ВВЕДЕНИЕ В .NET-ПРОГРАММИРОВАНИЕ

Платформа .NET, ее назначение и структура. Обзор технологий .NET

В 2000 году компания Microsoft объявила о создании нового языка программирования - языка C#. Эта акция стала частью более значительного события - объявления о платформе .NET (.NET Framework). Платформа .NET по сути представляла собой новую модель создания приложений, которая включает в себя следующие возможности:

1) использование библиотеки базовых классов, предлагающих целостную объектно-ориентированную модель программирования для всех языков программирования, поддерживающих .NET;
2) полное и абсолютное межъязыковое взаимодействие, позволяющее разрабатывать фрагменты одного и того же проекта на различных языках программирования;
3) общая среда выполнения приложений .NET, независимо от того, на каких языках программирования для данной платформы они были созданы; при этом среда берет на себя контроль за безопасностью выполнения приложений и управление ресурсами;
4) упрощенный процесс развертывания приложения, в результате чего установка приложения может свестись к простому копированию файлов приложения в определенный каталог.

Одним из основных элементов .NET Framework является библиотека классов под общим именем FCL (Framework Class Library), к которой можно обращаться из различных языков программирования, в частности, из C#. Эта библиотека разбита на модули таким образом, что имеется возможность использовать ту или иную ее часть в зависимости от требуемых результатов. Так, например, в одном из модулей содержатся "кирпичики", из которых можно построить Windows-приложения, в другом — "кирпичики", необходимые для организации работы в сети и т.д.

Кроме FCL в состав платформы .NET входит Common Language Runtime (CLR — единая среда выполнения программ), название которой говорит само за себя - это среда ответственна за поддержку выполнения всех типов приложений, разработанных на различных языках программирования с использованием библиотек .NET.

Замечания

Следует отметить, что основными языками, предназначенными для платформы .NET Framework, являются C#, VB.NET, Managed C++ и JScript .NET. Для данных языков Microsoft предлагает собственные компиляторы, переводящие программу в специальный код, называемый IL-кодом, который выполняется средой CLR.

Кроме Microsoft, еще несколько компаний и академических организаций создали свои собственные компиляторы, генерирующие код, работающий в CLR. На сегодняшний момент известны компиляторы для Pascal, Cobol, Lisp, Perl, Prolog и т.д. Это означает, что можно написать программу, например, на языке Pascal, а затем, воспользовавшись соответствующим компилятором, создать специальный код, который будет работать в среде CLR.

Процесс компиляции и выполнения программы в среде CLR более подробно будет рассмотрен позже.

Среда CLR берет на себя всю низкоуровневую работу, например, автоматическое управление памятью. В языках программирования предыдущих поколений управление ресурсами, в частности, управление памятью, являлось одной из важных проблем. Объекты, созданные в памяти, должны быть удалены из нее, иначе память будет исчерпана и программа не сможет продолжить выполнение. Т.к. это

достаточно сложный процесс, то часто программисты просто забывали удалять неиспользуемые объекты.

При разработке платформы .NET эту проблему постарались решить. Теперь управление памятью берет на себя среда CLR. В процессе работы программы среда следит за объектами и автоматически уничтожает неиспользуемые.

Замечание. *Система управления памятью называется Garbage Collector (GC) или сборщиком мусора.*

Среда CLR обеспечивает интеграцию языков и позволяет объектам, созданным на одном языке, использовать объекты, написанные на другом. Такая интеграция возможна благодаря стандартному набору типов и информации, описывающей тип (метаданным). Интеграция языков очень сложная задача, так как некоторые языки не учитывают регистры символов, другие не поддерживают методы с переменным числом параметров и т.д. Чтобы создать тип, доступный для других языков, придется задействовать лишь те возможности языка, которые гарантированно доступны в других языках. С этой целью Microsoft разработал:

1) общую систему типов (Common Type System, CTS), которая описывает все базовые типы данных, поддерживаемые средой CLR, и определяет, как эти типы будут представлены в формате метаданных .NET.
2) общеязыковую спецификацию (Common Language Specification, CLS), описывающую минимальный набор возможностей, который должен быть реализован производителями компиляторов, чтобы их продукты работали в CLR, а также определяющую правила, которым должны соответствовать видимые извне типы, чтобы к ним можно было получить доступ из любых других CLS-совместимых языков программирования.

Важно понимать, что система CLR/CTS поддерживает гораздо больше возможностей для программиста, чем спецификации CLS (см. Рис. 1. Схема пересечения возможностей языков).

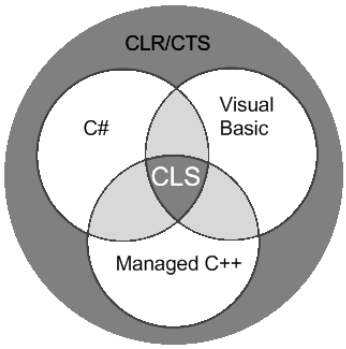


Рис. 1. Схема пересечения возможностей языков

Если при разработке какого-то типа требуется, чтобы он был доступен другим языкам, нельзя использовать возможности своего языка, выходящие за рамки возможностей, определяемых CLS. Иначе созданный тип может оказаться недоступным программистам, пишущим код на других языках. Если межъязыковое взаимодействие не требуется, то можно разрабатывать очень мощные типы, ограничиваясь лишь возможностями языка.

