

제6장

델리게이트(Delegate)

변영철

1. 델리게이트 의미

- 델리게이트(delegate)
- 대리인
- 대신 사용하는 것
- 무엇 대신? 함수 대신

delegate

- ▶ C# 응용에서 이벤트를 처리하는 기법으로 Delegate가 있음.
- ▶ 많은 코드가 미리 준비되어 있고(응용 프레임워크) 우리가 원하는 부분만 고쳐서 사용할 수 있음.

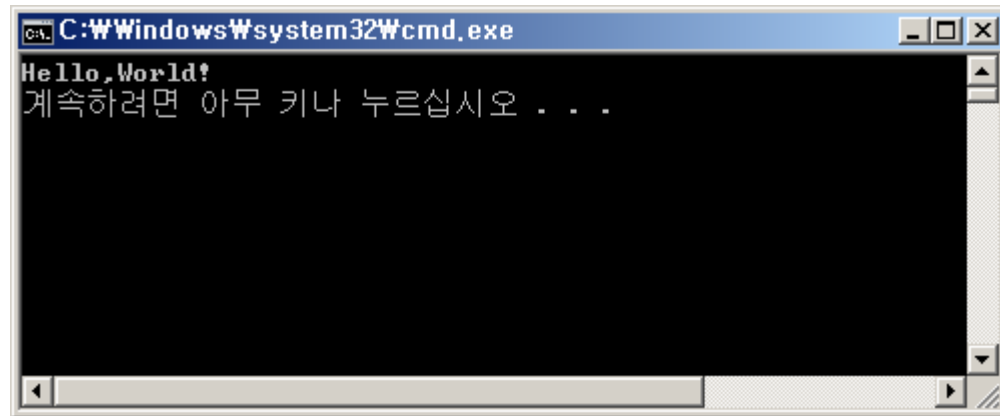
2. 프로젝트 생성

- 새로운 빈 프로젝트 Delegate 생성
- 프로젝트 | 새 항목 추가
 - Delegate.cs 파일

3. 코드 편집 및 작성

- 기본 코드를 “Hello,World!” 표시하도록 변경

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



3. 코드 편집 및 작성

- “클릭!” 표시하도록 변경

```
public class Delegate
{
    public static void Main() {
        System.Console.WriteLine("클릭!");
    }
}
```



4. 코드 추상화와 함수

- xxx 함수로 코드 추상화

```
using System;
```

```
public class Delegate  
{
```

```
    public void xxx() {  
        Console.WriteLine("클릭!");  
    }
```

```
    public static void Main() {  
        xxx(); //실행될까??  
    }
```

```
}
```

4. 코드 추상화와 함수

- 클래스는 객체(gildong) 만들라고 있는 것
- gildong이에게 xxx 해달라고 하자.

```
using System;
```

```
public class Delegate
{
    public void xxx() {
        Console.WriteLine("클릭!");
    }

    public static void Main() {
        Delegate gildong = new Delegate();
        gildong.xxx();
    }
}
```

5. 데이터 추상화 클래스

- Base 클래스 작성

```
public class Base  
{  
  
}
```


5. 데이터 추상화 클래스

- xxx 함수를 Base로 옮기기

```
public class Base
{
    public void xxx() {
        Console.WriteLine("클릭!");
    }
}
```

5. 데이터 추상화 클래스

- 기존 코드 수정

```
public class Delegate
{
    public static void Main() {
        Base gildong = new Base();
        gildong.xxx();
    }
}
```

6. 델리게이트로 호출하기

- 델리게이트 이용하기(xxx 대신 Click 사용)

```
public class Delegate
{
    public static void Main() {
        Base gildong = new Base();
        gildong.Click(); //Click은 xxx 함수의 델리게이트
    }
}
```

- ▶ xxx 멤버 함수를 호출했었는데 이제는 xxx 대신 Click 사용(호출)
- ▶ 만일 이렇게 하더라도 오류가 없다면
- ▶ Click은 xxx 멤버 함수의 델리게이트(대신 사용하는 것)

6. 델리게이트로 호출하기

- C언어 함수 포인터 변수 : 델리게이트와 유사

```
void xxx() {  
    printf("Hello,World!");  
}
```

```
void main()  
{  
    void xxx()* Click = null; //void (*click) (); 이 맞는 문법  
  
    Click= xxx;  
    Click();  
}
```

6. 델리게이트로 호출하기

- 함수 포인터 변수 Click, 객체
- Click은 변수 혹은 객체
- 객체를 만들려면 클래스가 있어야
- 클래스 이름이 DelegateClass일 경우

```
using System;
```

```
public class Base
{
    public DelegateClass Click = null;
    public void xxx() {
        Console.WriteLine("클릭!");
    }
}
```

6. 델리게이트로 호출하기

- 자료형 선언: delegate 키워드를 사용

```
using System;
```

```
public class Base  
{
```

```
    public delegate void DelegateClass();  
    public DelegateClass Click = null;
```

```
    public void xxx() {  
        Console.WriteLine("클릭!");  
    }
```

```
}
```

리턴값이 없고(void)
파라미터가 없는() 함
수를 대신 실행할 수
있음을 의미.

6. 델리게이트로 호출하기

- 델리게이트 객체생성 및 xxx 함수 설정

```
public class Base
{
    public delegate void DelegateClass();
    public DelegateClass Click = null;

    public Base() {
        Click = new DelegateClass(xxx);
    }

    public void xxx() {
        Console.WriteLine("클릭!");
    }
}
```

6. 델리게이트로 호출하기

- 컴파일 및 실행



6. 델리게이트로 호출하기

- 생성자 함수에서
델리게이트 Click을 xxx와
연결하지 않을 경우에는
어떤 일이?



```
public class Base
{
    public delegate void DelegateClass();
    public DelegateClass Click = null;

    public void xxx() {
        System.Console.WriteLine("Hello,World!");
    }
}

public class Delegate
{
    public static void Main() {
        Base gildong = new Base();
        gildong.Click();
    }
}
```

6. 델리게이트로 호출하기

- 이런 경우를 핸들링 하려면?

```
public class Base
{
    public delegate void DelegateClass();
    public DelegateClass Click = null;

    public void xxx() {
        System.Console.WriteLine("Hello,World!");
    }
}
```

```
public void OnClick() {
    if (Click != null)
        Click();
}
```

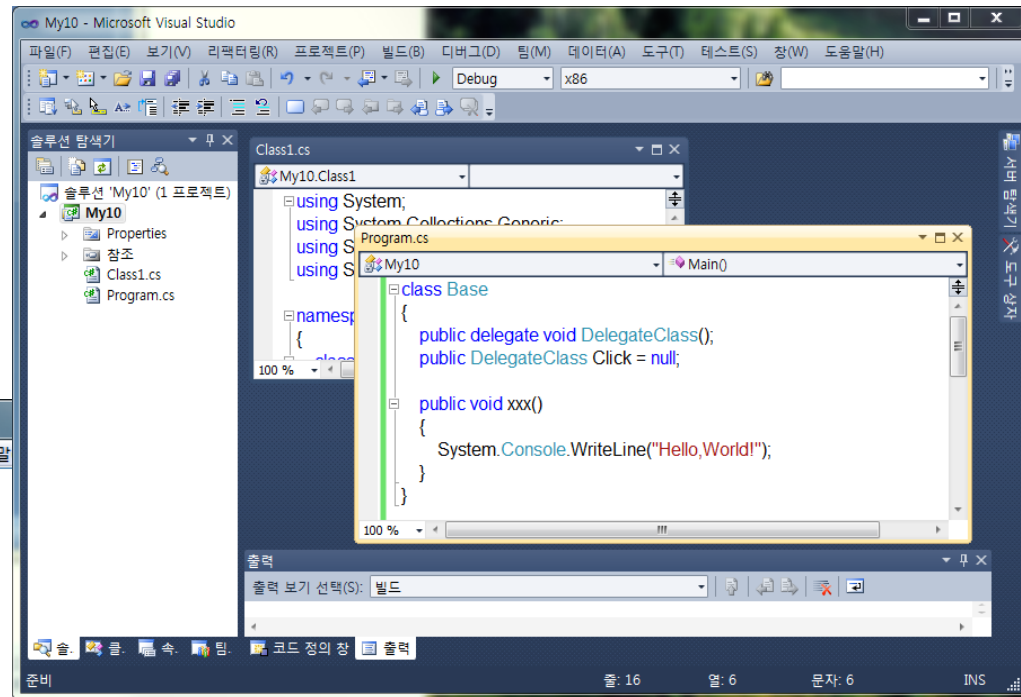
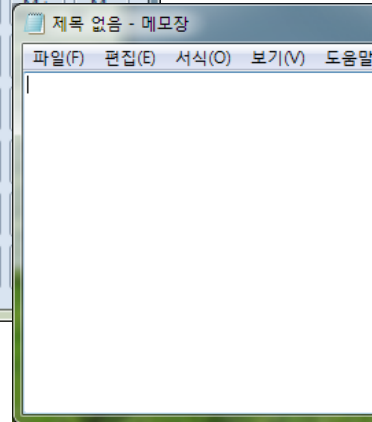
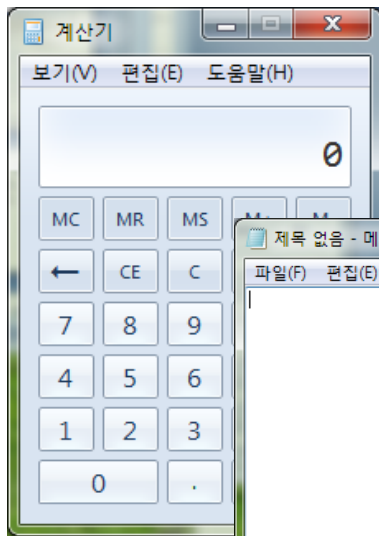
```
public class Delegate
{
    public static void Main() {
        Base gildong = new Base();
        gildong.OnClick();
    }
}
```

