Introduction to .NET

The philosophy behind .NET

- .NET and C# were introduced in 2002
  - Still very popular today
  - C# is one of the most used languages


.NET Framework
.NET Core
.NET

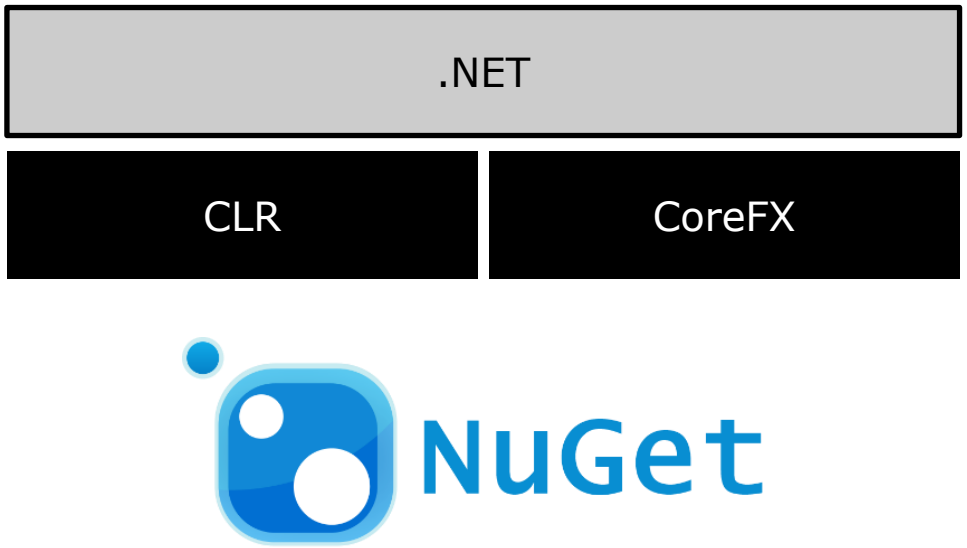
The philosophy behind .NET

- Key benefits of .NET
  - Numerous programming languages
    - C#, VB, F#...
  - A common runtime engine shared by all .NET-aware languages
  - Language integration
  - Base class library
  - Simplified deployment
  - Not tied only to Windows!!
    - MacOS, Linux can also run .NET code!

CLR, CTS, CLS

- .NET == a runtime + a base class library
- CLR = The runtime (Common Language Runtime)
  - Locate, load, and manage .NET objects for us
  - Also: memory management, application hosting, coordinating threads, and performing security checks
  - And basically a whole lot more of the low-level stuff!
- BCS or CoreFX = Base class library
  - A library with ready-to-go types to be used in your applications.

.NET



.NET is..

- Cross platform
- Fast
- Lightweight
- Open source

DotNet CLI

- Hosts the CLR
- Executes the app
- Cross-platform
- SDK

DotNet CLI: The demo

C#

- C# is one of the programming languages of .NET
  - Syntax is similar to Java
  - Are evolutions of C
    - Reason why they are similar
  - Also heavy influence of VB
    - Properties
    - Optional parameters
    - Enumerations
    - …
  - Also influenced by Haskell, LISP (functional languages)
    - Lambda expressions
    - Anonymous types
- → Cleaner than Java
- → Simple like VB
- → Powerful like C++
- It's a good choice to attend this course!

C#

- Core features of C# (1)
  - Automatic memory management through garbage collection
  - Formal syntactic constructs for classes, interfaces, structures, enumerations, and delegates
  - Overloading of operators
  - Attribute-based programming
  - Generic types and generic members
  - Anonymous methods
  - Partial classes
  - LINQ
  - Extension methods
  - Lambda expressions (=>)
  - Object and collection initialization

C#

- Optional parameters
- dynamic keyword
- Await/async support
  - Much easier to work with async development
  - Mainly added to support Windows 8/WinRT development

Other .NET languages

- VS supports
  - C#
  - VB
  - C++
  - F#

Why C# then?

- More than one language exists
- They all compile to same IL (Intermediate Language)

- All languages have their strengths and weaknesses
  - Mathematics
  - Graphical
  - …
- Once you know .NET, the switch from one language to another is pretty simple!

Assemblies in .NET

- Managed (compiled) code becomes a DLL or an EXE
  - Same as unmanaged from the outside
  - Entirely different though
  - .NET binaries do not contain platform-specific instructions, but rather platform-agnostic Intermediate Language(IL) and type metadata

Assemblies in .NET and CIL

- Compilation creates an assembly
  - Contains IL (Intermediate Language)
  - Not yet platform-specific code
  - Only happens when the code is really used (referenced for example)
- Assemblies also contain type metadata
  - Describes the characteristics of every “type” within the binary
    - Implemented interfaces
    - Member description…
- Assembly also describes itself using metadata

Some IL (C#)

```
// Calc.cs
using System;

namespace CalculatorExample
{
  // This class contains the app's entry point.
  class Program
  {
    static void Main()
    {
      Calc c = new Calc();
      int ans = c.Add(10, 84);
      Console.WriteLine("10 + 84 is {0}.", ans);

      // Wait for user to press the Enter key before shutting down.
      Console.ReadLine();
    }
  }

  // The C# calculator.
  class Calc
  {
    public int Add(int x, int y)
    { return x + y; }
  }
}
```

```
.method public hidebysig instance int32 Add(int32 x,
   int32 y) cil managed
{
  // Code size 9 (0x9)
  .maxstack  2
  .locals init (int32 V_0)
  IL_0000:  nop
  IL_0001:  ldarg.1
  IL_0002:  ldarg.2
  IL_0003:  add
  IL_0004:  stloc.0
  IL_0005:  br.s IL_0007
  IL_0007:  ldloc.0
  IL_0008:  ret
} // end of method Calc::Add
```

Compilation to platform-specific code

- Code will be compiled on the fly before use
  - JIT Compiler (JIT compilation)
  - The “Jitter”
  - A specific Jitter exists for different CPUs
    - Low memory
    - Mobile
- Compiled code is cached
  - First invocation is slower
  - Subsequent ones are fast

The Common Type System

- Type:
  - Class
  - Interface
  - Structure
  - Enumeration
  - delegate
    - Constantly used in our own code
- CTS: formal specification that documents how types must be defined in order to be hosted by the CLR
  - Most of the time not relevant for us (unless you build a compiler in your spare time!)
  - Supported by the .NET-aware languages