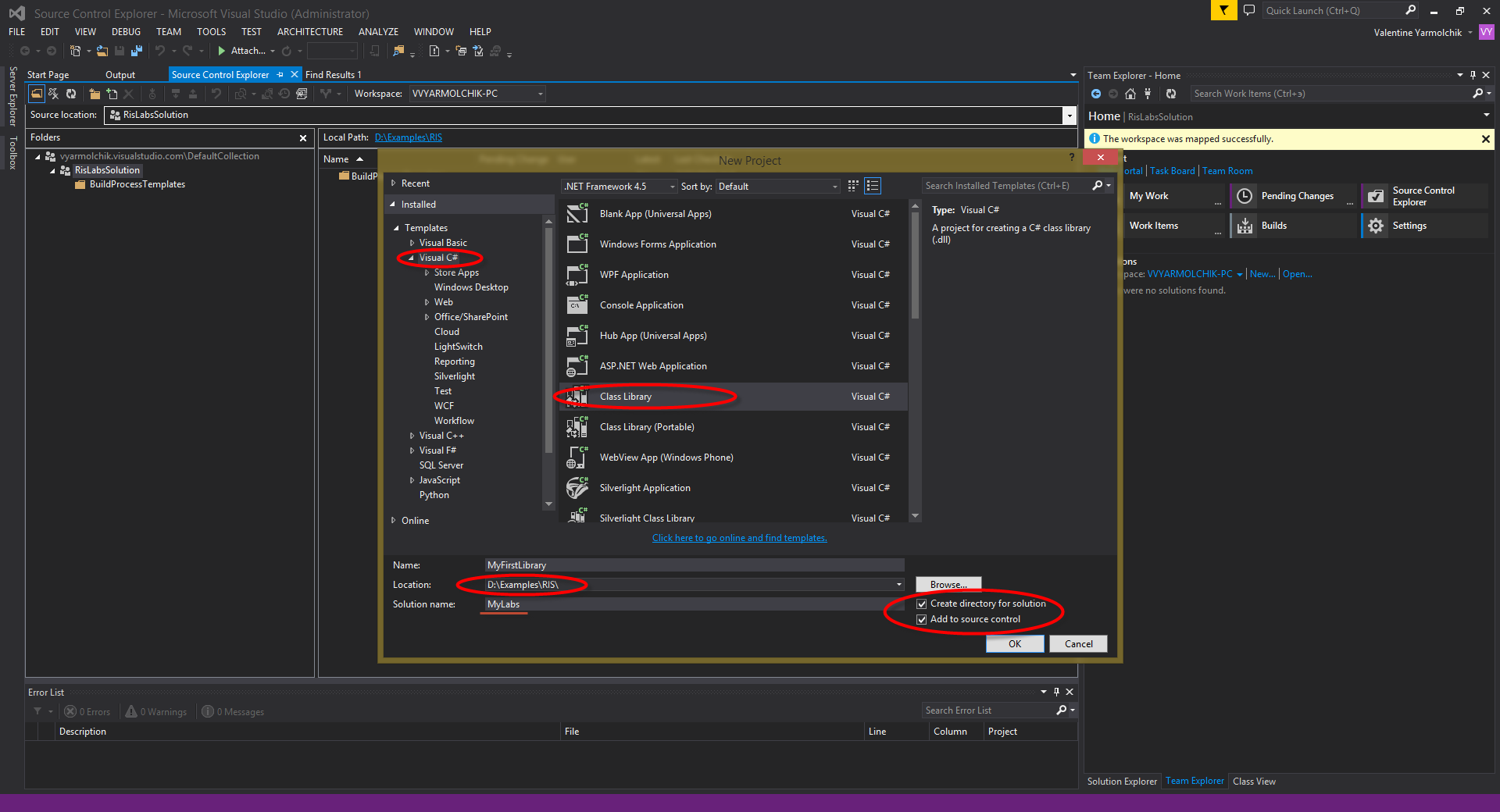
Выбираем папку на компьютере для маппинга вашего проекта:



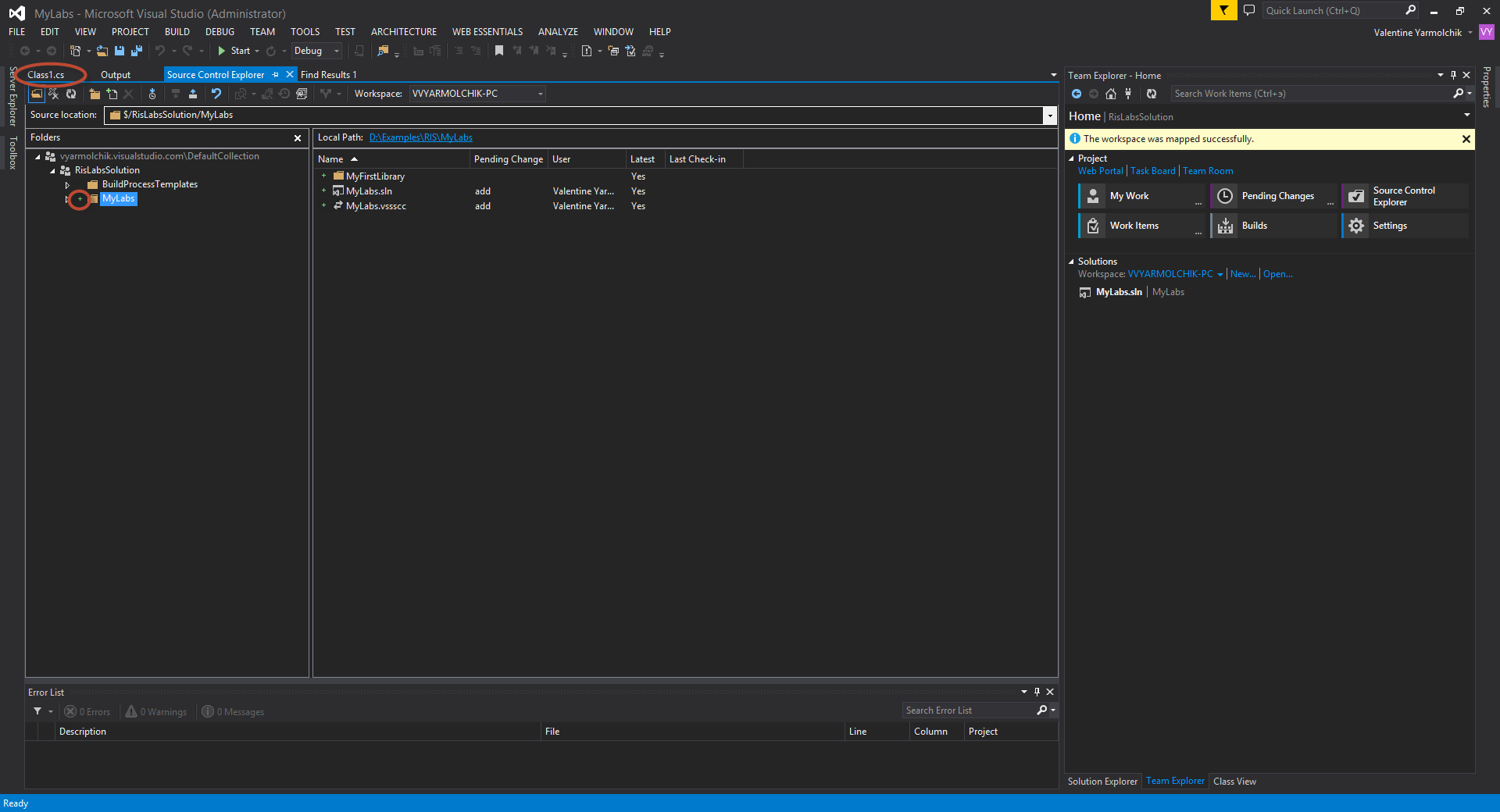
Создадим новый проект в созданной папке.



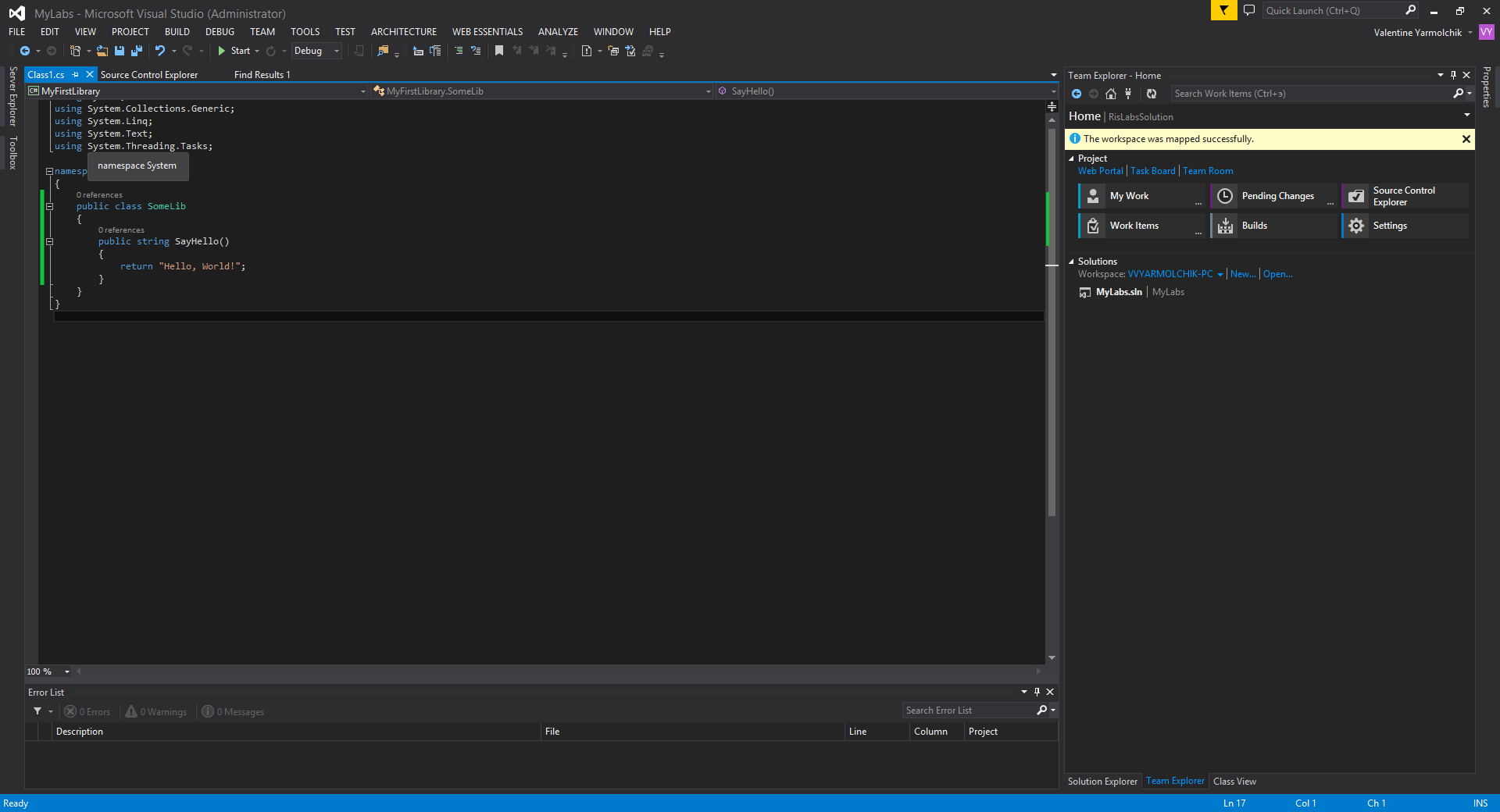
Не забудьте поставить галочки в нужные чекбоксы, выбрать путь для сохранения и заполнить названия проекта и решения.