Принцип компиляции и выполнения программы в среде CLR. Управляемый и неуправляемый код

Создание приложений с помощью .NET Framework означает написание программы на любом языке программирования, который поддерживается этой платформой. Для того чтобы написанная, например на C#, программа была выполнена, ее необходимо преобразовать в программу на языке, «понятном» компьютеру

(исполняемый код). Такой процесс преобразования называется компиляцией, а программа, которая его выполняет - компилятором. В прошлом почти все компиляторы генерировали код для конкретных процессорных архитектур. При разработке платформы .NET от этой зависимости постарались избавиться. Для этого ввели двухшаговую компиляцию.

На первом этапе все .NET компиляторы генерируют промежуточный код, код на языке Intermediate Language (IL — промежуточный язык) или IL-код. Т.е. компиляция со всех языков программирования .NET, включая C#, происходит в этот промежуточный язык. IL-код не является специфическим ни для какой операционной системы и ни для какого языка программирования. Он может быть выполнен в любой среде, для которой реализована CLR-система.

На втором этапе IL-код переводится в код, специфичный для конкретной операционной системы и архитектуры процессора. Эта работа возлагается на JIT-компилятор (Just In Time compiler - компилирование точно к нужному моменту). Только после этого операционная система может выполнить приложение.

***Замечание**. JIT-компилятор входит в состав среды CLR.*

IL-код, выполняемый под управлением CLR, называется управляемым (managed). Это означает, что среда CLR полностью управляет жизненным циклом программы: отслеживает безопасность выполнения команд программы, управляет памятью и т.д. Это несомненно является достоинством управляемого кода. Конечно, использовать приложения, разработанные на основе управляемого кода, можно только тогда, когда на компьютере установлена .NET Framework.

Использование IL-кода имеет и оборотную сторону – поддержка любой платформы означает отказ от функциональности, специфичной для конкретной платформы. Обойти это ограничение позволяет использование неуправляемого кода (unmanaged), т.е. кода, который не контролируется CLR или выполняется самой операционной системой. Такой код приходится использовать при необходимости обращения к низкоуровневым функциям операционной системы (например, понятие реестра существует только в Windows и функции, работающие с реестром, приходится вызывать из операционной системы). Иногда использование неуправляемого кода позволяет ускорить выполнение некоторых алгоритмов.

Назначение и возможности Visual Studio

В рамках данного курса мы будем изучать язык C# — один из языков программирования, который может использоваться для создания приложений, выполняемых в среде CLR. Этот язык был создан компанией Microsoft специально для использования на платформе .NET.

В общем случае создавать файлы с исходным кодом на языке C# возможно с помощью обычного текстового редактора, например, Блокнота. Затем необходимо будет скомпилировать их в управляемый код через командную строку. Однако наиболее удобно для этих целей использовать Microsoft Visual Studio, потому что:

1. Visual Studio автоматически выполняет все шаги, необходимые для создания IL-кода.
2. Текстовый редактор Visual Studio изначально настроен для работы с теми .NET языками, которые были разработаны Microsoft, в том числе C#, поэтому он может интеллектуально обнаруживать ошибки и «подсказывать» в процессе ввода, какой именно код можно использовать на данном этапе разработки (технология IntelliSense).
3. В состав Visual Studio входят средства, позволяющие создавать не только консольные, но и Windows-, и Web-приложения путем простого перетаскивания мышью элементов пользовательского интерфейса.

4. Многие типы проектов, создание которых возможно на C#, могут разрабатываться на основе готовых шаблонов проектов. Вместо того чтобы каждый раз начинать с нуля, Visual Studio позволяет использовать уже имеющиеся файлы с исходным кодом, что уменьшает временные затраты на создание проекта.

Перечислим некоторые типы приложений, которые позволяет создавать Visual Studio:

1. Console Application – позволяют выполнять вывод на «консоль», то есть в окно командного процессора. Данный тип приложений существует со времен операционных систем с текстовым пользовательским интерфейсом, например MS-DOS. Тем не менее, консольные приложения продолжают активно использоваться и в наши дни. Их применение может быть связано с отсутствием необходимости в графическом интерфейсе. Например, утилиты автоматической компиляции приложений, как правило, выполняются в заранее назначенные интервалы времени без участия пользователя рис. 2.

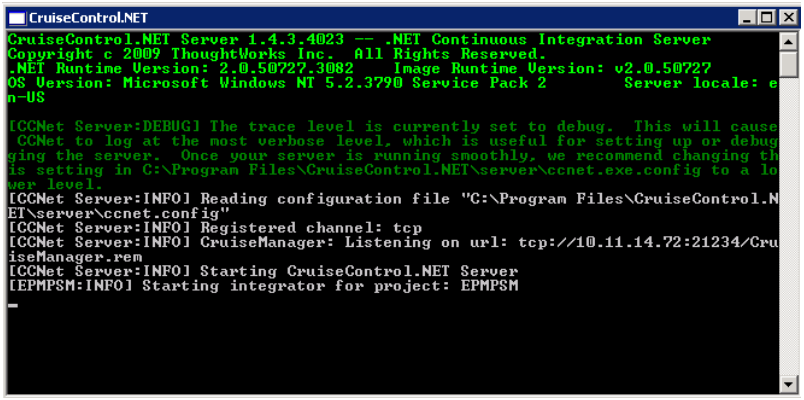


Рис. 2. Процедура запуска системы контроля кода проекта.

