Chapter 07 상속과 다형성

Section 01 상속과 다형성 소개(1)

- 상속과 다형성 : C#에서 반복을 줄이기 위해 사용하는 방법
  - 일반 응용 프로그램을 개발할 때 의식하면서 상속과 다형성을 활용하는 경우는 많지 않다.
- 강아지와 고양이를 나타내는 클래스 만들기

코드 7-1 Dog 클래스 /7장/AInheritance

```
01 class Dog
02 {
03     public int Age { get; set; }
04     public string Color { get; set; }
05
06     public Dog() { this.Age = 0; }
07
08     public void Eat() { Console.WriteLine("냠냠 먹습니다."); }
09     public void Sleep() { Console.WriteLine("쿨쿨 잠을 잡니다."); }
10     public void Bark() { Console.WriteLine("왈왈 짓습니다."); }
11 }
```

Section 01 상속과 다형성 소개(2)

코드 7-2 Cat 클래스 /7장/AInheritance

```
01 class Cat
02 {
03     public int Age { get; set; }
04
05     public Cat() { this.Age = 0; }
06
07     public void Eat() { Console.WriteLine("냠냠 먹습니다."); }
08     public void Sleep() { Console.WriteLine("쿨쿨 잠을 잡니다."); }
09     public void Meow() { Console.WriteLine("냥냥 웁니다."); }
10 }
```

Section 01 상속과 다형성 소개(3)

- Dog과 Cat 클래스의 인스턴스를 만들고 메서드 호출하기

코드 7-3 Dog와 Cat 클래스의 인스턴스를 만들고 메서드 실행 /7장/AInheritance

```
01 static void Main(string[] args)
02 {
03     List<Dog> Dogs = new List<Dog>() { new Dog(), new Dog(), new Dog() };
04     List<Cat> Cats = new List<Cat>() { new Cat(), new Cat(), new Cat() };
05
06     foreach (var item in Dogs)
07     {
08         item.Eat();
09         item.Sleep();
10         item.Bark();
11     }
12
13     foreach (var item in Cats)
14     {
15         item.Eat();
16         item.Sleep();
17         item.Meow();
18     }
19 }
```

실행 결과

```
냠냠 먹습니다.
쿨쿨 잠을 잡니다.
왈왈 짓습니다.
냠냠 먹습니다.
쿨쿨 잠을 잡니다.
왈왈 짓습니다.
냠냠 먹습니다.
쿨쿨 잠을 잡니다.
왈왈 짓습니다.
냠냠 먹습니다.
쿨쿨 잠을 잡니다.
냥냥 웁니다.
냠냠 먹습니다.
쿨쿨 잠을 잡니다.
냥냥 웁니다.
냠냠 먹습니다.
쿨쿨 잠을 잡니다.
냥냥 웁니다.
```

Section 02 상속(1)

- 클래스 사이에 부모 자식 관계를 정의하는 작업
- 부모 클래스 하나는 자식을 여러 개 가질 수 있음
- 현재 Dog 클래스와 Cat 클래스의 구성

