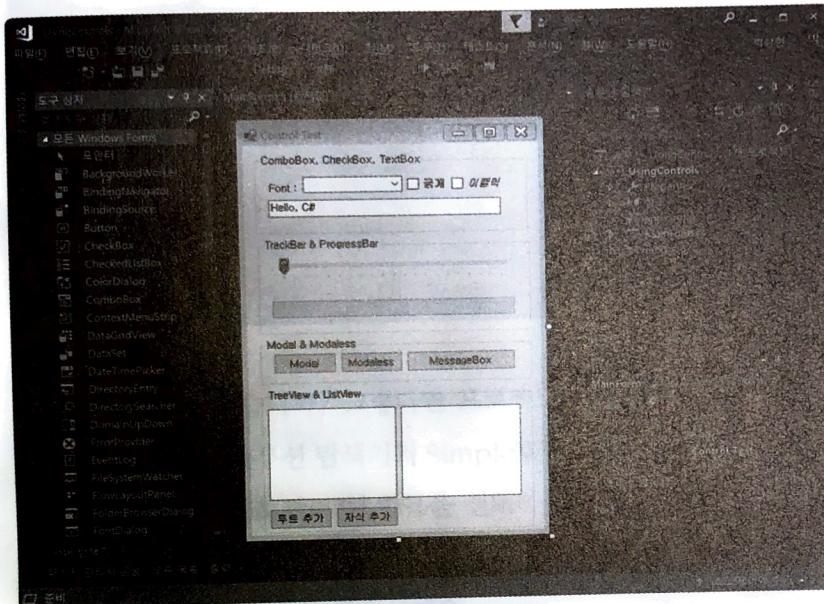


20.1 도대체 무슨 일이 일어나고 있는 걸까?

WinForm은 폼 디자이너라는 툴을 제공해서 프로그래머가 그림 그리듯 UI를 만들 수 있게 합니다. 다음 그림처럼 도구 상자에서 버튼이나 콤보박스 등의 컨트롤을 끌어다 윈도우 위에 올려놓으면 컴파일 후의 프로그램과 똑같은 UI가 만들어지지요. 이른바 WYSIWYG What You See Is What You Get 방식의 개발을 지원하는 겁니다.



컨트롤을 윈도우 위에 배치할 때마다 폼 디자이너는 우리에게 프로그램의 UI를 표시하는 한편, 뒤로는 관련 C# 코드를 자동으로 만들어 줍니다. 프로퍼티를 변경할 때, 이벤트 처리기를 추가할 때도 자동으로 코드를 수정해줍니다.

우리는 C# 프로그래머입니다. C# 코드만으로도 비주얼 스튜디오의 도움이 없이도 GUI를 구성할 수 있어야 하지 않을까요? 직접 코드를 작성하는 것보다는 폼 디자이너를 이용해서 UI를 만드는 편이 시간을 수십 배 절약할 수 있지만, 미세한 조정이 필요하거나 폼 디자이너에 문제가 생기는 경우 (마이크로소프트의 프로그래머들도 우리와 똑같은 '사람'입니다. 그들도 버그를 만들기 마련이죠)에는 직접 팔을 걷어붙이고 해결할 수 있어야 합니다.

이런 이유로, 저는 여러분들에게 C# 코드를 이용하여 WinForm UI를 만드는 방법을 먼저 설명하겠습니다. 다음 절에서는 UI의 바탕이 되는 윈도우를 만드는 방법을 설명하고, 그 이후의 절에서는 버튼이나 텍스트박스 같은 컨트롤을 윈도우 위에 올리는 방법을 다루겠습니다.

20.2 C# 코드로 WinForm 윈도우 만들기

백조들이 호수 위에 떠서 우아하게 유영하는 모습을 보면 한가로워 보입니다. 저도 TV를 통해 백조가 호수 위에서 노니는 모습을 볼 때면 저렇게 여유롭게 살았으면 좋겠다라는 생각을 했습니다. 그런데 그 우아한 유영의 속사정을 알고 난 후부터는 백조들이 조금은 측은하게 느껴졌습니다. 저보다 더 힘들게 살더군요. 그 속사정이 뭐냐고요? 그것은 바로 백조들이 우아하게(보이는) 유영을 하기 위해서는 물 속에서 발길질을 끊임없이 빠르게 해야 한다는 것입니다. 덩치에 비해 짧고 작은 발을 갖고 있는 백조의 아픔이라고 할 수 있죠.

뜬금없이 웬 백조 이야기를 하냐고요? 윈도우를 가지는 응용 프로그램들도 백조와 비슷한 치지이기 때문입니다. 윈도우 하나를 만들려고 꽤 번거로운 절차를 거쳐야 하거든요. 다음이 윈도우를 만드는 바로 그 절차입니다.

- ① 윈도우 클래스(OOP의 클래스와는 다릅니다. “윈도우에 대한 정보를 가지고 있는 구조체” 정도로 알아두세요)를 정의합니다.
- ② 정의된 윈도우 클래스를 등록합니다.
- ③ 윈도우를 생성합니다.
- ④ 윈도우를 사용자에게 보여줍니다.
- ⑤ 메시지 루프를 돌면서 프로그램을 시작합니다.

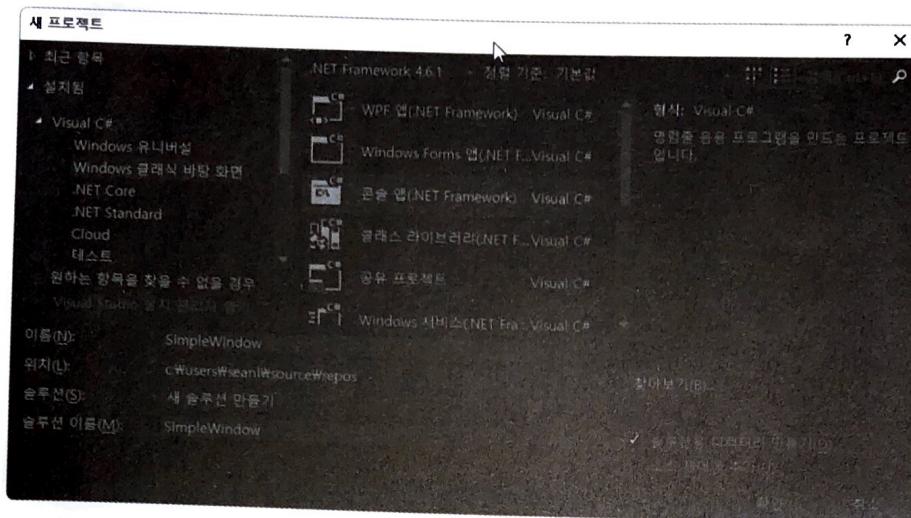
하지만 이것은 Win32 API를 이용하여 윈도우를 만들 때의 이야기입니다(더 자세히 알고 싶은 독자는 나중에 Win32 API를 공부해보세요). .NET 프레임워크는 이러한 과정들을 잘 포장해서 개발자들이 간편하게 윈도우를 만들 수 있도록 WinForm 클래스 라이브러리를 제공합니다. WinForm 클래스를 이용한 윈도우 생성 절차는 다음과 같습니다.

- ① System.Windows.Forms.Form 클래스에서 파생된 윈도우 폼 클래스를 선언합니다.
- ② ①번에서 만든 클래스의 인스턴스를 System.Windows.Forms.Application.Run() 메소드에 매개 변수로 넘겨 호출합니다.

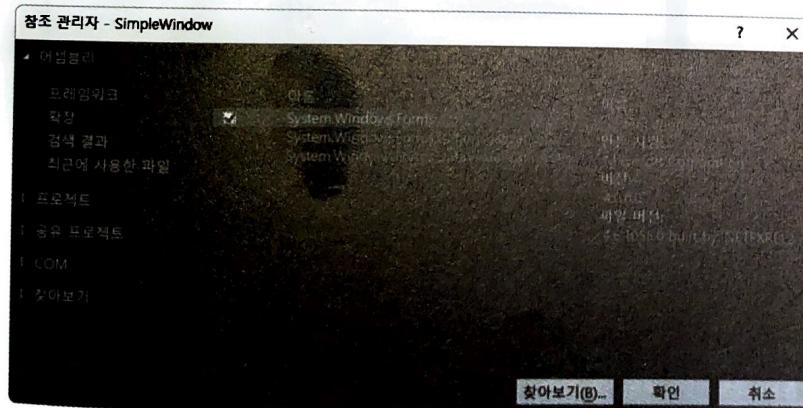
이게 전부입니다. 절차도 훨씬 간편하지만, 실제 코드의 양도 약 1/10 정도로 WinForm 클래스를 사용하는 코드가 더 적습니다. 이렇게 이야기만 할 게 아니라 프로그램을 한번 만들어 보죠.

비주얼 스튜디오를 실행하고 프로젝트 템플릿을 “콘솔 앱”으로 선택한 뒤, 프로젝트의 이름을 “SimpleWindow”라고 입력해서 새 프로젝트를 생성하세요(주의: “Windows Forms 앱” 템플릿

이 아닙니다. 이번 장에서는 모든 예제 프로그램 프로젝트를 같은 방법으로 생성해서 따라 만드시기 바랍니다).



프로젝트를 생성했습니까? 그렇다면 프로젝트 참조에 System.Windows.Forms 어셈블리를 추가할 차례입니다. 솔루션 탐색기의 SimpleWindow 프로젝트의 [참조] 항목에서 마우스 오른쪽 버튼을 클릭한 뒤 [참조 관리자] 항목을 선택하십시오. 그런 후 다음과 같은 참조 관리자 대화상자가 나타나면 [어셈블리] → [프레임워크] 항목을 선택하고 System.Windows.Forms 어셈블리를 찾아 선택하고 [확인] 버튼을 클릭하세요.



여기까지 하면 코딩을 시작할 준비가 됐습니다. 프로젝트의 Program.cs 파일을 MainApp.cs로 이름을 변경하고 다음의 코드를 따라 입력한 뒤 컴파일해서 실행해보세요.

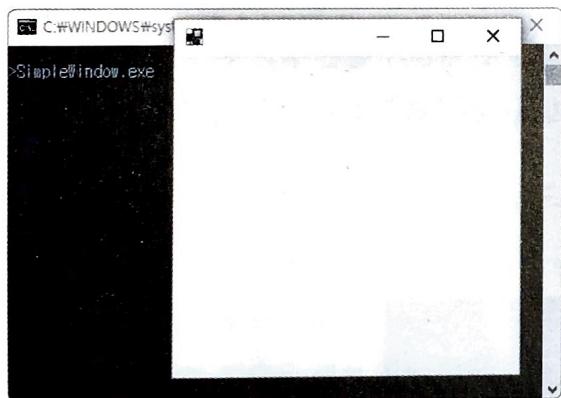
» 20장/SimpleWindow/MainApp.cs

```
01  using System;
02
03  namespace SimpleWindow
04  {
05      class MainApp : System.Windows.Forms.Form
06      {
07          static void Main(string[] args)
08          {
09              System.Windows.Forms.Application.Run(new MainApp());
10          }
11      }
12  }
```

MainApp은 System.Windows.Forms.Form 클래스로부터 상속받도록 선언합니다.

Application.Run() 메소드에 MainApp의 인스턴스를 매개 변수로 넘겨 호출합니다.

실행 결과 :



제 이야기가 맞지요? WinForm을 이용하면 아주 간단하게 윈도우를 생성할 수 있다는 것 말입니다. 그나저나 예제 프로그램의 코드를 보니 우리가 처음 만나는 클래스가 딱 두 가지가 있네요. System.Windows.Forms.Form 클래스와 System.Windows.Forms.Application 클래스 말입니다. 당연히 이 두 클래스에 대해 자세히 알아봐야겠지요? 덮어놓고 이들을 이용해서 프로그램을 만들 수는 없는 노릇이니 말입니다. Form 클래스는 일단 놔뒀다가 나중에 다루기로 하고, 먼저 Application 클래스부터 살펴보겠습니다.

20.3 Application 클래스

이번 절의 내용을 미리 요약하자면 “Application 클래스는 크게 두 가지 역할을 수행하는데 하나는 윈도우 응용 프로그램을 시작하고 종료시키는 메소드를 제공하는 것이고, 또 다른 하나는 윈도우 메시지를 처리하는 것”이라고 할 수 있습니다.

응용 프로그램을 시작하도록 하는 메소드는 우리가 조금 전에 만들었던 예제 프로그램에서 본 것처럼 Application.Run()입니다. 그리고 응용 프로그램을 종료시키는 메소드는 Application.Exit()입니다. 어느 곳에서든 Application.Exit() 메소드를 호출하면 해당 응용 프로그램은 종료합니다. 다음은 이 두 가지 메소드를 사용하는 예제 코드입니다.

```
class MyForm : System.Windows.Forms.Form
{
}

class MainApp
{
    static void Main(string[] args)
    {
        MyForm form = new MyForm();
        form.Click += new EventHandler((sender, eventArgs) =>
        {
            Application.Exit();
        });

        Application.Run(form);
    }
}
```

Form 클래스는 여러 가지 이벤트를 정의하고 있는데, 그 중 Click 이벤트는 윈도우를 클릭 했을 때 발생하는 이벤트입니다. 따라서 이 코드는 윈도우를 클릭하면 Application.Exit()를 호출하도록 합니다.