7. 델리게이트와 이벤트

Click	델리게이트, 델리게이트 객체, 델리게이트 인스턴스, 이벤트
DelegateClass	델리게이트 형(type)
xxx	핸들러 함수
OnClick	이벤트 Click을 fire 하는 함수

8. 응용 프레임워크와 델리게이트

- 윈도우 운영체제 특징
 - GUI: 응용 프로그램이 서로 비슷
 - 멀티태스킹 기능
 - 이벤트 처리 기능



따라서 수많은 중복되는 코드들이 존재한다.

→ 라이브러리로 만들어 놓고 필요할 때 사용하면 좋겠다.

8. 응용 프레임워크와 델리게이트

- 이전 코드 다시 보기

```
public class Base
{
    public delegate void DelegateClass();
    public DelegateClass Click = null;

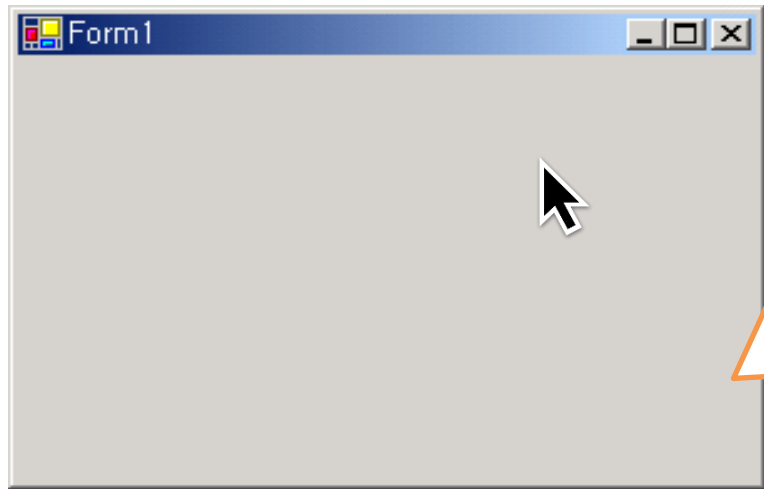
    public void xxx() {
        System.Console.WriteLine("Hello,World!");
    }
}
```

```
public void OnClick() {
    if (Click != null)
        Click();
}
```

```
public class Delegate
{
    public static void Main() {
        Base gildong = new Base();
        gildong.OnClick();
    }
}
```

8. 응용 프레임워크와 델리게이트

- 폼 윈도우와 이벤트



```
public class Delegate {  
    public static void Main() {  
        Base gildong = new Base();  
        gildong.얼굴보여줘();  
        이벤트 루프();  
    }  
}
```

1. 마우스로 폼 윈도우(길동이 얼굴) 클릭
2. 닷넷 런타임이 gildong 객체의 OnClick이라는 멤버 함수 호출
3. OnClick 메소드에서 Click이 널이 아니면 Click() → 이벤트 발생(fire)

8. 응용 프레임워크와 델리게이트

- OnClick 함수 → 구글 검색

8. 응용 프레임워크와 델리게이트

- 어떤 프로그램을 만들고 싶니?
- 어떤 프로그램을 만들지에 따라서 핸들러 함수(xxx)를 붙일 수도 있고, 안 붙일 수도 있고...
- 필요할 경우 개발자는 핸들러 함수(xxx)를 만들어 델리게이트(Click)에 연결만 하면 끝
- 닷넷 응용 프레임워크에서는 이미 델리게이트(Click)로 우리가 핸들러 함수를 실행하는 코드를 이미 만들어 놓았음.
- 즉, **델리게이트가 있는 이유**는 닷넷 응용 프레임워크에서 아직 우리가 만들지 않는 핸들러 함수 xxx를 호출할 수 있도록 하기 위하여

9. Base 클래스 라이브러리화

- (가정) 앞으로 여러분이 어떤 프로그램을 작성하더라도 항상 Base라는 클래스를 작성해야 한다면,

```
public class Base
{
    public delegate void DelegateClass();
    public DelegateClass Click = null;

    public void xxx() {
        System.Console.WriteLine("Hello,World!");
    }

    public void OnClick() {
        if (Click != null)
            Click();
    }
}
```

```
public class Delegate
{
    public static void Main() {
        Base gildong = new Base();
        gildong.OnClick();
    }
}
```

9. Base 클래스 라이브러리화

- 새로운 프로그램 작성할 때마다 매번 새로 작성하는 것 보다는 **갖고 있다가** 나중에 다시 사용하자.
(라이브러리).

9. Base 클래스 라이브러리화

- 문제점
 - xxx라는 핸들러 함수 이름은 프로그래머가 결정하는 것이므로 Base 클래스 안에 기술되어 있으면 Base는 라이브러리가 될 수 없음.
- Base를 상속받는 파생(derived) 클래스 Derived

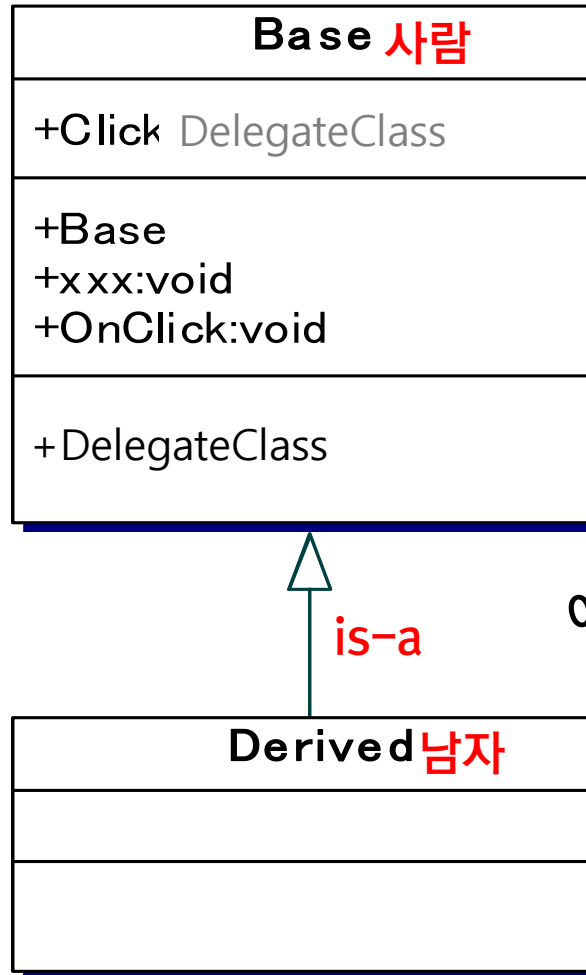
```
class Derived : Base
{

}
```

10. 비하인드 코드/클래스 Derived

- Derived 클래스 작성
 - Base 클래스 안에 작성된 코드가 고스란히 Derived 클래스 안으로 들어옴(재사용, 상속)
 - 따라서 Base 대신 Derived 클래스로 객체 만들어도 동일함
 - 개념적으로 볼 때 Derived 클래스 뒤쪽(behind)에 Base 클래스가 있는 모양
 - 따라서 Base를 **비하인드(behind) 클래스**, 혹은 비하인드 코드라고 부름. Derived는 프론트(front) 클래스

10. 비하인드 코드/클래스 Derived



아래 것(남자)은 위의 것(사람)이다.

