


# 01. 프로그래밍 언어의 역사

## ▶ C#의 역사

- ▶ C : 1972, Dennis Richie(데니스 리치) (AT/T) ,  
구조화된 Programming, 전역변수 관리 힘들다.
- ▶ C++ : 1983, Bjarne Stroustrup(브얀 스트로스트럽)(AT/T) ,  
객체중심의 언어 -> C 언어와 기본문법 유사,  
OOP 대중화 된 언어 , small talk(최초의 OOP)
- ▶ Java : 1995 , James Gosling(제임스 고스링)(Sun),  
플랫폼에 의존적이지 않은 언어(O/S 독립적인 언어)
- ▶ C# : 2002, Anders Hejlsberg(엔더스 헤즐스버그)(MS) ,  
JAVA 처럼 문법이 간결하면 서도 JAVA가 갖지 않는 기능을 추가.  
JAVA와 60% 유사. Component를 만들 때 편리하다.

# 01. .NET 개발환경

- ▶ Visual studio -> MS가 제공하는 개발도구(유로) , 통합개발 환경(편집, 빌드, 실행, 디버깅)
- ▶ .NET Framework(Common Language Runtime , .Net Framework class library)  
.NET은 응용프로그램을 개발(compile)하고 실행(run)하는데 필요한 환경  
IIS,WMI등과 같은 Middle ware를 필요로 한다.
-  CLR(Common Language Runtime) - java **Virtual Machine**과 같은 개념.  
IL Code에서 Machine Language가 되려면 CRL이 있어야 된다.  
응용프로그램을 개발 단순화,강건한 실행환경, 다양한 언어지원, 배포를 간단하게 지원한다.
- ▶ Net Framework Class library(BCL:Basic Class Library)  
이전 기술은 필요없다. 모든 라이브러리를 새규칙에 따라 정리되었으며,  
.NET Framework 라이브러리로 통일했다. .NET Class Library - 약 10000여개 준비,API

## C# 버전

C#은 .NET Framework 버전 및 Visual Studio 버전과 밀접한 관련이 있으며, 다음 도표에서 각 버전별 연관성을 살펴 볼 수 있다.

.NET 버전	C# 버전	Visual Studio
.NET 1.0	C# 1.0	Visual Studio .NET
.NET 1.1	C# 1.1	Visual Studio .NET 2003
.NET 2.0	C# 2.0	Visual Studio 2005
.NET 3.0	C# 2.0	Visual Studio 2005 Extensions
.NET 3.5	C# 3.0	Visual Studio 2008
.NET 4.0	C# 4.0	Visual Studio 2010
.NET 4.5	C# 5.0	Visual Studio 2012 Visual Studio 2013
.NET 4.6	C# 6.0	Visual Studio 2015
.NET 4.7	C# 7.0	Visual Studio 2017

- Visual Studio 2010은 .NET 4.0 뿐만 아니라 .NET 2.0, .NET 3.0, .NET 3.5도 지원한다.
- Visual Studio 2008은 .NET 3.5 뿐만 아니라 .NET 2.0, .NET 3.0도 지원한다.

## C# 버전별 주요 기능

다음은 C# 버전별로 새로 추가된 주요 기능들을 요약한 것이다.

C# 버전	주요 기능
C# 2.0	<ul style="list-style-type: none"><li>• <u>C# Generics</u></li><li>• <u>Anonymous Method (무명 메서드)</u></li><li>• <u>Nullable Type</u></li><li>• <u>Partial Type</u></li><li>• <u>C# yield 키워드</u></li><li>• Delegate 에 대한 Covariance / Contravariance</li></ul>
C# 3.0	<ul style="list-style-type: none"><li>• <u>Lambda Expression (람다식)</u></li><li>• <u>Anonymous Type (익명 타입)</u></li><li>• <u>Extension Method (확장 메서드)</u></li><li>• <u>C# var 키워드 (implicit type)</u></li><li>• <u>LINQ</u></li><li>• Expression Tree</li></ul>
C# 4.0	<ul style="list-style-type: none"><li>• <u>C# dynamic (Late binding)</u></li><li>• <u>Named Argument</u></li><li>• <u>Optional Argument</u></li><li>• <u>Indexed Property</u></li><li>• 보다 쉬운 Office COM API 지원</li></ul>