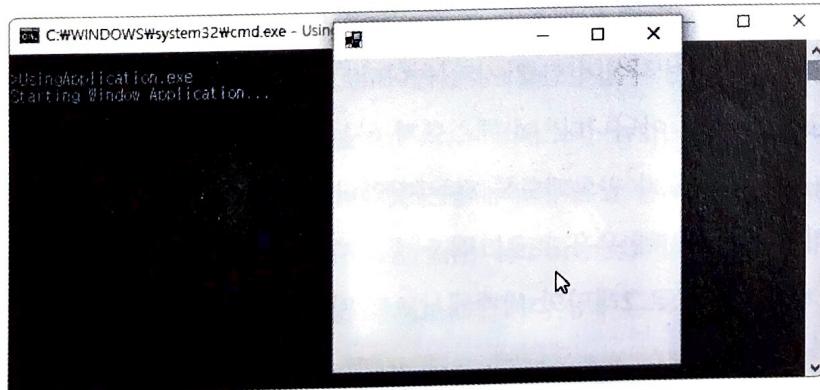
한편, Exit() 메소드에 대해 꼭 알아둬야 하는 사실이 한 가지 있습니다. Exit() 메소드가 호출된다 고 해서 응용 프로그램이 바로 종료되는 것은 아닙니다. 이 메소드가 하는 일은 응용 프로그램이 갖고 있는 모든 윈도우를 닫은 뒤 Run() 메소드가 반환되도록 하는 것입니다. 따라서 Run() 메소드 뒤에 자원을 정리하는 코드를 넣어두면 우아하게 응용 프로그램을 종료시킬 수 있습니다(19장에서 도 이야기했지만, 우리가 사는 세계에서나 프로그래밍의 세계에서도 갑자기 끝나거나 죽는 것은 썩 반길만한 일은 아닙니다).

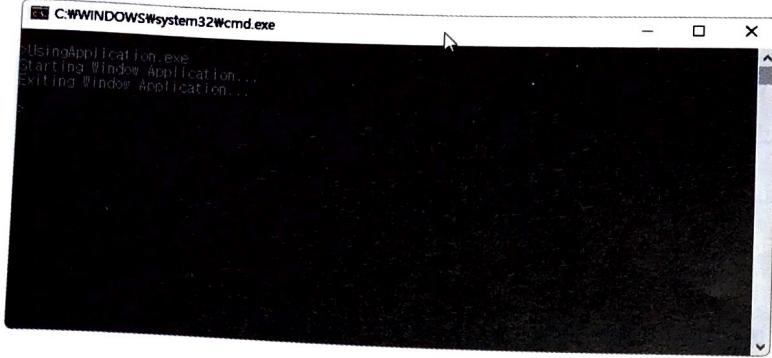
예제 프로그램을 하나 만들어 보겠습니다.

» 20장/UsingApplication/MainApp.cs

```
01  using System;
02  using System.Windows.Forms;
03
04  namespace UsingApplication
05  {
06      class MainApp : Form
07      {
08          static void Main(string[] args)
09          {
10              MainApp form = new MainApp();
11
12              form.Click += new EventHandler(
13                  (sender, eventArgs) =>
14                  {
15                      Console.WriteLine("Closing Window...");
16                      Application.Exit();
17                  });
18
19              Console.WriteLine("Starting Window Application...");
20              Application.Run(form);
21
22              Console.WriteLine("Exiting Window Application...");
23          }
24      }
25  }
```

실행 결과 :





이번에는 Application 클래스의 진짜 재미있는 기능인 “메시지 필터링Message Filtering”을 알아보겠습니다. 먼저 메시지가 뭔지부터 이야기를 시작해야겠지요?

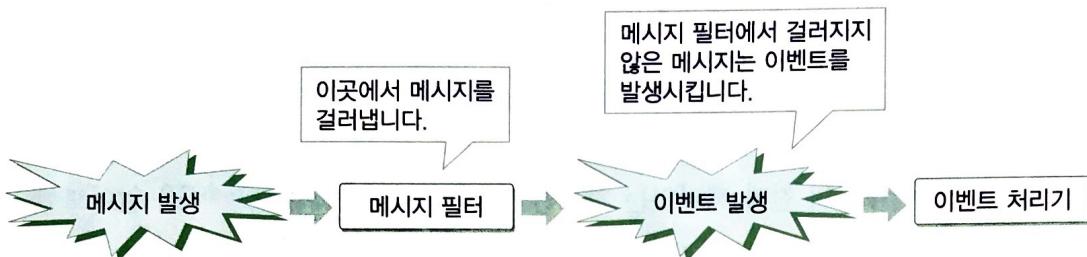
이 책의 1장부터 19장까지 만들었던 응용 프로그램들은 일정한 방향으로 코드가 실행되지만, 윈도우 기반의 응용 프로그램들은 갑자기 일어나는 사건(이벤트 : Event)에 반응해서 코드가 실행되는 이른바 Event Driven 방식으로 만들어집니다. 윈도우 응용 프로그램이 받아들이는 이벤트 중에는 바로 앞에서 만들었던 예제 프로그램에서처럼 마우스 “클릭”, 키보드 입력 등이 있습니다. 이 이벤트들은 일견 사용자가 직접 응용 프로그램에 대해 일으킨 것처럼 보이지만, 사실은 윈도우 운영체제가 일으킨 겁니다. 사용자가 마우스나 키보드 같은 하드웨어를 제어하면 인터럽트가 발생하고, 이 인터럽트를 윈도우 운영체제가 받아들입니다. 운영체제는 다시 이 인터럽트를 바탕으로 윈도우 메시지Windows Message를 만든 뒤 이벤트를 받아야 하는 응용 프로그램에게 보내주지요.

한편, 윈도우 메시지는 그 종류가 매우 다양합니다. 윈도우 응용 프로그램은 마우스 이동이나 클릭, 키보드 입력처럼 미리 시스템에 정의되어 있는 메시지를 받지만, 다른 응용 프로그램이 자체적으로 정의한 메시지도 받을 수 있습니다. 메시지 개수 자체가 많은 것은 말할 것도 없습니다(지금은 별로 공감이 안 되지요? 잠시 후에 만들 예제 프로그램의 실행 결과를 확인해 보면 실감할 수 있을 겁니다).

Application 클래스는 응용 프로그램이 받고 있는 수많은 메시지 중에 관심 있는 메시지만 걸러낼 수 있는 메시지 필터링Message Filtering 기능을 갖고 있습니다. 가령 여러분이 만든 응용 프로그램을 사용자가 [Alt] + [F4] 키를 눌러서 종료시키는 것을 막고 싶다면, 바로 이 기능을 이용해서 해당 키 입력 메시지를 걸러내면 응용 프로그램의 윈도우가 닫히는 것을 막을 수 있습니다. 물론 [Alt] + [F4] 키 입력만 걸러내므로 윈도우 종료 버튼이나 응용 프로그램 자체의 종료 기능은 정상적으로 동작합니다.

윈도우 운영체제에서 정의하고 있는 메시지는 식별 번호(ID)가 붙여져 있습니다. 예를 들어 WM_LBUTTONDOWN 메시지는 ID가 0x201로 정의되어 있지요. Application 클래스는 특정 ID를

갖는 메시지를 걸러내는 필터를 함께 등록해뒀다가 응용 프로그램에 메시지가 전달되면 해당 필터를 동작시킵니다. 만약 메시지의 ID가 필터에서 관심을 갖고 있는 값이라면 필터는 메시지를 요리하고, 그렇지 않다면 메시지를 거르지 않고 메시지를 받아야 하는 폼이나 컨트롤로 보내서 이벤트를 발생시킵니다.



`Application.AddMessageFilter()` 메소드는 응용 프로그램에 메시지 필터를 설치합니다. 이 메소드는 매개 변수로 `IMessageFilter` 인터페이스를 구현하는 파생 클래스의 인스턴스를 매개 변수로 받으며, `IMessageFilter`는 다음과 같이 `PreFilterMessage()` 메소드를 구현할 것을 요구합니다.

```

public interface IMessageFilter
{
    bool PreFilterMessage(ref Message m);
}

```

제가 맛보기로 `IMessageFilter` 인터페이스를 상속하는 클래스를 하나 보여드리겠습니다. 다음과 같이 우리는 `PreFilterMessage()` 메소드를 파생 클래스에서 구현해야 합니다. `IMessageFilter` 인터페이스의 구현 예는 다음과 같습니다.

```

public class MessageFilter : IMessageFilter
{
    public bool PreFilterMessage(ref Message m)
    {
        if (m.Msg >= 0x200 && m.Msg <= 0x20E)
        {
            Console.WriteLine("발생한 메시지: " + m.Msg);
            return true;
        }
    }
}

```

마우스 이동부터 마우스의 왼쪽, 오른쪽, 가운데 버튼 동작, 마우스 휠 굴림 메시지를 모두 걸러냅니다.

```

        }
        return false;
    }
}

```

위의 예제 코드에서 보는 것처럼 PreFilterMessage()를 구현할 때는 입력받은 메시지를 처리했으니 응용 프로그램은 관심을 가질 필요가 없다는 의미로 true를 반환하거나, 메시지를 건드리지 않았으니 응용 프로그램더러 처리해야 한다고 false를 반환하면 됩니다. 그리고 매개 변수로 받아들이는 Message 구조체는 다음과 같은 프로퍼티를 갖고 있는데, 이 중 Msg 프로퍼티는 메시지의 ID를 담고 있습니다.

프로퍼티	설명
HWnd	메시지를 받는 윈도우의 핸들(Handle)입니다. 핸들은 처음 보는 용어죠? 윈도우의 인스턴스를 식별하고 관리하기 위해 운영체제가 붙여놓은 번호가 바로 핸들입니다.
Msg	메시지 ID입니다.
LParam	메시지를 처리하는 데 필요한 정보가 담겨 있습니다.
WParam	메시지를 처리하는 데 필요한 부가 정보가 담겨 있습니다.
Result	메시지 처리에 대한 응답으로 원도우 운영체제에 반환되는 값을 지정합니다.

이렇게 메시지 필터를 구현했으면 다음과 같이 AddMessageFilter() 메소드를 호출하여 등록하면 됩니다.

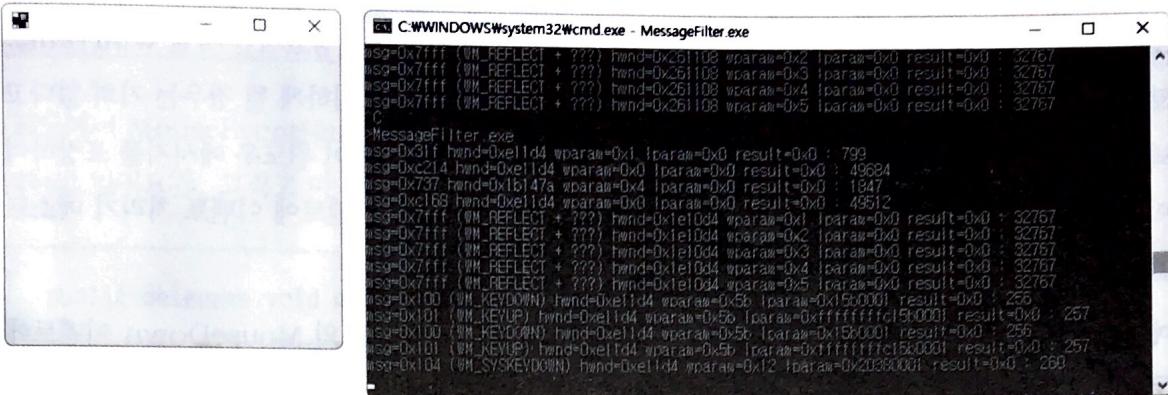
```
Application.AddMessageFilter( new MessageFilter() );
```

다음은 메시지 필터 예제 프로그램입니다. 이 예제 프로그램은 응용 프로그램이 윈도우로부터 전달받는 모든 메시지를 출력합니다. 단, WM_PAINT(0x0F), WM_MOUSEMOVE(0x200), WM_TIMER(0x113) 메시지만 제외하고 말입니다. 이들 메시지는 너무 자주, 많이 발생하기 때문에 이들 메시지들까지 출력한다면 다른 메시지를 확인하기가 너무 어려워지기 때문입니다(궁금한 분들은 예제 프로그램 코드에서 해당 코드를 주석 처리하고 다시 컴파일해서 실행해 보세요). 이 프로그램은 WM_LBUTTONDOWN(0x201) 메시지가 도착하면 Application.Exit()를 호출하여 종료합니다.

