CHAPTER 05

클래스 기본

- 01 클래스 개요
- 02 클래스 사용
- 03 클래스 생성
- 04 클래스의 변수
- 05 추상화

01 클래스 개요

- 세상 모든 것이 객체

TV

의자

책

집

카메라

컴퓨터

01 클래스 개요

- 클래스
  - 객체를 만들어내기 위해 정의된 설계도, 틀
  - 클래스는 객체가 아님. 실체도 아님
  - 멤버 변수와 멤버 함수 선언
- 객체
  - 객체는 생성될 때 클래스의 모양을 그대로 가지고 탄생
  - 멤버 변수와 멤버 함수로 구성
  - 메모리에 생성, 실체(instance)라고도 부름
  - 하나의 클래스 틀에서 찍어낸 여러 개의 객체 생성 가능
  - 객체들은 상호 별도의 공간에 생성

01 클래스 개요

클래스(Class)란?

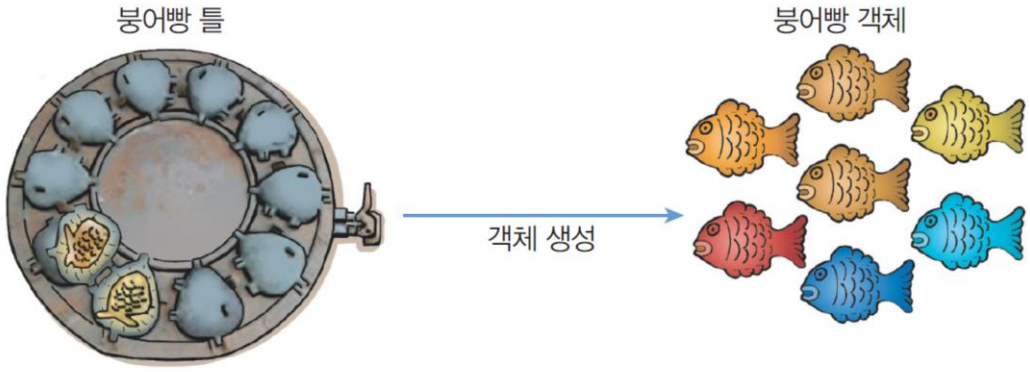
클래스는 객체를 만들기 위한 **틀(설계도)**
속성(데이터)과 동작(메서드)을 함께 정의

```
class Car
{
    public string color;  // 속성(필드)
    public int speed;

    public void Drive()   // 메서드(동작)
    {
        Console.WriteLine("자동차가 달립니다.");
    }
}
```

객체(Object)란?

객체는 클래스를 기반으로 실제로 생성된 **인스턴스(instance)**



(a) 붕어빵 틀과 붕어빵 객체들

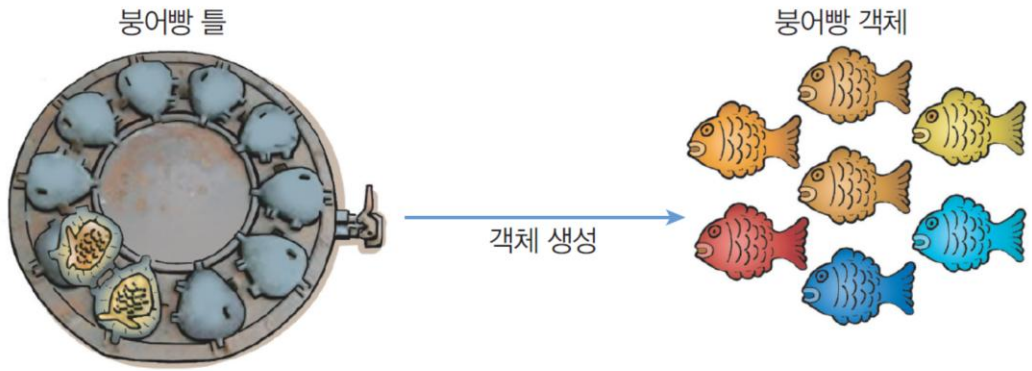
```
Car myCar = new Car();

myCar.color = "빨강";
myCar.speed = 100;

myCar.Drive();
```

01 클래스 개요

구분	클래스	객체
의미	설계도	실제 생성된 것
존재	코드 상	메모리 상
예	Car	myCar



(a) 붕어빵 틀과 붕어빵 객체들

■ 사용자 정의 자료형

- 주차 관리 시스템을 만들 때, 차량 번호(carNumber), 입차 시간(inTime), 출차 시간(outTime) 등을 사용할 것

```
int carNumber;
int inTime;
int outTime;
```

- 변수를 여러 개 만드는 것이 좋음

```
int carNumberA;
int inTimeA;
int outTimeA;

int carNumberB;
int inTimeB;
int outTimeB;

int carNumberC;
int inTimeC;
int outTimeC;
```

■ 사용자 정의 자료형

- 변수가 많아지면 배열이 훨씬 더 깔끔

```
int[] carNumbers = new int[10];
int[] inTimes = new int[10];
int[] outTimes = new int[10];
```

- **클래스 기반의 객체 지향 언어는 클래스를 사용해서 새로운 자료형을 정의**

```
class Car
{
    int carNumber;
    int inTime;
    int outTime;
}

static void Main(string[] args)
{
    Car[] cars = new Car[10];
}
```

■ 사용자 정의 자료형

- 클래스에는 속성을 나타내는 변수 외에도 행위를 나타내는 메서드를 넣을 수 있음

```
class Car
{
    int carNumber;
    DateTime inTime;
    DateTime outTime;

    public void SetInTime()
    {
        this.inTime = DateTime.Now;
    }

    public void SetOutTime()
    {
        this.outTime = DateTime.Now;
    }
}
```

```
static void Main(string[] args)
{
    Car car = new Car();
    car.SetInTime();
    car.SetOutTime();
}
```

■ 클래스와 인스턴스

그림 5-1 클래스와 관련된 기본 용어

- 클래스: 사용자 정의 자료형
- 인스턴스 또는 객체: 자료형을 변수로 선언한 것
- 생성자: 클래스 이름과 같은 메서드(클래스 이름 뒤에 괄호가 붙은 것)

■ Random 클래스

- Random 클래스는 임의의 숫자를 생성할 때에 사용하고, 다음과 같은 방법으로 인스턴스를 생성

Random random = new Random( )

그림 5-2 Random 클래스의 인스턴스에서 사용할 수 있는 메서드

■ Random 클래스

▪ 기본예제 5-1 Random 클래스를 사용한 임의의 정수 생성 /5장/RandomInteger

- **Random 클래스의 Next( ) 메서드는 임의의 정수를 생성할 때에 사용**

코드 5-1 Random 클래스를 사용한 임의의 정수 생성

```
01 static void Main(string[] args)
02 {
03     Random random = new Random();
04     Console.WriteLine(random.Next(10, 100));
05     Console.WriteLine(random.Next(10, 100));
06     Console.WriteLine(random.Next(10, 100));
07     Console.WriteLine(random.Next(10, 100));
08 }
```

실행 결과

```
86
48
76
49
```

■ Random 클래스

▪ **기본예제 5-1** Random 클래스를 사용한 임의의 정수 생성 /5장/RandomInteger

- 예제 코드를 입력하다 보면 자동 완성 기능 화면을 볼 수 있음

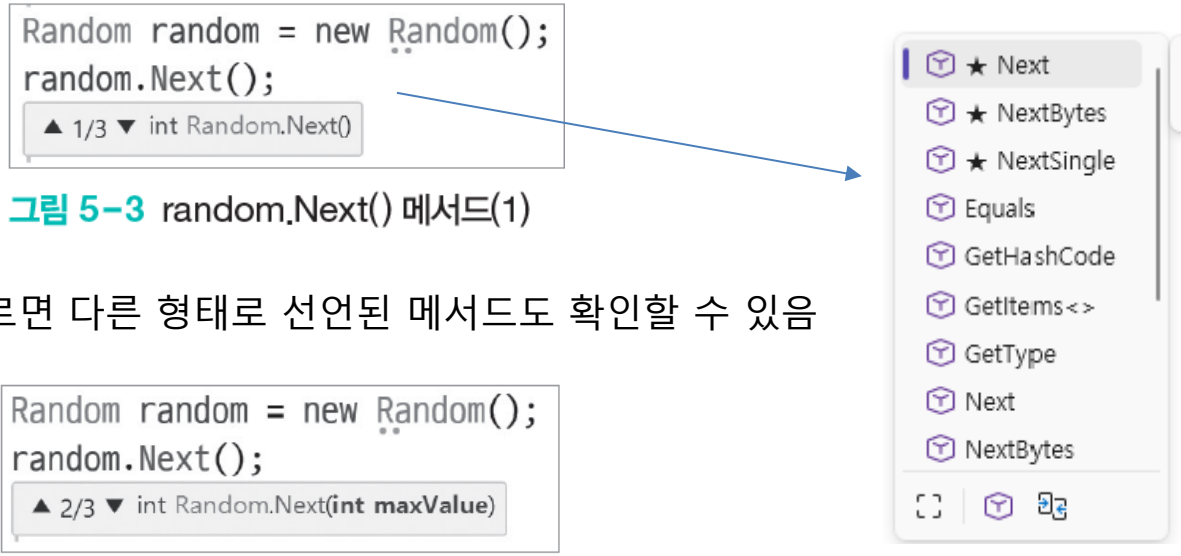


그림 5-3 random.Next() 메서드(1)

- 키보드 방향키를 위 아래로 누르면 다른 형태로 선언된 메서드도 확인할 수 있음

그림 5-4 random.Next() 메서드(2)

■ Random 클래스

▪ **기본예제 5-2** Random 클래스를 사용한 임의의 실수 생성 /5장/RandomDouble

• [코드 5-2]의 **NextDouble( ) 메서드는 0.0과 1.0 사이의 난수(임의의 수)를 반환**

코드 5-2 Random 클래스를 사용한 임의의 실수 생성

```
01 static void Main(string[] args)
02 {
03     Random random = new Random();
04     Console.WriteLine(random.NextDouble());
05     Console.WriteLine(random.NextDouble());
06     Console.WriteLine(random.NextDouble());
07     Console.WriteLine(random.NextDouble());
08 }
```

실행 결과

```
0.385116411086692
0.294557459789588
0.83222587864484
0.910290311514535
```

여기서 잠깐!

■ 원하는 범위의 실수 난수 생성

- 0.0부터 10.0 사이의 난수를 얻고 싶으면 어떻게 해야 할까?

