Advanced Development Techniques

DLL
Reflection

Advanced Development Techniques

DLL

Reflection

Executable files

- **DLL is a file type which contains code which can be run.**
- **Binary executables**
  - https://en.wikipedia.org/wiki/Comparison_of_executable_file_formats
  - Container format: executable code and all required data in one bundle
  - Same parts: Header, Imports, Data (RW/RO), Code Segment/.Text
- **Linux: ELF = Executable and Linkable Format**
  - Typically a binary executable (not a script executable!) or SO file
  - Extra feature: fatELF = several platform-dependent executables in one platform-independent file
- **Windows: PE = Portable Executable**
  - All EXE/DLL file
  - Extra feature: icon in the file (with ELF: elfres), „more simple” (non-global) import namespaces
  - https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/corkami/PE101-v20L.png

Executable files

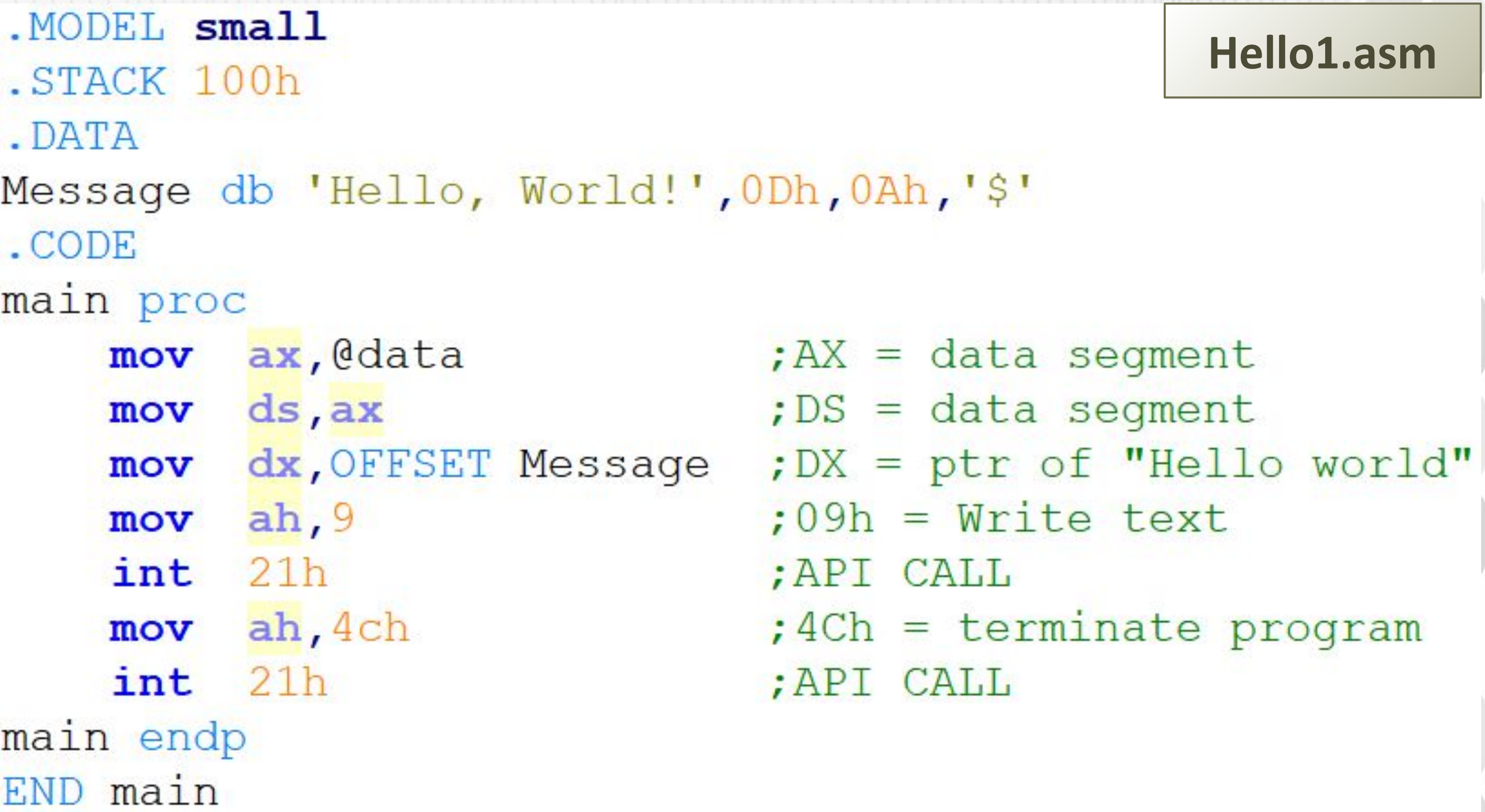
These (DLL and SO files) are not standalone runnable/executable. If you double click on them, nothing happens!

These can be used by “real” executable files, eg. a something.exe can use a someOtherThing.dll.

To better understand these, let’s see in depth what happens during compile process of a source code.

The most simple „Hello, World”

Hello1.asm

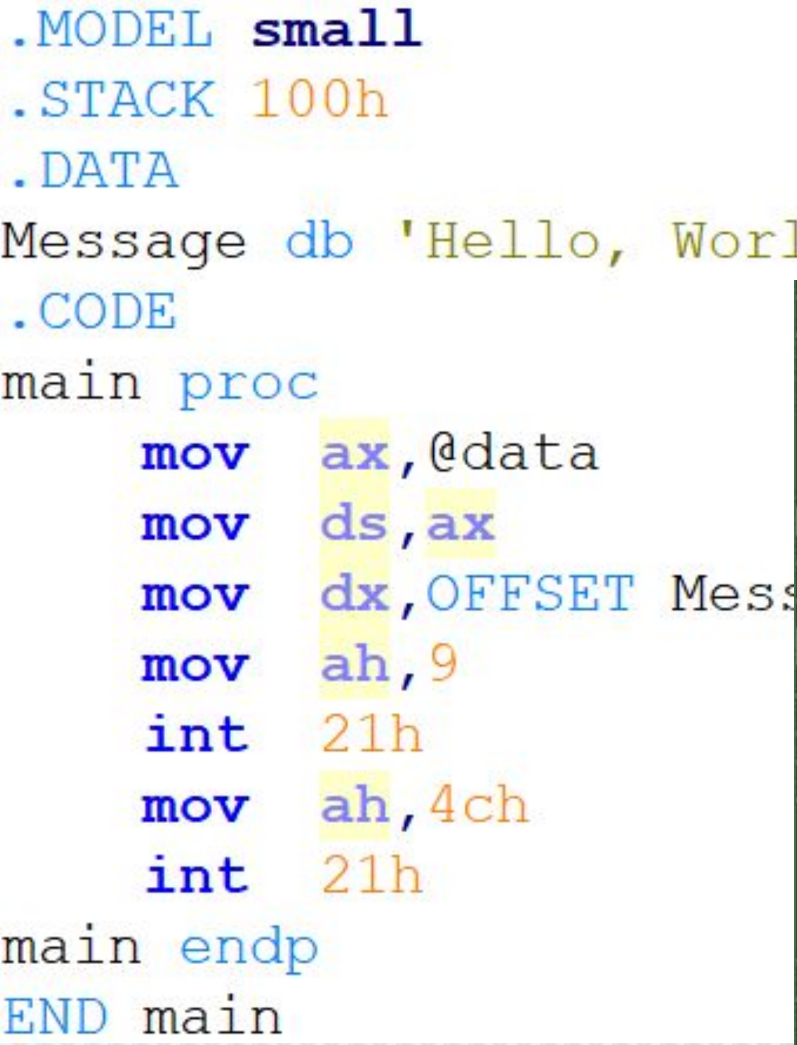


```
.MODEL small
.STACK 100h
.DATA
Message db 'Hello, World!',0Dh,0Ah,'$'
.CODE
main proc
    mov  ax,@data                ;AX = data segment
    mov  ds,ax                   ;DS = data segment
    mov  dx,OFFSET Message       ;DX = ptr of "Hello world"
    mov  ah,9                    ;09h = Write text
    int  21h                     ;API CALL
    mov  ah,4ch                  ;4Ch = terminate program
    int  21h                     ;API CALL
main endp
END main
```

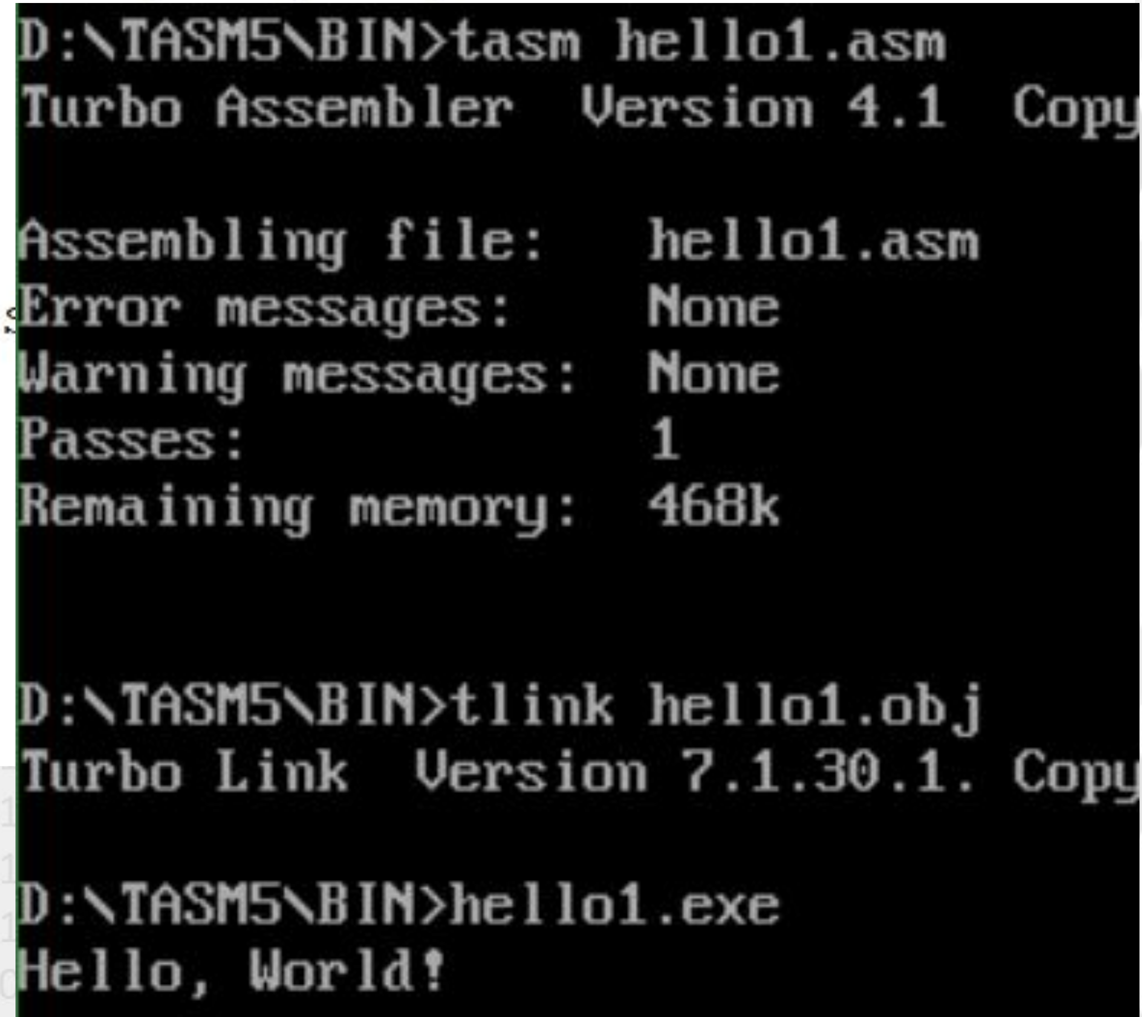
Compile of source code = compile + linking

The most simple „Hello, World”

Hello1.asm



```
.MODEL small
.STACK 100h
.DATA
Message db 'Hello, World!',0Dh,0Ah,'$'
.CODE
main proc
    mov   ax,@data
    mov   ds,ax
    mov   dx,OFFSET Mes
    mov   ah,9
    int   21h
    mov   ah,4ch
    int   21h
main endp
END main
```



```
D:\TASM5\BIN>tasm hello1.asm
Turbo Assembler  Version 4.1  Copy

Assembling file:   hello1.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  468k


D:\TASM5\BIN>tlink hello1.obj
Turbo Link  Version 7.1.30.1. Copy


D:\TASM5\BIN>hello1.exe
Hello, World!
```

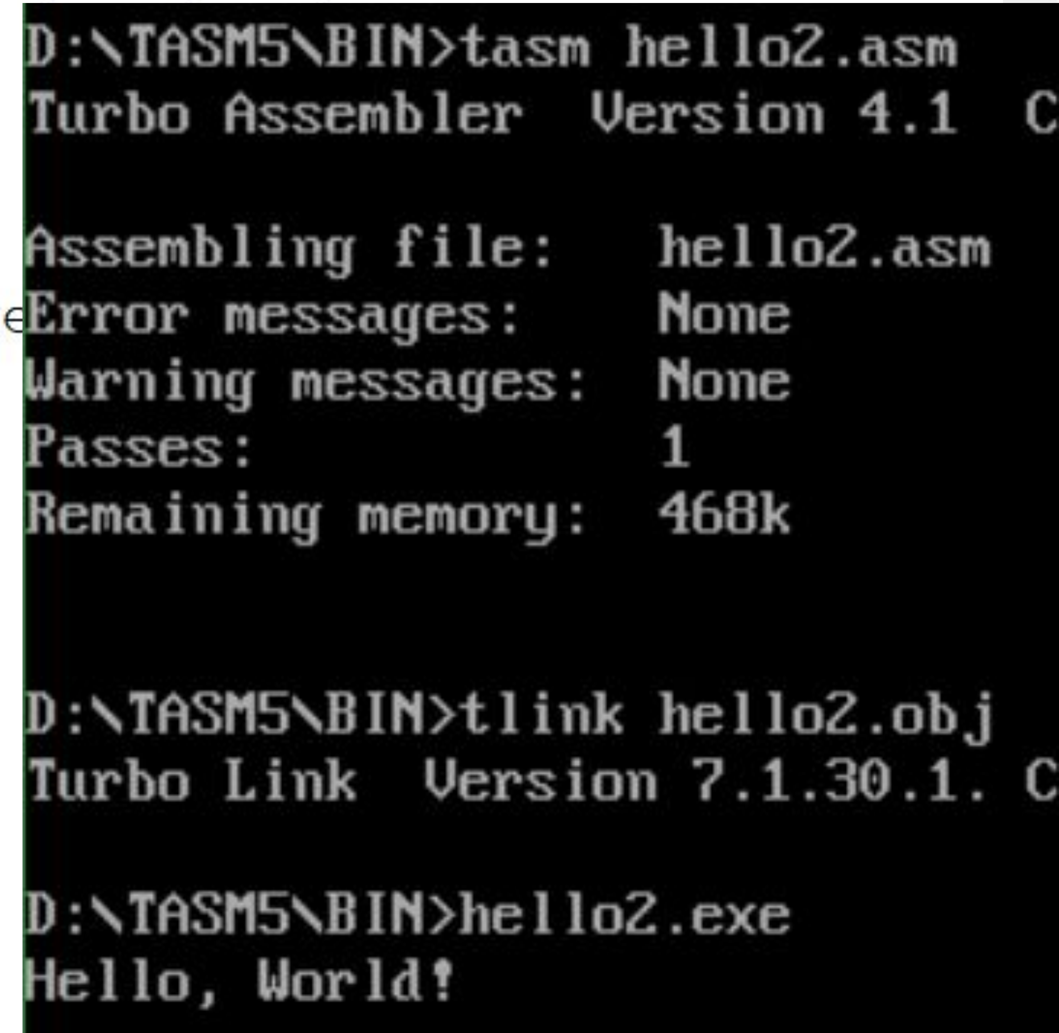
source code → obj binary machine code → linking → executable