» 20장/MessageFilter/MainApp.cs

```
01 using System;
02 using System.Windows.Forms;
03
04 namespace MessageFilter
05 {
06     class MessageFilter : IMessageFilter
07     {
08         public bool PreFilterMessage(ref Message m)
09         {
10             if (m.Msg == 0x0F || m.Msg == 0xA0 ||
11                 m.Msg == 0x200 || m.Msg == 0x113)
12                 return false;
13
14             Console.WriteLine($"{m.ToString()} : {m.Msg}");
15
16             if (m.Msg == 0x201)
17                 Application.Exit();
18
19             return true;
20         }
21     }
22
23     class MainApp : Form
24     {
25         static void Main(string[] args)
26         {
27             Application.AddMessageFilter(new MessageFilter());
28             Application.Run(new MainApp());
29         }
30     }
31 }
```

실행 결과 :



The screenshot shows two windows side-by-side. The left window is titled 'MessageFilter.exe' and has a mostly blank white interior. The right window is titled 'C:\WINDOWS\system32\cmd.exe - MessageFilter.exe' and contains a log of messages. The log entries are as follows:

```
msg=0x7fff (WM_REFLECT + ???) hnd=0x261108 wparam=0x2 lparam=0x0 result=0x0 : 32767  
msg=0x7fff (WM_REFLECT + ???) hnd=0x261108 wparam=0x3 lparam=0x0 result=0x0 : 32767  
msg=0x7fff (WM_REFLECT + ???) hnd=0x261108 wparam=0x4 lparam=0x0 result=0x0 : 32767  
msg=0x7fff (WM_REFLECT + ???) hnd=0x261108 wparam=0x5 lparam=0x0 result=0x0 : 32767  
C:\Windows\system32\cmd.exe  
msg=0x31f hnd=0x1e1d4 wparam=0x1 lparam=0x0 result=0x0 : 799  
msg=0xc214 hnd=0x1e1d4 wparam=0x0 lparam=0x0 result=0x0 : 49684  
msg=0x737 hnd=0x1e147a wparam=0x4 lparam=0x0 result=0x0 : 1847  
msg=0x168 hnd=0x1e1d4 wparam=0x0 lparam=0x0 result=0x0 : 49512  
msg=0x7fff (WM_REFLECT + ???) hnd=0x1e104 wparam=0x1 lparam=0x0 result=0x0 : 32767  
msg=0x7fff (WM_REFLECT + ???) hnd=0x1e104 wparam=0x2 lparam=0x0 result=0x0 : 32767  
msg=0x7fff (WM_REFLECT + ???) hnd=0x1e104 wparam=0x3 lparam=0x0 result=0x0 : 32767  
msg=0x7fff (WM_REFLECT + ???) hnd=0x1e104 wparam=0x4 lparam=0x0 result=0x0 : 32767  
msg=0x7fff (WM_REFLECT + ???) hnd=0x1e104 wparam=0x5 lparam=0x0 result=0x0 : 32767  
msg=0x100 (WM_KEYDOWN) hnd=0x1e1d4 wparam=0x50 lparam=0x1500001 result=0x0 : 256  
msg=0x101 (WM_KEYUP) hnd=0x1e1d4 wparam=0x50 lparam=0xfffffff1500001 result=0x0 : 257  
msg=0x100 (WM_KEYDOWN) hnd=0x1e1d4 wparam=0x50 lparam=0x1500001 result=0x0 : 256  
msg=0x101 (WM_KEYUP) hnd=0x1e1d4 wparam=0x50 lparam=0xfffffff1500001 result=0x0 : 257  
msg=0x104 (WM_SYSKEYDOWN) hnd=0x1e1d4 wparam=0x12 lparam=0x20260001 result=0x0 : 209
```

어때요, 재미있었습니까? 이 메시지들은 WinForm의 각 윈도우와 컨트롤에 전달되며, 윈도우와 컨트롤은 이 메시지를 받으면 미리 정의되어 있는 이벤트를 발생시키고, 각 이벤트는 프로그래머가 등록한 이벤트 처리기를 호출합니다. 다음 절에서는 윈도우를 표현하는 Form 클래스를 다룰 텐데, 조금 전에 언급한 이벤트와 이벤트 처리기에 대한 내용도 함께 설명합니다.

20.4 윈도우를 표현하는 Form 클래스

20장을 시작한 이후 몇 차례 윈도우를 만들어 띄우긴 했는데 이걸 가지고 뭘 제대로 해본 것이 없군요. 이번 장에서는 윈도우의 모양과 크기도 바꿔보고 버튼도 올려보겠습니다. 구체적으로는 다음의 내용을 다루려고 합니다.

- Form(과 컨트롤)에 정의되어 있는 이벤트와 이벤트 처리기 연결하기
- Form의 프로퍼티를 조절하여 윈도우 모양 바꾸기
- Form 위에 컨트롤 올리기

그럼 이벤트 이야기부터 시작해볼까요?

20.4.1 Form에 정의되어 있는 이벤트와 이벤트 처리기 연결하기

이벤트는 13장에서 공부한 바 있기 때문에 우리에게 그리 낯설지 않은 친구죠? Form 클래스는 운영체제가 보내는 메시지 중 일부에 대해 이벤트를 구현하고 있습니다. 가령 사용자가 Form의 인스턴스, 즉 윈도우 위에서 마우스의 왼쪽 버튼을 누르면 WM_LBUTTONDOWN 메시지가 Form

객체로 전달되고, Form 객체는 이에 대해 MouseDown 이벤트를 발생시킵니다. 이전 절에서 윈도우 메시지를 Application 클래스를 이용하여 직접 다루는 방법을 설명했지만 사실 WinForm으로 응용 프로그램을 만드는 동안에는 우리가 직접 윈도우 메시지를 요리하게 될 경우는 거의 없다고 봐도 됩니다. Form을 비롯한 WinForm의 윈도우와 컨트롤 클래스들이 윈도우 메시지를 포장하여 이벤트로 구현해놨기 때문입니다. 우리는 그저 미리 정의되어 있는 이벤트에 이벤트 처리기 메소드를 선언하여 등록해주기만 하면 됩니다.

예제 코드를 볼까요? 이벤트 처리기 메소드를 선언하고 Form 클래스의 MouseDown 이벤트에 등록해보겠습니다.

```
class MyForm : Form
{
    // 이벤트 처리기 선언
    private void Form_MouseDown(object sender, System.Windows.Forms.MouseEventArgs e)
    {
        MessageBox.Show("안녕하세요!");
    }

    public MyForm()
    {
        // 이벤트 처리기를 이벤트에 연결
        this.MouseDown += new System.Windows.Forms.MouseEventHandler (this.Form_MouseDown);
    }
}
```

Form 클래스에는 MouseDown 외에도 무수히 많은 이벤트가 선언되어 있습니다. 이들의 목록을 여기에 늘어놓으려고만 해도 몇 페이지가 필요할 정도라면 그 수를 대강 가늠할 수 있겠습니까? 하지만 전혀 걱정하지 마세요. 이벤트 처리기를 등록하고 이를 호출하는 메커니즘은 모든 폼과 컨트롤이 똑같기 때문입니다. Form 클래스에 정의되어 있는 이벤트들을 일일이 설명하는 것보다는 MouseDown 이벤트를 조금 더 자세히 설명하겠습니다. MouseDown 이벤트를 잘 이해하면 나머지 이벤트들도 똑같이 활용할 수 있을 테니까요. 그런 의미에서 다음에 있는 MouseDown 이벤트의 선언을 보시죠.

```
public event MouseEventHandler MouseDown;
```

위 코드에서 `MouseEventHandler`는 대리자입니다(이벤트는 대리자를 기반으로 선언된다는 것, 기억하고 있지요?). 그리고 이 대리자는 다음과 같이 선언되어 있습니다.

```
public delegate void MouseEventHandler( object sender, MouseEventArgs e );
```

위 선언 코드를 보면 이벤트 처리기가 어떤 매개 변수를 가져야 하는지 그리고 어떤 형식을 반환해야 하는지 알 수 있습니다. 일단 반환 형식이 `void`이므로 이벤트 처리기는 아무 값도 반환할 필요가 없습니다. 그리고 두 개의 매개 변수를 받아들이며, 첫 번째 매개 변수는 `object` 형식입니다. 첫 번째 매개 변수 이름이 `sender`이지요? `sender`는 이벤트가 발생한 객체를 가리키는데, 우리는 지금 `Form` 클래스의 이벤트 처리기에 대해 알아보고 있으니 `sender`는 `Form` 객체 자신입니다. 만약 `Button` 클래스의 이벤트 처리기였다면 `Button` 객체였겠지요. 두 번째 매개 변수는 `EventArgs` 형식인데, 다음과 같은 프로퍼티들을 제공함으로써 마우스 이벤트의 상세 정보를 제공합니다.

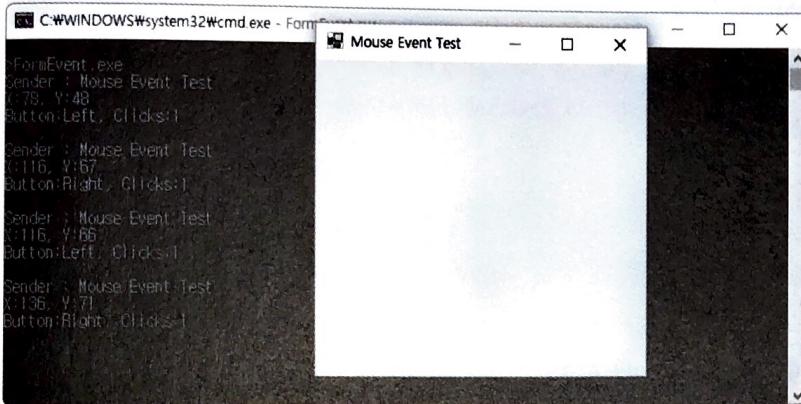
프로퍼티	설명
Button	마우스의 어떤 버튼(왼쪽, 오른쪽 또는 가운데)에서 이벤트가 발생했는지를 나타냅니다.
Clicks	마우스의 버튼을 클릭한 횟수를 나타냅니다. 사용자가 더블 클릭했을 때만 어떤 기능을 수행하고 싶다면 이 값이 2일 경우를 확인하면 됩니다.
Delta	마우스 휠의 회전 방향과 회전한 거리를 나타냅니다.
X	마우스 이벤트가 발생한 폼 또는 컨트롤 상의 x(가로) 좌표를 나타냅니다.
Y	마우스 이벤트가 발생한 폼 또는 컨트롤 상의 y(세로) 좌표를 나타냅니다.

이번에는 이벤트 처리기를 활용하는 예제 프로그램을 만들어 보겠습니다. 이 프로그램은 `Form` 클래스의 `MouseDown` 이벤트에 대한 이벤트 처리를 하는 프로그램입니다. 폼 위에서 마우스를 누를 때마다 이벤트를 발생시킨 객체(여기서는 `Form`), 마우스 버튼, 마우스 커서의 좌표 등을 콘솔에 출력합니다.

>>> 20장/FormEvent/MainApp.cs

```
01 using System;
02 using System.Windows.Forms;
03
04 namespace FormEvent
05 {
06     class MainApp : Form
07     {
08         public void MyMouseHandler(object sender, MouseEventArgs e)
09         {
10             Console.WriteLine($"Sender : {((Form)sender).Text}");
11             Console.WriteLine($"X:{e.X}, Y:{e.Y}");
12             Console.WriteLine($"Button:{e.Button}, Clicks:{e.Clicks}");
13             Console.WriteLine();
14         }
15
16         public MainApp(string title)
17         {
18             this.Text = title;
19             this.MouseDown +=
20                 new MouseEventHandler(MyMouseHandler);
21         }
22
23         static void Main(string[] args)
24         {
25             Application.Run(new MainApp("Mouse Event Test"));
26         }
27     }
28 }
```

실행 결과 :



20.4.2 Form의 프로퍼티를 조절하여 윈도우 모양 바꾸기

Form 클래스는 윈도우 모양을 결정짓는 크기, 배경색, 전경색, 투명도, 제목, 폰트 등 여러 가지 프로퍼티를 갖고 있습니다. 다음 표에는 Form 클래스의 프로퍼티 중 윈도우의 모습을 결정짓는 항목들이 나타나 있습니다(MSDN에는 더 많은 프로퍼티가 설명되어 있지만, 여기에서는 지면 문제 때문에 그 중에서 자주 사용하는 것들만 다룹니다).

종류	프로퍼티	설명
크기	Width	창의 너비를 나타냅니다.
	Height	창의 높이를 나타냅니다.
색깔	BackColor	창의 배경 색깔을 나타냅니다.
	BackgroundImage	창의 배경 이미지를 나타냅니다.
	Opacity	창의 투명도를 나타냅니다.
스타일	MaximizeBox	최대화 버튼을 설치할 것인지의 여부를 나타냅니다.
	MinimizeBox	최소화 버튼을 설치할 것인지의 여부를 나타냅니다.
	Text	창의 제목을 나타냅니다.

우리는 위 프로퍼티들 중에서 창의 크기를 결정하는 Width와 Height를 예제 프로그램을 통해 조정해보겠습니다. 다음 예제 프로그램은 창을 마우스 왼쪽 버튼으로 누르면 가로가 길게, 오른쪽 버튼으로 누르면 세로가 길게 크기를 바꿉니다.

