

제5장

닷넷 프레임워크와 어셈블리

변영철

1. 어셈블리 개요

- 메모장으로 간단한 C# 프로그램 작성하기
- C:\csharp> 폴더에 My.cs 파일

```
class My
{
    public static void Main()
    {
        System.Console.WriteLine("Hello,World!");
    }
}
```



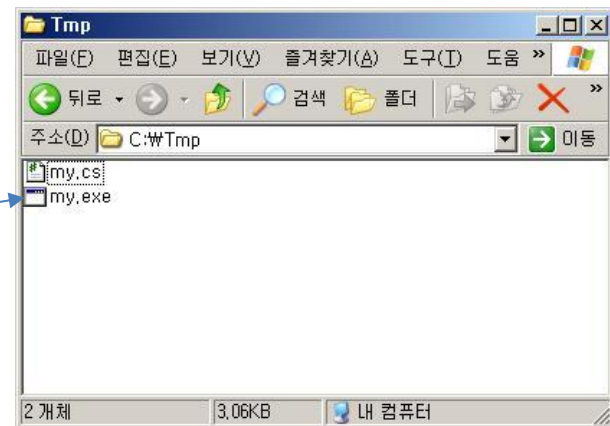
- 도스창 열고 컴파일하면 **exe 파일** 생성

C:\csharp>csc my.cs

- 실행

C:\csharp >my.exe

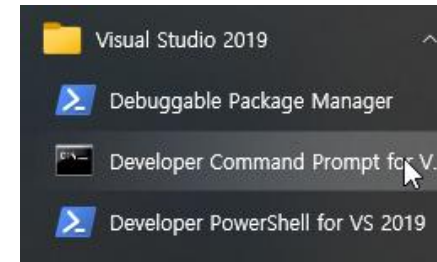
어셈블리



1. 어셈블리 개요

- 만일 (컴파일 불가) 오류가 나면?
- 컴파일 환경설정
 - 컴파일러 csc.exe가 있는 폴더를
환경변수 Path에 추가함
 - C:\WINDOWS\Microsoft.NET\Framework\v4.xx
- 혹은 시작 메뉴 클릭 후 Visual Studio에서 Developer Command 선택

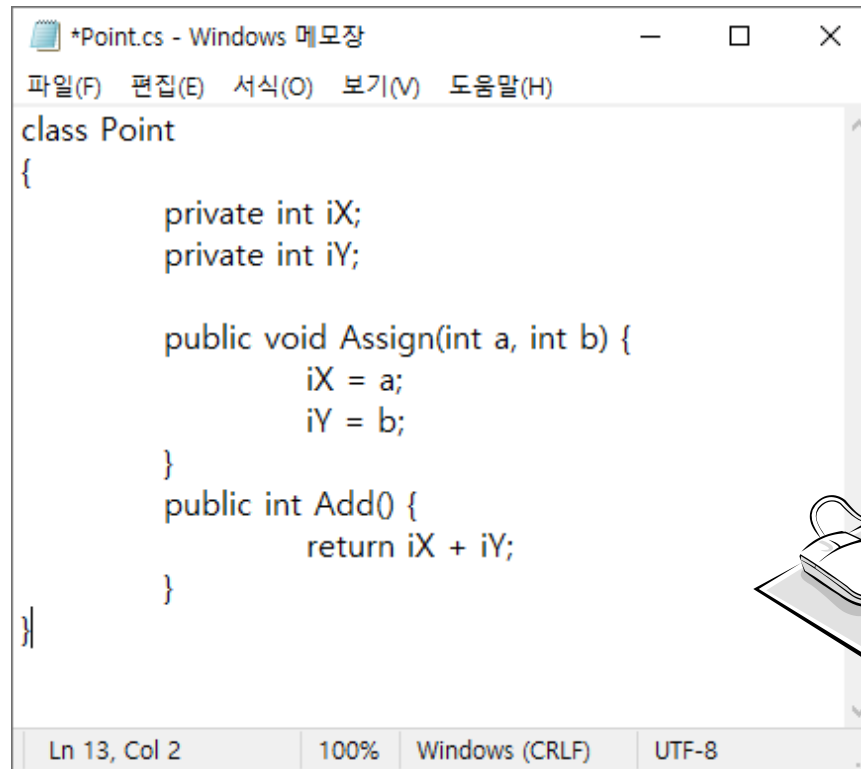
- 1.탐색기 실행
- 2.내 PC 위에서 오른쪽 버튼 클릭
- 3.속성 선택
- 4.고급 시스템 설정
- 5.환경변수 버튼 클릭
- 6.Path에 위 경로 추가



비주얼 스튜디오에서는
빌드 | 솔루션 빌드 메뉴 선택하면
→ 어셈블리 생성

1. 어셈블리 개요

- Main 함수가 없는 dll 라이브러리 어셈블리 생성해 보기



```
*Point.cs - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

class Point
{
    private int iX;
    private int iY;

    public void Assign(int a, int b) {
        iX = a;
        iY = b;
    }
    public int Add() {
        return iX + iY;
    }
}
```

Ln 13, Col 2 100% Windows (CRLF) UTF-8

- 도스창 열고 컴파일하면 dll 파일 생성

C:\csharp>csc /t:library Point.cs

어셈블리 생성

1. 어셈블리 개요

- My.cs에서 Point.dll 이용해보기

```
class My
{
    static void Main() {
        Point gildong = new Point();
        gildong.Assign(2, 3);
        double dArea = gildong.Add();

        System.Console.WriteLine("Area: " + dArea);
    }
}
```

C:\csharp>csc My.cs



Point 클래스가 My.cs 파일 안에
작성되어 있지 않아서

1. 어셈블리 개요

- My.cs 파일 안에 있으면 문제가 없을텐데...

```
class Point {  
    private int iX;  
    private int iY;  
  
    public void Assign(int a, int b) {  
        iX = a;  
        iY = b;  
    }  
    public int Add() {  
        return iX + iY;  
    }  
}
```

```
class My {  
    static void Main() {  
        Point gildong = new Point();  
        gildong.Assign(2, 3);  
        double dArea = gildong.Add();  
  
        System.Console.WriteLine("Area: " + dArea);  
    }  
}
```

C:\csharp>csc My.cs **OK!**

1. 어셈블리 개요

- 다른 파일에 있을 경우에는...

```
class My
{
    static void Main()
    {
        Point gildong = new Point();
        gildong.Assign(2, 3);
        double dArea = gildong.Add();

        System.Console.WriteLine("Area: " + dArea);
    }
}
```

My.cs

```
class Point {
    private int iX;
    private int iY;

    public void Assign(int a, int b) {
        iX = a;
        iY = b;
    }
    public int Add() {
        return iX + iY;
    }
}
```

Point.dll

C:\csharp>csc My.cs /reference:Point.dll

1. 어셈블리 개요

- public 클래스

```
class My
{
    static void Main()
    {
        Point gildong = new Point();
        gildong.Assign(2, 3);
        double dArea = gildong.Add();

        System.Console.WriteLine("Area: " + dArea);
    }
}
```

My.cs

```
public class Point {
    private int iX;
    private int iY;

    public void Assign(int a, int b) {
        iX = a;
        iY = b;
    }
    public int Add() {
        return iX + iY;
    }
}
```

Point.dll

C:\csharp>csc My.cs /reference:Point.dll

OK!

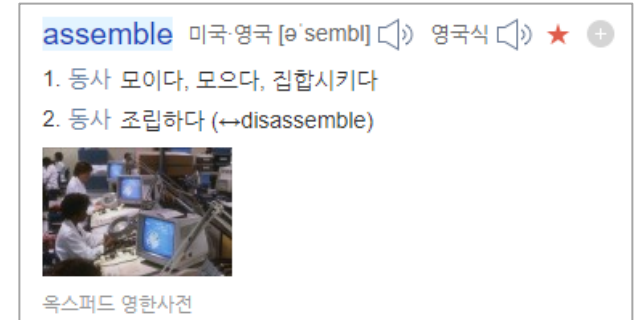
1. 어셈블리 개요

- my.exe, Point.dll

- 닷넷 환경에서 실행되는 파일 (여기에서는 C#으로 작성)
- Main이 없는, C#으로 만든 dll 라이브러리도 어셈블리
- 어셈블리(모은 것, **assembly**)라고 하는 이유는?
 - 여러분이 이제까지 봐왔던 실행 파일(가령 아래아 한글 hwp.exe)과 다르기 때문

- 어셈블리 = 모은 것

- 관련된 내용을 한 파일에 모은 것
- 그러면, 모으지 않은 것도 있나?
 - 아래아 한글 hwp.exe 파일은 기계어 실행 코드만 있지, 이와 관련된 것을 모은 것이 아니다.



