Лекция 1

Особенности платформы .Net

1. Первый взгляд на платформу .NET
2. Объектно-ориентированное программирование
3. Классы
4. Среда Visual Studio.NET
5. Консольные приложения

1. Первый взгляд на платформу .NET

Программист пишет программу, компьютер ее выполняет.

Программа создается на языке, понятном человеку, а компьютер умеет исполнять только программы, написанные на его языке – в машинных кодах. Совокупность средств, с помощью которых программы пишут, корректируют, преобразуют в машинные коды, отлаживают и запускают, называют *средой разработки*, или *оболочкой*.

Среда разработки обычно *содержит*:

- *текстовый редактор*, предназначенный для ввода и корректировки текста программы;
- *компилятор*, с помощью которого программа переводится с языка, на котором она написана, в машинные коды;
- *средства отладки и запуска* программ;
- общие *библиотеки*, содержащие многократно используемые элементы программ;
- *справочную систему* и другие элементы.

Под платформой понимается нечто большее, чем среда разработки для одного языка.

Платформа **.NET** (произносится «**дотнет**») включает не только среду разработки для нескольких языков программирования, называемую Visual Studio.NET, но и множество других средств, например, механизмы поддержки баз данных, электронной почты и коммерции.

В эпоху стремительного развития Интернета – глобальной информационной сети, объединяющей компьютеры разных архитектур, ***важнейшими задачами*** при создании программ становятся:

- *переносимость* – возможность выполнения на различных типах компьютеров;
- *безопасность* – невозможность несанкционированных действий;
- *надежность* – способность выполнять необходимые функции в предопределенных условиях; средний интервал между отказами;
- использование *готовых компонентов* – для ускорения разработки;
- межъязыковое *взаимодействие* – возможность применять одновременно несколько языков программирования.

Платформа .NET позволяет успешно решать все эти задачи.

Для *обеспечения переносимости:*

компиляторы, входящие в состав платформы, переводят программу ***не в машинные коды, а в промежуточный язык,*** **MicroSoft Intermediate Language**, **MSIL**, или просто **IL**, который не содержит команд, зависящих от языка,

операционной системы и

типа компьютера.

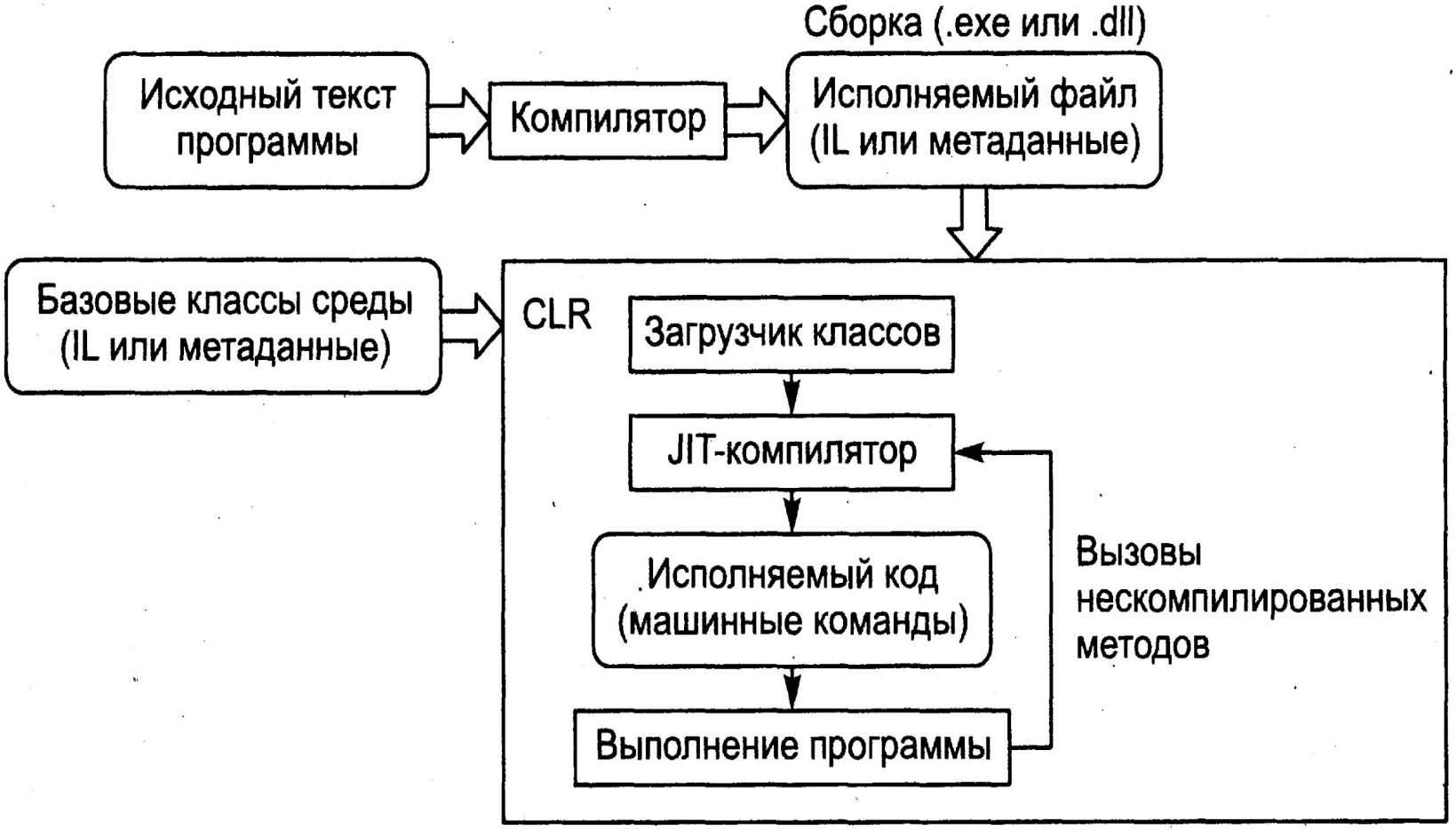
Программа на этом языке выполняется не самостоятельно, а *под управлением системы*, которая называется *общеязыковой средой выполнения* – ***C****ommon* ***L****anguage* ***R****untime*, ***CLR***.