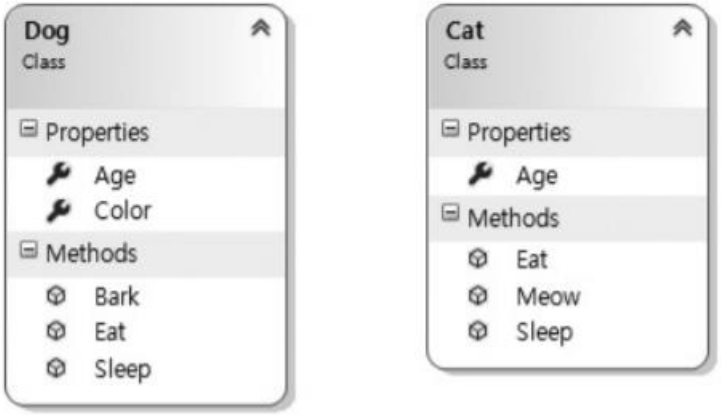


그림 7-1 분할되어 있는 클래스

- 부모 클래스(Animal)를 만드는 코드의 구성
- 중복되는 코드 줄일 수 있다.
- 양쪽 모두 추가하고 싶은 멤버가 있다면

  → Animal 클래스만 수정

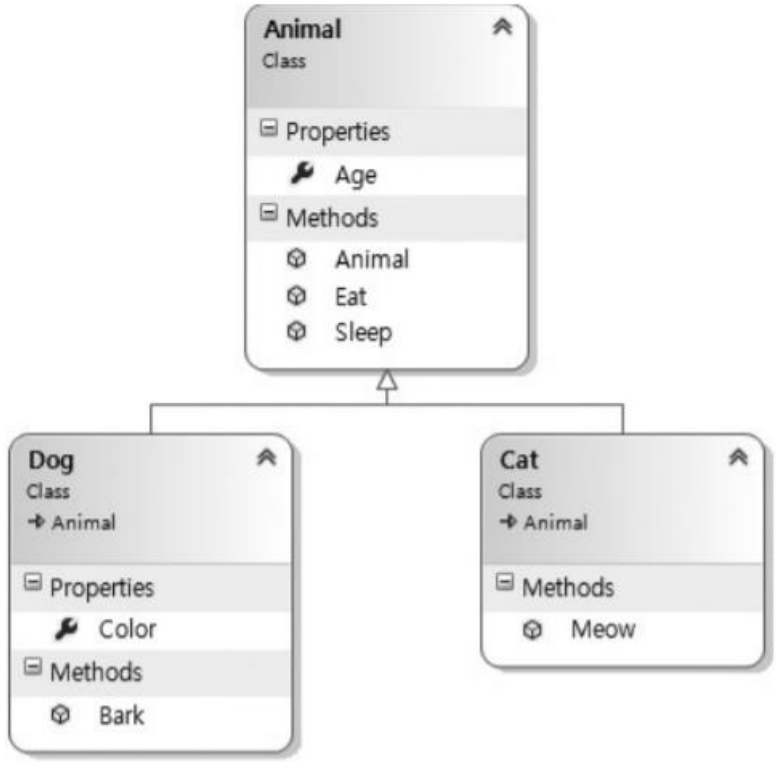


그림 7-2 상속을 사용한 구성

Section 02 상속(2)

- Animal 클래스 코드
  - 부모 클래스는 일반 클래스처럼 정의한다.

코드 7-4 Animal 클래스 /7장/BInheritance

```
01 class Animal
02 {
03     public int Age { get; set; }
04
05     public Animal() { this.Age = 0; }
06
07     public void Eat() { Console.WriteLine("냠냠 먹습니다."); }
08     public void Sleep() { Console.WriteLine("쿨쿨 잠을 잡니다."); }
09 }
```

Section 02 상속(3)

코드 7-5 Animal 클래스의 상속을 받는 Dog와 Cat 클래스 /7장/BInheritance

```
01 class Dog : Animal ——— Animal 클래스의 상속을 받습니다.
02 {
03     public string Color { get; set; }
04
05     public void Bark() { Console.WriteLine("왈왈 짖습니다."); }
06 }
07
08 class Cat : Animal ——— Animal 클래스의 상속을 받습니다.
09 {
10     public void Meow() { Console.WriteLine("냥냥 웁니다."); }
11 }
```

- 클래스의 부모 자식 관계가 형성되면 자식 클래스는 부모 클래스의 public 또는 protected 멤버에 접근 가능

Section 02 상속(4)

- 부모의 모든 멤버가 public이므로 Dog 클래스의 인스턴스를 만들면 해당 인스턴스에서 자신의 멤버는 물론 부모의 멤버에 모두 접근 가능

그림 7-3 Dog 클래스에서 접근할 수 있는 멤버 목록

■ 다른 접근 제한자

표 7-1 C#의 접근 제한자

접근 제한자	내부 클래스	외부 클래스	파생 클래스	프로젝트
public	O	O	O	O
internal	O	O	O	
protected	O		O	
private	O			
protected internal	O	사용하는 클래스가 같은 어셈블리 안에 있을 때 접근 가능	O	

NOTE(2)

■ base 키워드

- 자식 클래스에서 부모 클래스에서 정의한 멤버의 사용

코드 7-6 부모에게서 상속받은 메서드 호출 /7장/BInheritance

```
class Animal
{
    public void Eat() { Console.WriteLine("냠냠 먹습니다."); }
    public void Sleep() { Console.WriteLine("쿨쿨 잠을 잡니다."); }
}

class Dog : Animal
{
    public void Test()
    {
        Eat();
        Sleep();
    }
}
```

부모에게서 상속받은 Eat() 메서드와 Sleep() 메서드를 호출합니다.

NOTE(3)

- 이름이 겹치는 등 특수한 이유로 부모의 메서드에 접근 불가할 경우 this 키워드와 같은 형태로 base 키워드 사용(this : 자신 나타내는 키워드, base : 부모 나타내는 키워드)

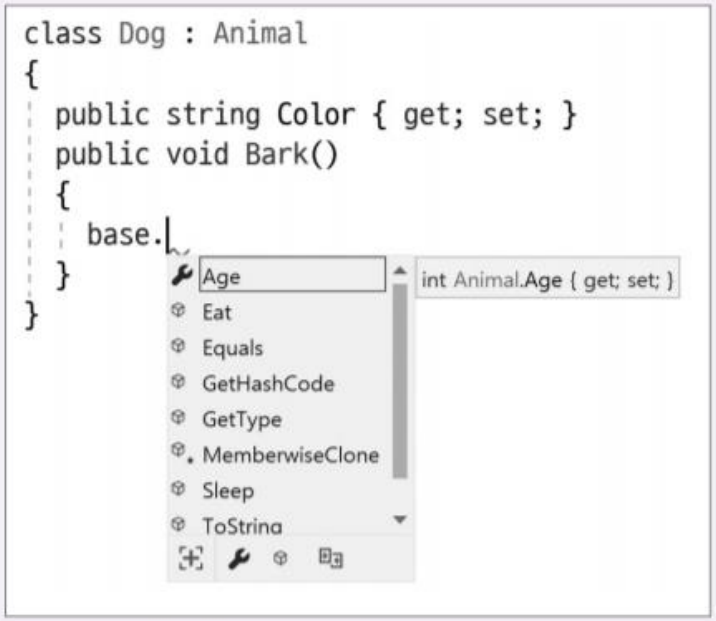


그림 7-4 부모를 나타내는 base 키워드

NOTE(4)