코드 5-3 원하는 범위의 실수 난수 생성 /5장/ClassUses

```
01 static void Main(string[] args)
02 {
03     Random random = new Random();
04     Console.WriteLine(random.NextDouble() * 10);
05     Console.WriteLine(random.NextDouble() * 10);
06     Console.WriteLine(random.NextDouble() * 10);
07     Console.WriteLine(random.NextDouble() * 10);
08 }
```

실행 결과

```
3.66736765190371
5.76678149205017
0.73619941749433
4.62893074593923
```

■ Random 클래스

```
random.Next(10)   //0 이상 10 미만의 정수 난수 생성
random.Next(10, 100) // 10이상 100 미만의 정수 난수 생성
random.NextDouble( ) // 0.0과 1.0 사이의 실수 난수를 반환
```

- 인스턴스 뒤에 점을 찍고 사용하는 멤버를 **인스턴스 멤버** 라고 부름
- 해당멤버가 변수이면 인스턴스 변수, 메서드이면 인스턴스 메서드, 속성이면 인스턴스 속성 등으로 부름

List 클래스

- 다음의 코드로 생성한 배열은 모두 10개의 저장 공간을 고정으로 갖게 됨

코드 5-4 배열 생성 /5장/ClassUses

```
01 static void Main(string[] args)
02 {
03     int[] intArray = new int[10];
04     long[] longArray = new long[10];
05     string[] stringArray = new string[10];
06 }
```

- **List 클래스를 사용하면 크기가 가변적인 배열을 만들 수 있음**
- C#에서는 클래스를 선언할 때 어떤 자료형인지를 알려주기 위해 제네릭(Generic)이라는 것을 사용

List 클래스

- 제네릭은 클래스 이름 뒤에 <와 > 괄호로 감싸 적용

코드 5-5 List 클래스의 인스턴스 생성 /5장/ClassUses

```
01 static void Main(string[] args)
02 {
03     // 인스턴스를 생성합니다.
04     List<int> list = new List<int>();
05 }
```

List 클래스

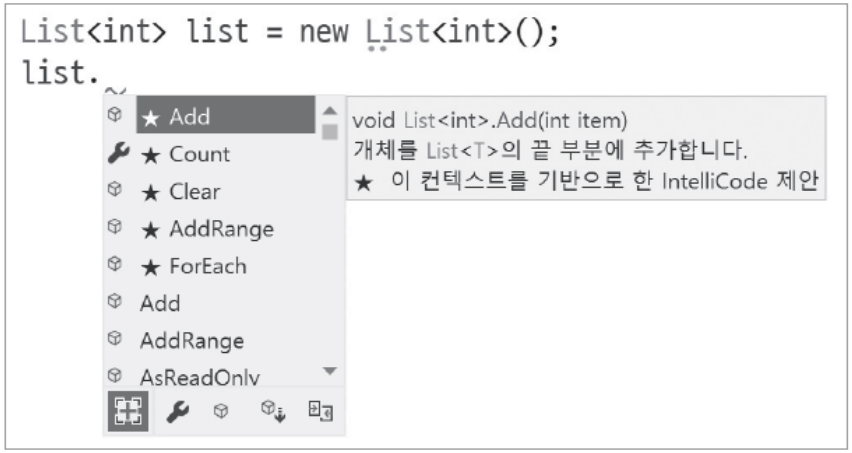


그림 5-9 List 인스턴스의 자동 완성 기능

- Add( ) 메서드를 사용하면 리스트에 요소를 추가할 수 있음

List 클래스

기본예제 5-3 리스트 요소 추가 /5장/ListElementAdd

코드 5-6 리스트 요소 추가

```
01 static void Main(string[] args)
02 {
03     // 인스턴스를 생성합니다.
04     List<int> list = new List<int>();
05
06     // 리스트에 요소를 추가합니다.
07     list.Add(52);
08     list.Add(273);
09     list.Add(32);
10     list.Add(64);
11
12     // 반복을 수행합니다.
13     foreach (var item in list)
14     {
15         Console.WriteLine("Count: " + list.Count + "\titem: " + item);
16     }
17 }
```

실행 결과

```
Count: 4        item: 52
Count: 4        item: 273
Count: 4        item: 32
Count: 4        item: 64
```

List 클래스

기본예제 5-4 리스트 요소 제거 /5장/ListElementRemove

- Remove( ) 메서드는 매개변수로 넣은 요소를 제거

코드 5-8 리스트 요소 제거

```
01 static void Main(string[] args)
02 {
03     // 인스턴스를 생성합니다.
04     List<int> list = new List<int>() { 52, 273, 32, 64 };
05
06     // 요소를 제거합니다.
07     list.Remove(52);
08
09     // 반복을 수행합니다.
10     foreach (var item in list)
11     {
12         Console.WriteLine("Count: " + list.Count + "\titem: " + item);
13     }
14 }
```

실행 결과

```
Count: 3        item: 273
Count: 3        item: 32
Count: 3        item: 64
```

Math 클래스

- Math 클래스는 수학과 관련된 변수 또는 메서드를 제공
- **Math 클래스는 인스턴스를 만들지 않고 클래스 이름 뒤에 . 기호를 찍고 곧바로 멤버를 사용**

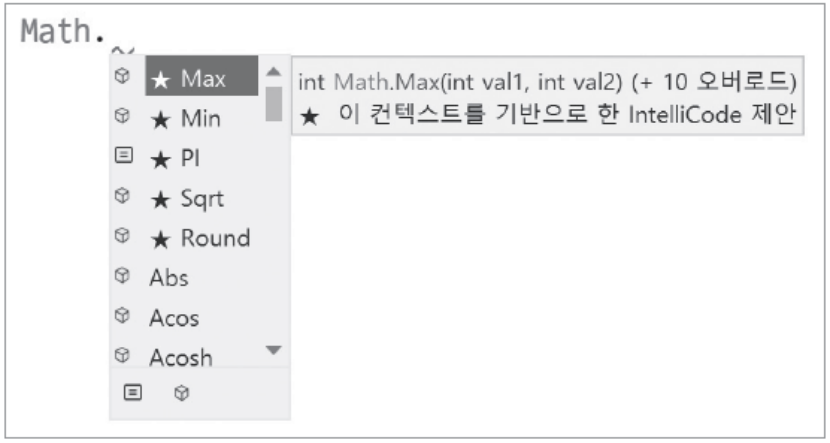


그림 5-10 클래스 메서드

표 5-1 Math 클래스의 메서드

메서드	설명
Abs(숫자)	절대 값을 구합니다.
Ceiling(숫자)	지정된 숫자보다 크거나 같은 최소 정수를 구합니다.
Floor(숫자)	지정된 숫자보다 작거나 같은 최대 정수를 구합니다.
Max(숫자, 숫자)	두 개의 매개변수 중에서 큰 값을 구합니다.
Min(숫자, 숫자)	두 개의 매개변수 중에 작은 값을 구합니다.
Round(숫자)	반올림합니다.

■ Math 클래스

▪ 기본예제 5-5 Math 클래스 활용

/5장/MathBasic

코드 5-9 Math 클래스 활용

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine(Math.Abs(-52273));
04     Console.WriteLine(Math.Ceiling(52.273));
05     Console.WriteLine(Math.Floor(52.273));
06     Console.WriteLine(Math.Max(52, 273));
07     Console.WriteLine(Math.Min(52, 273));
08     Console.WriteLine(Math.Round(52.273));
09 }
```

실행 결과

```
52273
53
52
273
52
52
```

Math 클래스

- 클래스 멤버: Math 클래스처럼 클래스 이름 뒤에 점을 찍고 사용하는 멤버

Math.PI

Math.Abs( )

- 해당 멤버가 변수이면 클래스 변수, 메서드이면 클래스 메서드, 속성이면 클래스 속성이라고 부름

■ 하나의 파일에 여러 개의 클래스 생성

- C# 콘솔 프로젝트를 생성하면 Program.cs라는 C# 파일이 기본으로 생성됨

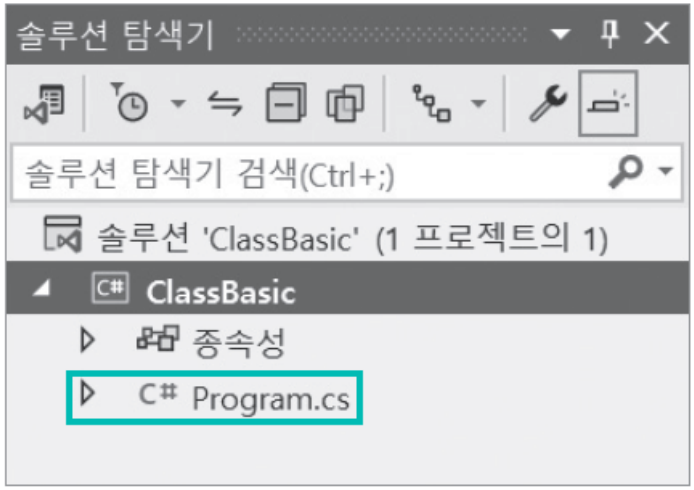


그림 5-13 콘솔 응용 프로그램의 기본 파일 구조

■ 하나의 파일에 여러 개의 클래스 생성

- 파일 내부에 다음과 같이 추가 클래스를 만들 수 있음

