C#

5주차 1차시

10장 고급 폼

메뉴 다루기

5주차 학습 내용

* 1차시
  * 10장 고급 폼
    * 메뉴 다루기
    * 4장 델리게이트, 이벤트
* 2차시
  * 10장 고급 폼
    * 마우스 이벤트 다루기
* 3차시
  * 10장 고급 폼
    * 키보드 이벤트 다루기
    * 5주차 과제

[클래스 멤버의 종류]
-자료 멤버
필드, 상수, 이벤트

-함수멤버
메서드, 생성자, 소멸자, 프러퍼티 (속성), 인덱서(색인자), 연산자 중복

-내포형 멤버
클래스형, 구조형, 열거형, 인터페이스형, 대리(델리게이트)형

[객체 지향 프로그램 구성]
-클래스 정의
-객체 선언
-객체 생성
-객체의 멤버 접근 및 활용

C# 프로그래밍

제 10장 고급 폼
메뉴 다루기

목차

- 메뉴 다루기
- 이벤트 핸들러
- 델리케이트
- 마우스 이벤트 다루기
- 키보드 이벤트 다루기

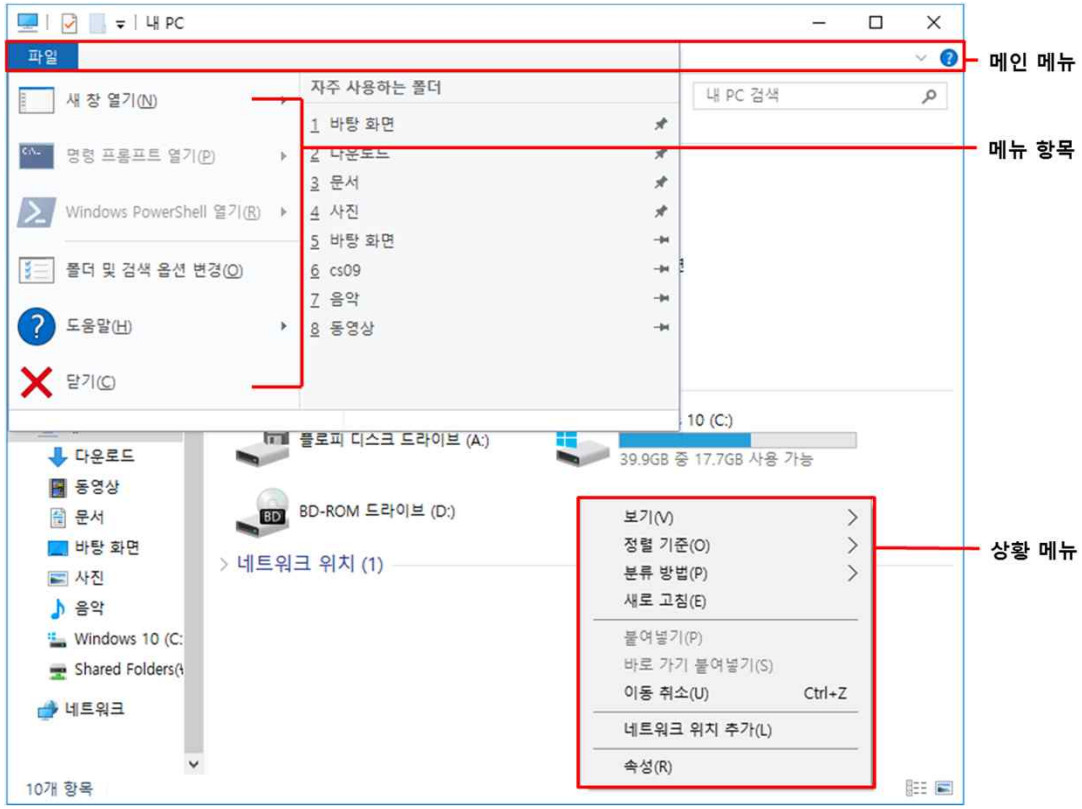
메뉴 다루기 [1/2]

- 메뉴
  - 윈폼 애플리케이션에서 가장 일반적인 사용자 인터페이스
  - 윈폼 애플리케이션이 제공하는 기능을 사용자가 쉽게 이해하고 사용할 수 있도록 도와주는 기능