V1.1

With subroutine call

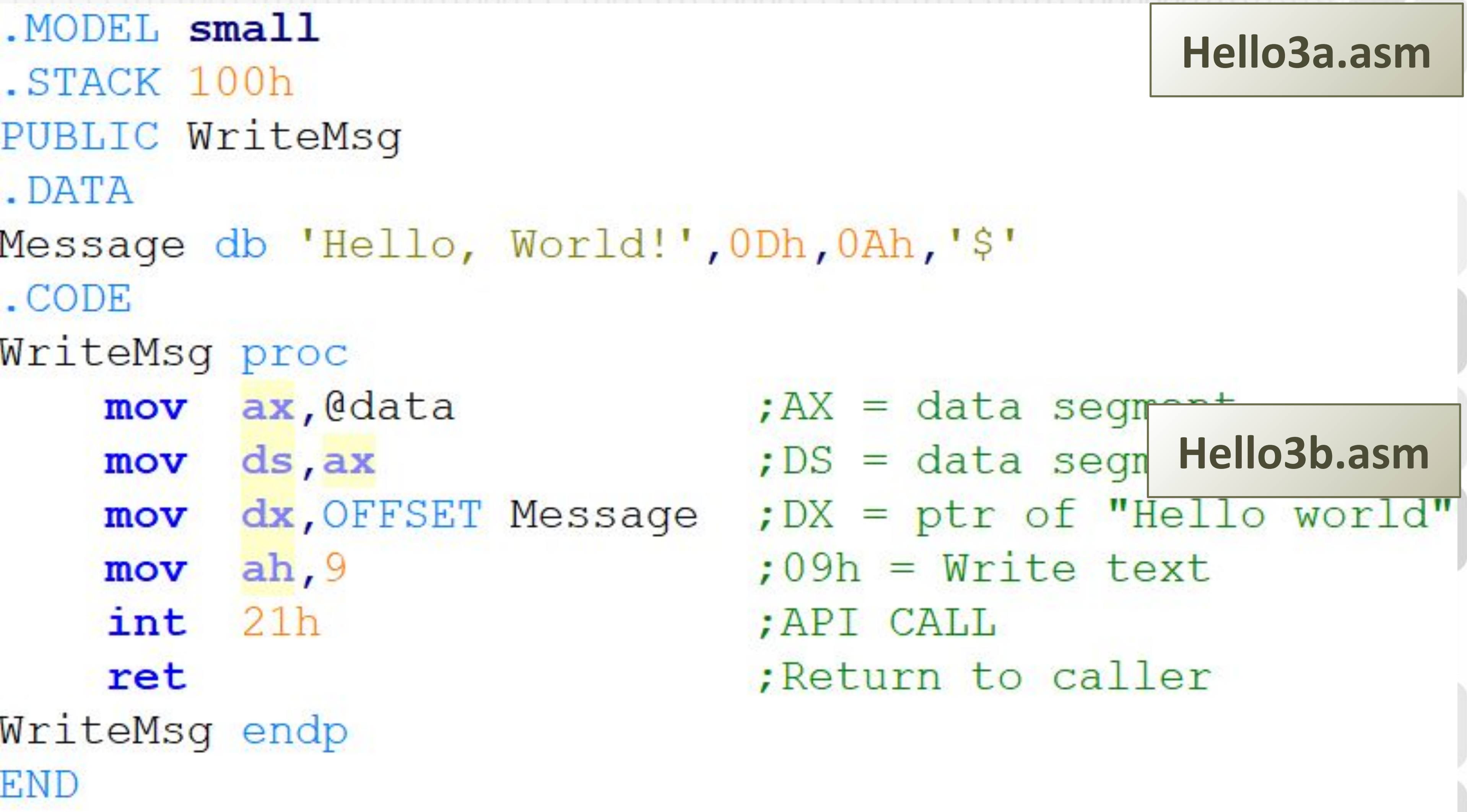
Hello2.asm

```
.MODEL small
.STACK 100h
.DATA
Message db 'Hello, World!',0Dh,0Ah,'$'
.CODE
WriteMsg proc
    mov   ax,@data
    mov   ds,ax
    mov   dx,OFFSET Message
    mov   ah,9
    int   21h
    ret
WriteMsg endp
main proc
    CALL WriteMsg
    mov   ah,4ch
    int   21h
main endp
END main
```



```
D:\TASM5\BIN>tasm hello2.asm
Turbo Assembler  Version 4.1  C

Assembling file:   hello2.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  468k


D:\TASM5\BIN>tlink hello2.obj
Turbo Link  Version 7.1.30.1. C

D:\TASM5\BIN>hello2.exe
Hello, World!
```

If methods are not in the same module…

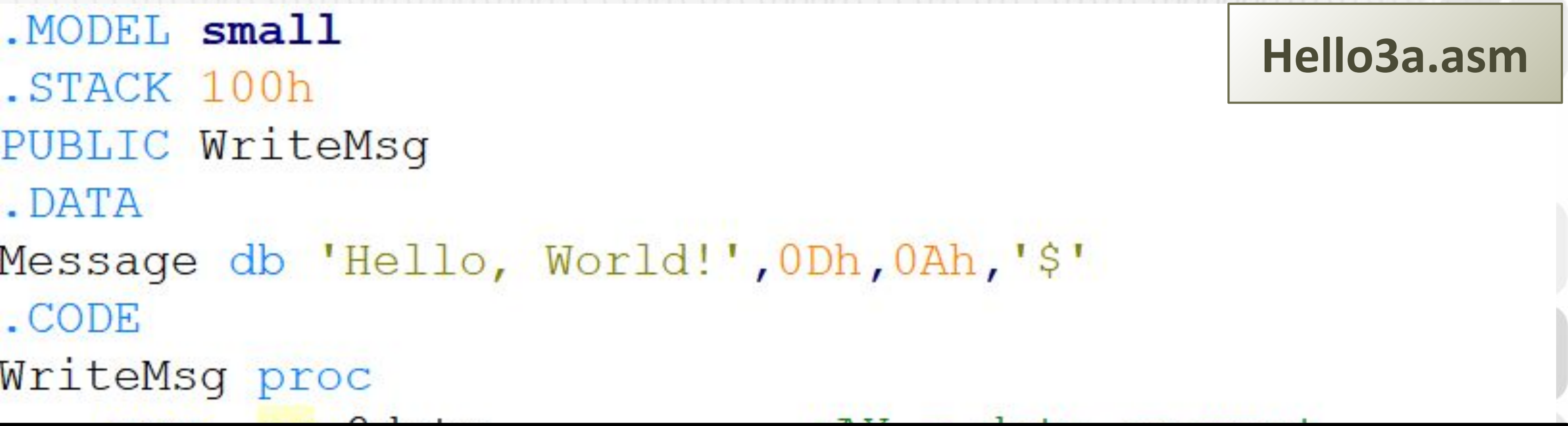


```
.MODEL small
.STACK 100h
PUBLIC WriteMsg
.DATA
Message db 'Hello, World!',0Dh,0Ah,'$'
.CODE
WriteMsg proc
    mov  ax,@data                ;AX = data segm
    mov  ds,ax                   ;DS = data segm
    mov  dx,OFFSET Message       ;DX = ptr of "Hello world"
    mov  ah,9                    ;09h = Write text
    int  21h                     ;API CALL
    ret                          ;Return to caller
WriteMsg endp
END
```

Hello3a.asm

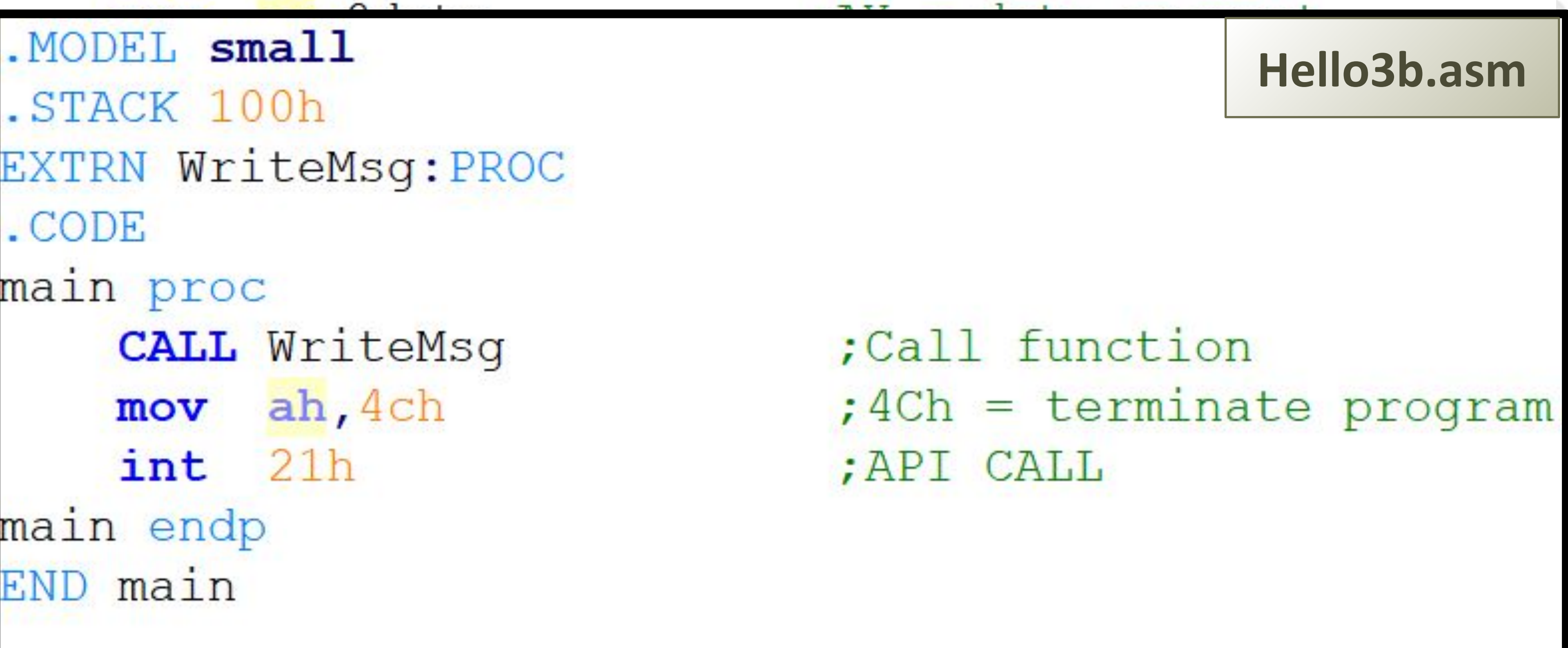
Hello3b.asm

If methods are not in the same module…



Hello3a.asm

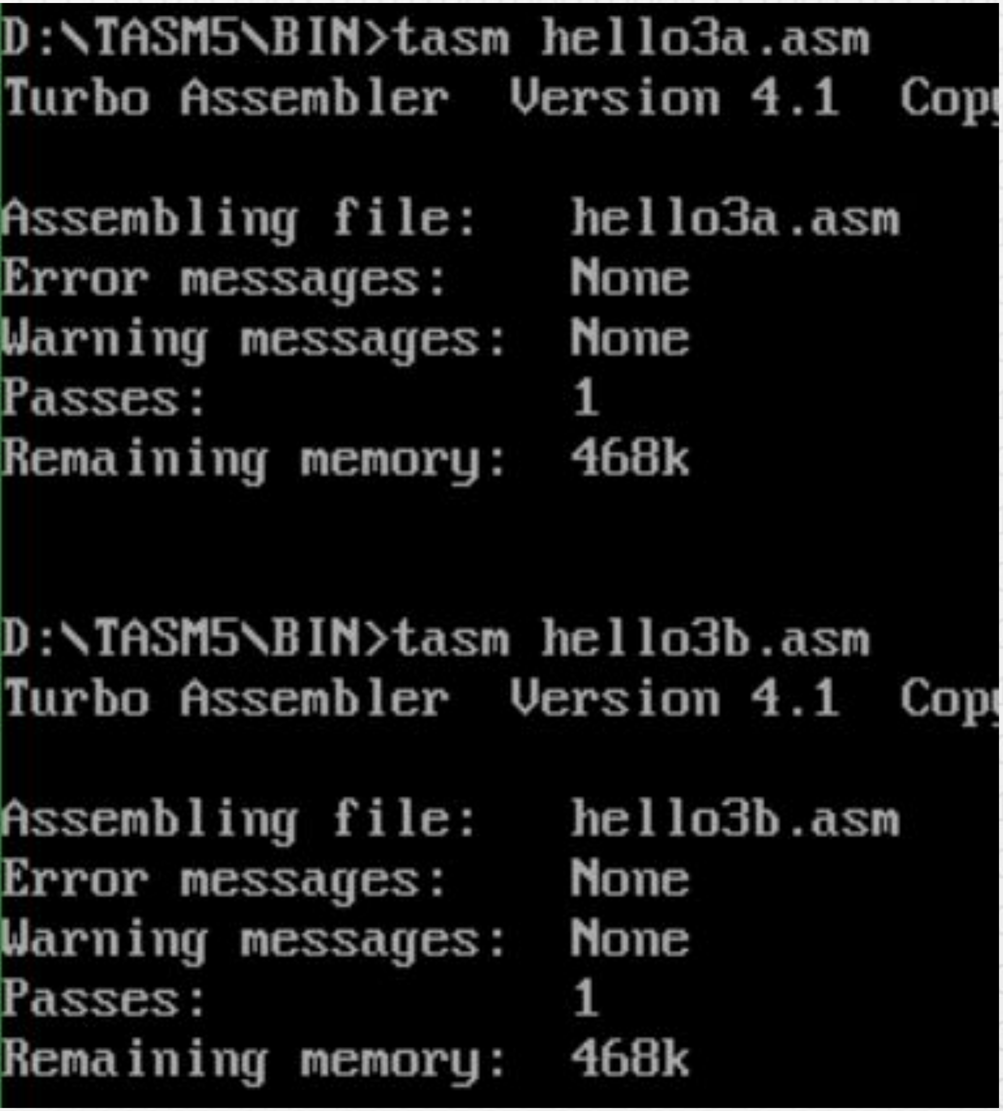
```
.MODEL small
.STACK 100h
PUBLIC WriteMsg
.DATA
Message db 'Hello, World!',0Dh,0Ah,'$'
.CODE
WriteMsg proc
```



