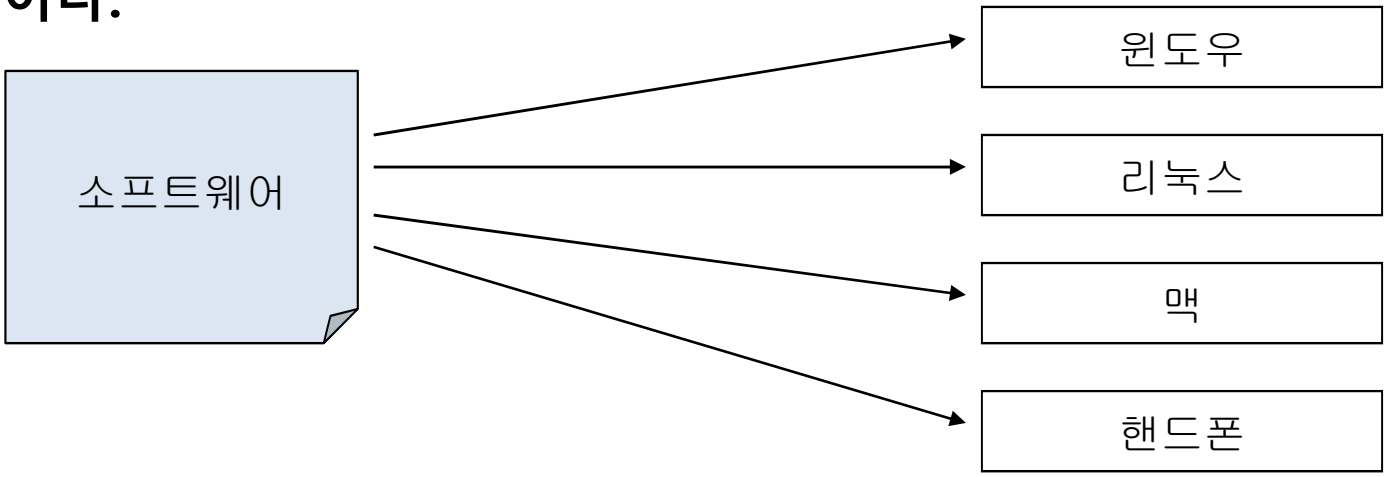
1장. 닷넷

등장 배경

- 2000년 6월 마이크로소프트가 발표한 플랫폼 독립적인 실행 환경
- 딱 하나의 소프트웨어만 만들고 어느 플랫폼에서나 실행하는 것이 목표이다.



- 분산 환경인 웹의 대중화로 인해 더 중요해졌다.
- C/C++은 소스 차원의 이식성만 지원한다.
- 이진 호환성까지 확보한 진정한 플랫폼 독립 실행 환경은 1995년에 발표된 자바(Java)이다.
- 서로 경쟁 발전하는 관계이다.

닷넷의 역사

- 프로그래밍 언어 순위 4~6위를 유지하고 있다.
- 닷넷과 C#, 비주얼 스튜디오는 각자 업그레이드된다.
- 최신 버전을 쓰기 보다는 안정된 버전을 쓰는 것이 좋다.

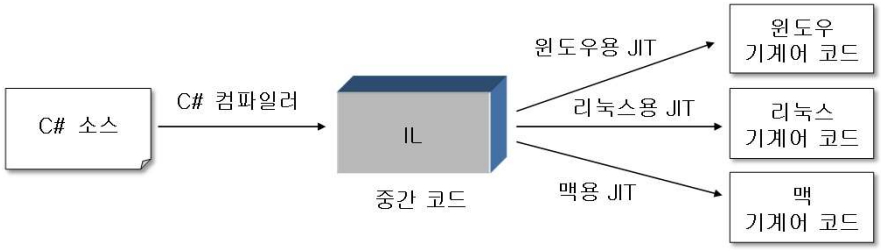
일자	닷넷	C#	주요 기능
2002년 1월	1.0	1.0	초기 버전
2005년 11월	2.0	2.0	제네릭, partial, 널 가능 타입, 익명 메서드, 반복자
2007년 11월	3.0, 3.5	3.0	var, 확장 메서드, 람다, LINQ
2010년 4월	4.0	4.0	선택적 인수, 이름 있는 인수, 지연 바인딩
2012년 8월	4.5	5.0	async, await 비동기 호출
2015년 7월	4.6	6.0	using static, 문자열 보간, 예외 필터, null 조건 연산자
2017년 5월	4.7	7.0~7.3	패턴 매칭, 튜플. 지역 함수, 리터럴 표기법 개선
2019년 4월	4.7.2	8.0	널 가능 참조형, Range, Index, switch 표현식

닷넷 코어

- 플랫폼 독립성은 현재까지는 뻥이었다.
- 2004년 자마린(Xamarin)은 리눅스에 닷넷 프레임워크를 이식하는 모노(Mono) 프로젝트를 발표
- 2016년 마이크로스프트가 자마린을 인수
- 닷넷 코어는 프레임워크의 핵심 부분만 추출하여 CoreFX 라이브러리와 CoreCLR 런타임을 최대한 작게 만든 것이다.
- 모듈식 설계로 모바일 장비에도 부담없이 배포할 수 있고 AOT 컴파일러로 직접 실행 가능한 네이티브 코드를 생성하여 더 빠르다.
- 완전한 플랫폼 독립성은 좀 더 기다려야 한다.
- 풀 버전을 먼저 연구해 두면 닷넷 코어는 쉽게 정복할 수 있다.

플랫폼 독립

- 닷넷 컴파일러는 IL(Intermediate Language) 코드를 생성하고 자바 컴파일러는 바이트 코드를 생성한다.
- 중간 코드를 기계어로 컴파일하는 작업은 JIT(Just In Time) 컴파일러가 담당한다.



- 닷넷 프로그램은 IL로 컴파일되었다가 실행 직전에 플랫폼별 JIT 컴파일러에 의해 실행 가능한 네이티브 코드로 다시 컴파일된다.
- IL을 한 번만 만들어 두면 어디서나 실행할 수 있어 플랫폼 독립성이 달성된다. IL로 컴파일된 코드를 관리 코드(Managed Code)라고 부른다.