- 메뉴의 종류
  - 메인메뉴 (main menu)
    - 폼의 상단에 배치되는 주요 메뉴
  - 상황메뉴 (context menu)
    - 마우스 상황메뉴 (context menu오른쪽 버튼을 클릭했을 때 나타나는 팝업 메뉴

메뉴 다루기 [2/2]

- 메뉴의 구성



메인 메뉴

- 폼의 상단에 배치되는 메뉴
- 마우스 클릭뿐만 아니라 단축키를 통해서는 접근할 수 있는 가장 기본적인 사용자 인터페이스
- 통합 개발 환경의 MenuStrip 컴포넌트를 통하여 작성

메인 메뉴의 작성 [1/2]

- 메뉴 항목의 추가
  - 메뉴에 단축문자를 부여하기 위한 방법
    - 사용할 단축문자 앞에 &를 붙임
    - <Alt>키와 단축문자를 눌러서 메뉴의 선택이 가능
- 메뉴 항목의 단축키 적용
  - 단축키를 적용할 메뉴 항목을 선택
- 구분선
  - 메뉴 항목을 그룹화하기 위하여 구분선을 사용
  - 메뉴 항목에 '-'를 입력

메인 메뉴의 작성 [2/2]

예제 10.1 [ex10_1_xxxx] : 메인 메뉴

메뉴 항목의 프로퍼티 정보를 참고하여 MainMenu를 작성한다.

Text 프로퍼티		Shortcut 프로퍼티
파일(&F)		
	새 파일(&N)	CtrlN(Control N)
	열기(&O)...	CtrlO(Control O)
	닫기(&C)	
	저장(&S)	CtrlS(Control S)
	다른 이름으로 저장(&A)...	
	-	
	인쇄(&P)...	CtrlP(Control P)
	미리 보기(&V)	
	-	
	종료(&X)	
편집(&E)		
	잘라내기(&T)	CtrlX(Control X)
	복사(&C)	CtrlC(Control C)
	붙여넣기(&P)	CtrlV(Control V)
도움말(&H)		
	프로그램 정보(&A)...	

메뉴 항목의 이벤트 [1/4]

- 메뉴 항목을 클릭하면 발생하는 이벤트
  - Click
    - 메뉴 항목을 클릭했을 때 발생
    - 메뉴와 관련된 이벤트 중에서 가장 많이 사용하는 이벤트

메뉴 항목의 이벤트 [2/4]

- 메뉴 항목과 관련된 Form 객체의 이벤트
  - MenuStart
    - 메뉴가 처음으로 입력 포커스를 얻을 때 발생
    - 폼의 사용자 인터페이스를 관리하기 위해서 사용
  - MenuComplete
    - 메뉴가 입력 포커스를 잃을 때 발생
    - 메뉴가 사라지는 순간을 확인하기 위해서 사용

메뉴 항목의 이벤트 [3/4]

예제 10.2 [ex10_2_xxxx] : 메인 메뉴

1) 디자인

컴포넌트 : (Name)	프로퍼티	값
MenuStrip : menuStrip1		

컨트롤 : (Name)	프로퍼티	값
Form : Form1	Text	MenuClickApp
Button : button1	Text	Clear
Button : button2	Text	Exit
ListBox : listBox1		

컨트롤 : (Name)	이벤트	메소드명
Button : button1	Click	button1_Click()
Button : button2	Click	button2_Click()
MenuItem : mnuFileNew	Click	mnuFileNew_Click()
MenuItem : mnuFileOpen	Click	mnuFileOpen_Click()
…		

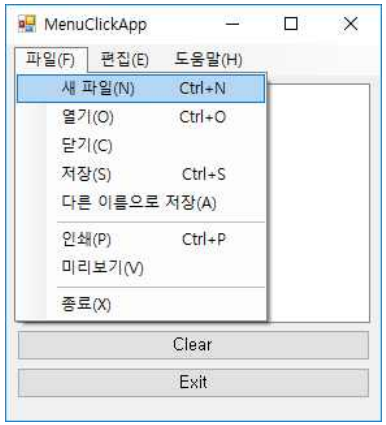
메뉴 항목의 이벤트 [4/4]