>> 20장/FormSize/MainApp.cs

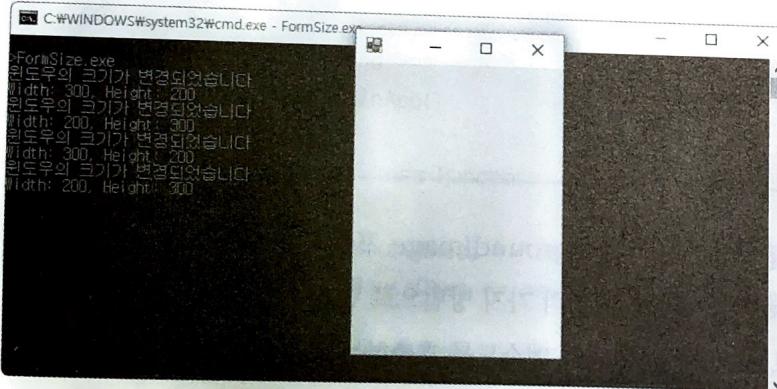
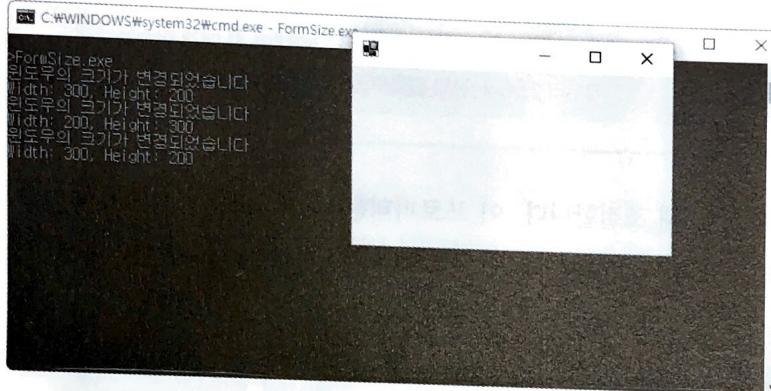
```
01 using System;
02 using System.Windows.Forms;
03
04 namespace FormSize
05 {
06     class MainApp : Form
07     {
08         static void Main(string[] args)
09         {
10             MainApp form = new MainApp();
11             form.Width = 300;
12             form.Height = 200;
13
14             form.MouseDown += new MouseEventHandler(form_MouseDown);
15
16             Application.Run(form);
17         }
18
19         static void form_MouseDown(object sender, MouseEventArgs e)
20         {
21             Form form = (Form)sender;
22             int oldWidth = form.Width;
23             int oldHeight = form.Height;
24
25             if (e.Button == MouseButtons.Left)
26             {
27                 if (oldWidth < oldHeight)
28                 {
29                     form.Width = oldHeight;
30                     form.Height = oldWidth;
31                 }
32             }
33             else if (e.Button == MouseButtons.Right)
34             {
35                 if (oldHeight < oldWidth)
36                 {
37                     form.Width = oldHeight;
38                     form.Height = oldWidth;
39                 }
39 }
```

```

40         }
41     }
42     Console.WriteLine("윈도우의 크기가 변경되었습니다");
43     Console.WriteLine($"Width: {form.Width}, Height: {form.Height}");
44 }
45 }

```

실행 결과 :



별 것 아니긴 하지만 사용자 인터페이스가 변하는 걸 보면 소소한 재미가 느껴지지 않습니까? 이번에는 창의 배경색과 투명도를 조절해보고, 더불어서 창의 배경 이미지도 바꿔보겠습니다.

창의 배경색은 BackColor 프로퍼티를 통해 바꿀 수 있는데, System.Drawing.Color 형식이기 때문에 다음과 같이 Color 클래스의 정적 메소드나 미리 정의되어 있는 상수값을 이용해서 값을 지정해야 합니다.

```
Form form = new Form();
```

```
form.BackColor = Color.Red;
```

Color 구조체에는 Red, Green, Blue 같은 대표적인 색부터 MistyRose, WhiteSmoke처럼 잘 알려지지 않은 색에 이르기까지 다양한 색상이 미리 정의되어 있습니다.

```
form.BackColor = Color.FromArgb(255, 255, 0, 0); // 불투명한 빨간색
```

프로그래머의 입맛에 맞춰 정확한 값을 지정하고 싶으면 FromArgb() 메소드를 이용합니다. 첫 번째 매개 변수는 투명도를 나타내는 Alpha, 두 번째는 Red, 세 번째는 Green, 네 번째는 Blue 값을 나타내며 각 매개 변수는 0부터 255 사이의 값을 가질 수 있습니다.

창의 투명도는 Opacity 프로퍼티를 통해 조절합니다. 이 프로퍼티는 double 형식으로 0.00부터 1.00 사이의 값을 가집니다. 0에 가까울수록 투명해지고, 1에 가까울수록 불투명해집니다. 사용 예는 다음과 같습니다.

```
Form form = new Form();
```

```
form.Opacity = 0.87; // 살짝 투명  
form.Opacity = 1.00; // 완전 불투명
```

창에 배경 이미지를 지정할 수도 있습니다. BackgroundImage 프로퍼티에 Image 형식의 인스턴스를 할당하면 됩니다. Image의 인스턴스는 여러 가지 방법으로 만들 수 있는데, 다음의 예제에서는 파일의 경로를 매개 변수로 넘겨FromFile() 메소드를 호출하는 방법을 보여줍니다.

```
Form form = new Form();
```

```
form.BackgroundImage = Image.FromFile( "MyImage.JPG" );
```

창의 색상, 투명도, 배경 이미지 등을 바꾸기 위한 기본 초식은 모두 익혔습니다. 지금까지 설명한 내용을 모아 프로그램으로 만들어 보겠습니다. 마우스의 왼쪽 버튼을 누르면 랜덤하게 창의 배경색을 변경하고 오른쪽 버튼을 누르면 배경 이미지를 표시하며, 창 위에서 마우스의 휠을 굴리면 투명

도가 변경되는 기능을 구현할 겁니다. 이번 장에서 쪽 해왔던 것처럼 '콘솔 앱(.NET Framework)' 템 플릿을 이용하여 프로젝트를 생성하고, System.Windows.Forms와 System.Drawing 어셈블리를 프로젝트 참조에 추가시키세요. 그리고 다음의 코드를 따라 입력한 뒤 컴파일해서 실행해보시길 바랍니다.

아차, 잠깐만요. 이 예제 프로그램은 이미지 파일 하나를 필요로 합니다. 프로그램을 테스트해보기 전에 여러분이 좋아하는 이미지를 sample.jpg라는 이름으로 저장해서 프로그램의 실행 파일과 같은 디렉토리에 복사해주세요.

»> 20장/FormBackground/MainApp.cs

```
01 using System;
02 using System.Drawing;
03 using System.Windows.Forms;
04
05 namespace FormBackground
06 {
07     class MainApp : Form
08     {
09         Random rand;
10         public MainApp()
11         {
12             rand = new Random();
13
14             this.MouseWheel += new MouseEventHandler(MainApp_MouseWheel);
15             this.MouseDown += new MouseEventHandler(MainApp_MouseDown);
16         }
17
18         void MainApp_MouseDown(object sender, MouseEventArgs e)
19         {
20             if (e.Button == MouseButtons.Left)
21             {
22                 Color oldColor = this.BackColor;
23                 this.BackColor = Color.FromArgb(rand.Next(0, 255),
24                                         rand.Next(0, 255),
25                                         rand.Next(0, 255));
26             }
27             else if (e.Button == MouseButtons.Right)
28             {
```

```

29         if (this.BackgroundImage != null)
30     {
31         this.BackgroundImage = null;
32         return;
33     }
34
35     string file = "sample.jpg";
36     if (System.IO.File.Exists(file) == false)
37         MessageBox.Show("이미지 파일이 없습니다.");
38     else
39         this.BackgroundImage = Image.FromFile(file);
40     }
41 }
42
43 void MainApp_MouseWheel(object sender, MouseEventArgs e)
44 {
45     this.Opacity = this.Opacity + ( e.Delta>0?0.1:-0.1 );
46     Console.WriteLine($"Opacity: {this.Opacity}");
47 }
48
49 static void Main(string[] args)
50 {
51     Application.Run(new MainApp());
52 }
53 }
54 }
```

실행 결과 :





지금쯤이면 여러분도 Form 클래스의 프로퍼티를 통해 창의 모양을 바꾸는 요령을 터득했을 겁니다. Form 클래스에는 창의 모습을 결정짓는 다양한 프로퍼티들이 많지만, 아쉬운 대로 최소화/최대화 버튼을 사라지게 하거나 나타나게 하는 MinimizeBox/MaximizeBox 프로퍼티와 창의 제목을 나타내는 Text 프로퍼티까지만 더 살펴보고 컨트롤 이야기로 넘어가겠습니다.

MinimizeBox와 MaximizeBox 프로퍼티는 boolean 형식으로, 버튼을 창에 표시하고자 할 때는 true를 입력하고 감추고자 할 때는 false를 입력합니다. 다음의 코드는 창에서 최대화 버튼은 표시하고 최소화 버튼은 감추도록 합니다.

```
Form form = new Form();
form.MaximizeBox = true;
form.MinimizeBox = false;
```

창의 제목을 나타내는 Text 프로퍼티는 string 형식입니다. 그냥 다음과 같이 표시하고자 하는 제목을 문자열로 입력해주면 창의 제목이 변경됩니다.

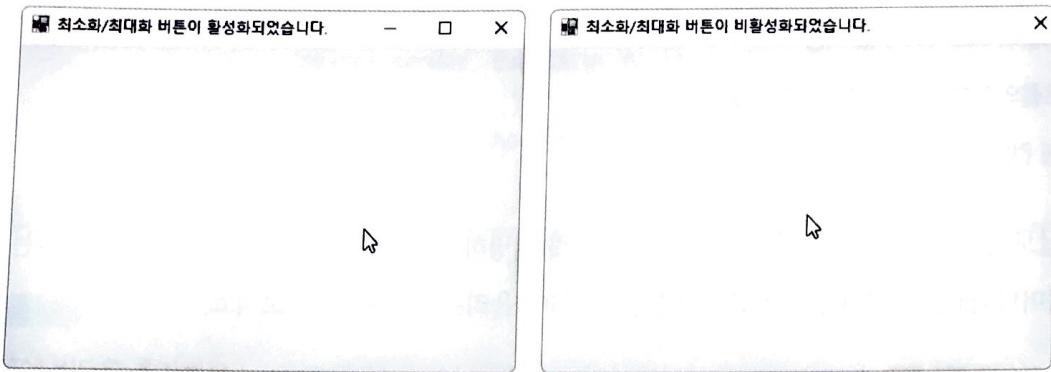
```
Form form = new Form();
form.Text = "Form 프로퍼티 테스트";
```

다음 예제 프로그램이 하나 더 준비되어 있습니다. 어떤 내용이 구현되어 있을지는 여러분도 예상할 수 있겠지요? 네, 애플리케이션 창의 최소화/최대화 버튼을 표시하고 감추는 기능과 창의 제목을 변경하는 기능이 구현되어 있습니다. 다음 코드를 입력한 뒤 컴파일해서 결과를 확인해보세요.

» 20장/FormStyle/MainApp.cs

```
01  using System;
02  using System.Windows.Forms;
03
04  namespace FormStyle
05  {
06      class MainApp : Form
07      {
08          static void Main(string[] args)
09          {
10              MainApp form = new MainApp();
11
12              form.Width = 400;
13              form.MouseDown += new MouseEventHandler(form_MouseDown);
14
15              Application.Run(form);
16          }
17
18          static void form_MouseDown(object sender, MouseEventArgs e)
19          {
20              Form form = (Form)sender;
21
22              if (e.Button == MouseButtons.Left)
23              {
24                  form.MaximizeBox = true;
25                  form.MinimizeBox = true;
26                  form.Text = "최소화/최대화 버튼이 활성화되었습니다.";
27              }
28              else if (e.Button == MouseButtons.Right)
29              {
30                  form.MaximizeBox = false;
31                  form.MinimizeBox = false;
32                  form.Text = "최소화/최대화 버튼이 비활성화되었습니다.";
33              }
34          }
35      }
36  }
```

실행 결과 :



20.4.3 Form 위에 컨트롤 올리기

사용자 인터페이스는 응용 프로그램과 사용자가 대화를 하는 창구입니다. 그런데 이번 장에서 지금 까지 만들어 온 예제 프로그램처럼 창 하나만 달랑 갖고 있어서는 사용자에게 어떤 정보도 제공할 수 없고 어떤 입력도 받을 수 없습니다.

윈도우 운영체제는 사용자 인터페이스를 위해 메뉴, 콤보박스, 리스트뷰, 버튼, 텍스트박스 등과 같은 표준 컨트롤을 제공합니다. .NET 프레임워크의 WinForm은 이들 표준 컨트롤을 아주 간편하게 창 위에 올릴 수 있도록 잘 포장해놨습니다. 이 컨트롤들을 제어하는 데 필요한 각종 메소드와 프로퍼티, 이벤트들이 잘 정리되어 있는 것은 더 말할 것도 없지요.

여기서 잠깐

★ 컨트롤이 뭔가요?