플랫폼 독립

- **여기저기서 사용할 수 있는 범용성이 특정 환경에 딱 맞춘 전용에 비해 크기나 성능면에서 좋을 리 없다.**
  - 속도가 느리다. CPU가 직접 실행하지 못하고 번역을 거쳐야 하기 때문에 네이티브 언어보다 느릴 수밖에 없다. 실행속도 뿐만 아니라 기동 속도도 굼떠 반응성이 떨어진다.
  - 단독으로는 실행할 수 없고 프레임워크나 가상 머신을 설치하여 실행 환경을 맞춰야 한다. 런타임의 용량이 거대해 배포하기 번거롭다.
  - 실행에 필요한 요소가 많아 메모리 요구량이 높고 시스템에 부담을 많이 준다. 원활하게 실행하려면 물리적인 하드웨어의 성능이 뒷받침되어야 한다.
- **성능보다 신속한 개발과 편리한 유지 보수, 안정성이 더 중요하다.**

언어별 속도 비교

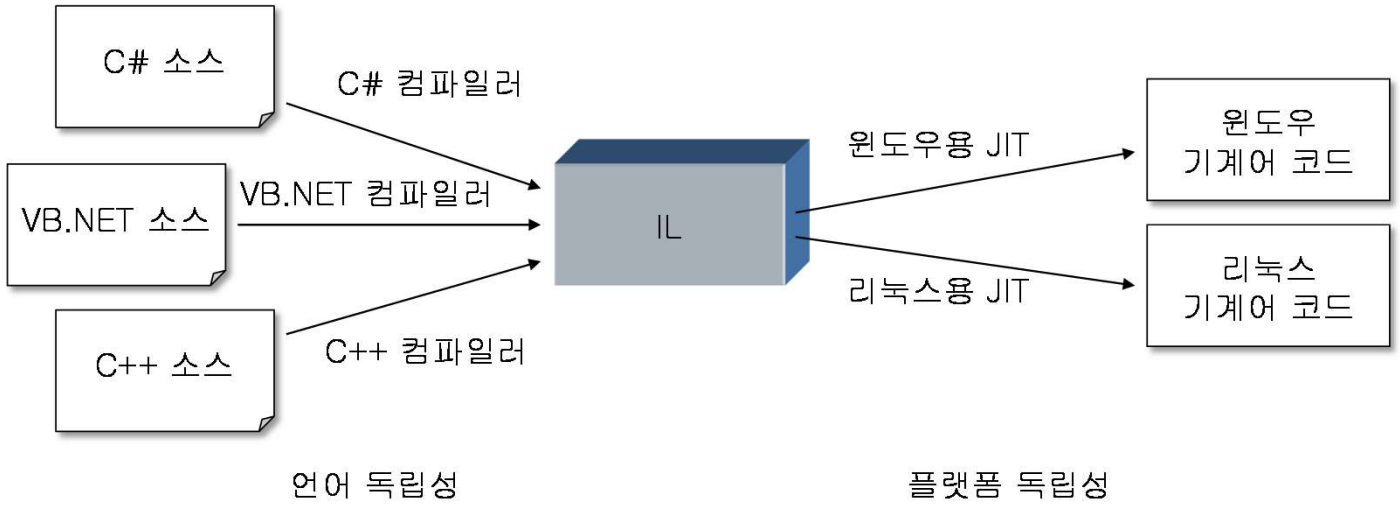
- **30만건의 무작위 자료에 대해 삽입 정렬 수행**

언어	C	C#	자바
1차(2008년)	42초	115초	107초
2차(2019년)	10초	15초	17초

- **컴파일러의 성능이 높아지고 고도의 최적화를 적용하여 1.5배 정도밖에 차이나지 않는다.**
- **파이썬은 2시간 23분이 걸려 대규모 반복 작업에 적합하지 않다.**

언어 독립성

- 문법에 상관없이 중간 코드인 IL만 제대로 만들면 어떤 언어든지 사용할 수 있다.
- 용도에 적합한 언어를 골라 쓸 수 있으며 개발자가 이미 습득한 지식을 계속 활용할 수 있다.
- C#, VB.NET, J#, MC++ 등이 있다.

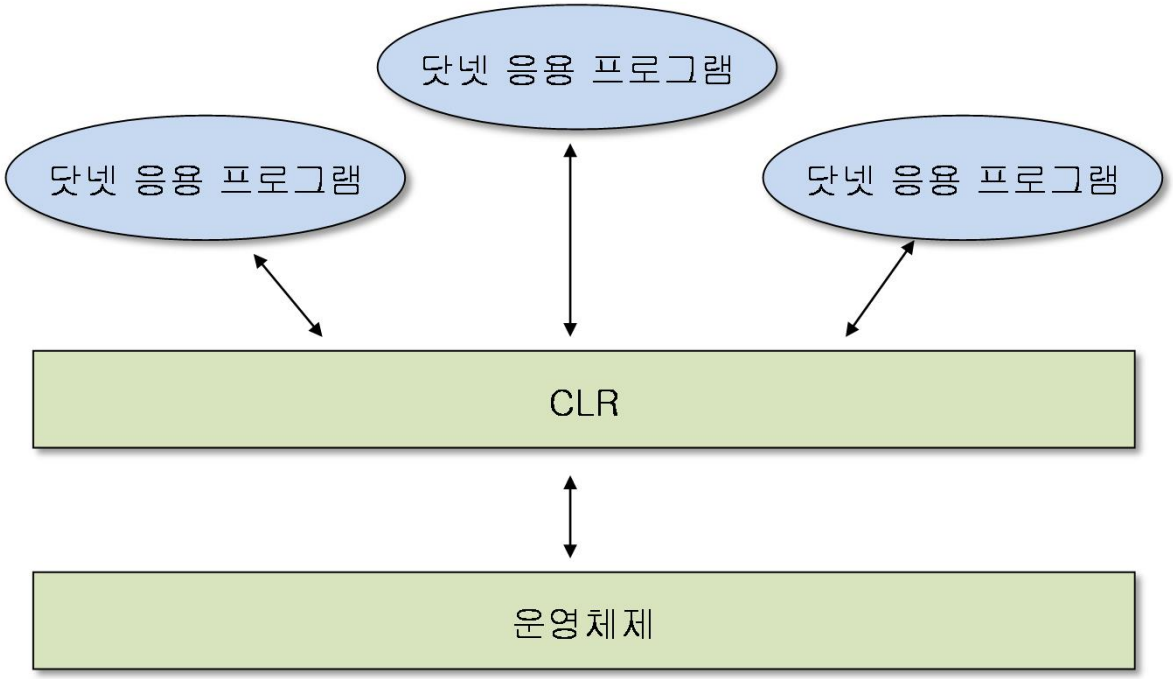


언어 독립성

- **익숙한 언어를 골라 쓸 수 있는 긍정적인 면이 있지만 부작용도 만만치 않다.**
- **초창기에 기존 개발자를 닷넷 진영으로 포섭하기 위해 억지로 급조한 경향이 있다.**
  - 기존 언어의 변형이 심해 다시 배워야 한다.
  - 모든 언어가 닷넷의 요구 사항을 다 충족할 수 없어 언어마다 기능상의 차이가 발생한다.
  - 복수 언어 지원을 위해 닷넷의 시스템 자체가 과도하게 복잡해졌다.
  - 문서 및 예제의 파편화가 심각해 딱 맞는 정보를 구하기 어렵다.
  - 여러 언어로 팀 프로젝트를 수행하면 의사 소통과 개발자 교체가 어렵다.
- **저변 확대를 위한 포석이었고 나름 효과를 봤지만 지금은 오히려 복잡성만 높이는 결과를 초래했다.**