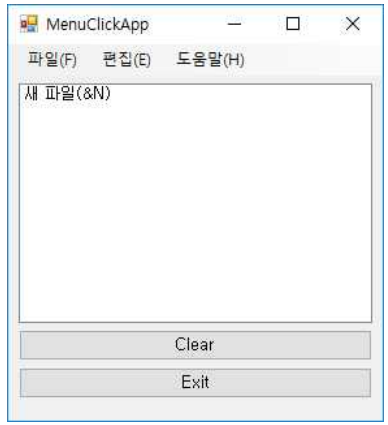
2) 코드

```
private void mnuFileNew_Click(object sender, EventArgs e) {
    listBox1.Items.Add(mnuFileNew.Text);
}
private void mnuFileOpen_Click(object sender, EventArgs e) {
    listBox1.Items.Add(mnuFileOpen.Text);
}
// 나머지 메뉴 항목도 동일하게 작성
private void button1_Click(object sender, EventArgs e) {
    listBox1.Items.Clear();
}
private void button2_Click(object sender, EventArgs e) {
    Application.Exit();
}
```

실행 방법 : 각 메뉴 항목을 클릭하여 Click 이벤트를 발생시킨다.

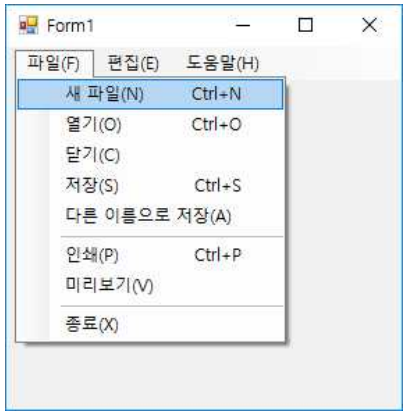
실행 결과 :

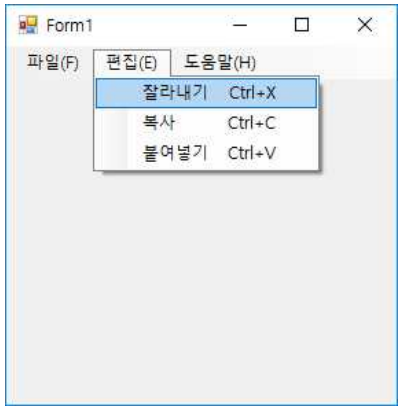




메뉴 - 단축 문자와 단축키 [1/2]

- 단축문자
  - 메뉴항목의 이름에 &를 붙인 형태
    - 파일(&F), 복사(&C)
  - 메뉴 표시줄에 나타나는 메인 메뉴 사이에서는 반드시 유일해야 함
  - 메인 메뉴의 서로 다른 메뉴 항목에 대해서는 중복 사용 가능





메뉴 - 단축 문자와 단축키 [2/2]

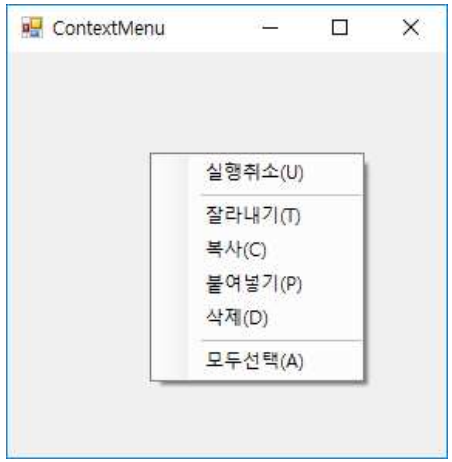
- 단축키
  - 메뉴항목의 Shortcut 프로퍼티를 통해 설정
  - 단축키는 하나의 애플리케이션에 포함된 모든 메뉴 항목에 대하여 유일하도록 설정
  - 중복하여 설정할 경우, 두 번째 이후로 설정된 메뉴 항목의 단축키는 반영되지 않음