■ protected 접근 제한자

- private과 비슷하지만 상속한 클래스(파생 클래스)에서는 접근 가능

코드 7-7 세 가지 접근 제한자 /7장/ThreeModifiers

```
class Program
{
    class Animal
    {
        private void Private() { }
        protected void Protected() { }
        public void Public() { }

        public void TestA()
        {
            Private();
            Protected();
            Public();
        }
    }
```

자신의 클래스 내부에서는 모든 멤버를 사용할 수 있습니다.

NOTE(5)

```
class Dog : Animal
{
    public void TestB()
    {
        Protected();
        Public();
    }
}

static void Main(string[] args)
{
    Dog dog = new Dog();
    dog.Public();
}
}
```

상속받은 클래스에서는 private 접근 제한자가 붙은 멤버를 사용할 수 없습니다.

이외의 모든 장소에서는 public 접근 제한자가 붙은 멤버만 사용할 수 있습니다.

실습 문제

1. (x, y)의 한 점을 표현하는 Point 클래스와 이를 상속받아 색을 가진 점을 표현하는 ColorPoint 클래스를 만든다.

```
private static void Main(string[] args)
{
        Point p = new Point();      // Point 객체 생성
        p.set(1, 2);                // Point 클래스의 set() 호출
        p.showPoint();

        ColorPoint cp = new ColorPoint();       // ColorPoint 객체 생성
        cp.set(3, 4);                           // Point 클래스의 set() 호출
        cp.setColor("red");                     // ColorPoint 클래스의 setColor() 호출
        cp.showColorPoint();                    // 컬러와 좌표 출력
}
```

```
(1,2)
red(3,4)
```

Section 03 다형성(2)

코드 7-8) 비슷한 코드 중복

```
static void Main(string[] args)
{
    // 코드 7-8: 코드 중복
    List<Dog> Dogs = new List<Dog>();
    List<Cat> Cats = new List<Cat>();
    foreach (var item in Dogs)
    {
        item.Eat();
        item.Sleep();
        item.Bark();
    }
    foreach (var item in Cats)
    {
        item.Eat();
        item.Sleep();
        item.Meow();
    }
```

[다형성]
→ 코드 중복 해결
→ 하나의 클래스가 여러 형태로 변환가능
→ 자식클래스가 부모 클래스로 위장

Section 03 다형성(1)

- 하나의 클래스가 여러 형태로 변환될 수 있는 성질
- 자식 클래스가 부모 클래스로 위장
- 예

```
Animal dog = new Dog();
Animal cat = new Cat();
```

- 변수 dog는 외관상으로 자료형 Animal이지만 실제 내부에는 Dog가 들어있음
- 외관상으로는 Animal 객체이므로 **사용할 수 있는 멤버는 Animal(부모) 클래스의 멤버뿐**

그림 7-5 부모로 위장한 자식은 부모의 멤버만 사용 가능

Section 03 다형성(2)

코드 7-9 다형성을 사용한 코드 중복 해결

```
01 static void Main(string[] args)
02 {
03     List<Animal> Animals = new List<Animal>()
04     {
05         new Dog(), new Cat(), new Cat(), new Dog(),
06         new Dog(), new Cat(), new Dog(), new Dog()
07     };
08
09     foreach (var item in Animals)
10     {
11         item.Eat();
12         item.Sleep();
13     }
14 }
```

하나의 리스트를 사용합니다.

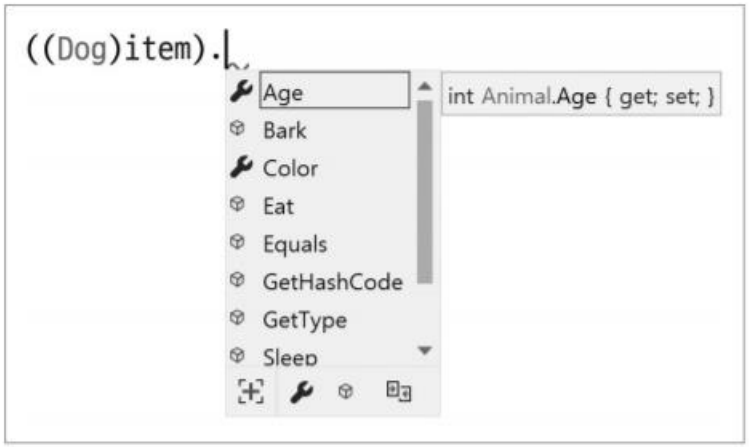
하나의 반복문을 사용합니다.