## C# 버전별 주요 기능

C# 5.0↗	<ul style="list-style-type: none"><li>• <u>C# async / await</u>↗</li><li>• Caller Information↗</li></ul>
C# 6.0↗	<ul style="list-style-type: none"><li>• <u>널 조건 연산자 (Null-conditional operator)</u>↗</li><li>• <u>문자열 내삽(內挿) 기능 (String Interpolation)</u>↗</li><li>• <u>Dictionary Initializer</u>↗</li><li>• <u>nameof 연산자</u>↗</li><li>• <u>using static 문</u>↗</li><li>• <u>catch/finally 블록에서 await 사용</u>↗</li><li>• <u>Exception Filter 지원</u>↗</li><li>• <u>자동 속성 초기자 (Auto-Property Initializer)</u>↗</li><li>• <u>읽기전용 자동 속성 (Getter only)</u>↗</li><li>• <u>Expression-bodied member 사용</u>↗</li></ul>

# 01. HELLO, WORLD! (5/6)

## ▶ HelloWorld 프로젝트 생성

### ▶ 6. 다음의 코드를 Program1.cs에 입력

```
using System;

namespace HelloWorld
{
    class Class1
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World.");
        }
    }
}
```

# 01. HELLO, WORLD! (6/6)

## ▶ Main () 메서드

CLR(Common Language Runtime) 에 의해서 최초로 호출되는 메서드이다. 메서드의 첫문자는 항상 대문자이다. (진입점 함수), 접근자 어떤것도 상관 없다. 자신이 호출안하고 CLR이 호출.

static 은 자신이 속해있는 class에 개체를 생성하지 않고도 호출될 수 있다. (소속이 없는 함수), 메인 메서드는 반드시 필요하다.

## ▶ Main 메서드의 return type은 void, int 만 가능하다.(CLR, Common Language Runtime 이 받아간다) .

인자는 String만 올 수 있다. 없으면 비워놓는다.

(인수가 없을 때 void 사 용할 수 없다.)

배열은 변수명 반드시 왼쪽에 와야 한다.

## 02. 첫 번째 프로그램 뜯어보기 (1/6)

### ▶ using System;

```
01 using System;  
02  
03 namespace HelloWorld  
04 {
```

- ▶ using System;은 한 덩어리 같지만 실은 세 가지 요소로 구성.
  - ▶ using : 네임스페이스를 사용하겠다고 선언하는 키워드
  - ▶ System : 숫자, 텍스트와 같은 데이터를 다룰 수 있는 기본적인 데이터 처리 클래스를 비롯하여 C# 코드가 기본적으로 필요로 하는 클래스를 담고 있는 네임스페이스
  - ▶ 세미콜론(;) : 컴파일러에게 문장의 끝을 알리는 기호



## 02. 첫 번째 프로그램 뜯어보기 (2/6)

### ▶ namespace HelloWorld{ }

```
03 namespace HelloWorld
04 {
05     class Class1
06     {
07     ...
08     }
09 }
```

- ▶ 네임스페이스는 성격이나 하는 일이 비슷한 클래스, 구조체, 인터페이스, 델리게이트, 열거 형식 등을 하나의 이름 아래 묶는 역할을 수행
  - ▶ 예) System.IO 네임스페이스에는 파일 입출력을 다루는 각종 클래스, 구조체, 델리게이트, 열거 형식들이 있고, System.Printing 네임스페이스에는 인쇄에 관련된 일을 하는 클래스 등등이 소속
  - ▶ .NET 프레임워크 라이브러리에 1만 개가 훨씬 넘는 클래스들이 있어도 프로그래머들이 전혀 혼란을 느끼지 않고 이들 클래스를 사용할 수 있는 비결임

## 02. 첫 번째 프로그램 뜯어보기 (3/6)

▶ `class ClassI{ }`

- ▶ `ClassI` 라는 클래스를 선언
- ▶ **클래스**는 C# 프로그램을 구성하는 **기본 단위**
- ▶ 클래스는 “데이터 + 메소드”로 이루어짐
- ▶ C# 프로그램은 최소 하나 이상의 클래스로 이루어지며 수백,수천의 클래스로 구성되기도 함

```
05 class ClassI
06 {
07     // 프로그램 실행이 시작되는 곳
08     static void Main(string[] args)
09     {
10         Console.WriteLine("Hello,World!");
11     }
12 }
```

