**План ознакомления:**

1. **.NET framework** – программная платформа

* Среда выполнения программ – CLR (Common Language Runtime). CLR – отвечает за исполнение .NET программ и предоставляет богатую библиотеку базовых классов. CLR управляет памятью, обслуживает приложения, обрабатывает потоки и выполняет различные проверки, связанные с безопасностью.
* Библиотека системных классов – CTS (Common Type System). Представляет полное описание поддерживаемых XLR типов данных и программных конструкций.
* Общеязыковая спецификация – CLS (Common Language Specification). Спецификация, в которой описано лишь то подмножество общих типов и программных конструкций, каковое способны воспринимать абсолютно все поддерживающие .NET языки программирования.
* Библиотека базовых классов. Содержит определения различных примитивов: потоки, файловый ввод-вывод,

Основными языками разработки для .NET fx были выбраны – C# и vb.net.

Язык C# сильно развивается, и с каждой версией в него вносятся новые изменения, которые делают его все более и более удобным для разработки. Текущая версия C# 6.

Программы на C#, vb.net и других .NET языков компилируются в MSIL код, совместимый для всех языков. Это дает возможность разрабатывать компоненты программы на разных языках. Возможно выполнять наследование классов, написанных на C#, в программах на vb.net (межязыковое наследование).

Весь код, выполняемый в среде .NET (CLR) называется управляемым (managed code). Есть возможность взаимодействовать с неуправляемым кодом, который выполняется вне .NET fx.

**Библиотека базовых классов**

Доступ к БД

API к GUI

Безопасность

API удаленной работы

Потоковая обработка

Файловый ввод-вывод

API для работы с веб

И другие

**Общеязыковая исполняющая среда (CLR)**

Общая система типов (CTS)

Общеязыковая спецификация (CLS)

1. С одной стороны .NET fx походит на JAVA Virtual machine – программа компилируется не в машинные коды, а в промежуточное представление (**MSIL**). С другой в .NET был применен иной подход:

* В JVM – байткод интерпретировался во время выполнения.
* В .NET – программа компилируется в промежуточное представление (MSIL), которое преобразуется в машинные коды в момент выполнения.

За перевод программы в машинные коды в момент выполнения отвечает JIT (Just In Time) компилятор. Причем JIT компилирует не всю программу целиком, а только те функциональные части, которые нужны для выполнения программы в данный момент. Если встречается вызов метода из сборки, который еще не скомпилирован, JIT выполняет компиляцию и передает управление в свежескомпилированный код.

С **JIT** связаны плюсы и минусы .NET:

**++** - программа компилируется в машинные коды, наиболее оптимальные для текущего аппаратного обеспечения (использование регистров, наборов команд)

**--** - из-за компиляции по мере необходимости, могут возникать «фризы» в первые запуски. Второй и последующие вызовы выполняются из уже скомпилированных кусочков и выполняются быстрее.

1. В .NET fx встроена автоматическая **сборка мусора**. Это означает, что с разработчика снимается необходимость следить за выделенными ресурсами.

Среда исполнения самостоятельно следит за используемыми объектами, и как только на объект не остается ни одной ссылки, объект удаляется из памяти.

1. **Язык C#** - основной язык разработки на .NET.

C# - язык ООП. Нет функций/методов или переменных не принадлежащих какому-либо классу (глобальных переменных). Все существует только в контексте какого-то класса.

C# может выступать как язык функционального программирования (когда функция является переменной более высокого порядка)

Также C# может применяться как язык компонентно-ориентированной, аспектной и событийной парадигмы программирования. Но в любом случае – в основе ООП подход.

*Более подробно язык посмотрим далее на примерах.*

Далее с демонстрацией в Visual Studio 2015

1. **Visual Studio** – среда разработки на C# (IDE).

Разработана Microsoft и наиболее полно поддерживает разработку на C#.

Имеет большой набор подготовленных шаблонов для создания .NET компонент.

Шаблон – это готовый проект, с преднастроенным множеством библиотек, классов, базовыми настройками для компилятора, некоторый набор классов.

*// Создаем простейшее консольное приложение HelloWorld*

1. **Проект** – базовая единица разработки. На выходе файл dll или exe.
2. В простейшем проекте (**Project**) мы видим уже подготовленный набор файлов. Несколько установленных ссылок.