CLR(Common Language Runtime)

- 닷넷의 실행 엔진이며 자바의 가상 머신(JVM)과 개념적으로 유사하다.
- CLR은 운영체제와 닷넷 응용 프로그램 사이에 위치하며 닷넷 프로그램 실행을 위한 모든 서비스를 제공하는 실행 주체
- 응용 프로그램은 운영체제와 직접 통신하지 않으며 CLR을 거친다. CLR은 운영체제를 추상화한다.



CTS(Common Type System)

- 닷넷 언어가 공통으로 사용하는 타입 체계이다.
- 함수의 인수, 리터값의 형태를 상호 알아야 한다.
- C에서 int 형의 값을 넘기면 VB에서 Integer 타입으로 받는다.

- 다른 언어가 정의한 객체를 사용하거나 클래스를 상속할 수 있다.
- 개별 언어가 타입을 제멋대로 정해서는 안되며 플랫폼 차원에서 공통의 타입을 정의해야 한다.
- 모든 언어가 CTS를 다 지원할 의무는 없다.
- 비주얼 베이직은 부호없는 정수형을 지원하지 않는다.

CLS(Common Language Specification)

- **닷넷 언어가 지켜야 할 최소한의 사양**
  - 전역 함수나 변수는 정의할 수 없다.
  - 부호없는 정수형은 인정하지 않는다.
  - 명칭은 대소문자를 구분하지 않는다.
  - 포인터를 사용할 수 없다.
- **최소한의 규정일 뿐이어서 의무적으로 지킬 필요는 없다.**
- **내부적으로 어떤 규칙을 사용해도 무방하며 외부로 공개하는 부분에 대해서만 CLS를 만족하면 된다.**

BCL(Base Class Library)

- 응용 프로그램이 사용하는 여러 가지 기능을 제공하는 클래스 라이브러리이다.
- 자주 사용하는 기능을 플랫폼 제작사가 미리 작성하여 제공한다.
- C의 표준 라이브러리나 윈도우의 API와 개념적으로 비슷하다.
- 닷넷을 배운다는 것은 BCL의 클래스를 하나씩 정복해 나가는 과정이다.

닷넷의 장점

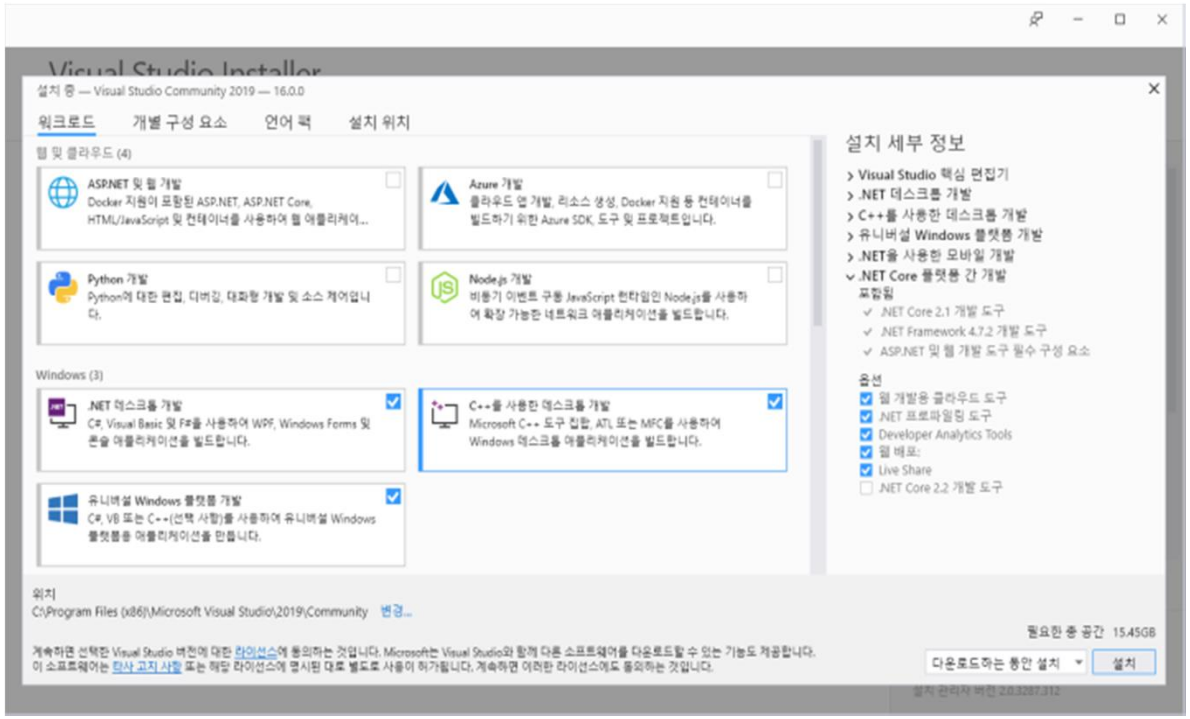
- **닷넷의 가장 큰 특징 및 장점은 플랫폼 독립성, 언어 독립성**
  - 완전한 객체지향 환경이어서 배우기 쉽고 사용하기 쉽다. 생산성이 높고 라이브러리의 지원이 방대해 대규모의 프로그램을 만들기 적합하다.
  - 실행 파일 내부에 메타 데이터가 내장되어 있어 복사만 하면 설치된다. 웹 환경에서는 클릭 한 번으로 설치할 수도 있다. DLL이나 COM에 비해 버전간 충돌 문제가 거의 없다.
  - 어떤 동작을 하는지에 대한 정보가 실행 파일에 들어 있어 부적절한 코드의 실행을 원천적으로 방지한다. 사용자에 따른 역할 기반 방식에 비해 코드 기반의 보안 방식이 더 우수하다.
  - 프로그래밍 환경이 쾌적하다. 통합개발환경이 잘 구성되어 있어 편리하며 가비지 컬렉션 기능을 제공하여 메모리 관리가 쉽다.
  - 지원 범위가 광범위하고 다재다능하다. 응용 프로그램은 물론이고 DB나 네트웍, 웹 프로그램까지 지원하며 모바일에도 쓸 수 있다.