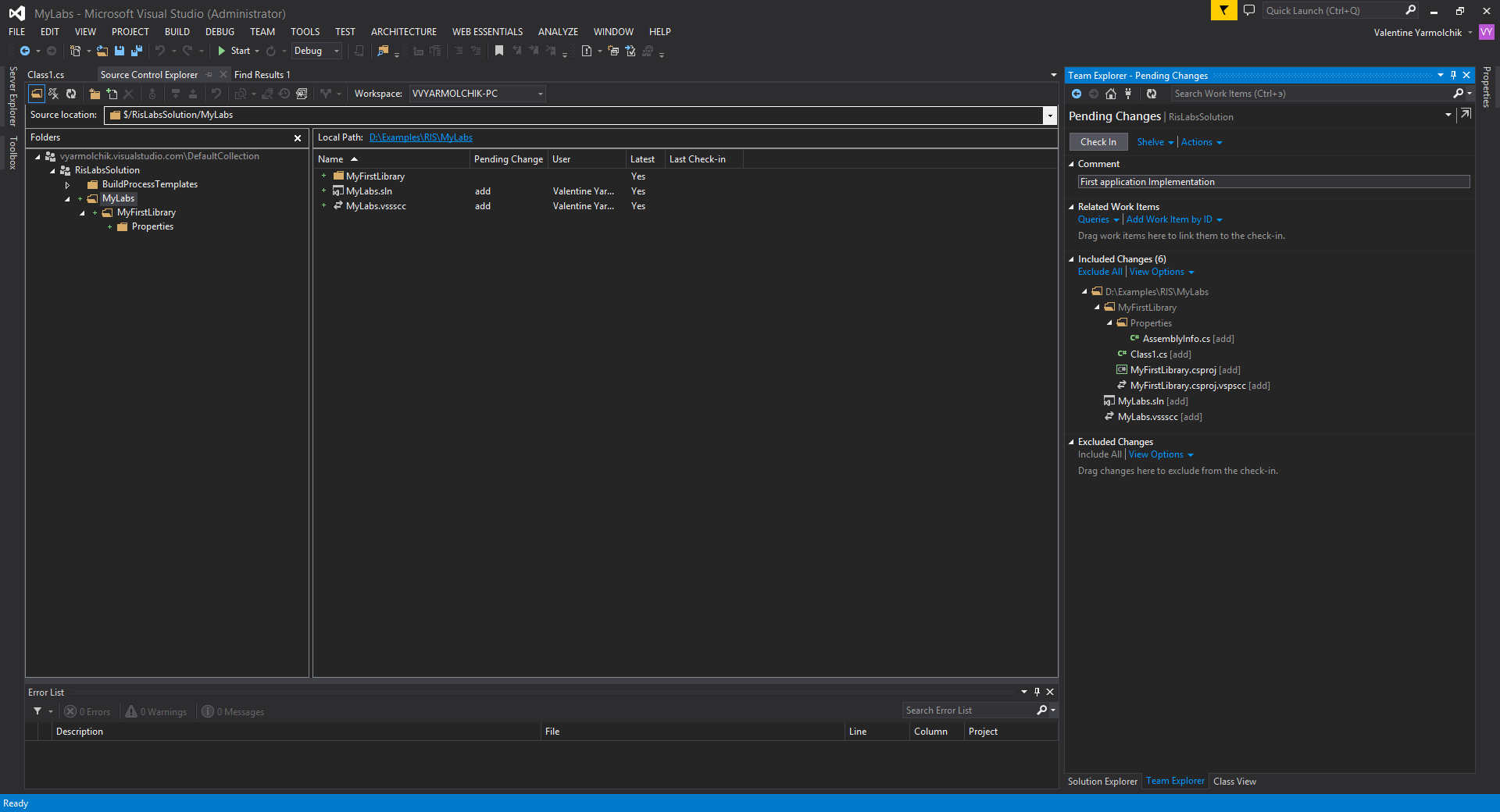
Проект добавился в Source Control Explorer и создался новый класс:

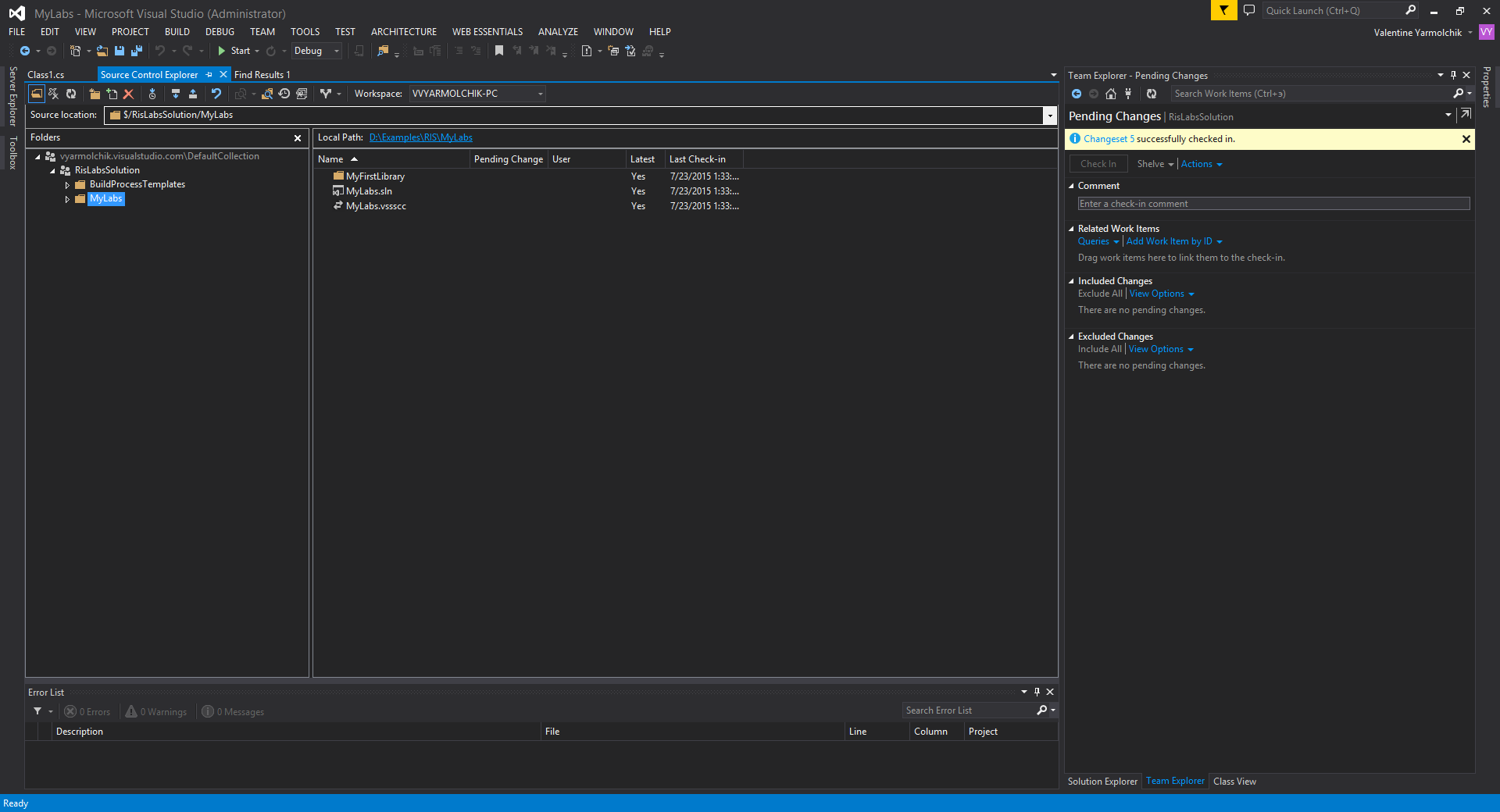


Изменим немного код класса и перейдем к чекину проекта:

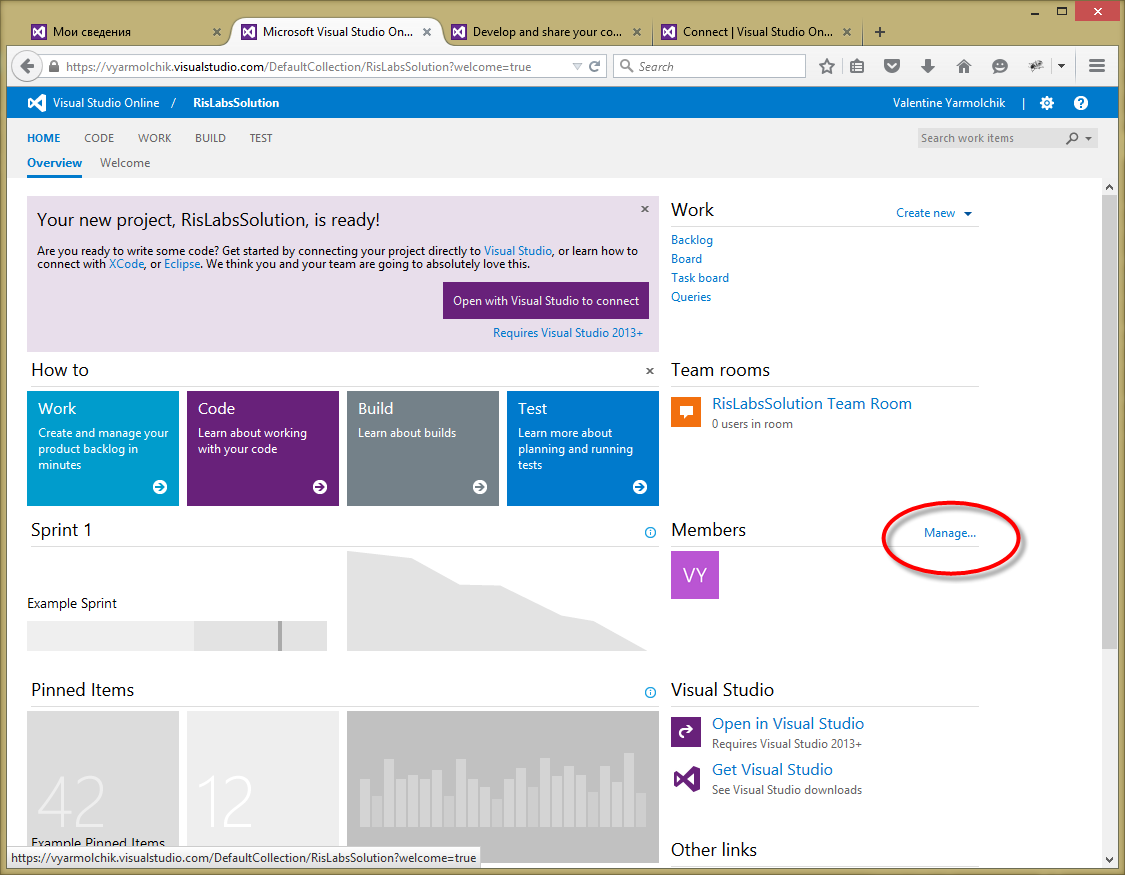


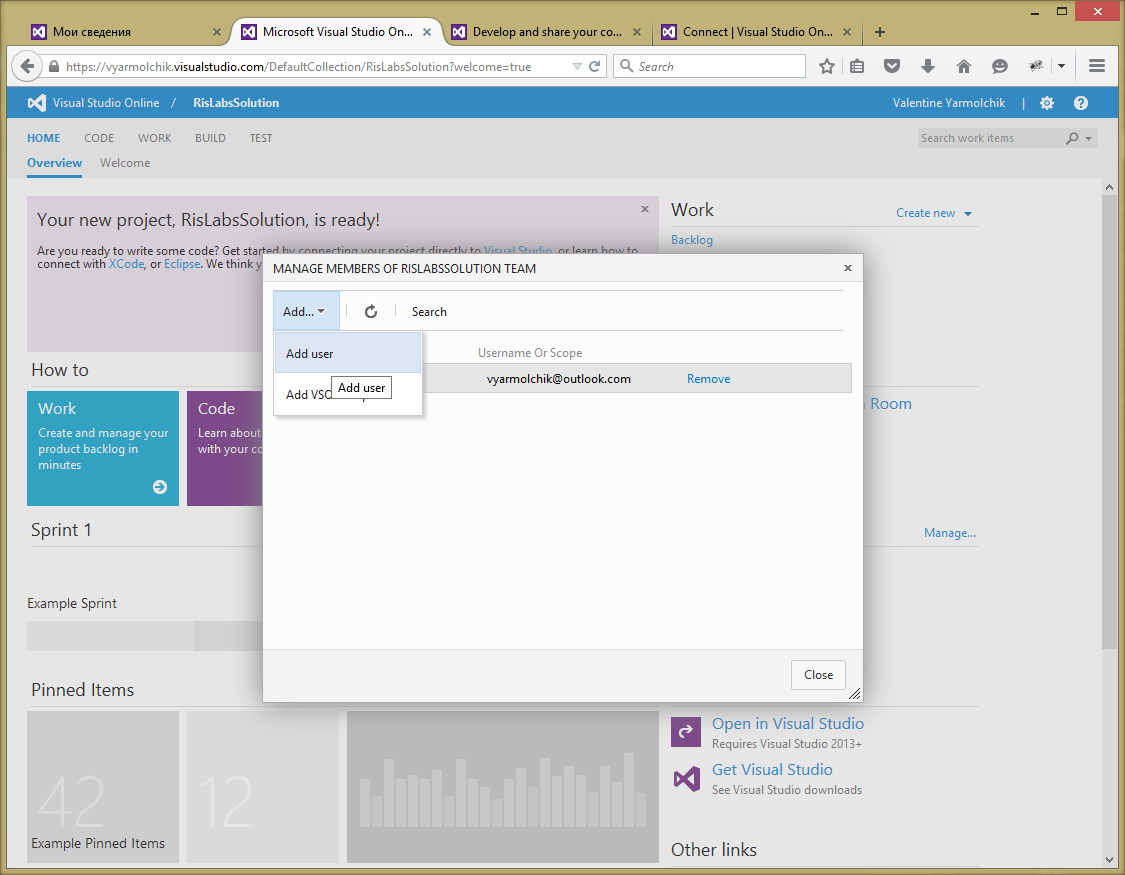
В Team Explorer выбираем вкладку Pending Changes, здесь отображаются файлы, которые были изменены во время работы и требующие сохранения на сервере, добавляем комментарий и нажимаем кнопку Check In:



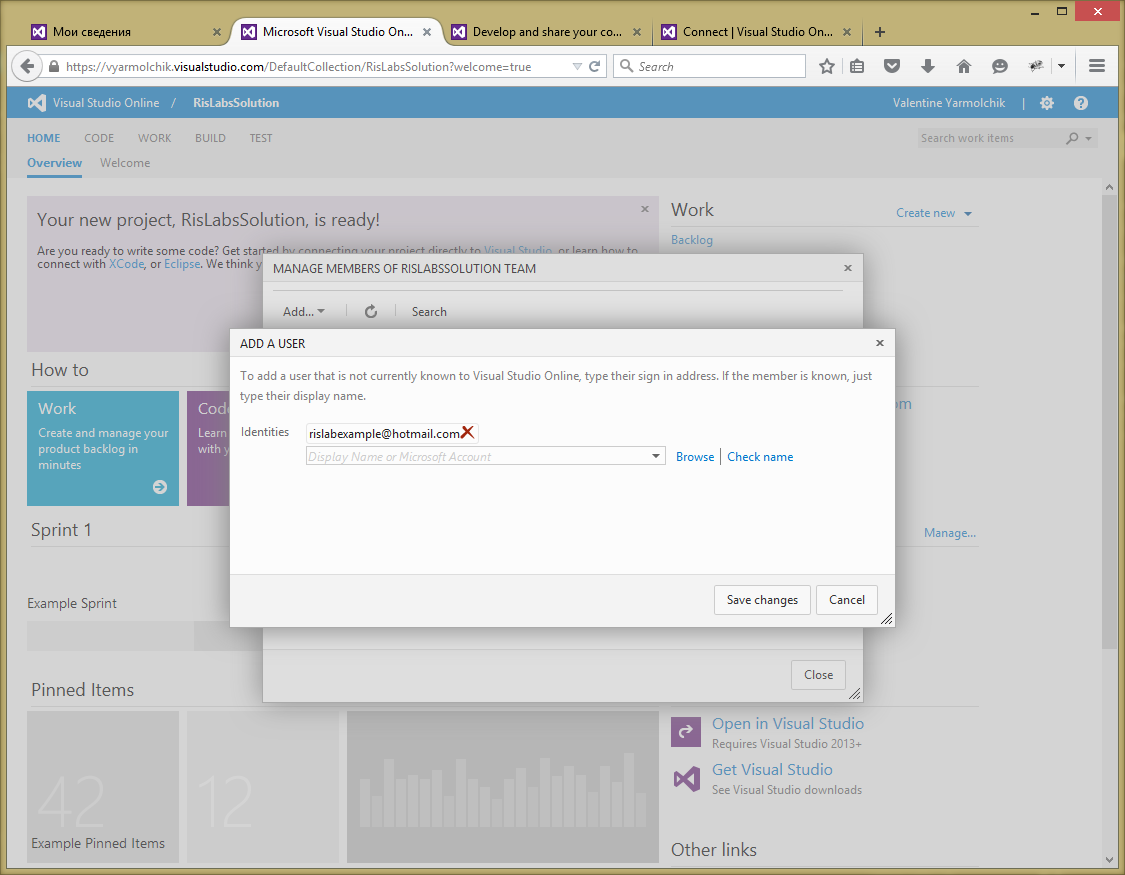


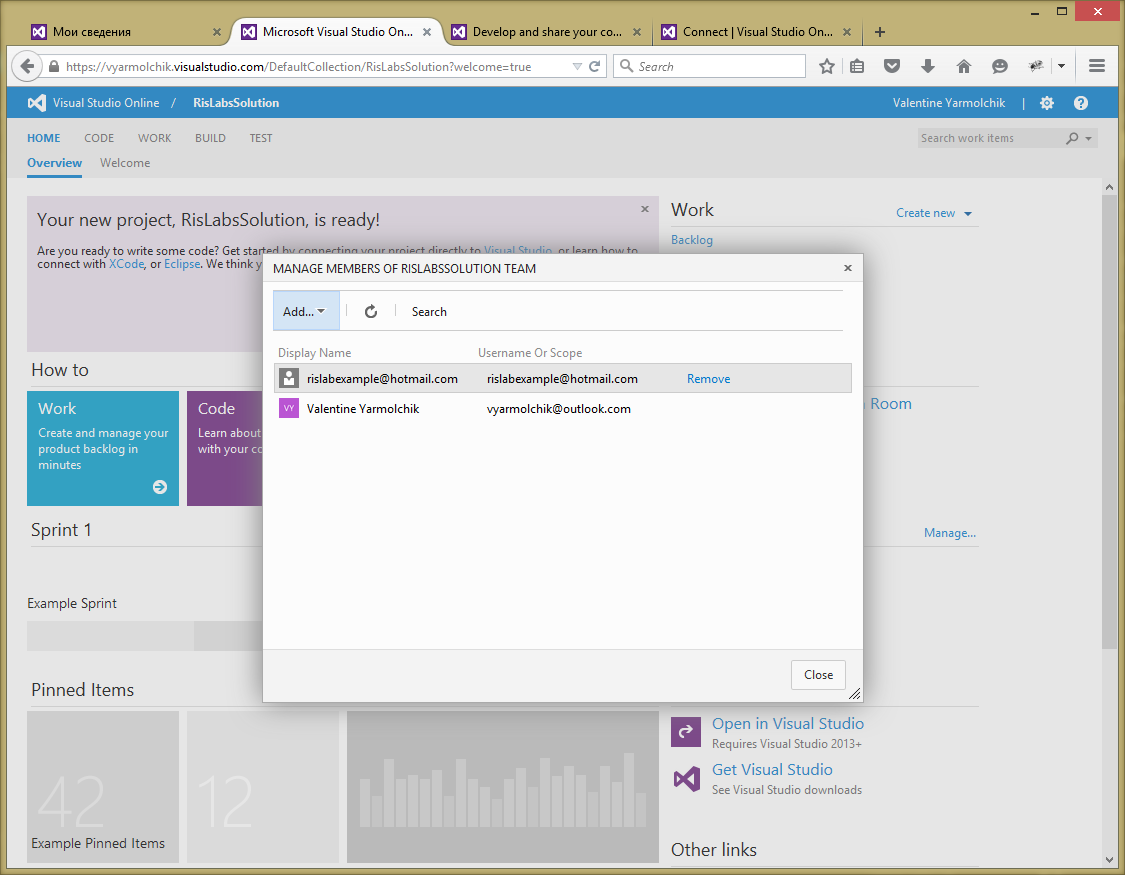
Теперь вернемся на страничку и добавим к проекту участников:



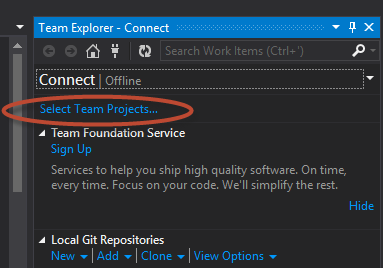


Для добавления вводим аккаунты Microsoft:

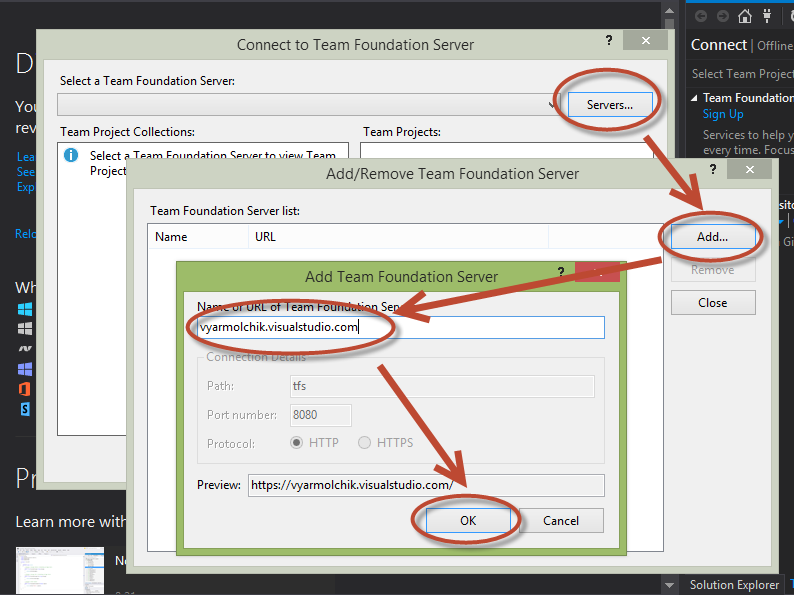




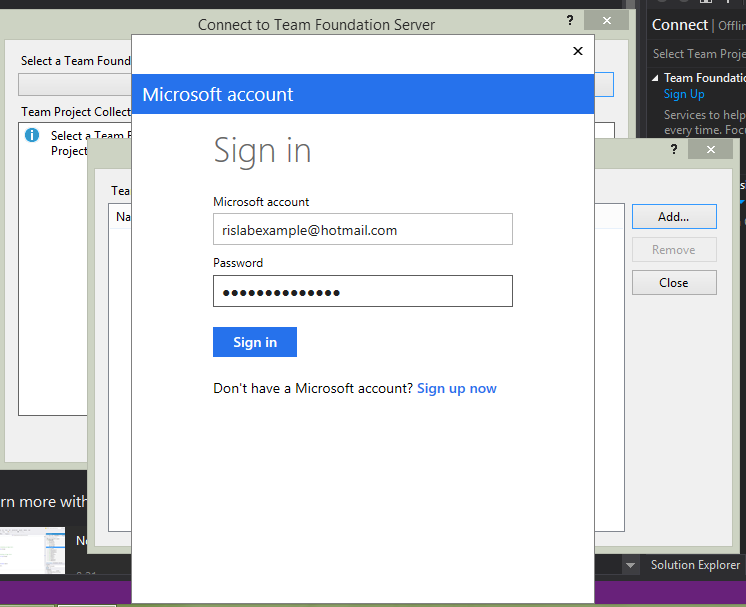
Теперь добавленный участник может настроить Team Explorer в своей Visual Studio. Нажимаем Select Team Projects…



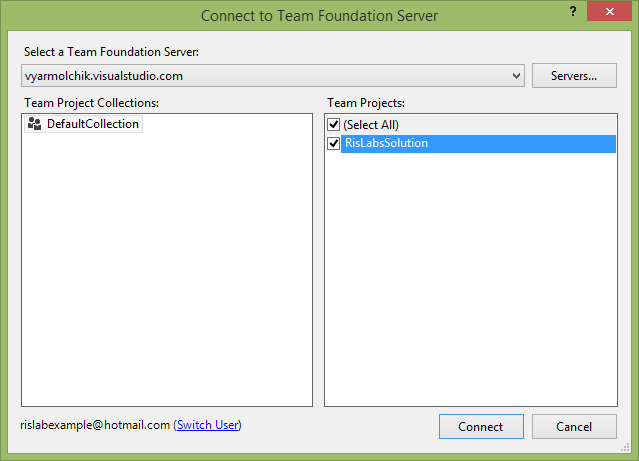
Нужно добавить наш Team Foundation Server, в поле для URL вводим адрес созданной ранее учетной записи владельца проекта.



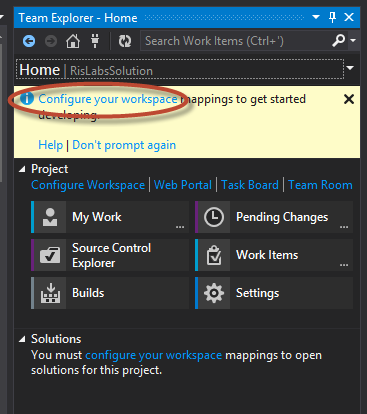
Появится окно авторизации, авторизацию проходит новый добавленный участник, которые хочет получить доступ к проекту



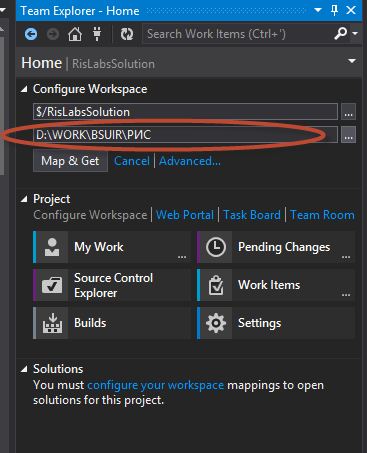
Выбираем нужный проект, в нашем случае в списке только один



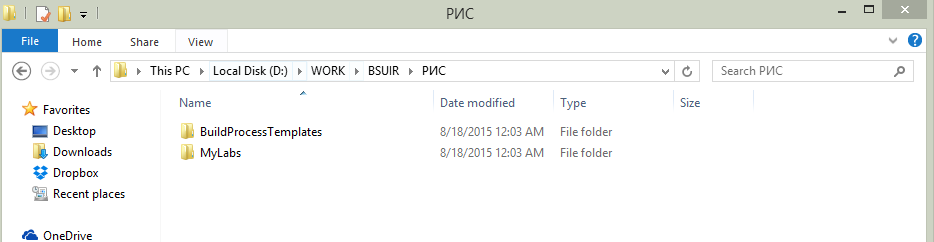
Соглашаемся настроить рабочую область, нажимаем Configure your workspace



Выбираем папку для скачивания и хранения проекта:



Проект скачался, можем открывать и работать. Все изменения вносятся в Тeam Explorer на вкладке Pending Changes.

Теоритическая часть

# Архитектура .NET

В середине 2000 года корпорация Microsoft объявила о работе над новой платформой для создания приложений, которая получила имя платформа .NET (.NET Framework). Платформа .NET образует каркас, включающий библиотеку классов, технологии для доступа к данным и построения оконных и вебприложений. Основным инструментом разработки для платформы .NET является интегрированная среда Microsoft Visual Studio. База платформы .NET - это общеязыковая среда исполнения (Common Language Runtime, CLR). CLR является «прослойкой» между операционной системой и приложением. Приложения для платформы .NET состоят из управляемого кода (managed code). Управляемый код является результатом компиляции исходных текстов. Скомпилированные файлы называются сборками (assembly).

Основная задача CLR – это манипулирование сборками: загрузка, JITкомпиляция, создание окружения для выполнения сборок. Важной функцией CLR является управление памятью при работе приложения и выполнение автоматической сборки мусора, то есть фонового освобождения неиспользуемой памяти. Кроме этого, CLR реализует в приложениях для .NET проверку типов, управление политиками безопасности при доступе к коду и другие функции. В состав платформы .NET входит обширная библиотека классов Framework Class Library (FCL). Элементом этой библиотеки является базовый набор классов Base Class Library (BCL). В BCL входят классы для работы со строками и коллекциями данных, для поддержки многопоточности и множество других классов. Частью FCL являются компоненты, поддерживающие различные технологии обработки данных и организации взаимодействия с пользователем. Это классы для работы с XML и базами данных, для создания пользовательских интерфейсов.

Microsoft .NET поддерживает не только языковую независимость, но и языковую интеграцию. Это означает, что разработчик может наследовать от классов, обрабатывать исключения и использовать преимущества полиморфизма при одновременной работе с несколькими языками. Платформа .NET Framework предоставляет такую возможность с помощью спецификации CTS ( Common Type System — общая система типов), которая полностью описывает все типы данных, поддерживаемые средой выполнения, определяет, как одни типы данных могут взаимодействовать с другими и как они будут представлены в формате метаданных .NET. Например, в .NET любая сущность является объектом какого-нибудь класса, производного от корневого класса System.Object. Спецификация CTS поддерживает такие общие понятия, как классы, делегаты (с поддержкой обратных вызовов), ссылочные и размерные типы.

Важно понимать, что не во всех языках программирования .NET обязательно должны поддерживаться все типы данных, которые определены в CTS. Спецификация CLS (Common Language Specification — общая языковая спецификация ) устанавливает основные правила, определяющие законы, которым должны следовать все языки: ключевые слова, типы, примитивные типы, перегрузки методов и т. п. Спецификация CLS определяет минимальные требования, предъявляемые к языку платформы .NET. Компиляторы, удовлетворяющие этой спецификации, создают объекты, способные взаимодействовать друг с другом. Любой язык, соответствующий требованиям CLS, может использовать все возможности библиотеки FCL (Framework Class Library — библиотека классов платформы). CLS позволяет и разработчикам, и поставщикам, и производителям программного обеспечения не выходить за пределы общего набора правил для языков, компиляторов и типов данных.