Код программы уже частично написан. Точка входа определена. Определен какой-то базовый класс. Средствами VS можно переименовать все вхождения класса (в рамках Solution).

Просмотрев настройки проекта можно увидеть, что для проекта уже заданы настройки, чтобы компилятор собирал консольное приложение/библиотеку/WinForm приложение.

1. Проект не может существовать без Решения (**Solution**). VS всегда по умолчанию создает Solution для проекта. Проект может входить в множество решений.

Solution – это логическая единица, позволяющая собрать в одной сессии IDE те проекты, над которыми одновременно ведется работа (связанные проекты).

1. Следующий элемент платформы .NET - понятие Сборки (**Assembly**).

Сборка – это любой .exe или .dll файл, который получается в результате компиляции проекта. Отличие exe от dll большей частью заключается в наличии в первом типе файлов точки входа, куда передается управление при запуске

Сборка – это базовая структурная единица .NET , на уровне которой проходит контроль версий, развертывание и конфигурация приложения.

Сборки имеют:

* Манифест, который содержит метаданные сборки (имя, версию, язык, строгое имя, ссылки на другие сборки, ссылки на типы)
* Метаданные типов, описанных в сборке
* MSIL код
* Ресурсы

1. Разделяемые сборки – **GAC**

Некоторые сборки, которые используются несколькими приложениями можно размещать в глобальном кэше – GAC.

Сборка помещается в GAC с помощью специальной утилиты gacutil.exe.

На сборки в GAC можно ссылаться без указания пути расположения.

Сборка в GAC имеет «строгое имя». И если в пользовательском приложении указать ссылку на эту сборку, среда выполнения будет в первую очередь брать сборку из GAC и только потом из локального пути. *Это осложняет обновление сборок в будущем.*

1. В .NET (и в C# в частности) введена логическая единица – пространство имен (**namespace**), которое позволяет группировать классы в какие-то логические группы. Каждый класс должен принадлежать какому-то пространству имен. При создании проекта базовое пространство создается автоматически.

namespace – может быть составным: **GPB.DBProviders.Oracle** (объединяет все классы для работы с Oracle)

Класс, видит все классы, объявленные в том же namespace, что и он.

К конкретному классу можно обращаться по полному пути – с указанием всего namespace: System.IO.FileDescription, либо можно подключить пространство имен к текущему (делает доступным все классы объявленные внутри namespace):

using GPB.DBProviders.Oracle

С помощью команды using можно настраивать псевдоним для какого-то конкретного класса:

using MyFileLis = System.Collection.List<System.IO.FileDescription>;

Новая возможность C# 6.0 (Visual Studio 2015) – импорт функциональности класса (статических методов):

using static System.Console

после чего в программе становятся доступны статические функции класса System.Console:

WriteLine, ReadKey и т.п.

1. **Определение переменных**.

При определении переменной, необходимо указать ее тип и имя:

string str;

Можно сразу же инициировать ее каким-то значением:

string str = “”;

string str = “Тестовая строка”;

В свежих версиях C# возможно использовать вывод типа переменной:

var str = “test string”;

В зависимости от типа правой части определяется тип переменной слева. При этом VS уже на этапе работы с кодом будет определять тип и выдавать соответствующие подсказки.

1. **Типы в C#**

В C# выделяют две разновидности типов:

* **Типы значений**
* **Ссылочные типы**

Основные типы или определенные пользователем структуры – типы значений. string, массивы и классы – это ссылочные типы.

Отличие заключается в том, где хранятся значения этих типов и как с ними работать при передаче в качестве параметров функции.

**Тип значения** хранит свое содержимое в памяти, выделенной в стеке. При передачи в качестве параметра, будет создана копия значения.

**Ссылочный тип** размещается в куче. Эта память не возвращается в кучу при завершении метода, она освобождается системой сборки мусора.

**Упаковка и распаковка** – два встречных процесса преобразования значения из типа значения в ссылочный и наоборот. Среда .NET автоматически применяет методы упаковки и распаковки при необходимости.

int intValue = 4;

System.Console.WriteLine(intValue.ToString()) // <- вызов упаковки

Переменная типа значения не может быть NULL

1. **Типы значений** – производные от System.ValueType:

* Целочисленные типы (byte, int, long)
* Типы с плавающей запятой
* decimal
* bool
* Структуры (**struct**)
* Перечисления (**enum**)

Переменная типа значения не может содержать значение null.