1. 어셈블리 개요

- 기존의 실행 파일, 라이브러리 파일의 문제점
 - 가령 hwp.exe , 혹은 C:\Windows\System32 폴더에 있는 수많은 dll 파일들
 - 다른 OS(가령 유닉스)에서 실행 불가 (문제점1)
 - dll을 사용하는 프로그램을 작성하려면 dll을 설명하는 헤더 파일(가령 C언어의 stdio.h, C++의 iostream.h 등)이 따로 준비되어 있어야 함. (문제점2)
- 그렇다면 어셈블리는 무엇을 한 파일에 모은 것일까?
 - 실행 코드
 - 실행 코드에 대한 관련 정보를 함께 모음 (일종의 헤더 파일과 유사한 정보 포함)

2. 닷넷과 어셈블리

- 닷넷, 닷넷 런타임, 닷넷 프레임워크, 닷넷 플랫폼
 - 마이크로소프트에서 만든 자바 가상머신(?)
- 결국 닷넷은 어셈블리 실행환경
 - 어셈블리는 닷넷에서 실행되는 코드 (자바 바이트 코드?)



2. 닷넷과 어셈블리

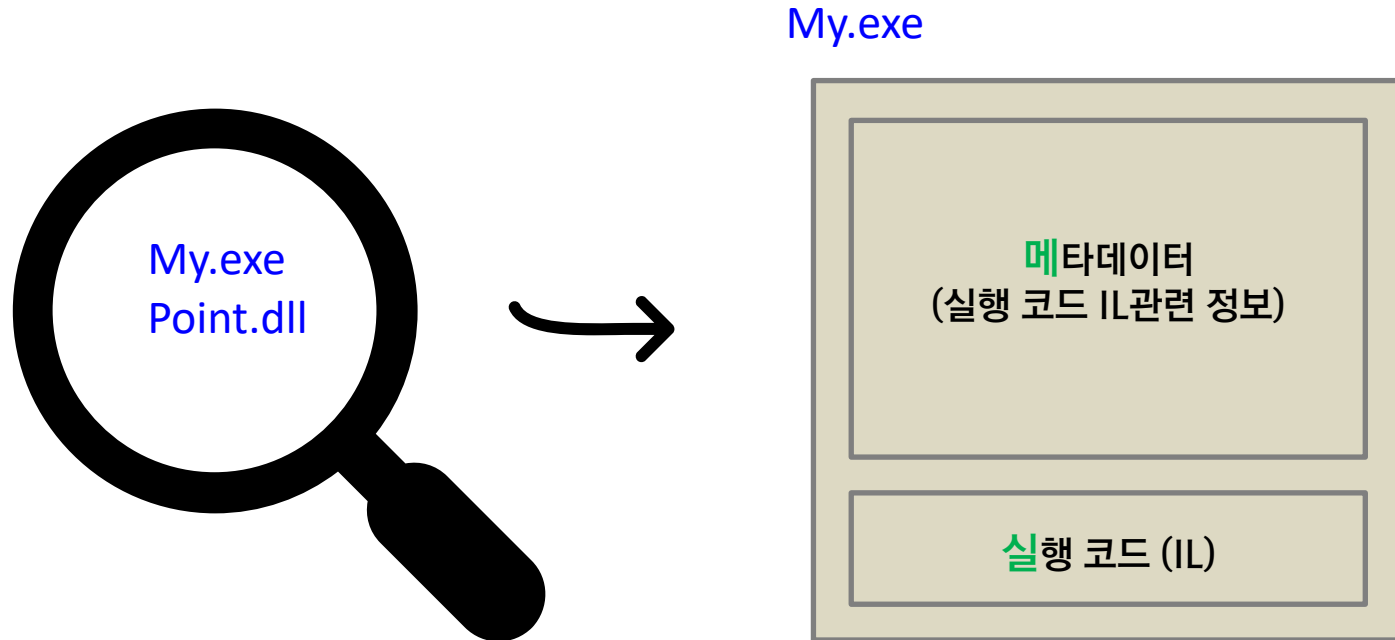
- 닷넷 런타임이 설치되어 있기만 하면
 - 윈도우, 맥 OS, 유닉스, 리눅스 등 운영체제에 관계없이 어셈블리 실행 가능
- 어셈블리 = 실행 코드 + 관련정보(메타데이터)
 - 실행 코드는 기계어가 아닌 중간 형태의 코드 (IL, Intermediate Language). 자바 바이트 코드와 유사
 - 또한, 실행 코드(IL)에 대한 정보 (metadata)도 선언되어 있음.

3. 어셈블리 살펴보기

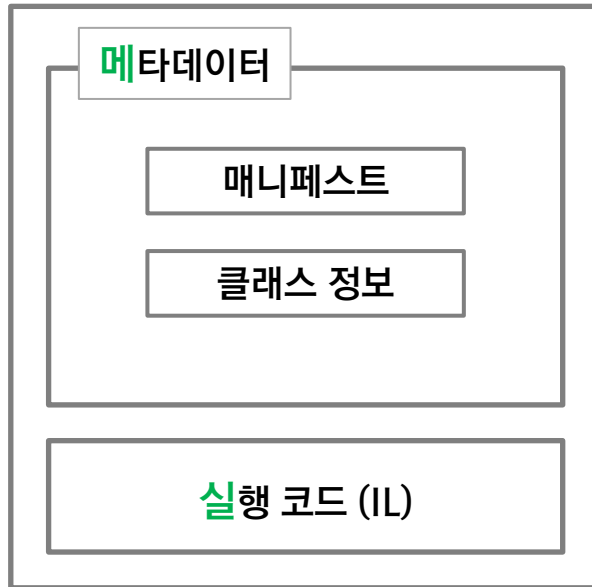


어셈블리 = 실행 코드 + 관련정보(메타데이터)

3. 어셈블리 살펴보기



3. 어셈블리 살펴보기



매니페스트의 의미

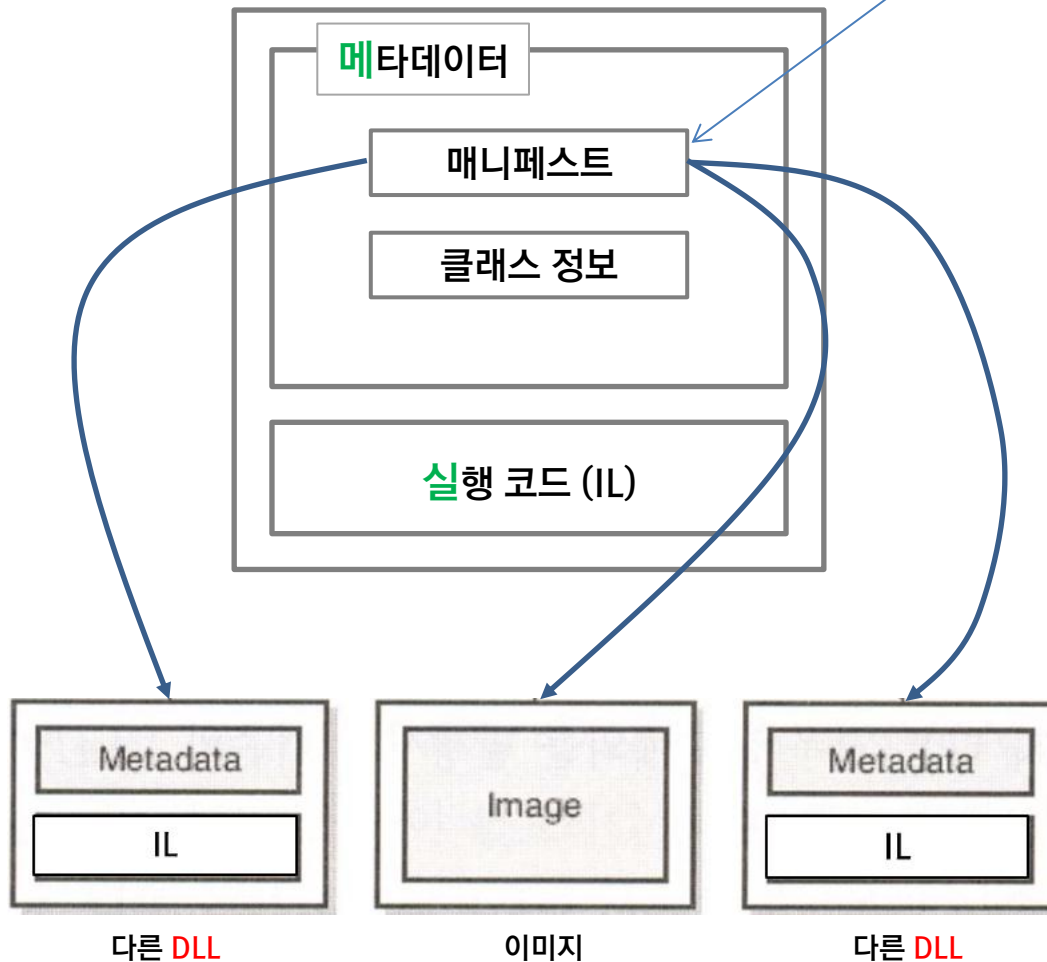
- 명백한, 뚜렷한 (손으로 만져서 알 수 있을 만큼)
- 어셈블리에 대한 정보를 명확하게 보여주는 것

