

HCI 프로그래밍

1. C# 프로그램 기초

HCI

Human Computer Interaction

Visual Studio 2019 통합 개발 환경

메뉴 및 툴바

솔루션탐색기

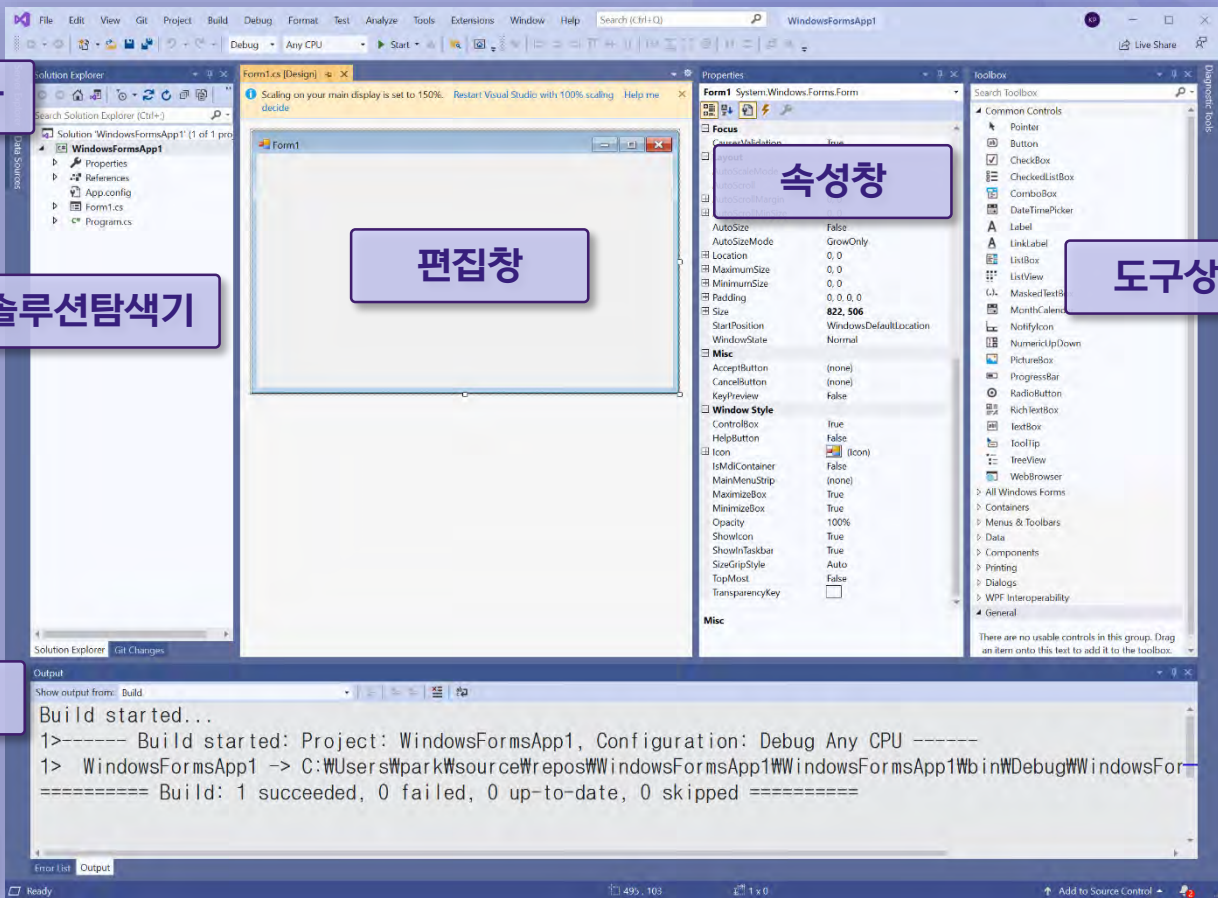
편집창

속성창

도구상자

출력창

상태표시줄



- 🔔 **The Editor**
 - VB/C/C++/C# 코드를 작성하고 수정하기 위한 환경
- 🔔 **The Compiler**
 - 소스 코드를 오브젝트 코드로 변환
- 🔔 **The Linker**
 - 수행에 필요한 모듈들을 결합
- 🔔 **The Libraries**
 - 미리 작성되어진 루틴들의 집합