실행 결과

```
냠냠 먹습니다.
쿨쿨 잠을 잡니다.
냠냠 먹습니다.
쿨쿨 잠을 잡니다.
냠냠 먹습니다.
쿨쿨 잠을 잡니다.
냠냠 먹습니다.
쿨쿨 잠을 잡니다.
냠냠 먹습니다.
쿨쿨 잠을 잡니다.
냠냠 먹습니다.
쿨쿨 잠을 잡니다.
냠냠 먹습니다.
쿨쿨 잠을 잡니다.
냠냠 먹습니다.
쿨쿨 잠을 잡니다.
```

Section 03 다형성(3)

- 자식 클래스에 있는 메서드를 사용하기 위해, 자식 클래스로 자료형 변환
  - Animal 리스트에 있는 요소들이 Dog 클래스라면 Bark() 메서드 호출, Cat 클래스라면 Meow() 메서드 호출하게 만들기 위함!



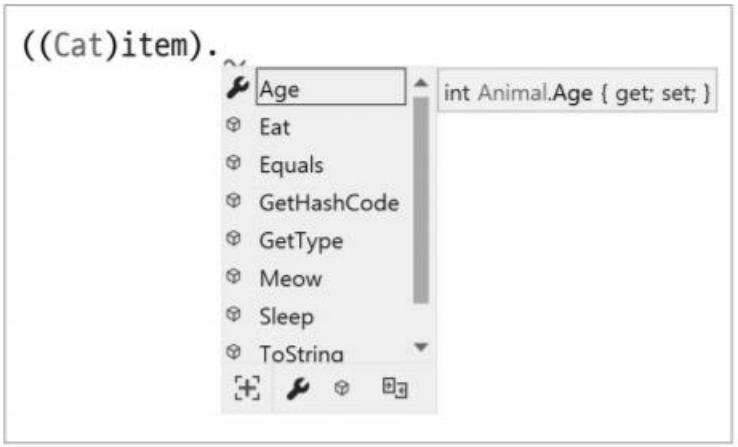


그림 7-6 위장을 해제하고 자신의 멤버를 사용

Section 03 다형성(4)

- 무작정 Cat 클래스로 변환해서 사용

코드 7-10 무작정 변환해서 메서드 호출

```
01 foreach (var item in Animals)
02 {
03     item.Eat();
04     item.Sleep();
05     ((Cat)item).Meow();
06 }
```

실행 결과

```
처리되지 않은 예외: System.InvalidCastException: 'Dog' 형식 개체를 'Cat' 형식으로 캐스팅할 수 없습니다.
…생략…
```

- 내부에 조건문 넣어 Dog 객체는 Dog 객체로 변환 Bark ( ) 메서드 호출, Cat 객체는 Cat 객체로 변환 Meow ( ) 메서드 호출

■ 최상위 객체

- C#에서 만드는 모든 객체는 Object라는 객체의 상속을 받게 됨

코드 7-11 Object 객체의 선언

```
class Object
{
    public Object();

    public virtual bool Equals(object obj);
    public static bool Equals(object objA, object objB);
    public virtual int GetHashCode();
    public Type GetType();
    protected object MemberwiseClone();
    public static bool ReferenceEquals(object objA, object objB);
    public virtual string ToString();
}
```

NOTE(2)