매니페스토(manifesto)?

- 자신의 (정치적) 주장과 견해를 분명히 밝히는 연설(문)
- 정당의 정책과 정강

3. 어셈블리 살펴보기

어셈블리 이름, 버전, 사용하는
다른 어셈블리(라이브러리)
정보, 검증 키 등을 **명확하게**
보여주는 것

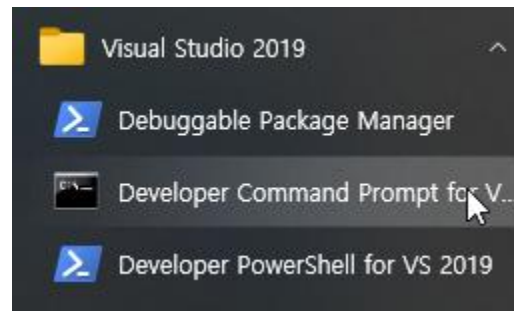


3. 어셈블리 살펴보기

- 닷넷 응용 프로그램 작성하는 것 = 어셈블리 만드는 것
- 모든 정보가 한 파일에 들어 있어서 쉽게 설치 및 배포 가능 → 설치/배포 기본 단위 (장점)
- 닷넷 런타임이 설치되어 있기만 하면 어떤 플랫폼(윈도, 맥, 리눅스 등)에서도 어셈블리 실행 가능 (장점)

4. 어셈블리 분석하기

- 어셈블리 분석 도구인 ildasm.exe를 이용하면
- 어셈블리(my.exe)에 들어있는 내용을 볼 수 있음 → 내용이 뭘까? (메실, 메타데이터와 IL 실행코드)
- ildasm.exe 도구
 - C:\Program Files (x86)\Microsoft SDKs\Windows \v10.0A\Bin\NETFX 4.8 Tools 폴더에 있음 (PATH에 추가)
- 혹은 시작 메뉴 클릭 후 Visual Studio에서 Developer Command 선택



4. 어셈블리 분석하기

- My.cs 파일 수정 및 어셈블리 생성

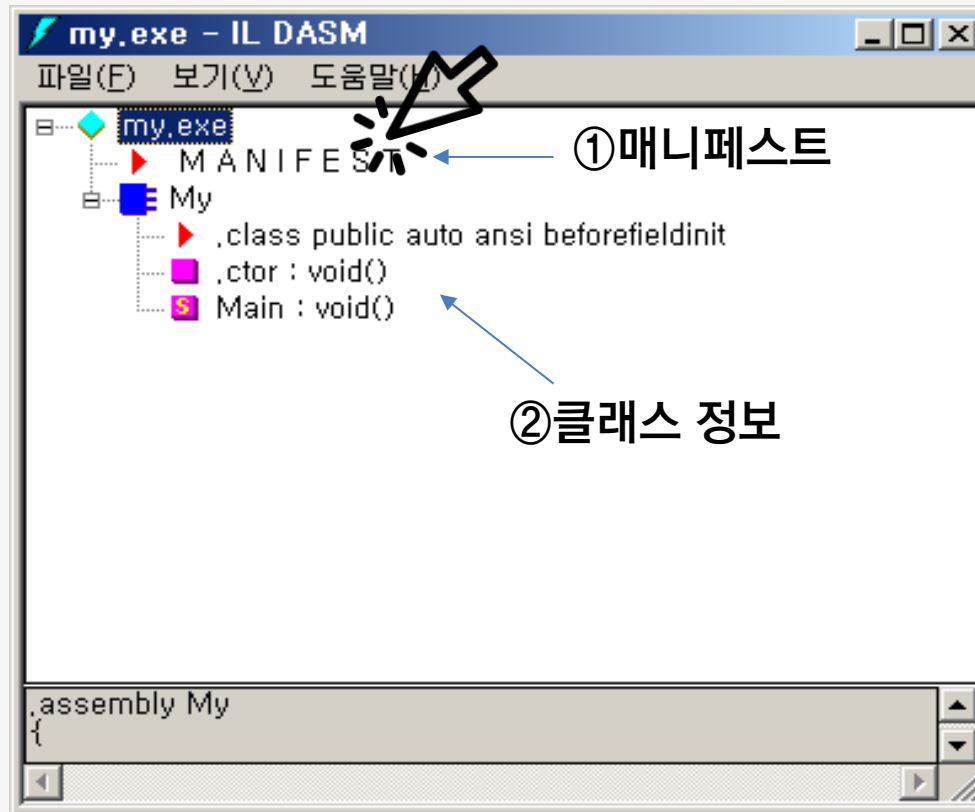
```
public class My
{
    public static void Main()
    {

    }
}
```

```
C:\csharp>csc my.cs
```