```
namespace ClassBasic
{
    class FirstClass
    {
    }

    class SecondClass
    {
    }

    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

■ 하나의 파일에 여러 개의 클래스 생성

- Main( ) 메서드에서 코드를 입력하면 FirstClass와 SecondClass 클래스가 생성된 것을 확인할 수 있음

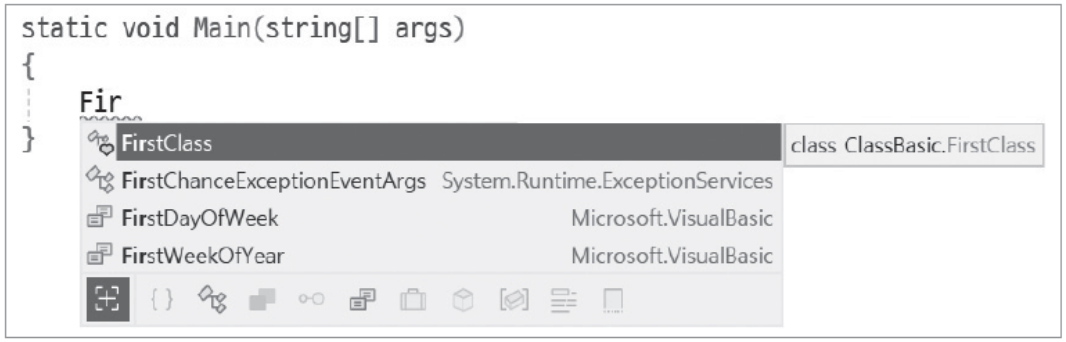


그림 5-14 생성된 클래스의 확인

■ 클래스 내부에 클래스 생성

- 하나의 클래스 내부에도 여러 개의 클래스를 만들어줄 수 있음
- 다음과 같이 Program 클래스 내부에 추가적인 클래스를 만들 수 있음

```
class Program
{
    class FirstClass
    {
    }

    class SecondClass
    {
    }

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

■ 서로 다른 파일에 클래스 생성

- 이렇게 파일을 만들고 클래스를 작성하는 방법을 알아야 함
- 마우스 오른쪽 버튼을 누르고 [추가]-[새 항목] (또는 [New Item])을 선택

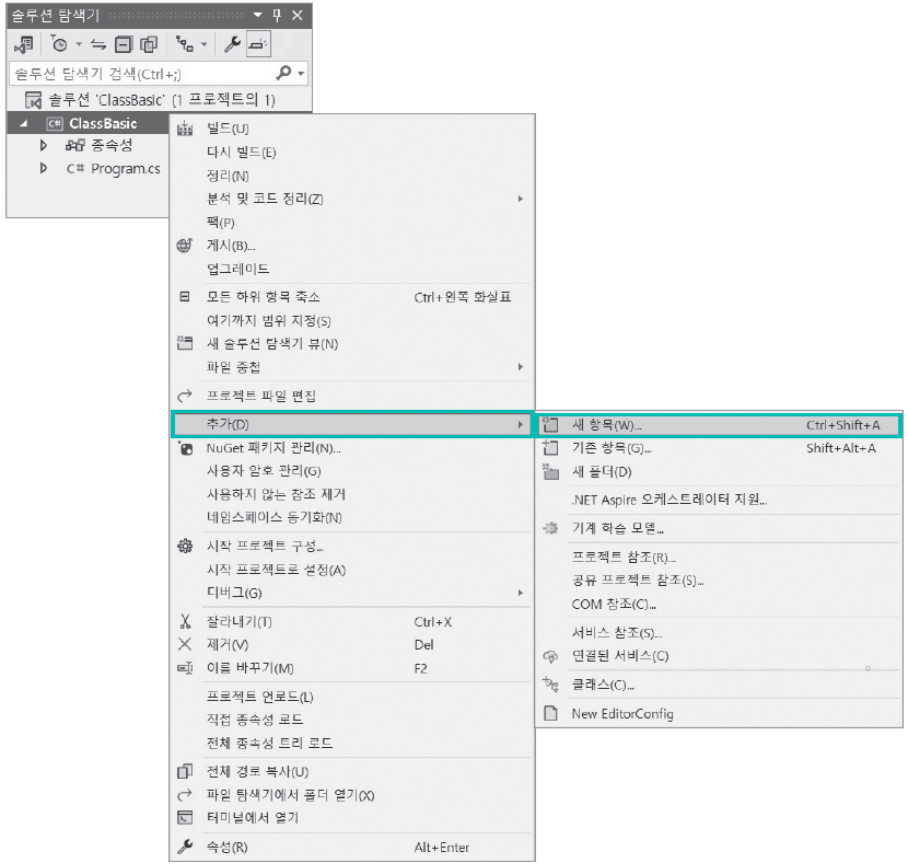


그림 5-15 새 항목 추가 메뉴

■ 서로 다른 파일에 클래스 생성

- [새 항목 추가] 대화상자가 열리면 원하는 클래스 이름(예제에서는 OtherFileClass.cs)을 입력하고 [추가]

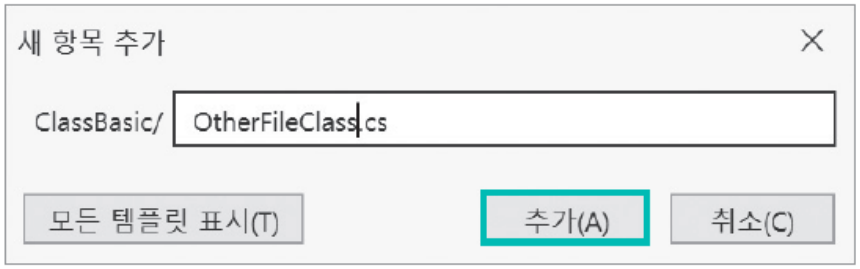


그림 5-16 클래스 추가 방법(1)

■ 서로 다른 파일에 클래스 생성

- [추가]-[새 항목]-[클래스]를 눌러 클래스를 추가

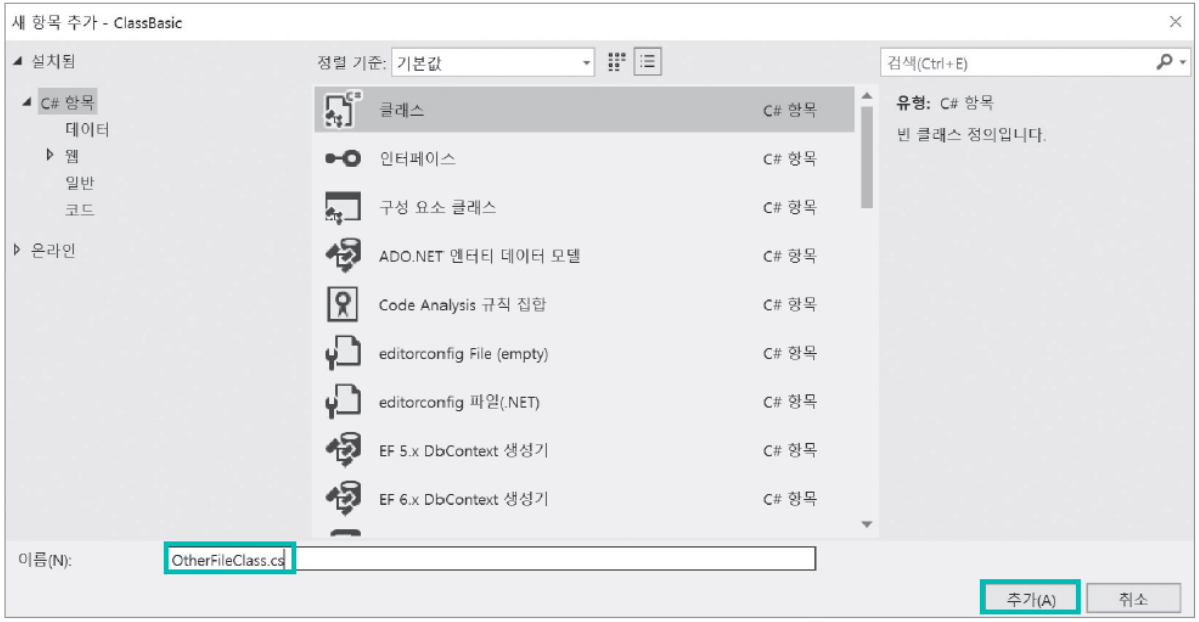


그림 5-17 클래스 추가 방법(2)

■ 서로 다른 파일에 클래스 생성

- 파일 목록에 새로운 파일이 생성되어 있는 것을 확인

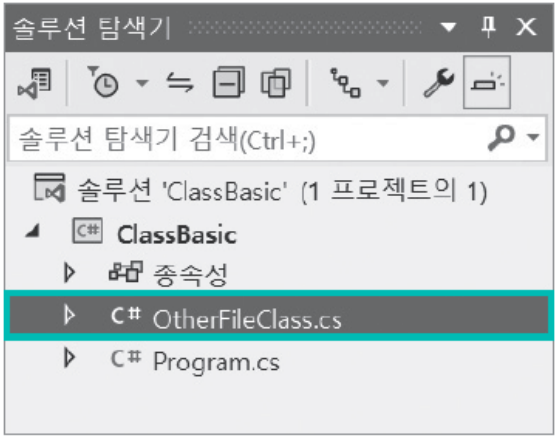


그림 5-18 생성된 클래스 파일

■ 서로 다른 파일에 클래스 생성

- 새로 생성된 파일을 열어보면 파일과 같은 이름의 클래스가 선언되어 있음

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace ClassBasic
{
    internal class OtherFileClass
    {
    }
}
```

■ 인스턴스 변수

- 인스턴스 변수는 [인스턴스].[변수 이름]의 형태로 사용하고 다음과 같은 방법으로 생성

```
[접근 제한자] [자료형] [이름]
```

- 인스턴스 변수는 C# 개발자들 사이에서 소문자로 시작한다는 규칙으로 사용

코드 5-11 인스턴스 변수 선언 /5장/InstanceVariables

```
01 class User
02 {
03     public string name;
04     public string password;
05     public string address;
06     public string phoneNumber;
07     public DateTime regDate;
08 }
```

■ 인스턴트 변수

▪ 기본예제 5-6 인스턴스 변수 생성과 사용 /5장/InstanceVariable

- 인스턴스 변수는 클래스의 내부에 만들어줌

코드 5-12 인스턴스 변수 생성과 사용

```
01 class Program
02 {
03     class Product
04     {
05         public string name;
06         public int price;
07     }
08
09     static void Main(string[] args)
10     {
11         Product product = new Product();
12     }
13 }
```

■ 인스턴트 변수

▪ **기본예제 5-6** 인스턴스 변수 생성과 사용 /5장/InstanceVariable

- 생성한 인스턴스 변수는 지금까지 살펴보았던 다른 멤버처럼 자동 완성 기능에서 확인 할 수 있음

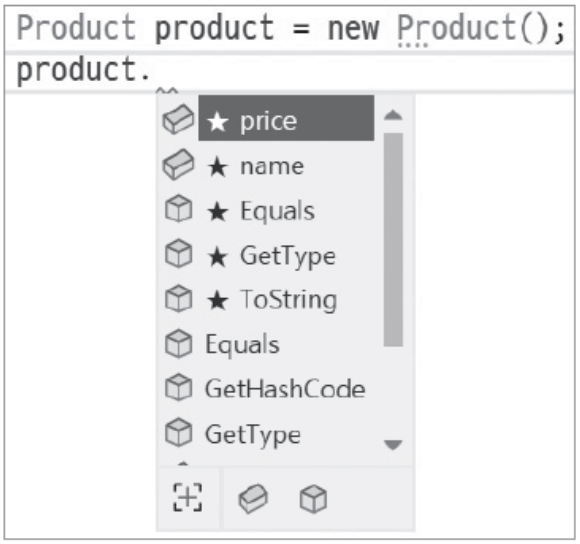


그림 5-22 생성된 인스턴스 변수

■ 인스턴트 변수

▪ 기본예제 5-6 인스턴스 변수 생성과 사용 /5장/InstanceVariable

코드 5-13 인스턴스 변수 사용

```
01  static void Main(string[] args)
02  {
03      // 인스턴스를 생성합니다.
04      Product product = new Product();
05
06      // 인스턴스 변수를 변경합니다.
07      product.name = "감자";
08      product.price = 2000;
09
10      // 인스턴스 변수를 출력합니다.
11      Console.WriteLine(product.name + " : " + product.price + "원");
12  }
```

실행 결과

```
감자 : 2000원
```

■ 인스턴트 변수

- 클래스 변수도 일반적인 지역 변수처럼 생성할 때 초기화할 수 있음

코드 5-14 인스턴스 변수를 생성할 때 초기화 /5장/InstanceVariables

```
01 class Product
02 {
03     public string name = "default";
04     public int price = 1000;
05 }
```

■ 인스턴트 변수

- 인스턴스를 생성할 때 뒤에 중괄호를 사용하고 값을 입력해줌

코드 5-15 인스턴스 변수 생성과 동시에 초기화 /5장/InstanceVariables

```
01  class Program
02  {
03      class Product
04      {
05          public string name;
06          public int price;
07      }
08
09      static void Main(string[] args)
10      {
11          Product productA = new Product() { name = "감자", price = 2000 };
12          Product productB = new Product() { name = "고구마", price = 3000 };
13      }
14  }
```

■ 클래스 변수

- 클래스 이름으로 곧바로 사용하는 변수와 메서드를 클래스 변수와 클래스 메서드라고 부름

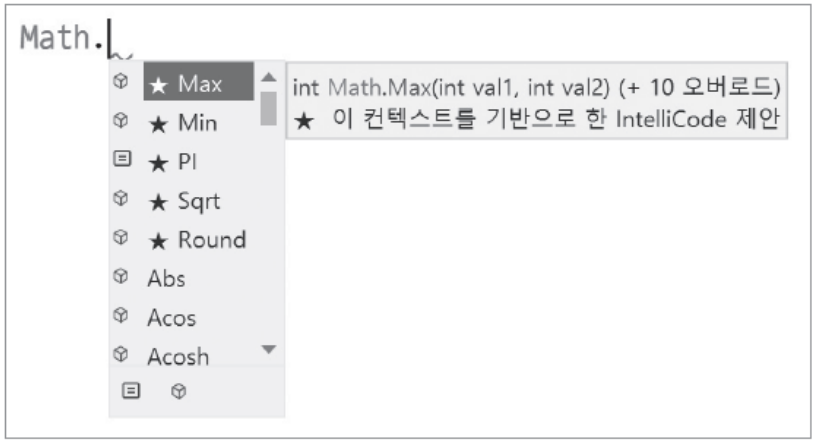


그림 5-23 Math 클래스의 클래스 메서드

■ 클래스 변수

- 클래스 변수를 만들 때는 static 키워드를 사용

```
[접근 제한자] static [자료형] [이름]
```

- **기본예제 5-7** 클래스 변수 생성과 사용 /5장/ClassVariable

코드 5-16 클래스 변수 생성과 사용

```
01 class Program
02 {
03     class MyMath
04     {
05         public static double PI = 3.141592;
06     }
07
08     static void Main(string[] args)
09     {
10         Console.WriteLine(MyMath.PI);
11     }
12 }
```

실행 결과

```
3.141592
```

05 추상화

- 클래스는 왜 사용하는 것일까?
- 일반적으로 클래스 기반의 객체 지향 프로그래밍 언어는 다음 네 가지 특징이 있고, 이것들이 클래스를 사용하는 이유
  - 추상화
  - 캡슐화
  - 상속
  - 다형성
- 여기서는 추상화를 살펴볼 것

05 추상화

- 세상의 모든 사물에는 굉장히 많은 속성이 있음

> 저는 키가 176cm이고…….
> 저는 몸무게가 52kg이고…….
> 저는 어디 학교에 다니고 있고…….
> 저는 어디 학과에 있고…….
> …….

- 계속 생각해보면 더 나옴

> 저는 100미터를 16초 안에 달릴 수 있고…….
> 혼자서 치킨 한 마리를 먹을 수 있어요…….
> …….