Среда CLR может быть реализована для любой операционной системы. При выполнении программы CLR вызывает так называемый ***JIT***-компилятор, *переводящий код с языка IL в машинные команды конкретного процессора*, которые немедленно выполняются.

JIT означает «just in time», что можно перевести как «вовремя», то есть компилируются только те части программы, которые требуется выполнить в данный момент. Каждая часть программы компилируется один раз и сохраняется в кэше для дальнейшего использования.

Примечание – Кэш – область оперативной памяти, предназначенная для временного хранения информации.

Схема выполнения программы при использовании платформы **.NET** приведена на рисунке:



Компилятор в качестве результата своего выполнения создает так называемую

сборку – файл с расширением **`*.exe`** или **`*.dll`**, который содержит код на **языке `IL`** и метаданные.

Метаданные представляют собой сведения об объектах, используемых в программе, а также сведения о самой сборке. Они позволяют организовать межъязыковое взаимодействие, обеспечивают безопасность и облегчают развертывание приложений, то есть установку программ на компьютеры пользователей.

Примечание – Сборка может состоять из нескольких модулей. В любом случае она представляет собой *программу, готовую для установки* и не требующую для этого ни дополнительной информации, ни сложной последовательности действий. Каждая сборка имеет *уникальное имя*.

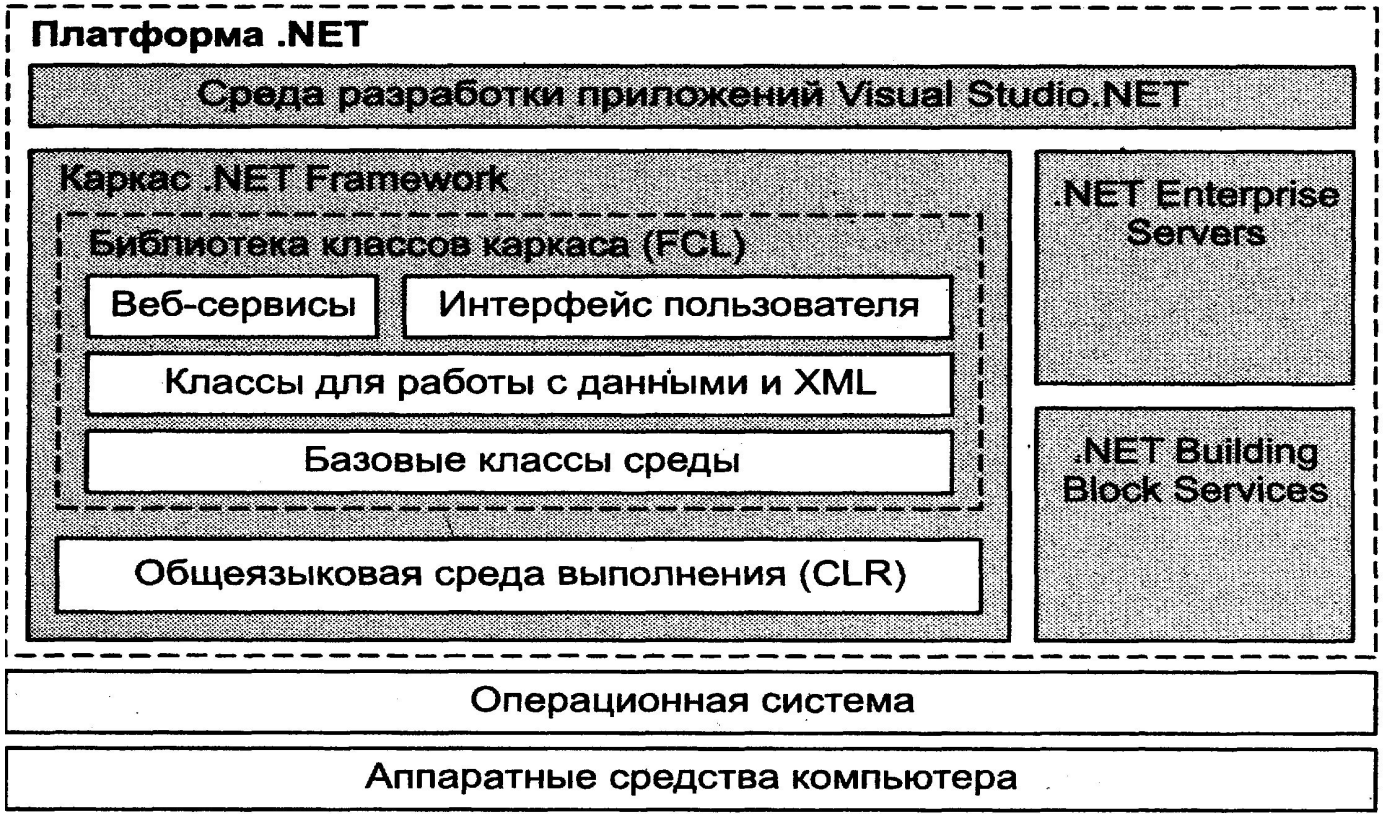
Во время работы программы **среда CLR следит** за тем, чтобы:

- *выполнялись только разрешенные операции,*
- *осуществляет распределение и очистку памяти,*
- *обрабатывает возникающие ошибки*.

Это многократно повышает безопасность и надежность программ.

Платформа .NET содержит огромную библиотеку классов, которые можно использовать при программировании на любом языке .NET.

Общая **структура библиотеки** приведена на рисунке:



Структура платформы **.NET**

Библиотека имеет несколько ***уровней***.

На самом нижнем находятся ***базовые классы*** среды, которые используются при создании любой программы:

- классы ввода-вывода,
- обработки строк,
- управления безопасностью,
- графического интерфейса пользователя,
- хранения данных и пр.

Примечание – Понятие класса рассматривается в последующем. Пока можно считать, что класс служит для реализации некоторых функций.

Над этим слоем находится ***набор классов, позволяющий работать с базами данных и XML***.

Классы самого верхнего уровня поддерживают ***разработку распределенных приложений***, а также ***веб-*** и ***Windows-приложений***.

Программа может использовать классы любого уровня.

Подробное изучение библиотеки классов **.NET** – необходимая, но и наиболее трудоемкая задача программиста при освоении этой платформы.

Библиотека классов вместе с ***CLR*** образуют ***каркас*** (**framework**), то есть основу платформы.