4. 어셈블리 분석하기

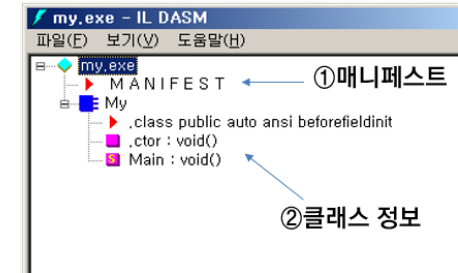
C:\csharp>ildasm my.exe



} 메타데이터

4. 어셈블리 분석하기

• 매니페스트 살펴보기

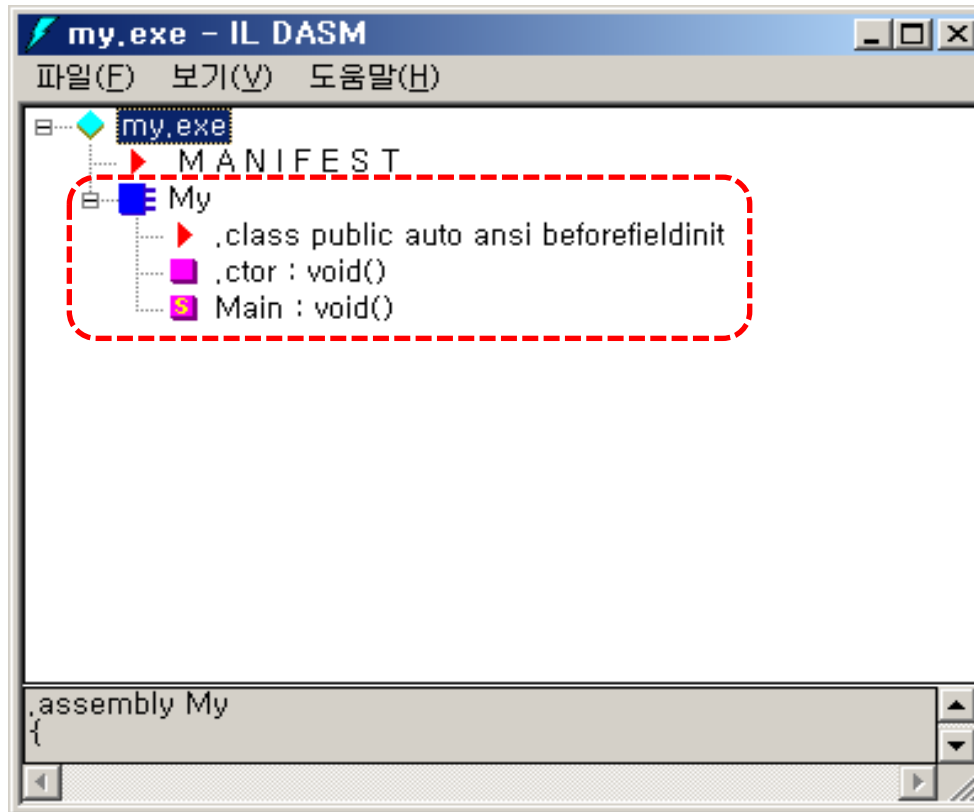
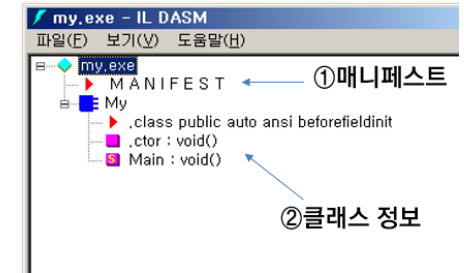


```
MANIFEST
.assembly extern mscorlib
{
    .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z
    .ver 1:0:5000:0
}
.assembly My
{
    // --- 다음 사용자 지정 특성이 자동으로 추가됩니다. 주석으로 처리하십시오.
    // .custom instance void [mscorlib]System.Diagnostics.DebuggableAttribute:
    //
    .hash algorithm 0x00000004
    .ver 0:0:0:0
}
.module My.exe
// MVID: {F4544A98-5AF3-4BA4-9F0F-D5D35333ED0B}
.imagebase 0x00400000
.subsystem 0x00000003
.file alignment 512
.corflags 0x00000001
// Image base: 0x00d90000
```

- 현 어셈블리 이름(My.exe)
- 현 어셈블리가 사용하는 다른 어셈블리 이름(mscorlib)과 어셈블리 버전(x:x:x:x),
- 사용하는 어셈블리(mscorlib)를 인증하는 키(publickeytoken)
→ 엉뚱한 mscorlib가 사용되는 것

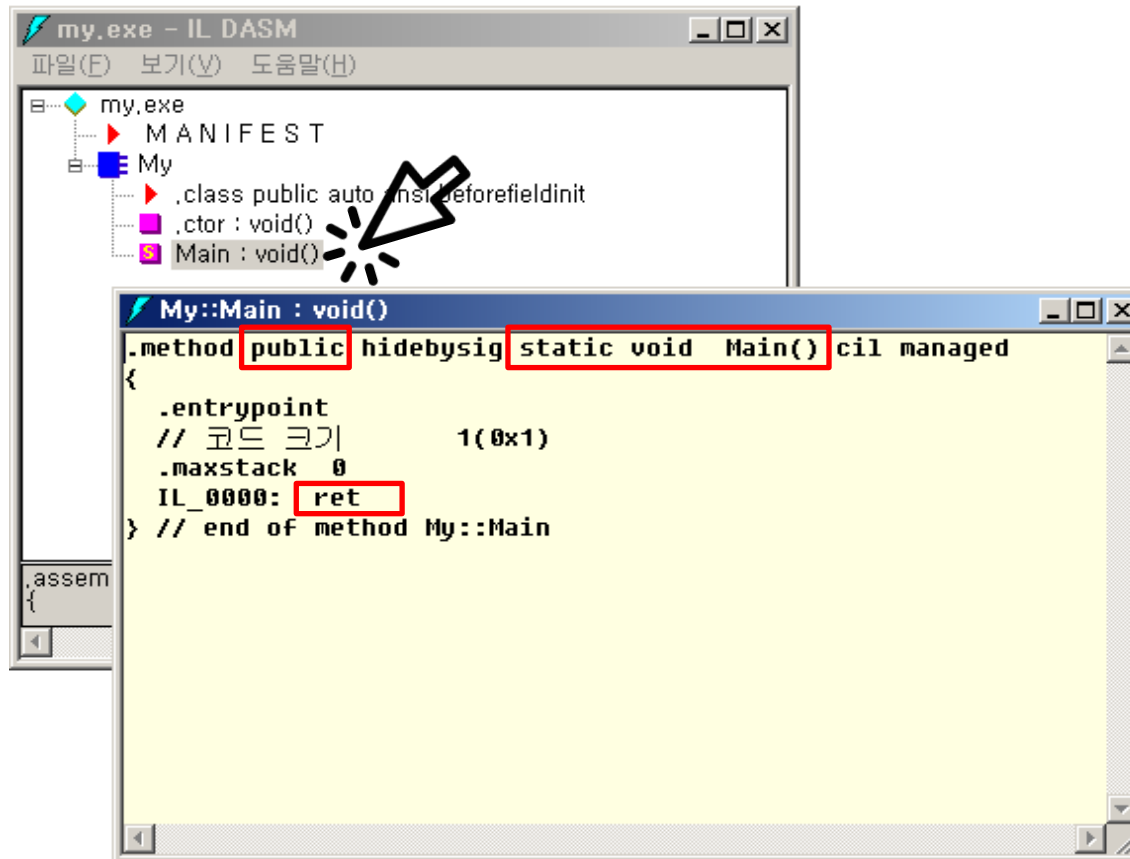
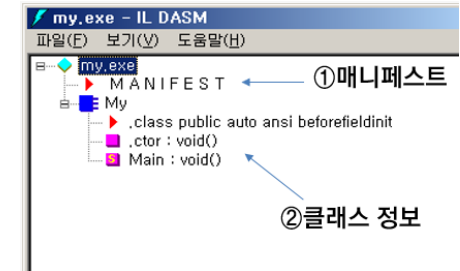
4. 어셈블리 분석하기

- 어떤 클래스가 있는지 볼 수 있음.



4. 어셈블리 분석하기

- IL 실행 코드 보기



4. 어셈블리 분석하기

```
class My
```

```
{
```

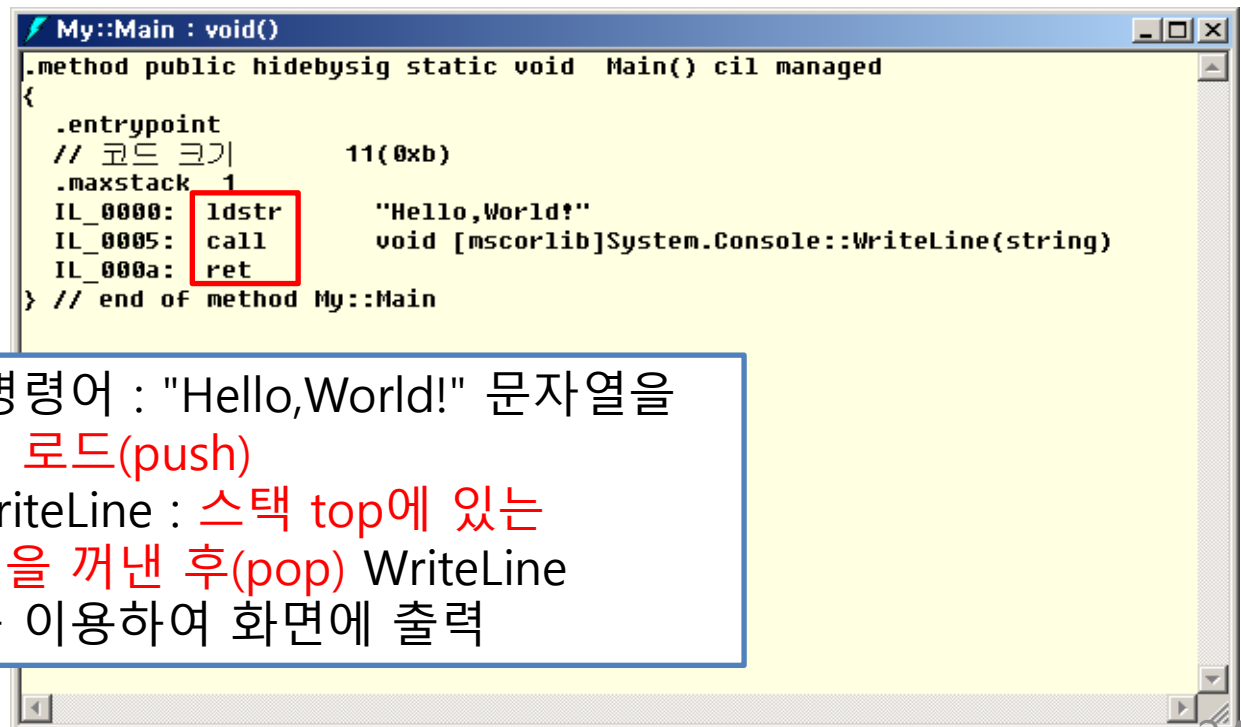
```
    public static void Main()
```

```
{
```

```
        System.Console.WriteLine("Hello,World!");
```

```
}
```

```
}
```



```
My::Main : void()
.method public hidebysig static void Main() cil managed
{
    .entrypoint
    // 코드 크기      11(0xb)
    .maxstack 1
    IL_0000: ldstr      "Hello,World!"
    IL_0005: call       void [mscorlib]System.Console::WriteLine(string)
    IL_000a: ret
} // end of method My::Main
```