Hello3b.asm

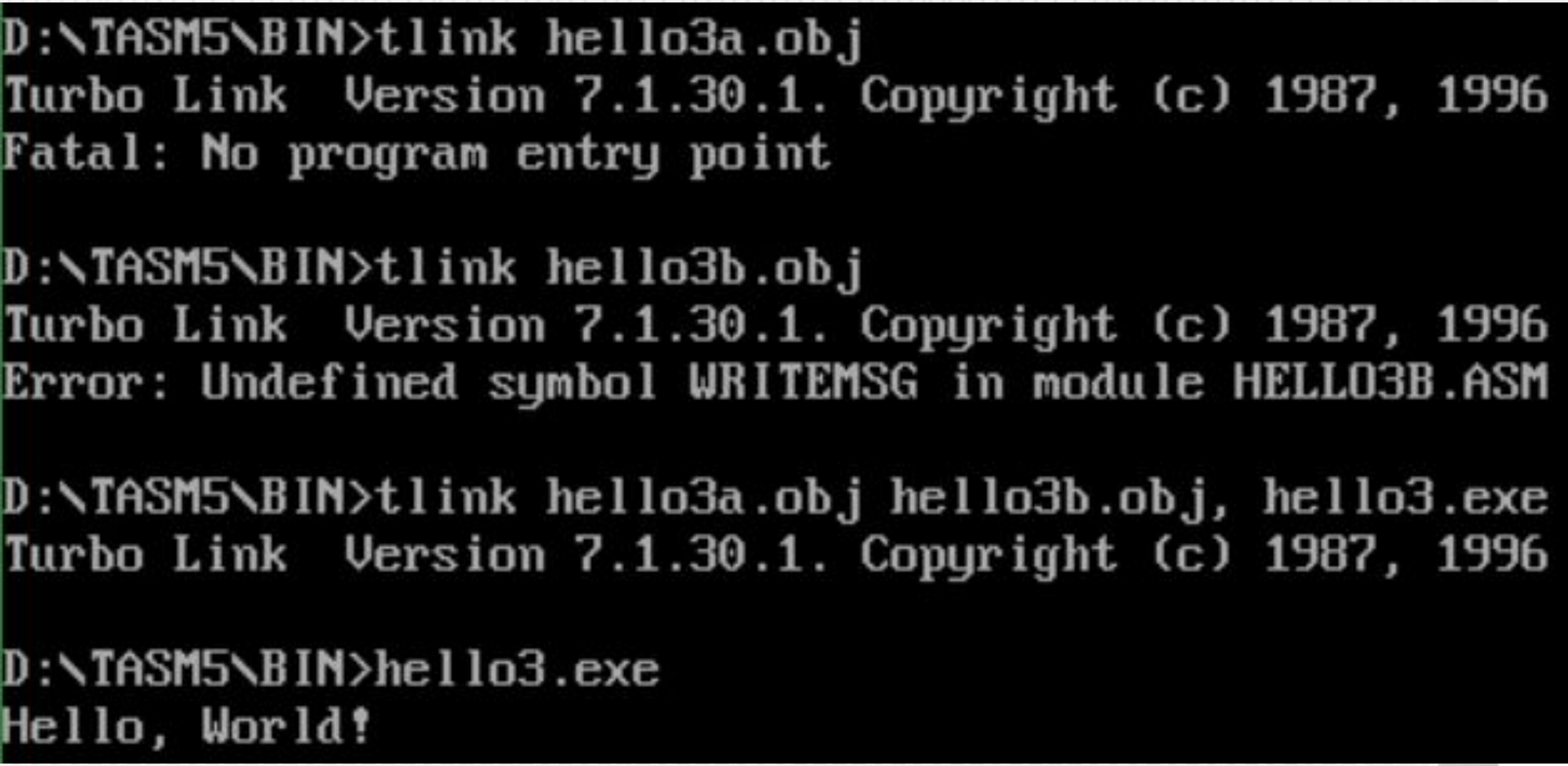
```
.MODEL small
.STACK 100h
EXTRN WriteMsg:PROC
.CODE
main proc
    CALL WriteMsg                ;Call function
    mov  ah,4ch                  ;4Ch = terminate program
    int  21h                     ;API CALL
main endp
END main
```

Code, Compile, LINK – static linking!



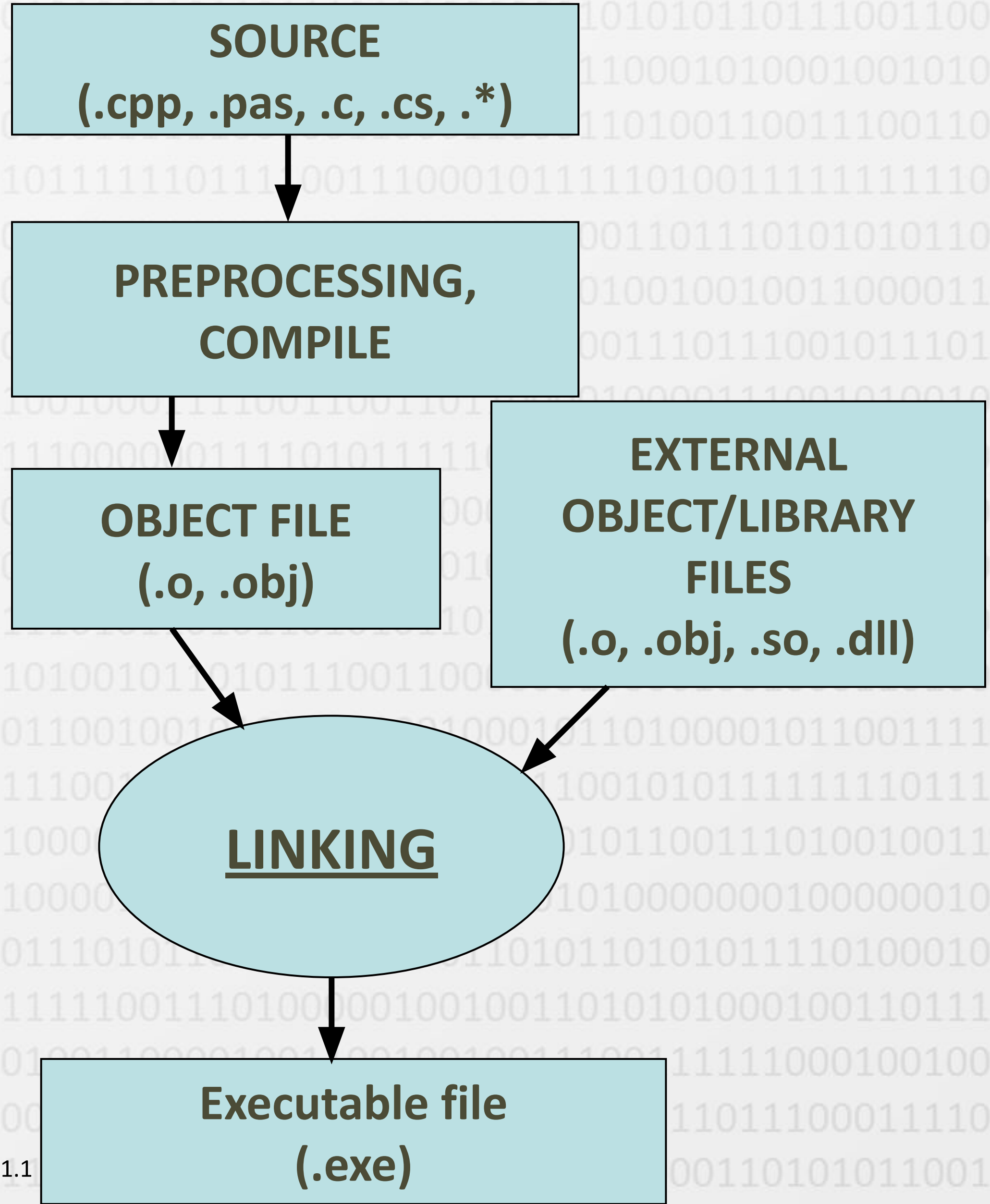
```
D:\TASM5\BIN>tasm hello3a.asm
Turbo Assembler  Version 4.1  Copy

Assembling file:   hello3a.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  468k


D:\TASM5\BIN>tasm hello3b.asm
Turbo Assembler  Version 4.1  Copy

Assembling file:   hello3b.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  468k
```

Code, Compile, LINK – static linking!



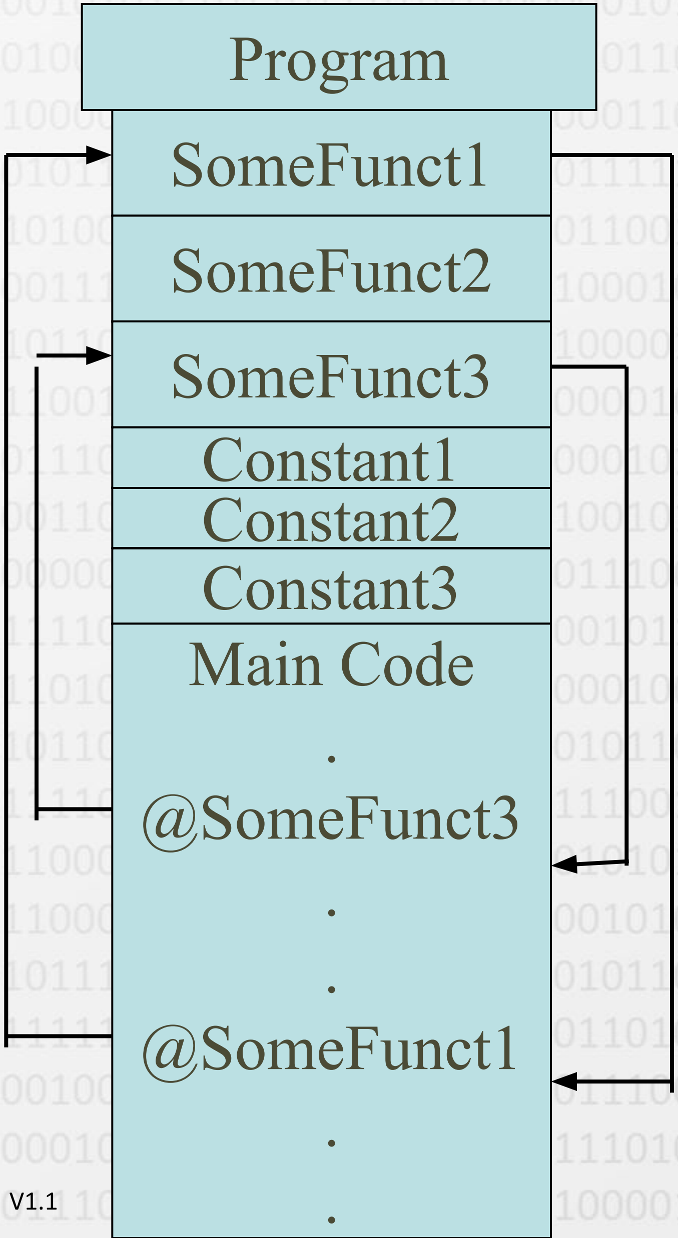
```
D:\TASM5\BIN>tlink hello3a.obj
Turbo Link  Version 7.1.30.1. Copyright (c) 1987, 1996
Fatal: No program entry point

D:\TASM5\BIN>tlink hello3b.obj
Turbo Link  Version 7.1.30.1. Copyright (c) 1987, 1996
Error: Undefined symbol WRITEMSG in module HELLO3B.ASM

D:\TASM5\BIN>tlink hello3a.obj hello3b.obj, hello3.exe
Turbo Link  Version 7.1.30.1. Copyright (c) 1987, 1996

D:\TASM5\BIN>hello3.exe
Hello, World!
```

Classic compile process



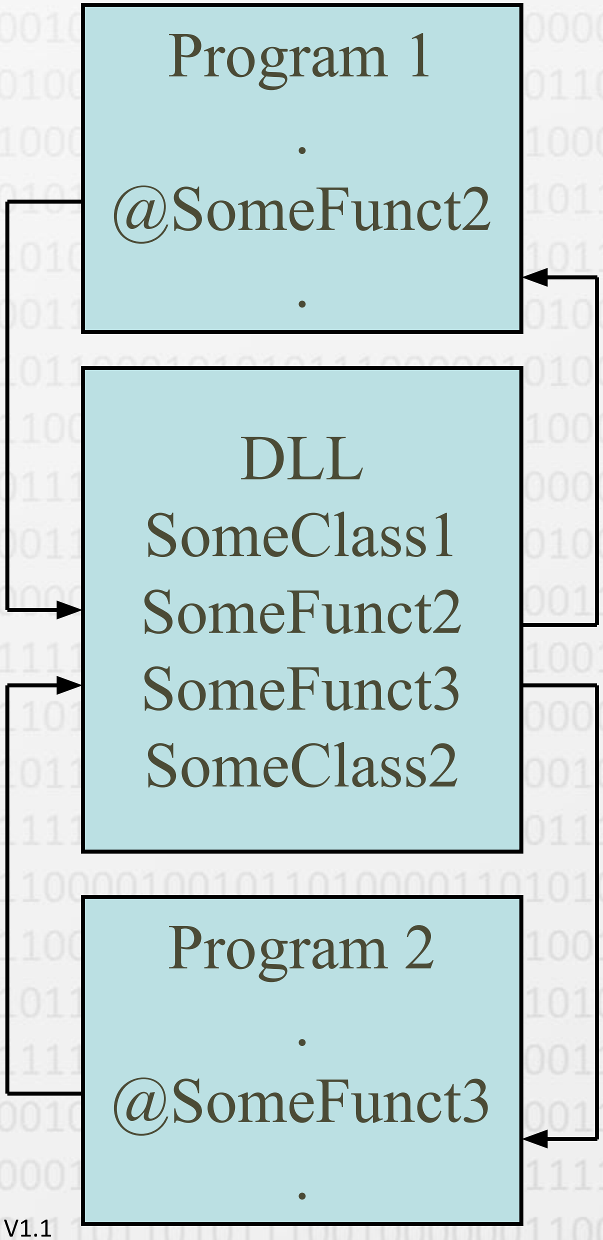
- **Object File: An intermediate binary machine code representation, generated by the compiler from the source**
- **Contains: the compiled code, relocation data, used by the linker to generate the executable**
- **The external library code is merged into the EXE file if static linking is used. It is contained in an external file if dynamic linking is used**