닷넷의 단점과 한계

- **태생적인 단점과 한계도 존재한다.**
  - 중간 코드를 생성하는 한 아무리 최적화를 해도 크고 느린 성능상의 약점은 어쩔 수 없다. 아무리 하드웨어가 발전해도 아직까지 속도가 최우선인 분야가 있다. 비주얼 스튜디오를 닷넷으로 만들 수는 없으며 오피스 또한 마찬가지이다. 성능 한계로 인한 활용 범위의 제약이 존재한다.
  - 단독 실행 파일을 만들 수 없어 닷넷 프레임워크를 먼저 설치해야 하는 부담이 있다. 윈도우에 내장된 프레임워크도 항상 업데이트가 필요하고 PC보다 자원이 빈약한 모바일 환경이나 임베디드 환경에 배포하기에는 너무 거대하다.
  - 쓰기 쉬운 것은 사실이지만 배우기 쉽지 않고 익숙해지기는 더 어렵다. 선수과목이 많고 다양한 기술이 같이 사용되어 초보자가 실무에 쓰기에는 많은 시간이 걸린다. 한 번 배워두면 두루 활용할 수 있지만 그 한 번이 어렵다.
- **대통합은 실패했고 수많은 개발 방법 중의 하나로 성공적으로 자리매김했다.**

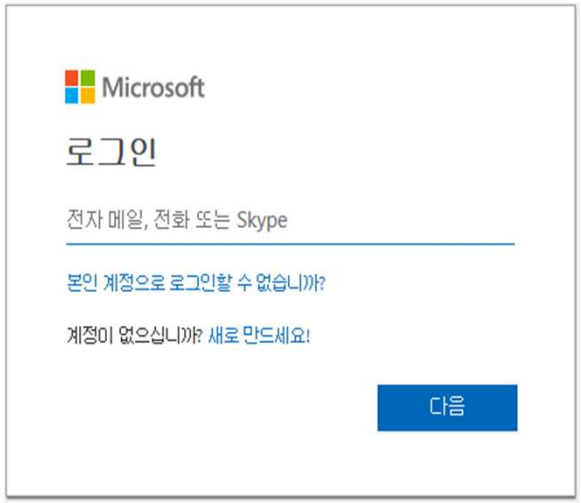
비주얼 스튜디오

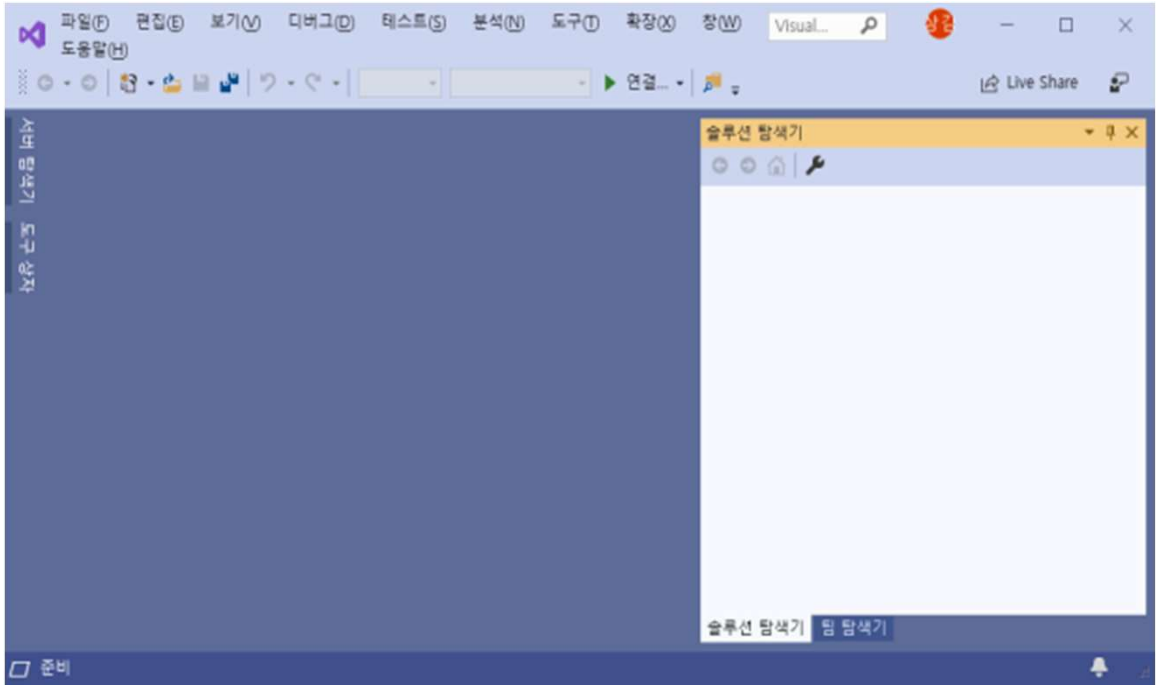
- 다음 주소에서 무료 버전을 다운로드 받는다.
- https://visualstudio.microsoft.com/ko/
- 2019 커뮤니티 버전이면 학습에는 충분하다.
- 필요한 구성요소를 선택한다.



비주얼 스튜디오

- 최초 실행시 로그인한다.
- 무료로 제공하지만 누가 어떤 목적으로 쓰는지 파악하겠다는 의도이다.
- 한번만 로그인하면 된다.





옵션 조정

- **취향에 따라 옵션을 적당히 조정한다.**
  - 텍스트 편집기/모든 언어/CodeLens 페이지에서 CodeLens 옵션을 해제한다.
  - 텍스트 편집기/C#/탭 페이지의 탭 유지 옵션을 선택한다.
  - 텍스트 편집기/C#/ 코드 스타일/서식/줄 추가의 옵션을 조정한다.