Project

- 프로그램을 구성하기 위한 모든 것을 담고 있는 것
- 프로젝트 폴더가 생성
- 프로젝트의 정보는 XML 형태로 .csproj에 저장



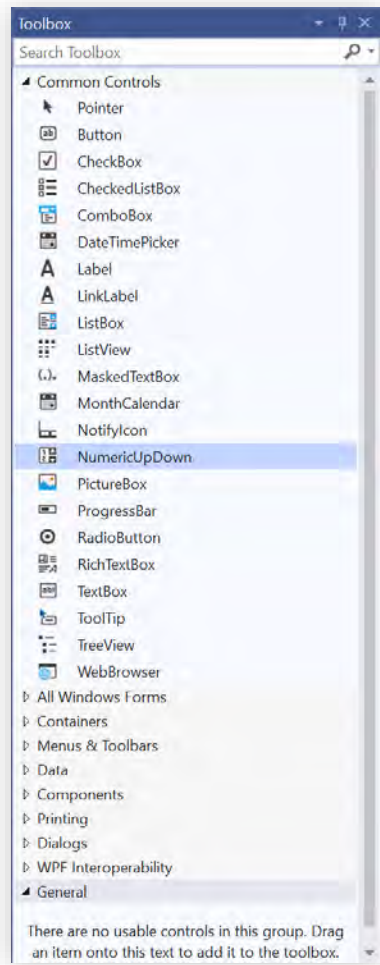
The Libraries

- 미리 작성되어진 루틴들의 집합



Toolbox (도구상자)

- 윈도우 컨트롤
- 웹폼 컨트롤
- 데이터 컨트롤
- 자주 사용되는 코드 등록 가능
- 컨트롤 추가 방법
 - ✓ 컨트롤을 폼으로 드래그,
도구상자의 컨트롤 더블 클릭





XML

- 데이터 교환의 표준으로 사용
- 데이터 교환 시 SOAP 이용
- XML과 XSD를 간단하게 작성할 수 있도록 지원



Server Explorer (서버탐색기)

- 서버 상태 리소스를 한눈에 보임
- 현재 사용 중인 모듈과 프로세스의 리스트, 서비스 항목을 볼 수 있음



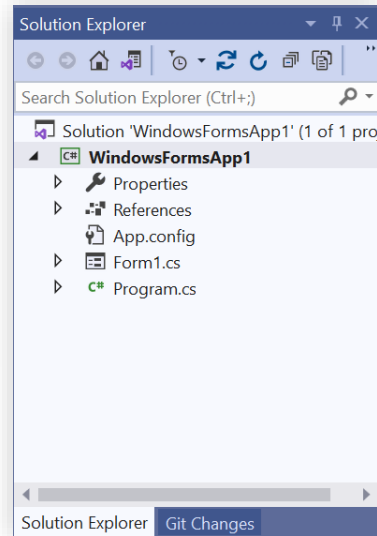
Code Editor (코드 편집기)

- 코딩시 라인별 오류를 알려줌
- 프로시저별로 코드 내용을 펼치거나 줄여서 볼 수 있음



Solution Explorer (솔루션 탐색기)

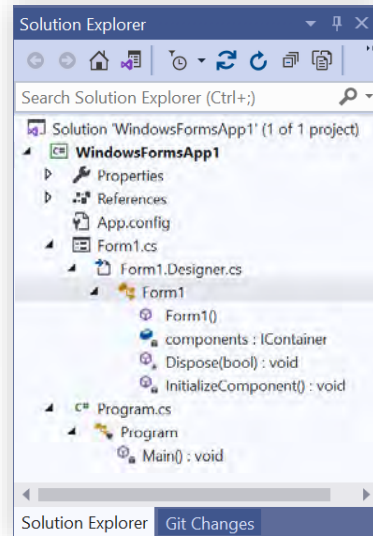
- 솔루션
 - ✓ 프로젝트보다 큰 범위, 연관된 다수의 프로젝트 포함가능
- 해당 솔루션을 구성하는 프로젝트들과 각각의 프로젝트를 구성하는 네임스페이스, 클래스, 소스 코드 등을 디렉토리화 하여 관리 가능
- 팝업 메뉴를 통한 기능 지원





Solution Explorer (솔루션 탐색기)