Static linking



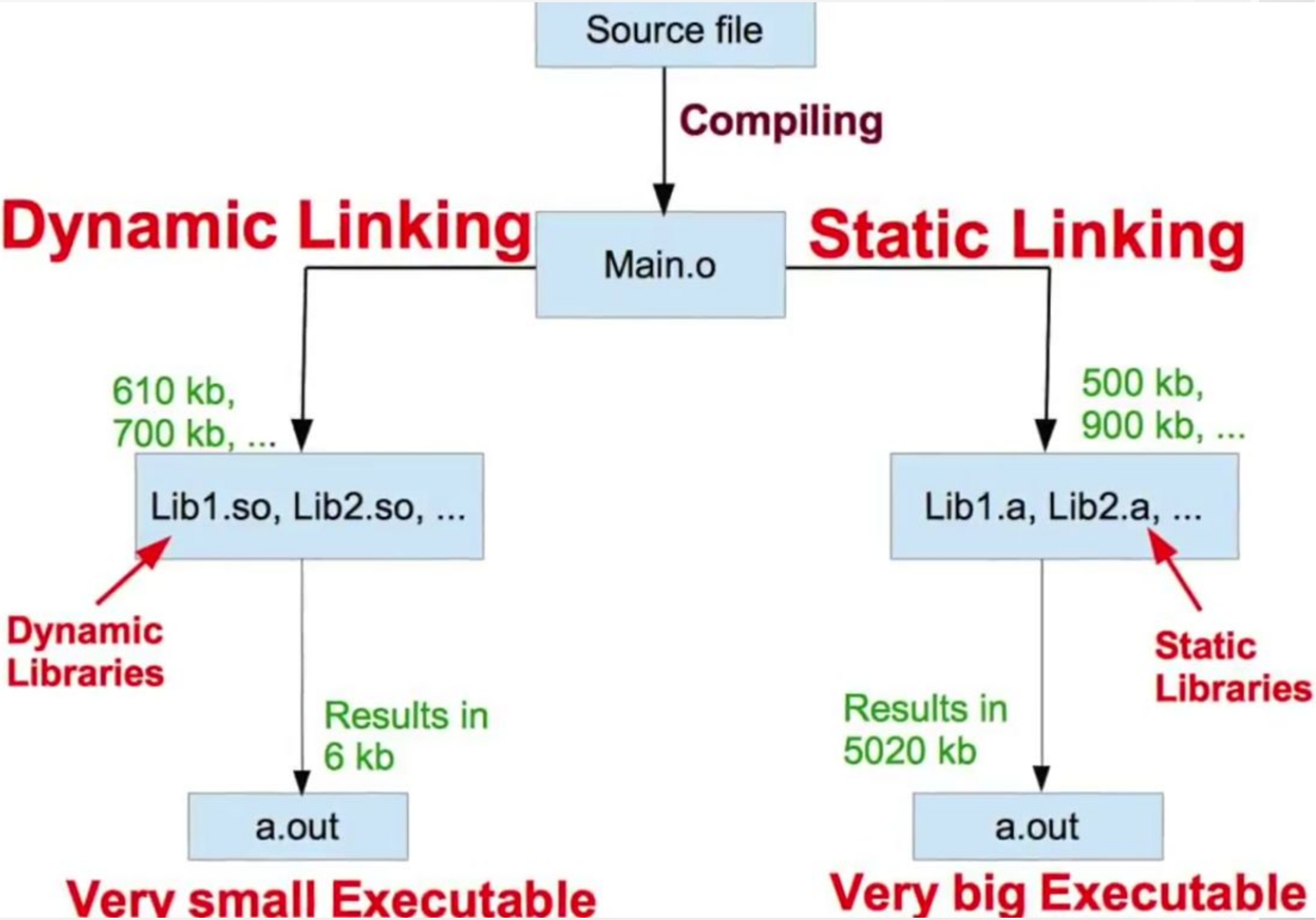
- All functions/resources are within the executable file
- The location of those are known to the compiler
- So the references to those can be pre-defined, because all functions are located in the same address space
- The same function/resource is loaded by all programs using the given feature □ waste of resources
- All classic (non-overlay) DOS programs (e.g.: Turbo Pascal)

Dynamic linking

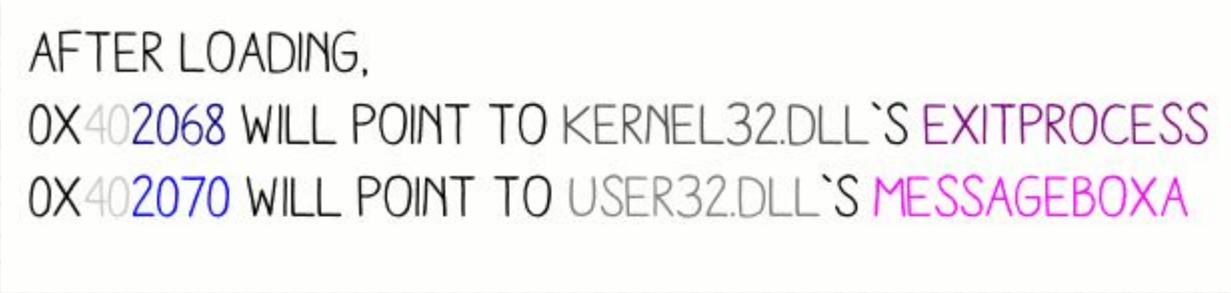


- Some methods/resources are outside the memory allocated to the process
- The exact location of those are not known to the compiler
- The reference to those methods are dynamic, determined during run-time (not known during compile-time, the DLL will be loaded by the OS on demand)
- The same method can be used by multiple processes shared (~ Shared Object)
- Most of the modern programs work this way (static linking: rare)

Static vs Dynamic Linking



RDATA, DATA, CODE/TEXT

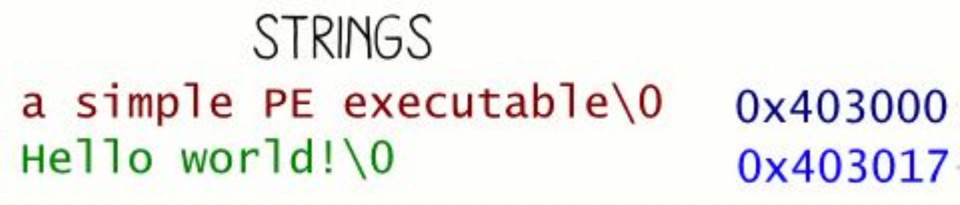


AFTER LOADING,

0X402068 WILL POINT TO KERNEL32.DLL`S EXITPROCESS

0X402070 WILL POINT TO USER32.DLL`S MESSAGEBOXA

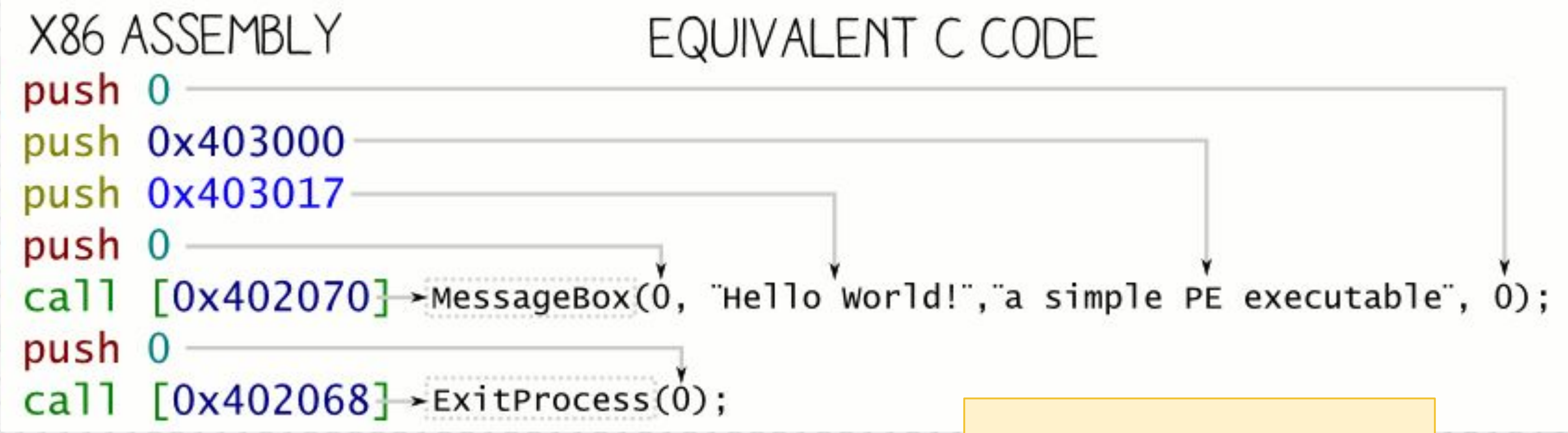
Executable contains what DLL’s which method should be used.



STRINGS

a simple PE executable\0	0x403000
Hello world!\0	0x403017

Executable contains the constants.



X86 ASSEMBLY | EQUIVALENT C CODE

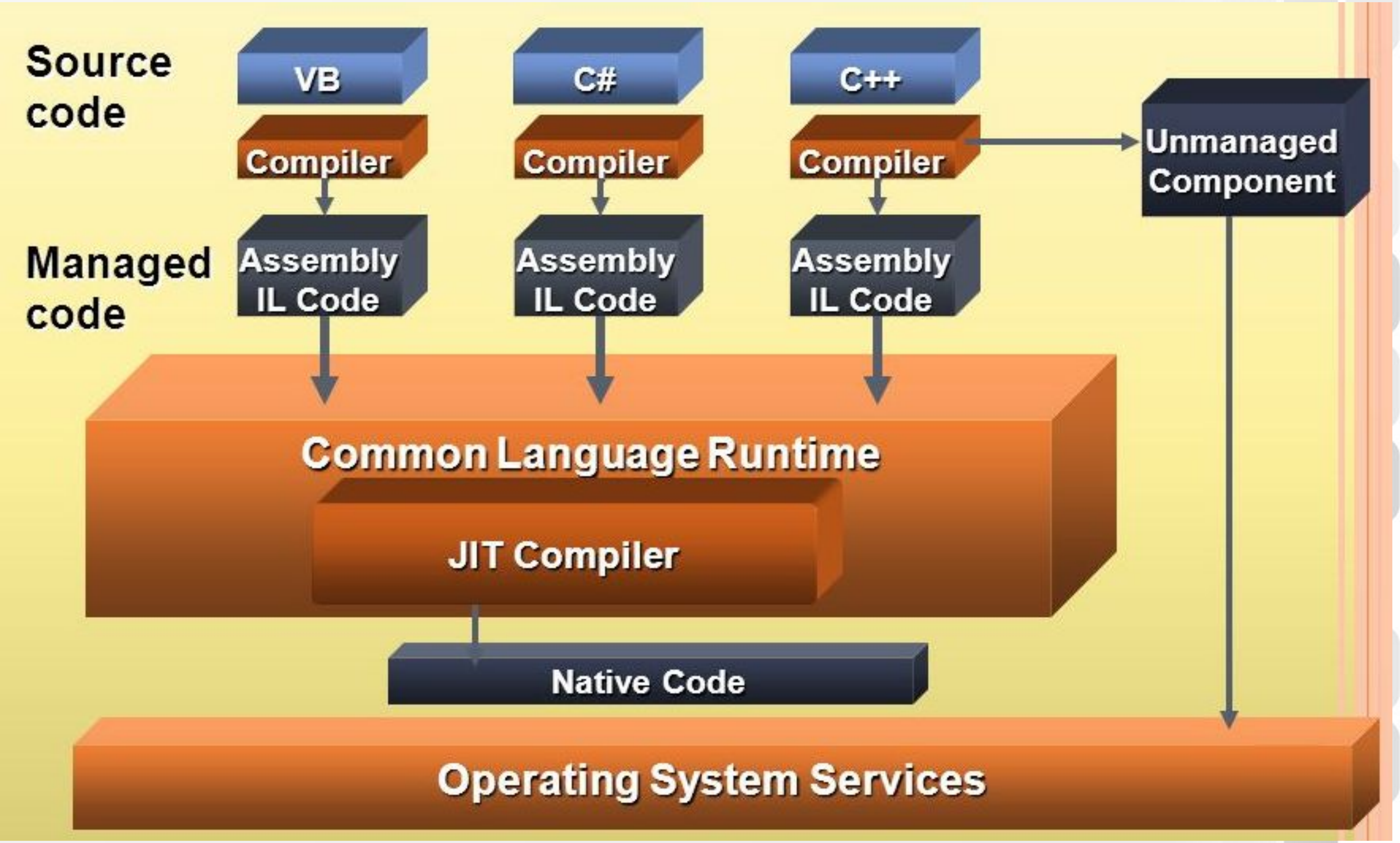
```
push 0
push 0x403000
push 0x403017
push 0
call [0x402070] →MessageBox(0, ¨Hello World!¨,¨a simple PE executable¨, 0);
push 0
call [0x402068] →ExitProcess(0);
```

When in the executable there is a method call, it is called through the DLL.

Dynamic Link Library / Shared Object

- **Almost all modern applications work this way; static linking is still possible but almost never used**
- **Same task: Windows has DLL files, Linux has SO files**
  - Separately compiled, only attached to the executable during the execution (= dynamic, true for both)
  - Only loaded to the memory once, multiple programs can use the same file (= shared, true for both, in.NET only partially (AppDomain))
- **Windows OS: distinguish between native and managed executables:**
  - *Native / Unmanaged*: It contains binary code that is processed directly by the OS/CPU; procedural code, simple types, direct HW access
    Language-independent (anything can be compiled to this kind of EXE/DLL files), but platform-independent
  - *Managed*: Contains one (or more) classes, only the .NET/JVM framework can work with it
    Language- and platform-independent (if the.NET/JVM system is available)
    With .NET, these are DLL/EXE files, in JVM these are CLASS/JAR files

Managed/Unmanaged



DLL types

Native DLL

Managed DLL

Native DLL files

- **Loaded from the current directory or from %PATH%**
  - **%PATH% = a WINDOWS, SYSTEM, SYSTEM32 folders**
- **Slow load time**
- **DLL HELL**
  - No solution for versioning
  - Different apps require different versions of the same DLL
  - The different versions might cause problems, especially when uninstalling
  - „Solution”: *DLL stomping* OR all DLL next to the EXE…
  - Linux solution: file-level package manager + versioned symlinks
  - .NET solution: GAC = Global Assembly Cache
- **Windows API**
  - A set of unmanaged DLL files, containing system methods
  - Low-level operations, HW access
  - All public functionality of the OS is available
  - The more important features have.NET wrappers, but the lowest level still uses WINAPI calls (e.g. System.Diagnostics.Stopwatch = QueryPerformanceCounter)

Calling a native DLL

- **Platform-dependent**
  - 32 / 64 bits (CPU bit length)
  - OS dependent
- **Language independent?**
  - Theoretically yes, but…
    - Passing parameters
    - Returning a result (complex types?)
    - Who frees what?
  - Possibilities: cdecl, stdcall, fastcall, …
  - https://en.wikipedia.org/wiki/X86_calling_conventions
- **Accessible methods?**
  - Cannot be queried from code, must be verified <u>during development</u>
  - dumpbin /exports → can show what callable methods are in a dll file
- **The dependencies of the used DLL**
  - Cannot be queried from code, must be verified during development
  - Dependency Walker → can show if X dll uses an Y dll which uses Z dll… → dll dependencies

Write something in C and use it in C# as DLL → theoretically possible but ...

Make sure that the **caller** and the **called** method uses the same calling convention.

