프로그래밍 기초 with C#

01. Hello, World

학습목표

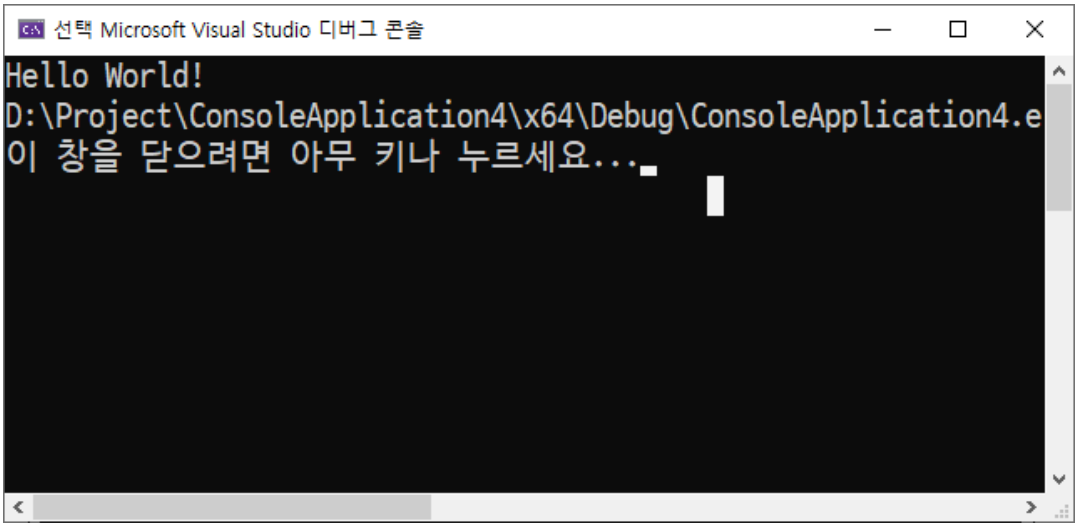
- C#의 구성 요소 중 주석, 클래스, 구문, 이름 공간 등이 무엇인지 알 수 있다.
- C#의 표준 클래스 라이브러리가 무엇인지 이해할 수 있다.
- 컴파일 오류가 무엇인지 알고 이를 고칠 수 있다.

Hello, World

Hello, World 프로그램은 가장 기본적인 프로그램이다. 아래의 코드를 작성해보자.

```
class Program
{
  // 이것은 주석입니다.
  static void Main()
  {
    System.Console.WriteLine("Hello World!");
  }
}
```

코드를 실행 가능한 프로그램으로 만드는 과정을 **빌드**(Build)라고 했다. 빌드를 한 뒤 실행하면 아래와 같은 창을 확인할 수 있다.



이러한 창을 콘솔(Console)이라고 한다.

프로그램의 명령어는 위에서 아래로 순차적으로 읽어나가면 된다. 각각이 무얼 뜻하는지 가볍게 알아보도록 하자. 상세한 것은 나중에 다시 배우게 된다.

클래스

```
class Program { }
```

프로그래밍 언어에는 객체, 이름 공간, 인터페이스 등등 여러 가지 구성 요소가 존재한다. 이것은 Program이라는 이름을 가진 **클래스**(Class)인데, C# 프로그램을 구성하는 기본 단위로 데이터와 기능을 묶기 위해서 사용한다. {}에는 클래스를 구성하는 하위 요소인 **멤버**(Member)가 작성된다.