## 02. 첫 번째 프로그램 뜯어보기 (4/6)

▶ `static void Main( string[] args ) { }` (1/2)

- ▶ 프로그램의 **진입점**(Entry Point)으로써, 프로그램을 시작하면 실행되고, 이 **메소드가 종료되면 프로그램도 역시 종료**
- ▶ 모든 프로그램은 반드시 Main이라는 이름을 가진 메소드를 가지고 있어야 함

```
05 class Class1
06 {
07     // 프로그램 실행이 시작되는 곳
08     static void Main(string[] args)
09     {
10         Console.WriteLine("Hello, World!");
11     }
12 }
```

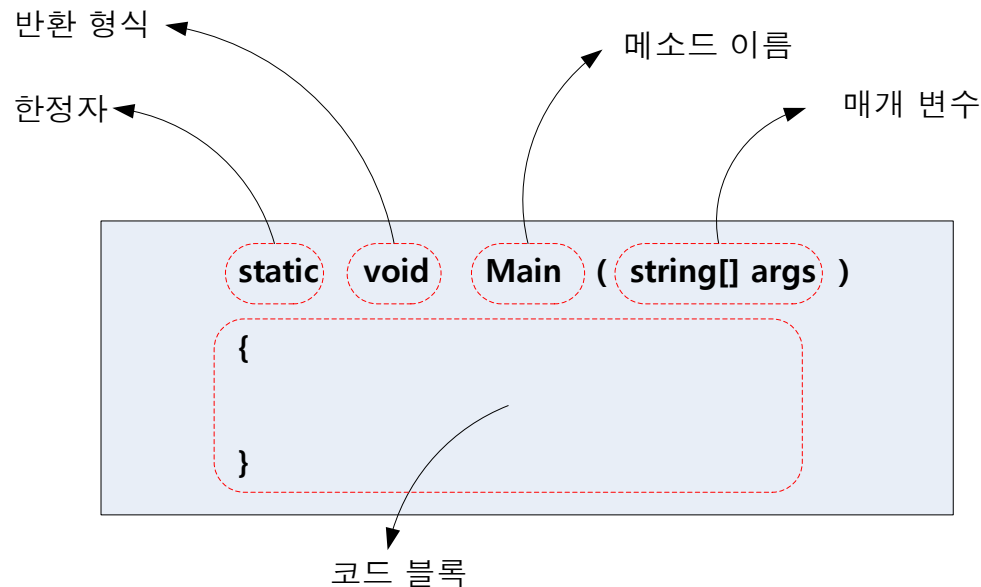
## 02. 첫 번째 프로그램 뜯어보기 (5/6)

<cmd>

C:#User> ch10.exe simple a.txt

▶ `static void Main( string[] args ) { }` (2/2)

- ▶ 한편, 메소드는 C 프로그래밍 언어에서는 함수(Function)라 불림
  - ▶ 함수는 입력을 받아 계산한 후, 출력함
- ▶ 다음은 Main() 메소드를 이루는 각 구성요소의 명칭을 나타냄