05 추상화

- 모든 객체는 자신을 무한 개의 속성으로 나타낼 수 있음

> 저의 눈썹 크기는 1.7cm×8cm이고…….
> 소장의 융털에 번호를 붙였을 때에 1번 융털의 길이는 0.42mm이고…….
> 소장의 융털에 번호를 붙였을 때에 2번 융털의 길이는 0.41mm이고…….
> …….

- 한 객체를 완벽하게 나타내려면 무한한 속성을 모두 입력해야 하지만 그런 성능의 컴퓨터가 없음
- 가장 쉬운 방법은 상황에 맞는 속성만 사용하는 것
- 예를 들어 학생 관리 프로그램에는 학번, 이름, 생년월일, 학과, 학년 등이 핵심적인 속성
- 추상화: 프로그램에 사용되는 핵심적인 부분을 추출하는 것

05 추상화

- 학생이라는 객체에서 핵심적인 부분을 추출한 코드는 다음과 같음

코드 5-17 학생 추상화 /5장/InstanceVariables

```
01 class Student
02 {
03     public string id;
04     public string name;
05     public int grade;
06     public string major;
07     public DateTime birthday;
08
09     /* 계속해서 생각해보세요. */
10 }
```

06 함께하는 응용예제

▪ 응용예제 5-1 모델 클래스와 List 클래스 /5장/ModelClassWithList

- 변수만 가지고 있는 클래스를 모델 클래스라고 부름

코드 5-18 모델 클래스

```
01 class Student
02 {
03     public string id;
04     public string name;
05     public int grade;
06     public string major;
07     public DateTime birthday;
08 }
```

06 함께하는 응용예제

▪ 응용예제 5-1 모델 클래스와 List 클래스 /5장/ModelClassWithList

- 다음 코드는 Student 클래스를 리스트(List 클래스의 인스턴스)에 저장하고 출력

코드 5-19 모델 클래스와 List 클래스

```
01 class Program
02 {
03     class Student
04     {
05         public string name;
06         public int grade;
07     }
08
09     static void Main(string[] args)
10     {
11         List<Student> list = new List<Student>();
12         list.Add(new Student() { name = "윤인성", grade = 1 });
13         list.Add(new Student() { name = "연하진", grade = 2 });
14         list.Add(new Student() { name = "윤아린", grade = 3 });
15         list.Add(new Student() { name = "윤명월", grade = 4 });
16         list.Add(new Student() { name = "구지연", grade = 1 });
17         list.Add(new Student() { name = "김연화", grade = 2 });
18
19         foreach (var item in list)
20         {
21             Console.WriteLine(item.name + " : " + item.grade);
22         }
23     }
24 }
```

실행 결과

```
윤인성 : 1
연하진 : 2
윤아린 : 3
윤명월 : 4
구지연 : 1
김연화 : 2
```

06 함께하는 응용예제

▪ 응용예제 5-1 모델 클래스와 List 클래스 /5장/ModelClassWithList

- 리스트를 처음 생성할 때 다음과 같이 초기화하는 것도 가능

코드 5-20 리스트와 모델 클래스 동시 초기화

```
01 List<Student> list = new List<Student>()
02 {
03     new Student() { name = "윤인성", grade = 1 },
04     new Student() { name = "연하진", grade = 2 },
05     new Student() { name = "윤아린", grade = 3 },
06     new Student() { name = "윤명월", grade = 4 },
07     new Student() { name = "구지연", grade = 1 },
08     new Student() { name = "김연화", grade = 2 }
09 };
```

06 함께하는 응용예제

- **응용예제 5-2** List 클래스 요소 제거와 역 반복문 /5장/ElementRemoveWithReverse
  - 응용예제를 기반으로 1학년보다 학년이 높은 학생을 리스트에서 제거하는 코드를 작성
  - 리스트(List 클래스의 인스턴스)에서 요소를 제거할 때는 [표 5-2]의 메서드를 사용

표 5-2 리스트 요소 제거에 사용하는 메서드

메서드	설명
Remove(object element)	특정 요소를 리스트에서 제거합니다(객체를 지정).
RemoveAt(int index)	특정 위치에 있는 요소를 리스트에서 제거합니다(인덱스를 지정).

06 함께하는 응용예제

■ **응용예제 5-2** List 클래스 요소 제거와 역 반복문 /5장/ElementRemoveWithReverse

• foreach 반복문 내부에서 조건문으로 grade 속성이 1보다 큰 요소를 찾아 제거

코드 5-21 foreach 반복문으로 요소 제거

```
01 class Program
02 {
03     class Student
04     {
05         public string name;
06         public int grade;
07     }
08
09     static void Main(string[] args)
10     {
11         List<Student> list = new List<Student>();
12         list.Add(new Student() { name = "윤인성", grade = 1 });
13         list.Add(new Student() { name = "연하진", grade = 2 });
14         list.Add(new Student() { name = "윤아린", grade = 3 });
15         list.Add(new Student() { name = "윤명월", grade = 4 });
16         list.Add(new Student() { name = "구지연", grade = 1 });
17         list.Add(new Student() { name = "김연화", grade = 2 });
```

06 함께하는 응용예제

- **응용예제 5-2** List 클래스 요소 제거와 역 반복문 /5장/ElementRemoveWithReverse

```
18
19          foreach (var item in list)
20          {
21              if (item.grade > 1)
22              {
23                  list.Remove(item);
24              }
25          }
26
27          foreach (var item in list)
28          {
29              Console.WriteLine(item.name + " : " + item.grade);
30          }
31      }
32  }
```

06 함께하는 응용예제

- **응용예제 5-2** List 클래스 요소 제거와 역 반복문 /5장/ElementRemoveWithReverse
  - 코드를 실행하면 예외가 발생
  - foreach 반복문 내부에서는 반복되고 있는 리스트에 추가 또는 제거가 불가능

실행 결과

```
처리되지 않은 예외: System.InvalidOperationException: 컬렉션이 수정되었습니다. 열거 작업이 실행
되지 않을 수도 있습니다.
   위치: System.ThrowHelper.ThrowInvalidOperationException(ExceptionResource resource)
   위치: System.Collections.Generic.List`1.Enumerator.MoveNextRare()
   위치: System.Collections.Generic.List`1.Enumerator.MoveNext()
…생략…
```

06 함께하는 응용예제

■ **응용예제 5-2** List 클래스 요소 제거와 역 반복문 /5장/ElementRemoveWithReverse

- for 반복문을 사용하면 어떨까?

코드 5-22 for 반복문으로 요소 제거

```
01 for (int i = 0; i < list.Count; i++)
02 {
03     if (list[i].grade > 1)
04     {
05         list.RemoveAt(i);
06     }
07 }
```

실행 결과

```
윤인성 : 1
윤아린 : 3
구지연 : 1
```

- 실행은 되지만 실행 결과가 원하는 대로 되지 않음
- 이는 요소가 지워지면서 인덱스가 밀려서 발생하는 일

06 함께하는 응용예제

- **응용예제 5-2** List 클래스 요소 제거와 역 반복문 /5장/ElementRemoveWithReverse
  - 요소가 4개 있을 때 문제가 생기는 부분까지의 실행 결과
    - 인덱스가 0일 때
      - » 리스트에는 윤인성(0), 연하진(1), 윤아린(2), 윤명월(3)
      - » 0번째 요소 확인: 윤인성 => 유지
    - 인덱스가 1일 때
      - » 리스트에는 윤인성(0), 연하진(1), 윤아린(2), 윤명월(3)
      - » 1번째 요소 확인: 연하진 => 제거
    - 인덱스가 2일 때
      - » 리스트에는 윤인성(0), 윤아린(1), 윤명월(2)
      - » 2번째 요소 확인: 윤명월 => 제거[윤아린을 검사하지 않고 넘어감]
  - 요소가 하나 제거되면서 인덱스가 앞으로 하나씩 밀리면서 생략하고 지나가는 일이 발생해버린 것

06 함께하는 응용예제

- **응용예제 5-2** List 클래스 요소 제거와 역 반복문 /5장/ElementRemoveWithReverse
  - 역 for 반복문을 사용하면 됨

코드 5-23 역 for 반복문을 사용한 요소 제거

```
01  for (int i = list.Count - 1; i >= 0; i--)
02  {
03      if (list[i].grade > 1)
04      {
05          list.RemoveAt(i);
06      }
07  }
```