상황 메뉴

- 컨트롤 위에서 마우스의 오른쪽 버튼을 클릭하였을 때 표시되는 팝업 메뉴
  - 현재 애플리케이션의 상태가 반영
  - 상황에 따라 독자적인 메뉴 항목을 가짐

상황 메뉴의 작성

- ContextMenuStrip 컴포넌트의 추가
- 상황 메뉴는 메인 메뉴와 동일한 프로퍼티와 이벤트를 가짐
- 완성된 상황 메뉴를 해당 폼 또는 컨트롤의 ContextMenu 프로퍼티에 설정
  - 컨트롤마다 상황 메뉴를 가질 수 있기 때문에 적용하고자 하는 컨트롤의 ContextMenu 프로퍼티에 설정
  - 폼의 ContextMenu 프로퍼티에 작성된 contextMenu1 컴포넌트를 지정한 예



C# 프로그래밍

제 4 장 클래스
델리게이트와 이벤트

델리게이트

- 델리게이트
  - 메서드를 참조하기 위한 기법
  - 하나의 클래스에서 다른 것은 필요 없고, 단지 특정 객체의 메서드 만 사용 하고자 할 때 사용하는 기법
  - 이벤트와 스레드를 처리하는데 주로 사용
  - C/C++ 언어에서 함수 포인터와 유사한 기능을 갖고 있지만 포인터보다는 객체 지향적이며 타입 안정적임
  - 묵시적으로 System.Delegate 클래스를 상속한 클래스형으로 간주

※ 스레드(thread)?
-컴퓨터 프로그램 작성에 있어서 하나의 프로세스
-프로그램의 일부가 되는 일련의 프로세스

[클래스 멤버의 종류]
-자료 멤버
필드, 상수, 이벤트

-함수멤버
메서드, 생성자, 소멸자, 프러퍼티 (속성), 인덱서(색인자), 연산자 중복

-내포형 멤버
클래스형, 구조형, 열거형, 인터페이스형, 대리(델리게이트)형

[객체 지향 프로그램 구성]
-클래스 정의
-객체 선언
-객체 생성
-객체의 멤버 접근 및 활용

델리게이트를 만드는 기본적 방법

- 1단계 : delegate 할 메서드를 정함
  - 어떠한 메서드가 대리자를 통해서 호출될 것인가를 정하는 것

```
using System;
class Example
{
    public void Fuc1()              //Delegate할  메서드1
    {
        System.Console.WriteLine("Example.Fuc1");
    }
    public void Fuc2(int x)         //Delegate할메서드2
    {
        System.Console.WriteLine("Example.Fuc2 x=" + x);
    }
}
```

[델리게이트 구성]
-델리게이트 정의
-델리게이트 객체 선언
-델리게이트 객체 생성
-델리게이트 객체 호출

델리게이트를 만드는 기본적 방법

- 2단계 : 메서드에 맞는 델리게이트 선언하기
  - 메서드의 시그너처(signature)를 정확하게 일치
    - delegate와 메서드의 리턴타입과 매개변수를 정확하게 일치시켜야 함

```
using System;
delegate void SimpleDelegate1();              //Delegate 선언1  void F1()
delegate void SimpleDelegate2(int i);         //Delegate 선언2 void F2()
class Example
{
    public void Fuc1()              //Delegate할메서드1
    {
        System.Console.WriteLine("Example.Fuc1");
    }
    public void Fuc2(int x)         //Delegate할메서드2
    {
        System.Console.WriteLine("Example.Fuc2 x=" + x);
    }
}
```

[델리게이트 구성]
-델리게이트 정의
-델리게이트 객체 선언
-델리게이트 객체 생성
-델리게이트 객체 호출

델리게이트를 만드는 기본적 방법

- 3단계 : 임의의 객체 만들기
  - 메서드가 포함된 클래스의 객체를 만들어야 함
    - 특정 객체의 메서드가 하나의 독립적인 delegate로서 활동할 수 있음

```
using System;
delegate void SimpleDelegate1();              //Delegate 선언1  void F1()
delegate void SimpleDelegate2(int i);         //Delegate 선언2 void F2()
class Example
{     public void Fuc1()              //Delegate할메서드1
      {       System.Console.WriteLine("Example.Fuc1");    }
      public void Fuc2(int x)         //Delegate할메서드2
      {       System.Console.WriteLine("Example.Fuc2 x=" + x);       }
}
class DeleTest                                                  //class
{
      public static void Main()
      {
              Example exam = new Example();  //객체생성
             //  exam.Fuc1( );  //메서드 호출

              }
}
```