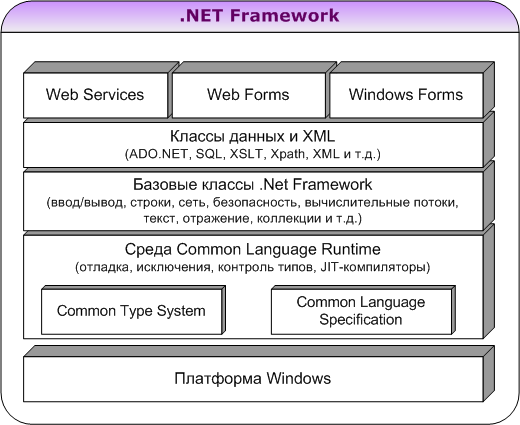


Рисунок 1 – Архитектура .NET Framework

*Преимущества .NET*

**Объектно-ориентированное программирование**. И среда .NET Framework, и С# изначально полностью базировались на объектно-ориентированных прин­ципах.

**Хороший дизайн**. Библиотека базовых классов, которая спроектирована "с нуля", исключительно интуитивно понятным образом.

**Независимость от языка**. Благодаря .NET, код всех языков, то есть VisualBasic .NET, С#, J# управляемого С++, компилируется в общий язык промежуточного уровня — Intermediate Language. Это значит, что все эти языки обладают возмож­ностями взаимодействия, как никогда ранее.

**Лучшая поддержка динамических** **Web-страниц**. Хотя ASP  предлагал высо­кую степень гибкости, он также был и неэффективен из-за своих интерпрети­руемых сценарных языков, а недостаток объектно-ориентированного дизайна часто приводил к запутанному коду ASP. .NET предлагает интегрированную под­держку Web-страниц с применением новой технологии — ASP.NET. В ASP.NET код ваших страниц компилируется и может быть написан на языке высокого уровня, поддерживающего .NET— таком как С# или Visual Basic 2012.

**Эффективный доступ к данным**. Набор компонентов .NET, известный под об­щим названием ADO.NET, предоставляет эффективный доступ к реляционным базам данных и широкому разнообразию других источников данных. Также имеются компоненты, предоставляющие доступ к файловой системе и катало­гам. В частности, в .NET встроена поддержка XML, что позволяет манипули­ровать данными, которые могут быть импортированы их других, не-Windows платформ, и экспортированы на них.

**Разделение кода**. Среда .NET полностью изменила способ разделения кода ме­жду приложениями, введя концепцию сборки (assembly), которая заменила тра­диционные библиотеки DLL. Сборки имеют форматные средства для указания версий, и одновременно в системе могут существовать разные версии одних и тех же сборок.

**Повышенная безопасность**. Каждая сборка также может содержать встроен­ную информацию безопасности, которая в точности описывает, кому и каким категориям пользователей или процессов какие методы каких классов разреше­но вызывать. Это обеспечивает очень высокую степень контроля за тем, как могут использоваться сборки, которые вы поставляете.

**Инсталляция с нулевым воздействием**. Существует два типа сборок: разде­ляемые и нриватные. Разделяемые сборки — это обычные библиотеки, дос­тупные всему программному обеспечению, в то время как приватные сборки предназначены для использования совершенно определенными программами. Приватные сборки полностью самодостаточны, поэтому процесс инсталляции прост. Нет никаких элементов реестра, нужные файлы просто помещаются в

**Поддержка Web-служб**. .NET предлагает полностью интегрированную под­держку разработки Web-служб — все так же просто, как и создание приложений любого другого типа.

**Visual Studio** **2015**. .NET поставляется со средой разработкиVisual Studio 2015, которая одинаково хорошо справляется с языками С++, С#, J# и Visual Basic 2012, а также с ASP.NET. Visual Studio 2015 интегрирует в себе все лучшие средст­ва соответствующих специфичных языковых сред.

**C#**. C# представляет собой новый объектно-ориентированный язык, предназна­ченный для применения с .NET.

# Основы С#

Пример кода:

using System;

namespace Bsuir

{

public class MyFirstClass

{

static void Main()

{

Concole.WriteLine(“Hello from BSUIR”);

Console.ReadLine();

return;

}

}

}

Чтобы скомпилировать эту программу, запустите компилятор командной строки C# (csc.exe)и передайте ему имя исходного файла:

csc First.cs

В C# имеются две категории типов: **типы значений и ссылочные типы**. Разница между ними в том, что тип значения хранит данные непосредственно, в то время как ссылочный тип хранит ссылку на значение. Эти типы сохраняются в разных местах памяти: типы значений хранятся в области, известной как стек, а ссылочные типы – в области, которая называется управляемой кучей.

C# поддерживает следующие встроенные типы данных.

|  |  |
| --- | --- |
| **Тип данных** | **Диапазон** |
| **byte** | 0 .. 255 |
| **sbyte** | -128 .. 127 |
| **short** | -32,768 .. 32,767 |
| **ushort** | 0 .. 65,535 |
| **int** | -2,147,483,648 .. 2,147,483,647 |
| **uint** | 0 .. 4,294,967,295 |
| **long** | -9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807 |
| **ulong** | 0 .. 18,446,744,073,709,551,615 |
| **float** | -3,402823e38 .. -3,402823e38 .. |
| **double** | -1,79769313486232e308 .. 1,79769313486232e308 |
| **decimal** | -79228162514264337593543950335 .. 79228162514264337593543950335 |
| **char** | Символ Юникода. |
| **string** | Строка символов Юникода. |
| **bool** | true или false |
| **object** | Объект. |

Эти типы данных называются псевдонимами для предопределенных типов в пространстве имен [System](https://msdn.microsoft.com/ru-ru/library/system(v=vs.90).aspx). Все эти типы, за **исключением** object и string, являются типами значений.

**Выведения типа из контекста** осуществляется с помощью ключевого слова var. Компилятор «выводит» тип переменной на основе того, как она инициализируется.

int number = 0; var number = 0;

В программировании на C# пространства имен используются с полной нагрузкой по двум направлениям. Во-первых, платформа .NET Framework использует пространства имен для организации большинства классов. Это выполняется следующим образом.

System.Console.WriteLine("Hello World!");

System — это пространство имен, а Console — класс в нем. Использование ключевого слова using может отменить необходимость полного имени, как показано в следующем примере.

Во-вторых, объявление собственного пространства имен поможет в управлении областью действия имен классов и методов в крупных программных проектах.

Пространства имен имеют следующие свойства.

* Организация крупных проектов по созданию кода.
* Для их разделения используются оператор ..
* Директива using исключает требование на указание имени пространства имен для каждого класса.
* Пространство имен global является корневым пространством имен: global::System всегда будет ссылаться на пространство имен платформы .NET Framework System.

# Классы и структуры. Частичные классы

Класс является основным пользовательским типом. Синтаксис объявления класса в C# следующий:

<модификаторы> class <имя класса> { [<элементы класса>] }

Класс может содержать следующие элементы:

* **Поле** - это переменная любого типа, которая объявлена непосредственно в классе.
* **Константа** - это переменная, значение которой нельзя изменить. Константы используются для гарантирования того, что данные в этой переменной не изменятся. Для того, чтобы объявить константу, перед обычным объявлением переменной нужно добавить ключевое слово const.
* **Метод** представляет собой блок кода, содержащий набор инструкций.
* **Свойство** - это член, предоставляющий гибкий механизм для чтения, записи или вычисления значения частного (private) поля. Свойства можно использовать, как если бы они являлись открытыми членами данных, хотя в действительности они являются специальными методами, называемыми методами доступа. Это обеспечивает простой доступ к данным и позволяет повысить уровень безопасности и гибкости методов.

Базовый синтаксис описания свойства:

<модификаторы> <тип свойства> <имя свойства>

{

get {<блок кода>}

set {<блок кода>}

}

Пример:

public class Student

{

private int age;

private string name;

public int Age

{

get { return age; }

set { age = value; }

}

public string Name

{

get { return “My name is” + name; }

set { name = value; }

}

public string Faculty { get; set; }

}

Свойства транслируются при компиляции в вызовы методов. В скомпилированный код класса добавляются методы со специальными именами get\_Name() и set\_Name(), где Name – это имя свойства. Побочным эффектом такой трансляции является тот факт, что пользовательские методы с данными именами допустимы в классе, только если они имеют сигнатуру, отличающуюся от методов, соответствующих свойству.

* **Индексатор** - это свойство-коллекция, отдельный элемент которого доступен по индексу.
* **Конструктор**.
* **Деструктор**.
* **Событие**
* **Операция** - язык C# допускает перегрузку некоторых операций для объектов класса.
* **Вложенный пользовательский тип** - тип, определенный внутри класса.

Все типы и члены типов имеют уровень доступности, который определяет возможность их использования из другого кода в сборке разработчика или других сборках. Можно использовать следующие модификаторы доступа для указания доступности типа или члена при объявлении этого типа или члена.

[**public**](http://msdn.microsoft.com/ru-ru/library/yzh058ae.aspx)

Доступ к типу или члену возможен из любого другого кода в той же сборке или другой сборке, ссылающейся на него.

[**private**](http://msdn.microsoft.com/ru-ru/library/st6sy9xe.aspx)

Доступ к типу или члену можно получить только из кода в том же классе или структуре.

[**protected**](http://msdn.microsoft.com/ru-ru/library/bcd5672a.aspx)

Доступ к типу или элементу можно получить только из кода в том же классе или структуре, либо в производном классе.

[**internal**](http://msdn.microsoft.com/ru-ru/library/7c5ka91b.aspx)

Доступ к типу или члену возможен из любого кода в той же сборке, но не из другой сборки.

**protected internal**

Доступ к типу или элементу может осуществляться любым кодом в сборке, в которой он объявлен, или из наследованного класса другой сборки. Доступ из другой сборки должен осуществляться в пределах объявления класса, производного от класса, в котором объявлен защищенный внутренний элемент, и должен происходить через экземпляр типа производного класса.

Имеется возможность разделить определение [класса](http://msdn.microsoft.com/ru-ru/library/0b0thckt.aspx) или [структуры](http://msdn.microsoft.com/ru-ru/library/ah19swz4.aspx), [интерфейса](http://msdn.microsoft.com/ru-ru/library/87d83y5b.aspx) или метода между двумя или более исходными файлами. Каждый исходный файл содержит определение типа или метода, и все части объединяются при компиляции приложения.

public partial class Student

{

public void DoWork() {}

}

public partial class Student

{

public void GoToSleep() {}

}

Ключевое слово **partial** указывает на то, что другие части класса, структуры или интерфейса могут быть определены в пространстве имен. Все части должны использовать ключевое слово **partial**. Для формирования окончательного типа все части должны быть доступны во время компиляции. Все части должны иметь одинаковые модификаторы доступности, например **public**, **private** и т.д.