- 상속 관계

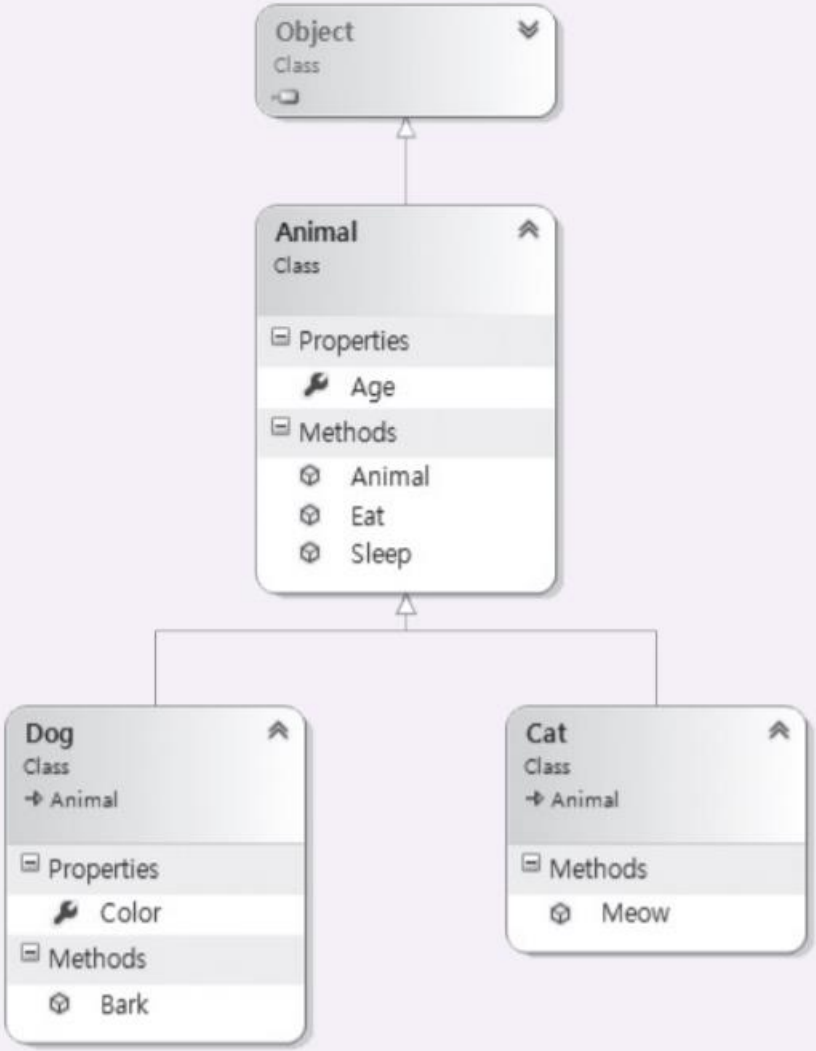


그림 7-7 최상위 Object 클래스

NOTE(3)

코드 7-12 Object 객체의 다형성 예제(1) /7장/CInheritance

```
List<Object> listOfObject = new List<Object>();
listOfObject.Add(new Dog());
listOfObject.Add(new Cat());
```

코드 7-13 Object 객체의 다형성 예제(2) /7장/CInheritance

```
List<Object> listOfObject = new List<Object>();
listOfObject.Add(new Dog());
listOfObject.Add(new Cat());
listOfObject.Add(52);
listOfObject.Add(52l);
listOfObject.Add(52.273f);
listOfObject.Add(52.273);
```

Section 04 is 키워드(1)

- 특정 객체의 클래스 확인
- is 키워드 형태 : 객체가 특정한 클래스라면 true 반환

```
객체 is 클래스
```

코드 7-14 is 키워드 /7장/DInheritance

```
01 static void Main(string[] args)
02 {
03     List<Animal> Animals = new List<Animal>() { /* 생략 */ }
04
05     foreach (var item in Animals)
06     {
07         item.Eat();
08         item.Sleep();
09
10         if (item is Dog) { } ——— 만약 변수 item이 Dog 클래스라면
11         if (item is Cat) { } ——— 만약 변수 item이 Cat 클래스라면
12     }
13 }
```

Section 04 is 키워드(3)

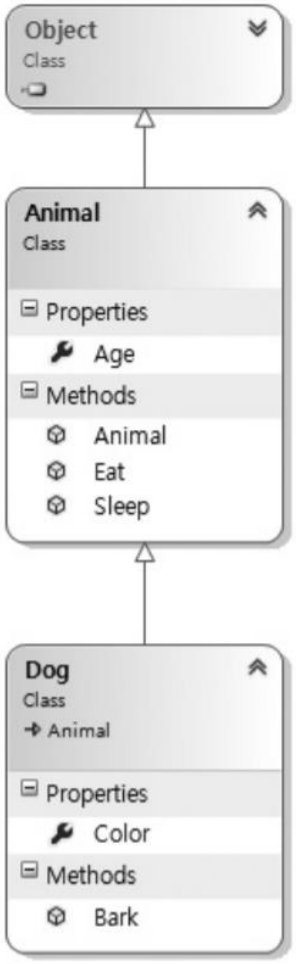


그림 7-8 Dog 클래스의 상속 관계

다형성 개념이 적용되므로,
item이 Dog 클래스의 클래스라면,
이는 Animal 클래스의 클래스이기도 하다.
최종적으로 올라가면 Object 클래스의
클래스라고도 할 수 있다.

item is Dog
item is Animal ⇨ 모두 true
item is Object

Section 05 클래스 자료형 변환(1)

■ 일반적인 자료형 변환

- 형태

```
(클래스) 변수
```

- 예

코드 7-15 일반적인 자료형 변환 /7장/DInheritance

```
01 foreach (var item in Animals)
02 {
03     item.Eat();
04     item.Sleep();
05
06     if (item is Dog) { ((Dog)item).Bark(); }
07     if (item is Cat) { ((Cat)item).Meow(); }
08 }
```

Section 05 클래스 자료형 변환(2)

■ as 키워드로 자료형 변환

- 형태

```
변수 as 클래스
```

→ 변환에 실패해도 예외가 발생하지 않음, 다만 null을 넣게 됨

- 예

코드 7-17 as 키워드를 사용하는 경우의 일반적인 형태 /7장/DInheritance

```
01 foreach (var item in Animals)
02 {
03     item.Eat();
04     item.Sleep();
05
06     var dog = item as Dog;
07     if (dog != null) { dog.Bark(); }
08
09     var cat = item as Cat;
10     if (cat != null) { cat.Meow(); }
11 }
```

Section 06 상속의 생성자(1)