11. Base 클래스 라이브러리화

```
class Derived : Base
```

```
{
```

```
    public Derived() {
```

```
        Click += new DelegateClass(xxx);
```

```
    }
```

```
    public void xxx() {
```

```
        Console.WriteLine("클릭!");
```

```
    }
```

```
}
```

라이브러리화 할 수 없는
코드들을 Derived로
밀어버림.

11. Base 클래스 라이브러리화

```
class Delegate
```

```
{
```

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

```
        Derived gildong = new Derived();
```

```
        gildong.OnClick();
```

```
    }
```

```
}
```

클래스(Derived)는 뭐하라고 있는 것?

12. Application 응용 클래스 작성

- Main 함수 내의 코드를 Run으로 추상화

```
public class Delegate
{
    public void Run()
    {
        Form gildong = new Derived();
        gildong.OnClick();
    }

    public static void Main()
    {
        Run();
    }
}
```

12. Application 응용 클래스 작성

- 객체 정의 및 호출

```
public class Delegate
{
    public void Run()
    {
        Base gildong = new Derived();
        gildong.OnClick();
    }

    public static void Main()
    {
        Delegate cheolsu = new Delegate();
        cheolsu.Run();
    }
}
```

12. Application 응용 클래스 작성

- 정적 멤버로 선언

```
public class Delegate
{
    public static void Run() {
        Base gildong = new Derived();
        gildong.OnClick();
    }

    public static void Main() {
        Run();
    }
}
```

12. Application 응용 클래스 작성

- 독립된 클래스로 작성 → 라이브러리

```
public class Application
{

}
```

```
public class Delegate
{
    public static void Run() {
        Base gildong = new Derived();
        gildong.OnClick();
    }

    public static void Main() {
        Run();
    }
}
```

12. Application 응용 클래스 작성

- 독립된 클래스로 작성 → 라이브러리

```
public class Application
{
    public static void Run() {
        Base gildong = new Derived();
        gildong.OnClick();
    }
}
```

```
public class Delegate
{
    public static void Main() {
        Application.Run();
    }
}
```

13. Application 클래스 라이브러리화

- 문제점

- 우선, Derived 클래스는 라이브러리화 할 수 있다, 없다?
- 그런 Derived를 사용하는 Application 클래스는?

```
public class Application
{
    public static void Run() {
        Base gildong = new Derived();
        gildong.OnClick();
    }
}
```

13. Application 클래스 라이브러리화

- 라이브러리가 될 수 있을까?

```
public class Application
{
    public static void Run(Base gildong) {
        gildong.OnClick();
    }
}
```

```
public class Delegate {
    public static void Main()
    {
        Application.Run(new Derived());
    }
}
```

14. 가상 함수를 이용한 이벤트 처리

- Base 클래스의 OnClick을 가상 함수로 만들기

```
public virtual void OnClick() {  
    if(Click != null)  
        Click();  
}
```

이 함수가 마음에 안들면
재정의해(overriding).
그러면 이 함수는
'가상(가짜)' 함수가 돼. 즉,
없는 것처럼 돼.

14. 가상 함수를 이용한 이벤트 처리

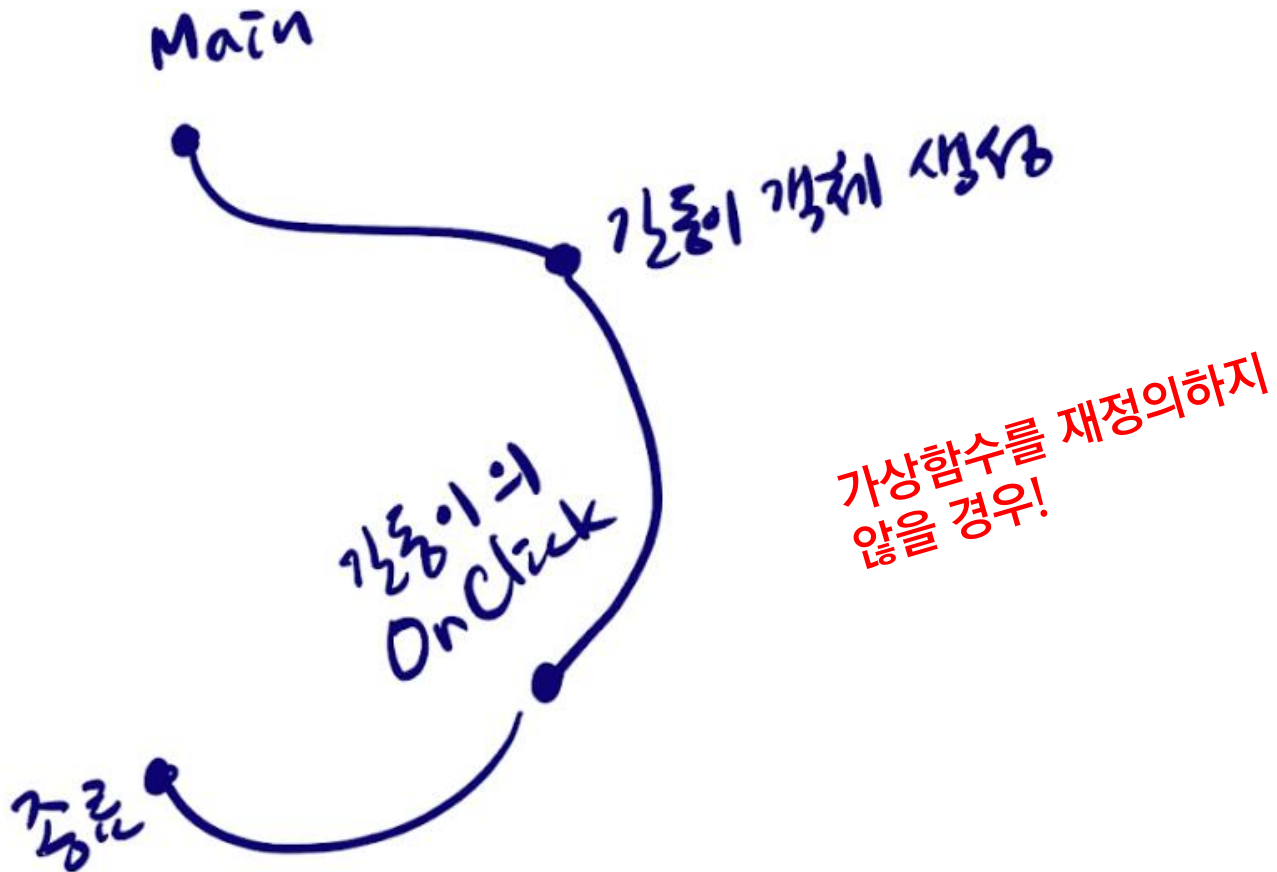
- Derived 클래스에서 가상함수 재정의(오버라이딩)

```
public class Derived : Base
{
    public Derived() {
        Click += new DelegateClass(xxx);
    }

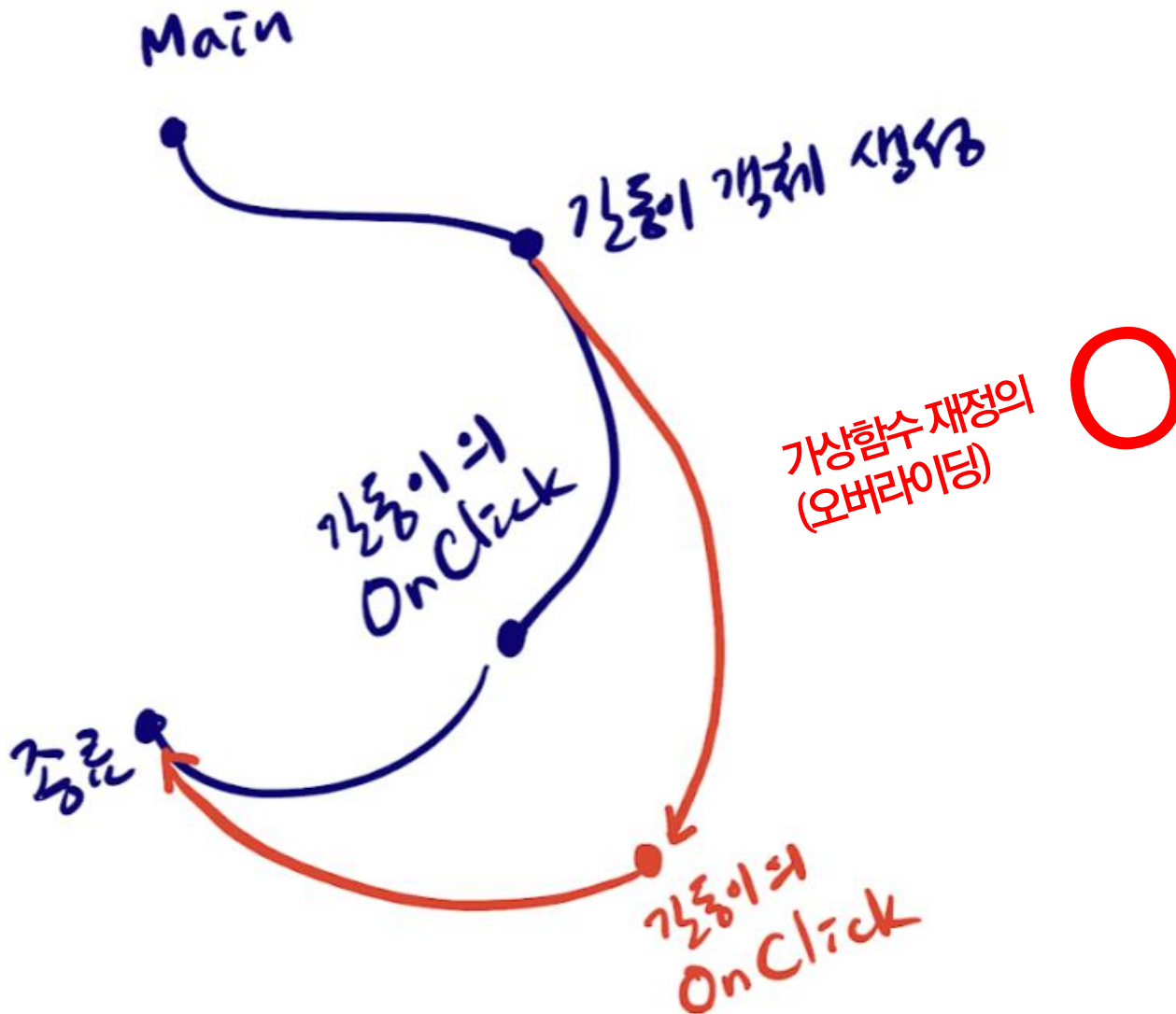
    public void xxx() {
        Console.WriteLine("클릭!");
    }

    public override void OnClick() {
    }
}
```