새로운 용어가 등장했지요? 컨트롤 Control이란, 윈도우 운영체제가 제공하는 사용자 인터페이스 요소를 말합니다. 응용 프로그램을 제어하는 데 사용하는 도구라는 의미에서 붙여진 이름이지요. 조금 전에 언급했던 버튼, 텍스트박스 등이 컨트롤의 예입니다. 유닉스의 모티프나 자바의 스윙 같은 GUI 플랫폼에서는 이것을 위젯 (Window Gadget을 줄여 Widget이라고 부르게 됐습니다)이라고 부르고 델파이에서는 VCL Visual Component Library라고 부릅니다.

어디 한번 컨트롤을 창 위에 올려볼까요? 우리의 맛밋한 애플리케이션 창을 바꿔보면 좋겠네요. 컨트롤을 폼 위에 올리려면 다음과 같은 과정을 따라 코드로 작성해주면 됩니다.

- ① 컨트롤의 인스턴스 생성
- ② 컨트롤의 프로퍼티에 값 지정
- ③ 컨트롤의 이벤트에 이벤트 처리기 등록
- ④ 폼에 컨트롤 추가

이렇게 글로 설명하는 것보단 아무래도 코드를 통해 설명하는 편이 이해에 도움이 되겠죠? 우리는 프로그래머니까요. 위 절차를 거치면서 버튼을 창 위에 올리는 코드를 같이 보시죠.

step 1

컨트롤의 인스턴스 생성

WinForm의 모든 컨트롤은 System.Windows.Forms.Control을 상속합니다. 이 형식이 모든 윈도우 컨트롤이 지원해야 하는 그래픽이나 동작, 이벤트 등을 제공하기 때문에 이 컨트롤로부터 상속받는 어떤 클래스도 Form 위에 올려서 윈도우 사용자 인터페이스 요소로 사용할 수가 있습니다. 우리는 버튼을 창 위에 올리기로 했지요? System.Windows.Forms.Button 클래스의 인스턴스를 만들겠습니다.

```
Button button = new Button();
```

step 2

컨트롤의 프로퍼티에 값 지정

인스턴스를 만들었으면 각 프로퍼티에 값을 지정해서 컨트롤의 모양을 결정합니다.

```
button.Text = "Click Me!";
button.Left = 100;
button.Top = 50;
```

step 3

컨트롤의 이벤트에 이벤트 처리기 등록

컨트롤은 애플리케이션의 정보를 표시하는 기능을 하기도 하지만 사용자로부터 입력을 받는 창구이기도 합니다. 사용자가 버튼을 클릭하면 메시지 박스를 띄우도록 이벤트 처리기를 선언하고 이벤트에 등록합니다. 다음의 코드는 지금까지 작성해왔던 이벤트 처리기와는 모습이 조금 다르지요? 설명의 흐름을 끊지 않기 위해 따로 이벤트 처리기를 메소드로 선언하지 않고 람다식으로 구현해봤습니다.

```
button.Click +=  
    (object sender, EventArgs e) =>  
    {  
        MessageBox.Show("딸깍!");  
    };
```

step 4

폼에 컨트롤 추가

이제 창에 버튼을 올릴 준비가 다 됐습니다. Form의 인스턴스를 생성하고, 이 인스턴스에서 Controls 프로퍼티의 Add() 메소드를 호출하여 우리가 선언한 button 객체를 Form에 올립니다.

```
MainApp form = new MainApp();  
form.Controls.Add(button);
```

이것이 끝입니다. 이제 이 코드를 실행하면 버튼을 가진 창이 나타나고, 그 버튼을 사용자가 클릭할 때마다 “딸깍!”이라는 메시지를 출력하는 메시지 박스가 나타날 겁니다. 다음 예제를 통해 직접 확인해보겠습니다.

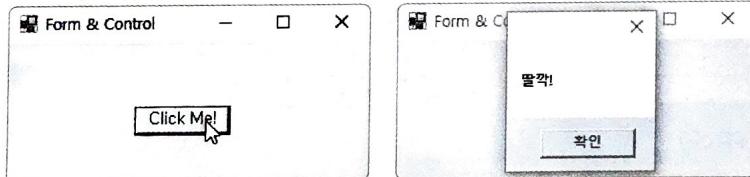
»> 20장/FormAndControl/MainApp.cs

```
01  using System;  
02  using System.Windows.Forms;  
03  
04  namespace FormAndControl  
05  {  
06      class MainApp : Form  
07      {  
08          static void Main(string[] args)  
09          {  
10              Button button = new Button();  
11  
12              button.Text = "Click Me!";  
13              button.Left = 100;
```

```

14         button.Top = 50;
15
16         button.Click +=
17             (object sender, EventArgs e) =>
18             {
19                 MessageBox.Show("딸깍!");
20             };
21
22         MainApp form = new MainApp();
23         form.Text = "Form & Control";
24         form.Height = 150;
25
26         form.Controls.Add(button);
27
28         Application.Run(form);
29     }
30 }
31 }
```

실행 결과 :

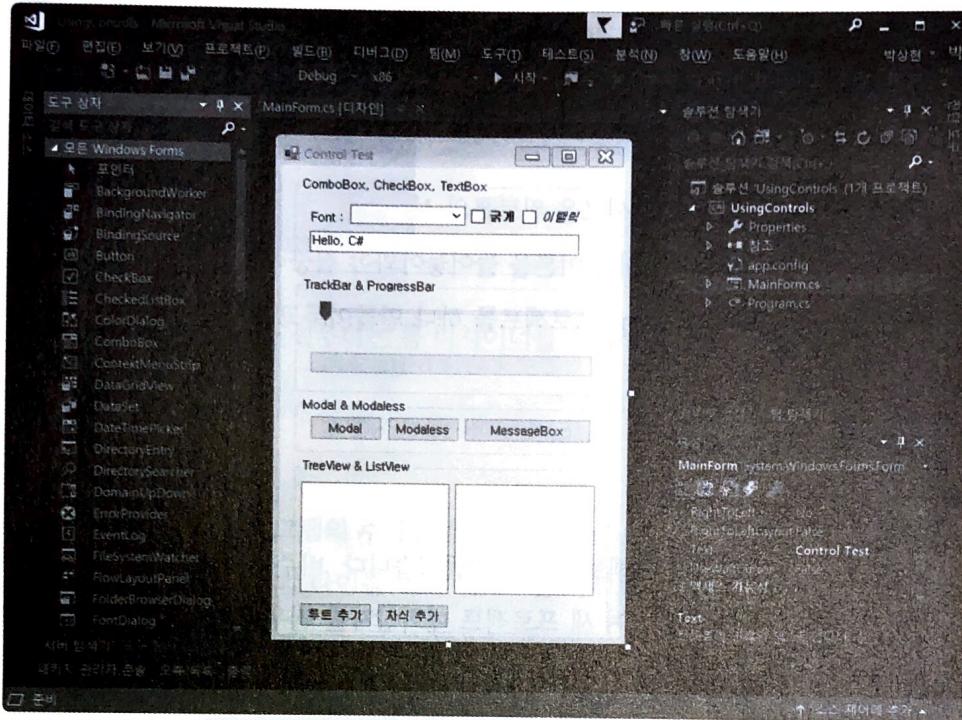


20.5 폼 디자이너를 이용한 WinForm UI 구성

비주얼 스튜디오는 UI계의 포토샵이라고 할만한, 사용하기는 쉬우면서도 강력한 폼 디자이너^{Form Designer}를 제공합니다. 폼 디자이너는 비주얼 스튜디오 IDE의 일부로, 우리가 코드를 통해 컨트롤을 폼 위에 배치하고 프로퍼티를 변경했던 작업을 마우스 클릭만으로 가능하게 해줍니다. 이러한 작업 편의성뿐 아니라, 폼 디자이너는 우리가 수작업을 통해 만든 것보다 품질이 우수한 UI를 제공합니다.

폼 디자이너는 새 프로젝트를 “Windows Form 응용 프로그램” 템플릿을 선택하여 만들면 나타납니다. 다음 그림에서 비주얼 스튜디오 IDE의 중앙에 있는 것이 바로 폼 디자이너입니다. 마치 실제

프로그램의 UI가 떠 있는 것처럼 보이지 않습니까? 하지만 저것은 실행 중인 프로그램이 아닙니다. 폼 디자이너로 디자인 중인 UI입니다. 이처럼 폼 디자이너는 WYSIWYG What You See Is What You Get 방식의 UI 디자인을 지원합니다.



폼 디자이너는 도구 상자(위 그림에서 IDE의 왼편에 있는 것이 도구 상자입니다)와 함께 사용해야 합니다. 이 도구 상자에는 WinForm에서 제공하는 수많은 컨트롤들을 담고 있어서 “컨트롤 팔레트 Control Palette”라는 이름으로 불리기도 합니다. 도구 상자와 폼 디자이너를 이용해서 UI를 구성하는 방법은 다음과 같습니다.

- ① 도구 상자에서 사용할 컨트롤을 골라 마우스 커서를 위치시키고 왼쪽 버튼을 클릭합니다.
- ② 마우스 커서를 그대로 폼 디자이너 위로 옮긴 뒤 다시 왼쪽 마우스 버튼을 클릭합니다.
- ③ 폼 위에 올려진 컨트롤의 위치 및 크기, 프로퍼티를 수정합니다.

이 순서가 글로 쓰여 있어서 폼 디자이너의 사용 방법이 어려워 보일 수도 있지만, 실제로 사용해보면 사용법이 굉장히 간단하다는 것을 느낄 수 있습니다. 실습은 잠시 후에 해보겠습니다.

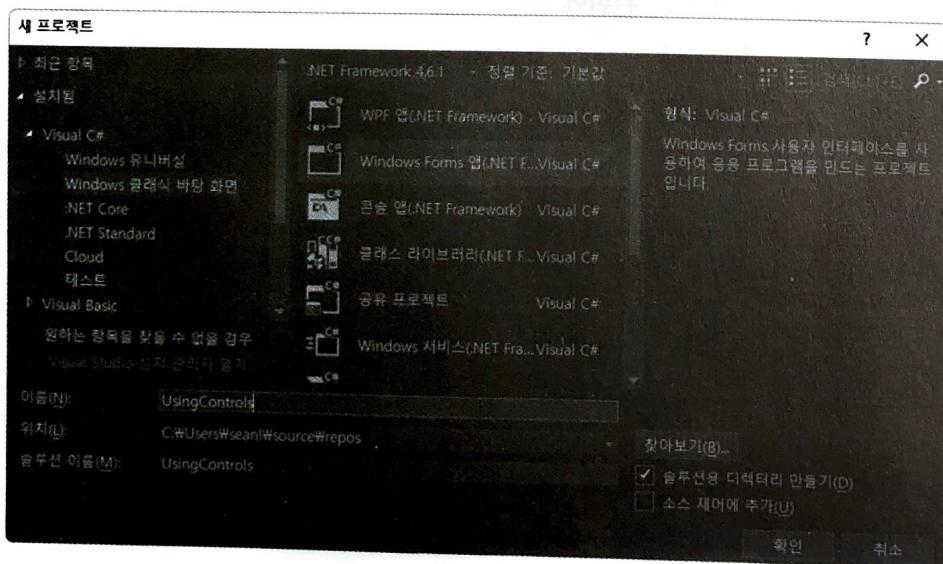
위의 그림에서 도구 상자 쪽을 유심히 본 독자들은 눈치챘을 텐데, WinForm은 앞에서 다뤘던 버튼

컨트롤 외에도 수많은 컨트롤들을 제공하고 있습니다. 이 컨트롤들을 이 책에서 모두 설명할 수 있으면 좋겠지만 안타깝게도 지면이 한정되어 있기 때문에(지면을 늘릴 수도 있지만, 그럼 다른 두 가지도 늘어납니다. 책의 가격과 여러분의 학습량이요) 저는 다소 아쉬운 선택을 하기로 했습니다. 몇 가지 컨트롤만 소개하기로 말입니다. 그나마 다행스러운 사실은 WinForm 컨트롤은 프로퍼티와 이벤트를 다루는 요령만 이해하면 낯선 컨트롤도 손쉽게 사용 방법을 익힐 수 있다는 것입니다. 따라서 이 책을 공부한 후 여러분이 필요에 의해 새로운 컨트롤들을 접하게 되더라도 별 어려움 없이 사용 방법을 익힐 수 있을 겁니다.

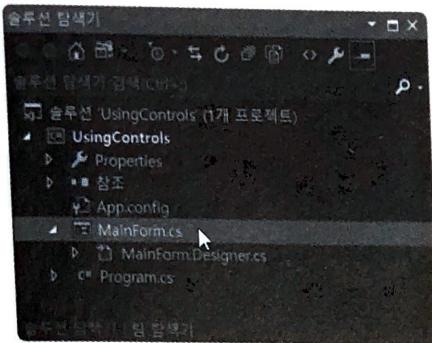
이어지는 절에서는 폼 디자이너의 사용법과 새로운 컨트롤의 사용법을 실습 위주로 설명합니다. 다양하면서도 소소한 내용들에 대해 백과사전처럼 이론을 늘어놓아봐야 집중력만 떨어지고 재미도 없을 것 같아서요. 그래서 일단 예제 WinForm 프로젝트를 하나 만들어두고, 이 프로젝트에 컨트롤을 조금씩 올려가면서 설명하겠습니다.