1. ldstr 명령어 : "Hello,World!" 문자열을
스택에 로드(push)
2. call WriteLine : 스택 top에 있는
문자열을 꺼낸 후(pop) WriteLine
함수를 이용하여 화면에 출력

4. 어셈블리 분석하기

- Point.cs 파일 수정 (Main 함수가 없음)

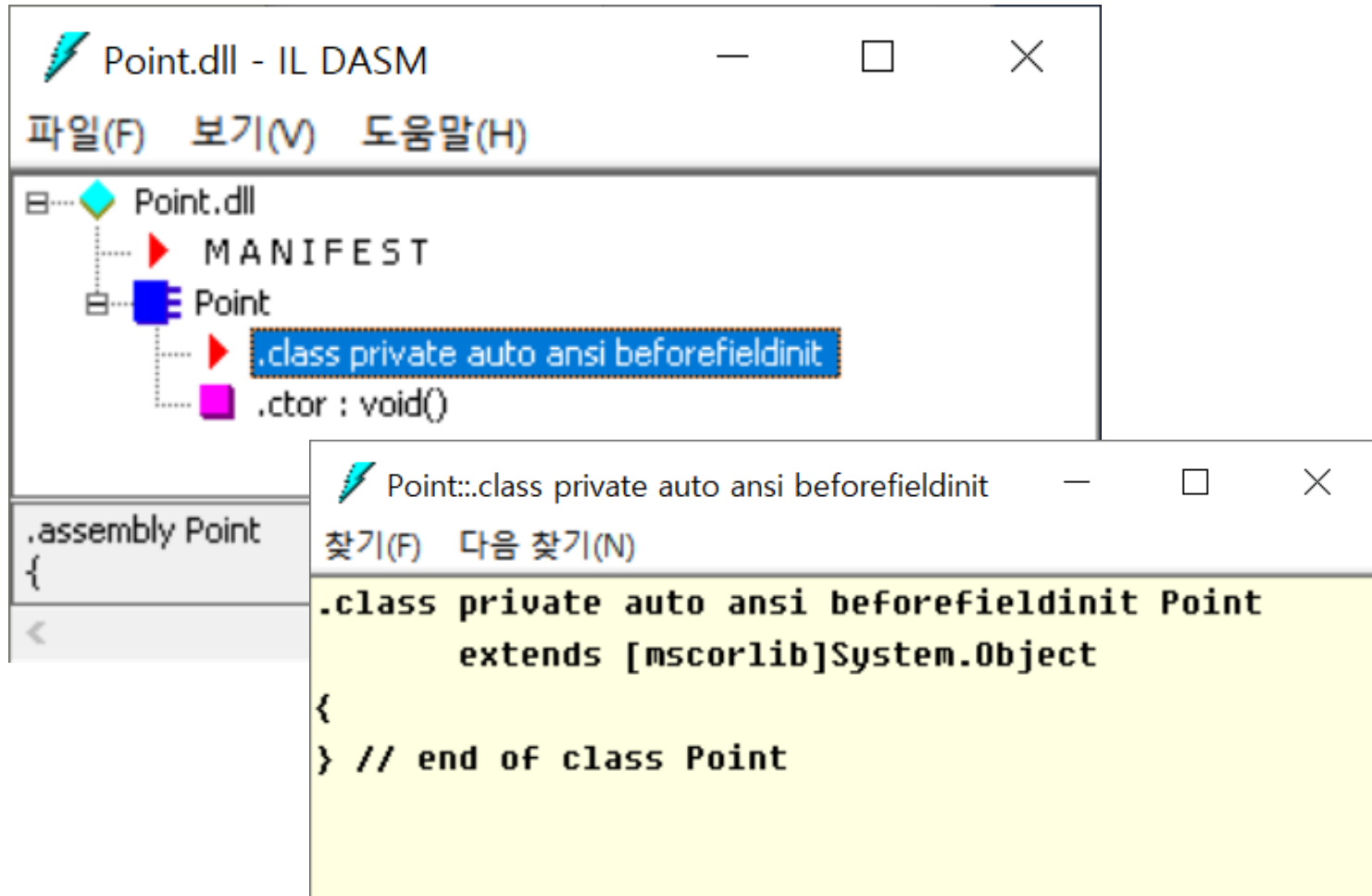
```
public class Point
{
    public Point()
    {
    }
}
```

} 생성자 함수

C:\W\Tmp>csc /t:library Point.cs

4. 어셈블리 분석하기

- ildasm으로 분석하기

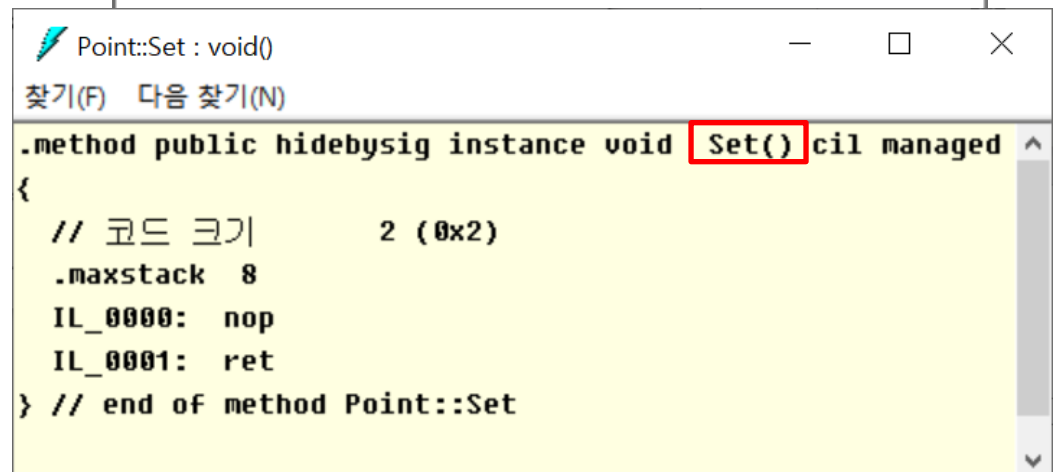
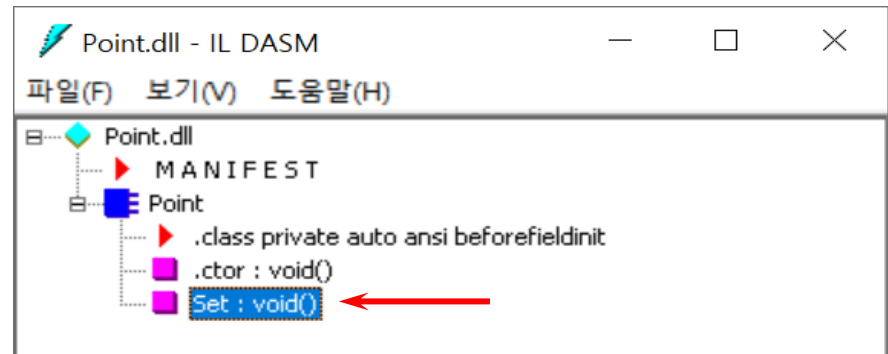