실행 결과

```
윤인성 : 1
구지연 : 1
```

  - 리스트의 요소를 제거할 때는 반드시 역 for 반복문을 사용

07 윈도 폼: 윈도 폼 기본 익히기

■ 프로젝트 생성

- [파일]-[새로 만들기]-[프로젝트]
- Windows Forms 앱(.NET Framework)을 선택

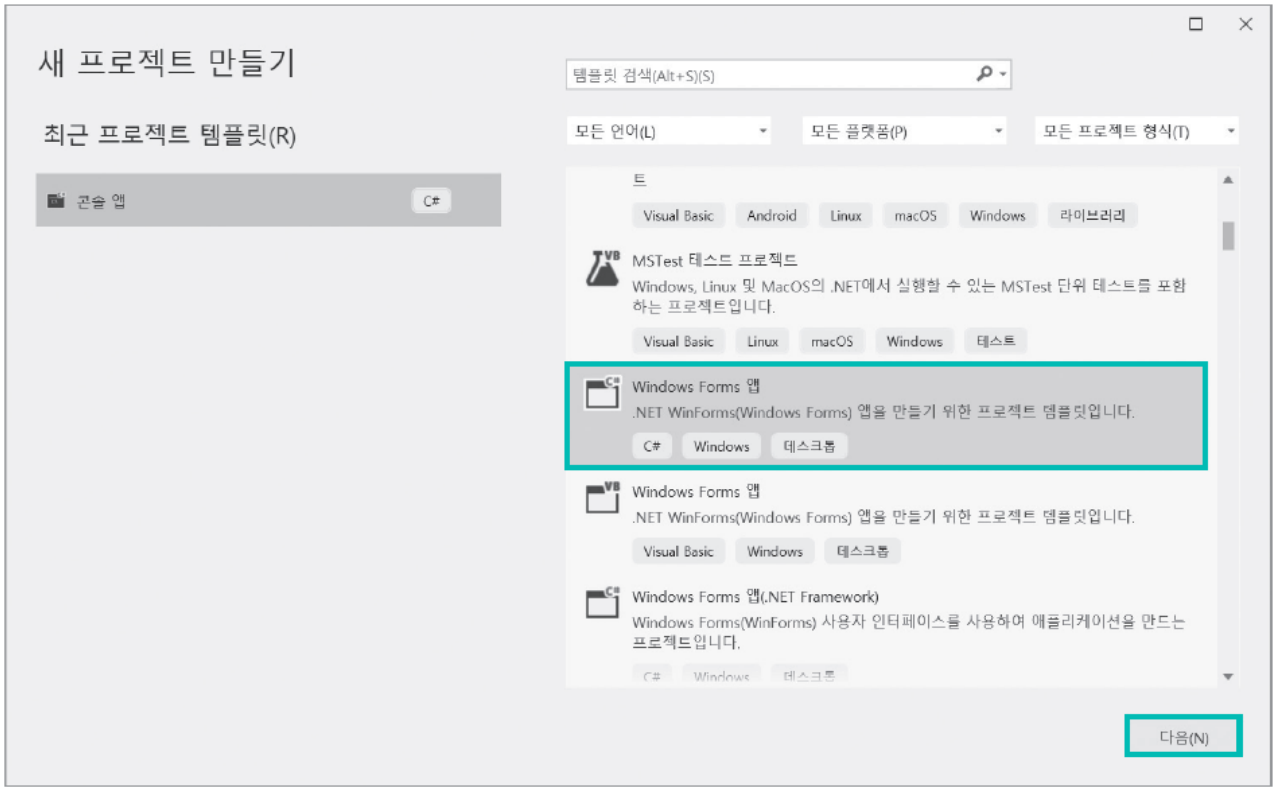


그림 5-24 새 프로젝트 대화상자

07 윈도 폼: 윈도 폼 기본 익히기

■ 프로젝트 생성

- 프로젝트의 이름은 "FirstFormApplication"으로 지정

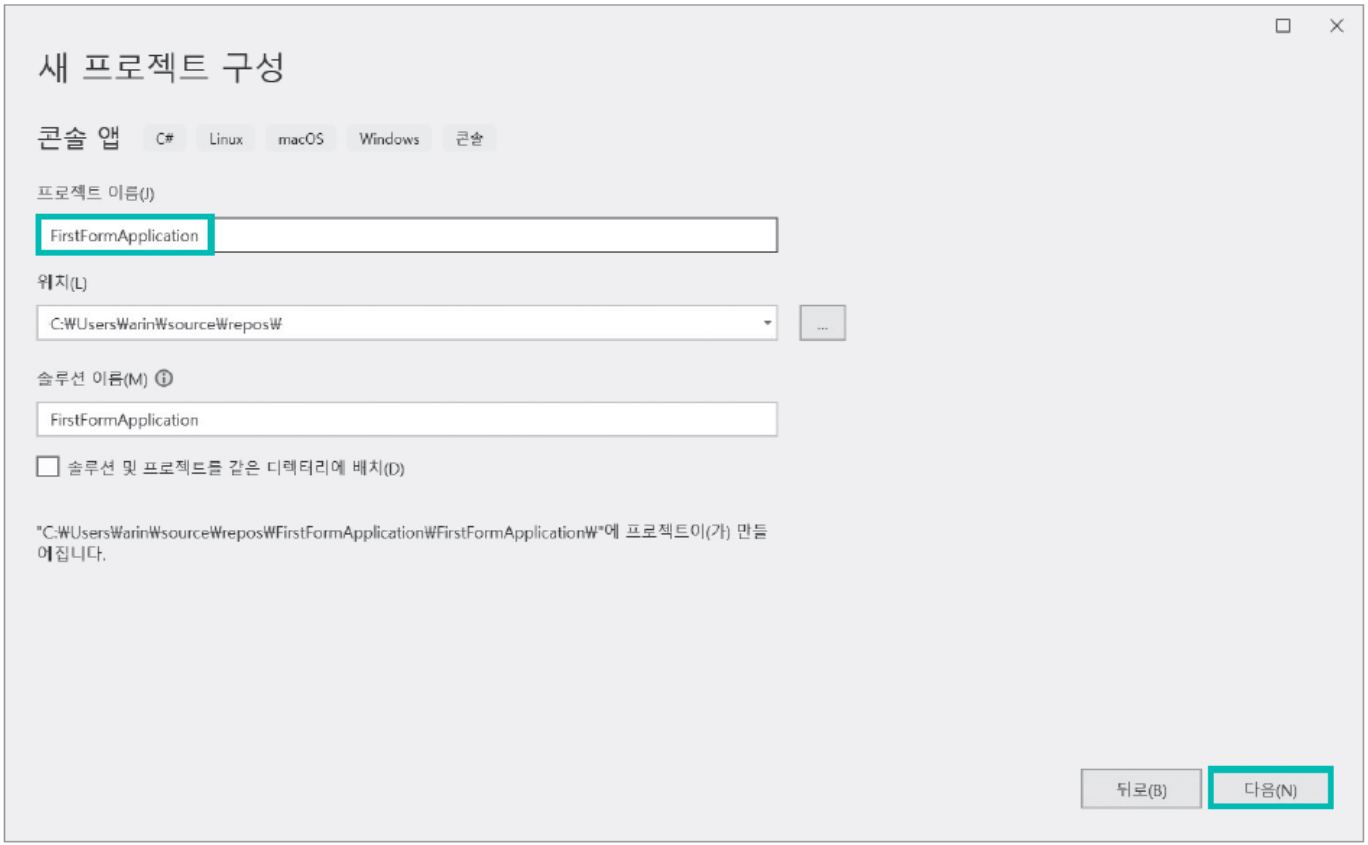


그림 5-25 프로젝트 이름 지정

■ 프로젝트 생성

- 대상 프레임워크 선택 화면이 나오면, [.NET8.0(장기 지원)]으로 지정하고, [만들기]

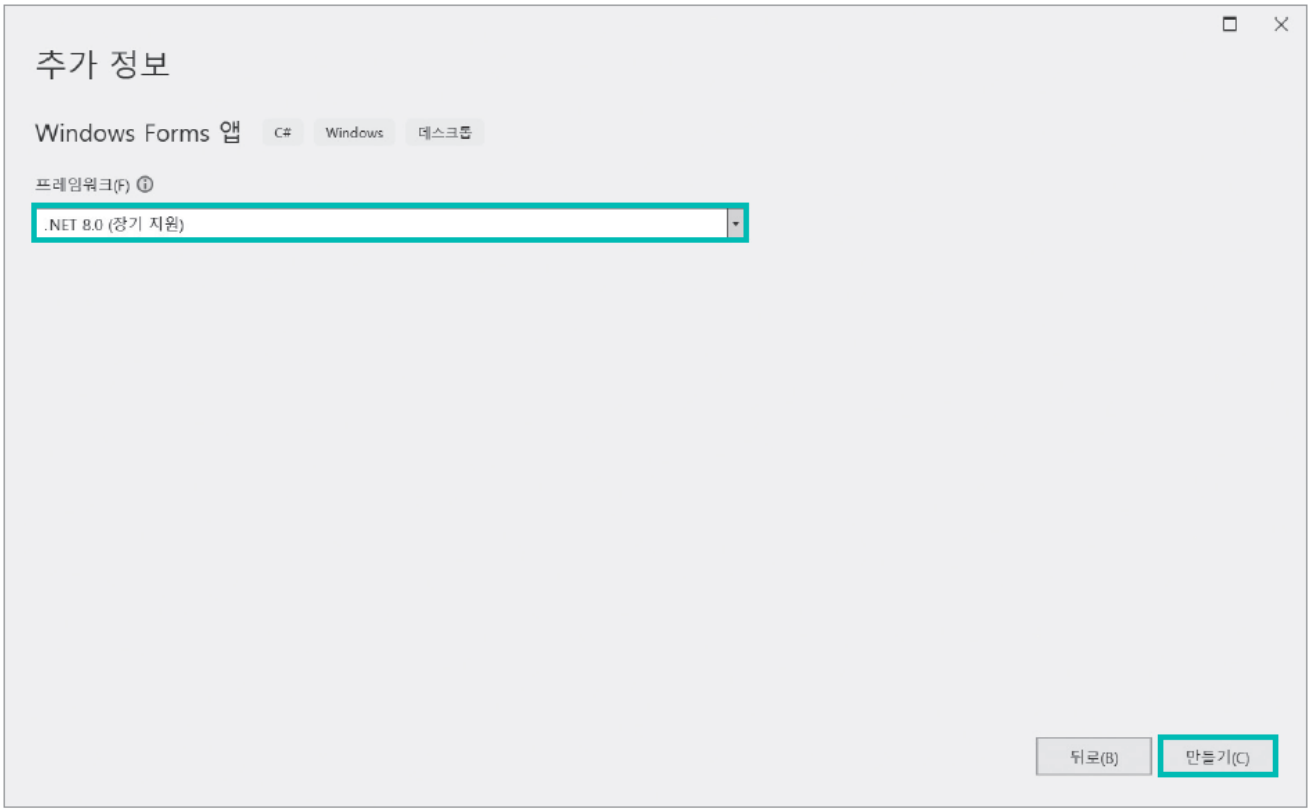


그림 5-26 대상 프레임워크 선택 화면

■ 프로젝트 생성

- 프로젝트를 실행하면 다음과 같이 GUI 화면이 함께 있는 응용 프로그램이 실행됨

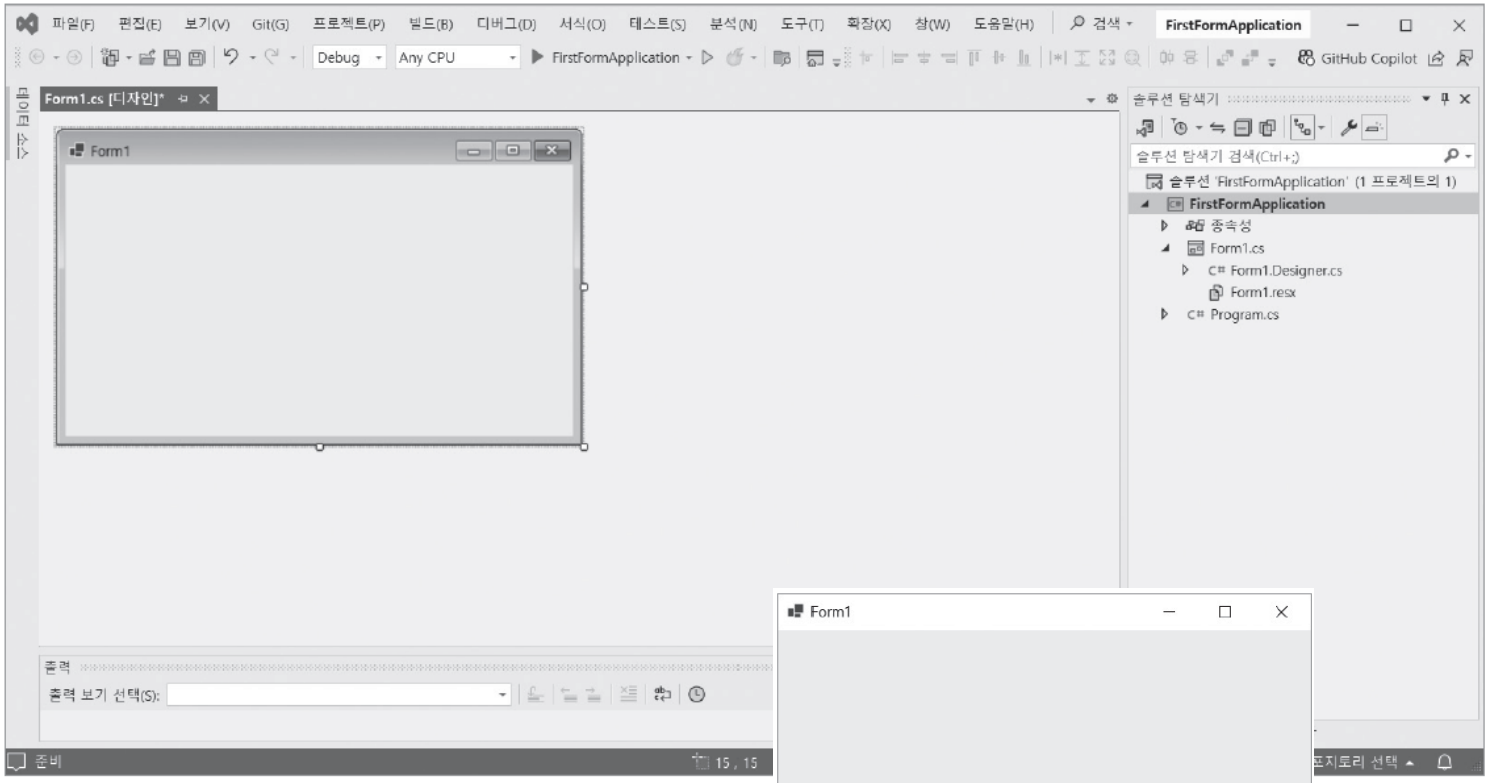


그림 5-27 생성된 프로젝트

그림 5-28 실행된 프로젝트

07 윈도 폼: 윈도 폼 기본 익히기

■ 윈도 폼의 기본 구조

- 솔루션 탐색기를 보면 윈도 폼 응용 프로그램은 [그림 5-29]와 같은 구조

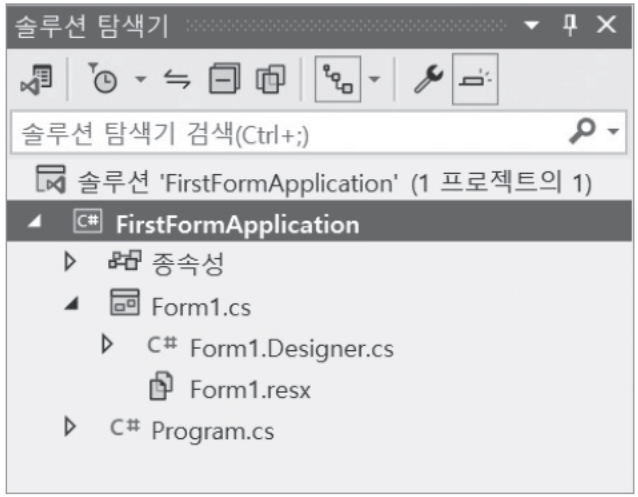


그림 5-29 윈도 폼 응용 프로그램 프로젝트의 기본 구조

07 윈도 폼: 윈도 폼 기본 익히기

■ 윈도 폼의 기본 구조

- Program.cs 파일은 지금까지 살펴보았던 것처럼 Main( ) 메서드가 있는 파일
- 응용 프로그램과 관련된 설정을 수행하고 new Form1( )로 Form1 클래스의 인스턴스를 생성하고 실행하는 코드

코드 5-24 Program.cs 파일

```
01 namespace FirstFormApplication
02 {
03     internal static class Program
04     {
05         /// <summary>
06         ///  The main entry point for the application.
07         /// <summary>
08         [STAThread]
09         static void Main()
10         {
11             // To customize application configuration such as set high DPI
                   settings or default font,
12             // see https://aka.ms/applicationconfiguration.
13             ApplicationConfiguration.Initialize();
14             Application.Run(new Form1());
15         }
16     }
17 }
```

Form1 클래스의 인스턴스를 생성하고 실행합니다.