14. 가상 함수를 이용한 이벤트 처리

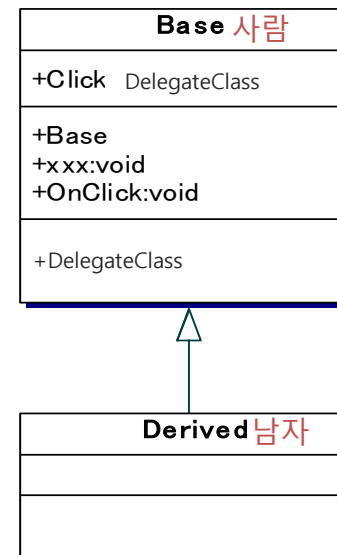


14. 가상 함수를 이용한 이벤트 처리



14. 가상 함수를 이용한 이벤트 처리

```
class Delegate
{
    static void Main(string[] args) {
        Base gildong = new Derived();
        gildong.OnClick();
    }
}
```



14. 가상 함수를 이용한 이벤트 처리

- 기존에 있는 가상함수 OnClick이 쓸모가 없거나 마음에 들지 않으면 오버라이딩 하자.

```
public override void OnClick() {  
    Console.WriteLine("-----");  
    Console.WriteLine("클릭!");  
    Console.WriteLine("-----");  
}
```

14. 가상 함수를 이용한 이벤트 처리

- 쓸모없는 것이 아니라 불충분하면 내용을 추가하여 보완하자.

```
public override void OnClick()
{
    Console.WriteLine("-----");
    base.OnClick();
    Console.WriteLine("-----");
}
```

14. 가상 함수를 이용한 이벤트 처리

- 따라서 이벤트가 발생할 경우 무엇인가를 하고 싶으면
- 다음과 같이 두 가지 방법 중 하나를 이용
 - 핸들러 함수(xxx)를 작성하여 연결
 - 또는 가상 함수(OnClick)를 오버라이딩