4. 어셈블리 분석하기

- 멤버함수 set 선언

```
public class Point
{
    public Point()
    {
    }
}
```

```
public void Set()
{
}
}
```



```
C:\wsharp>csc /t:library Point.cs
```

```
C:\wsharp>ildasm Point.dll
```

4. 어셈블리 분석하기

• 문자열 출력

```
public class Point
{
    public Point()
    {
    }

    public void Set()
    {
    }
}
```

1. 문자열 "Hello,World!"를 스택 top에 로드(ldstr)
2. 닷넷 프레임워크에 있는 기본 클래스 라이브러리(Basic Class Library) Console의 WriteLine 함수 호출(call). WriteLine 함수는 스택 top에 있는 문자열, "Hello,World!"를 꺼내어(pop) 화면에 출력
3. return

```
.method public hidebysig instance void set() cil managed
{
    // Code size          11 (0xb)
    .maxstack 1
    IL_0000: ldstr      "Hello,World!"
    IL_0005: call       void [mscorlib]System.Console::WriteLine(
    IL_000a: ret
} // end of method My::set
```

`System.Console.WriteLine("Hello,World!");`

```
C:\wsharp>csc /t:library Point.cs
```

```
C:\wsharp>ildasm Point.dll
```

4. 어셈블리 분석하기

• 멤버변수 선언

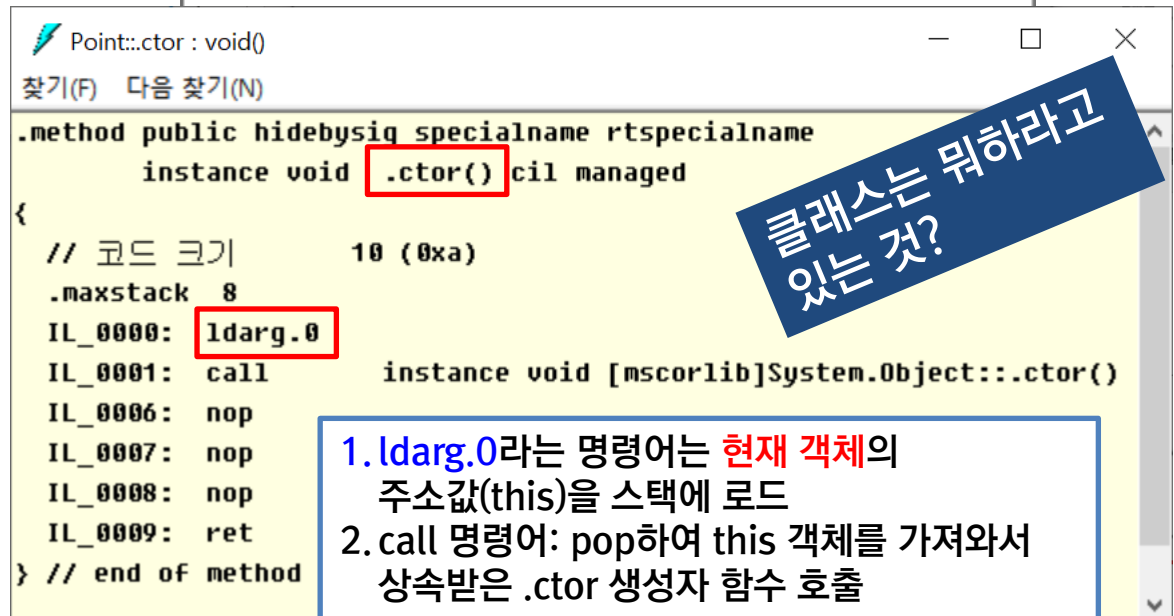
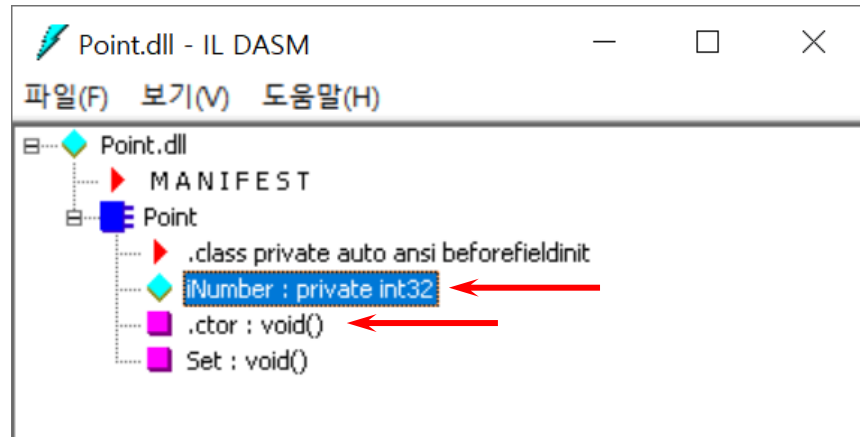
using System;

```
public class Point
{
```

private int iNumber;

```
public Point()
{
}
```

```
public void Set()
{
}
}
```



4. 어셈블리 분석하기

• 멤버변수 초기화

using System;

```
public class Point
{
```

```
    private int iNumber = 0;
```

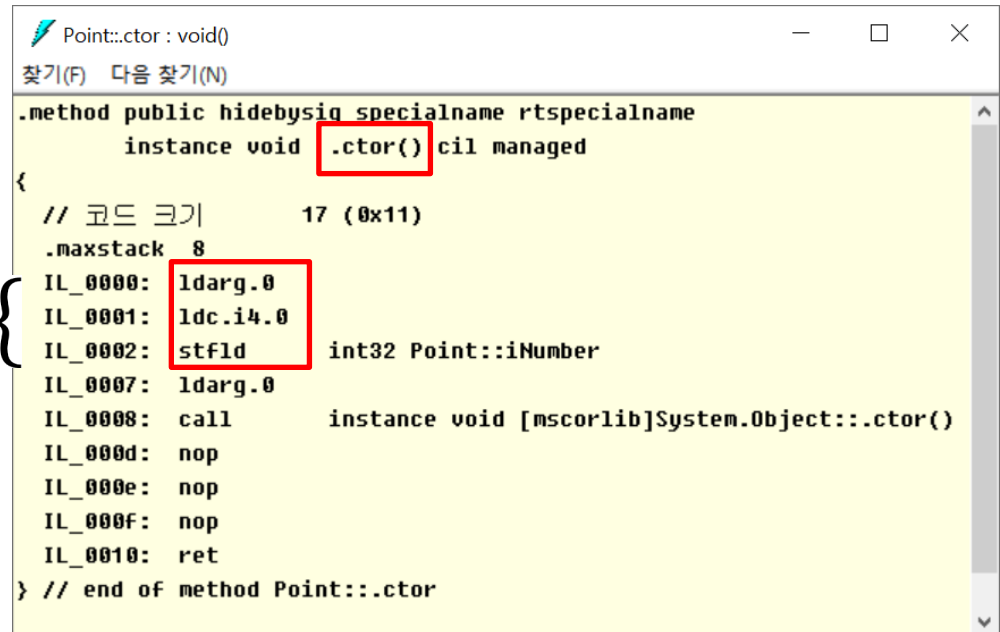
```
    public Point()
    {
```

```
    }
```

```
    public void Set()
    {
```

```
    }
```

```
}
```



```
Point.ctor : void()
찾기(F) 다음 찾기(N)

.method public hidebysig specialname rtspecialname
    instance void .ctor() cil managed
{
    // 코드 크기      17 (0x11)
    .maxstack 8
    IL_0000: ldarg.0
    IL_0001: ldc.i4.0
    IL_0002: stfld      int32 Point::iNumber
    IL_0007: ldarg.0
    IL_0008: call      instance void [mscorlib]System.Object::.ctor()
    IL_000d: nop
    IL_000e: nop
    IL_000f: nop
    IL_0010: ret
} // end of method Point::.ctor
```

1. ldarg.0 명령어에 의해 스택에 0번 파라미터 this를 로드
2. ldc.i4.0 명령어에 의해 0값을 스택에 push
3. stfld 명령어는 스택에 있는 두 값 this와 0을 pop하여 this.iNumber 멤버 변수에 0값을 저장하여 초기화