Назначение остальных частей платформы – по мере изучения материала.

Примечание – Термин «приложение» можно воспринимать как синоним слова «программа». Например, вместо фразы «программа, работающая под управлением Windows», говорят «Windows-приложение» или «приложение».

Платформа **.NET** рассчитана на объектно-ориентированную технологию создания программ, поэтому прежде чем начинать изучение языка C#, необходимо познакомиться с *основными понятиями объектно-ориентированного программирования* (ООП).

2. Объектно-ориентированное программирование

Принципы ООП проще всего понять на примере программ моделирования.

В реальном мире каждый предмет или процесс обладает набором статических и динамических характеристик, иными словами, *свойствами* и *поведением*. Поведение объекта зависит от его состояния и внешних воздействий.

Пример. Объект «автомобиль» никуда не поедет, если в баке нет бензина, а если повернуть руль, изменится положение колес.

Понятие объекта в программе совпадает с обыденным смыслом этого слова: объект представляется как

- совокупность данных, характеризующих его состояние, и

- функций их обработки, моделирующих его поведение.

Вызов функции на выполнение часто называют *посылкой сообщения объекту*.

Примечание – Например, вызов функции «повернуть руль» интерпретируется как посылка сообщения «автомобиль, поверни руль!».

При создании объектно-ориентированной программы предметная область представляется в виде ***совокупности объектов***.

Выполнение программы состоит в том, что объекты обмениваются сообщениями. Это позволяет использовать при программировании понятия, более адекватно отражающие предметную область.

При представлении реального объекта с помощью программного необходимо выделить в нем его *существенные особенности*. Их список зависит от *цели моделирования*.

Пример. Объект «крыса» с точки зрения биолога, изучающего миграции, ветеринара или, скажем, повара будет иметь совершенно разные характеристики.

Выделение существенных с той или иной точки зрения свойств называется ***абстрагированием***. Таким образом, программный объект – это абстракция.

Важным свойством объекта является его ***обособленность***.

Детали реализации объекта, то есть внутренние структуры данных и алгоритмы их обработки, скрыты от пользователя объекта и недоступны для непреднамеренных изменений.

Объект используется через его ***интерфейс*** – совокупность правил доступа.

Скрытие деталей реализации называется ***инкапсуляцией*** (от слова «капсула»).

Ничего сложного в понятии ***инкапсуляции*** нет: ведь и в обычной жизни мы пользуемся объектами через их интерфейсы.

Пример. Сколько информации пришлось бы держать в голове, если бы для просмотра новостей надо было знать устройство телевизора!

Таким образом, объект является «черным ящиком», замкнутым по отношению к внешнему миру. Это позволяет представить программу в укрупненном виде – на уровне объектов и их взаимосвязей, а следовательно, управлять большим объемом информации и успешно отлаживать сложные программы.

Сказанное можно сформулировать более кратко и строго:

объект – это инкапсулированная абстракция с четко определенным интерфейсом.

Инкапсуляция позволяет *изменить* ***реализацию*** *объекта без модификации основной части программы, если его интерфейс остался прежним*. *Простота модификации* является очень важным критерием качества программы, ведь любой программный продукт в течение своего жизненного цикла претерпевает множество изменений и дополнений.

Кроме того, инкапсуляция позволяет использовать объект в другом окружении и быть уверенным, что он не испортит не принадлежащие ему области памяти, а также создавать библиотеки объектов для применения во многих программах.

Каждый год в мире пишется огромное количество новых программ, и важнейшее значение приобретает ***возможность многократного использования кода***.

Преимущество ООП состоит в том, что для объекта можно определить ***наследников***, корректирующих или дополняющих его поведение. При этом нет необходимости не только повторять исходный код родительского объекта, но даже иметь к нему доступ.

Наследование является мощнейшим инструментом ООП и применяется для следующих взаимосвязанных целей:

- исключения из программы повторяющихся фрагментов кода;
- упрощения модификации программы;
- упрощения создания новых программ на основе существующих.

Кроме того, только благодаря наследованию появляется возможность *использовать объекты, исходный код которых недоступен*, но в которые требуется внести изменения.

Наследование позволяет создавать ***иерархии объектов***.