07 윈도 폼: 윈도 폼 기본 익히기

■ 윈도 폼의 기본 구조

- Form1.cs 파일을 더블클릭하면 다음과 같은 화면이 나옴

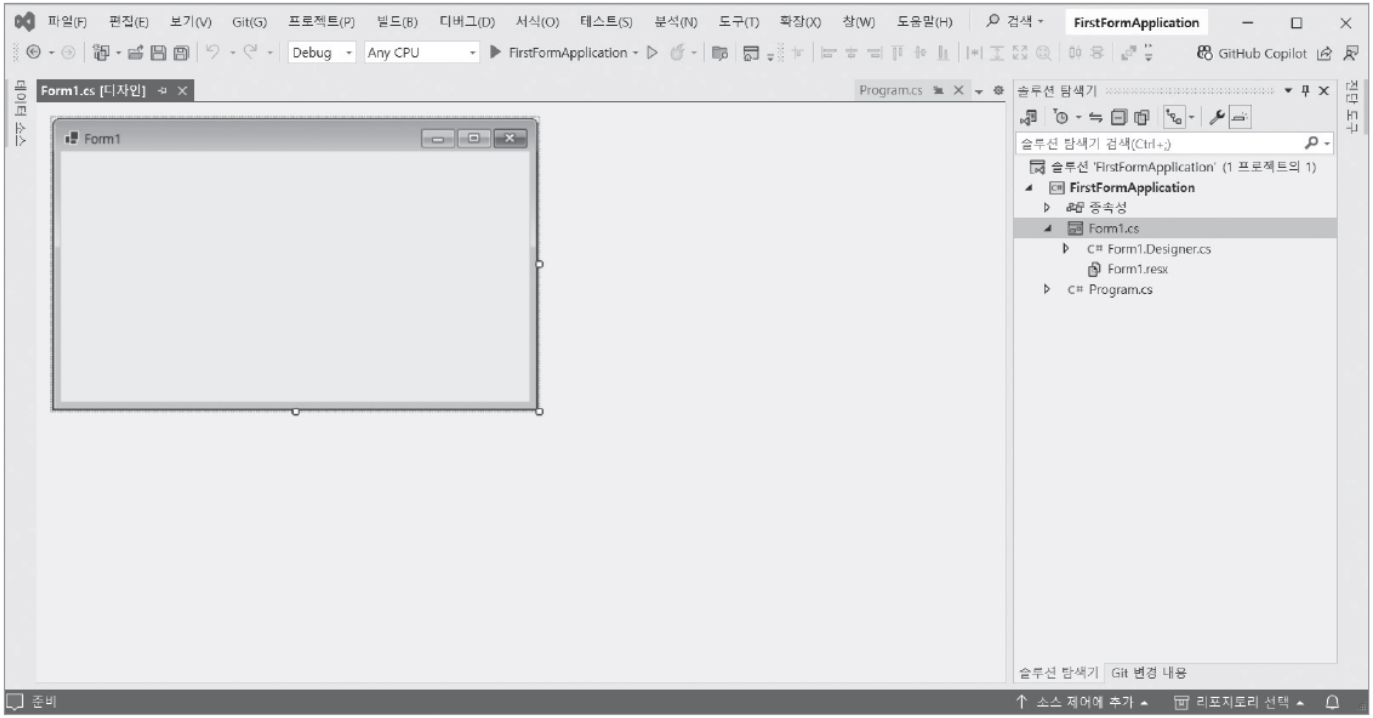


그림 5-30 Form1.cs 파일을 열었을 때의 화면

07 윈도 폼: 윈도 폼 기본 익히기

■ 윈도 폼의 기본 구조

- 이 화면은 바로 Form1 클래스의 디자인을 지정하는 부분
- 지정된 디자인은 [그림 5-30]에 있는 Form1.Designer.cs 파일에 자동으로 반영됨
- 간단하게 Form1.Designer.cs 파일을 열어보기

코드 5-25 Form1.Designer.cs 파일

```
01 namespace FirstFormApplication
02 {
03     partial class Form1
04     {
05         /// <summary>
06         /// Required designer variable
07         /// </summary>
08 
09         private System.ComponentModel.IContainer components = null;
10         /// <summary>
11         ///  Clean up any resources being used.
12         /// </summary>
13         /// <param name="disposing">true if managed resources should be disposed;
               otherwise, false.</param>
14         protected override void Dispose(bool disposing)
15         {
```

07 윈도 폼: 윈도 폼 기본 익히기

■ 윈도 폼의 기본 구조

```
16              if (disposing && (components != null))
17              {
18                  components.Dispose();
19              }
20              base.Dispose(disposing);
21          }
22
23          #region Windows Form Designer generated code
24
25          /// <summary>                                    ← 써있는 그대로, 절대로 수정하지 마세요.
26          ///  Required method for Designer support - do not modify
27          ///  the contents of this method with the code editor.
28          /// </summary>
29          private void InitializeComponent()
30          {
31              this.components = new System.ComponentModel.Container();
32              this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
33              this.ClientSize = new System.Drawing.Size(800, 450);
34              this.Text = "Form1";
35          }
36
37          #endregion
38      }
39  }
```

■ 윈도 폼의 기본 구조

- Form1.cs 파일에서 마우스 오른쪽 버튼을 누르고 [코드 보기]를 선택

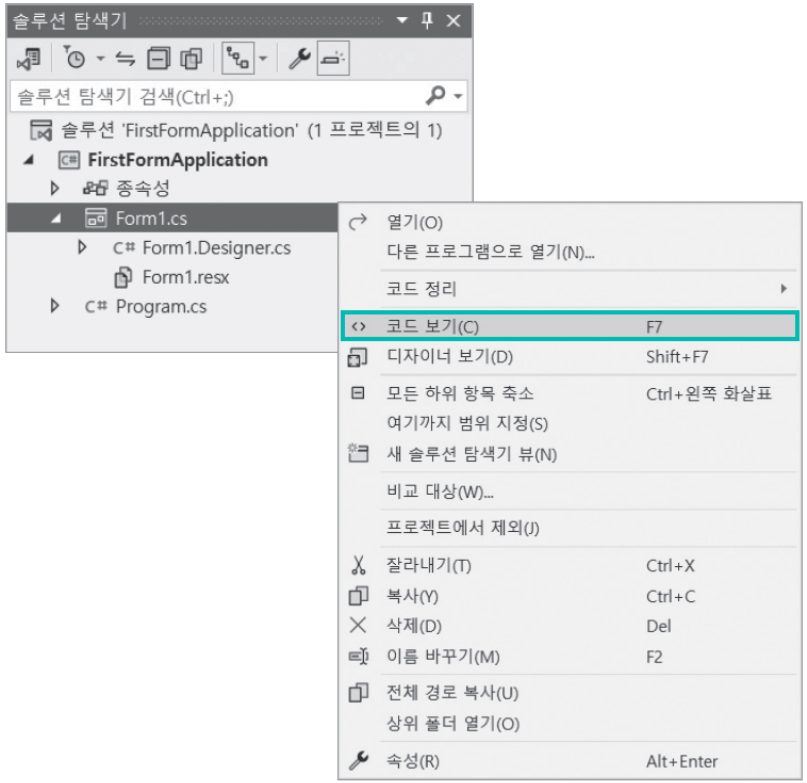


그림 5-31 Form1.cs 파일의 코드 보기

■ 윈도 폼의 기본 구조

- 다음과 같은 코드가 나옴
- 이것이 바로 Form1.cs 파일의 본체이며 여기에 우리가 원하는 코드를 작성

코드 5-26 Form1.cs 파일

```
01 namespace FirstFormApplication
02 {
03     public partial class Form1 : Form
04     {
05         public Form1()
06         {
07             InitializeComponent();
08         }
09     }
10 }
```

여기서 잠깐!

■ partial 클래스

- partial 키워드를 붙이면 클래스를 여러 곳에서 정의할 수 있음

코드 5-27 일반적인 클래스

```
01 class Example
02 {
03     public int a;
04     public int b;
05 }
```

- 지금은 간단한 클래스이므로 상관없지만 이후에 규모가 커지면 별로 보여주고 싶지 않은 지저분한 코드들이 있을것

■ partial 클래스

- partial 키워드를 사용하면 [코드 5-27]의 클래스를 다음과 같이 분할할 수 있음

코드 5-28 partial 클래스

```
01 partial class Example
02 {
03     public int a;
04 }
05
06 partial class Example
07 {
08     public int b;
09 }
```

■ 디자인 화면에서 요소 생성

- Form1.cs를 연 상태에서 화면의 왼쪽에 있는 도구 상자를 눌러서 도구 상자를 열면 모든 Windows Forms에 사용할 수 있는 다 양한 요소들이 나옴