※ ldarg: load argument (파라미터를 로드)
ldc: load constant (상수를 로드)

4. 어셈블리 분석하기

- 매개변수 선언

```
using System;
```

```
public class Point  
{
```

```
    private int iNumber = 0;
```

```
    public Point()  
    {  
  
    }
```

```
    public void Set(int i)  
    {  
  
    }  
}
```

```
.method public hidebysig instance void Set(int32 i) cil managed  
{  
    // 코드 크기      1(0x1)  
    .maxstack 0  
    IL_0000: ret  
} // end of method My::set
```

4. 어셈블리 분석하기

• 코드 추가

```
using System;
```

```
public class Point  
{
```

```
    private int iNumber = 0;
```

```
    public Point()  
{
```

```
}
```

```
    public void Set(int i)  
{
```

```
        iNumber = i;
```

```
}
```

```
}
```

- Set 멤버함수 호출하는 코드의 예

```
Point gildong = new Point();  
gildong.Set(7);
```

- 하지만 아래와 같이 호출되는 형식임.

```
set(gildong, 7);
```

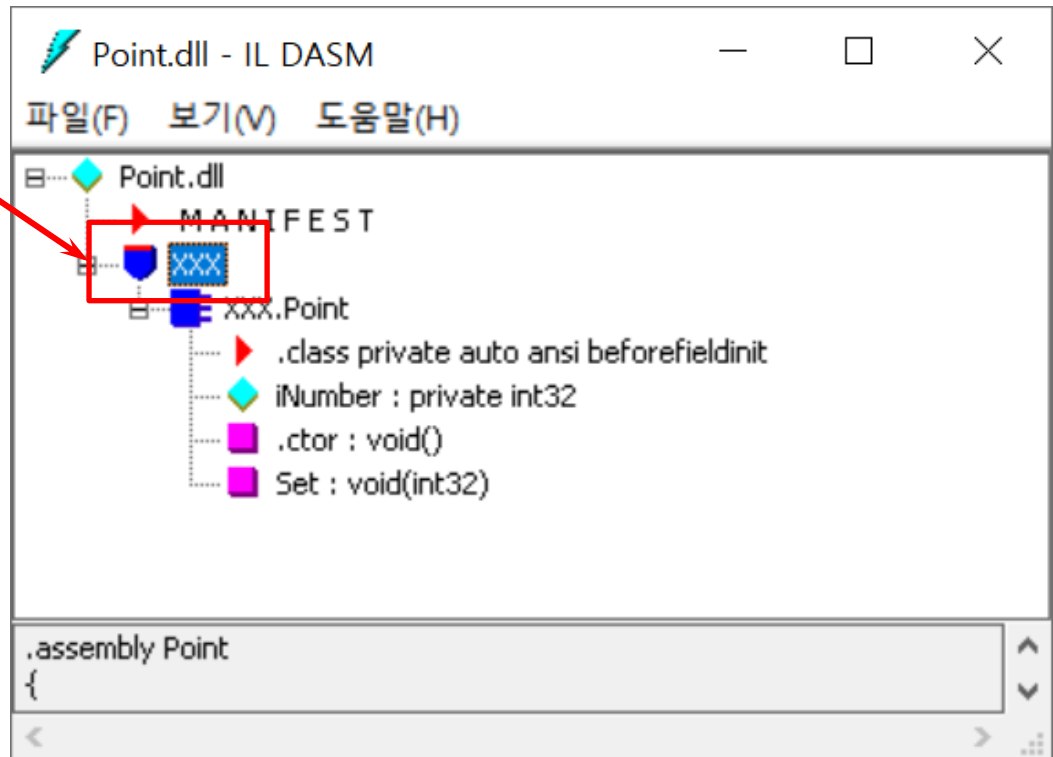
- 즉, 객체 gildong의 주소값(this)이 0번째 인자로 전송

```
.method public hidebysig instance void Set(int32 i) cil managed  
{  
    // 코드 크기      8(0x8)  
    .maxstack 2  
    IL_0000: ldarg.0 //0번째 인자(숨겨진 포인터 this)를 스택에 push  
    IL_0001: ldarg.1 //1번째 인자 i에 있는 값을 스택에 push  
    //this 객체의 멤버 iNumber에 1번 인자의 값을 저장함  
    IL_0002: stfld    int32 Point::iNumber  
    IL_0007: ret  
} // end of method Point::Set
```


4. 어셈블리 분석하기

- 네임스페이스 선언

```
namespace XXX {  
    using System;  
  
    public class Point  
    {  
        private int iNumber = 0;  
  
        public Point()  
        {  
  
        }  
  
        public void Set(int i)  
        {  
            iNumber = i;  
        }  
    }  
}
```



5. CLR과 기본 클래스 라이브러리

- 기본 클래스 라이브러리

- Base Class Library

- Console과 같이 기본적으로 존재하는 클래스들

- 기본 클래스 라이브러리는

C:\Windows\Microsoft.NET\Framework\v4.0.30319 폴더에 있는 **mscorlib.dll** 파일에 **System** 네임스페이스에 작성되어 있음

```
public class My
{
    public static void Main()
    {
        System.Console.WriteLine("Hello,World!");
    }
}
```

5. CLR과 기본 클래스 라이브러리

- 비주얼 베이직 프로그래밍, My7.vb

```
Module My7
```

```
    Sub Main()
```

```
        System.Console.WriteLine("Hello,World!")
```

```
    End Sub
```

```
End Module
```