Еще одним вариантом применения консольного ввода/вывода является встраивание его в программы с графическим интерфейсом. Дело в том, что современные программы содержат очень большое число команд, значительная часть которых никогда не используется обычными пользователями. В то же время, эти команды должны быть доступны в случае необходимости. Ярким примером использования данного подхода являются компьютерные игры рис 3.

Рис. 3. Пример использования консоли в игре.

2. Windows Forms – используют элементы графического оконного интерфейса, включая формы, кнопки, флажки и т.д. Приложения такого типа более удобны для пользователя, так как позволяют ему отдавать команды щелчком мыши, а не ручным вводом команд, что позволяет значительно повысить скорость работы по сравнению с консольными приложениями. Типичным примером приложения, построенного с применением графического интерфейса, является MS Word рис.4.

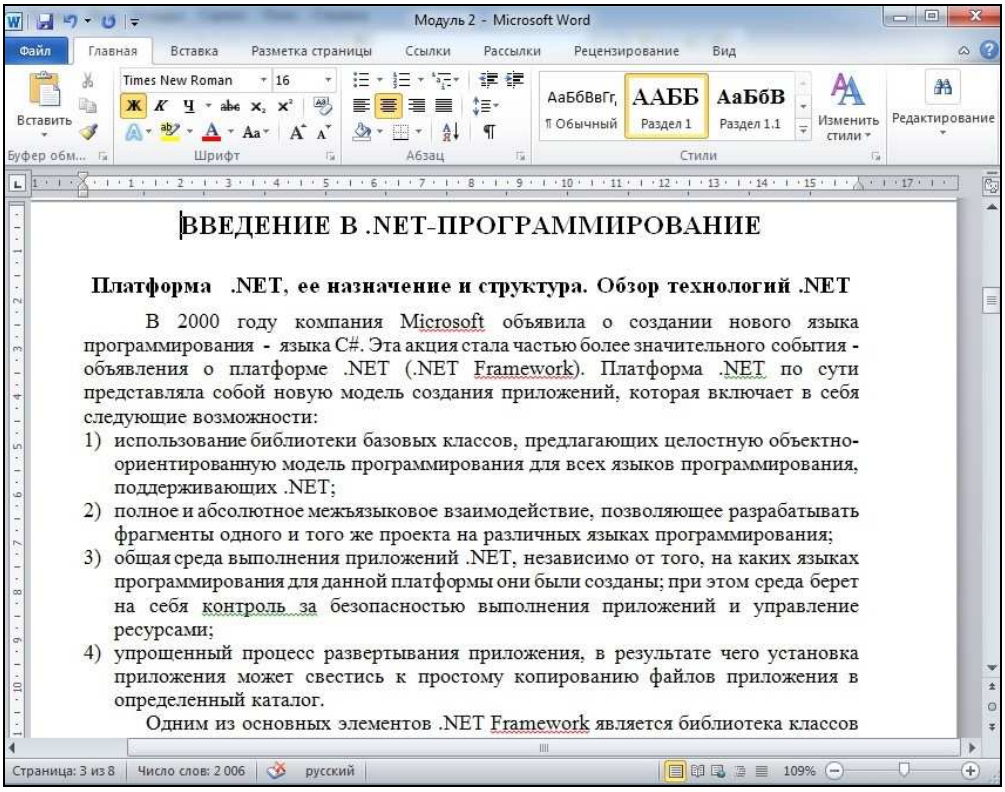


Рис. 4. Графический интерфейс пользователя на примере MS Word 2007.

3. Библиотека классов (Class Library) – представляют собой библиотеки, содержащие классы и методы. Библиотеки не являются полноценным самостоятельными приложениями, но могут использоваться в других программах. Как правило, в библиотеки помещают алгоритмы и структуры данных, которые могут быть полезны более чем одному приложению.

Специфика платформы .NET такова, что она «подходит» для разработки «офисных» приложений, Web-приложений, сетевых приложений и приложений для мобильных устройств. В то же время она не предназначена для создания операционных систем и драйверов. Подумайте почему.

В рамках данного курса мы рассмотрим основы программирования на языке C#, разрабатывая консольные приложения в среде Visual Studio.

Приложение, находящееся в процессе разработки, называется проектом (project). Несколько проектов могут быть объединены в решение (solution). Совсем не обязательно, чтобы проекты в решении были одного типа. Например, в одно решение могут быть объединены Web-приложение и библиотеки, которые оно использует.
Сначала мы будем создавать решение, состоящее из одного проекта.

Создание первого проекта в среде Visual Studio

Для создания проекта следует запустить Visual Studio. Пред вами откроется окно:

Замечание. *В зависимости от версии Visual Studio и от того, как в данный момент настроена среда, вид экрана может немного отличаться. Здесь и далее мы приводим примеры с использованием Visual C# 2010 Express.*