[델리게이트 구성]
-델리게이트 정의
-델리게이트 객체 선언
-델리게이트 객체 생성
-델리게이트 객체 호출

델리게이트를 만드는 기본적 방법

- 4단계 : delegate 생성과 호출
  - delegate를 생성하여 해당 delegate를 호출
    - 일반 메서드를 호출하듯이 간단하게 delegate를 이용하여 메서드 호출

```
using System;
delegate void SimpleDelegate1();                //Delegate 선언1  void F1()
delegate void SimpleDelegate2(int i);           //Delegate 선언2 void F2()
class Example
{    public void Fuc1()                 //Delegate할메서드1
     {        System.Console.WriteLine("Example.Fuc1");    }
     public void Fuc2(int x)            //Delegate할메서드2
     {        System.Console.WriteLine("Example.Fuc2 x=" + x);        }
}
class DeleTest                                                  //class
{    public static void Main()
     {  Example exam = new Example();        //객체생성
        SimpleDelegate1 s1 = new SimpleDelegate1(exam.Fuc1); //Delegate 생성
        SimpleDelegate2 s2 = new SimpleDelegate2(exam.Fuc2); //Delegate 생성
        s1();                                   //Delegate를이용한호출1
        s2(1000);                               //Delegate를이용한호출2
     }
 }
```

[델리게이트 구성]
-델리게이트 정의
-델리게이트 객체 선언
-델리게이트 객체 생성
-델리게이트 객체 호출

델리게이트 예제

예제 4.20 [ex4_20_xxxx] : 델리게이트 => 콘솔로

```
using System;
  delegate void DelegateOne();      // delegate with no params
  delegate void DelegateTwo(int i); // delegate with 1 param

  class DelegateClass {
     public void MethodA() {
        Console.WriteLine("In the DelegateClass.MethodA ...");
     }
     public void MethodB(int i){
        Console.WriteLine("DelegateClass.MethodB, i = " + i);
     }
  }

  class DelegateCallApp {
     public static void Main() {
        DelegateClass obj = new DelegateClass();
        DelegateOne d1 = new DelegateOne(obj.MethodA);
        DelegateTwo d2 = new DelegateTwo(obj.MethodB);
        //obj.MethodA(); obj.MethodB();
        d1();          // invoke MethodA() in DelegateClass
        d2(10);         // invoke MethodB(10) in DelegateClass
     }
  }
```

실행 결과 :

```
In the DelegateClass.MethodA ...
DelegateClass.MethodB, i = 10
```

[델리게이트 구성]
-델리게이트 정의
-델리게이트 객체 선언
-델리게이트 객체 생성
-델리게이트 객체 호출

이벤트

- 이벤트
  - 사용자 행동에 의해 발생하는 사건을 의미
  - 어떤 사건이 발생한 것을 알리기 위해 보내는 메시지

- **이벤트 처리기(event handler)**
  - 특정 이벤트가 발생하면 그에 등록된 메서드를 통해 처리

- 이벤트-주도 프로그래밍(event-driven programming)
  - 이벤트와 이벤트 처리기를 통하여 객체에 발생한 사건을 다른 객체에 통지하고 그에 대한 행위를 처리하도록 시키는 구조
  - 프로그램의 복잡도를 줄임
  - 각 이벤트에 따른 작업을 독립적으로 기술
  - 프로그램의 구조가 체계적/구조적이며 복잡도를 줄일 수 있음

[클래스 멤버의 종류]
-자료 멤버
필드, 상수, 이벤트

-함수멤버
메서드, 생성자, 소멸자, 프러퍼티 (속성), 인덱서(색인자), 연산자 중복

-내포형 멤버
클래스형, 구조형, 열거형, 인터페이스형, 대리(델리게이트)형

[객체 지향 프로그램 구성]
-클래스 정의
-객체 선언
-객체 생성
-객체의 멤버 접근 및 활용

이벤트 정의

- 정의 형태

```
[event-modifier] event DelegateType EventName;
```

- 수정자
  - 접근 수정자
  - new, static, virtual, sealed, override, abstract, extern
  - 이벤트 처리기는 메소드로 배정되기 때문에 메소드 수정자와 종류/의미가 같음