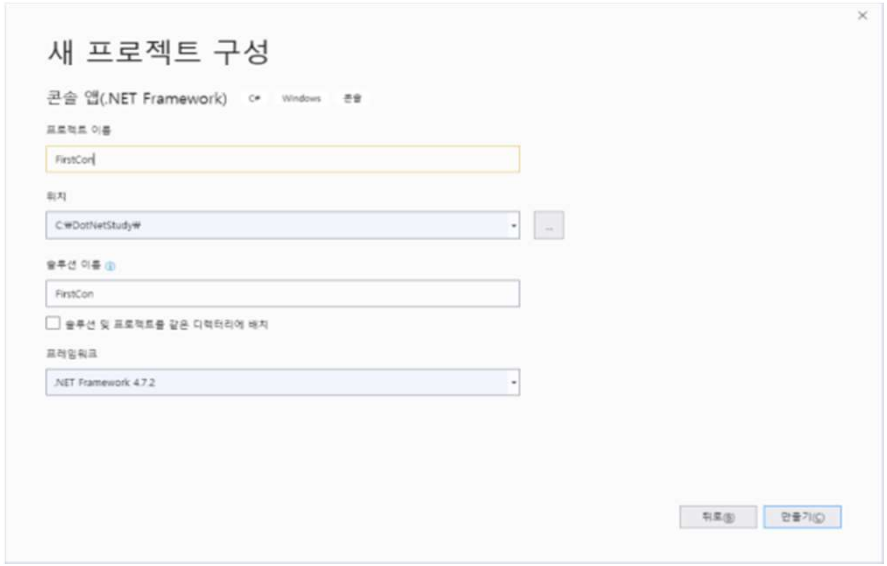
```
class CSTest {
    static void Main() {
        if (...) {
        } else {
        }
    }
}
```

```
class CSTest
{
    static void Main()
    {
        if (...)
        {
        }
        else
        {
        }
    }
}
```

  - 디버그 메뉴의 시작 명령은 단축키를 단일키로 지정한다.
  - 환경/글꼴 및 색 페이지에서 글꼴을 적당히 조정한다.

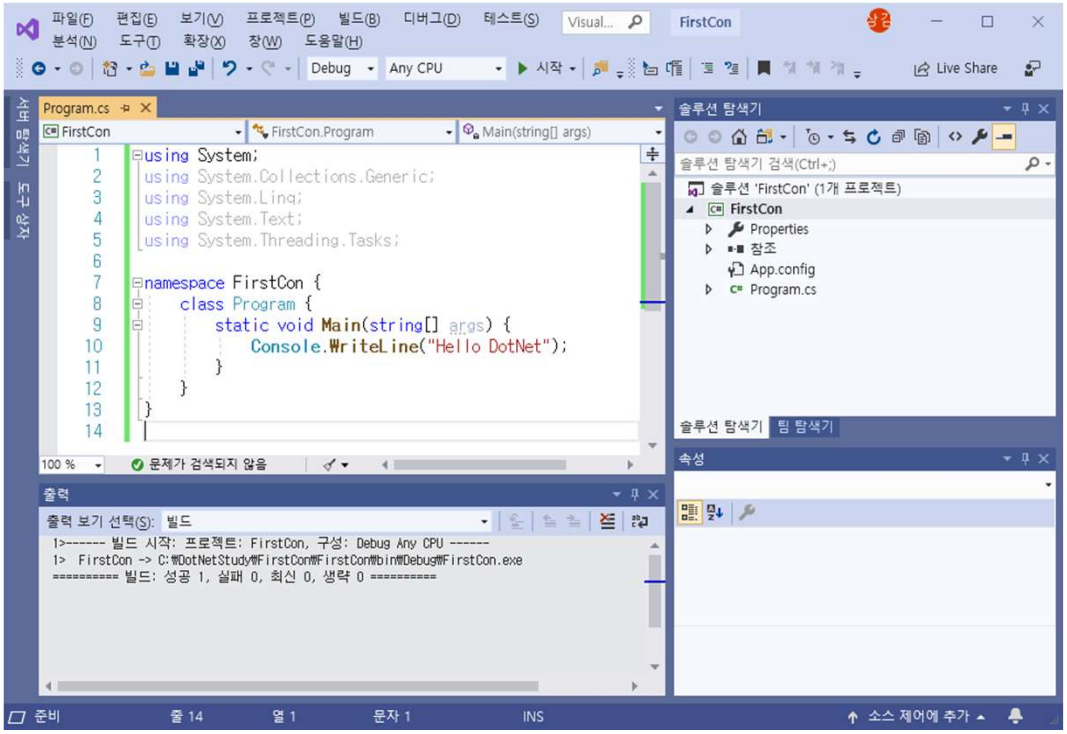
첫 번째 예제

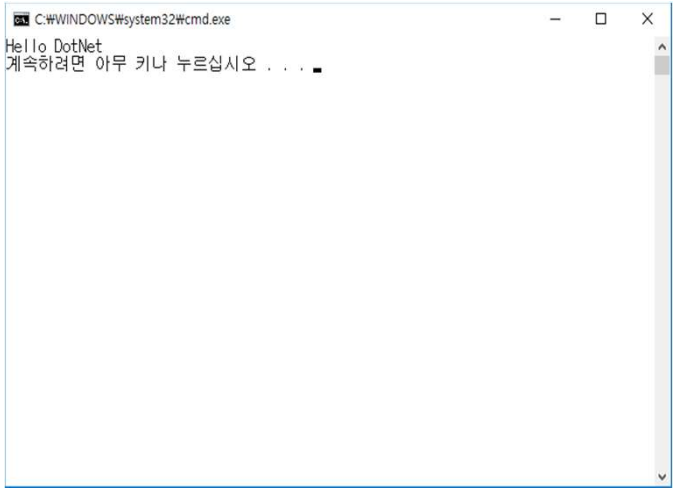
- C:₩DotNetStudy 실습 디렉터리를 만든다.
- 파일/새로 만들기/프로젝트 항목을 선택한다.
- "콘솔 앱(.Net Framework)" 항목을 선택한다.
- 프로젝트 이름을 FirstCon으로 입력한다.
- 프로젝트를 생성하고 기본 소스 파일을 만들어 준다.