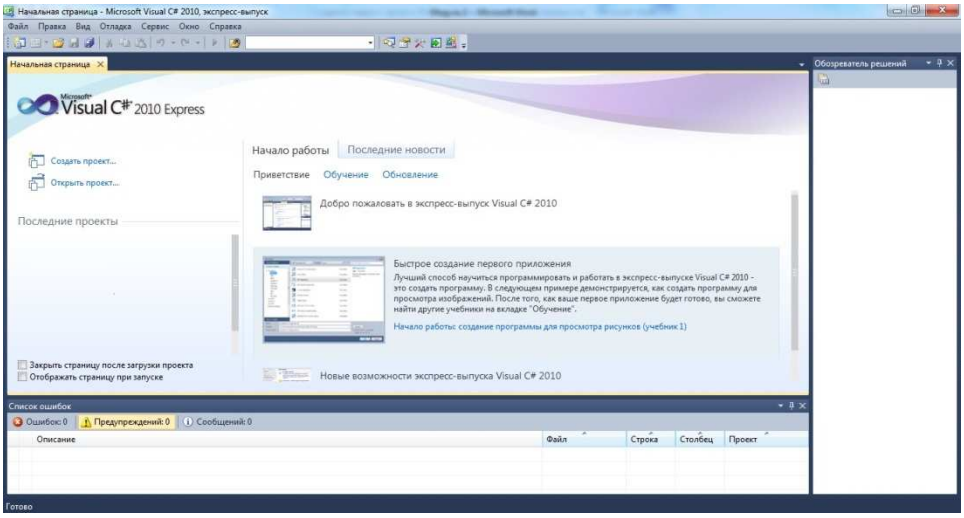


Рис. 5. Начальная страница.

Затем на начальной странице приложения необходимо нажать ссылку ***«Создать проект…»*** (см. рис.5) или в главном меню выбрать команду ***Файл – Создать проект…***. После этого откроется диалоговое меню ***«Создать проект»*** рис.6.

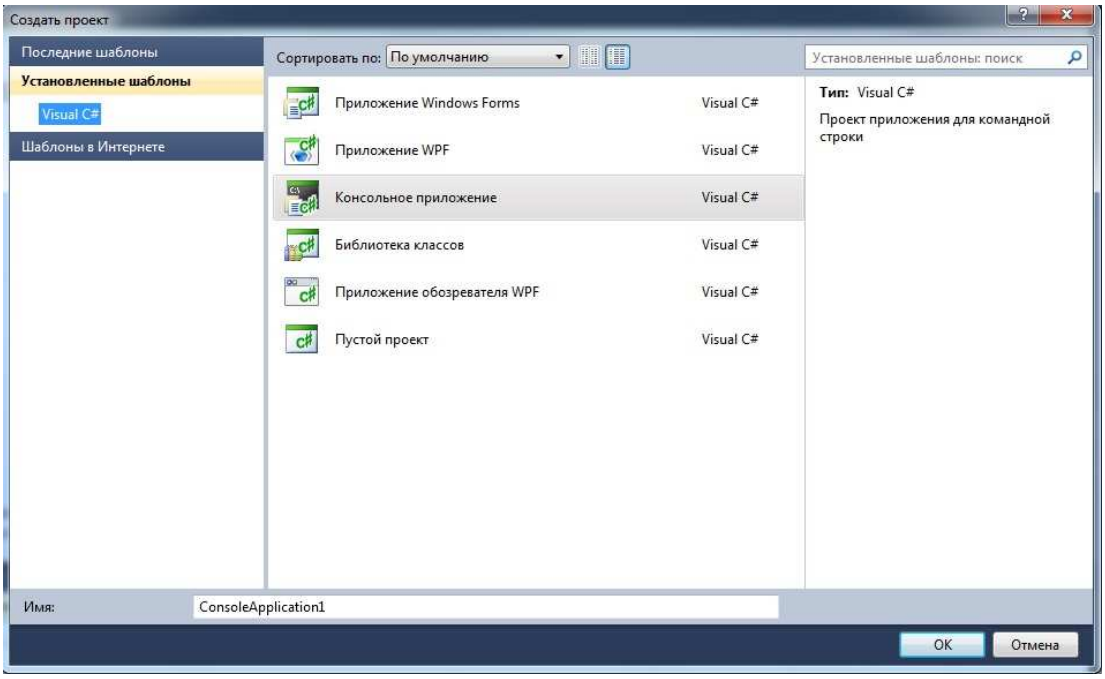


Рис. 6. Диалоговое окно «Создать проект».

Далее следует выбрать тип создаваемого приложения – ***Консольное приложение***. В строчке ***Имя*** следует ввести новое имя приложения ***Hello*** и нажать кнопку ***OK***. Экран примет вид, изображенный на рис. 7.