Иерархия представляется в виде ***дерева***, в котором более общие объекты располагаются ближе к корню, а более специализированные – на ветвях и листьях. Наследование облегчает использование библиотек объектов, поскольку программист может взять за основу объекты, разработанные кем-то другим, и создать наследников с требуемыми свойствами.

Объект, на основании которого строится новый объект, называется ***родительским объектом***, ***объектом-предком***, ***базовым классом***, или ***суперклассом***,

а унаследованный от него объект – ***потомком***, ***подклассом***, или ***производным*** *классом*.

ООП позволяет писать гибкие, расширяемые и читабельные программы. Во многом это обеспечивается благодаря

полиморфизму, под которым понимается *возможность во время выполнения программы с помощью одного и того же имени выполнять разные действия или обращаться к объектам разного типа*.

Чаще всего понятие полиморфизма связывают с механизмом виртуальных методов, который мы рассмотрим далее.

Подводя итог сказанному, сформулируем ***достоинства ООП***:

- использование при программировании понятий, близких к предметной области;
- возможность успешно управлять большими объемами исходного кода благодаря инкапсуляции, то есть скрытию деталей реализации объектов и упрощению структуры программы;
- возможность многократного использования кода за счет наследования;
- сравнительно простая возможность модификации программ;
- возможность создания и использования библиотек объектов.

Эти преимущества особенно явно проявляются при разработке *программ большого объема* и классов программ.

Однако ничто не дается даром: создание объектно-ориентированной программы представляет собой весьма *непростую задачу*, поскольку *требует разработки иерархии объектов*, а плохо спроектированная иерархия может свести к нулю все преимущества объектно-ориентированного подхода.

Кроме того, идеи ООП не просты для понимания и, в особенности, для практического применения. Чтобы эффективно использовать готовые объекты из библиотек, необходимо освоить большой объем достаточно сложной информации.

Неграмотное же применение ООП способно привести к созданию излишне сложных программ, которые невозможно отлаживать и усовершенствовать.

3. Классы

Для представления объектов в языках C#, Java, C++, Delphi и т. п. используется понятие класс, аналогичное обыденному смыслу этого слова в контексте «класс членистоногих», «класс млекопитающих», «класс задач» и т. п.

Класс является обобщенным понятием, определяющим характеристики и поведение некоторого множества конкретных объектов этого класса, называемых экземплярами класса.

«Классический» класс содержит:

- ***данные***, задающие ***свойства*** объектов класса, и
- ***функции***, определяющие их ***поведение***.

В последнее время в класс часто добавляется ***третья составляющая – события***, на которые может реагировать объект класса.

Примечание – Это оправдано для классов, использующихся в программах, построенных на основе событийно-управляемой модели, например, при программировании для Windows.

Все классы библиотеки .NET, а также все классы, которые создает программист в среде .NET, имеют ***одного общего предка*** – класс **`object`** и организованы в ***единую иерархическую структуру***.

Внутри нее классы логически сгруппированы в так называемые ***пространства имен***, которые служат для *упорядочивания имен классов и предотвращения конфликтов имен*: в разных пространствах имена могут совпадать. Пространства имен могут быть вложенными, их идея аналогична знакомой вам иерархической структуре каталогов на компьютере.

Любая программа, создаваемая в .NET, использует ***пространство имен*** **`System`**.

В нем определены классы, которые **обеспечивают *базовую функциональность***, например, поддерживают:

- выполнение математических операций,
- управление памятью,
- ввод-вывод.

Обычно в одно пространство имен объединяют взаимосвязанные классы.

`Пример`. Пространства содержат:

- **`System.Net`** - классы, относящиеся к передаче данных по сети,
- **`System.Windows.Forms`** – элементы графического интерфейса пользователя, такие как формы, кнопки и т. д. Имя каждого пространства имен представляет собой неделимую сущность, однозначно его определяющую.

Последнее, о чем необходимо поговорить, прежде чем начать последовательное изучение языка C#, – среда разработки Visual Studio .NET.

4. Среда Visual Studio.NET

Среда разработки Visual Studio.NET предоставляет *мощные и удобные средства написания, корректировки, компиляции, отладки и запуска приложений*, использующих **.NET**-совместимые языки.