- 생성자 : 인스턴스 초기화할 때 사용.
- 자식 인스턴스 생성하면, 부모의 멤버 초기화 위해 부모 생성자도 자동으로 호출

코드 7-18 상속했을 때 기본적인 생성자 호출 순서 /7장/ConstructorSequences

```
01 class Program
02 {
03     class Parent
04     {
05         public Parent()
06         {
07             Console.WriteLine("부모 생성자");
08         }
09     }
10
11     class Child : Parent
12     {
13         public Child()
14         {
15             Console.WriteLine("자식 생성자");
16         }
17     }
18
19     static void Main(string[] args)
20     {
21         Child child = new Child(); — 자식 인스턴스를 생성합니다.
22     }
23 }
```

실행 결과

```
부모 생성자
자식 생성자
```

Section 06 상속의 생성자(2)

- 부모 생성자 호출을 명시적으로 지정할 때

코드 7-19 base 키워드를 사용한 생성자 지정(1) /7장/ConstructorSequences

```
01 class Program
02 {
03     class Parent
04     {
05         public Parent() { Console.WriteLine("부모 생성자"); }
06     }
07
08     class Child : Parent
09     {
10         public Child() : base()  ——— base 키워드를 사용합니다.
11         {
12             Console.WriteLine("자식 생성자");
13         }
14     }
15
16     static void Main(string[] args)
17     {
18         Child child = new Child();
19     }
20 }
```

Section 06 상속의 생성자(3)

- 매개변수가 있는 메서드를 호출하고 싶을 때

코드 7-20 base 키워드를 사용한 생성자 지정(2) /7장/ConstructorSequences

```
class Program
{
    class Parent
    {
        public Parent() { Console.WriteLine("Parent()"); }
        public Parent(int param) { Console.WriteLine("Parent(int param)"); }
        public Parent(string param) { Console.WriteLine("Parent(string param)"); }
    }

    class Child : Parent
    {
        public Child() : base(10)
        {
            Console.WriteLine("Child() : base(10)");
        }

        public Child(string input) : base(input)
        {
            Console.WriteLine("Child(string input) : base(input)");
        }
    }

    static void Main(string[] args)
    {
        Child childA = new Child();
        Child childB = new Child("string");
    }
}
```

Parent(int param) 부모 생성자를 호출합니다.

Parent(string param) 부모 생성자를 호출합니다.

실행 결과

```
Parent(int param)
Child() : base(10)
Parent(string param)
Child(string input) : base(input)
```

■ 클래스 변수 상속

- 클래스 변수는 상속되어도 공유

코드 7-21 클래스 변수 상속 /7장/ClassVariableOnInheritance

```
class Program
{
    class Parent
    {
        public static int counter = 0;

        public void CountParent()
        {
            Parent.counter++;
        }
    }

    class Child : Parent
    {
        public void CountChild()
        {
            Child.counter++;
        }
    }

    static void Main(string[] args)
    {
        Parent parent = new Parent();
        Child child = new Child();

        parent.CountParent();
        child.CountChild();

        Console.WriteLine(Parent.counter);
        Console.WriteLine(Child.counter);
    }
}
```

클래스 변수 counter를 선언합니다.

Parent 클래스의 counter 변수를 증가시킵니다.

Child 클래스의 counter 변수를 증가시킵니다.

실행 결과

```
2
2
```

Section 07 섀도잉과 하이딩(1) – 그냥 참고만!

■ 섀도잉 : 특정한 영역에서 이름이 겹쳐 다른 변수 가리는 것

- 예

코드 7-22 섀도잉 /7장/ShadowAndHide

```
01 class Program
02 {
03     public static int number = 10;
04
05     static void Main(string[] args)
06     {
07         int number = 20;
08         Console.WriteLine(number);
09     }
10 }
```

static 메서드 내부에서 사용할 수 있게 static 변수로 만들었습니다.

변수의 이름이 겹칠 때 자신과 가장 가까운 변수를 사용

실행 결과

```
20
```

Section 07 섀도잉과 하이딩(2) – 그냥 참고만!

- **하이딩 : 부모 클래스와 자식 클래스에 동일 이름으로 멤버 만들 때 발생**
  - 변수 하이딩 예

코드 7-23 변수 하이딩 /7장/ShadowAndHide

```
01 class Program
02 {
03     class Parent
04     {
05         public int variable = 273;
06     }
07
08     class Child : Parent
09     {
10         public string variable = "shadowing";
11     }
12
13     static void Main(string[] args)
14     {
15         Child child = new Child();
16         Console.WriteLine(child.variable);
17     }
18 }
```

→ 자신과 가장 가까운 변수를 사용

실행 결과

```
shadowing
```

Section 07 섀도잉과 하이딩(3) – 그냥 참고만!

- 부모에 있는 int 자료형의 변수 사용할 때
  - 부모 자료형으로 변환하고 사용

코드 7-24 숨겨진 멤버를 찾는 방법 /7장/ShadowAndHide

```
01 static void Main(string[] args)
02 {
03     Child child = new Child();
04     Console.WriteLine(((Parent) child).variable);
05 }
```

Section 07 섀도잉과 하이딩(4) – 그냥 참고만!