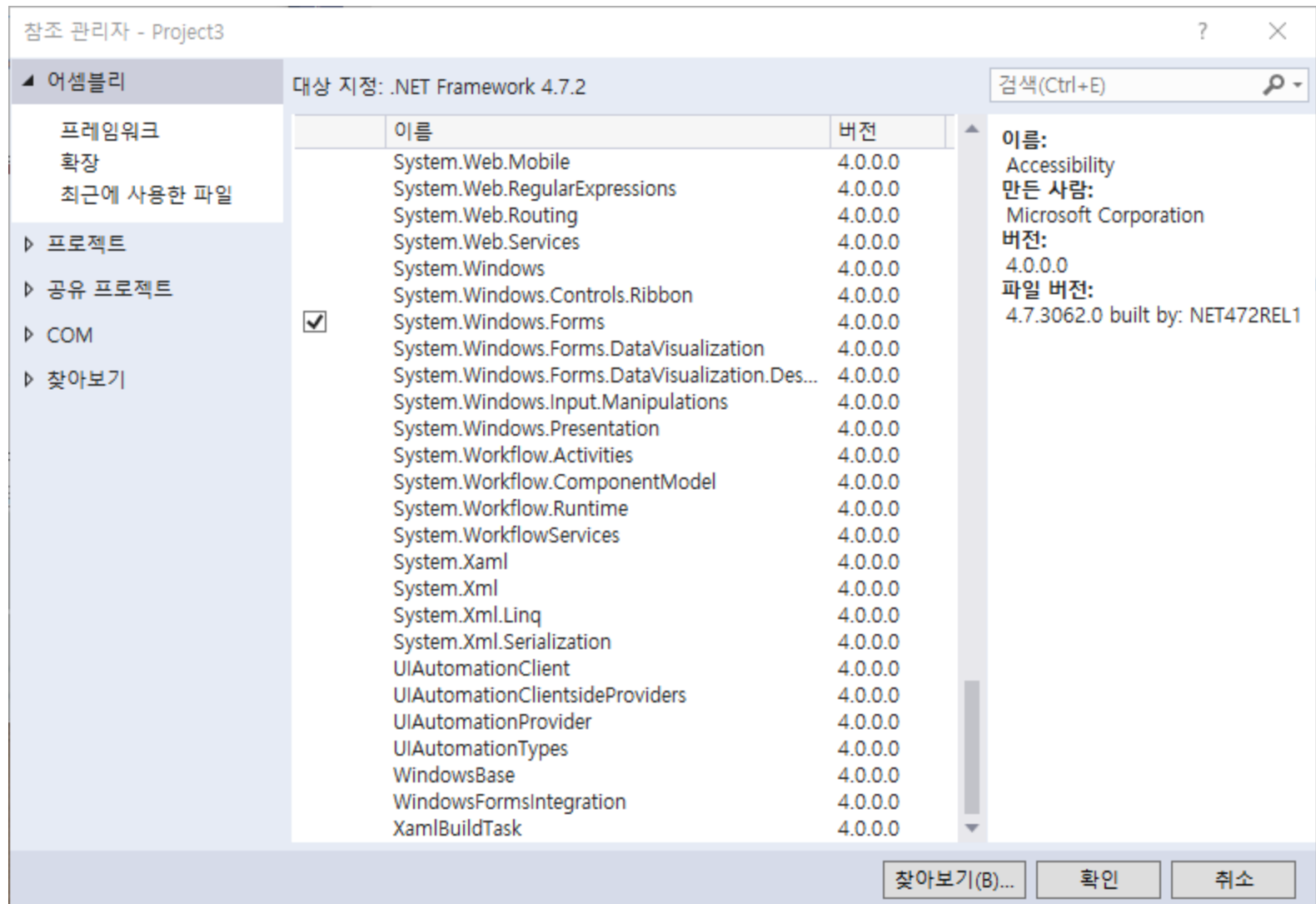
15. 코드 다듬기

- 클래스 이름 바꾸기
 - Base → Form
 - DelegateClass → EventHandler

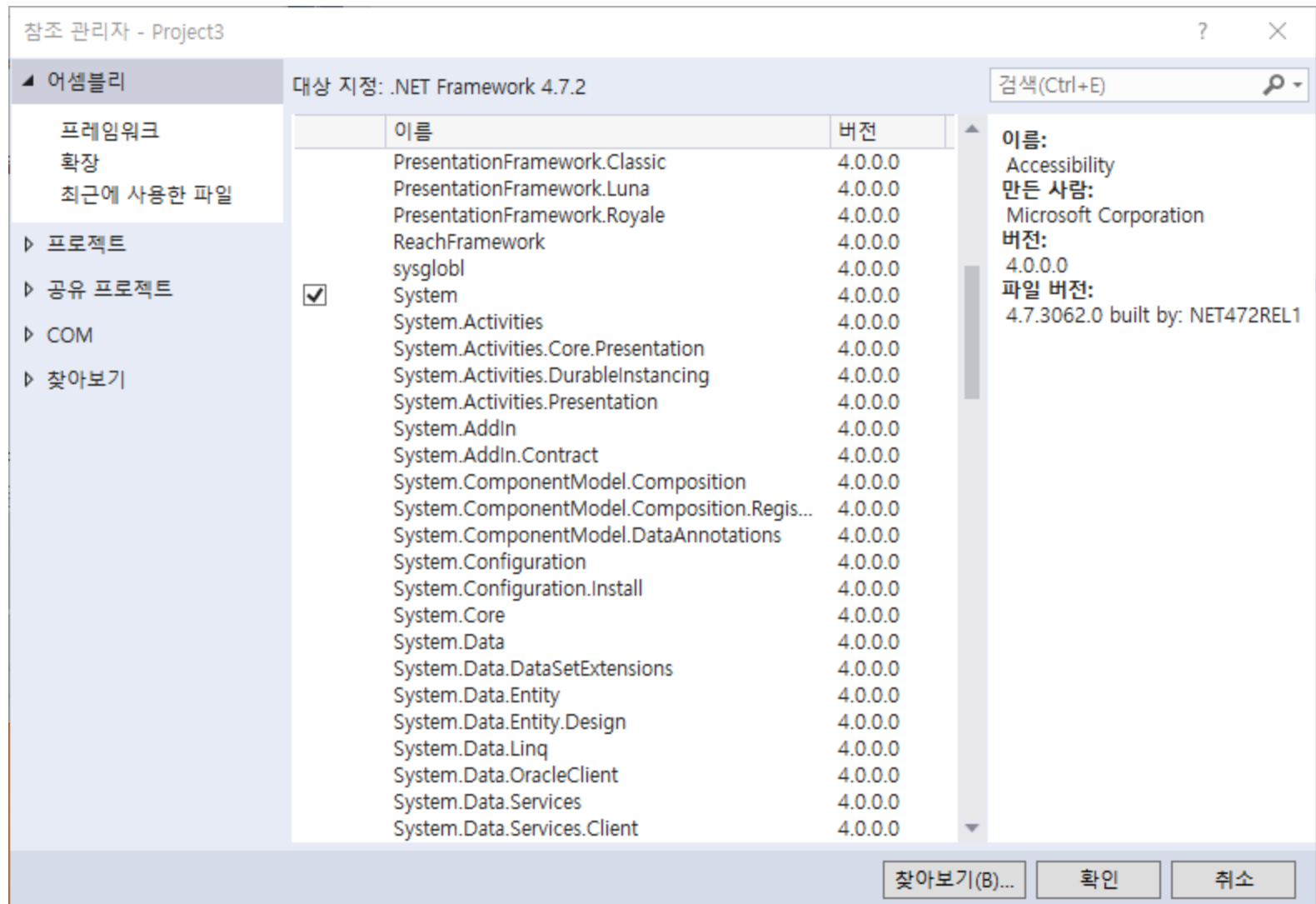
16. C# 클래스 라이브러리 이용하기

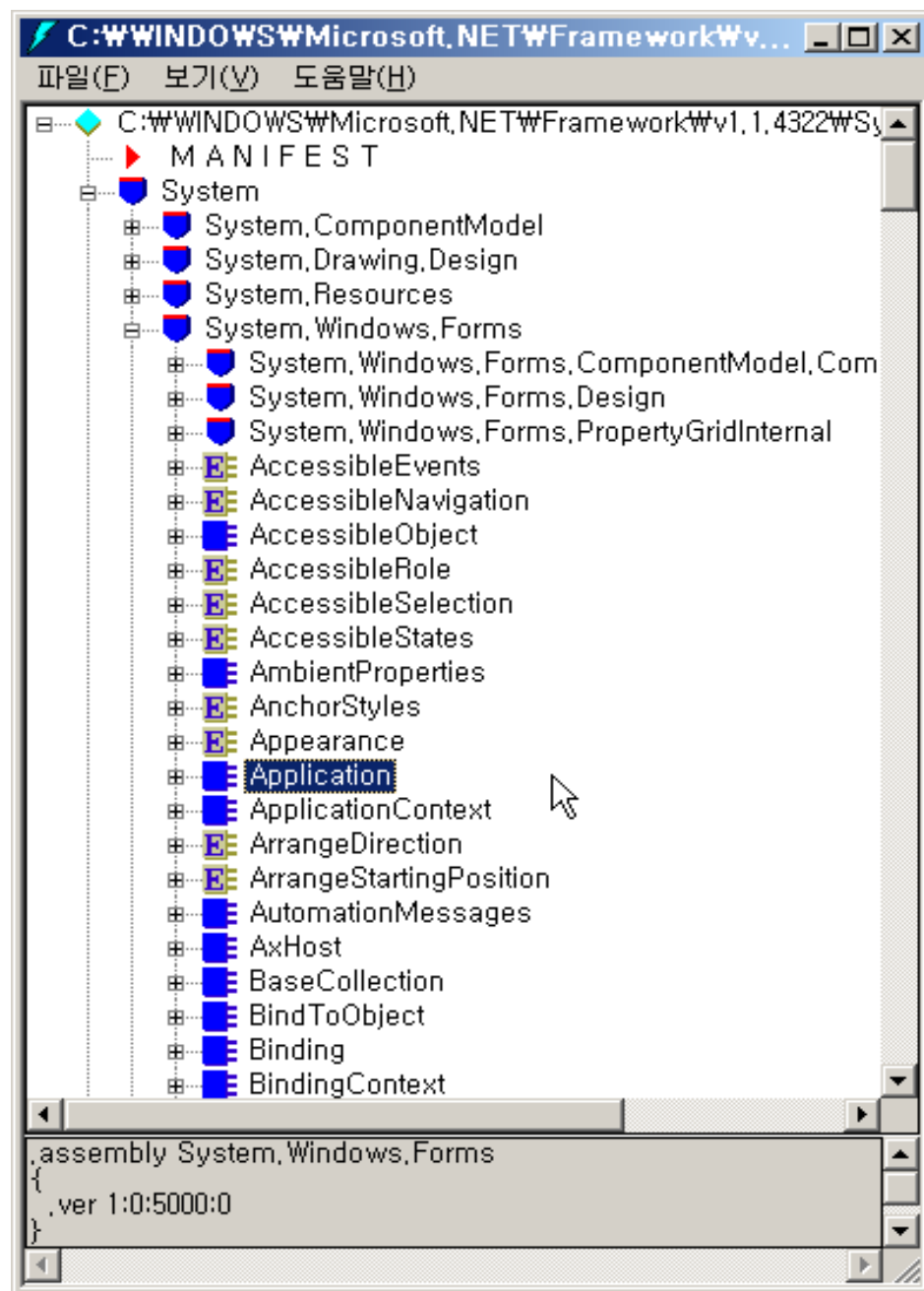
- Form, Application 클래스는 이미 닷넷 라이브러리에 존재함.
- 따라서 우리가 직접 힘들게 작성할 필요가 있다, 없다?
- 있는 것을 사용하자.
- 프로젝트 | 참조추가 메뉴 선택
 - Windows.System.Forms
 - System

16. C# 클래스 라이브러리 이용하기



16. C# 클래스 라이브러리 이용하기





16. C# 클래스 라이브러리 이용하기

- 이미 있는 것 이용하면 우리가 만든 클래스는 제거해도 됨.
- 아래 코드 추가하기

`using System.Windows.Forms;`

- EventHandler 선언 고치기

`public delegate void EventHandler(Object o,
EventArgs e);`

코드를 위와 같이 고쳤다. 그 의미는?