Корпорация Microsoft включила в платформу *средства разработки для четырех языков: C#, VB.NET, C++ и J#.*

Платформа **.NET** является ***открытой средой***. Это значит, что компиляторы для нее могут поставляться и сторонними разработчиками.

К настоящему времени разработаны десятки компиляторов для **.NET**, например, Ada, COBOL, Delphi, Eiffel, Fortran, Lisp, Oberon, Perl и Python.

Все **.NET**-совместимые языки должны отвечать требованиям ***общеязыковой спецификации*** – ***C****ommon* ***L****anguage* ***S****pecification*, ***CLS***, – в которой описывается набор общих для всех языков характеристик.

Это позволяет использовать для разработки приложения несколько языков программирования и вести полноценную межъязыковую отладку.

Все программы независимо от языка используют одни и те же базовые классы библиотеки **.NET**.

Приложение в процессе разработки называется ***проектом***.

Проект объединяет все необходимое для создания приложения: файлы, папки, ссылки и прочие ресурсы.

Среда **Visual Studio .NET** позволяет создавать ***проекты различных типов***, например:

- ***Windows-приложение*** использует элементы интерфейса Windows, включая формы, кнопки, флажки и пр.;
- ***консольное*** приложение выполняет вывод «на консоль», то есть в окно командного процессора;
- ***библиотека*** *классов* объединяет классы, которые предназначены для использования в других приложениях;
- ***веб-приложение*** – это приложение, доступ к которому выполняется через браузер (например, Internet Explorer) и которое по запросу формирует веб-страницу и отправляет ее клиенту по сети;
- ***веб-сервис*** – компонент, методы которого могут вызываться через Интернет.

Несколько проектов можно объединить в ***решение –*** solution.

Это облегчает совместную разработку проектов.

5. Консольные приложения

Среда Visual Studio.NET работает на платформе Windows и ориентирована на создание Windows- и веб-приложений, однако разработчики предусмотрели работу и с консольными приложениями.

При запуске консольного приложения операционная система создает так называемое консольное окно, через которое идет весь ввод-вывод программы. Внешне это напоминает работу в операционной системе в режиме командной строки, когда ввод-вывод представляет собой поток символов.

Консольные приложения наилучшим образом подходят для изучения языка, так как в них не используется множество стандартных объектов, необходимых для создания графического интерфейса.

Вначале будем создавать только консольные приложения, чтобы сосредоточить внимание на базовых свойствах языка C#.

Рассмотрим самые простые действия в среде: создание и запуск на выполнение консольного приложения на C#.

Создание проекта. Основные окна среды

Для создания проекта следует после запуска **Visual Studio .NET** в главном меню выбрать команду **File ► New ► Project...**

В левой части открывшегося диалогового окна нужно выбрать пункт **Visual C# Projects**, в правой – пункт **Console Application**.

В поле **Name** можно ввести *имя проекта*, а

в поле **Location** – *место его сохранения на диске*, если заданные по умолчанию значения вас не устраивают.