20.5.1 새 프로젝트 만들기

조금 전에 이야기한 것처럼 WinForm 프로젝트를 하나 만들겠습니다. 비주얼 스튜디오를 실행하고 [파일] → [새 프로젝트] 메뉴 항목을 클릭하여 새 프로젝트 대화상자를 띄우세요. 이 대화상자에서 템플릿으로는 “Windows Forms 앱”을 선택하고 이름에는 “UsingControls”를 입력한 후 <확인> 버튼을 클릭하세요. 창이 닫히면서 새 프로젝트가 만들어질 겁니다.



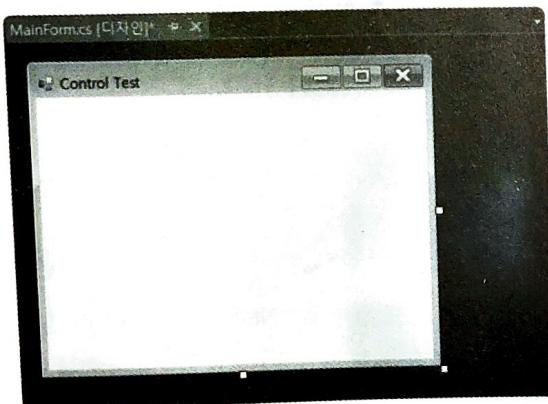
프로젝트가 생성됐으면 다음 그림과 같이 솔루션 탐색기에서 Form1.cs 파일의 이름을 MainForm.cs로 변경하세요.



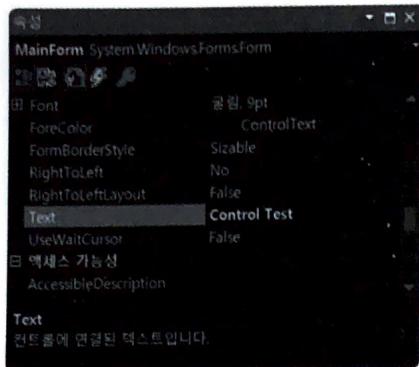
준비가 다 됐군요. 다음 절부터는 폼 디자이너를 이용해서 UI를 꾸며보겠습니다.

20.5.2 Form

우리가 만들 예제 프로그램의 주 윈도우가 될 MainForm의 속성을 “폼 디자이너를 이용해서” 변경해 보겠습니다. 윈도우의 타이틀 바의 텍스트만 바꿔보죠. 먼저 폼 디자이너에서 폼을 왼쪽 마우스로 한 번 선택하세요.



그럼 다음과 같이 속성 창(속성 창은 IDE의 우측 하단에 있습니다. 만약 속성 창이 IDE 어디에도 안 보인다면 [보기] → [속성 창]을 선택하면 나타날 겁니다)에서 'MainForm'의 프로퍼티와 값 목록을 출력할 겁니다. 이 속성 창에서 'Text' 프로퍼티를 찾아 'Control Test'로 값을 변경하세요.

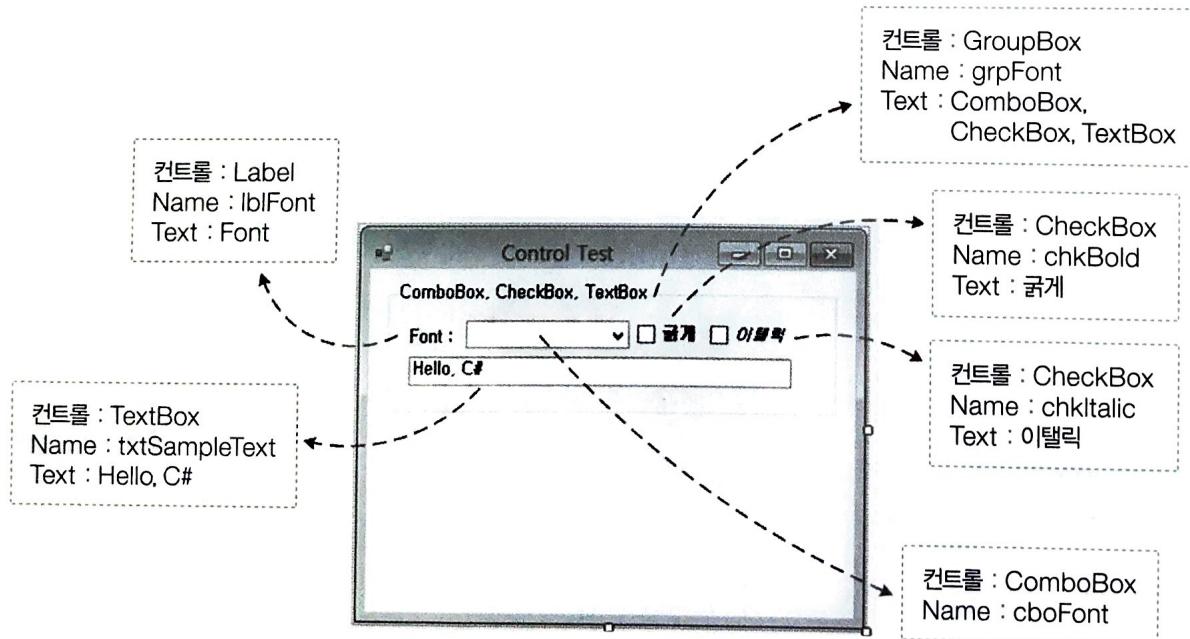


앞으로 이 폼 위에 올릴 컨트롤의 프로퍼티들도 폼에 대해 작업한 것과 같은 요령으로 변경하면 됩니다. 다음 절을 같이 볼까요?

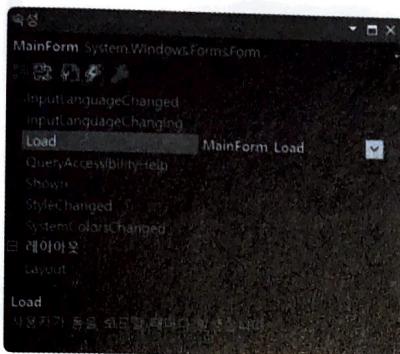
20.5.3 GroupBox, Label, ComboBox, CheckBox, TextBox

step 1

지금부터는 한번에 여러 개의 컨트롤을 폼 위에 배치하겠습니다. 다음 그림과 같이 컨트롤을 배치하고 프로퍼티를 변경하되, GroupBox 컨트롤부터 배치한 뒤 이 위에 나머지 컨트롤을 배치하시기 바랍니다.



컨트롤 배치를 마쳤으면 이젠 각 컨트롤에 이벤트 처리기를 등록할 차례입니다. 다른 컨트롤보다도 MainForm의 Load 이벤트에 대한 처리기를 먼저 등록하겠습니다. 이 이벤트가 다른 어떤 컨트롤의 이벤트보다도 가장 먼저 발생할 이벤트이기 때문입니다. 폼 디자이너에서 MainForm을 선택하고 다음 그림에서처럼 속성 창에서 <이벤트> 버튼(번개 아이콘)을 클릭해서 Form 컨트롤의 이벤트 목록을 여세요. 그리고 이벤트 목록에서 'Load' 항목을 찾아 더블 클릭하세요.



여기까지 작업했다면 IDE가 MainForm_Load() 이벤트 처리기의 템플릿을 만들면서 코드 편집창을 자동으로 열었을 겁니다. 이 템플릿에 다음과 같이 코드를 추가해서 이벤트 처리기를 완성해주세요.

```
private void MainForm_Load(object sender, EventArgs e)
{
    var Fonts = FontFamily.Families;           // 운영체제에 설치되어 있는 폰트 목록 검색
    foreach (FontFamily font in Fonts)          // cboFont 컨트롤에 각 폰트 이름 추가
        cboFont.Items.Add(font.Name);
}
```

step 2

코드 편집창을 연 김에 다음 메소드도 MainForm 클래스 안에 추가해주세요. 이 메소드는 cboFont와 chkBold, chkItalic 컨트롤의 이벤트 처리기에서 호출하기 위한 것으로, txtSampleText의 문자열의 폰트를 변경하는 기능을 합니다.

```

void ChangeFont()
{
    if (.cboFont.SelectedIndex < 0) // cboFont에서 선택한 항목이 없으면 메소드 종료
        return;

    FontStyle style = FontStyle.Regular; // FontStyle 객체를 초기화합니다.

    if (chkBold.Checked) // "굵게" 체크 박스가 선택되어 있으면 Bold 논리합 수행
        style |= FontStyle.Bold;

    if (chkItalic.Checked) // "이탤릭" 체크 박스가 선택되어 있으면 Italic 논리합 수행
        style |= FontStyle.Italic;

    txtSampleText.Font = // txtSampleText의 Font 프로퍼티를 앞에서 만든 style로 수정
        new Font((string)cboFont.SelectedItem, 10, style);
}

```

step 3

다음 표에 있는 각 컨트롤에 대해 이벤트 처리기 캡데기를 만들어 주세요. 요령은 MainForm_Load() 이벤트 처리기를 만들 때와 같습니다.

컨트롤	이벤트	이벤트 처리기
cboFont	SelectedIndexChanged	cboFont_SelectedIndexChanged
chkBold	CheckedChanged	chkBold_CheckedChanged
chkItalic	CheckedChanged	chkItalic_CheckedChanged

이벤트 처리기 캡데기들을 만들었으면 다음과 같이 코드를 추가해서 완성하세요. 세 개의 이벤트 처리기 모두 동일하게 조금 전에 선언한 ChangeFont() 메소드를 호출합니다.

```

private void cboFont_SelectedIndexChanged(object sender, EventArgs e)
{
    ChangeFont();
}

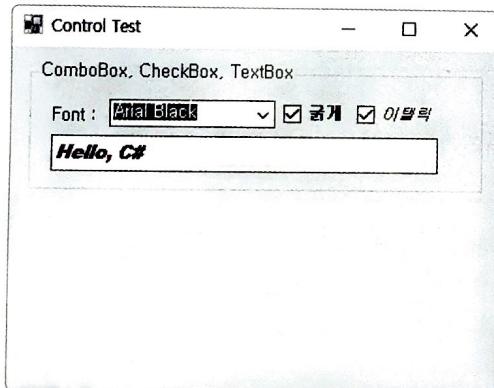
private void chkBold_CheckedChanged(object sender, EventArgs e)
{
    ChangeFont();
}

private void chkItalic_CheckedChanged(object sender, EventArgs e)
{
    ChangeFont();
}

```

step 4

지금까지 작업한 내용을 테스트해보겠습니다. [F5] 키를 눌러 프로그램을 디버깅 모드로 실행해서 테스트해보세요. cboFont 콤보 박스에 폰트 목록이 잘 나오는지, 폰트를 선택하면 txtSampleText의 텍스트 폰트가 변경되는지를 보면 됩니다. “굵게” 체크박스와 “이탤릭” 체크박스의 텍스트도 잊으면 안됩니다.

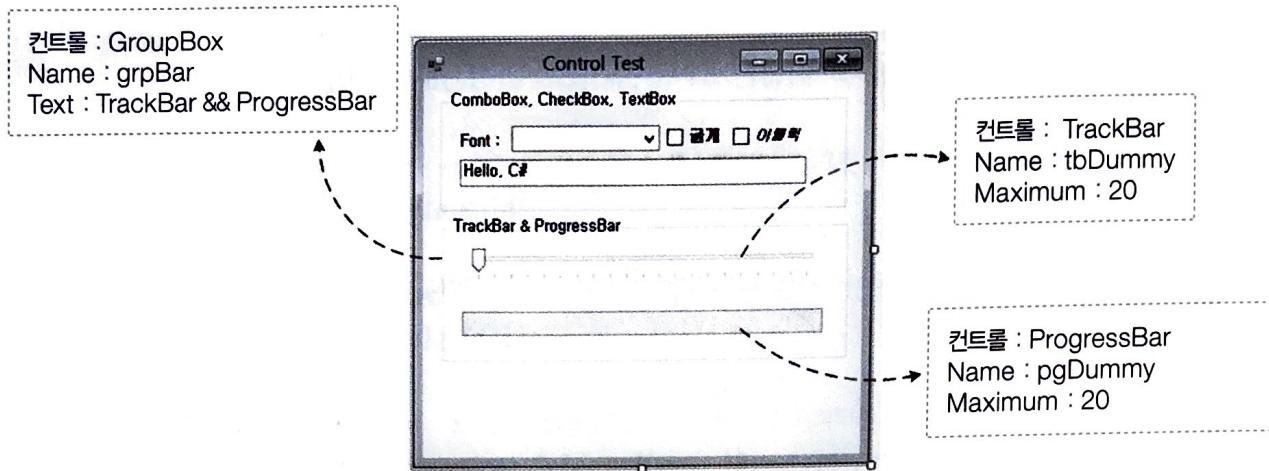


20.5.4 TrackBar, ProgressBar

이번엔 TrackBar와 ProgressBar 컨트롤을 테스트해보겠습니다.

step 1

다음 그림과 같이 컨트롤을 배치하고 프로퍼티를 변경하되, 아까처럼 GroupBox 컨트롤부터 배치한 뒤 이 위에 나머지 컨트롤을 배치하시기 바랍니다.



step 2

다음 표에 있는 각 컨트롤에 대해 이벤트 처리기 캡테기를 만들어 주세요. 요령은 MainForm_Load() 이벤트 처리기를 만들 때와 같습니다.