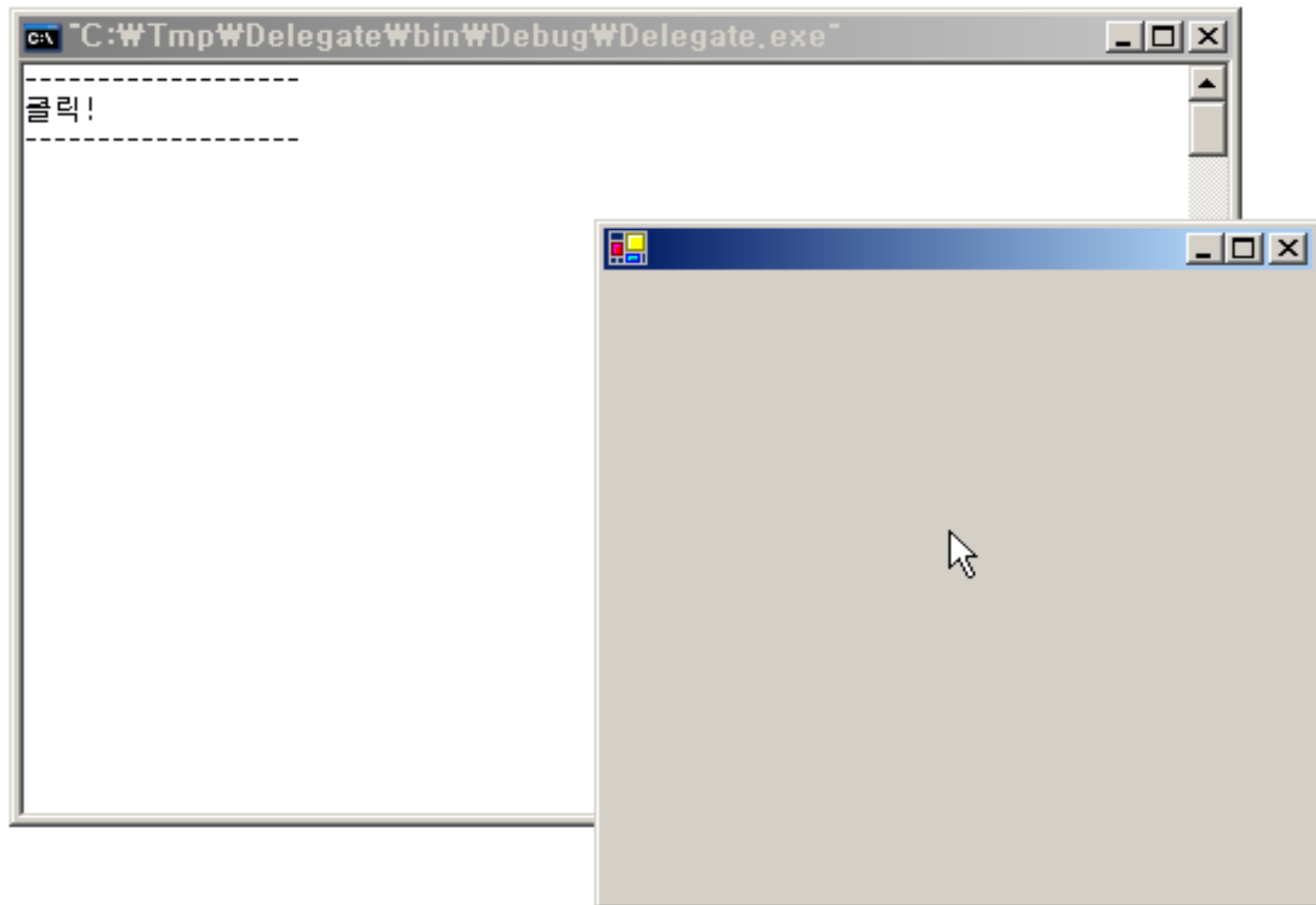
16. C# 클래스 라이브러리 이용하기

- 핸들러 함수 xxx 수정

```
public void xxx(Object o, EventArgs e)
{
    Console.WriteLine("클릭!");
}
```

- OnClick 가상함수 수정

```
protected override void OnClick(EventArgs e)
{
    Console.WriteLine("-----");
    base.OnClick(e);
    Console.WriteLine("-----");
}
```



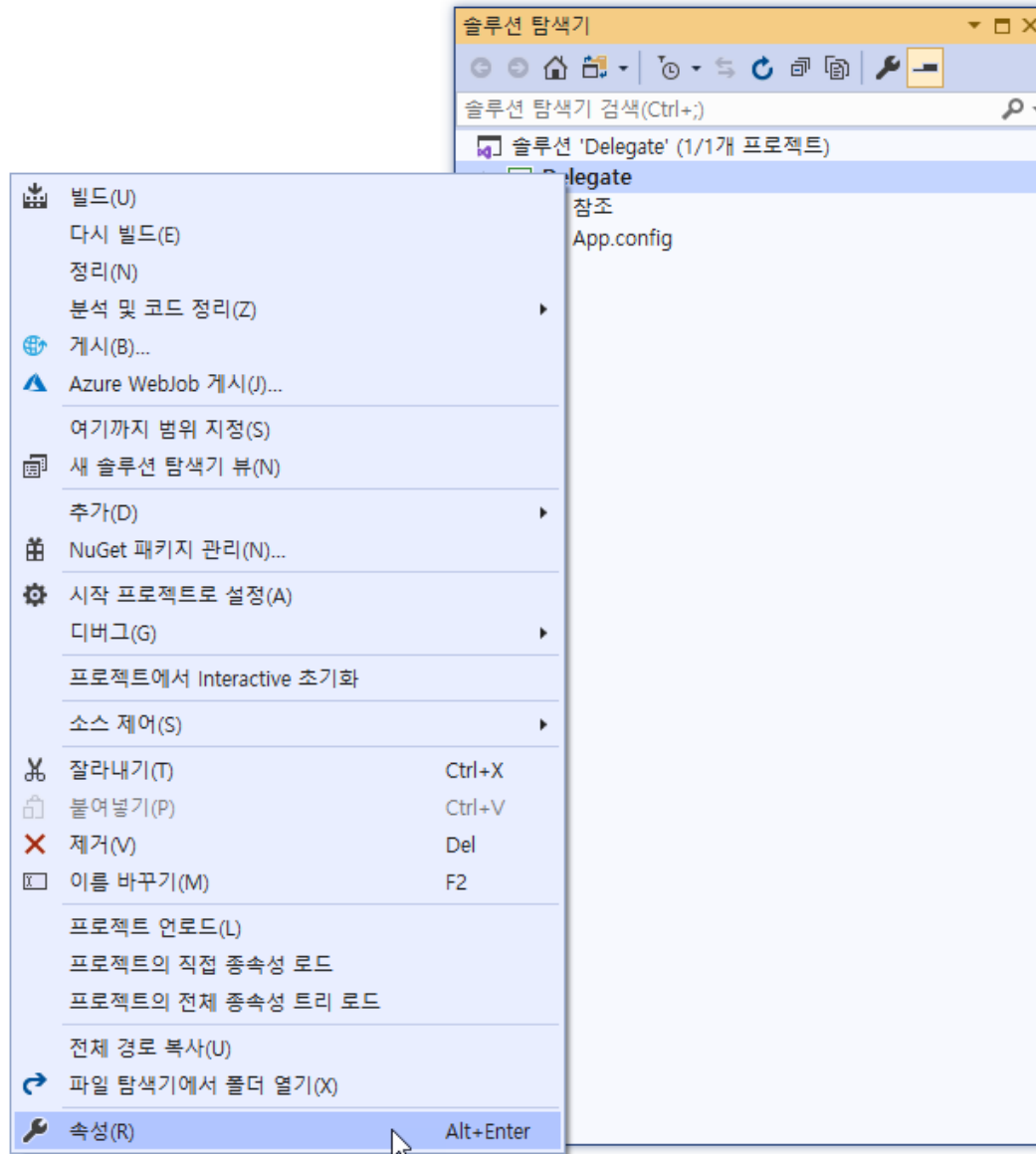
17. 이벤트 처리과정 정리

- 프로그램을 실행하면
 - 닷넷 런타임이 Main 함수 호출
 - Main 함수에서 Run 함수 호출
 - Run 함수에서 폼 객체(길동이) 얼굴(윈도우) 표시 #
 - 이후 Run 함수에서 길동이 호주머니(이벤트 큐)를 계속 확인 (무한루프) #
- 자, 이제 여러분이 폼 위에서 마우스를 클릭 (이벤트 발생)
 - 운영체제(윈도)에서 Click 이벤트를 만들어 호주머니에 넣음.
 - 호주머니를 계속 체크하던 Run 함수에서 이벤트를 꺼내 확인
 - Click 이벤트임을 알고 길동이 OnClick 가상함수 호출
 - OnClick 가상함수 에서 Click 델리게이트가 null이 아니면 여러분이 작성한 핸들러 함수 xxx 호출
- 여러분이 윈도우를 닫을 경우 (이벤트 발생)
 - Close 이벤트 발생
 - 운영체제(윈도)에서 Close 이벤트를 만들어 길동이 호주머니에 넣음.
 - Run 함수에서 길동이 OnClose 가상함수 호출
 - OnClose 함수에서 프로그램 종료