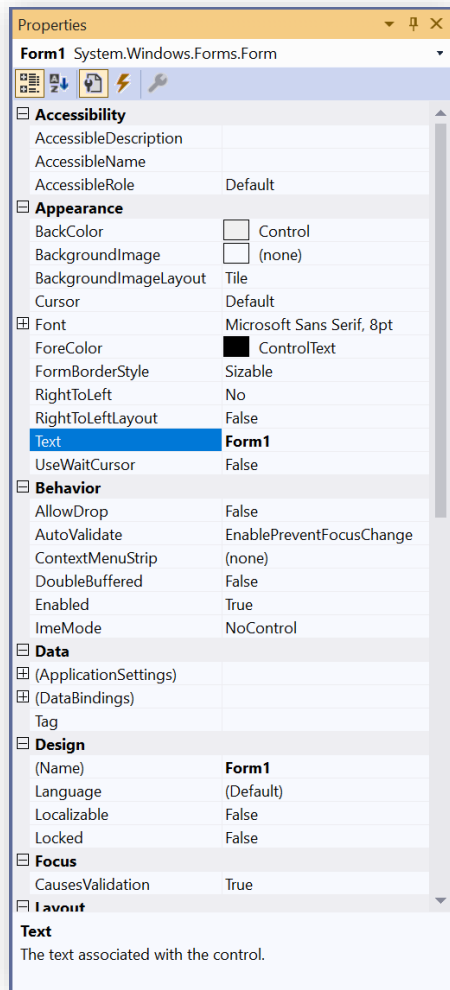
- Class View (클래스 뷰) 클래스와 메소드 정보 표시, 편집 시 사용





Property Window (속성 창)

- 컨트롤의 속성과 이벤트에 대한 설정 기능
- 기본 내용에서 바뀐 부분은 볼드체로 표시
- 컨트롤 더블 클릭 시 이벤트 핸들러에 자동으로 바인딩 기능 지원

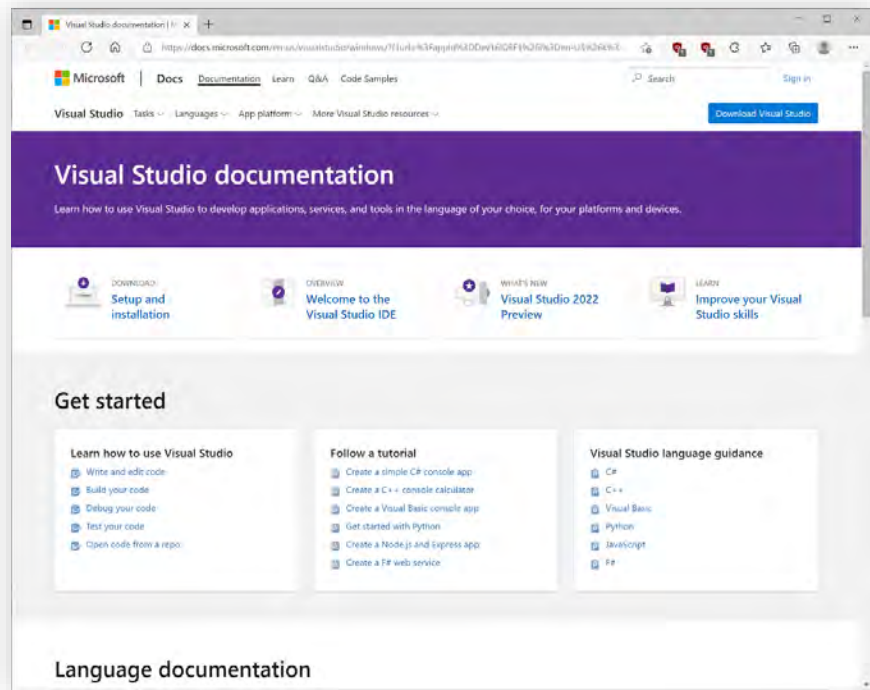




Help (도움말)

- 현재 개발자가 시행하고 있는 작업 내용을 자동으로 보여줌
- VS2010 부터 선호하는 브라우저를 사용하여 온라인 또는 오프라인으로 문서를 볼 수 있으며, 최신 문서를 필요에 따라 다운로드하고 단순화된 목차를 사용하여 탐색가능

Visual Studio





Task (작업 목록) & Output Window (출력 창)