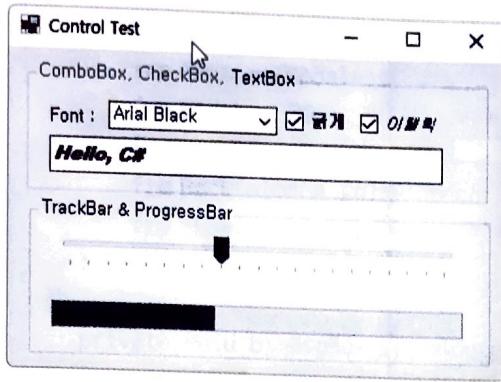
컨트롤	이벤트	이벤트 처리기
tbDummy	Scroll	tbDummy_Scroll

다음과 같이 코드를 입력하여 tbDummy_Scroll() 이벤트 처리기를 완성하세요.

```
private void tbDummy_Scroll(object sender, EventArgs e)
{
    pgDummy.Value = tbDummy.Value; // 슬라이더의 위치에 따라 프로그레스바의 내용도 변경
}
```

step 3

작업한 내용을 확인해보겠습니다. **F5** 키를 눌러 프로그램을 디버깅 모드로 실행하고 TrackBar의 슬라이더를 이리저리 옮기면서 ProgressBar의 내용도 따라 변경되는지 확인해보세요.



20.5.5 Button, Form, Dialog

Button 컨트롤은 이미 구면이죠? 그래도 폼 디자이너로는 Button 컨트롤을 배치해본 적 없으니 처음 보는 것처럼 읽어주세요. 그래야 덜 지루하죠. 이번에는 Button 컨트롤을 클릭했을 때 Modal 창, Modaless 창, MessageBox를 띄우는 기능을 구현해보겠습니다.

여기서 잠깐

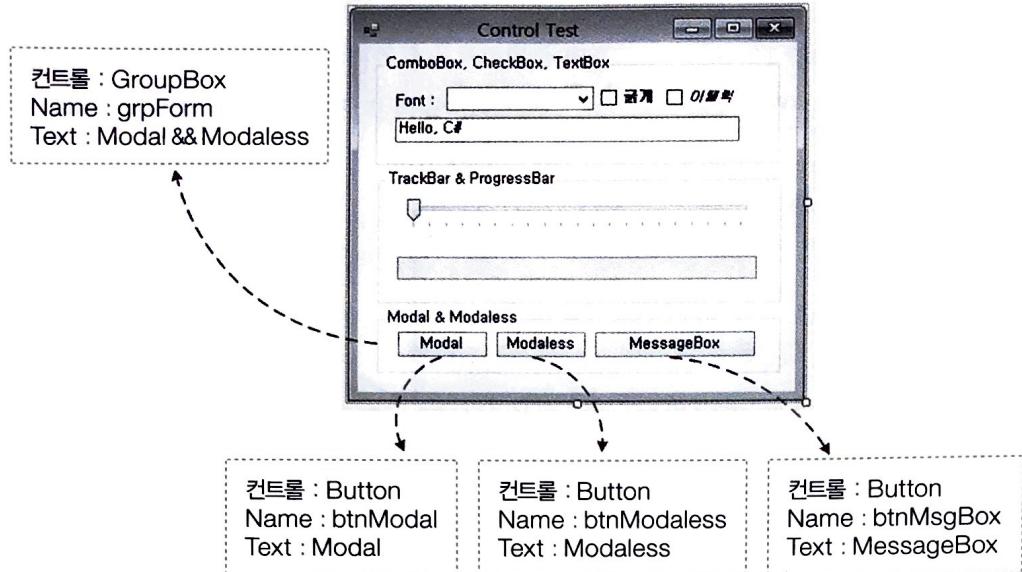
Modal? Modaless?

윈도우 프로그램은 두 가지 모드 자식 창을 띄울 수 있는데 그 중 한 가지가 Modal, 또 다른 한 가지가 Modaless입니다. Modal 창은 일단 띄우고 나면 창을 닫을 때까지 프로그램의 다른 UI를 절대 사용할 수 없다는 것이 특징입니다. 프로그램이 심각한 정보를 표시해야하거나 사용자로부터 중요한 결정 사항을 입력받아 다음 단계를 진행해야 할 때 주로 사용하곤 합니다.

이외는 달리 Modaless 창은 띄우고 난 뒤에도 프로그램의 다른 UI에 사용자가 접근할 수 있습니다. 웹 브라우저의 파일 다운로드 창이 Modaless 창의 좋은 예죠. 파일을 다운로드하면 자식 창이 떠서 파일 다운로드를 수행하지만 사용자는 여전히 웹 브라우저로 다른 페이지를 탐색할 수 있거든요.

step 1

다음 그림과 같이 컨트롤을 배치하고 프로퍼티를 변경하세요. 요령은 이전과 동일합니다.



step 2

다음 표와 같이 각 버튼에 대해 이벤트 처리기 캡데기를 만들어 주세요.

컨트롤	이벤트	이벤트 처리기
btnModal	Click	btnModal_Click
btnModaless	Click	btnModaless_Click
btnMsgBox	Click	btnMsgBox_Click

이벤트 처리기 캡데기를 만들었으면 다음과 같이 코드를 입력하여 완성하세요.

```

private void btnModal_Click(object sender, EventArgs e)
{
    Form frm = new Form();
    frm.Text = "Modal Form";
    frm.Width = 300;
    frm.Height = 100;
    frm.BackColor = Color.Red;
    frm.ShowDialog(); // Modal 창을 띄웁니다.
}

private void btnModaless_Click(object sender, EventArgs e)
{
}

```

```

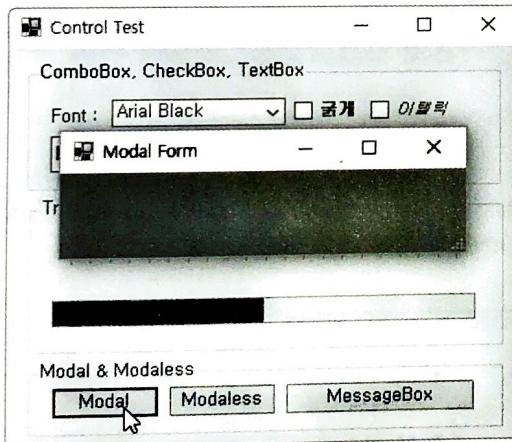
        Form frm = new Form();
        frm.Text = "Modaless Form";
        frm.Width = 300;
        frm.Height = 300;
        frm.BackColor = Color.Green;
        frm.Show(); // Modaless 창을 띄웁니다.
    }

    private void btnMsgBox_Click(object sender, EventArgs e)
    {
        MessageBox.Show(txtSampleText.Text,
            "MessageBox Test", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }

```

step 3

작업한 내용을 확인해보겠습니다. **F5** 키를 눌러 프로그램을 디버깅 모드로 실행하세요. Modal 창을 띄웠을 때 제가 설명한 대로 프로그램의 다른 UI를 사용할 수 없는지, 또는 Modaless 창을 띄웠을 때 프로그램의 다른 UI를 사용할 수 있는지 확인해보길 바랍니다.

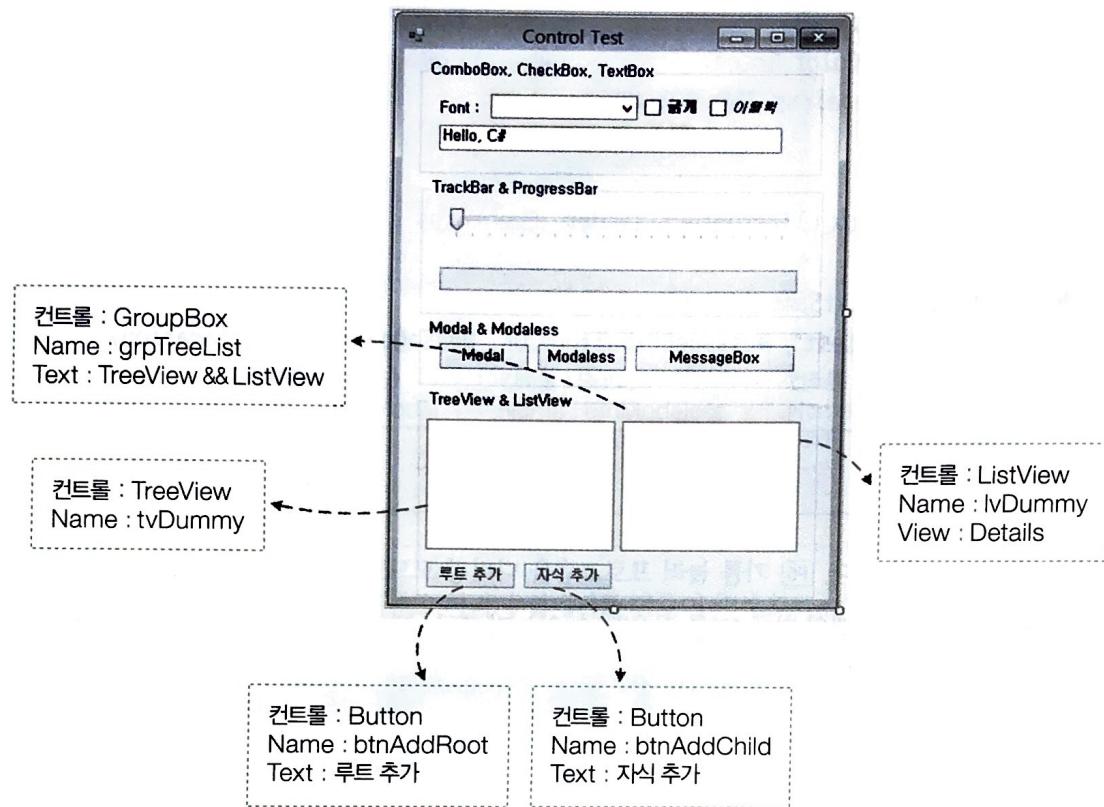


20.5.6 TreeView, ListView

마지막으로 윈도우 탐색기에서 항상 볼 수 있는 TreeView와 ListView 컨트롤을 사용해보겠습니다.

step 1

먼저 다음 그림과 같이 컨트롤을 배치하고 프로퍼티를 변경하세요.



step 2

MainForm.cs를 코드 편집기로 열어서 다음과 같이 필드 하나를 추가해 주세요. TreeView의 노드 이름으로 사용할 난수 생성기입니다.

```
public partial class MainForm : Form
{
    Random random = new Random(37);

    // ...
}
```

step 3

MainForm() 생성자에도 추가할 코드가 있습니다. 다음과 같이 lvDummy에 컬럼을 생성하는 코드를 입력해주세요.

```
public MainForm()
{
    InitializeComponent();

    lvDummy.Columns.Add("Name");
    lvDummy.Columns.Add("Depth");
}
```

step 4

코드 편집기를 연 김에 다음의 TreeToList() 메소드도 추가합시다. 이 메소드는 TreeView의 각 노드를 ListView로 옮겨 표시하는 기능을 합니다.

```
void TreeToList()
{
    lvDummy.Items.Clear();
    foreach (TreeNode node in tvDummy.Nodes)
        TreeToList(node);
}

void TreeToList(TreeNode Node)
{
    lvDummy.Items.Add(
        new ListViewItem(
            new string[] { Node.Text,
                Node.FullPath.Count(f => f == '\\').ToString() }));
}

foreach (TreeNode node in Node.Nodes)
{
    TreeToList(node);
}
}
```

TreeNode 형식의 FullPath 프로퍼티는
루트 노드부터 현재 노드까지의 경로를
나타내며, 각 경로는 '\'로 구분합니다.

step 5

다시 폼 디자이너로 돌아와서, 다음 표와 같이 <루트 추가> 버튼과 <자식 추가> 버튼에 대해 이벤트 처리기
껍데기를 만들어 주세요.