첫 번째 예제

- **Main 함수 안쪽에 다음 코드를 작성한다.**
  - Console.WriteLine("Hello DotNet");
- **빌드/솔루션 빌드 메뉴(단축키 F7)를 선택하여 빌드한다.**
- **디버그/디버그하지 않고 시작 메뉴(단축키 Ctrl+F5)를 선택하여 실행한다.**





프로젝트 분석

- using은 시스템 라이브러리를 사용하겠다는 표현
- using System 선언에 의해 System 네임스페이스의 모든 클래스를 자유롭게 사용할 수 있다.
- 네임스페이스는 명칭을 저장하는 영역이며 주로 클래스를 담는다.
- Main은 닷넷 프로그램의 시작점이며 이 메서드에서부터 실행을 시작한다.
- 에러 발생시 코드를 수정한 후 다시 컴파일하면 된다.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FirstCon {
        class Program {
                static void Main(string[] args) {
                        Console.WriteLine("Hello DotNet");
                }
        }
}
```

CSTest 프로젝트

- 프로젝트를 일일이 만드는 것은 너무 귀찮다.
- CSTest 실습 프로젝트를 만들어 두고 이후부터 계속 재사용한다.

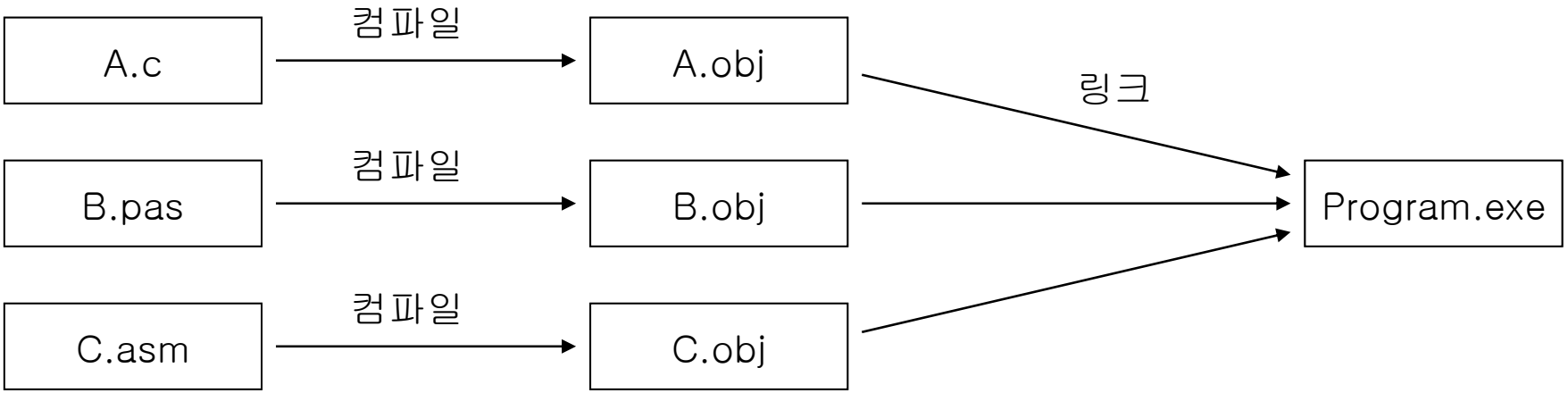
```
using System;