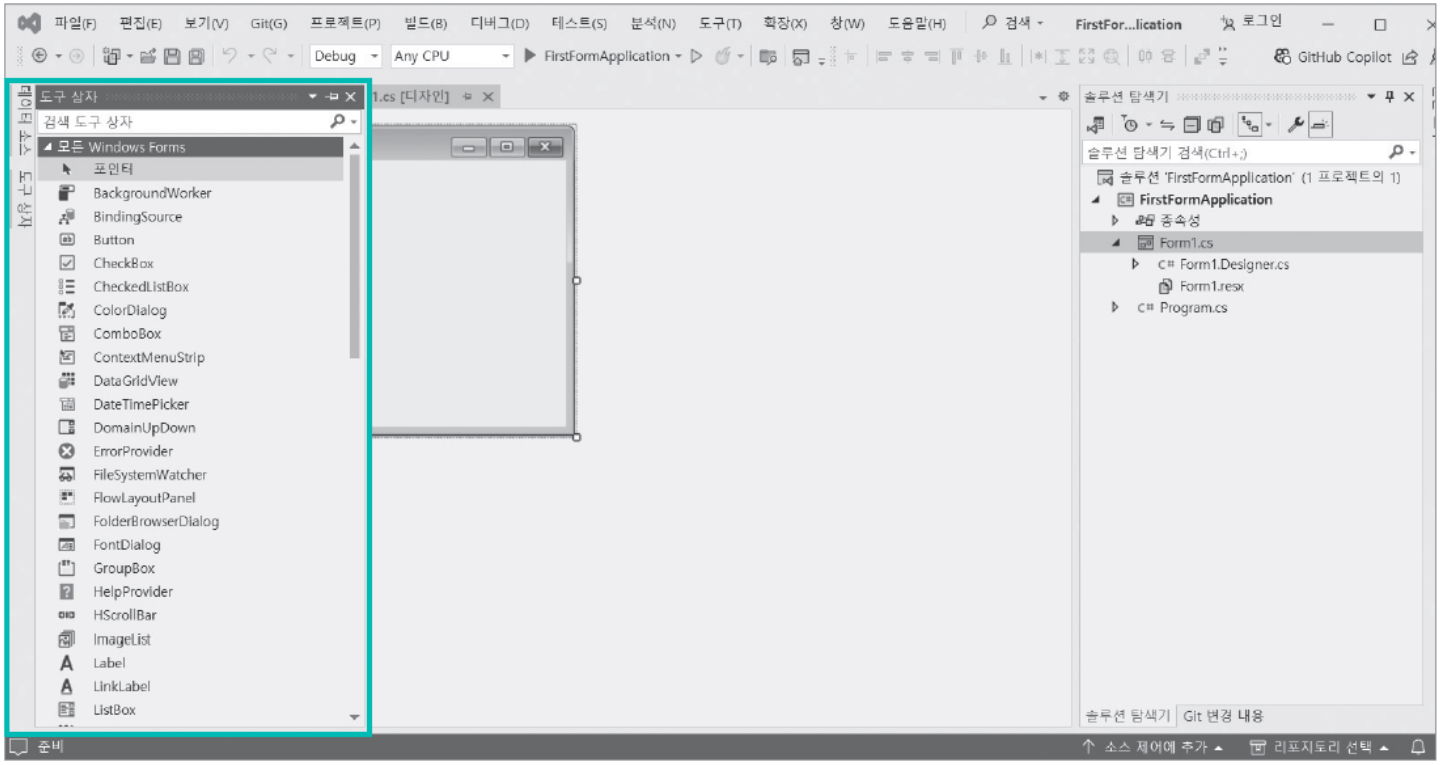


그림 5-32 도구 상자

■ 디자인 화면에서 요소 생성

- 도구 상자에서 원하는 요소를 드래그해서 폼 위에 올려주기
- Button을 폼 위로 드래그

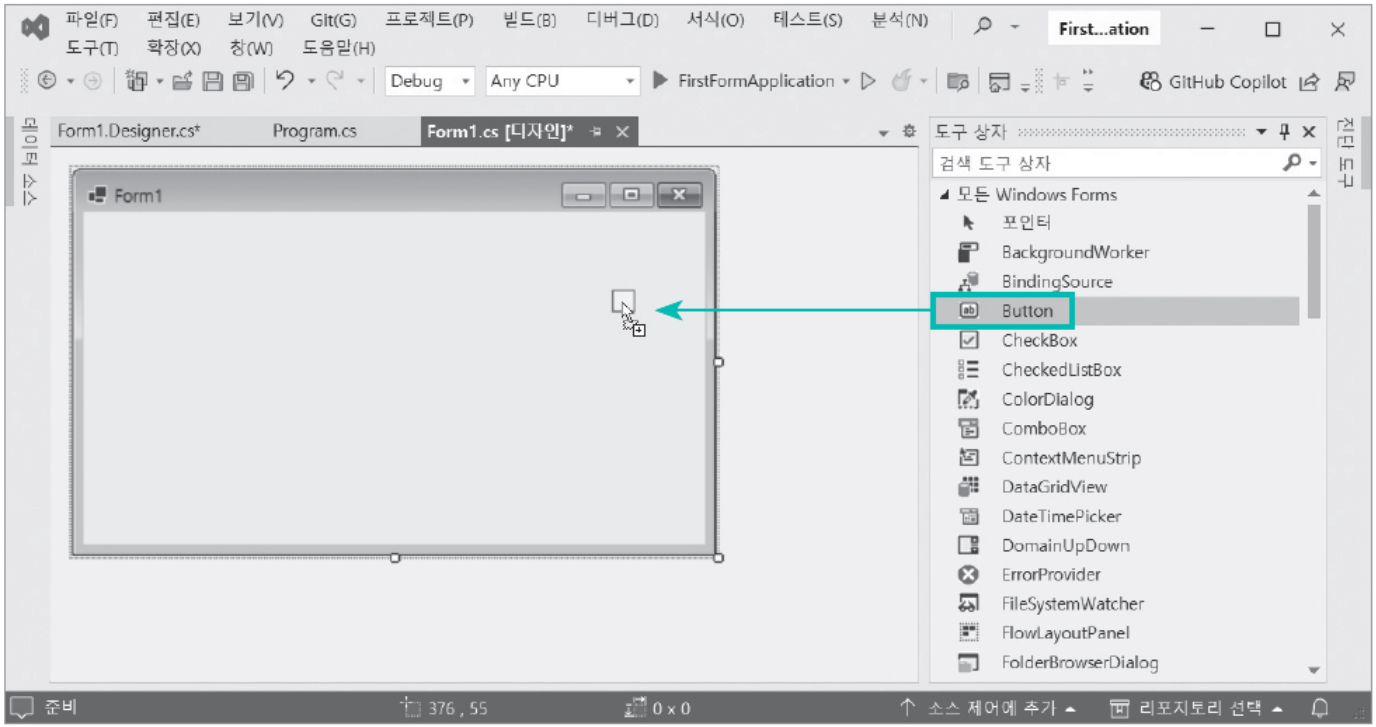


그림 5-33 도구 상자의 사용

07 윈도 폼: 윈도 폼 기본 익히기

■ 디자인 화면에서 요소 생성

- 버튼을 드래그해서 폼 위에 올리면 [그림 5-34]처럼 출력
- 비주얼 스튜디오의 속성화면(기본적으로는 화면의 오른쪽 아래에 나옵니다)을 보면 해당 버튼과 관련된 다양한 속성을 살펴볼 수 있음

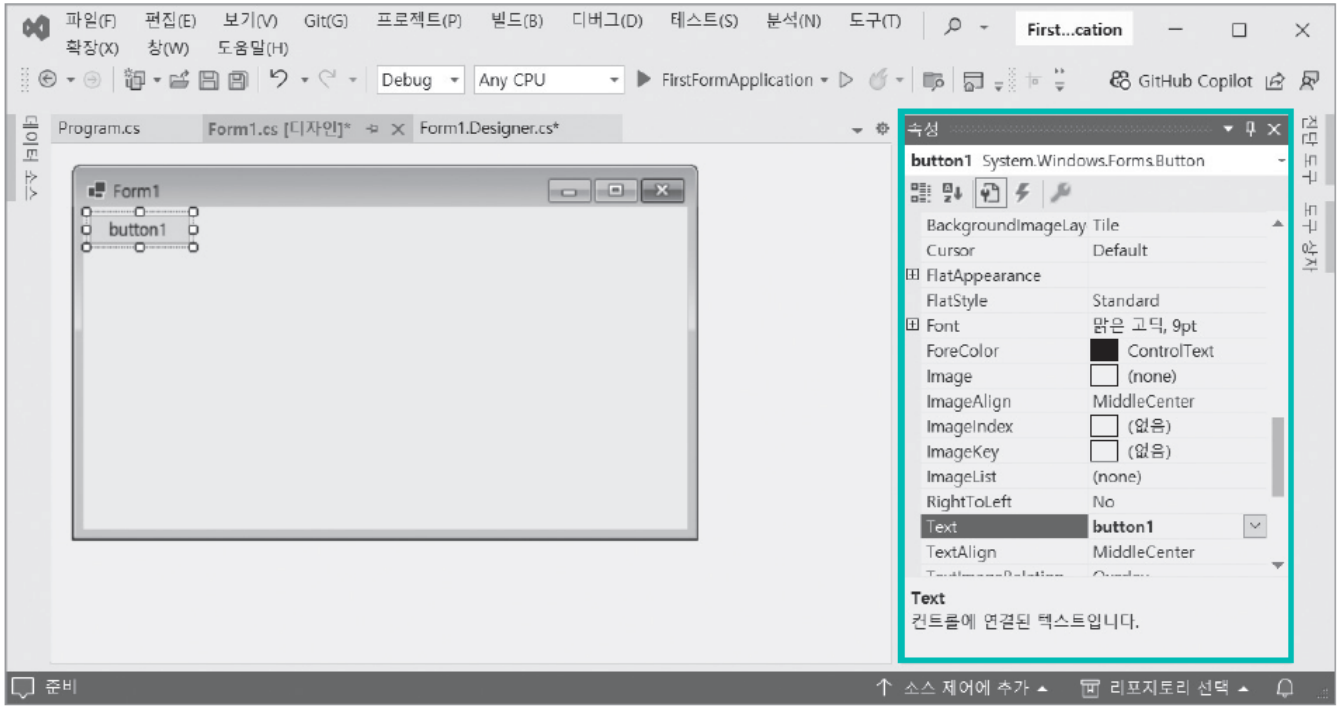


그림 5-34 속성 메뉴

■ 디자인 화면에서 요소 생성

- 속성에는 특정한 요소와 관련된 다양한 설정을 지정할 수 있음

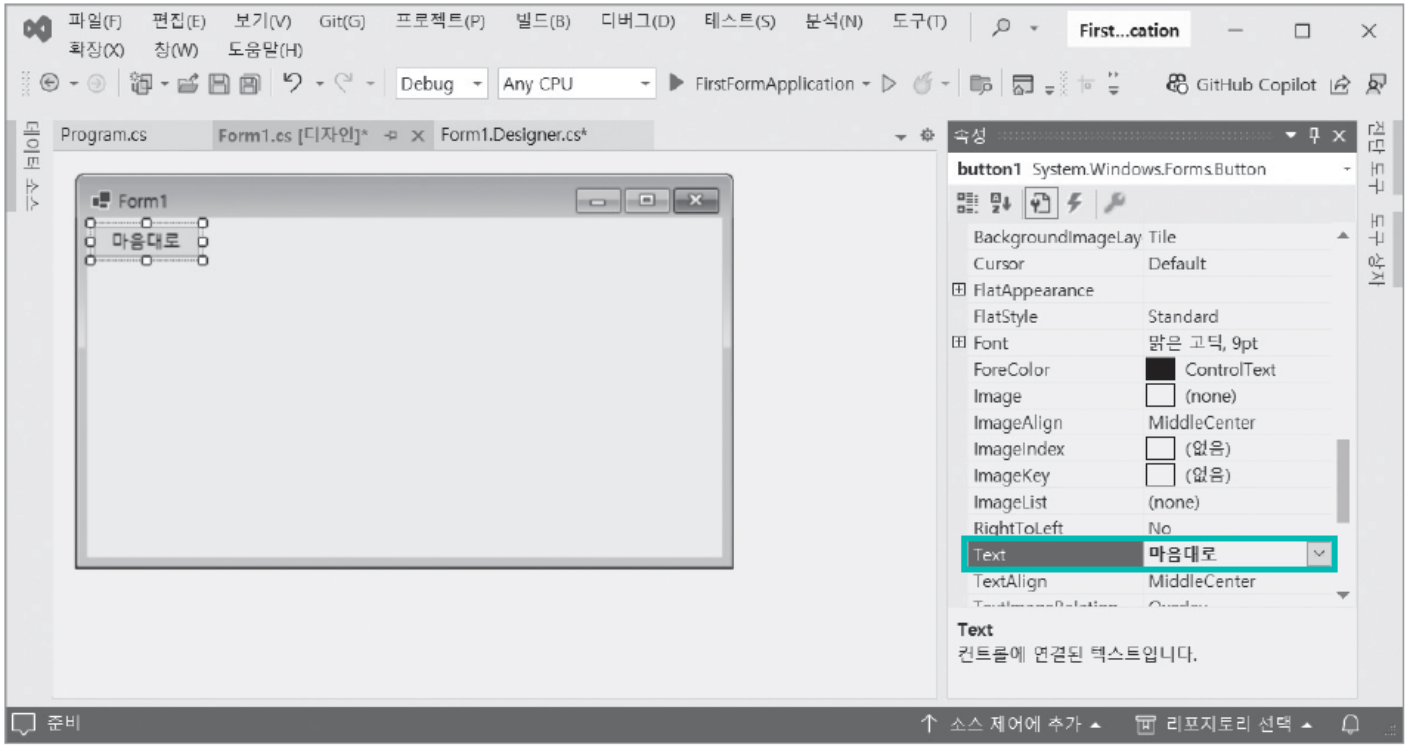


그림 5-35 속성의 지정

■ 디자인 코드

- 디자인 코드를 살펴보기 전에 버튼을 선택하고 속성 화면에서 Name이라는 이름의 속성을 찾아보자
- button1, button2, button3처럼 이름이 붙음
- 간단하게 myButton이라는 이름으로 바꾸어 보기