컨트롤	이벤트	이벤트 처리기
btnAddRoot	Click	btnAddRoot_Click
btnAddChild	Click	btnAddChild_Click

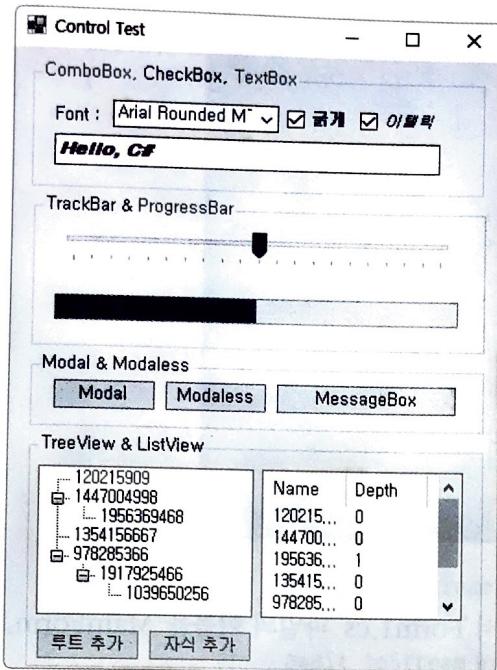
이벤트 처리기 콜데기를 만들었으면 다음과 같이 코드를 입력하여 완성하세요.

```
private void btnAddRoot_Click(object sender, EventArgs e)
{
    tvDummy.Nodes.Add(random.Next().ToString());
    TreeToList();
}

private void btnAddChild_Click(object sender, EventArgs e)
{
    if (tvDummy.SelectedNode == null)
    {
        MessageBox.Show("선택된 노드가 없습니다.",
            "TreeView Test", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }
    tvDummy.SelectedNode.Nodes.Add(random.Next().ToString());
    tvDummy.SelectedNode.Expand();
    TreeToList();
}
```

step 6

작업한 내용을 확인해야겠지요? **F5** 키를 눌러 프로그램을 디버깅 모드로 실행하세요. <루트 추가> 버튼과 <자식 추가> 버튼을 눌러 TreeView에 노드를 생성하고, TreeView에 생성된 노드들이 ListView에도 표시되는지 확인해보세요.



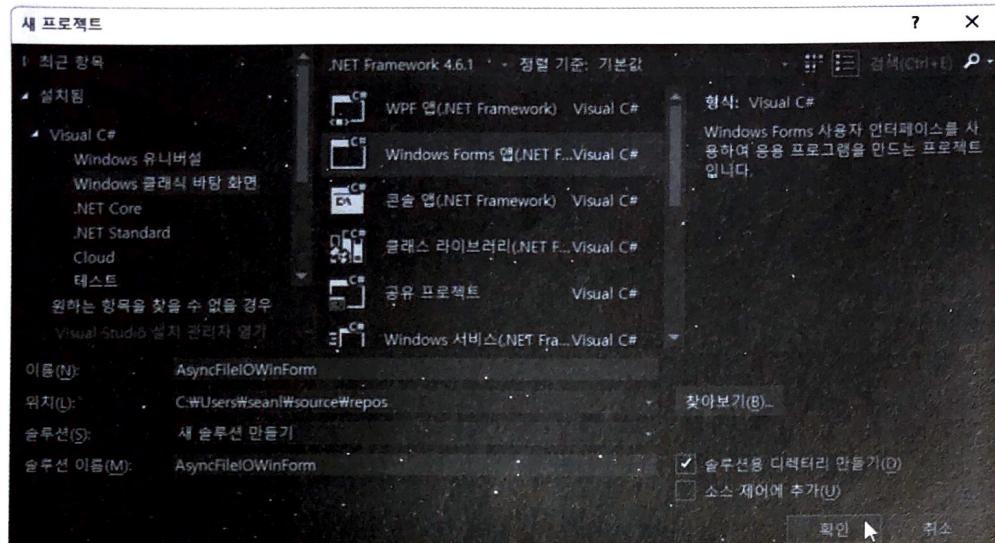
20.6 사용자 인터페이스와 비동기 작업

백그라운드 작업을 수행하면서도 사용자에게 여전히 잘 응답하는 프로그램을 만들기는 쉽지만 한일은 아니었습니다. `async`와 `await`가 C#에 도입되기 전까지는 말입니다. 19장에서 비동기 코드를 설명하면서 `async`와 `await`의 진가를 20장에서 확인하자고 했던 약속을 지킬 때가 왔습니다.

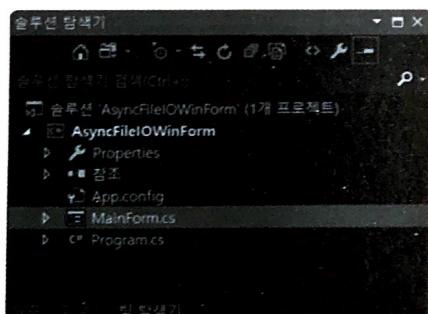
지금부터는 파일 복사를 하는 윈도우 프로그램을 만들어 보겠습니다. 이 프로그램의 동기/비동기 파일 복사 코드는 19장에서 이미 다뤘고, 사용자 인터페이스 구성 또한 간단하므로 동작 원리에 대한 별도의 설명은 생략하겠습니다. 다음 순서를 따라 예제 프로젝트를 생성하고 코드를 작성해보세요.

20.6.1 새 프로젝트 만들기

비주얼 스튜디오를 실행하고 [파일] → [새 프로젝트] 메뉴 항목을 클릭하여 새 프로젝트 대화 상자를 띄우세요. 이 대화상자에서 템플릿으로는 'Windows Forms 앱'을 선택하고 '이름'에는 'AsyncFileIOWinForm'을 입력한 후 <확인> 버튼을 클릭하세요. 창이 닫히면서 새 프로젝트가 만들어집니다.



프로젝트가 생성됐으면 다음 그림과 같이 솔루션 탐색기에서 'Form1.cs' 파일의 이름을 'MainForm.cs'로 변경하세요.



20.6.2 CopySync() / CopyAsync() 메소드 구현하기

동기 파일 복사를 하는 `CopySync()`와 비동기 파일 복사를 하는 `CopyAsync()`의 내용은 19장에서 만들었던 예제 프로그램과 내용이 거의 같습니다. 파일 복사 상태를 프로그래스바로 표시하는 부분만 다를 뿐입니다. 다음 `CopySync()` / `CopyAsync()` 코드를 `MainForm.cs`에 추가하세요.

```

private async Task<long> CopyAsync(string FromPath, string ToPath)
{
    btnSyncCopy.Enabled = false;
    long totalCopied = 0;

    using (FileStream fromStream =
        new FileStream(FromPath, FileMode.Open))
    {
        using (FileStream toStream =
            new FileStream(ToPath, FileMode.Create))
        {
            byte[] buffer = new byte[1024 * 1024];
            int nRead = 0;
            while ((nRead =
                await fromStream.ReadAsync(buffer, 0, buffer.Length)) != 0)
            {
                await toStream.WriteAsync(buffer, 0, nRead);
                totalCopied += nRead;

                // 프로그레스바에 현재 파일 복사 상태 표시
                pbCopy.Value =
                    (int)((double)totalCopied / (double)fromStream.Length)
                    * pbCopy.Maximum;
            }
        }
    }

    btnSyncCopy.Enabled = true;
    return totalCopied;
}

private long CopySync(string FromPath, string ToPath)
{
    btnAsyncCopy.Enabled = false;
    long totalCopied = 0;

    using (FileStream fromStream =
        new FileStream(FromPath, FileMode.Open))
    {
        using (FileStream toStream =

```

```

        new FileStream(ToPath, FileMode.Create))
    {
        byte[] buffer = new byte[1024 * 1024];
        int nRead = 0;
        while ((nRead =
            fromStream.Read(buffer, 0, buffer.Length)) != 0)
        {
            toStream.Write(buffer, 0, nRead);
            totalCopied += nRead;

            // 프로그레스바에 현재 파일 복사 상태 표시
            pbCopy.Value =
                (int)((double)totalCopied / (double)fromStream.Length)
                    * pbCopy.Maximum);
        }
    }

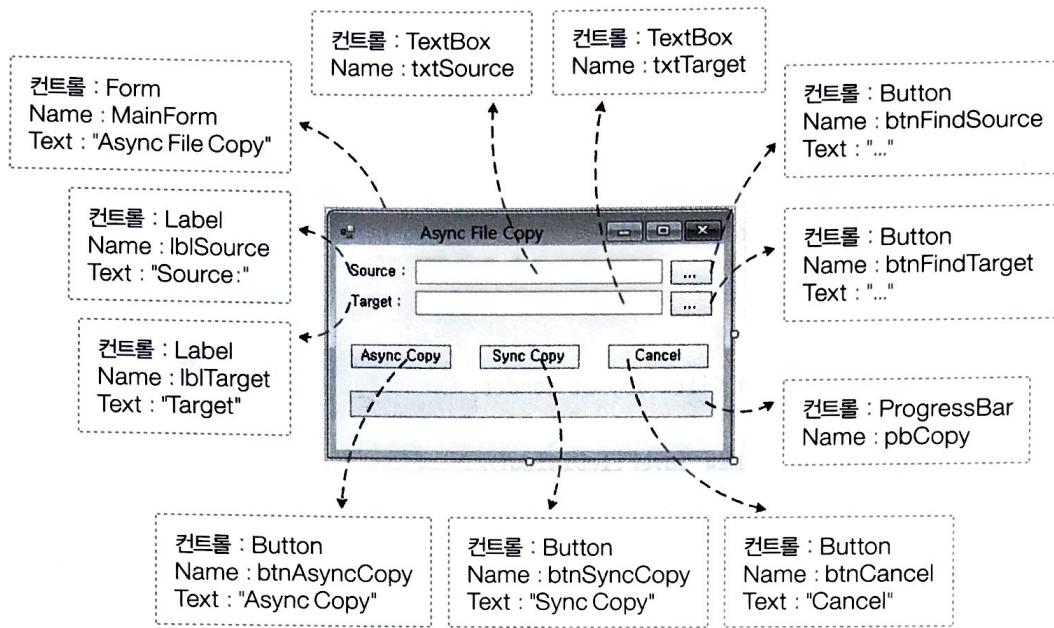
    btnAsyncCopy.Enabled = true;
    return totalCopied;
}

```

20.6.3 UI 구성하기

step 1

이제 폼 디자이너를 이용해서 UI를 구성할 차례입니다. MainForm 위에 다음 그림과 같이 컨트롤을 배치하고 프로퍼티를 변경하세요.



step 2

다음 표에 있는 각 컨트롤에 대해 이벤트 처리기 캡데기를 만들어 주세요.

컨트롤	이벤트	이벤트 처리기
btnFindSource	Click	btnFindSource_Click
btnFindTarget	Click	btnFindTarget_Click
btnAsyncCopy	Click	btnAsyncCopy_Click
btnSyncCopy	Click	btnSyncCopy_Click
btnCancel	Click	btnCancel_Click

이벤트 처리기 캡데기를 만들었으면 다음과 같이 코드를 입력하여 완성하세요.

```
private void btnFindSource_Click(object sender, EventArgs e)
{
    OpenFileDialog dlg = new OpenFileDialog();
    if (dlg.ShowDialog() == System.Windows.Forms.DialogResult.OK)
    {
        txtSource.Text = dlg.FileName;
    }
}
```

```

private void btnFindSource_Click(object sender, EventArgs e)
{
    OpenFileDialog dlg = new OpenFileDialog();
    if (dlg.ShowDialog() == System.Windows.Forms.DialogResult.OK)
    {
        txtSource.Text = dlg.FileName;
    }
}

private void btnFindTarget_Click(object sender, EventArgs e)
{
    SaveFileDialog dlg = new SaveFileDialog();
    if (dlg.ShowDialog() == System.Windows.Forms.DialogResult.OK)
    {
        txtTarget.Text = dlg.FileName;
    }
}

private async void btnAsyncCopy_Click(object sender, EventArgs e)
{
    long totalCopied = await CopyAsync(txtSource.Text, txtTarget.Text);
}

private void btnSyncCopy_Click(object sender, EventArgs e)
{
    long totalCopied = CopySync(txtSource.Text, txtTarget.Text);
}

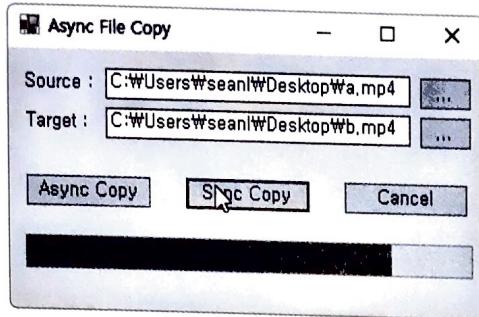
private void btnCancel_Click(object sender, EventArgs e)
{
    MessageBox.Show("UI 반응 테스트 성공.");
}

```

step 3

코딩은 끝났습니다. 이제 **F5** 키를 눌러서 프로그램을 테스트해봅시다. 먼저 <btnFindSource> 버튼을 클릭해서 복사 원본 파일을 찾고(동영상 같이 큰 파일이 테스트에 유리합니다), <btnFindTarget> 버튼을 클릭해서 복사 대상 파일을 입력한 후 <Sync Copy> 버튼을 클릭하세요. 다음과 같이 파일을 복사하고 있을

때 <Cancel> 버튼을 눌러 UI가 반응하는지 확인해보세요. 거의 반응하지 못할 겁니다(아쉽게도 다음 그림은 인쇄물의 한계로 UI가 반응하지 못하는 상황을 잘 표현하지 못하고 있습니다).



파일 복사가 끝났으면 이번엔 <Async Copy> 버튼을 클릭해서 파일 복사를 시작하고, 파일 복사가 진행 중 일 때 <Cancel> 버튼을 클릭해서 사용자 인터페이스가 반응하는지 보세요.