class CSTest {
        static void Main() {
                // 여기에 코드를 작성한다.
        }
}
```

- Main의 안쪽에만 코드를 작성하면 된다.
- 배포 예제 모음집의 코드를 복사한 후 실행한다.

2장. C# 언어

언어 독립성

- 한 프로젝트내에서 여러 언어를 사용할 수 있다.
- 서로의 함수를 호출하거나 상속까지 가능하다.
- 과거에도 오브젝트 파일을 경유한 혼합 프로그래밍이 가능했다.



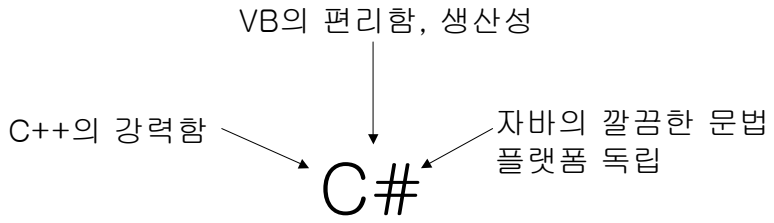
- 가능성일 뿐 아무 언어나 쓸 수는 없고 CLS 규격은 만족해야 한다.
- 닷넷 개발에 가장 적합한 언어는 C#이며 VB.NET 정도만 보조적으로 활용한다.

C#의 특징

- **C++을 참조했음을 숨기지 않는다.**

C ⟹ C++ ⟹ C++++ ⟹ C#

- **여러 언어의 장점을 취합하여 작성했다.**



- **가장 최신인만큼 문법적으로 확실한 우위에 있으며 이전 세대 언어의 모든 시행 착오를 극복했다.**
- **문법 구조가 가장 발전된 형태임은 자타가 공인하는 사실이다.**

C#의 특징

- **C++과는 다른 특징이 있다.**
  - 완전한 객체지향 언어이다. 전역 함수나 변수를 만들 수 없으며 모든 것은 클래스에 소속된다.
  - 다중 상속이나 포인터같은 복잡한 기능을 과감하게 빼 버려 단순해졌다.
  - 가비지 컬렉션 기능이 있어 메모리를 수동으로 관리할 필요가 없다.
  - 타입 체크와 문법이 엄격해져 실수를 방지한다. 정수형과 논리형이 분리되었으며 초기화하지 않은 변수는 에러로 처리한다. 배열 범위 초과와 오버플로우도 예외로 처리한다.
- **C++에 비해 부족한 면도 있다.**
  - 안정성에 중점을 두어 비트필드, 공용체가 없다.
  - 다중 상속을 지원하지 않는다.
  - 전처리기가 없다.
- **C# 설계자는 앤더스 헤일스버그(Anders Hejlsberg:덴마크)이며 그래서 델파이와 유사하다.**

구성요소

- **소스를 구성하는 요소에는 키워드, 명칭, 연산자, 구두점, 상수, 주석, 공백 등 7가지가 있다.**
- **키워드는 C++, 자바와 유사하다.**
  - abstract as base bool break byte case....
- **한줄 주석인 //와 블록 주석인 /* */가 있고 XML 형식의 ///가 있다.**
- **명칭은 자유롭게 이름을 붙이되 다음 규칙을 준수한다.**
  - 키워드는 쓸 수 없다.
  - 대소문자를 구분한다.
  - 영문, 숫자, _로 구성하되 첫 문자로 숫자는 쓸 수 없다.
- **@ 문자를 명칭에 사용할 수 있다. 유니코드를 지원하여 한글 명칭도 가능하다.**
- **국제화 시대에 바람직하지는 않다.**

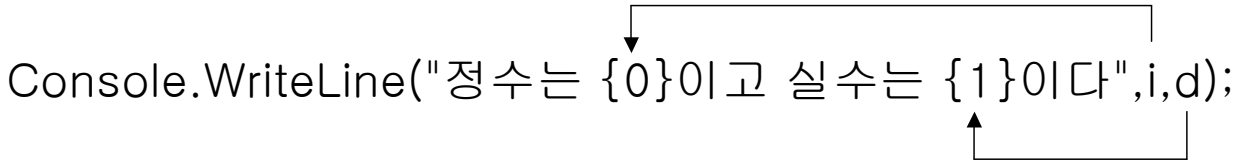
```
int 제어변수;
int 합계 = 0;
for (제어변수 = 1; 제어변수 <= 100; 제어변수
++) {
        합계 = 합계 + 제어변수;
}
Console.WriteLine(합계);
```

출력

- **실습을 위해 가장 먼저 배워야 한다.**
- **System.Console의 WriteLine 메서드로 출력하며 대부분의 타입을 지원한다.**
  - public static void WriteLine();
  - public static void WriteLine(int value);
  - public static void WriteLine(double value);
  - public static void WriteLine(string value);
  - public static void WriteLine(string format, params object[] arg);
- **값을 출력한 후 자동으로 개행한다.**
- **개행하지 않고 여러 개의 변수를 한 줄에 출력하려면 Write 메서드를 사용한다.**

여러 변수값 출력

- **여러 개의 변수를 출력할 때는 문자열과 + 연산자를 사용하여 연결한다. 또는 서식화하여 출력한다.**
  - Console.WriteLine("정수는 " + i + "이고 실수는 " + d + "이다.");
  - Console.WriteLine("정수는 {0}이고 실수는 {1}이다.", i, d);
- **서식은 { } 괄호 안에 0부터 시작하는 인덱스를 적고 문자열 뒤에 서식의 개수만큼 실제 출력할 값을 나열한다.**



- **{ 인덱스, 폭 : 형식 } 으로 폭, 정렬 방식, 정밀도를 지정한다.**

{0}	{0,6}	{0,-6}
->123<-	-> 123<-	->123 <-
4칸 차지	6칸 차지, 오른쪽 정렬	6칸 차지, 왼쪽 정렬

문자열 보간

- **서식 조립은 불편하고 위험한 면이 있다.**
- **C# 6.0은 문자열 보간(Interpolation) 출력 방법을 추가했다.**
- **문자열 앞에 $기호를 쓰고 { } 괄호안에 변수명을 적는다.**
  - Console.WriteLine($"안녕하세요. {age}세 {name}입니다.");
  - 안녕하세요. 18세 김한슬입니다.
- **컴파일러가 변수의 값을 문자열 안에 넣어 준다.**
- **{ } 괄호안에 모든 서식을 적용할 수 있고 연산식이나 메서드 호출도 가능하다. 오타나 잘못된 구문은 에러 처리된다.**
- **{ } 괄호 자체는 {{ }}로 두번 적는다.**

입력

- **다음 메서드로 문자열을 입력받는다.**
  - public static string ReadLine ()
- **정수는 문자열로 입력받은 후 변환해서 사용한다.**
  - Age = Convert.ToInt32(Console.ReadLine());
- **키 자체를 입력받을 때는 다음 메서드를 호출한다.**
  - public static ConsoleKeyInfo ReadKey([bool intercept]);
- **문자가 아닌 기능키도 입력받을 수 있다.**
- **입력받은 키 정보를 ConsoleKeyInfo 구조체로 리턴하며 Key 멤버에 사용자가 누른 키의 이름이 전달된다.**
- **방향 키 : LeftArrow, RightArrow, UpArrow, DownArrow**

콘솔 관리

- 윈도우의 콘솔은 전통적인 콘솔에 비해 기능이 확장되어 있다.
- 색상을 바꾸거나 커서 위치를 옮길 수 있다.
- 그래픽을 출력할 수 없어 실습용으로만 콘솔 환경을 사용한다.

멤버	설명
Title	콘솔창의 제목 문자열이다.
BackgroundColor, ForegroundColor	전경색, 배경색의 색상이다.
CursorSize	커서의 높이를 지정한다.
CursorVisible	커서의 보임/숨김을 지정한다.
CursorLeft, CursorTop	커서의 현재 위치이다.
Clear()	화면을 지운다.
Beep()	삑 소리를 낸다.
ResetColor()	디폴트 색상으로 변경한다.
SetCursorPosition(x,y)	커서의 위치를 옮긴다.

명칭의 충돌

- **공용 라이브러리인 BCL은 거대하고 복잡하다.**
- **방대한 라이브러리를 효율적으로 관리하기 위해 네임스페이스로 구역을 나누어 놓았다.**
- **같은 범위에서 명칭이 충돌해서는 안된다.**
  - class MyClass { 어쩌고 저쩌고 }
  - class MyClass { 이러쿵 저러쿵 }
- **다른 범위에서는 같은 이름을 사용해도 상관없다.**
  - class SomeClass {
  - int value;
  - }
  - class OtherClass {
  - double value;
  - }
- **클래스 이름도 충돌 가능하다. 근본적인 충돌 방지 대책이 필요하다.**

네임스페이스

- **네임스페이스는 명칭을 저장하는 장소이며 명칭의 범위를 격리시킨다.**
  - namespace A {
  - class MyClass { ... }
  - }
  - namespace B {
  - class MyClass { ... }
  - }
- **네임스페이스없이 외부에 선언하면 기본 네임스페이스에 소속되며 명칭만으로 참조한다.**
- **네임스페이스끼리 중첩 가능하며 이 경우 A.B.MyCalss로 참조한다.**
  - namespace A {
  - namespace B {
  - class MyClass { ... }
  - }
  - }

using

- 네임스페이스를 잘 활용하면 명칭을 체계적으로 관리할 수 있고 누가 만든 명칭인지 금방 알 수 있다.
- 구분은 되지만 참조문이 길다.
  - SoenSoft.MainTeam.KimSangHyung.MyClass

```
namespace SoenSoft {
        namespace MainTeam {
                namespace KimSangHyung {
                        class MyClass { ... }
                }
                namespace ParkDaeHee { ... }
        }

        namespace NetworkTeam { .... }
        namespace GraphicTeam { .... }
}
```

- 입력하기 번거롭고 오타 가능성이 높다.
- 긴 참조문을 짧게 쓸 수 있는 using 문장이 제공된다.
  - using SoenSoft.MainTeam.KimSangHyung;
- 컴파일러는 지정한 네임스페이스를 자동 검색하여 MyClass 명칭을 바로 사용할 수 있다.
- using System; 문에 의해 System의 모든 명칭을 바로 사용할 수 있다.

using static

- 클래스에 소속된 정적 멤버를 모두 임포트하여 메서드 이름만으로 호출할 수 있다.