Calling native DLL in C#

- **Attached during execution time**
- **No verification (for existence of the DLL/Entry point) is performed by the compiler**

- **Platform Invoke (P/Invoke: call native DLL method from managed environment) DllImport attribute**
  - `using System.Runtime.InteropServices;`

  -
    ```
    [DllImport("winmm.dll", SetLastError = true)]
    static extern bool PlaySound(string pszSound,
                                 UIntPtr hmod, uint fdwSound);
    ```

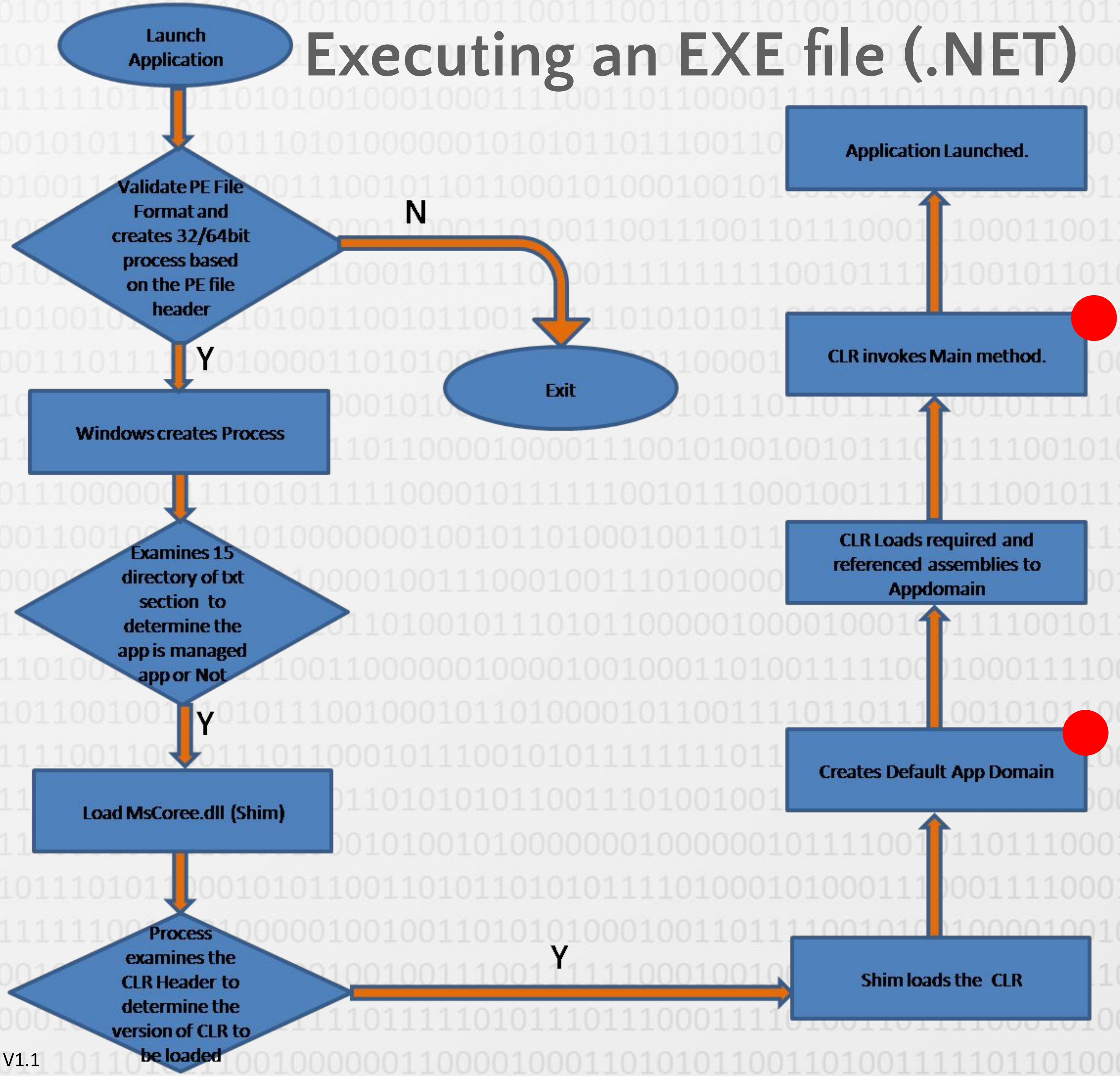
  -
    ```
    string fname = @"c:\Windows\Media\tada.wav";
    PlaySound(fname, UIntPtr.Zero, 1);
    ```

- **WINAPI signatures, imports: www.pinvoke.net**

Managed DLL files (executables)

- EVERY method call we ever had in C# was a DLL call
  - A project's „References" part will store which DLLs are accessible from the project
  - The compiler checks the existence of the DLL and the class/method
  - Fast load time, the same speed as with our own code
- **Calling managed DLL files**
  - Project/Add reference
  - OR: install via Nuget
  - After this, the namespaces and the classes/methods in the DLL are accessible
- **EXE or DLL?**
  - In .NET, no big difference, both contain managed classes, and same IL code
  - The classic (PE) part of the EXE only loads up the CLR interpreter
  - The EXE must contain a single `static void/int Main()`
    - while DLLs not
  - Project types: Console App / WPF or Windows Forms / Class Library

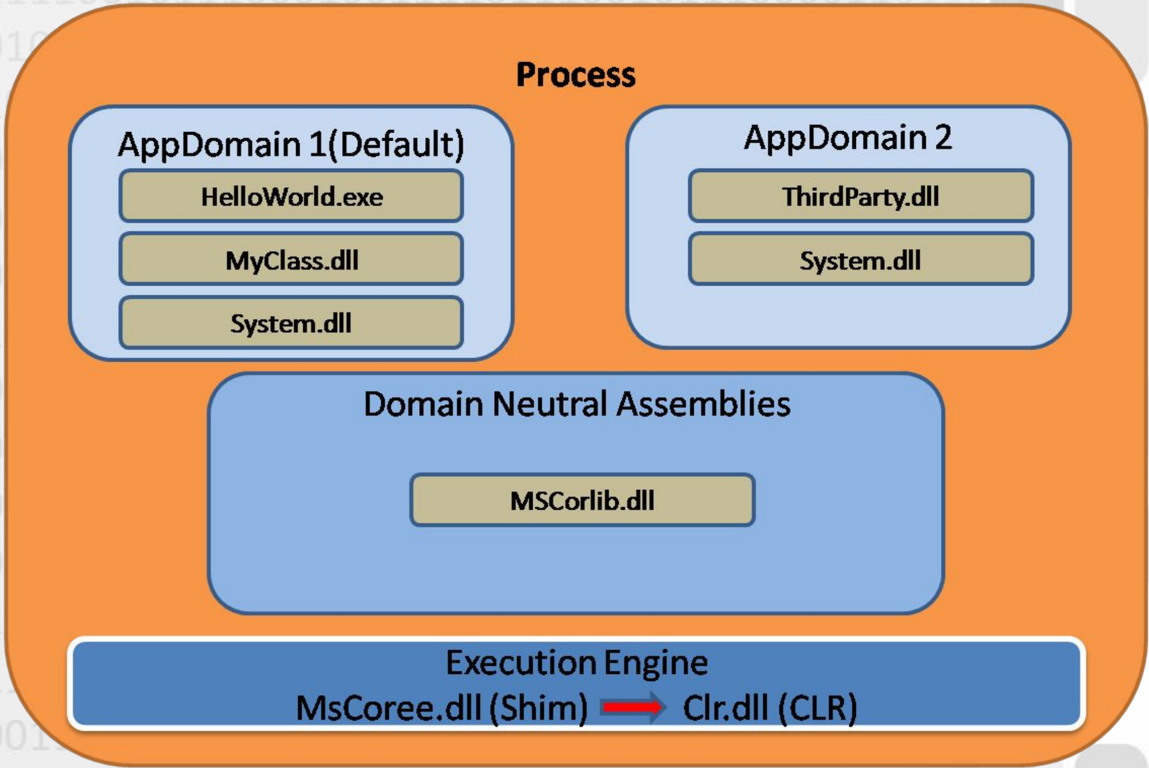
Executing an EXE file (.NET)



Launch Application
Validate PE File Format and creates 32/64bit process based on the PE file header
N
Exit
Y
Windows creates Process
Examines 15 directory of txt section to determine the app is managed app or Not
Y
Load MsCoree.dll (Shim)
Process examines the CLR Header to determine the version of CLR to be loaded
Y
Shim loads the CLR
Creates Default App Domain
CLR Loads required and referenced assemblies to Appdomain
CLR invokes Main method.
Application Launched.

Managed Assembly – Sandboxing

- **AppDomain**
  - A security layer between the .NET assembly and the OS process
  - Regulates the execution of the code and the access of resources
  - For example when using a web application, the .NET applications of a single website (Virtual Directory) can access the same resources even if they are different assemblies. Different websites can access different resources, even if the same .EXE file is launched
  - There can be multiple AppDomain inside a single Win32 process
  - Separation of resources □ sandboxing



Process

AppDomain 1(Default)

HelloWorld.exe

MyClass.dll

System.dll

AppDomain 2

ThirdParty.dll

System.dll

Domain Neutral Assemblies

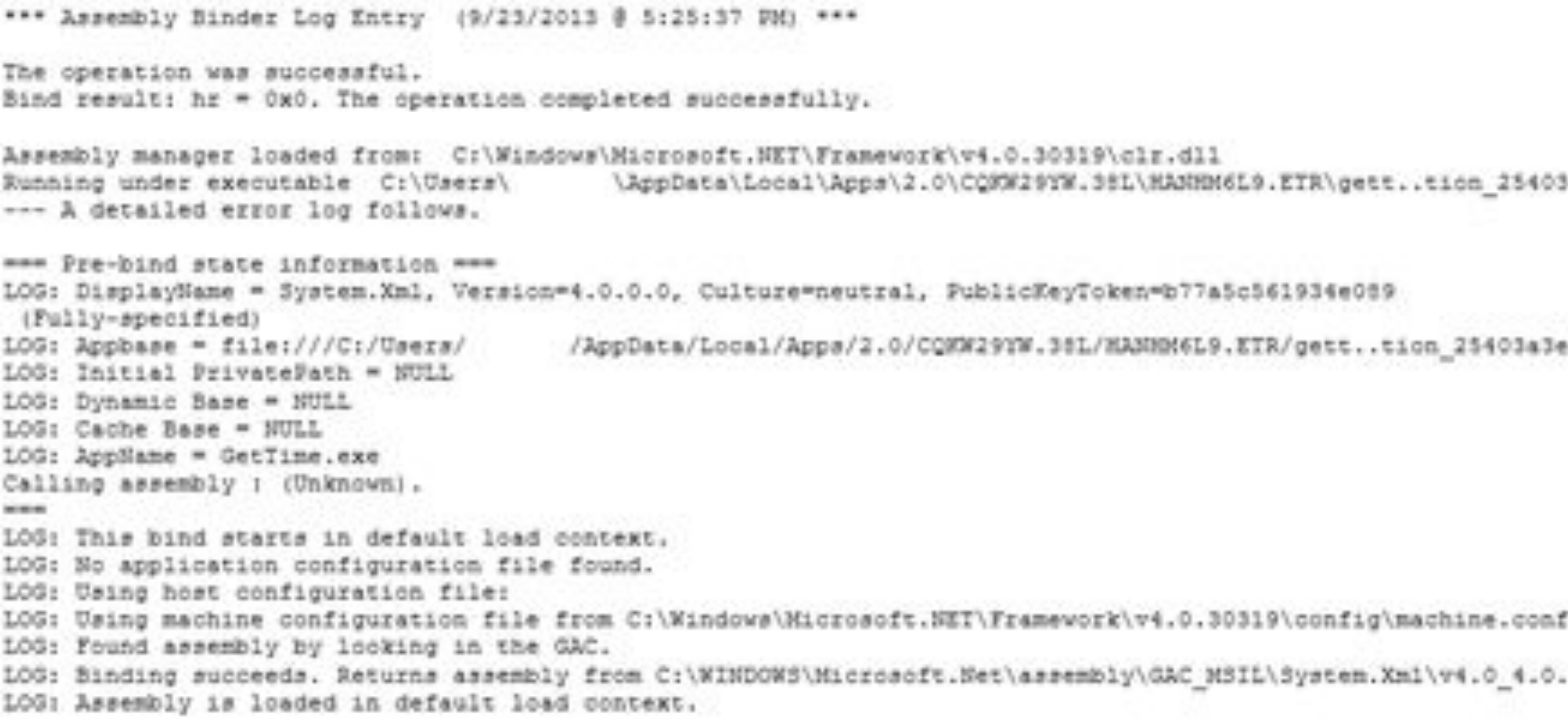
MSCorlib.dll

Execution Engine

MsCoree.dll (Shim) → Clr.dll (CLR)

Loading managed DLL files

- **Fusion**
  - A .NET module that performs the loading of managed DLL files
  - „Assembly binding”: the series of steps that are executed when an executable’s external references are searched and loaded
  - Enable logs: fuslogvw.exe / Registry entries



```
*** Assembly Binder Log Entry  (9/23/2013 @ 5:25:37 PM) ***

The operation was successful.
Bind result: hr = 0x0. The operation completed successfully.

Assembly manager loaded from:  C:\Windows\Microsoft.NET\Framework\v4.0.30319\clr.dll
Running under executable  C:\Users\        \AppData\Local\Apps\2.0\CQWW29YW.38L\MANHM6L9.ETR\gett..tion_25403
--- A detailed error log follows.

=== Pre-bind state information ===
LOG: DisplayName = System.Xml, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
 (Fully-specified)
LOG: Appbase = file:///C:/Users/        /AppData/Local/Apps/2.0/CQWW29YW.38L/MANHM6L9.ETR/gett..tion_25403a3e
LOG: Initial PrivatePath = NULL
LOG: Dynamic Base = NULL
LOG: Cache Base = NULL
LOG: AppName = GetTime.exe
Calling assembly : (Unknown).
===
LOG: This bind starts in default load context.
LOG: No application configuration file found.
LOG: Using host configuration file:
LOG: Using machine configuration file from C:\Windows\Microsoft.NET\Framework\v4.0.30319\config\machine.conf
LOG: Found assembly by looking in the GAC.
LOG: Binding succeeds. Returns assembly from C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Xml\v4.0_4.0.
LOG: Assembly is loaded in default load context.
```

Tools

- **gacutil.exe**
  - Register / unregister DLL files from the GAC; this includes official .NET DLLs
  - Possibility to handle versions and dependencies
  - In the docs it can be checked that a class/namespace is found in which DLL
- **NuGet**
  - Central .NET package manager, typically for managed DLL files
  - Tools/NuGet Package Manager/Manage NuGet Packages for Solution
  - Can be used from a Powershell commandline (Install-Package)
  - Almost all C# library/tool is downloadable
  - Handles dependencies/updates
  - Consolidate: handle different versions in one solution
- **Dotpeek (ILDasm, Reflector …)**
  - They allow the inspection of IL codes inside .NET DLL/EXE files
  - Can show information accessed via Reflection (*later*)
  - Reverse engineer into C# code (usually in a readable format, except if a Code Obfuscator is used)

Managed Assembly contents

- **Assembly ~ Executable unit ~ managed .NET EXE/DLL file (Absolutely no relations with the assembly language!)**
- **Assembly Manifest/Metadata**
  - Name
  - Version
  - Culture/Localization info
  - Internal file/resource list
  - Type metadata
  - List of references
- **Type metadata**
  - All information about the contained classes/types
  - Can be processed using reflection
- **IL/CLR code (decompile: with DotPeek/ILDasm…)**
- **Resources**