주석

주석(Comment)은 프로그래머만을 위한 기능으로 코드에 대한 정보를 남길 때 사용한다. 주석은 프로그램에 아무런 영향을 주지 않으며, 실제로 빌드 될 땐 모두 빈 칸으로 바뀐다. 주석에 대한 문법은 아래와 같다.

```
// 이것은 주석입니다.
/*
이렇게 쓴다면
여러 줄의 주석을 작성할 수 있습니다.
```

```
*/
```

올바른 주석은 코드를 설명하는 것이 아니라 코드의 문맥을 설명한다. 다시 말해 왜 이 코드를 작성하게 됐는지를 설명한다.

```
System.Console.WriteLine("Hello, World"); // Bad : Hello, World를 출력합니다.
```

예를 들어 위와 같은 주석은 쓸모가 없다. 이미 코드 자체로 어떤 동작을 수행하는지 나타나기 때문이다. 코드의 동작을 설명하기 위한 주석이 필요하다면 코드를 재작성할 필요가 있다는 의미다.*

* 여기서 좋은 주석을 작성하기 위한 규칙을 제공하고 있다. 참고하자.

주석과 관련된 유명한 명언이 있다. “나쁜 코드를 설명하지 마라. 다시 작성하라”

Main()

모든 프로그램에는 **시작점**(Entry Point)이 있다. 바로 Main()다.

```
static void Main() { }
```

메소드(Method)는 어떤 기능을 수행하는 단위이다. Main()은 딱 하나만 존재해야 한다. Main()의 내부를 보자.

구문

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

컴퓨터에게 명령어를 내리는 것을 **구문**(Statement)이라고 하며, 구문은 세미콜론(;)으로 구분한다. Main()에는 현재 하나의 구문만 존재한다.

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

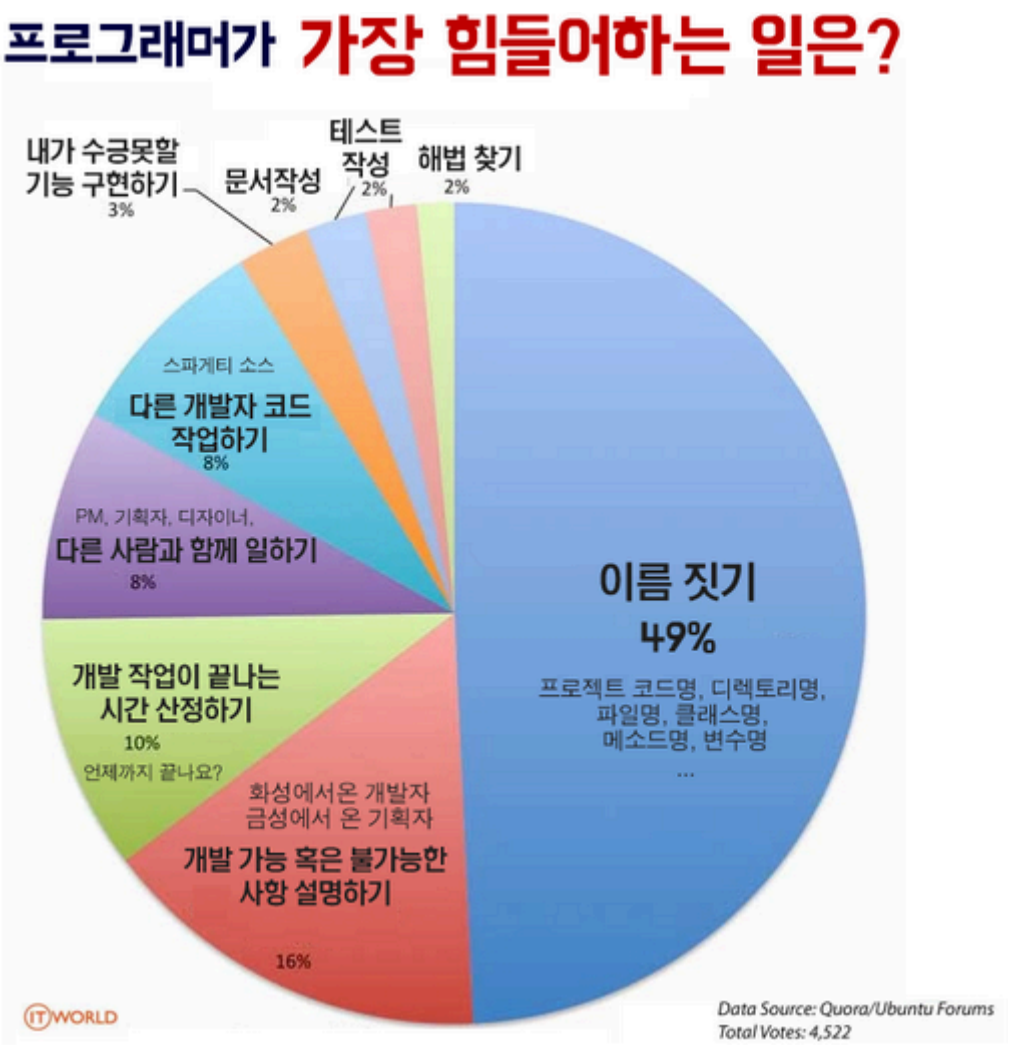
구문에는 여러 종류가 있는데, 위와 같은 것은 **식 구문**(Expression Statement)라고 한다. 식은 연산자와 피연산자를 가지고 계산하는 것이다. 이 구문에 있는 연산자로는 **멤버 접근 연산자**(Member Access Operator)와 **주문 연산자**(Invocation Operator)가 있다. 멤버 접근 연산자는 '~의' 라는 뜻을 갖고 있고, 주문 연산자는 메소드를 호출(Call)하는 것이다. 호출이라는 것은 메소드를 실행하는 것을 말한다. 위의 구문을 해석하면 'System의 Console의 WriteLine()를 호출하라'는 것이다.