```
using System;
using static System.Console;

class CSTest {
        static void Main() {
                System.Console.WriteLine("안녕하세요.");
                Console.WriteLine("안녕하세요.");
                WriteLine("안녕하세요.");
        }
}
```

- using static System.Console; 문장에 의해 WriteLine만으로 호출 가능하다.
- 상수 멤버와 열거형에도 사용할 수 있다.

using문의 부작용

- **using은 어디까지나 소속을 찾는 약식 방법일 뿐이어서 완벽하지는 않다.**
- **양쪽을 모두 using 선언하면 MyClass 참조문이 모호해진다.**
- **한쪽의 using 선언을 없애든가 아니면 명칭앞에 정확한 네임스페이스를 밝혀야 한다.**
- **네임스페이스에 별명을 부여한다.**
  - using 별명 = 네임스페이스;
  - using SMK = SoenSoft.MainTeam.KimSangHyung;
- **SMK.MyClass 형식으로 짧게 참조할 수 있다.**

```
using A;
using B;

namespace A {
        class MyClass { int i; }
}
namespace B {
        class MyClass { double d; }
}

class CSTest {
        public static void Main() {
                MyClass objA = new MyClass();
                MyClass objB = new MyClass();
        }
}
```

BCL의 구조

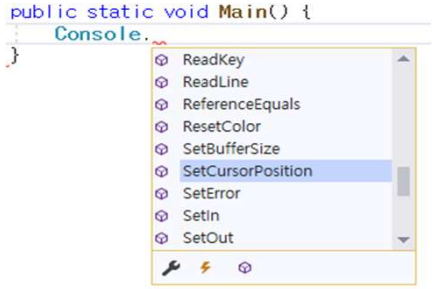
- 네임스페이스는 명칭 충돌 방지 외에 클래스를 기능별로 분류하는 역할도 한다.
- 클래스를 기능에 따라 분류하여 네임스페이스에 체계적으로 정리해 놓았다. 클래스의 소속을 알아 두어야 한다.

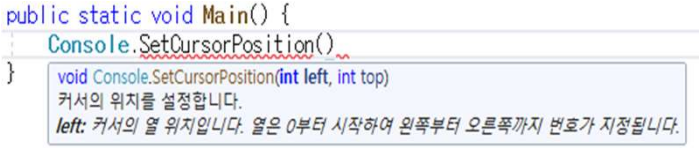
네임스페이스	설명
System	타입, 메모리 관리 등 핵심 클래스들
System.Collections	배열, 연결 리스트 등의 컬렉션 클래스
System.IO	파일 입출력 및 네트워크 관련 클래스
System.Windows.Forms	윈도우 폼과 컨트롤
System.Drawing	GDI+
System.Web	웹 개발에 관련된 클래스
System.Xml	XML 관련 클래스들
System.Security	보안, 암호, 권한 관련 클래스

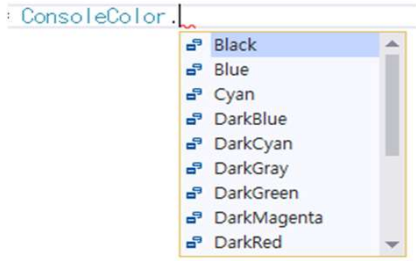
- 사용할 클래스의 네임스페이스를 using 선언해야 한다.
- C# 소스는 항상 using System; 으로 시작한다.

도움말 얻기

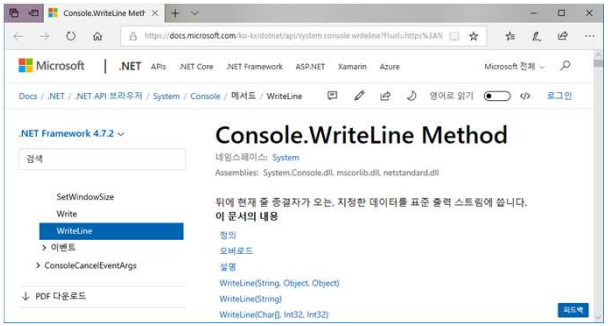
- 기본 도움말은 MSDN이며 웹에 공개되어 있다.
- 인텔리센스 기능으로 편집기에서 정보를 바로 볼 수 있다.
- 멤버 목록, 함수 도움말 등을 즉시 보여 준다.





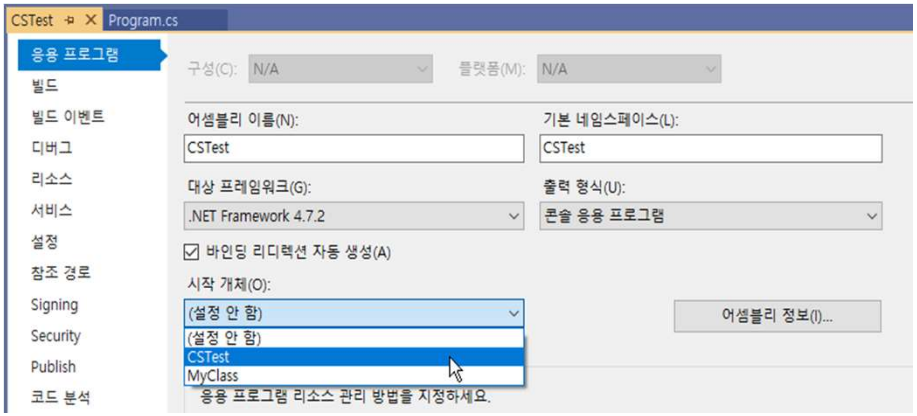


- 알고 싶은 명칭에 대해 F1키를
- 누르면 MSDN을 열어 준다.



Main 메서드

- 닷넷 프로그램의 진입점은 Main 메서드이다.
- 객체가 생성되기 전에 먼저 호출되어야 하므로 static이어야 하며 외부에서 호출되므로 public 액세스 지정을 가진다.
- 인수와 리턴값 유무에 따라 4가지 원형이 있다.
  - public static void Main();
  - public static int Main();
  - public static void Main(string[] args);
  - public static int Main(string[] args);
- 여러 개의 Main이 각 클래스에 있을 때는 시작 객체를 지정한다.



명령행 컴파일러

- **명령행에서도 닷넷 개발을 할 수 있다.**
  - C:₩DotNetStudy>csc Program.cs
  - Microsoft (R) Visual C# 컴파일러 버전 3.0.19.17001 (1deafee3)
  - Copyright (C) Microsoft Corporation. All rights reserved.

  - C:₩DotNetStudy>program
  - Hello DotNet
- **스크립트로 빌드를 자동화할 수 있는 이점이 있지만 불편하다.**
- **통합 개발 환경이 있으므로 이 방법을 굳이 쓸 필요는 없다.**