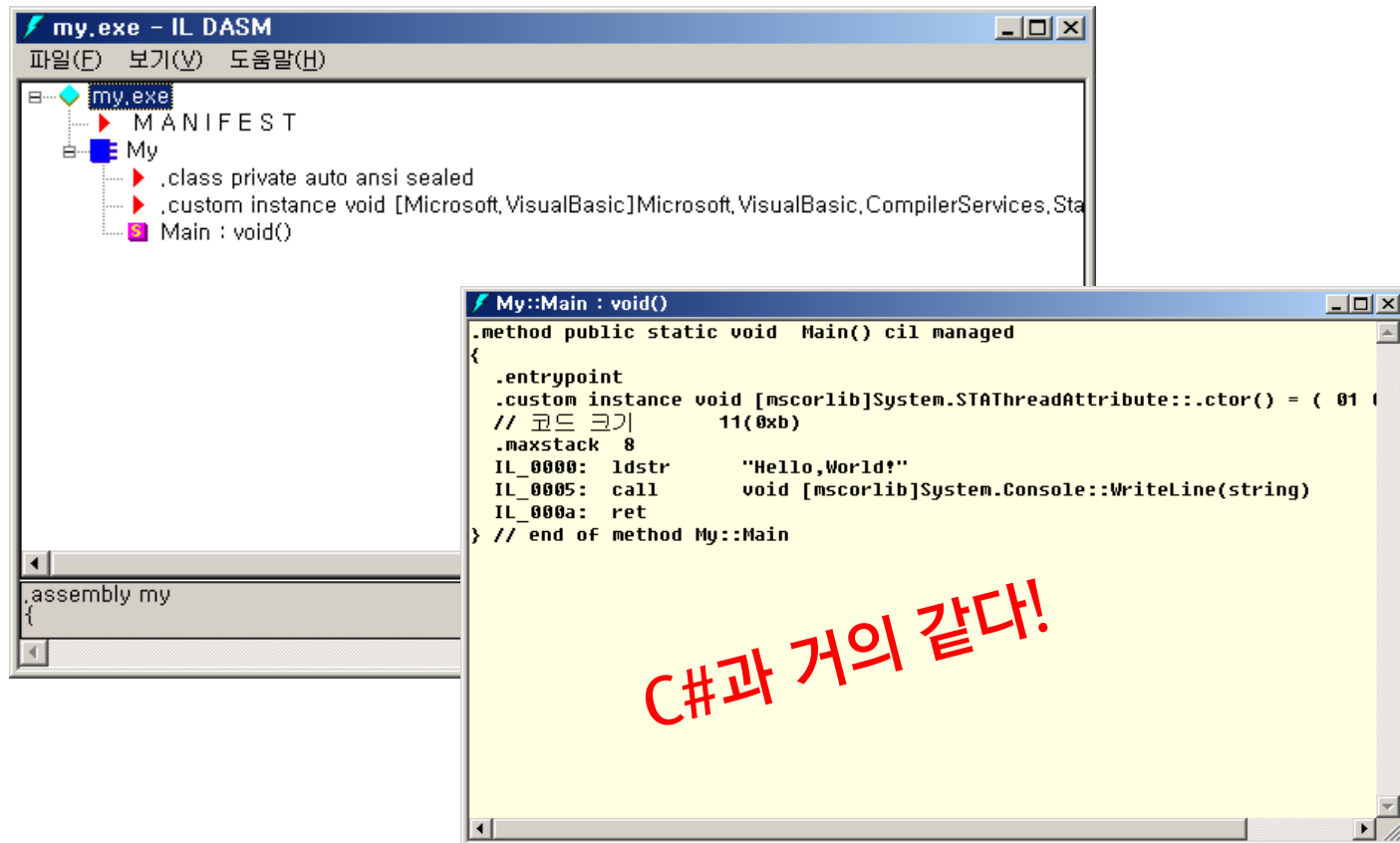
- 컴파일

```
C:\wsharp> vbc My7.vb
```

5. CLR과 기본 클래스 라이브러리

- 어셈블리 분석

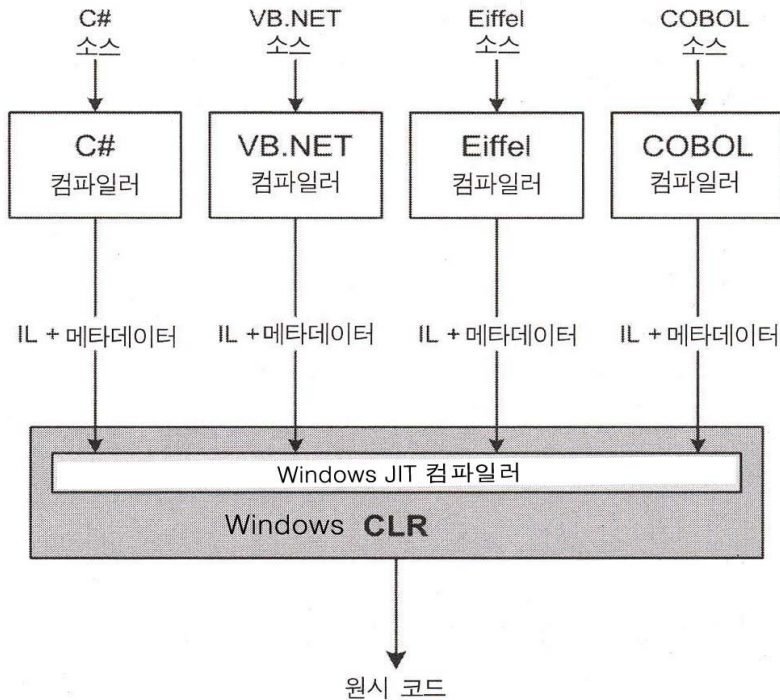
C:\csharp>ildasm My.exe



5. CLR과 기본 클래스 라이브러리

- C# vs. VB

- 어떤 언어로 작성하든 어셈블리는 동일



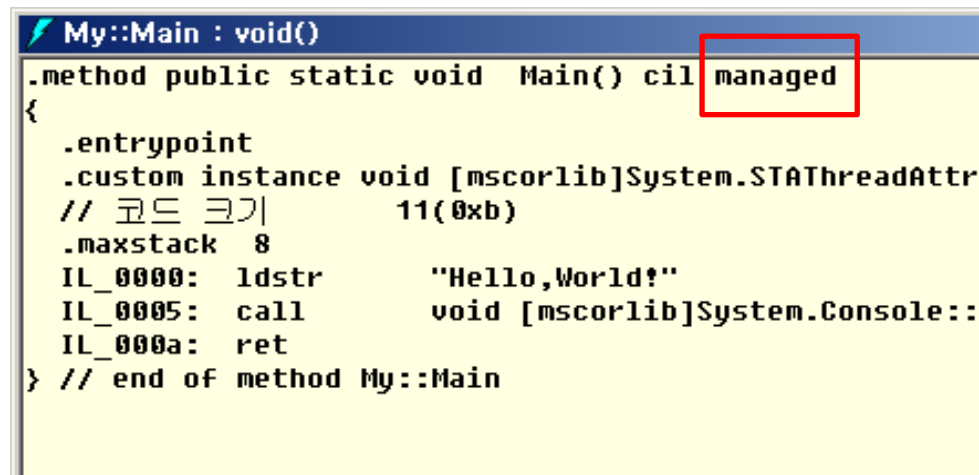
Java와
가장 다른 점 중 하나!!

5. CLR과 기본 클래스 라이브러리

- 따라서 중간 언어, IL을 도입한 이유
 - 운영체제 혹은 플랫폼(MS, 맥 OS, 리눅스 등)에 관계없이 닷넷 런타임만 설치되어 있으면 IL을 실행
 - 본인이 좋아하는 언어(C#, VC++, VB 등)로 프로그램을 작성할 수 있음.
 - 언어는 달라도 생성되는 IL 코드가 같음으로써 닷넷이 설치가 되어 있기만 하면 실행 가능

5. CLR과 기본 클래스 라이브러리

- CLR(Common Language **Runtime**)
 - 런타임, **가상머신** (자바가상머신?)
 - 런타임은 동적 할당된 객체를 자동으로 제거(garbage collection)
 - 이처럼 .NET에 의해 객체가 제거되고 관리되는 코드를 관리 코드(**managed** code)라 함.

A screenshot of the ILASM (Intermediate Language Assembler) window. The title bar reads "My::Main : void()". The main text area shows the assembly code for a method named "Main". The first line is ".method public static void Main() cil managed", where the word "managed" is highlighted with a red rectangular box. The code continues with a block opening brace "{", followed by ".entrypoint", ".custom instance void [mscorlib]System.STAThreadAttr", a comment "// 코드 크기" followed by "11(0xb)", ".maxstack 8", and three instructions: "IL_0000: ldstr 'Hello,World!'", "IL_0005: call void [mscorlib]System.Console::", and "IL_000a: ret". The code ends with a block closing brace "}" and a comment "// end of method My::Main".

```
My::Main : void()
.method public static void Main() cil managed
{
    .entrypoint
    .custom instance void [mscorlib]System.STAThreadAttr
    // 코드 크기      11(0xb)
    .maxstack 8
    IL_0000: ldstr      "Hello,World!"
    IL_0005: call        void [mscorlib]System.Console::
    IL_000a: ret
} // end of method My::Main
```

5. CLR과 기본 클래스 라이브러리

- JIT(Just-In-Time) 컴파일러
 - (실행될)때맞춰, 시간에 맞춰, 실행될 때 바로
 - CLR 구성 요소 중 하나
 - 실행하는 my.exe 어셈블리를 해당 플랫폼(가령 윈도우 운영체제)에서 실행되는 exe 파일로 just-in-time(실행될 때 바로, 제때에) 변환한 후 실행
 - 실행 속도가 빠름.

5. CLR과 기본 클래스 라이브러리

- 닷넷 자료형 분류 방법 1

```
int a;  
Point gildong = new Point();
```

- 기본(built-in) 자료형 (int 등 **기존에 있는 것**)
- **사용자 정의**(user-defined) 자료형 (Point와 같이 우리가 만든 것)

- 닷넷 자료형 분류 방법 2

```
int a; //new 없음  
Point gildong = new Point();
```

- **값**(value) 자료형
 - int, double 등. new 생성자 없이 객체가 생성됨.
- **참조**(reference) 자료형
 - *new* 명령어로 객체를 생성해야 함.
 - **사용자 정의 클래스** + Object + String 자료형

5. CLR과 기본 클래스 라이브러리

- 정리

- 어셈블리(assembly)
- 중간 언어(IL) 코드
- 메타데이터
- 매니페스트(manifest)
- ildasm 도구
- CLR(Common Language Runtime) 혹은 런타임
- JIT 컴파일러
- 쓰레기 수집
- 관리 코드
- 기본 클래스 라이브러리(Base Class Library)
- mscorlib.dll 어셈블리
- .NET 자료형