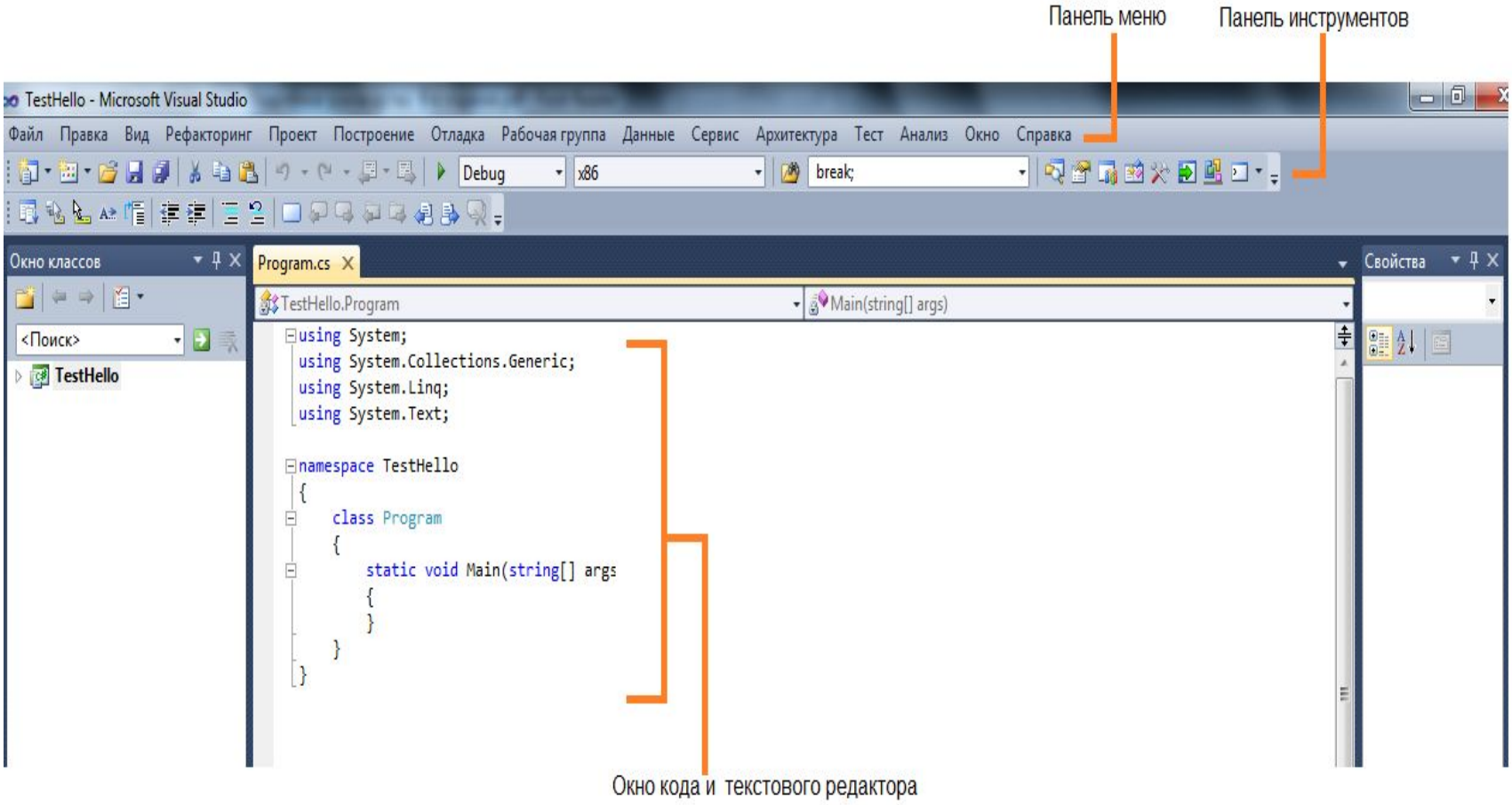
После щелчка на кнопке ОК среда создает решение и проект с указанным именем.

Примерный вид экрана приведен на рисунке.

В верхней части экрана располагается ***главное меню*** (с разделами File, Edit, View и т. д.) и ***панели инструментов*** (toolbars).

Панелей инструментов в среде великое множество, и если включить их все (View ► Toolbars...), они займут половину экрана.

Примерный вид экрана после создания проекта консольного приложения приведен на рисунке



Начальный код проекта

В верхней левой части экрана располагается окно управления проектом **Solution Explorer** (если оно не отображается, следует воспользоваться командой View ► Solution Explorer главного меню). В окне перечислены все ресурсы, входящие в проект: ссылки на библиотеку (System, System.Data, System.XML), файл ярлыка (App.ico), файл с исходным текстом класса (Classl .cs) и информация о сборке (Assemblylnfo.cs).

Примечание – Как говорилось ранее, сборка является результатом работы компилятора и содержит код на промежуточном языке и метаданные.

В этом же окне можно увидеть и другую информацию, если перейти на вкладку Class View, ярлычок которой находится в нижней части окна. На вкладке Class View представлен список всех классов, входящих в приложение, их элементов и предков.

Примечание – Какие же файлы создала среда для поддержки проекта? С помощью проводника Windows можно увидеть, что на заданном диске появилась папка с указанным именем, содержащая несколько других файлов и вложенных папок. Среди них – файл проекта (с расширением **`csproj`**), файл решения (с расширением **`sln`**) и файл с кодом класса (**`Class1.cs`**).