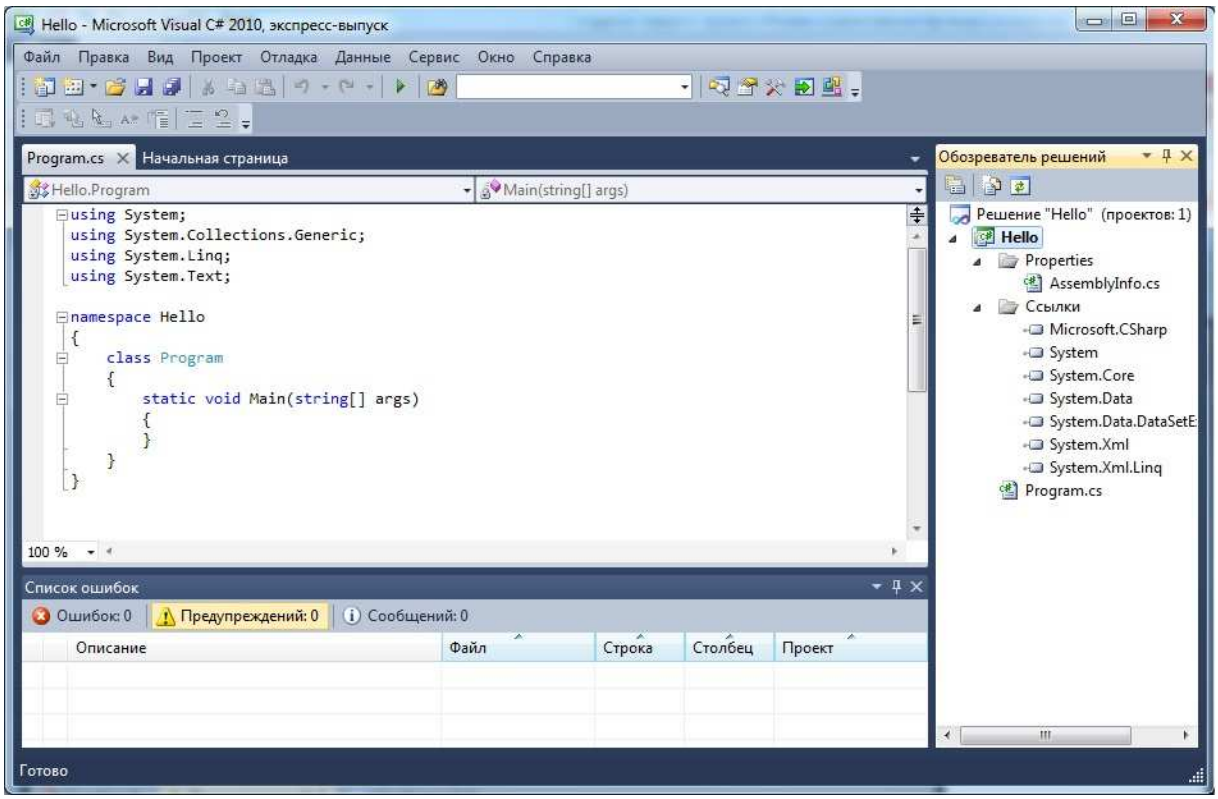


Рис. 7. Рабочая область проекта ***Hello***.

В правой части экрана располагается окно управления проектом ***Обозреватель решений***.

Замечание. Если вы случайно закроете данное окно, то его можно включить командой *Вид – Другие окна – Обозреватель решений*. Посмотрите, какие еще окна могут быть подключены к вашему проекту.

В этом окне перечислены все ресурсы, входящие в проект:

1) ***Properties*** (Свойства), содержит файл AssemblyInfo.cs – *информация о сборке*.
   Компилятор в качестве результата своего выполнения создает *сборку* – файл с расширением exe, или dll, который содержит IL-код.
2) ***Ссылки*** на стандартные библиотеки System, System.Data, System.Xml и т.д.
3) ***Program.cs*** - исходный текст программы на языке C#.

В левой нижней части экрана располагается окно ***Список ошибок***, которое во время отладки проекта позволит нам получать информацию о локализации и типе ошибок.

Основное пространство экрана занимает окно редактора, в котором располагается текст программы, созданный средой автоматически. Текст представляет собой каркас, в который программист будет добавлять нужный ему код и изменять его. При этом зарезервированные (ключевые) слова отображаются синим цветом, комментарии – зеленым, основной текст – черным. Текст структурирован. Щелкнув на знаке минус в первой колонке текста, мы скроем блок кода, щелкнув на знаке плюс – откроем.

Для того чтобы сохранить проект нужно в главном меню выполнить команду ***Файл – Сохранить все***. Откроется диалоговое окно ***«Сохранить проект»*** рис. 8.

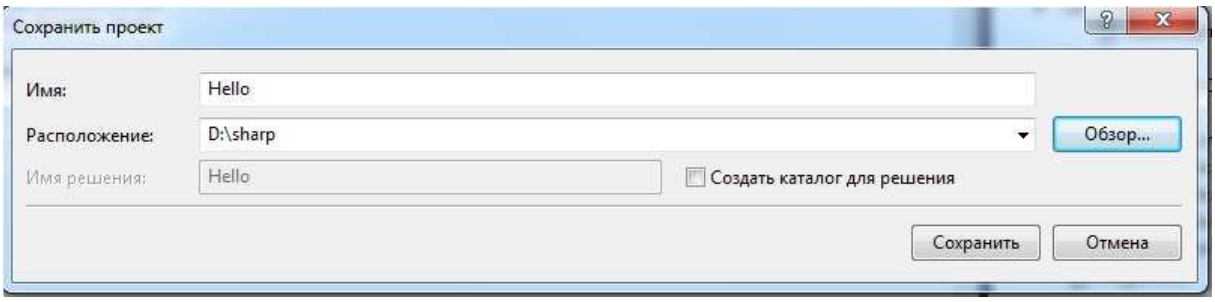


Рис. 8. Диалоговое окно «Сохранить проект».

Имя проекта вы определили на этапе его создания. При необходимости это имя можно изменить.

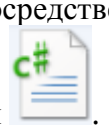
Расположение проекта зависит от установок, которые можно изменить через кнопку ***Обзор***. Мы будем сохранять проекты на диск ***D*** в папку ***sharp***.

Для того чтобы запустить приложение нужно нажать на кнопку *Start* ( ) стандартной панели инструментов VS. В результате программа скомпилируется в IL-код и этот код будет передан CLR на выполнение. Программа запустится, и, так как в данный момент наш код не содержит никаких команд, тут же завершит свою работу. Визуально это будет выглядеть как быстро мелькнувший на экране черный прямоугольник.

Откройте папку, содержащую проект, и изучите ее структуру. На данном этапе особый интерес для нас будут представлять следующие файлы:

1. *Hello.sln* – основной файл, отвечающий за всё решение. Если необходимо открыть решение для работы, то нужно выбрать именно этот файл. Остальные файлы откроются автоматически.