# Делегаты

**Делегат** – это пользовательский тип, который инкапсулирует метод. В C# делегат объявляется с использованием ключевого слова *delegate*. При этом указывается имя делегата, тип и сигнатура инкапсулируемого метода:

public delegate string JustFunction(string s);

public delegate void SecondFunc(int i);

После объявления делегата можно объявить переменные этого типа:

JustFunction F;

SecondFunc S;

Переменные делегата инициализируются конкретными методами при использовании конструктора делегата с одним параметром – именем метода (или именем другого делегата). Если делегат инициализируется статическим методом, требуется указать имя класса и имя метода, для инициализации экземплярным методом указывается объект и имя метода. При этом метод должен обладать подходящей сигнатурой:

F = new JustFunction(ClassName.SomeStaticFunction);

S = new SecondFunc (obj.SomeInstanceMethod);

Для инициализации делегата можно использовать упрощенный синтаксис – достаточно указать имя метода без применения new().

F = ClassName.SomeStaticFunction;

S = obj.SomeInstanceMethod;

После того как делегат инициализирован, инкапсулированный в нем метод вызывается с указанием аргументов метода непосредственно после имени переменной-делегата: F(“Я аргумент.”);.

# Универсальные шаблоны

Универсальные шаблоны (generics) позволяют при разработке пользовательского типа или метода указать в качестве параметра тип, который конкретизируется при использовании. Универсальные шаблоны применимы к классам, структурам, интерфейсам, делегатам и методам.

Для описания универсального типа используется следующий синтаксис: после имени класса в угловых скобках указывается параметр типа, допускается использовать несколько параметров-типов.

public class List<T>

{

private T[] \_items;

public void Add(T item) {. . .}

public T GetItem(int i) {. . .}

}

C# допускает указание ограничения (constraint) для каждого параметра универсального типа. Только тип, удовлетворяющий ограничениям, может быть применѐн для записи сконструированного типа.

Ограничения объявляются с использованием ключевого слова where, после которого указывается параметр, двоеточие и список ограничения. Элементом списка ограничения на тип могут являться:

Ключевое слово class (требование, чтобы тип был ссылочным) или ключевое слово struct (требование, чтобы тип был типом значения).

Имя класса (требование, чтобы тип приводился к этому классу).

Интерфейс или список интерфейсов (требование, чтобы тип реализовывал эти интерфейсы).

Конструкция new() (требование, чтобы у типа был конструктор без параметров).

Порядок элементов в списке ограничений имеет значение. Правильный порядок соответствует порядку в списке, приведенном выше. В следующем примере используется несколько ограничений на различные параметры универсального типа (предполагается, что интерфейс IComparable<K> содержит метод CompareTo()):

public class EntityTable<K, E>

where K : IComparable<K>, IPersistable

where E : Entity, new()

{

public void Add(K key, E entity) {

. . .

if (key.CompareTo(x) < 0) { . . . }

. . .

}

}

# Коллекции

Платформа .NET включает большой набор типов для предоставления стандартных коллекций - списков, множеств, словарей. Эти типы можно разделить на несколько категорий: базовые интерфейсы и вспомогательные классы, классы для коллекций-списков и словарей, набор классов для построения собственных коллекций.

Опишем набор интерфейсов, реализуемых практически всеми типами коллекций. Основу набора составляют интерфейсы IEnumerable<T> и IEnumerable. Они отражают фундаментальное свойство любой коллекции – возможность перечислить еѐ элементы. Слаботипизированные словари реализуют интерфейс IDictionaryEnumerator для перебора своих пар «ключ-значение».

ICollection – это интерфейс для коллекций, запоминающих число хранимых элементов. Интерфейс определяет свойство Count, а также метод для копирования коллекции в массив и свойства для синхронизации коллекции при многопоточном использовании.

Универсальный интерфейс ICollection<T> также поддерживает свойство для количества элементов. Кроме этого, он предоставляет методы для добавления и удаления элементов, копирования элементов в массив, поиска элемента.

Интерфейсы Ilist и IList<T> описывают набор данных, которые проецируются на массив. Дополнительно к функциональности, унаследованной от IEnumerable и ICollection, интерфейс IList позволяет обращаться к элементу по индексу, добавлять, удалять и искать элементы.

Интерфейсы IDictionary и IDictionary<TKey, TValue> определяют протокол взаимодействия для коллекций-словарей (KeyValuePair<TKey, TValue> - это вспомогательная структура, у которой определены свойства Key и Value).

Для работы с коллекциями-множествами предназначен интерфейс ISet<T>. Его набор методов отражает типичные операции для множеств.

# Язык интегрированных запросов

Платформа .NET версии 3.5 представила новую технологию работы с коллекциями - Language Integrated Query (LINQ). По типу обрабатываемой информации LINQ делится на LINQ to Objects – библиотеки для обработки коллекций объектов в памяти; LINQ to SQL – библиотеки для работы с базами данных; LINQ to XML предназначена для обработки XML-информации. В данном параграфе акцент сделан LINQ to Objects. Технически, LINQ to Objects – это набор классов, содержащих типичные методы обработки коллекций: поиск данных, сортировка, фильтрация. Ядром LINQ to Objects является статический класс Enumerable, размещенный в пространстве имен System.Linq1. Этот класс содержит набор методов расширения интерфейса IEnumerable<T>, которые в дальнейшем будут называться операторами LINQ. Стандартное деление операторов LINQ на группы в зависимости от выполняемых действий:

1. Оператор условия Where (отложенные вычисления).

**public** **void** Linq1()

    {

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

        var lowNums =

            from n **in** numbers

            where n < 5

            select n;

        Console.WriteLine("Numbers < 5:");

**foreach** (var x **in** lowNums)

        {

            Console.WriteLine(x);

        }

    } //находим все элементы массива < 5.

1. Операторы проекций (отложенные вычисления).
2. **public** **void** Linq7()
3. {
4. List<Product> products = GetProductList();
6. var productNames =
7. from p **in** products
8. select p.ProductName;
10. Console.WriteLine("Product Names:");
11. **foreach** (var productName **in** productNames)
12. {
13. Console.WriteLine(productName);
14. }
15. } // возвращаем только список названий из списка продуктов
16. Операторы упорядочивания (отложенные вычисления).

**public** **void** Linq30()

{

    List<Product> products = GetProductList();

    var sortedProducts =

        from p **in** products

        orderby p.ProductName

        select p;

    ObjectDumper.Write(sortedProducts);

} // упорядочиваем список продуктов по названию

1. Оператор группировки GroupBy (отложенные вычисления).

**public** **void** Linq41()

    {

**string**[] words = { "blueberry", "chimpanzee", "abacus", "banana", "apple", "cheese" };

        var wordGroups =

            from w **in** words

            group w by w[0] into g

            select **new** { FirstLetter = g.Key, Words = g };

**foreach** (var g **in** wordGroups)

        {

            Console.WriteLine("Words that start with the letter '{0}':", g.FirstLetter);

**foreach** (var w **in** g.Words)

            {

                Console.WriteLine(w);

            }

        }

    } // разделяем список слов по их первой букве

1. Операторы соединения (отложенные вычисления).

**public** **void** Linq102()

{

**string**[] categories = **new** **string**[]{

        "Beverages",

        "Condiments",

        "Vegetables",

        "Dairy Products",

        "Seafood" };

    List<Product> products = GetProductList();

    var q =

        from c **in** categories

        join p **in** products on c equals p.Category

        select **new** { Category = c, p.ProductName };

**foreach** (var v **in** q)

    {

        Console.WriteLine(v.ProductName + ": " + v.Category);

    }

}

// В этом примере показано, как эффективно присоединиться элементами двух

// последовательностей на основе равенства между ключевыми выражениями над ними.

1. Операторы работы с множествами (отложенные вычисления).

**public** **void** Linq48()

{

**int**[] numbersA = { 0, 2, 4, 5, 6, 8, 9 };

**int**[] numbersB = { 1, 3, 5, 7, 8 };

    var uniqueNumbers = numbersA.Union(numbersB);

    Console.WriteLine("Unique numbers from both arrays:");

**foreach** (var n **in** uniqueNumbers)

    {

        Console.WriteLine(n);

    }

} // Используем Union, чтобы получить последовательность из всех элементов   
 // каждого массива

1. Операторы агрегирования.

**public** **void** Linq84()

{

    List<Product> products = GetProductList();

    var categories =

        from p **in** products

        group p by p.Category into g

        let minPrice = g.Min(p => p.UnitPrice)

        select **new** { Category = g.Key, CheapestProducts = g.Where(p => p.UnitPrice == minPrice) };

    ObjectDumper.Write(categories, 1);

} // Используем min для получения продуктов с наименьшей ценой по каждой категории

8. Операторы генерирования (отложенные вычисления).

9. Операторы кванторов и сравнения.

10. Операторы разбиения (отложенные вычисления).

11. Операторы элемента.

12. Операторы преобразования.

(Больше примеров <https://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b>)

# Работа с файлами, консолью

Пространство имен System.IO содержит все необходимые классы, методы и свойства для манипуляций с каталогами и файлами.

|  |  |
| --- | --- |
| Класс | Применение |
| BinaryReader и BinaryWriter | Чтение и запись простых типов данных |
| Directory, File, DirectoryInfo и FileInfo | Создание, удаление и перемещение файлов и директорий. Получение подробной информации о файлах, при помощи свойств, определенных в этих классах. |
| FileStream | Доступ к файлам потоковым способом |
| MemoryStream | Доступ к данным храняшимся в памяти |
| StreamWriter и StreamReader | Чтение и запись текстовой информации |
| StringReader и StringWriter | Чтение и запись текстовой информации из строкового буфера |

### Работа с классами DirectoryInfo и FileInfo

Класс DirectoryInfo содержит методы для создания, перемещение и удаление каталогов. Чтобы использовать вышеприведённые свойства, необходимо создать объект класса DirectoryInfo как показано в примере:

DirectoryInfo dir1 = new DirectoryInfo(@"F:\WINNT");

После этого уже можно просмотреть свойства директории при помощи объекта dir1, как показано на фрагмент кода:

Console.WriteLine("Full Name is : {0}", dir1.FullName);

Console.WriteLine("Attributes are : {0}",

dir1.Attributes.ToString());