- 프로젝트가 컴파일 될 때의 정보를 표시
- 직접 입력 및 VS.NET에 의한 자동 생성

```
Output
Show output from: Build
Build started...
1>----- Build started: Project: WindowsFormsApp1, Configuration: Debug Any CPU -----
1> WindowsFormsApp1 -> C:\Users\park\source\repos\WindowsFormsApp1\WindowsFormsApp1\bin\Debug\WindowsFor
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

- .NET CLR안에서 실행되는 언어 중 하나
- 다른 언어들의 장점을 도입하고 문제점을 제거하여 설계
- 데이터에 특정한 형식이 부여되면 그와 관계없는 형식으로 변환할 수 없음
- 형식의 안전을 위해 길고 장황한 코드를 작성필요
- .NET 코드라이브러리가 제공하는 모든 기능들을 완전하게 활용



— C#으로 작성할 수 있는 응용 프로그램

- Windows 응용 프로그램
- Web 응용 프로그램
- Web Service
- 데이터 베이스 액세스를 위한 응용 프로그램(ADO.NET)
- 네트워킹 구성 요소, 그래픽 출력, 복잡한 수학 연산 등을 위한 도구

C#

C# 프로그램의 구조

- 객체 지향 언어, 클래스 단위의 프로그램 방식
- 반드시 하나 이상의 Main 메소드(method)를 갖는 클래스가 존재

```
Program.cs [X]
C# ConsoleApp1 ConsoleApp1.Program Main(string[] args)
1 using System;
2
3 namespace ConsoleApp1
4 {
5     0 references
6     class Program
7     {
8         0 references
9         static void Main(string[] args) Main() 메소드
10        {
11            Console.WriteLine("Hello World!");
12        }
13    }
14 }
```

— C# 프로그램의 구성

- 프로그램 설명 (a.k.a header comments) (optional)
- Library imports (optional)
- 하나 이상의 클래스 (그리고 namespace)를 포함
 - ➔ 하나의 클래스(class)는 하나 이상의 메소드 (method)를 포함
 - ➔ 하나의 메소드(method)는 program statements을 포함



클래스

```
// comments about the class
```

```
class HelloWorld
```

← class header | 일반적으로 클래스 이름은 대문자로 시작

```
{
```

} class body

```
}
```

주석문(comments)은 어디에서나 사용가능

— class

- C++나 Java의 클래스와 같은 개념
- C# 소스 파일 하나 안에는 여러 개의 클래스가 존재 가능
- 그러나 단일 클래스 하나를 C# 소스 파일 두 개에 연이어서 작성할 수 없음

C#

— 메소드

```
using System;  
  
// comments about the class  
class HelloWorld  
{  
  
    // comments about the method  
    static void Main (string[] args)  
    {  
        Console.Write("Hello World!");  
        Console.WriteLine("Hello HCI Programming");  
    }  
}
```

namespace

- Java naming과 같은 개념
- 클래스들을 그룹핑한 단위를 네임스페이스라고 함
- 모든 .NET 라이브러리 코드는 네임스페이스로 정리되어 있음
- 네임스페이스에 있는 코드를 참조하려면 반드시 해당 이름을 부르거나
 - ➔ `System.Console`또는 명시적으로 import
 - ➔ `using System;`해야 함

```
using System;
namespace A {
    class MyClass {
        int value = 1;
        public void MyMethod() { Console.WriteLine(value); }
    }
}
namespace B {
    class MyClass {
        double value = 2.0;
        public void MyMethod() { Console.WriteLine(value); }
    }
}
```

```
namespace MyProgram {
    class Program {
        static void Main(string[] args)
        {
            A.MyClass a = new A.MyClass();
            a.MyMethod();
            B.MyClass b = new B.MyClass();
            b.MyMethod();
        }
    }
}
```

using

- Java에 import와 같은 것
- 다른 네임스페이스의 클래스를 접근하여 사용하고자 하면 using 문을 사용하여 해당 namespace를 접근함
- using문을 사용해서 System이란 네임스페이스를 열어서 Console.WriteLine의 사용이 가능해짐