В нижней левой части экрана расположено окно свойств **`Properties`** (если окна не видно, воспользуйтесь командой View ► Properties главного меню). В окне свойств отображаются важнейшие характеристики выделенного элемента.

Например, чтобы изменить имя файла, в котором хранится класс Classl, надо выделить этот файл в окне управления проектом и задать в окне свойств новое значение свойства FileName (ввод заканчивается нажатием клавиши Enter).

Основное пространство экрана занимает окно редактора, в котором располагается текст программы, созданный средой автоматически. Текст представляет собой каркас, в который программист добавляет код по мере необходимости.

Ключевые (зарезервированные) слова отображаются синим цветом, комментарии различных типов – серым и темно-зеленым, остальной текст – черным.

Примечание –

Ключевые слова – это слова, имеющие специальное значение для компилятора. Их можно использовать только в том смысле, в котором они определены.

Комментарии предназначены для программиста и позволяют обеспечить документирование программы.

Слева от текста находятся символы структуры: щелкнув на любом квадратике с минусом, можно скрыть соответствующий блок кода. При этом минус превращается в плюс, щелкнув на котором, можно опять вывести блок на Экран. Это средство хорошо визуально структурирует код и позволяет сфокусировать внимание на нужных фрагментах.

Заготовка консольной программы. Рассмотрим каждую строку заготовки программы. Ваша цель – изучить принципы работы в оболочке, а досконально разбираться в программе мы будем позже.

```
using System;//директива разрешает использовать имена
стандартных
// классов из пространства имен System непосредственно
namespace ConsoleApplication1//создает для проекта собственное
//пространство имен, названное по умолчанию ConsoleApplication1
{
/// <summary>//Строки, начинающиеся с двух или трех косых черт,
//являются комментариями
/// Summary description for Class1.
/// </summary>
class Class1 //Описание класса Class1
{
/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main(string[] args)
{
//
// TODO: Add code to start application her
//
```

В данном случае внутри класса только один элемент – метод **Main**. Каждое приложение должно содержать метод **Main** – с него начинается выполнение программы. Все методы описываются по единым правилам.

Упрощенный синтаксис метода:

```
[спецификаторы] тип имя_метода ([параметры])
{
тело метода: действия, выполняемые методом
}
```

Среда заботливо поместила внутрь метода **Main** комментарий:

```
// TODO: Add code to start application here
```

Это означает: «Добавьте сюда код, выполняемый при запуске приложения».

Последуем совету и добавим после строк комментария (но не в той же строке!) строку

```
Console.WriteLine("Ур-ра!Зар-работало!(с)Кот Матроскин");
```

Здесь **Console** – это имя стандартного класса из пространства имен **System**. Его метод **WriteLine** выводит на экран заданный в кавычках текст. Как видите, для обращения к методу класса используется конструкция **имя_класса.имя_метода**

Программа должна приобрести вид, приведенный в листинге (для того чтобы вы могли сосредоточиться на структуре программы, из нее убраны все комментарии и другие пока лишние для нас детали):

```
using System;
namespace ConsoleApplication1
{
   class Class1
      {
         static void Main()
         {
Console.WriteLine( "Ур-ра! Зар-работало! (с) Кот Матроскин" );
         }
      }
}
```

Запуск программы

Самый простой способ запустить программу – нажать клавишу F5 (или выбрать в меню команду Debug ► Start), Если программа написана без ошибок, то фраза

Ур-ра! Зар-работало! (с) Кот Матроскин

промелькнет перед вашими глазами в консольном окне, которое незамедлительно закроется. Это хороший результат, но для того чтобы пронаблюдать его спокойно, следует воспользоваться клавишами Ctrl+F5 (или выбрать в меню команду Debug ► Start Without Debugging).

После внесения изменений компилятор может обнаружить в тексте программы синтаксические ошибки. Он сообщает об этом в окне, расположенном в нижней части экрана.

Итак,
среда разработки Visual Studio.NET предоставляет программисту ***мощные и удобные средства написания, корректировки, компиляции, отладки и запуска приложений***.

В процессе изучения языка C# желательно постепенно изучать эти возможности, ведь чем лучше вы будете владеть инструментом, тем эффективнее и приятнее будет процесс программирования.