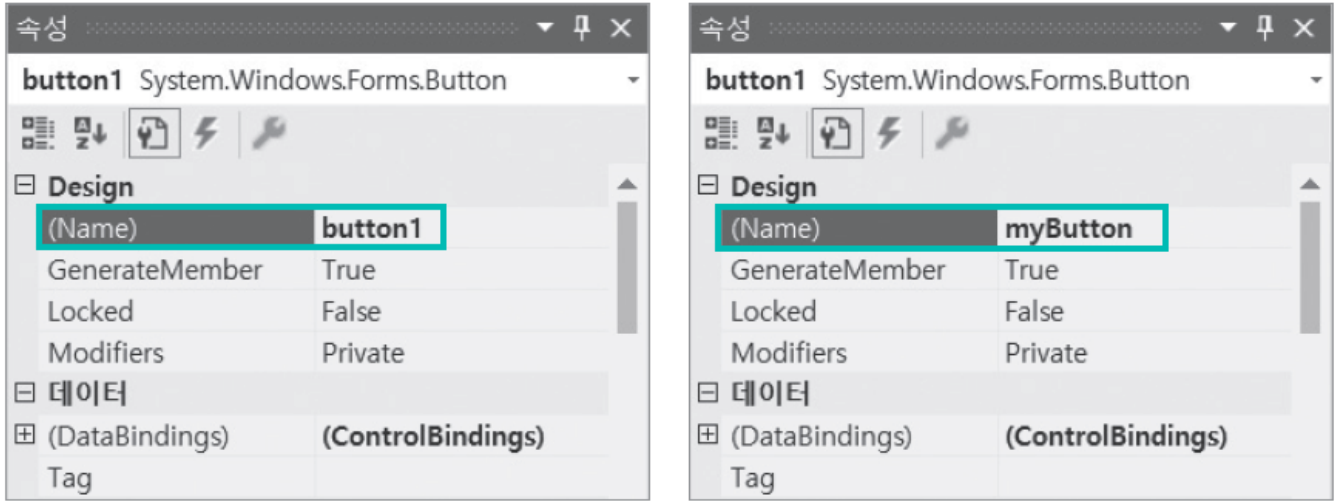


그림 5-36 Name 속성

■ 디자인 코드

- Form1.Designer.cs 파일을 다시 살펴보면 버튼과 관련된 코드들이 추가된 것을 확인할 수 있음

코드 5-29 버튼이 추가된 Form1.Designer.cs 파일

```
01 namespace FirstFormApplication
02 {
03     partial class Form1
04     {
           ...생략...
25         /// <summary>
26         ///  Required method for Designer support - do not modify
```

07 윈도 폼: 윈도 폼 기본 익히기

■ 디자인 코드

```
27          ///  the contents of this method with the code editor.
28          /// </summary>
29          private void InitializeComponent()
30           {
31              myButton = new Button();
32              SuspendLayout();
33              //
34              // myButton
35              //
36              myButton.Location = new Point(4, 5);
37              myButton.Name = "myButton";
38              myButton.Size = new Size(75, 23);
39              myButton.TabIndex = 0;
40              myButton.Text = "마음대로";
41              myButton.UseVisualStyleBackColor = true;
42              //
43              // Form1
44              //
```

07 윈도 폼: 윈도 폼 기본 익히기

■ 디자인 코드

```
45              AutoScaleDimensions = new SizeF(10F, 25F);
46              AutoScaleMode = AutoScaleMode.Font;
47              ClientSize = new Size(800, 450);
48              Controls.Add(myButton); — myButton을 화면에 추가합니다.
49              Name = "Form1";
50              Text = "Form1";
51              ResumeLayout(false);
52          }
53
54          #endregion
55
56          private Button myButton; ——— 변수 myButton이 만들어졌습니다.
57      }
58  }
```

코드에서 요소의 속성 지정

- 디자인 코드에서 myButton 변수를 생성했으므로

  Form1 클래스의 메서드 내부에서는 myButton 객체를 활용할 수 있음

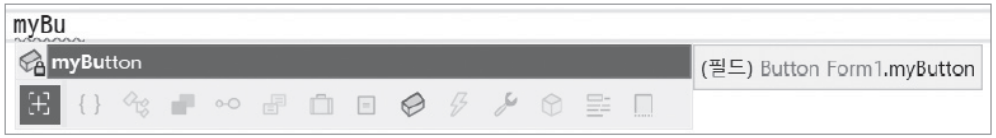


그림 5-37 myButton 객체

- Form1.cs 파일에서 myButton 객체의 속성을 원하는 대로 변경할 수 있음

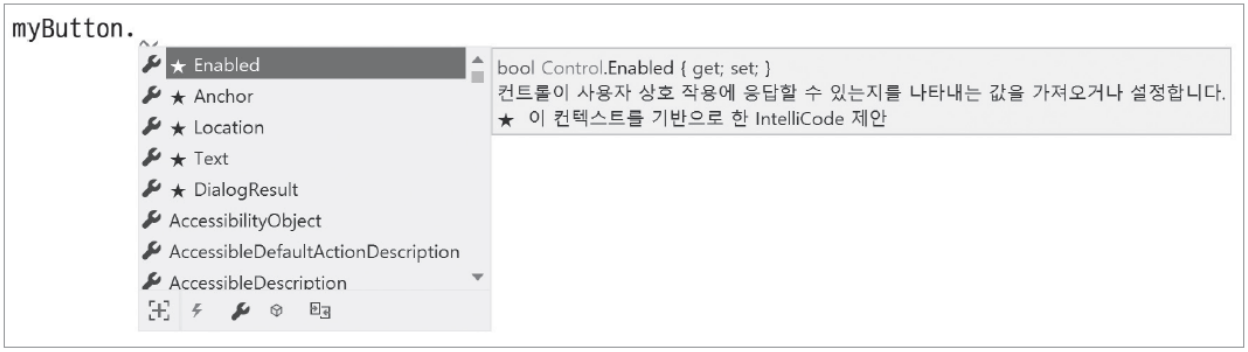


그림 5-38 myButton 객체의 속성

■ 코드에서 요소의 속성 지정

▪ 코드로도 Text 속성을 바꿔보기

그림 5-39 myButton 객체의 Text 속성

코드 5-30 Text 속성을 코드에서 변경

```
01 public partial class Form1 : Form
02 {
03     public Form1()
04     {
05         InitializeComponent();
06
07         myButton.Text = "코드에서 변경!";
08     }
09 }
```

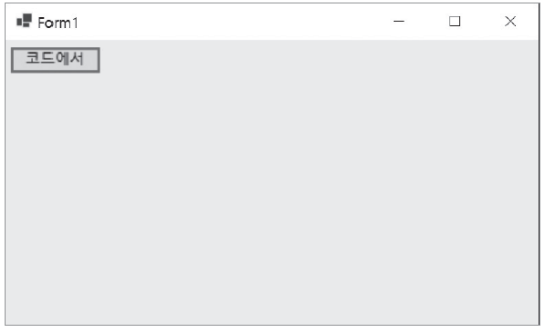


그림 5-40 코드에서 변경한 글자

■ 코드에서 요소의 속성 지정

- 속성도 코드에서 지정해줄 수 있음

코드 5-31 다양한 속성 변경

```
01 public partial class Form1 : Form
02 {
03     public Form1()
04     {
05         InitializeComponent();

06         // 출력이 깨질 경우 아래의 100과 23을 더 크게 변경해서 사용해주세요.
07         // 관련된 내용은 여기서 잠깐!을 참고해주세요.
08         myButton.Text = "코드에서 변경!";
09         myButton.Width = 100;
10         myButton.Height = 23;
11     }
12 }
```

■ 코드에서 요소 생성

▪ 코드에서 직접 버튼을 만들려면

Form1.designer.cs 파일에서 자동 생성되었던 코드를 직접 입력

코드 5-32 Button 클래스의 인스턴스 생성

```
01 public partial class Form1 : Form
02 {
03     public Form1()
04     {
05         InitializeComponent();
06
07         myButton.Text = "코드에서 변경!";
08         myButton.Width = 100;
09         myButton.Height = 23;
10         Button button = new Button();
11     }
12 }
```

07 윈도 폼: 윈도 폼 기본 익히기

■ 코드에서 요소 생성

- 생성한 버튼을 화면에 붙이려면 Form1 클래스가 가지고 있는 Controls 속성을 사용
- 상속을 아직 배우지 않았으므로 그냥 요소를 추가할 때는 Controls 속성을 사용한다고 간단하게 기억하기

코드 5-33 생성한 버튼 추가

```
01 public partial class Form1 : Form
02 {
03     public Form1()
04     {
05         InitializeComponent();
06
07         myButton.Text = "코드에서 변경!";
08         myButton.Width = 100;
09         myButton.Height = 23;
10         Button button = new Button();
11         Controls.Add(button);
12     }
13 }
```

상속입니다. 이와 관련된 내용은 7장에서 다룹니다. 일단 이때 상속을 사용했었다고 대충 기억해두면 나중에 공부할 때 도움이 될 것입니다.

■ 코드에서 요소 생성

- 새로 만든 button 객체에 Location 속성을 지정해보기

그림 5-42 Location 속성

- Point 클래스의 인스턴스를 만들어 넣어주면 됨

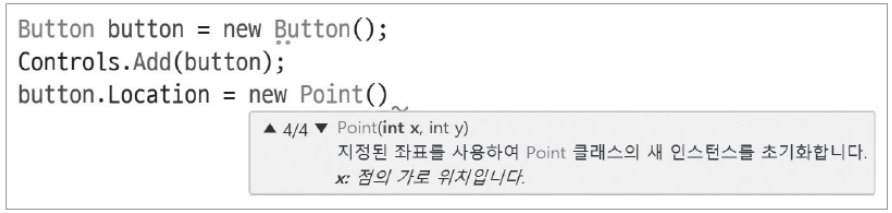


그림 5-43 Point 클래스

■ 코드에서 요소 생성

- 다음과 같은 코드를 작성하면 화면에 버튼이 추가됨

코드 5-34 생성한 버튼의 속성 지정

```
01 public partial class Form1 : Form
02 {
03     public Form1()
04     {
05         InitializeComponent();
06
07         myButton.Text = "코드에서 변경!";
08         myButton.Width = 100;
09         myButton.Height = 23;
10         Button button = new Button();
11         Controls.Add(button);
12     }
13 }
```

07 윈도 폼: 윈도 폼 기본 익히기

■ 코드에서 요소 생성

- 디자인에서 만들어도 되지만 코드를 입력하면 반복문, 조건문으로 일괄 처리 가능

코드 5-35 반복문으로 여러 개의 버튼 생성

```
01 public partial class Form1 : Form
02 {
03     public Form1()
04     {
05         InitializeComponent();
06         // 숫자를 적절하게 조절해서 사용해주세요.
07         int width = 100;
08         int height = 23;
09         int margin = 6;
10         myButton.Text = "코드에서 변경!";
11         myButton.Width = width;
12
13         for (int i = 0; i < 5; i++)
14         {
15             Button button = new Button();
16             Controls.Add(button);
17             button.Location = new Point(margin, (height + margin) * (i + 1) + margin);
18             button.Text = "동적 생성 " + i + "번째";
19             button.Width = width;
20             button.Height = height;
21         }
22     }
23 }
```

■ 코드에서 요소 생성

- 코드를 실행하면 다음과 같이 출력됨

그림 5-44 동적으로 생성된 버튼