Advanced Development Techniques

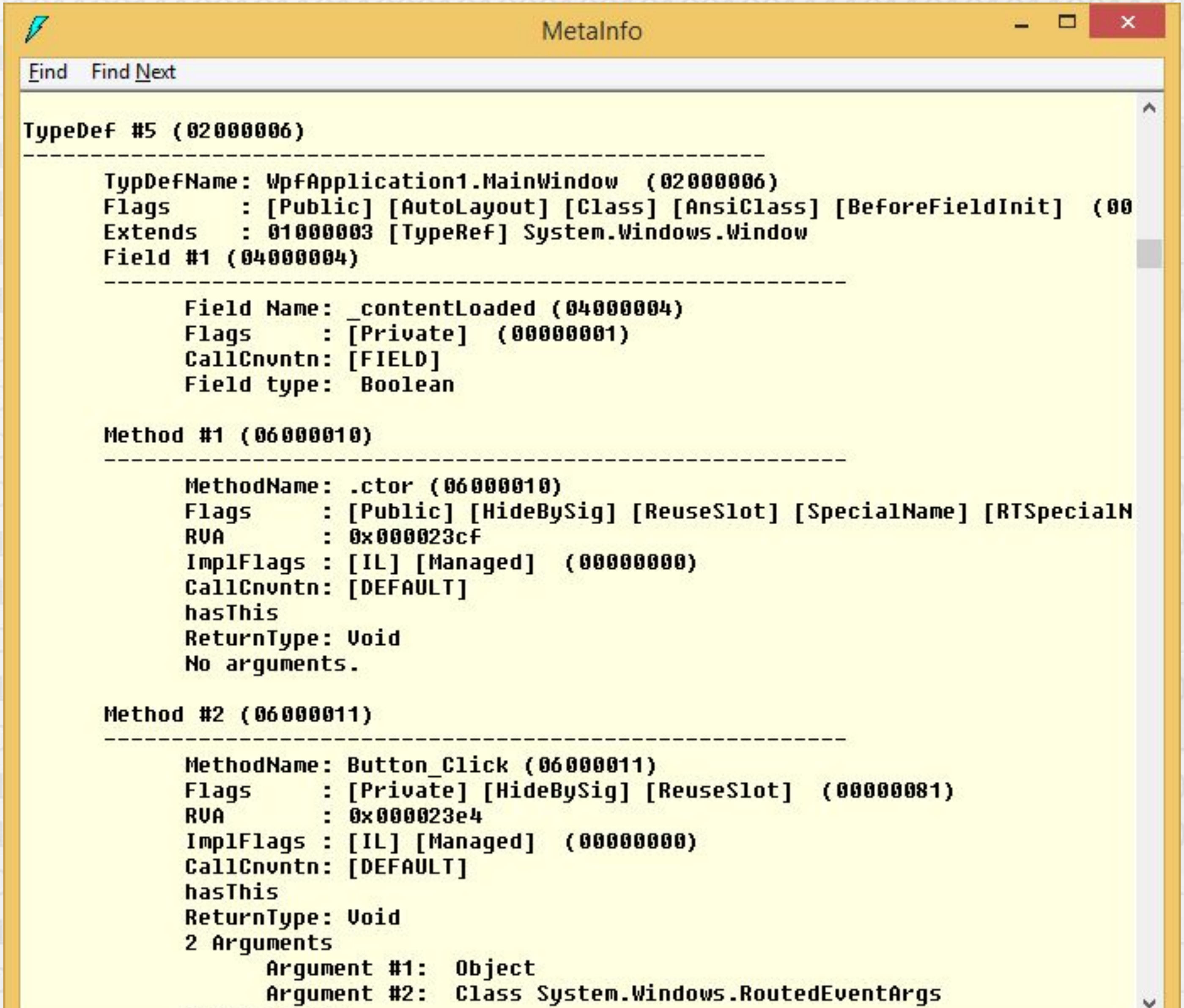
DLL
Reflection

Reflection

- **A program/class can analyze and change its own structure and behavior in run-time**
  - High-level language (Java, PHP, … C#)
  - Different approaches / support in different languages
- **C#: System.Reflection namespace**
- **In .NET, we usually use it to analyze types during run-time**
  - It would be possible to create type/methods/blocks: System.Reflection.Emit
- **This is possible because of the meta-data (descriptor information) located next to the types in the .NET assemblies**
  - Assembly: .exe, .dll (sort-of)
  - Assembly metadata: references, types, namespaces ...
  - Type metadata: interfaces, base classes, members
  - Member metadata: visibility, parameters, property methods...
- **Used by multiple .NET technologies**
  - Tests, Intellisense, Serialization, WCF, EF

Metadata

- **Visual Studio Command Prompt / Ildasm.exe, Ctrl+M**



```
TypeDef #5 (02000006)
-------------------------------------------------------
	TypDefName: WpfApplication1.MainWindow  (02000006)
	Flags     : [Public] [AutoLayout] [Class] [AnsiClass] [BeforeFieldInit]  (00
	Extends   : 01000003 [TypeRef] System.Windows.Window
	Field #1 (04000004)
	-------------------------------------------------------
		Field Name: _contentLoaded (04000004)
		Flags     : [Private]  (00000001)
		CallCnvntn: [FIELD]
		Field type:  Boolean

	Method #1 (06000010)
	-------------------------------------------------------
		MethodName: .ctor (06000010)
		Flags     : [Public] [HideBySig] [ReuseSlot] [SpecialName] [RTSpecialN
		RVA       : 0x000023cf
		ImplFlags : [IL] [Managed]  (00000000)
		CallCnvntn: [DEFAULT]
		hasThis
		ReturnType: Void
		No arguments.

	Method #2 (06000011)
	-------------------------------------------------------
		MethodName: Button_Click (06000011)
		Flags     : [Private] [HideBySig] [ReuseSlot]  (00000081)
		RVA       : 0x000023e4
		ImplFlags : [IL] [Managed]  (00000000)
		CallCnvntn: [DEFAULT]
		hasThis
		ReturnType: Void
		2 Arguments
			Argument #1:  Object
			Argument #2:  Class System.Windows.RoutedEventArgs
```

Assembly

- **Assembly a = Assembly.GetExecutingAssembly();**
- **Assembly a = Assembly.LoadFrom(„Path.To.Assembly”);**
- **Assembly a = Assembly.Load(bytes);**
- **Assembly a = type.Assembly;**

- **a.GetTypes()** – types in the assembly
- **a.EntryPoint** – entry point (Main() in exe files)

Type

- **Type t = assembly.GetType(„Type.Name.In.Assembly”);**
- **Type t = typeof(int);**
- **Type t = typeof(T);**
- **Type t = obj.GetType();**
- **Type t = Type.GetType(„Type.Name.In.Any.Assembly”);**
  - Full „assembly-qualified name” might be required

- **t.FullName, t.AssemblyQualifiedName** – name of type
- **t.BaseType, t.IsSubclassOf(anotherType), t.IsAssignableFrom(anotherType)** – examine base/inheritance

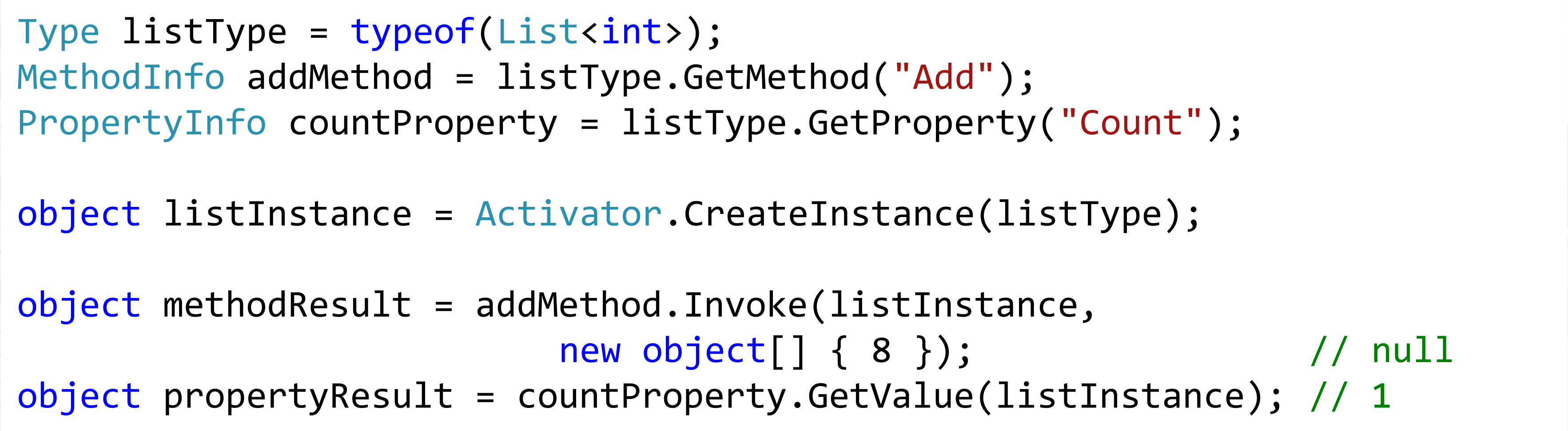
MethodInfo, PropertyInfo, FieldInfo

- PropertyInfo pi = t.GetProperty("PropName");
- PropertyInfo[] pis = t.GetProperties();
- FieldInfo fi = t.GetField("FieldName");
- FieldInfo[] fis = t.GetFields();
- MethodInfo mi = t.GetMethod("MethodName");
- MethodInfo mis = t.GetMethods();

- We can use the BindingFlags parameter to filter the results
- PropertyInfo pi = t.GetProperty("PropName", BindingFlags.Static | BindingFlags.NonPublic)
  - We can access non-public members
  - SHOULD NOT BE USED to bypass visibility
  - VERY SLOW!!!

Example

- The accessed types/members can be used in execution time
- List<int> something = new List<int>();
  something.Add(8);
  int cnt = something.Count;



```
Type listType = typeof(List<int>);
MethodInfo addMethod = listType.GetMethod("Add");
PropertyInfo countProperty = listType.GetProperty("Count");

object listInstance = Activator.CreateInstance(listType);

object methodResult = addMethod.Invoke(listInstance,
                                new object[] { 8 });                // null
object propertyResult = countProperty.GetValue(listInstance); // 1
```

- Slower than the normal code □ Only if not doable in any other way (e.g. we want to work with SOMETHING that has an Add method and a Count property)
- Flexible code, less overhead, no compiler/intellisense support, not in our schedule: dynamic (DLR)

Example

- **The accessed types/members can be used in execution time**
- **List<int> something = new List<int>();**
  something.Add(8);
  int cnt = something.Count;

```
Type listType = typeof( ------ );
MethodInfo addMethod = listType.GetMethod("Add");
PropertyInfo countProperty = listType.GetProperty("Count");

object listInstance = Activator.CreateInstance(listType);

object methodResult = addMethod.Invoke(listInstance,
                             new object[] { 8 });                // null
object propertyResult = countProperty.GetValue(listInstance); // 1
```

- **I don’t know anything about the given type!**
  - ○ **Object is often used, because “I work with something”**
  - ○ **I only know that it has X method or Y property.**

Attributes

- **We can add custom metadata**
  - Assembly, type or member metadata
- **System.Attribute descendants**
  - Several pre-existing attributes exist
  - We can create our own attributes
- **Special usage**
  - ABOVE the namespace, class, method, property, field
  - We use the text [XXX], if the class is named XXXAttribute
  - Later we can use the attributes with reflection

```
[Obsolete("Do not use this method, use New() instead.")]
static void OldMethod()
{ }

static void NewMethod() { }

static void Main(string[] args)
{
    OldMethod();    // Create Warning in the intellisense window
}
```

Typical use cases

- **The attribute does not affect the normal use of the decorated code**
  - All methods, properties, etc. can be accessed/called in a normal way
- **We need “someone else” that will use reflection to check for the existence of the attribute, and perform operations**
- **Typical use case: automatic operations/checks**
  - Help other programmer:
    - Obsolete, DisplayName, Description
  - Affect the behavior of the Visual Studio debugger:
    - DebuggerDisplay, DebuggerStepThrough
  - Visual Studio, automatic code generation:
    - WebMethod, ServiceContract, OperationContract, FaultContract, DataContract, DataMeber
  - Usage of code:
    - Serializable, Flags, ThreadStatic, DllImport
  - Support various automatic features:
    - TestClass, TestMethod, Key, ForeignKey, Column
  - Other (self-made) metadata:
    - with self-made attribute classes

Attributes

- **CallerMemberName**
  - If the parameter is not specified, then the caller name will be substituted

```
protected void OnPropertyChanged([CallerMemberName] string propertyName = null)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

```
class Settings : Bindable
{
    private string setting1;
    public string Setting1
    {
        get { return setting1; }
        set { setting1 = value; OnPropertyChanged(); }
    }
}
```

Attributes

- **Serialization (binary, xml, json…)**

```
[Serializable]
class Settings
{
    public string Setting1 { get; set; }
    public int Setting2 { get; set; }
    [NonSerialized]
    private int temp;
}

class Program
{
    static void Main(string[] args)
    {
        Settings settings = new Settings();
        //...
        BinaryFormatter formatter = new BinaryFormatter();
        using (FileStream stream =
                    new FileStream("settings.dat", FileMode.Create))
        {
            formatter.Serialize(stream, settings);
        }
    }
}
```

Self-made attribute

- **Create the attribute class, specifying where it can be applied to**

```
[AttributeUsage(AttributeTargets.Property)]
class HelpAttribute : Attribute
{
    public string HelpURL { get; private set; }

    public HelpAttribute( string helpURL)
    {
        this.HelpURL = helpURL;
    }
}
```

!

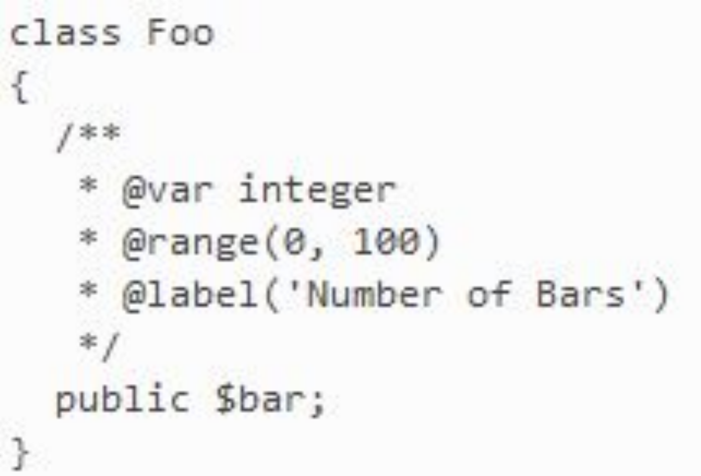
```
[Help("http://path.to.my.help.for.setting1.html")]
public string Setting1 { get; set; }
```

- **Access the attribute with reflection**
  - Memberinfo is required, all the other attributes are used this way too (by VS)

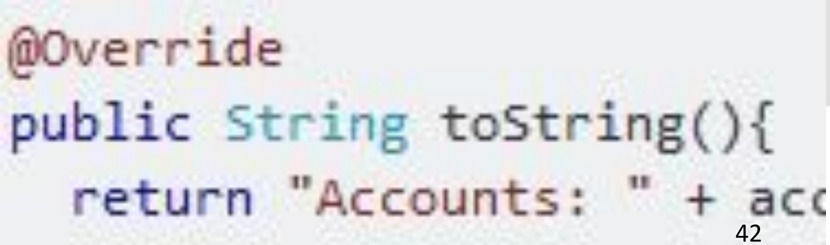
```
//PropertyInfo propertyInfo = typeof(Settings).GetProperty("Setting1");
HelpAttribute helpAttribute =
     propertyInfo.GetCustomAttribute<HelpAttribute>();

Console.WriteLine(helpAttribute.HelpURL);
```

Annotations

- **A similar language construct in other languages (Java/PHP)**
- **PHP**
  - In the comment section
  - Typically used by the IDE/external tools
  - Not really used in run-time
- **Java**
  - Interpreted by the compiler
  - Stays in the compiled classes too
  - Useable during run-time: https://en.wikipedia.org/wiki/Java_annotation

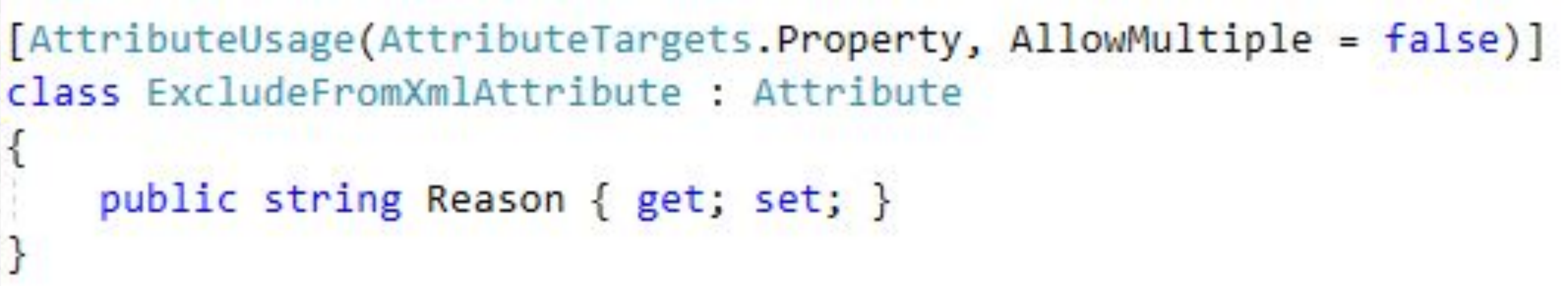