18. 가장 간단한 C# 폼 프로그램

- 새로운 빈 프로젝트 My 생성
- 레퍼런스 2개 추가
 - System.dll
 - System.Windows.Forms.dll
- My.cs 파일 추가
- 가장 간단한 C# 폼 프로그램 작성
- 컴파일 & 실행
- Click 이벤트 처리 (Derived 클래스)

19. 윈도우 응용 설정하기



19. 윈도우 응용 설정하기

Delegate

Delegate

어셈블리 이름(N): Delegate

기본 네임스페이스(L): Delegate

대상 프레임워크(G): .NET Framework 4.7.2

출력 형식(U):
콘솔 애플리케이션
Windows 애플리케이션
콘솔 애플리케이션
클래스 라이브러리

☒ 바인딩 리디렉션 자동 생성(A)

시작 개체(O): (설정 안 함)

어셈블리 정보(I)...

리소스

애플리케이션 리소스 관리 방법을 지정하세요.

☒ 아이콘 및 매니페스트(C)
매니페스트는 애플리케이션의 특정 설정을 결정합니다. 사용자 지정 매니페스트를 포함하려면 먼저 해당 매니페스트를 프로젝트에 추가한 다음 아래 목록에서 선택하세요.

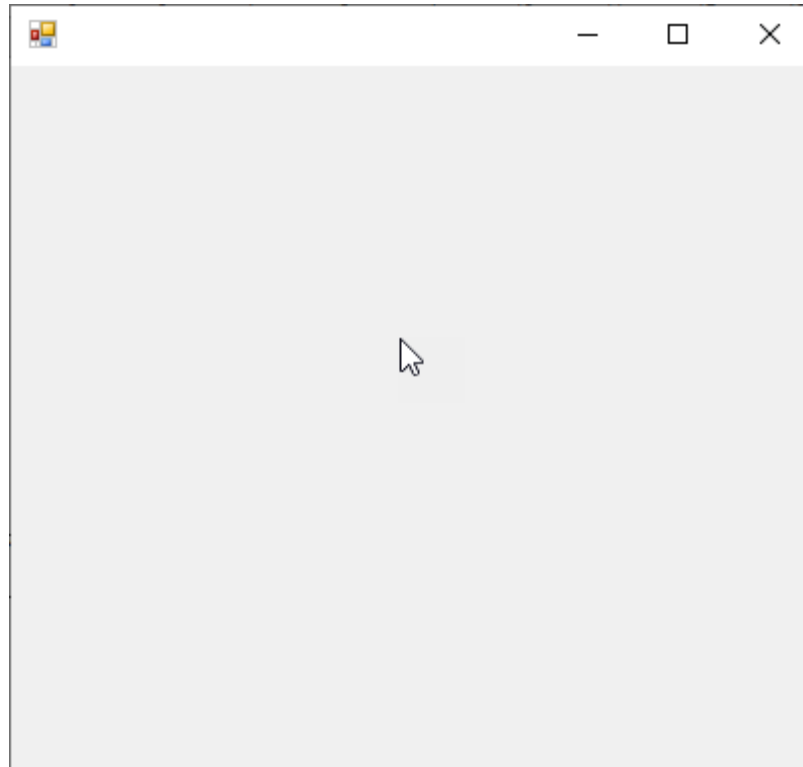
아이콘: (기본 아이콘) 찾아보기...

매니페스트: 기본 설정으로 구성된 매니페스트 포함

☐ 리소스 파일(S): 찾아보기...

19. 윈도우 응용 설정하기

- 도스창은 표시되지 않고 **폼** 윈도우만 표시됨.



20. 코드 다듬기

- Derived 클래스 → MainForm 클래스로 이름 변경
- MainForm 클래스 생성자 함수 내의 코드를 **InitializeComponent** 함수로 추상화
- 네임 스페이스 XXX 지정

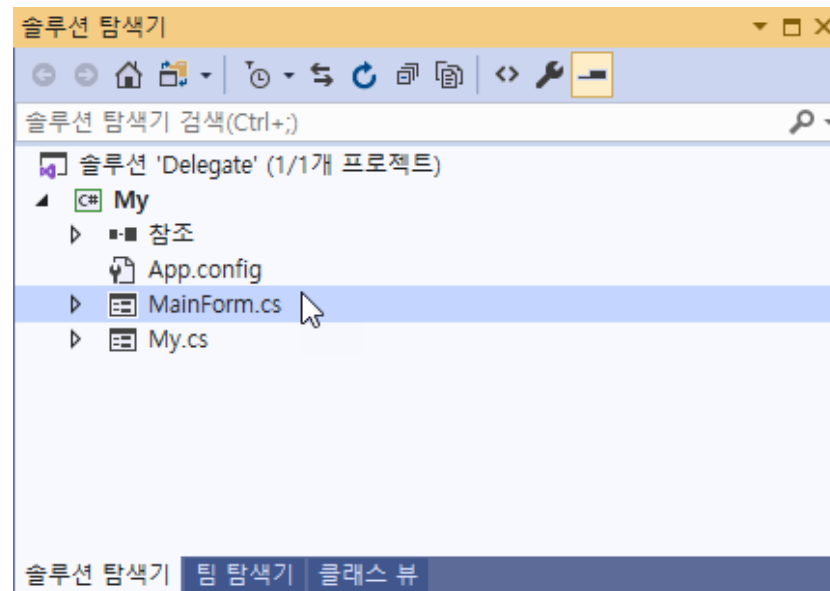
```
namespace XXX  
{  
  
}
```

21. 코드 분리

- 네임 스페이스 XXX 분리 선언
 - MainForm 클래스, My 클래스 따로 선언
- 새로운 파일 MainForm.cs을 새로 작성
 - 프로젝트 | 새 항목 추가
 - 코드 파일 선택
 - MainForm.cs 이름 입력
- MainForm 클래스 이동