이벤트

이벤트 정의 순서

① 이벤트 처리기를 작성

② 이벤트 처리기의 형태와 일치하는 델리게이트를 정의
(또는 System.EventHandler 델리게이트를 사용)

③ 델리게이트를 이용하여 이벤트를 선언
(미리 정의된 이벤트인 경우에는 생략)

④ 이벤트에 이벤트 처리기를 등록

⑤ 이벤트를 발생
(미리 정의된 이벤트는 사용자 행동에 의해 이벤트가 발생)

[이벤트 정의 순서]
① 이벤트 처리기 작성
② 이벤트를 위한 델리게이트 정의
③ 이벤트 선언
④ 이벤트에 이벤트 처리기를 등록
⑤ 이벤트를 발생

- 이벤트가 발생되면 등록된 메소드가 호출되어 이벤트를 처리
  - 미리 정의된 이벤트 발생은 사용자의 행동에 의해서 발생
  - 사용자 정의 이벤트인 경우에는 명시적으로 델리게이트 객체를 호출함으로써 이벤트 처리기를 작동

이벤트 예제

예제 4.23 [ex4_23_xxxx] : 이벤트 => 콘솔로

```
using System;
public delegate void MyEventHandler()                    // ② 이벤트를 위한 델리게이트 정의
class Button {
    public event MyEventHandler Push;                    // ③이벤트 선언
    public void OnPush() {
        if (Push != null)
            Push();                                      // ⑤ 이벤트 발생
              //=> 이벤트 호출 시 매개변수는 델리게이트와 같은 조건의 실 매개변수를 넣어줄 것!
              // Push(3)이면 델리게이트 매개변수 타입과 달라서 오류 발생!!!
    }
}
class EventHandlerClass {
    public void MyMethod() {                             // ① 이벤트 처리기 작성
        Console.WriteLine("In the EventHandlerClass.MyMethod ...");
    }
}
class EventHandlingApp {
    public static void Main() {
        Button button = new Button();
        EventHandlerClass obj = new EventHandlerClass();
        button.Push += new MyEventHandler(obj.MyMethod);  // ④ 이벤트 등록
         //obj.MyMethod();
        button.OnPush();
    }
}
```

실행 결과 :

```
In the EventHandlerClass.MyMethod ...
```

[이벤트 정의 순서]
① 이벤트 처리기 작성
② 이벤트를 위한 델리게이트 정의
③ 이벤트 선언
④ 이벤트에 이벤트 처리기를 등록
⑤ 이벤트를 발생

이벤트의 활용

- C# 언어에서의 이벤트 사용
  - 프로그래머가 임의의 형식으로 델리게이트를 정의하고 이벤트를 선언할 수 있도록 허용
  - .NET 프레임워크는 이미 정의된 System.EventHandler 델리게이트를 이벤트에 사용하는 것을 권고
  - System.EventHandler

```
delegate void EventHandler(object sender, EventArgs e);
```

- 이벤트와 윈도우 환경
  - 이벤트는 사용자와 상호작용을 위해 주로 사용
  - 윈도우 프로그래밍 환경에서 사용하는 폼과 수많은 컴포넌트와 컨트롤에는 다양한 종류의 **이벤트**가 존재

- 이벤트의 활용
  - 이벤트 : Click
  - 이벤트 처리기 : button1_Click()
  - 이벤트 발생 주체 : 프로그래머가 아닌 **User**