```
using System;
```

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

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

— 닷넷의 namespace

- 네임스페이스의 기본적인 기능은 명칭의 선언 영역을 지정하여 명칭끼리의 충돌을 방지하는 것
- 닷넷 네임스페이스는 클래스를 기능별로 분류하는 역할로 함

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

C# 콘솔 응용프로그램

Create a new project

Search for templates (Alt+S)

Recent project templates

Windows Forms App (.NET Framework) C#

All languages

All platforms

All project types



Console Application

A project for creating a command-line application that can run on .NET Core on Windows, Linux and macOS

C#

Linux

macOS

Windows

Console



Console Application

A project for creating a command-line application that can run on .NET Core on Windows, Linux and macOS

Visual Basic

Linux

macOS

Windows

Console



ASP.NET Core Web App

A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.

C#

Linux

macOS

Windows

Cloud

Service

Web



Blazor WebAssembly App

A project template for creating a Blazor app that runs on WebAssembly and is optionally hosted by an ASP.NET Core app. This template can be used for web apps with rich dynamic user interfaces (UIs).

C#

Linux

macOS

Windows

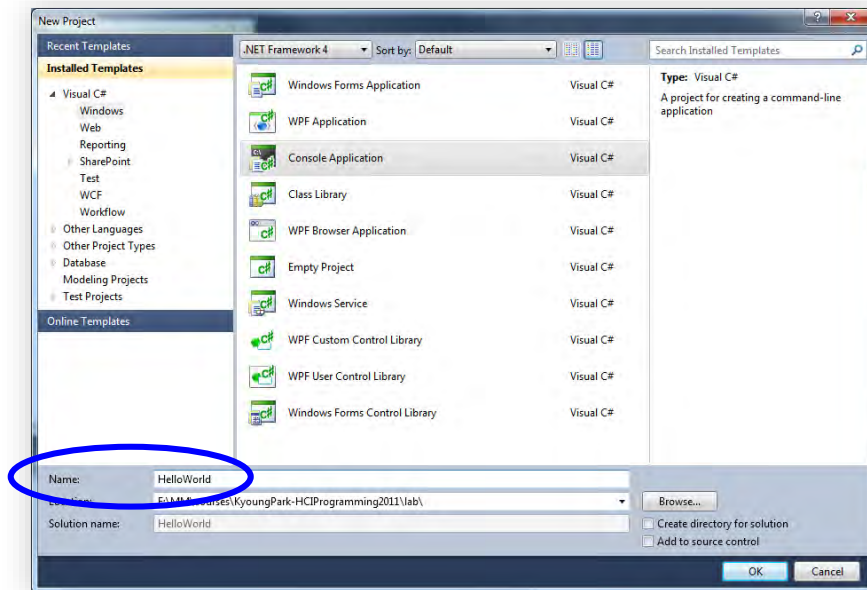
Cloud

Web

Next

C# 콘솔 응용프로그램

- Visual C# 프로젝트에서 콘솔 응용 프로그램 (Console Application) 선택
- HelloWorld 프로젝트 이름 입력



The image shows a screenshot of the Visual Studio IDE. The main window displays the source code for a C# console application named "ConsoleApp1". The code is as follows:

```
1 using System;
2
3 namespace ConsoleApp1
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             Console.WriteLine("Hello World!");
10        }
11    }
12 }
13
```

The Solution Explorer on the left shows the project structure:

- Solution 'ConsoleApp1' (1 of 1 project)
 - ConsoleApp1
 - Dependencies
 - Program.cs
 - Program

The Output window at the bottom shows the following text:

```
Show output from: Debug
ConsoleApp1.exe (CoreCLR: clrhost). Loaded C:\Program Files\dotnet\shar...
'ConsoleApp1.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shar...
'ConsoleApp1.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shar...
'ConsoleApp1.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shar...
The program '[18440] ConsoleApp1.exe' has exited with code 0 (0x0).
```

The status bar at the bottom indicates "Ready" and "Add to Source Control".

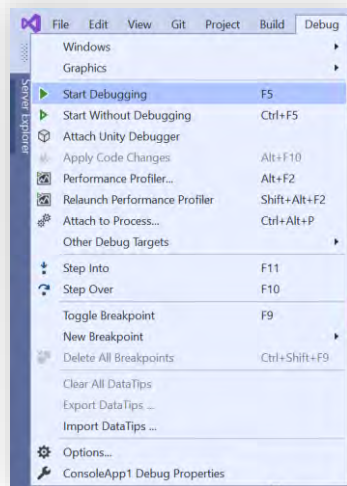
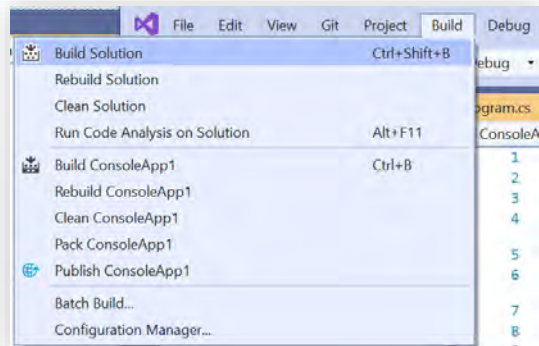
— 코드추가