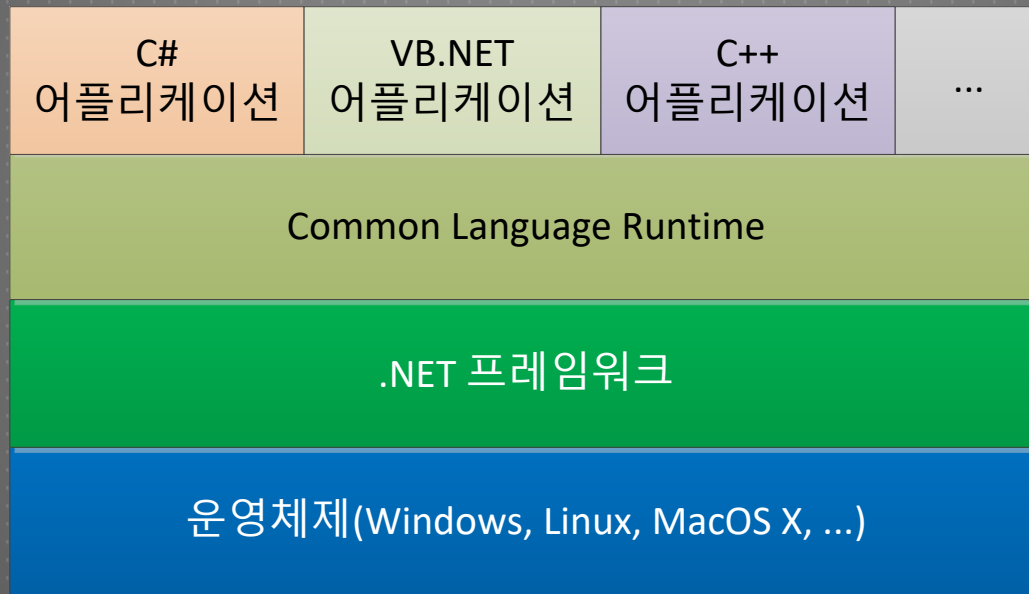
## 02. MAIN() 메서드가 여러 개

- ▶ 클래스가 다를 때 Main 메서드는 여러 개 지정 가능하며 이 때 시작 Main 메서드를 지정해야 한다.  
(같은 클래스에서는 Main 메서드 하나는 반드시 필요함)
- ▶ project명 선택-속성-시작개체 에서 main() 메서드를 시작할 클래스를 선택

```
class Program1
{
    static void Main(string[] args)
    {
        Console.WriteLine("program1 Main() class.");
    }
}
class Program2
{
    static void Main(string[] args)
    {
        Console.WriteLine("program2 Main() class.");
    }
}
```

## 03. CLR에 대하여 (1/3)

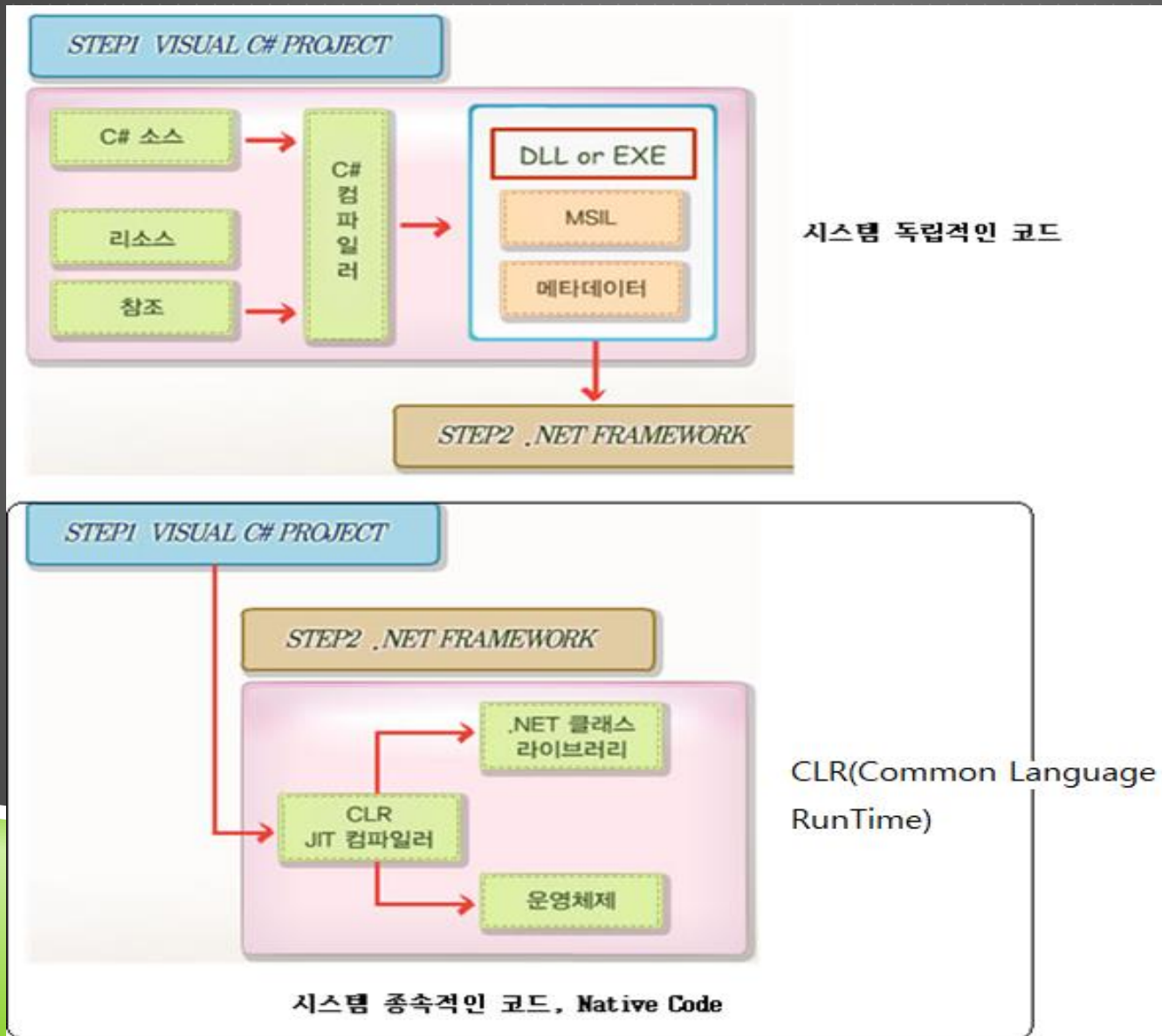
- ▶ C# 언어로 작성한 모든 프로그램은 CLR(Common Language Runtime) 위에서 동작함