   Данный файл помечается пиктограммой  .
2. *Program.cs* – файл, в котором содержится исходный код, написанный на языке C#. Именно с этим файлом мы и будем непосредственно работать.

   Данный файл помечается пиктограммой  .
3. *Hello\bin\Debug\Hello.exe* – файл, в котором содержатся сгенерированный IL-код проекта. Другими словами, этот файл и есть готовое приложение, которое может выполняться на любом компьютере, на котором установлена платформа .NET.

   Данный файл помечается пиктограммой  .

Теперь рассмотрим сам текст программы.

using System – это директива, которая разрешает использовать имена стандартных классов из пространства имен ***System*** непосредственно, без указания имени пространства, в котором они были определены. Так, например, если бы этой директивы не было, то нам пришлось писать бы System.Console.WriteLine (назначение данной команды мы рассмотрим позже). Конечно, писать полное пространство имен каждый раз очень неудобно. При указании директивы using можно писать просто имя, например, Console.WriteLine.

Задание. Используя дополнительные источники информации, узнайте за что отвечают директивы System.Collections.Generic, System.Linq, using System.Text.

Для консольных программ ключевое слово ***namespace*** создает свое собственное пространство имен, которое по умолчанию называется именем проекта. В нашем

случае пространство имен называется Hello. Однако программист вправе указать другое имя.

Каждое имя, которое встречается в программе, должно быть уникальным. В больших и сложных приложениях используются библиотеки разных производителей. В этом случае трудно избежать конфликта между используемыми в них именами. Пространства имен предоставляют простой механизм предотвращения конфликтов имен. Они создают разделы в глобальном пространстве имен.

C# – объектно-ориентированный язык, поэтому написанная на нем программа будет представлять собой совокупность взаимодействующих между собой классов. Автоматически был создан класс с именем ***Program***. Данный класс содержит только один метод – метод ***Main(),*** который является точкой входа в программу. Это означает, что именно с данного метода начнется выполнение приложения. Каждая консольная программа на языке C# должна иметь метод Main ().

Замечания.

Перед объявлением типа возвращаемого значения ***void,*** *который означает, что метод не возвращает значение, стоит ключевое слово* ***static*** *- это означает что метод Main() можно вызывать, не создавая экземпляр класса типа* ***Program***.

После имени метода в круглых скобках записано ***string[] args*** *– это означает, что в метод Main при запуске могут быть переданы некоторые параметры. На данном этапе передачу параметров мы проводить не будем, поэтому данный фрагмент кода (а именно* ***string[] args****) можно стереть.*

Тело метода Main() ограничивают парные фигурные скобки. Добавим в тело метода следующий код:

Console.WriteLine("Hello!");

Здесь *Console* имя стандартного класса из пространства имен System, отвечающего за базовую систему ввода/вывода данных, т.е. клавиатуру и экран. Его метод *WriteLine* выводит на экран текст, заданный в кавычках.

Обратите внимание на то, как работает технология IntelliSense при вводе кода в текст программы рис.9 .