- 메서드 하이딩 예

코드 7-25 메서드 하이딩 /7장/ShadowAndHide

```
class Program
{
    class Parent
    {
        public void Method()
        {
            Console.WriteLine("부모의 메서드");
        }
    }

    class Child : Parent
    {
        public void Method()
        {
            Console.WriteLine("자식의 메서드");
        }
    }

    static void Main(string[] args)
    {
        Child child = new Child();
        child.Method();
        ((Parent)child).Method();
    }
}
```

Section 07 섀도잉과 하이딩(5) – 그냥 참고만!

- 실행은 정상적이나 개발 환경에 경고 메시지 뜸

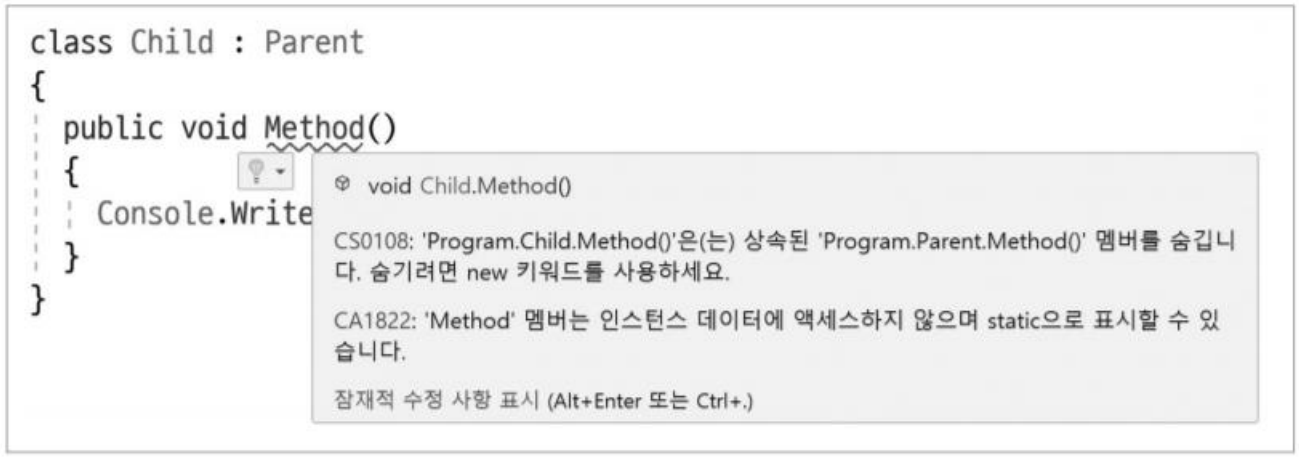


그림 7-9 경고 메시지

- 메서드는 변수와 다르게 충돌이 발생할 때 하이딩할지 오버라이딩할지 결정 가능

```
public new void Method()
{
    Console.WriteLine("자식의 메서드");
}
```
하이딩합니다.

```
public override void Method()
{
    Console.WriteLine("자식의 메서드");
}
```
오버라이딩합니다.

Section 08 하이딩과 오버라이딩(1)

- **오버라이딩Overriding : 부모 클래스의 메서드를 자식 클래스에서 재구현**
  - 하이딩의 형태로 메서드 작성 후 앞에 virtual이라는 키워드 붙임
  - 하이딩은 멤버 전체(변수, 메서드 등)에서 발생
  - 오버라이딩은 메서드 관련만 발생

Section 08 하이딩과 오버라이딩(2)

■ new 메서드

- 하이딩한다는 표시를 위해 메서드 이름 앞에 new 키워드 붙임
- 예

코드 7-26 new 메서드를 사용한 하이딩 /7장/NewMethods

```
class Program
{
    class Parent
    {
        public int variable = 273;
        public void Method()
        {
            Console.WriteLine("부모의 메서드");
```

Section 08 하이딩과 오버라이딩(3)

```
    }
}

class Child : Parent
{
    public new string variable = "hiding";
    public new void Method()
    {
        Console.WriteLine("자식의 메서드");
    }
}

static void Main(string[] args)
{
    Child child = new Child();
    child.Method();
    ((Parent)child).Method();
}
}
```

new 키워드를 사용해 변수를 하이딩하겠다고 선언합니다.

new 키워드를 사용해 메서드를 하이딩하겠다고 선언합니다.

Section 08 하이딩과 오버라이딩(4)

■ virtual과 override 메서드

- 예

코드 7-27 virtual과 override 메서드를 사용한 오버라이딩 /7장/OverrideMethods

```
class Program
{
   class Parent
   {
      public virtual void Method()
      {
         Console.WriteLine("부모의 메서드");
```

부모의 메서드에 virtual 키워드를 적용합니다.