## 03. CLR에 대하여 (2/3)

- ▶ CLR은 .NET 프레임워크와 함께 OS 위에 설치
- ▶ 네이티브 코드로 작성되어 있는 프로그램들은 운영체제가 직접 실행할 수 있지만 **C# 컴파일러가 만들어 낸 실행 파일**은 운영체제가 이해할 수 없는 코드로 구성되어 있기 때문에 실행 불가
- ▶ C# 컴파일러는 C# 소스 코드를 읽어서 **IL(Intermediate Language)**이라는 중간 언어로 작성된 실행 파일을 생성
  - ▶ 사용자가 이 파일을 실행시키면 **CLR이 실행 파일 내의 중간 코드를 읽어들이** 다시 OS가 이해할 수 있는 **네이티브 코드로 컴파일** 후 실행
  - ▶ 이것을 **JIT(Just-In-Time) 컴파일** 또는 적시 컴파일이라고 부름
- ▶ JIT 컴파일은 실행시에 이루어지는 컴파일 비용을 요구하지만, **플랫폼에 최적화된 코드를 만들어낸다는 장점**이 있음.

## C# 컴파일 과정 (3/3)





## 04 NAMESPACE, CLASS, METHOD 구분하기

- ▶ **namespace** : 성격이나 하는 일이 비슷한 클래스, 구조체, 인터페이스 등을 하나의 이름으로 묶어서 관리하는 개념.
- ▶ **class** : C# 프로그램을 구성하는 기본 단위로 “데이터 + 메소드”로 이루어지며, 여러 개의 클래스를 가질 수 있다.
- ▶ **method** : 어떤 일을 수행하는 하나의 실행단위.
- ▶ **partial 클래스** : 클래스 정의를 여러 파일로 분할할 수 있다.  
대규모 프로젝트에서 작업하는 경우 클래스를 개별 파일에 분산하면 여러 프로그래머가 동시에 클래스에 대해 작업할 수 있다.  
클래스 정의를 분할하려면 다음과 같이 **partial** 키워드 한정자를 사용한다.

## 05 C#의 구성요소

NameSpace	
	<b>Type</b> <ul style="list-style-type: none"><li>1. class</li><li>2. struct</li><li>3. enum</li><li>4. interface</li><li>5. delegate</li></ul>
	<b>Member</b> <ul style="list-style-type: none"><li>1. Fields</li><li>2. Method</li><li>3. Property</li><li>4. Event</li></ul>

**\*\* type의 종류**

. class : 한 application은 여러 개의 파일로 구성될 수 있지만, 하나의 class가 여러 개의 파일로 쪼갤 수 없다.  
class 파일과 Source파일의 이름이 같을 필요가 없다.  
결과파일은 프로젝트를 기준으로 한다.

. struct  
. enum  
. interface  
. delegate

**\*\* member의 종류**

Field, Method, Property, Event