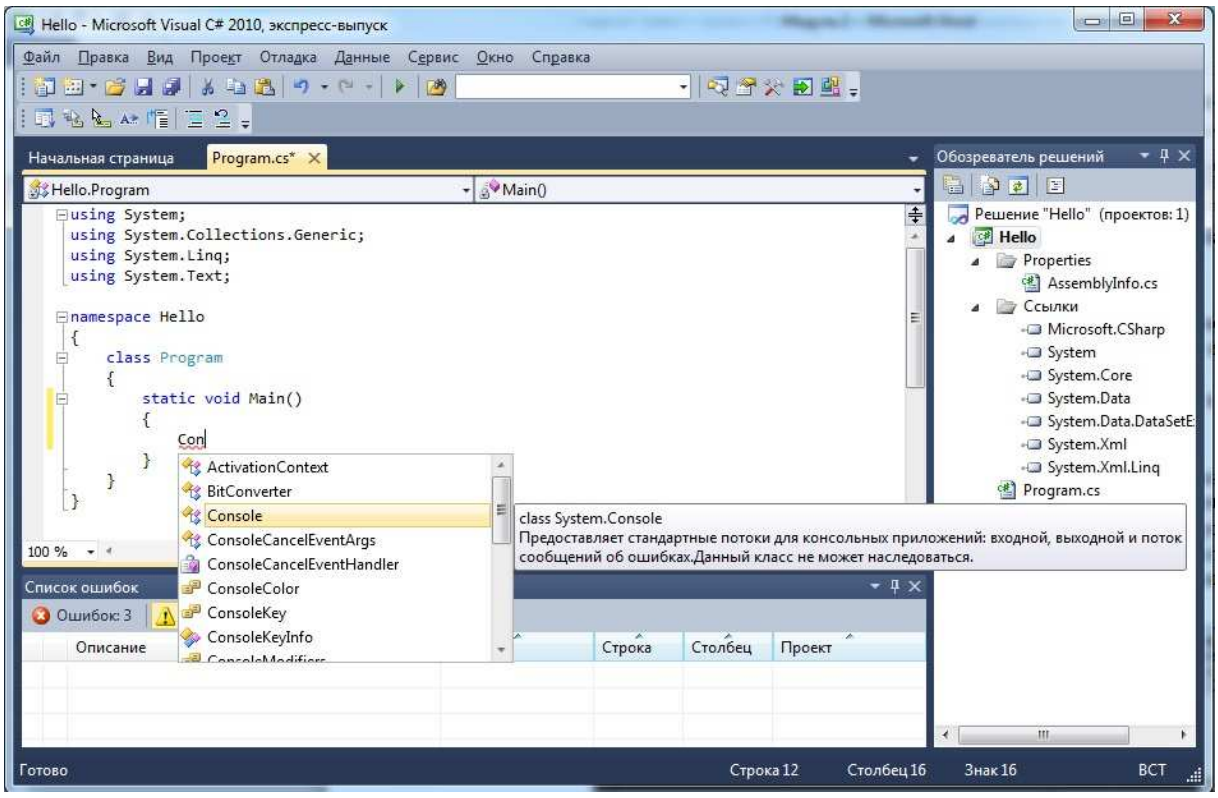


Рис. 9 . Технология IntelliSense

Уже в начале процесса ввода кода открывается контекстное меню, которое подсказывает все «знакомые» ей слова, начинающиеся на буку С. Когда будет введено словосочетания «Con», контекстное меню высветит слово Console. Если теперь вы нажмете на клавишу Enter, то данное слово автоматически вставится в текст программы.

Теперь поставьте точку и наберите букву W (см.рис.10). Контекстное меню будет не только показывать все методы и свойства, начинающиеся на букву W, но и приводить справочную информацию о том, за что отвечает тот или иной метод или свойство класса.

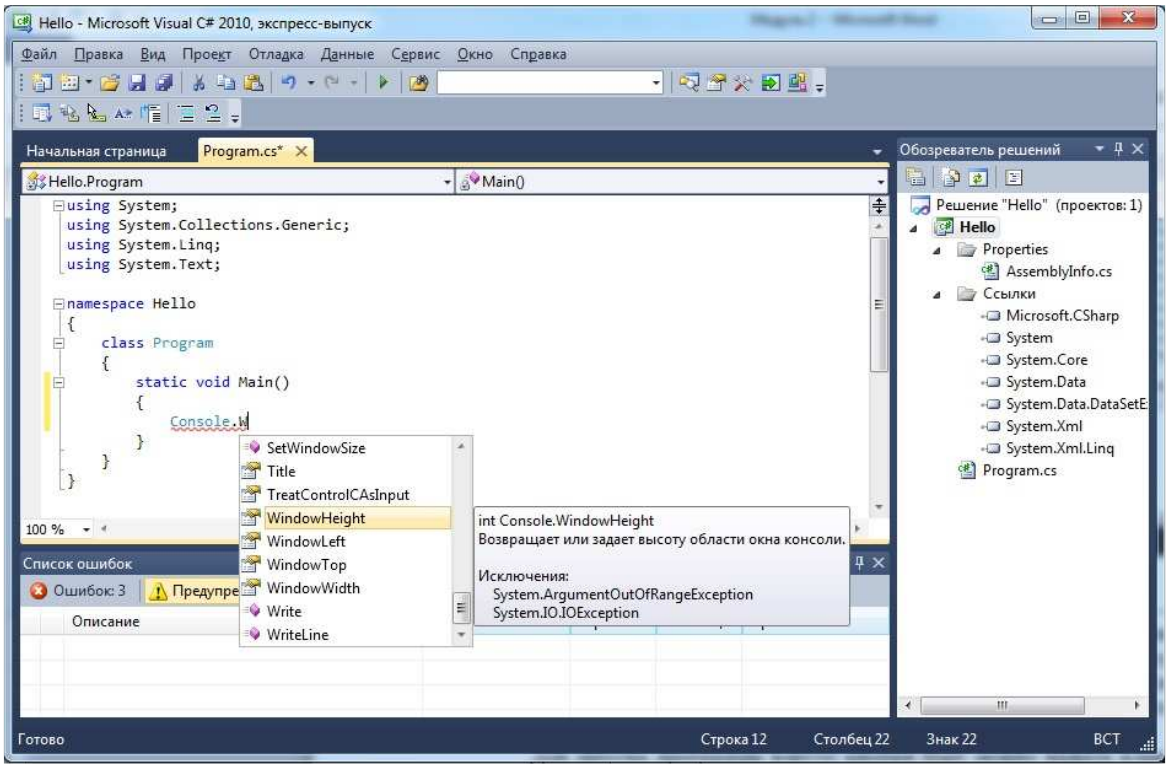


Рис. 10 . Получение справочной информации с использованием технологии IntelliSense

Если вы щелкните левой кнопкой мышки на имени метода WritelLine, то вы получите справочную информацию о данном методе, если дважды щелкните левой кнопкой мышки, то имя метода автоматически добавится в текст программы.

Для запуска программы вместо кнопки Start можно нажать клавишу F5 или выполнить команду ***Отладка – Начать отладку.*** Если код программы не содержит ошибок, то сообщение выведется в консольное окно, которое мелькнет и быстро закроется. Чтобы просмотреть сообщение в нормальном режиме, нужно нажать клавиши Ctrl+F5**.** В нашем случае откроется консольное окно, которое показано на рис.11.