이름 공간

위 구문에서 System은 **이름 공간**(Namespace)이다. 좋은 코드는 빠르면서도 올바르게 이해할 수 있다는 특성을 가져야 하는데, 이 특성의 기초가 되는 것이 바로 이름이다. 즉, 코드가 자기 설명적(Self-Descriptive)이기 위해서는 명확한 이름을 지어야 한다. 코드는 작성하는 시간보다 읽는 시간이 압도적으로 많기 때문에 명확한 이름을 짓는 것에 많은 노력을 기울이도록 하자.*

* 명명과 관련해서는 여기를 참고하자.

```
// 만약 Write가 아닌 W라는 이름을 갖고 있었다면?
// 코드만으로는 무엇을 하는지 명확하게 알 수 없기 때문에
// 이 코드를 설명하기 위한 주석이 필요하다.
System.Console.W("P"); // Bad : 플레이어를 출력한다.
```



이름을 짓는 것이 얼마나 어려운가. 오죽하면 [이런 사이트](#)도 존재한다. 요즘은 AI를 통해서 좋은 이름을 제안받을 수 있다.

때에 따라(아니 사실은 종종) 이름이 겹치는 경우도 존재한다. 이름이 겹치면 컴파일러 입장에서는 어떤 요소인지 알 수 없으므로 컴파일 오류가 발생하게 된다. 가령 A가 미사일을 만든다고 해보자.

```
// A가 만든 미사일
class Missile { }
```

마침 B도 미사일을 만들었다고 해보자.

```
// B가 만든 미사일
class Missile { }
```

그럼 우리가 사용할 때, 어떤 게 A가 만든 것이고 어떤 게 B가 만든 것인지 구별할 수 있을까? 현재는 불가능하다.

```
class Missile { }

class Program
{
  static void Main()
  {
    Missile missile; // 누구의 미사일일까?
  }
}
```

아래와 같이 이름 공간을 만든다면 쉽게 분류할 수 있다.

```
// A가 만든 것들의 모음
namespace A
{
  // 이름 공간 안에는 여러 가지 구성 요소를 넣을 수 있다.
  class Missile { }
}

// B가 만든 것들의 모음
namespace B
{
  class Missile { }
}

class Program
{
  static void Main()
  {
    A.Missile missile; // 이제는 A의 Missile을 사용한다는 것을 알 수 있다.
  }
```

```
}
```