```
Program.cs  + X
C# ConsoleApp1 ConsoleApp1.Program Main(string[] args)
1 using System;
2
3 namespace ConsoleApp1
4 {
5     0 references
6     class Program
7     {
8         0 references
9         static void Main(string[] args)
10        {
11            Console.WriteLine("Hello World!");
12            Console.WriteLine("Hello HCI Programming!");
13        }
14    }
15 }
```

Hello World!
Hello HCI Programming!

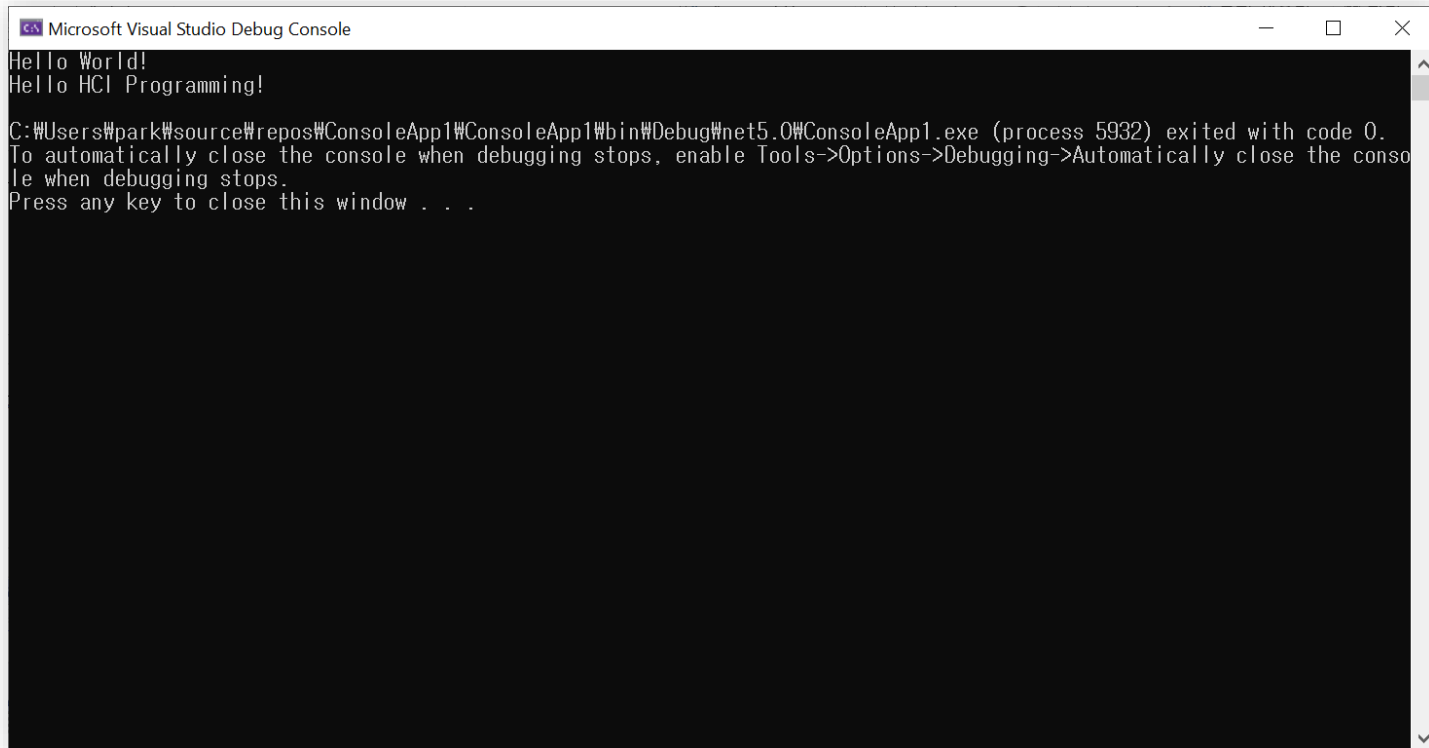
— 컴파일과 실행

- 메뉴에서 [빌드->솔루션 빌드]를 선택하여 컴파일
- [디버그->시작]을 선택하여 실행한다.
- 컴파일과 실행을 일괄적으로 처리하기 위해선 Ctrl+F5를 누르면 된다.



C# 콘솔 응용프로그램

— 실행 결과 화면



The screenshot shows the Microsoft Visual Studio Debug Console window. The title bar reads "Microsoft Visual Studio Debug Console". The console output is as follows:

```
Hello World!  
Hello HCI Programming!  
  
C:\Users\#park\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\net5.0\ConsoleApp1.exe (process 5932) exited with code 0.  
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.  
Press any key to close this window . . .
```

— Program.cs 분석

- 자바와 같이 main() 메소드를 포함하고 있는 클래스 이름과 파일 이름이 같아야 할 필요 없음
- BCL(Base Class Library)중 **System namespace** 안에 정의된 클래스 사용 명시

```
using System;
```

- **ConsoleApp1** 란 이름의 **namespace**로 정의 유지보수와 프로그램 이해 이점

```
namespace ConsoleApp1
```

```
{  
    ...//  
}
```

— class란 키워드를 사용하여 Program 클래스 선언

```
class Program  
{  
    ...  
}
```

— 메인 함수를 하나의 싱글 쓰레드 안에서 실행하기 위해 Attribute를 선언

- Main() 메소드 안에서 멀티 쓰레드를 구현하기 위해서는 [MTAThread]로 설정

```
[STAThread]
```

— Main() 메소드는 응용프로그램의 시작점

- static
 - ✓ 모든 클래스에서 공유하기 위한 멤버를 선언하는데 사용객체를 생성하지 않고 시작이 가능
- string[] args
 - ✓ 매개변수, 명령 행의 명령어를 인자로 받음