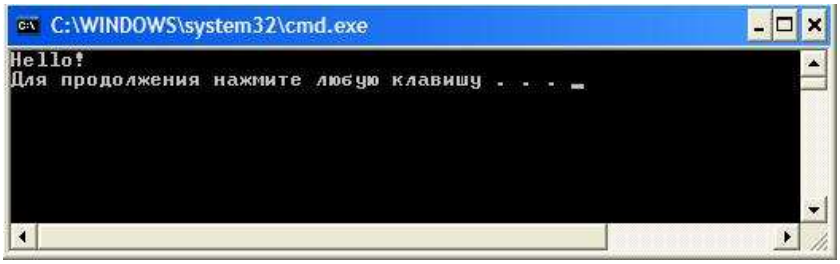


Рис. 11. Окно консольного приложения.

Замечание*. В более сложных проектах имеет смысл явно вставить в конец программы команду Console.Readline(), позволяющую подождать до тех пор, пока пользователь не*

нажмет на клавишу Enter. Данный вариант является более предпочтительным, потому что нажатие Ctrl+F5 приводит к запуску приложения с отключенным режимом отладки, который довольно часто бывает необходим для поиска ошибок.

Если код программы будет содержать ошибки, например, пропущена точка с запятой после команды вывода, то после нажатия клавиши F5 откроется диалоговое окно, в котором выведется сообщение о том, что обнаружена ошибка, и вопрос, продолжать ли работу дальше. Если вы ответите ***Yes***, то будет выполнена предыдущая удачно скомпилированная версия программы (если такая компиляция уже была). Иначе процесс будет остановлен и появится окно ошибок ***Список ошибок*** см.рис.12.

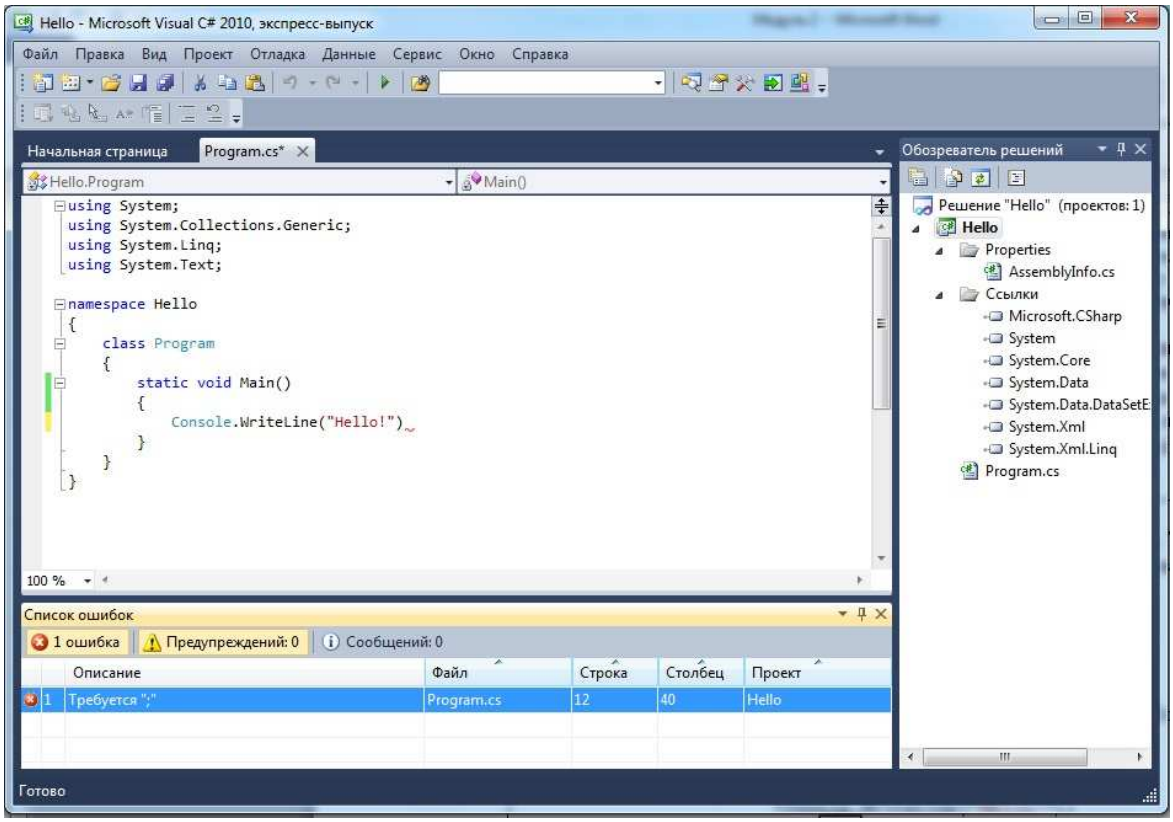


Рис. 12. Отладка ошибок.

Задания

1. Измените текст кода так, чтобы на экран выводилось сообщение:
«Привет! Меня зовут Вася».

2. Изучите, чем метод Write отличется от метода WriteLine.

3. Попробуйте сделать несколько ошибок, например, не закрыть скобку или поставить лишнюю скобку, допустить ошибку в служебных словах, и посмотрите, какие сообщения появятся в окне Списка ошибок.

Задания для самостоятельной работы

1. Познакомьтесь с программой Microsoft DreamSpark https://www.dreamspark.com/ и узнайте какие возможности она дает вам для изучения данного курса.
2. Познакомьтесь с порталом MSDN http://msdn.microsoft.com/ru-RU/. Особое внимание следует уделить ресурсам, посвященным Visual C# http://msdn.microsoft.com/ru-ru/vstudio/hh341490.
3. Произведите установку необходимого программного обеспечения на свой персональный компьютер.