그래서 이름 공간은 말 그대로 이름이 서로 겹치는 것을 방지하기 위해 존재하며 성격이나 하는 일이 비슷한 클래스, 인터페이스, 대리자 등 여러 구성 요소를 묶는 카테고리 같은 것이라고 생각하면 된다. 매번마다 이름 공간을 작성하는 것이 귀찮을 수 있다. 이때는 using 지시자를 사용할 수 있다.

```
// using 지시자를 이용하면 컴파일러가 어떤 구성 요소를 찾을 때
// 아래의 이름 공간에서도 찾는다.
using System;

class Program
{
  static void Main()
  {
    // 이제는 System을 안 붙여도 된다.
    // using 지시자 덕분에 Console이라는 이름을
    // System 이름 공간에서도 찾기 때문이다.
    Console.WriteLine("Hello, World!");
  }
}
```

클래스 라이브러리

자, 그런데 한 가지 의문점이 생긴다. Console은 우리가 만든 것이 아님에도 불구하고 잘 동작하고 있다. 컴퓨터 세상에서 마법 같이 동작하는 것은 없다. 일일이 컴퓨터에게 알려줘야 한다. 즉, Console을 누군가가 만들었다*는 얘기다.

* 남의 만든 기술을 사용하는 걸 **기술 부채**(Technical Dept)라 한다. 부채이기 때문에 기술 부채도 갚아야 한다. 갚는 방법은 그 기술이 어떻게 동작하는지 이해하는 것이다. 현실의 부채와 달리 모든 것을 꼭 다 갚을 필요는 없다.

처음에 프로그래밍에서 배울 때 기억하는가? 프로그래밍 언어는 크게 코어와 라이브러리로 나눌 수 있다고 했다. 그렇다. Console은 언어 개발자가 만든 라이브러리다. C#에서는 모두 라이브러리가 클래스로 되어 있기 때문에 이를 **표준 클래스 라이브러리**(Standard Class Library)라고 한다. MS에서 제공해주고 있는 클래스 라이브러리의 개수는 1만 여가지다. 이 모든 것을 외울 필요는 없고*, 그때그때 찾아서 사용하면 된다. 목록은 여기서 확인할 수 있다.

* C#을 만든 사람도 불가능하다.

컴파일 오류

아래의 코드에서 잘못된 부분을 찾을 수 있는가?

```
class Program
{
  static void Main()
  {
    System.console.WriteLine("Hello, World!");
  }
}
```

여러분이 이 코드를 직접 작성해본다면 컴파일러가 빨간 줄로 무엇인가를 표시해줄 것이다. 이런 오류를 **컴파일 오류**(Compile Error)라고 한다. 빌드에는 여러 가지 세부 과정이 있는데*, 그 중 하나가 바로 컴파일이다. 컴파일 오류는 컴파일 과정에서 문제가 생긴 것으로 다시 말해 번역이 불가능하다는 것을 의미한다. 컴파일 오류가 일어나는 원인은 문법을 지키지 않아서다. IDE가 알려주는 오류는 검색으로 해결 방법을 대부분 찾을 수 있으니 오류가 났다면 당황하지 말고, 침착하게 어떤 오류인지 확인하고 수정할 수 있어야 한다. 이 외에도 다른 오류들이 있지만, 차차 알아보도록 한다.

* 이에 대해선 추후 자세히 배운다.

더 나아가기

1. 좋은 이름을 지으려면 어떻게 하는 것이 좋을까요?
2. 주석은 언제 활용하는 것이 좋을까요?
3. 컴파일 오류란 무엇인가요? 언제 발생하나요? 어떻게 해결할 수 있나요?
4. 여러분의 이름을 콘솔 창에 출력해보세요.
5. Main() 메소드를 2개 이상 작성할 경우 어떤 일이 일어나는지 확인해보세요.