Перечисления – это набор поименованных констант. Enum представляется в памяти целочисленным типом. Возможно обратное преобразование.

enum Range : long { Max = 2147483648L, Min = 255L };

Возможно описать перечисление с атрибутом Flags, который позволит использовать элементы с бинарными операциями ИЛИ или И.

[Flags]

public enum CarOptions

{ SunRoof = 0x01, Spoiler = 0x02, FogLights = 0x04, TintedWindows = 0x08, }

1. Особенность хранения строк в .NET

Многострочные значения.

Специальные символы.

String.Format()

1. Ссылочные типы – все типы, относледованные от System.Object. Все классы.
2. **Управляющие структуры**:
   1. for
   2. foreach
   3. while
   4. do… while (true)
   5. if

Логические операции ==, !=, <=, >=, <, >

* 1. switch (case…: break; default:…)

1. **Описание функций**

В C# есть только функции. Если функция не возвращает результата, при ее описании в качестве возвращаемого значения указывают **void**.

[<модификатор прав доступа>] [static] <возвращаемый тип> <имя функции>(<параметры>) {<тело функции>}

private int GetInt(string strValue) {return int.Parse(strValue);}

public static GetStr (double inValue) {return inValue.ToString();}

1. **Позиционные** и **Именованные** параметры функций
2. **Перегрузка функций** – задание функции с тем же именем, но иной ***сигнатурой***
3. **Параметры по умолчанию** – можно задать значение по умолчанию для параметра функции. Если пользователь явно не укажет значение, будет использоваться то, что указано при описании функции.

В качестве значения можно указывать только такое, которое может быть определено во время компиляции (указать DateTime.Now нельзя).

1. **Модификаторы прав доступа**:

* **private**
* **protected**
* **public**
* **internal** (для классов) – класс будет доступен внутри проекта, но не будет доступен внешним приложениям (где сборка подключена по ссылке)
* **protected internal**

1. **Static** поля и методы

Имеется возможность описывать методы и поля, которые могут быть доступны без создания объектов класса. Такие поля и методы называют статическими (**Static**).

Аналог статических полей и методов – глобальные переменные и методы.

Используются чаще всего для скрытия в классе функционала, единого для всех объектов класса.

Бывают статические классы – это классы, у которых нельзя создать экземпляр.

1. **Поля** – это переменные, объявленные в области класса.
2. **Свойства** – это методы, доступ к которым осуществляется также как к полям, но есть возможность добавить дополнительную логику на операции получения и установки значения.

Можно сделать ограниченным доступ на изменения значения, но оставить публичным получение значения свойства.

1. Константы (**const**) – поля, значения которых устанавливаются во время написания программы и не изменяются в ходе выполнения.
2. **Abstract** class – абстрактные классы. Определяют класс, у которого нельзя создать экземпляр. Используются для описания общего функционала для ряда классов, с предопределенной общей функциональностью.
3. **Interface** – определяет набор свойств и методов класса. Не содержит реализации. Предъявляет требования к функционалу производного класса.
4. **Delegate** – определяет сигнатуру функции. Используются для описания событий.
5. События (**event**) – специальный механизм, позволяющий уведомлять другие объекты о различных случаях (нажатие кнопки, вывод в лог и т.п.)
6. **Attribute** – конструкция платформы .NET, позволяющая добавить метаданных классам. Некоторые атрибуты используются компилятором, но никто не запрещает использовать атрибуты для работы собственных приложений.
7. **Массивы, Коллекции.** Пространство имен System.Collection.
8. **Generic-коллекции.**
   1. List<T>
   2. SortedList<TKey, TValue>
   3. Queue<T>
   4. Stack<T>
9. **Сортировка** коллекций.
10. Использование **Generic’**ов в функциях
11. **Event** и **Delegate.** Описание, использование.
12. **ООП**
    1. **Инкапсуляция** – все данные внутри класса.
    2. **Полиморфизм** – описание функций как Virtual. Использование override.
    3. **Наследование**  - наследование только от одного класса. Но возможна реализация многих интерфейсов.
13. Extension методы
14. Классы, **неявное приведение типов**:

Public static implicit <TargetType>(ObjectType object)

1. Классы, **иттераторы**