```
class Foo
{
  /**
   * @var integer
   * @range(0, 100)
   * @label('Number of Bars')
   */
  public $bar;
}
```

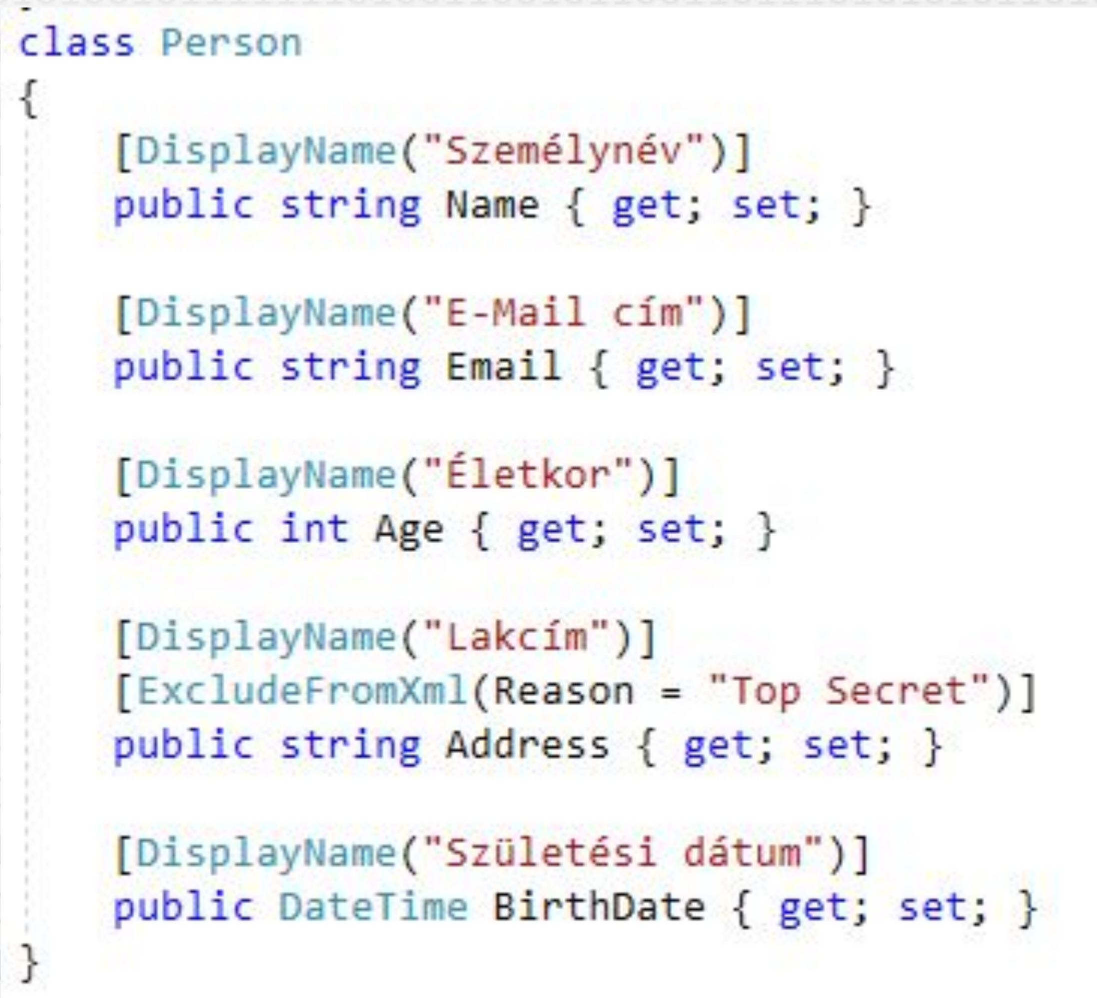


```
@Override
public String toString(){
  return "Accounts: " + acc
```

Example – XmlSerializer

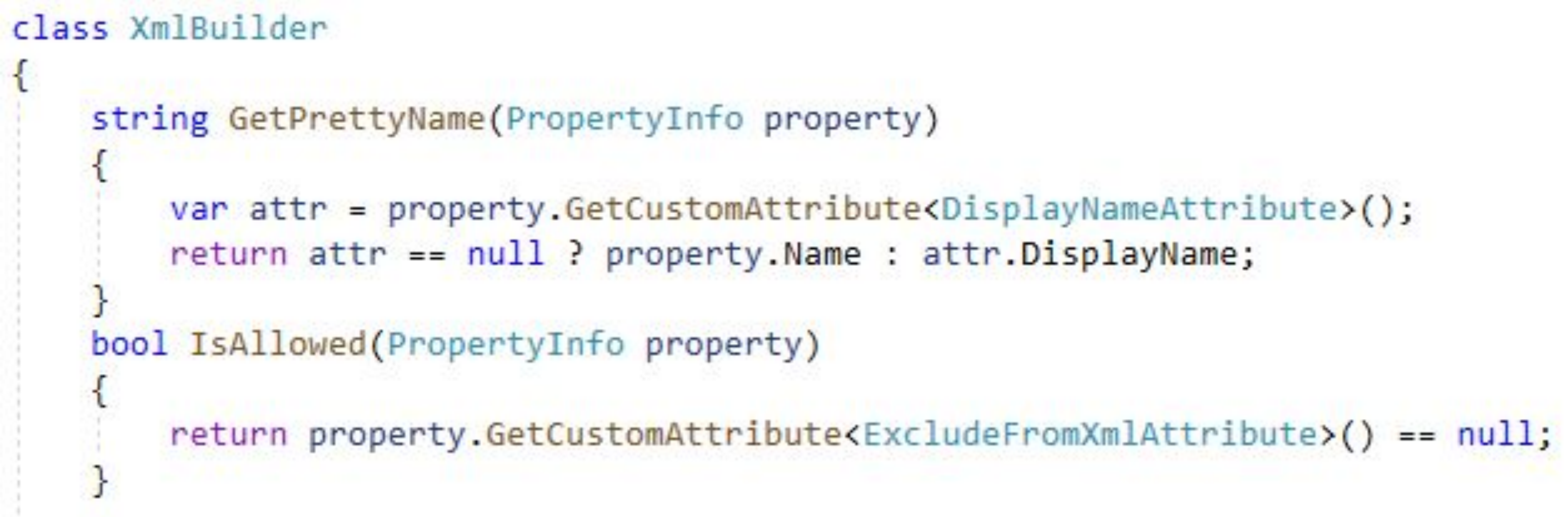


```
[AttributeUsage(AttributeTargets.Property, AllowMultiple = false)]
class ExcludeFromXmlAttribute : Attribute
{
    public string Reason { get; set; }
}
```



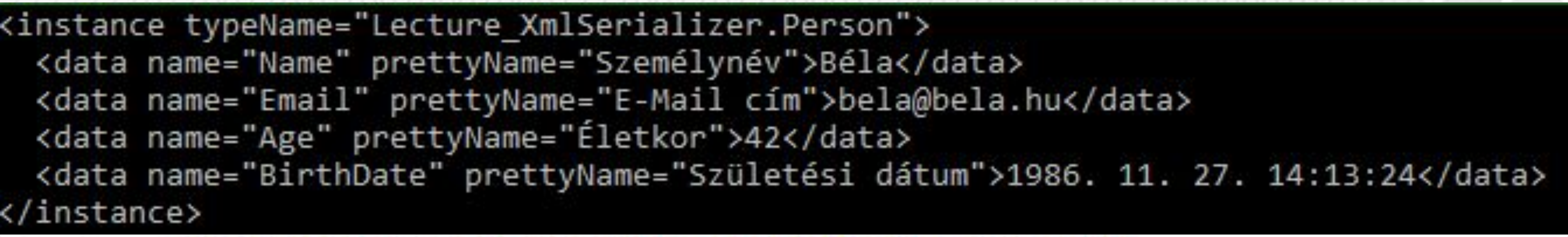
```
class Person
{
    [DisplayName("Személynév")]
    public string Name { get; set; }

    [DisplayName("E-Mail cím")]
    public string Email { get; set; }

    [DisplayName("Életkor")]
    public int Age { get; set; }

    [DisplayName("Lakcím")]
    [ExcludeFromXml(Reason = "Top Secret")]
    public string Address { get; set; }

    [DisplayName("Születési dátum")]
    public DateTime BirthDate { get; set; }
}
```

Example – XmlSerializer

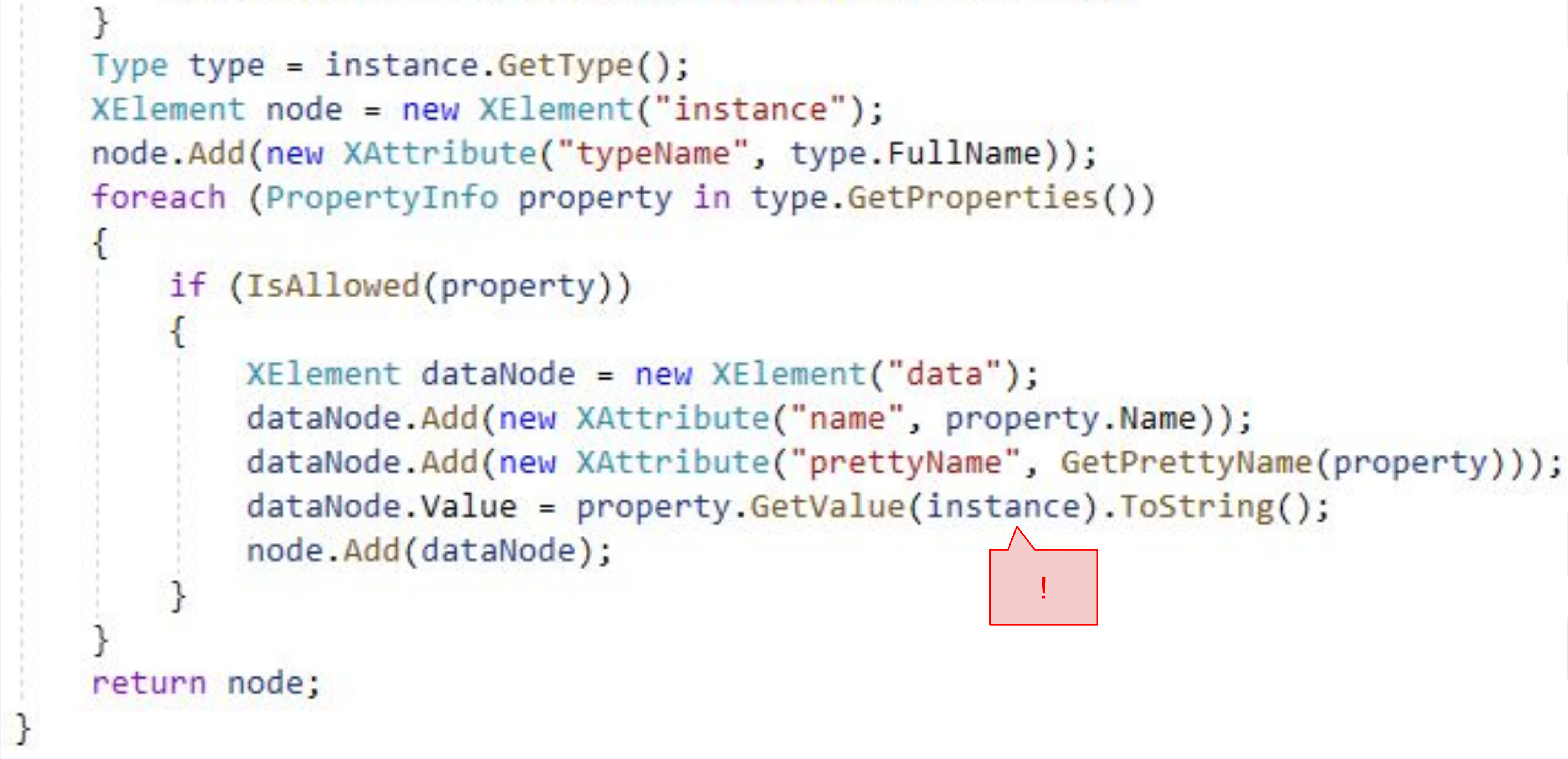