```
static void Main(string[] args)
```

— Console 클래스는 콘솔 응용프로그램에 대한 표준 입출력 및 오류 스트림을 말함

- Console 클래스의 WriteLine() 메소드를 이용하여 “ ” 내의 Hello World! 문자열 출력

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

— Write/WriteLine 메소드

- 스크린 모니터 상에 정보를 보여줌
- WriteLine은 캐리지 리턴 문자를 포함하여 출력 후에 다음 행으로 이동
- Write/WriteLine은 오버로드 되어있으므로 인자 형태로 숫자, 문자열 등 여러 형태가 가능
- {index [,alignment][:formatting]}를 사용하여 인자를 출력
- Java에서와 같이 사용
 - ➔ "Test" + a

— Read/ReadLine 메소드

- 키보드로 값을 입력 받을 때 사용
- Read 메소드는 키보드로부터 하나의 문자를 입력 받음
- ReadLine 메소드는 한 줄을 입력 받음

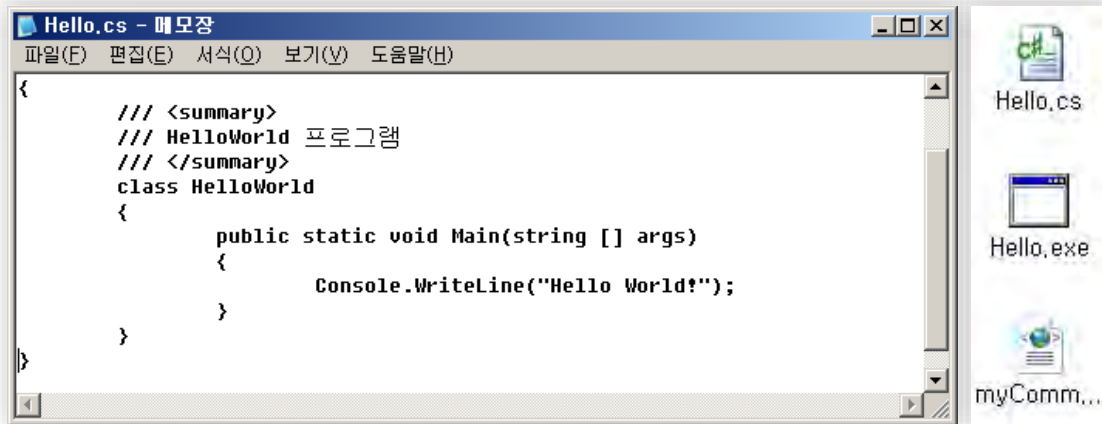
— 주석문

- 컴파일 시 무시되며, 소스파일에서만 관리됨
- `//`를 사용하여 한 줄을 주석 처리함
- `/* */`를 사용하면 여러 라인을 주석처리 할 수 있음

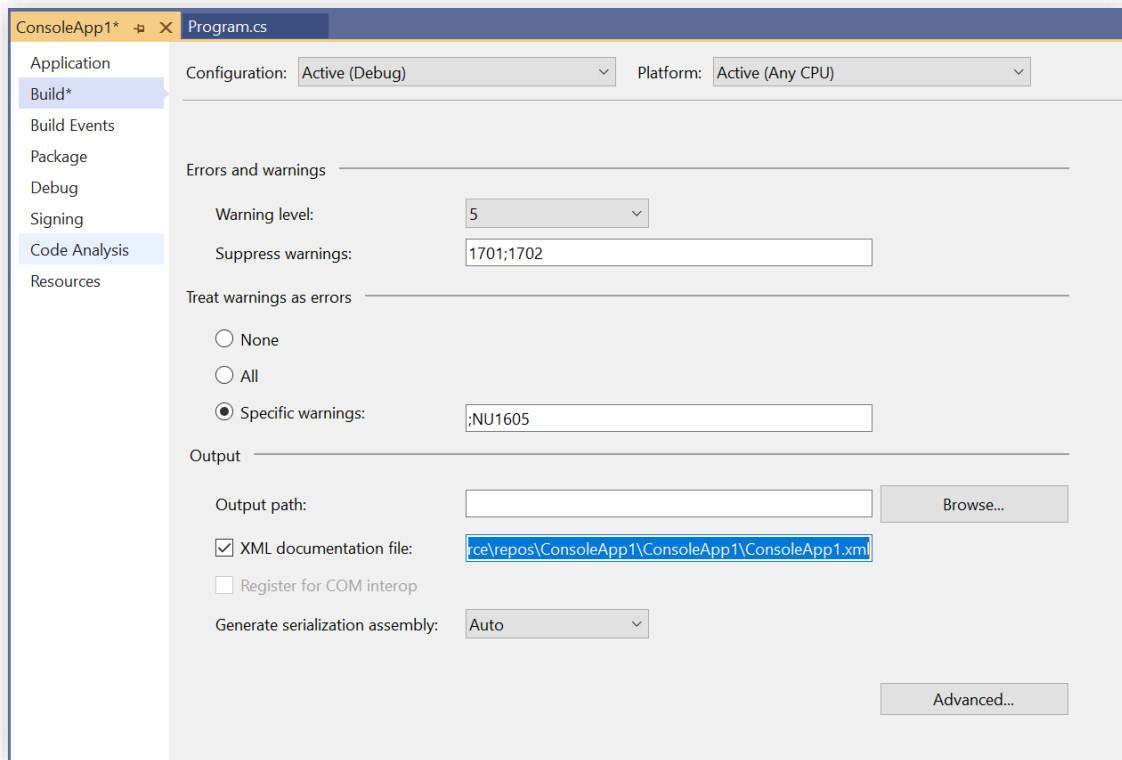
— XML 문서 만들기

- 소스 상의 주석으로 입력했던 내용을 문서화할 때 사용
- ///로 XML 문서 부분을 표시해줌
- 컴파일 시 XML 문서옵션을 주어 XML문서를 만들

```
~>csc.exe Hello.cs /doc:myComment.xml
```



Visual Studio에서 XML문서 만들기 설정



Console vs. Windows Application

— 콘솔 응용프로그램 (Console Application)

- Visual component 지원하지 않음
- 텍스트 입력과 출력만 지원
- 도스창에서 실행

Console vs. Windows Application

— 윈도우 응용프로그램 (Windows Application)

- Graphical User Interfaces (GUI)를 통한 다양한 입출력 지원
- 메시지 박스
 - ➔ System.Windows.Forms 네임스페이스에 있음

```
using System;
using System.Windows.Forms;
class Welcome
{
    static void Main( string[] args )
    {
        MessageBox.Show( "Welcome\nto\nC#\nprogramming!" );
    }
}
```