*Пример создания консольного приложения представлен на рисунках 2.1 – 2.5*

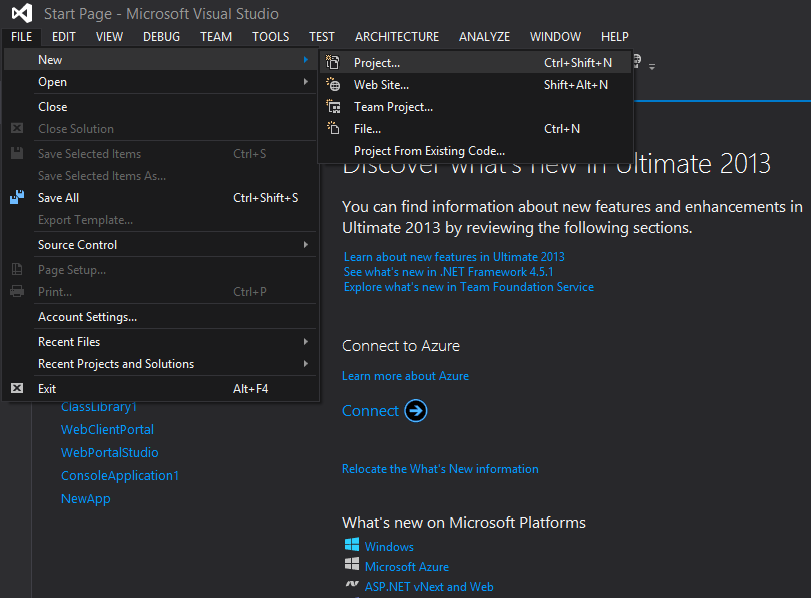


Рисунок 2.1 – Создание нового проекта

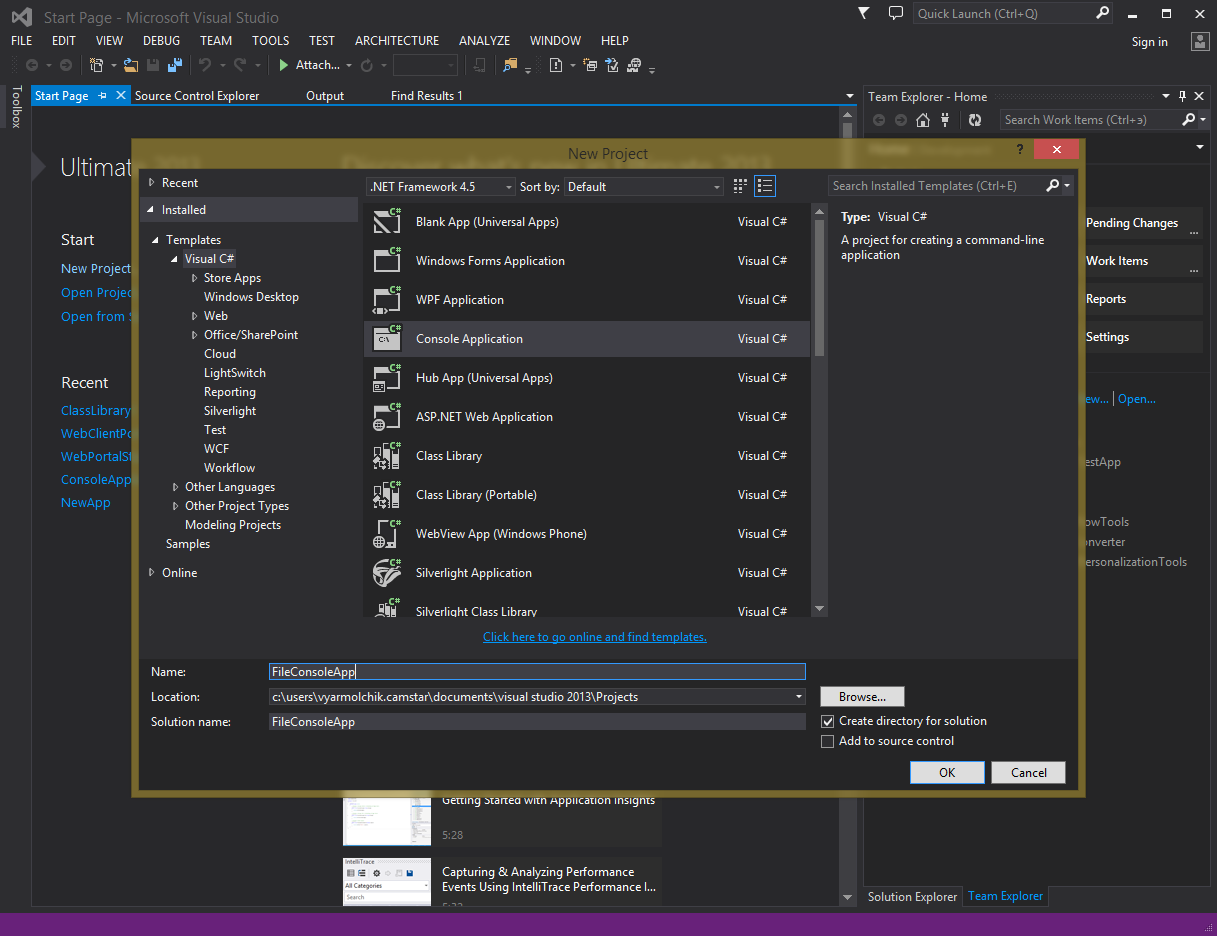


Рисунок 2.2 – Выбираем тип проекта (консольное приложение) и вводим название

Используем простанство имен System.IO, создаем объект класса DirectoryInfo. Для работы с консолью используем статический класс Console, который предоставляет стандартные потоки для консольных приложений: входной, выходной и поток сообщений об ошибках. Доступные методы, свойства и их описание у статических классов и объектов классов можно увидеть по нажатию сочетания клавиш CTRL+Пробел. Все это представлено на рисунках 2.3-2.4. запустить выполнение программы можно по нажатию клавиши F5 или же кнопки Start. Результат работы программы и её код можно увидеть на рисунке 2.5.

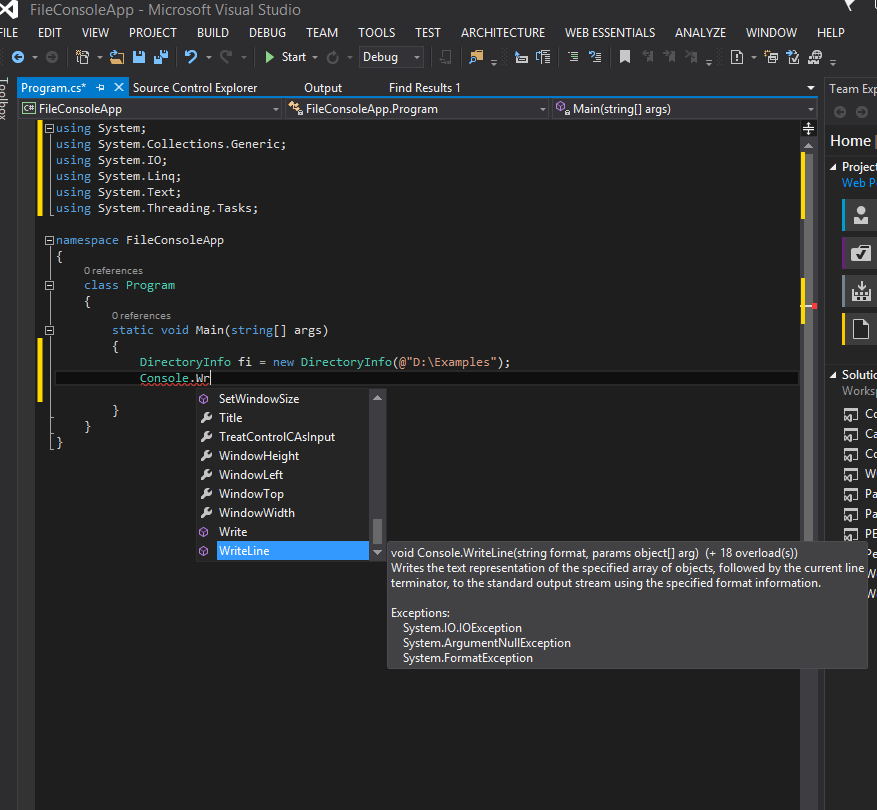


Рисунок 2.3 – Написание содержимого программы

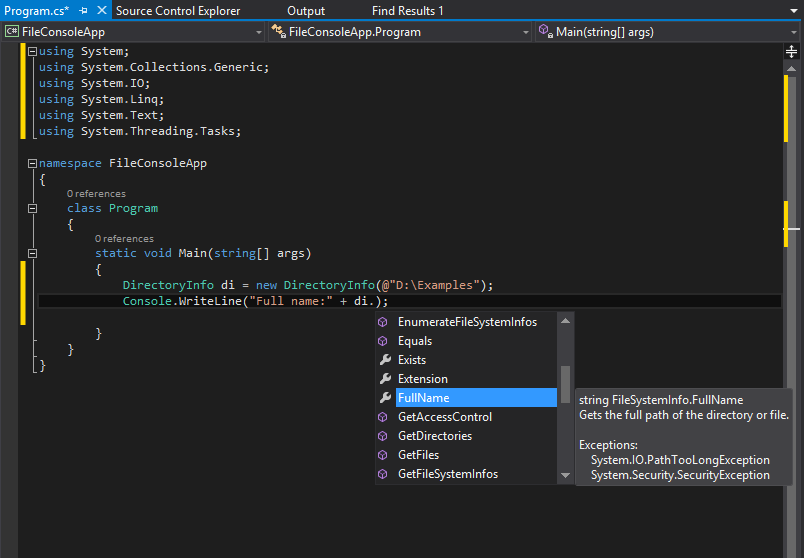


Рисунок 2.4 – Список доступных методов и свойств для объекта класса DirectoryInfo

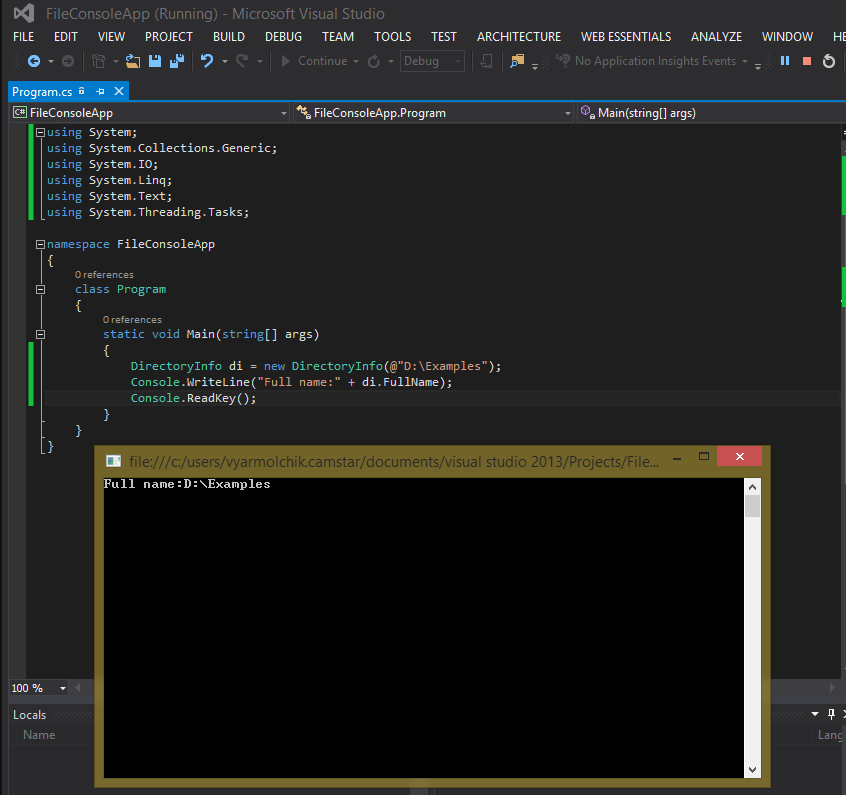


Рисунок 2.5 – Результат работы программы

### Работа с файлами в директории

Предположим, вы хотите получить список всех файлов с расширением BMP в папке F:\Pictures. Для этого можно использовать следующий код:

DirectoryInfo dir = new DirectoryInfo(@"F:\WINNT");

FileInfo[] bmpfiles = dir.GetFiles("\*.bmp);

Console.WriteLine("Total number of bmp files", bmpfiles.Length);

Foreach( FileInfo f in bmpfiles)

{

Console.WriteLine("Name is : {0}", f.Name);

Console.WriteLine("Length of the file is : {0}", f.Length);

Console.WriteLine("Creation time is : {0}", f.CreationTime);

Console.WriteLine("Attributes of the file are : {0}",

f.Attributes.ToString());

}

### Создание подкаталогов

Следующий фрагмент кода описывает как можно создать поддиректорию MySub в директории Sub:

DirectoryInfo dir = new DirectoryInfo(@"F:\WINNT");

try

{

dir.CreateSubdirectory("Sub");

dir.CreateSubdirectory(@"Sub\MySub");

}

catch(IOException e)

{

Console.WriteLine(e.Message);

}

### Создание файлов при помощи класса FileInfo

Класс FileInfo позволяет создавать новые файлы, получать информацию, удалять и перемещать их. В этом классе также есть методы для открытия, чтения и записи в файл. В следующем примере показано, как можно создать текстовый файл и получить доступ к его информации (времени его создания, полное имя, и так далее):

FileInfo fi = new FileInfo(@"F:\Myprogram.txt");

FileStream fstr = fi.Create();

Console.WriteLine("Creation Time: {0}",f.CreationTime);

Console.WriteLine("Full Name: {0}",f.FullName);

Console.WriteLine("FileAttributes: {0}",f.Attributes.ToString());

//Удаление файла Myprogram.txt.

Console.WriteLine("Press any key to delete the file");

Console.Read();

fstr.Close();

fi.Delete();

### LINQ to XML

LINQ to XML — это оснащенный средствами LINQ и встроенный в память программный интерфейс XML, позволяющий работать с XML-файлами внутри языков программирования .NET Framework.

LINQ to XML подобен модели DOM в том отношении, что загружает XML-документ в память. К такому документу можно направить запрос, его можно изменить, а после изменения его можно сохранить в файле или сериализовать и передать через Интернет. Однако между интерфейсом LINQ to XML и моделью DOM существуют отличия: интерфейс реализует более легкую и простую в работе модель объектов.

В дополнение к функциям, реализованным в LINQ, LINQ to XML предоставляет усовершенствованный интерфейс программирования XML.Благодаря LINQ to XML появляются следующие возможности:

* Загружать XML из файлов или потоков.
* Сериализовывать XML в файлы или в потоки.
* Создание XML-кодов «с чистого листа» с помощью функционального построения.
* Обращение с запросами к XML с помощью XPath-подобных осей.
* Манипулирование загруженным в память XML-деревом с помощью таких методов, как [Add](http://msdn.microsoft.com/ru-ru/library/system.xml.linq.xcontainer.add.aspx), [Remove](http://msdn.microsoft.com/ru-ru/library/system.xml.linq.xnode.remove.aspx), [ReplaceWith](http://msdn.microsoft.com/ru-ru/library/system.xml.linq.xnode.replacewith.aspx) и [SetValue](http://msdn.microsoft.com/ru-ru/library/system.xml.linq.xelement.setvalue.aspx).
* Проверять XML-деревья на соответствие схеме XSD.
* Использование сочетания этих функций для преобразования XML-деревьев из одной формы в другую.

Одним из наиболее важных преимуществ программирования с использованием LINQ to XML является простота создания XML-деревьев. Так, для создания небольшого XML-дерева можно написать следующий код C#.

XElement contacts =

new XElement("Contacts",

new XElement("Contact",

new XElement("Name", "Patrick Hines"),

new XElement("Phone", "206-555-0144",

new XAttribute("Type", "Home")),

new XElement("phone", "425-555-0145",

new XAttribute("Type", "Work")),

new XElement("Address",

new XElement("Street1", "123 Main St"),

new XElement("City", "Mercer Island"),

new XElement("State", "WA"),

new XElement("Postal", "68042")

)

)

);

В этом примере используется следующий XML-документ:

<?xml version="1.0"?>

<PurchaseOrder PurchaseOrderNumber="99503" OrderDate="1999-10-20">

<Address Type="Shipping">

<Name>Ellen Adams</Name>

<Street>123 Maple Street</Street>

<City>Mill Valley</City>

<State>CA</State>

<Zip>10999</Zip>

<Country>USA</Country>

</Address>

<Address Type="Billing">

<Name>Tai Yee</Name>

<Street>8 Oak Avenue</Street>

<City>Old Town</City>

<State>PA</State>

<Zip>95819</Zip>

<Country>USA</Country>

</Address>

<DeliveryNotes>Please leave packages in shed by driveway.</DeliveryNotes>

<Items>

<Item PartNumber="872-AA">

<ProductName>Lawnmower</ProductName>

<Quantity>1</Quantity>

<USPrice>148.95</USPrice>

<Comment>Confirm this is electric</Comment>

</Item>

<Item PartNumber="926-AA">

<ProductName>Baby Monitor</ProductName>

<Quantity>2</Quantity>

<USPrice>39.98</USPrice>

<ShipDate>1999-05-21</ShipDate>

</Item>

</Items>

</PurchaseOrder>

В примере показано, как выполнить поиск элемента Address, имеющего атрибут Type со значением «Billing».

XElement root = XElement.Load("PurchaseOrder.xml");

IEnumerable<XElement> address =

from el in root.Elements("Address")

where (string)el.Attribute("Type") == "Billing"

select el;

foreach (XElement el in address)

Console.WriteLine(el);

Этот код выводит следующие результаты.

<Address Type="Billing">

<Name>Tai Yee</Name>

<Street>8 Oak Avenue</Street>

<City>Old Town</City>

<State>PA</State>

<Zip>95819</Zip>

<Country>USA</Country>

</Address>

Практическая часть

Программа предстваляет из себя консольное приложение, работающее с файлами и xml-данными. Элементы, которые **обязательно** должны присутстваовать в программной реализации:

- базовый класс, минимум 3 метода, 1-2 из которых виртульные;

- partial класс, унаследованный от базового, в первом файле класса объявить защищеные поля и доступ к ним предоставить через свойства, во втором файле прописать методы, константы, read-only поля, статические поля и методы и другие элементы класса;

- методы расширения;

- считывание информации из xml-файла с использованием технологии LINQ to XML;

- коллекции объектов;

- запись данных из коллекции в файл с применением операторов LINQ to Objects;

- делегат.

1. Программа для размещения депозита в банке. Изучить предметную область, её основные сущности. Система, дает возможность клиенту выбрать депозит, данные о которых (не менее 5) хранятся в xml-файле. После выбора клиенту становятся доступными дополнительные функции: просмотр остатка на определенное число, расчет процентых начислений за выбранный период, снятие процентов, пополнение счета (можно дополнить). Предусмотреть сохрание истории операций и формирование выписки (отчета) о проделанных операциях за выбранный промежуток времени.
2. Программа продажи билетов на автобус с регистрацией пассажиров. Данные об автобусах и количестве свободных мест хранятся в xml-файле (минимум 3 автобуса на несколько дней). Клиенту предоставляется возможность выбрать маршрут, а потом доступную дату, на которую есть билеты. Клиент вводит свои данные (ФИО, номер телефона, серия и номер паспорта – предусмотреть валидацию). После подтверждения, xml-файл с данными маршрута обновляется. Данные о пассажире сохраняются. Предусмотреть возможность отменить регистрацию и просмотреть текстовый документ со всеми зарегистрировавшимися.
3. Система учета проката велосипедов. Данные о велосипедах хранятся в xml-файле. Клиент может просматривать доступные велосипеды в разное время (например, да пару дней). Предусмотреть несколько вариантов проката (1 час, 3 часа, сутки – например, как минимум). При заказе оформляется договор с данными клиента, клиент может получить свою копию договора. Вести учет сданных в прокат велосипедов. Предусмотреть комиссию за просроченную сдачу велосипеда обратно, рассчитать сумму. Для владельца прокатом разработать форму для расчета периода окупаемости велосипеда при различных факторах.
4. Программа для сдачи в аренду квартир. Данные о квартирах хранятся в xml-формате. Собственник может добавить объявление о квартире, предусмотреть разделение по регионам. Клиенты выбирают регион и квартиру из списка для просмотра, предусмотреть подсчет просмотров для каждой квартиры. Клиент может сохранить понравившуюся квартиру в текстовый файл, название файла (доп. – папка хранения) вводится клиентом через консоль вручную. Предусмотреть возможные ошибки. Добавить поиск и несколько фильтров для поиска квартир.
5. Система ведения бонусной программы магазина. Регистрировать новые бонусные карточки клиентов, данные хранить в xml-формате. При совершении покупки вводить номер бонусной клиента (предусмотреть валидацию) и зачислять процент от всей стоимости покупки. Снимать баллы в счет оплаты. Для клиента предусмотреть просмотр баланса бонусов. Делать отчет по карточке (зачисление, снятие) в текстовом формате.

Контрольные вопросы:

1. Что представляет из себя .NET?
2. Что такое CLR и какие основные задачи она выполняет?
3. Что такое CTS и каково её предназначение?
4. Назовите преимущества .NET.
5. Какие категории типов существуют в C# и в чем их отличие?
6. Что такое пространства имен и какие свойства они имеют?
7. Какие элементы может содержать класс в C#?
8. Что такое делегат и как он объявляется?
9. Чем структура отличается от класса?
10. Что такое универсальный шаблон?
11. Какой набор типов включает в себя .NET для предоставления стандратных коллекций?
12. Что такое LINQ? Какие существуют типы LINQ?