```
class XmlBuilder
{
    string GetPrettyName(PropertyInfo property)
    {
        var attr = property.GetCustomAttribute<DisplayNameAttribute>();
        return attr == null ? property.Name : attr.DisplayName;
    }
    bool IsAllowed(PropertyInfo property)
    {
        return property.GetCustomAttribute<ExcludeFromXmlAttribute>() == null;
    }
```

Example – XmlSerializer



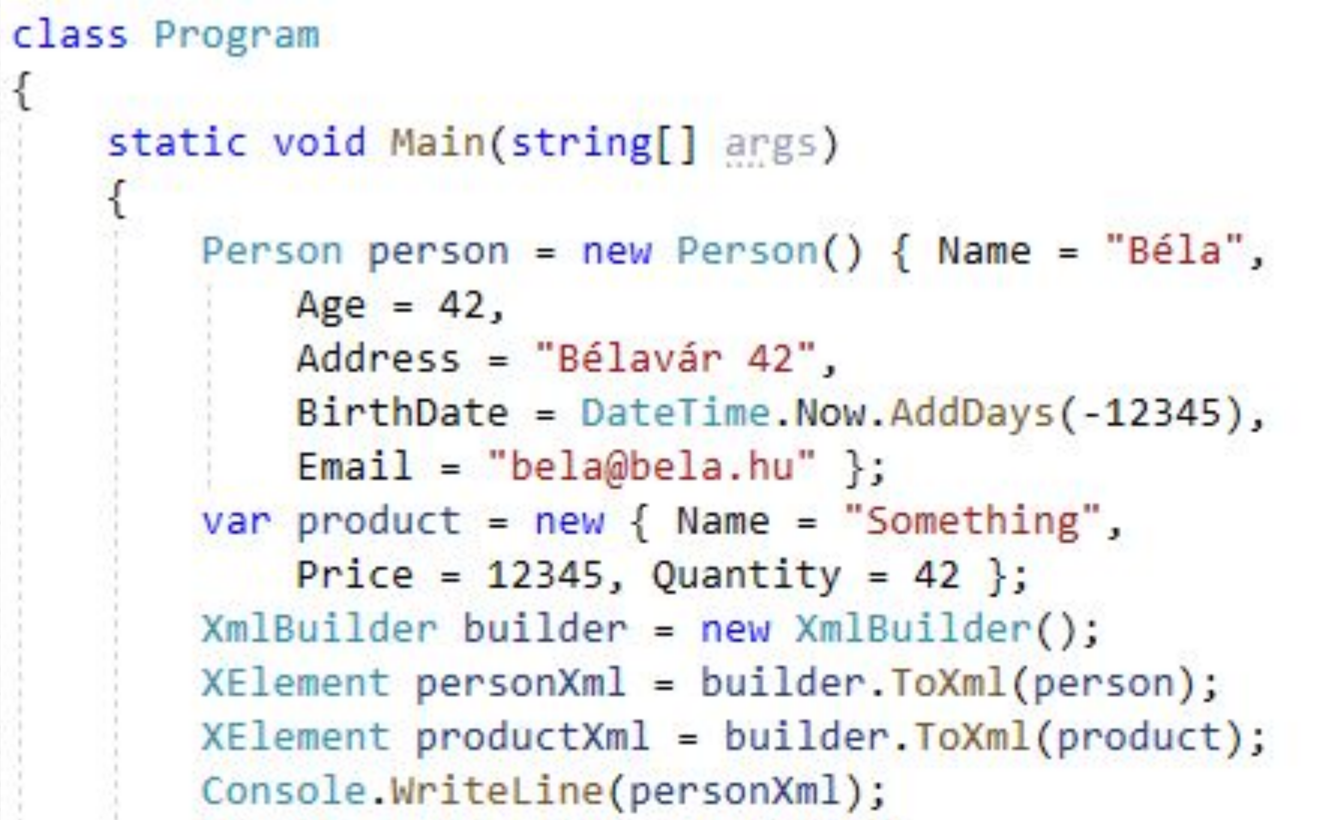
```
<instance typeName="Lecture_XmlSerializer.Person">
  <data name="Name" prettyName="Személynév">Béla</data>
  <data name="Email" prettyName="E-Mail cím">bela@bela.hu</data>
  <data name="Age" prettyName="Életkor">42</data>
  <data name="BirthDate" prettyName="Születési dátum">1986. 11. 27. 14:13:24</data>
</instance>
```



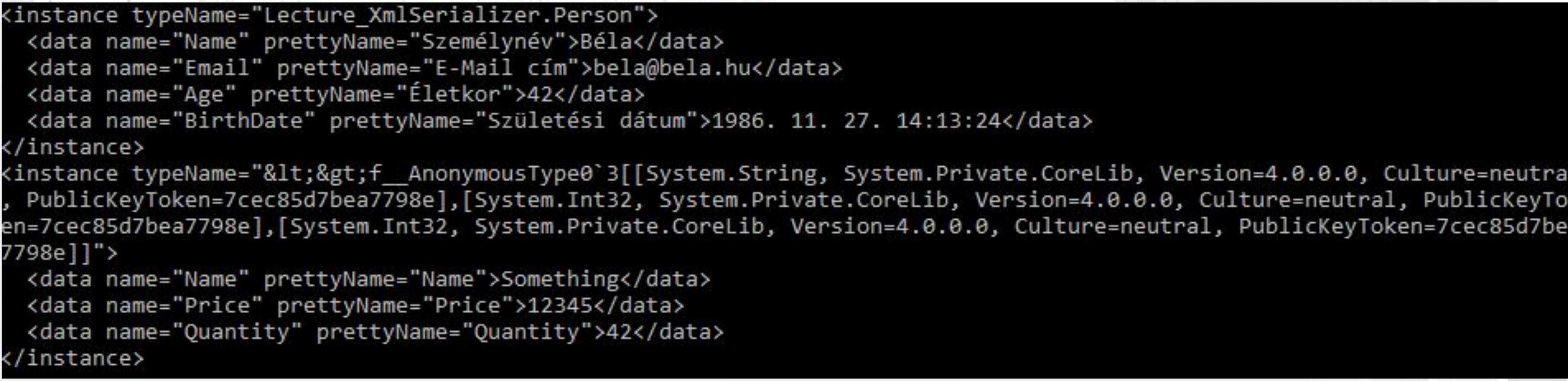
```
    }
    Type type = instance.GetType();
    XElement node = new XElement("instance");
    node.Add(new XAttribute("typeName", type.FullName));
    foreach (PropertyInfo property in type.GetProperties())
    {
        if (IsAllowed(property))
        {
            XElement dataNode = new XElement("data");
            dataNode.Add(new XAttribute("name", property.Name));
            dataNode.Add(new XAttribute("prettyName", GetPrettyName(property)));
            dataNode.Value = property.GetValue(instance).ToString();
            node.Add(dataNode);
        }
    }
    return node;
}
```

!

Example – XmlSerializer

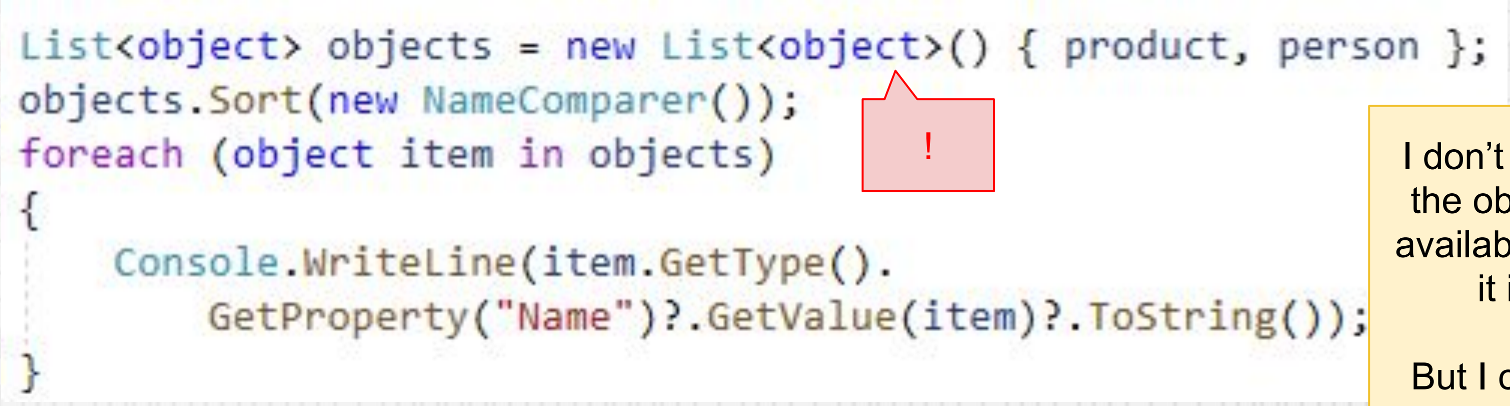


```
class Program
{
    static void Main(string[] args)
    {
        Person person = new Person() { Name = "Béla",
            Age = 42,
            Address = "Bélavár 42",
            BirthDate = DateTime.Now.AddDays(-12345),
            Email = "bela@bela.hu" };
        var product = new { Name = "Something",
            Price = 12345, Quantity = 42 };
        XmlBuilder builder = new XmlBuilder();
        XElement personXml = builder.ToXml(person);
        XElement productXml = builder.ToXml(product);
        Console.WriteLine(personXml);
```



```
<instance typeName="Lecture_XmlSerializer.Person">
  <data name="Name" prettyName="Személynév">Béla</data>
  <data name="Email" prettyName="E-Mail cím">bela@bela.hu</data>
  <data name="Age" prettyName="Életkor">42</data>
  <data name="BirthDate" prettyName="Születési dátum">1986. 11. 27. 14:13:24</data>
</instance>
<instance typeName="&lt;&gt;f__AnonymousType0`3[[System.String, System.Private.CoreLib, Version=4.0.0.0, Culture=neutra
, PublicKeyToken=7cec85d7bea7798e],[System.Int32, System.Private.CoreLib, Version=4.0.0.0, Culture=neutral, PublicKeyTo
en=7cec85d7bea7798e],[System.Int32, System.Private.CoreLib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=7cec85d7be
7798e]]">
  <data name="Name" prettyName="Name">Something</data>
  <data name="Price" prettyName="Price">12345</data>
  <data name="Quantity" prettyName="Quantity">42</data>
</instance>
```

Example – Sort by names

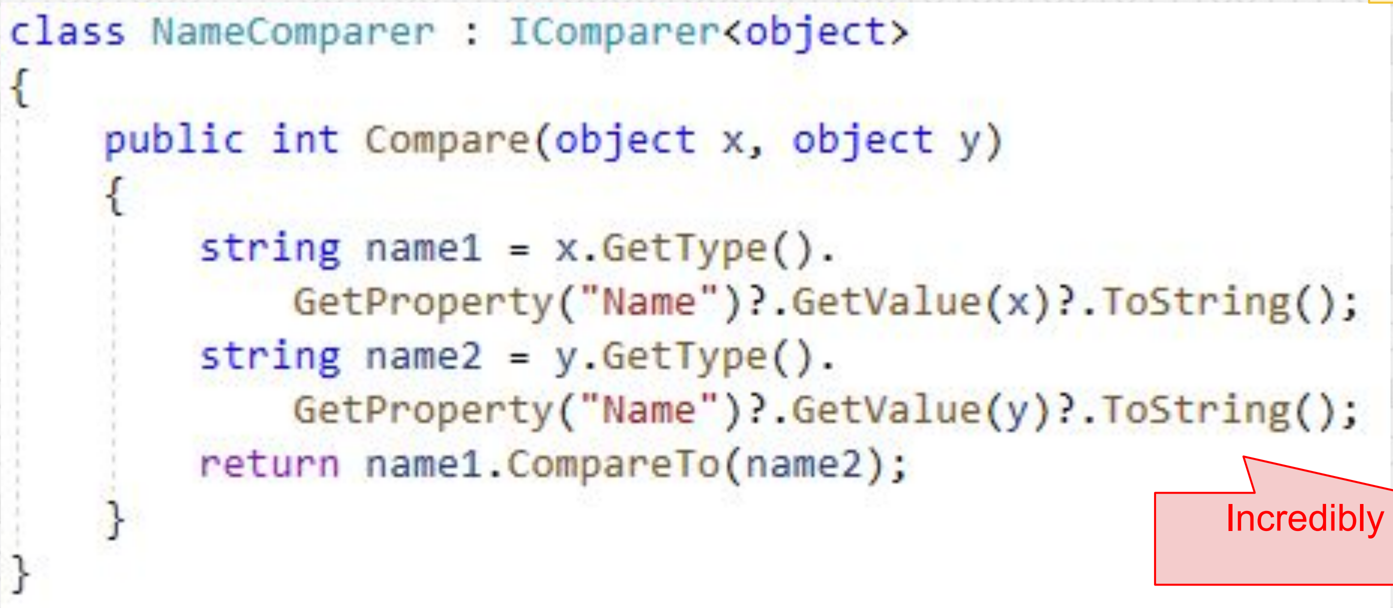


```
List<object> objects = new List<object>() { product, person };
objects.Sort(new NameComparer());
foreach (object item in objects)
{
    Console.WriteLine(item.GetType().
        GetProperty("Name")?.GetValue(item)?.ToString());
}
```

!

I don't know anything about the object, there is nothing available on the object since it is OBJECT type!

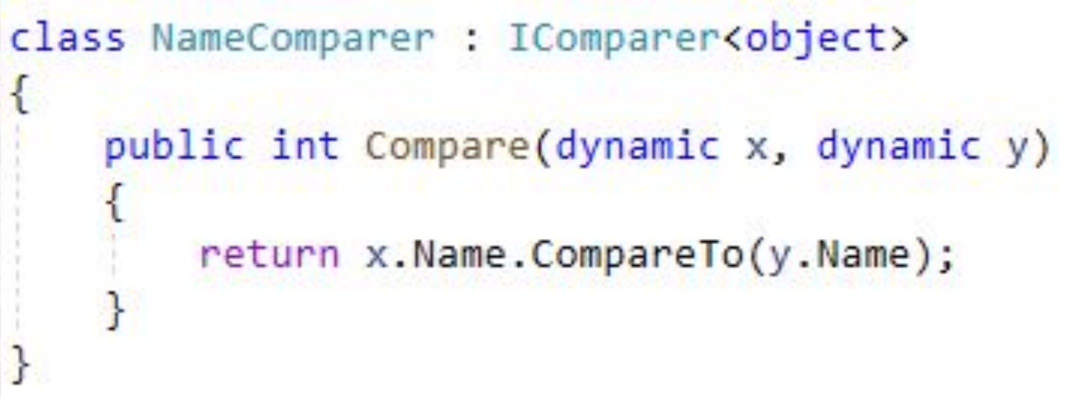
But I can check for it's *real* type and get it's value.



```
class NameComparer : IComparer<object>
{
    public int Compare(object x, object y)
    {
        string name1 = x.GetType().
            GetProperty("Name")?.GetValue(x)?.ToString();
        string name2 = y.GetType().
            GetProperty("Name")?.GetValue(y)?.ToString();
        return name1.CompareTo(name2);
    }
}
```

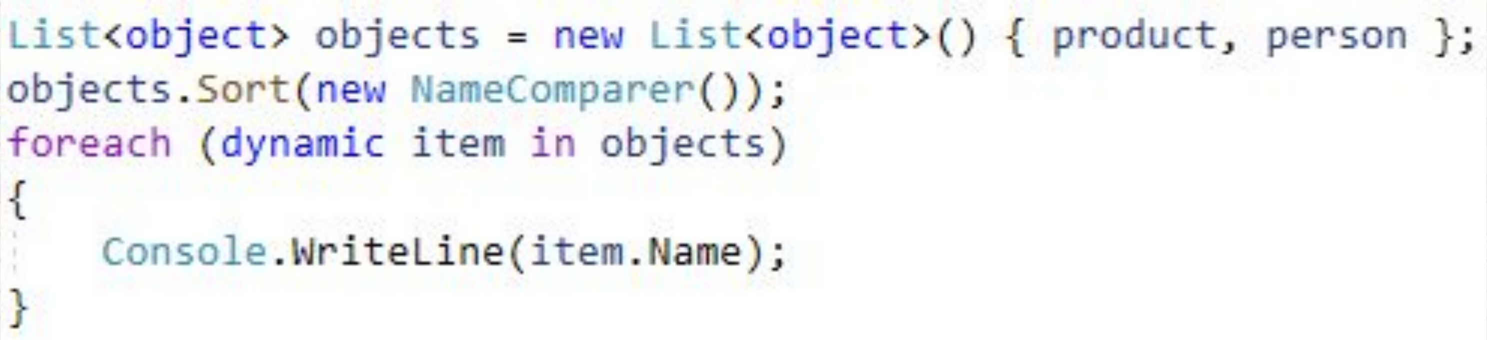
Incredibly slow, better avoid these...

Example – Sort by names



```
class NameComparer : IComparer<object>
{
    public int Compare(dynamic x, dynamic y)
    {
        return x.Name.CompareTo(y.Name);
    }
}
```

Dynamic won’t be asked back during the semester, but something interesting worth mentioning...



```
List<object> objects = new List<object>() { product, person };
objects.Sort(new NameComparer());
foreach (dynamic item in objects)
{
    Console.WriteLine(item.Name);
}
```

Dynamic means that the compiler should trust in me, that given X variable WILL BE type Y and WILL HAVE property Z.

In this case I have to type blindly “x.Name”... since the intellisense won’t understand / try to understand this, thanks to the ‘dynamic’ keyword.

Exercise / Reflection

- **Create a class that is capable of checking a custom object instance if it fulfils a set of custom rules (~ validation)**

- **Use reflection**
  - Using the RangeAttribute we want to set an upper and lower limit for an int property
  - Using the MaxLengthAttribute we want to set the maximum length of a string property
  - The matching MaxLengthValidation and RangeValidation classes will perform the true validation. Both classes should implement the IValidation interface, and the validtation should be done using a **`Validate(xxx)`** method
  - The ValidationFactory class is responsible for creating the good validator for the specified attribute instance
  - The Validator class should have a **`public bool Validate(object instance)`** method that actually performs the validation. It should get the appropriate validator instances from the factory, and call the **`Validate(xxx)`** method to perform the necessary checks

SOLID principles

- **S = Single Responsibility**