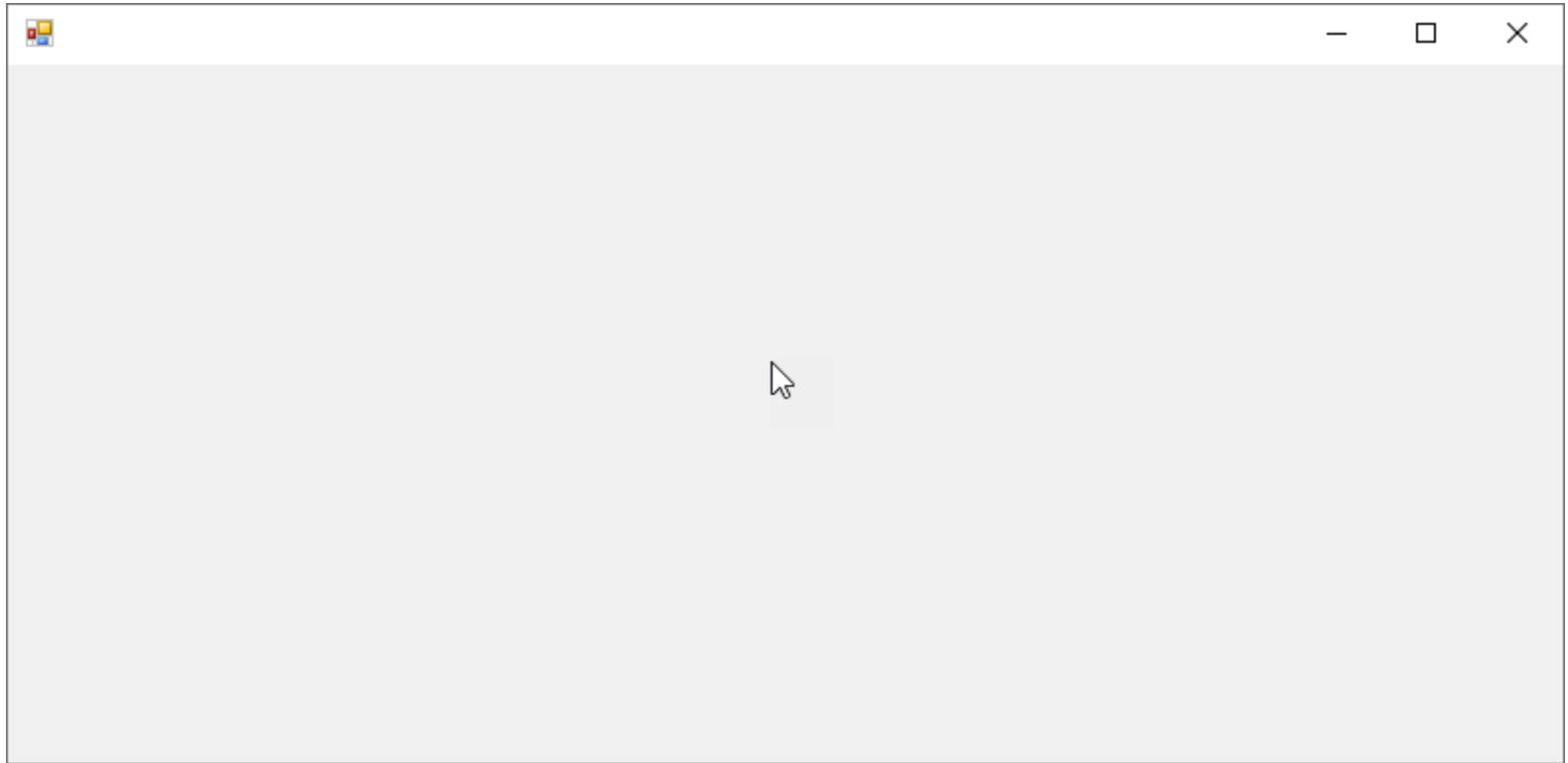
22. 디자인 편집기를 이용한 UI 디자인

- 비주얼 디자인 (MainForm.cs 두 번 클릭)



22. 디자인 편집기를 이용한 UI 디자인

- 폼 윈도우 크기 변경하기

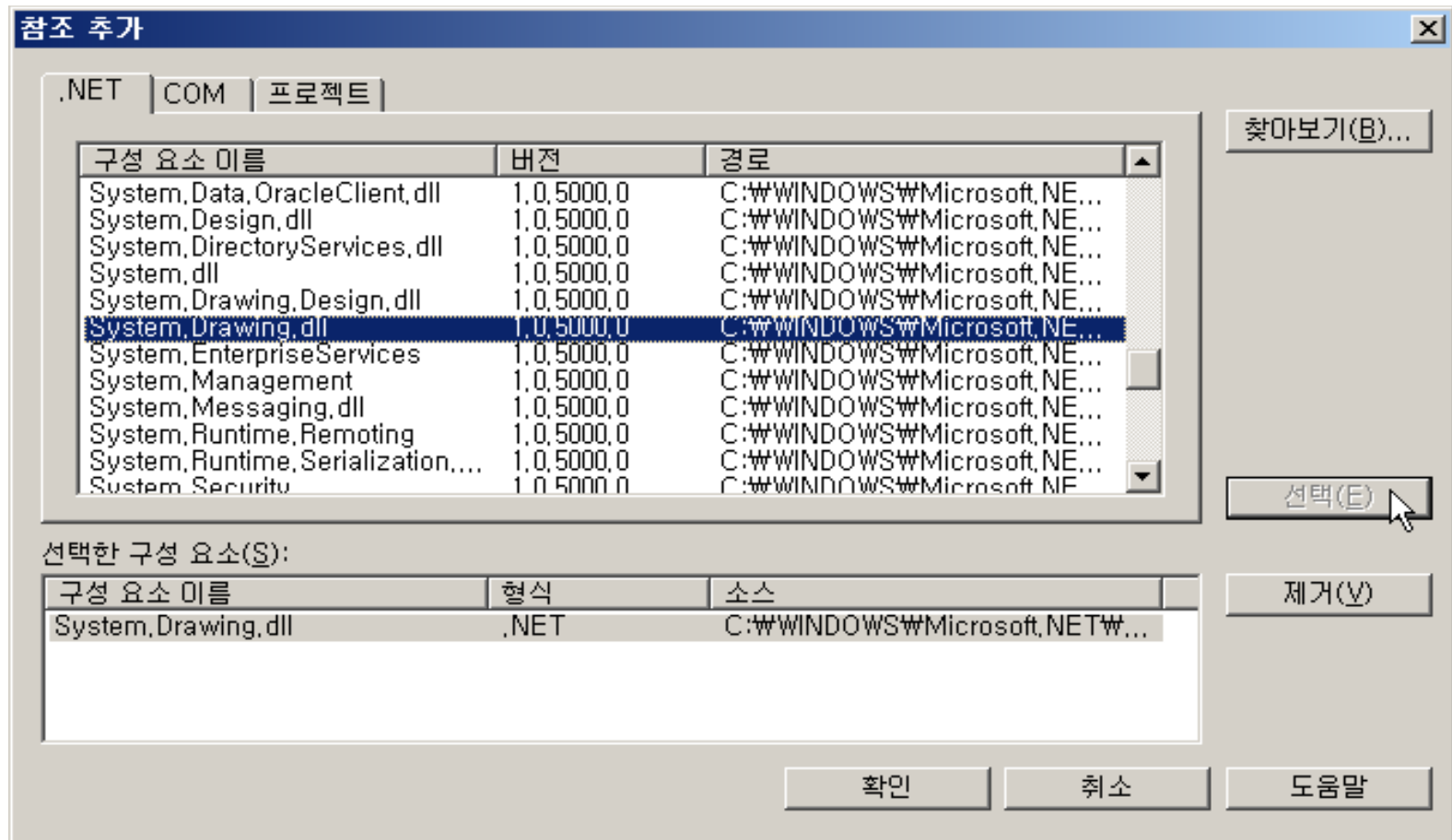


22. 디자인 편집기를 이용한 UI 디자인

```
private void InitializeComponent()
{
    this.SuspendLayout();
    //
    // MainForm
    //
    this.ClientSize = new System.Drawing.Size(440, 209);
    this.Name = "MainForm";
    this.Load += new
        System.EventHandler(this.MainForm_Load);
    this.Click += new System.EventHandler(this.xxx);
    this.ResumeLayout(false);
}
```

System.Drawing.dll
참조를 추가함

22. 디자인 편집기를 이용한 UI 디자인



22. 디자인 편집기를 이용한 UI 디자인

- 디자인 편집기를 이용하면
 - 코드가 **자동으로 생성**됨 → InitializeComponent 멤버 함수에
 - 결국 InitializeComponent 멤버 함수에 자동으로 작성되는 코드는 ‘**디자인 코드**’
 - 디자인 코드가 **아닌** 것도 있지? 가령 이벤트 핸들러 함수인 xxx

23. 디자인코드와 일반코드의 분리

- MainForm 클래스를 하나 더 작성 (partial로)
- InitializeComponent 멤버 함수를 그 곳으로 옮김.

```
public partial class MainForm
{
    private void InitializeComponent()
    {
        디자인 코드가 여기에 들어감!
    }
}
```

23. 디자인코드와 일반코드의 분리

```
using System.Windows.Forms;

public partial class MainForm : Form
{
    public MainForm()
    {
        InitializeComponent();
    }
}
```

```
using System.Windows.Forms;

public class My
{
    static void Main()
    {
        Application.Run(new MainForm());
    }
}
```

24. 품응용프로젝트

- 빈 프로젝트 **My2** 생성
 - Windows Forms 앱(.NET Framework) 프로젝트
- 클래스, 파일 이름 바꾸기 (솔루션 탐색기 창)
 - Form1.cs → **MainForm.cs**
 - Program.cs → **My2.cs**
- 코드 정리 후 분석
- 클래스 이름 바꾸기
- 핸들러 함수 추가 (속성창)
 - Click 이벤트 핸들러 함수 **xxx**