이벤트

예제 4.23 [ex4_23_xxxx] : 이벤트 => 콘솔로

```
using System;
   public delegate void MyEventHandler() ;              // ② 이벤트를 위한 델리게이트 정의 => 상위 클래스 정의
   class Button {
      public event MyEventHandler Push;                 // ③ 이벤트 선언  => 상위 클래스 정의
      public void OnPush() {
         if (Push != null)
            Push();                                     // ⑤ 이벤트 발생 => 자동 발생
      }
   }
   class EventHandlerClass {
      public void MyMethod() {                                  // ① 이벤트 처리기 작성 => 프로그래머
         Console.WriteLine("In the EventHandlerClass.MyMethod ...");
      }
   }
   class EventHandlingApp {
      public static void Main() {
         Button button = new Button();
         EventHandlerClass obj = new EventHandlerClass();
         button.Push += new MyEventHandler(obj.MyMethod);//④ 이벤트 등록 => initializeComponent()에
         //obj.MyMethod();
         button.OnPush();
      }
   }
```

실행 결과 :

```
In the EventHandlerClass.MyMethod ...
```

이벤트의 활용 : 퀴즈10_1

quiz 10.1 [quiz10_1_xxxx] : 이벤트 정의 순서를 나타내시오. => 콘솔로

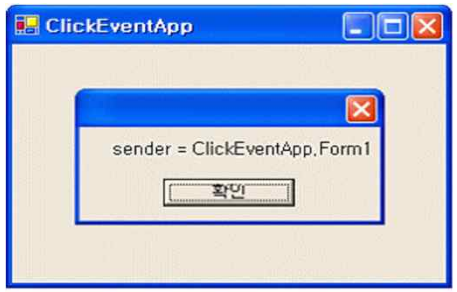
```
using System;
using System.Windows.Forms;      // 참조 추가

class ClickEventApp : Form {
    public ClickEventApp() {
        this.Text = "ClickEventApp";
        this.Click += new EventHandler(ClickEvent);
    }

    private void ClickEvent(object sender, EventArgs e) {
        MessageBox.Show(" sender = " + sender.GetType());
    }

    public static void Main() {
        Application.Run(new ClickEventApp());
    }
}
```

실행 결과 :



[이벤트 정의 순서]
① 이벤트 처리기 작성
② 이벤트를 위한 델리게이트 정의
③ 이벤트 선언
④ 이벤트에 이벤트 처리기를 등록
⑤ 이벤트를 발생

9장 다중 폼 : 정보 교환 3 : 모덜리스 방식

- Form2 대화상자를 모덜리스 대화 상자로 사용 시 항상 열어둠

다중폼3 [multiform3_xxxx] : 다중 폼 모덜리스 방식 3

★ Form1

1) 실행

2) Form1 소스

```
public partial class Form1 : Form
{   private Form2 dlg;
    private void OnApply(object sender, EventArgs e)
    {   label1.Left = dlg.LabelX;
        label1.Top = dlg.LabelY;
        label1.Text = dlg.LabelText;
    }
    private void button1_Click(object sender, EventArgs e)
    {   if (dlg == null || dlg.IsDisposed)
        {   dlg = new Form2();
            dlg.Owner = this;
            dlg.Apply += new EventHandler(OnApply);
            dlg.LabelX = label1.Left;
            dlg.LabelY = label1.Top;
            dlg.LabelText = label1.Text;
            dlg.Show();
        }
    }   }
```

★ Form2

9장 다중 폼 : 정보 교환 3 : 모덜리스 방식

- Form2 대화상자를 모덜리스 대화 상자로 사용시 항상 열어둠

다중폼3 [multiform3_xxxx] : 다중 폼 모덜리스 방식 3

★ Form2의 button1, button2 속성 설정 : DialogResult :None

3) Form2 소스

```
public partial class Form2 : Form
    {    public int LabelX
         {   get { return Convert.ToInt32(textBox1.Text); }
             set { textBox1.Text = value.ToString(); }        }
         public int LabelY
         {   get { return Convert.ToInt32(textBox2.Text); }
             set { textBox2.Text = value.ToString(); }        }
         public string LabelText
         {  get { return textBox3.Text; }
             set { textBox3.Text = value; }        }
      public event EventHandler Apply;
         private void button1_Click(object sender, EventArgs e)
         {   if (Apply != null)
             {
                 Apply(this, new EventArgs()); //매개변수 타입이 같아야 함!
             }
         }
         private void button2_Click(object sender, EventArgs e)
         {
             Dispose();
         }
}
```