Section 08 하이딩과 오버라이딩(5)

```
    }
}

class Child : Parent
{                                    자식의 메서드에 override 키워드를 적용합니다.
    public override void Method()
    {
        Console.WriteLine("자식의 메서드");
    }
}

static void Main(string[] args)
{
    Child child = new Child();
    child.Method();
    ((Parent)child).Method();
}
}
```

실행 결과

```
자식의 메서드
자식의 메서드
```

- 오버라이딩하면 클래스형을 어떻게 변환해도 자식에서 다시 정의한 메서드 호출

Section 08 하이딩과 오버라이딩(6)

■ 활용

- 하이딩

→ 부모 메서드만 실행

코드 7-28 하이딩 /7장/UsageOfHidding

```
class Program
{
    class Animal
    {
        public void Eat()
        {
            Console.WriteLine("냠냠 먹습니다.");
        }
    }

    class Dog : Animal
    {
        public void Eat()
        {
            Console.WriteLine("강아지 사료를 먹습니다.");
        }
    }

    class Cat : Animal
    {
        public void Eat()
        {
```

같은 이름을 재사용했습니다.

Section 08 하이딩과 오버라이딩(7)

```
            Console.WriteLine("고양이 사료를 먹습니다.");
        }
    }

    static void Main(string[] args)
    {
        List<Animal> Animals = new List<Animal>()
        {
            new Dog(), new Cat(), new Cat(), new Dog(),
            new Dog(), new Cat(), new Dog(), new Dog()
        };

        foreach (var item in Animals)
        {
            item.Eat();
        }
    }
}
```

Eat 메서드를 호출합니다.

실행 결과

```
냠냠 먹습니다.
냠냠 먹습니다.
냠냠 먹습니다.
냠냠 먹습니다.
냠냠 먹습니다.
냠냠 먹습니다.
냠냠 먹습니다.
냠냠 먹습니다.
```

Section 08 하이딩과 오버라이딩(8)

- 오버라이딩

코드 7-29 오버라이딩 /7장/UsageOfOverriding

```
class Animal
{
    public virtual void Eat()
    {
        Console.WriteLine("냠냠 먹습니다.");
    }
}

class Dog : Animal
{
    public override void Eat()
    {
        Console.WriteLine("강아지 사료를 먹습니다.");
    }
}

class Cat : Animal
{
    public override void Eat()
    {
        Console.WriteLine("고양이 사료를 먹습니다.");
    }
}
```

오버라이딩합니다.

실행 결과

```
강아지 사료를 먹습니다.
고양이 사료를 먹습니다.
고양이 사료를 먹습니다.
강아지 사료를 먹습니다.
강아지 사료를 먹습니다.
고양이 사료를 먹습니다.
강아지 사료를 먹습니다.
강아지 사료를 먹습니다.
```

Section 08 하이딩과 오버라이딩(9)

- new 키워드를 사용하는 하이딩

코드 7-30 new 키워드를 사용하는 경우

```
class Animal
{
    public virtual void Eat()
    {
        Console.WriteLine("냠냠 먹습니다.");
    }
}

class Dog : Animal
{
    public new void Eat()
    {
        Console.WriteLine("강아지 사료를 먹습니다.");
    }
}

class Cat : Animal
{
    public override void Eat()
    {
        Console.WriteLine("고양이 사료를 먹습니다.");
    }
}
```

부모 메서드 호출

하이딩으로 변경합니다.

실행 결과

```
냠냠 먹습니다.
고양이 사료를 먹습니다.
고양이 사료를 먹습니다.
냠냠 먹습니다.
냠냠 먹습니다.
고양이 사료를 먹습니다.
냠냠 먹습니다.
냠냠 먹습니다.
```

Section 09 상속과 오버라이딩 제한(1)

■ sealed 키워드 : 클래스 적용(상속 제한), 메서드 적용(오버라이딩 제한)

- 상속 제한 오류 예

코드 7-31 sealed 클래스 오류 /7장/Sealed

```
class Program
{
    sealed class Parent
    {
        public void Test() { }
    }

    class Child : Parent
    {
        public void Test() { }
    }

    static void Main(string[] args)
    {
        Parent parent = new Parent();
        Child child = new Child();

        parent.Test();
        child.Test();
    }
}
```

sealed 클래스로 선언했습니다.

여기서 오류가 발생합니다.

그림 7-10 sealed 클래스 오류

Section 09 상속과 오버라이딩 제한(2)

- 메서드 오버라이딩 제한 오류 예

코드 7-32 sealed 메서드 오류

```
01 class Parent
02 {
03     public virtual void Test() { }
04 }
05
06 class Child : Parent
07 {
08     sealed public override void Test() { }
09 }
10
11 class GrandChild : Child
12 {
13     public override void Test() { }
14 }
```

sealed 메서드로 변경했습니다.

여기서 오류가 발생합니다.

```
class GrandChild : Child
{
    public override void Test() { }
}
```

void GrandChild.Test()

CS0239: 'Program.GrandChild.Test()': 상속된 'Program.Child.Test()' 멤버는 봉인되어 있으므로 재정의할 수 없습니다.

그림 7-11 sealed 메서드 오류

Section 09 상속과 오버라이딩 제한(3)

■ abstract 키워드 : 무조건 상속, 또는 메서드는 반드시 오버라이딩

- 상속 제한 오류 예

코드 7-33 abstract 클래스 오류 /7장/Abstract

```
01 class Program
02 {
03     abstract class Parent
04     {
05         public void Test() { }
06     }
07
08     class Child : Parent
09     {
10         public void Test() { }
11     }
12
13     static void Main(string[] args)
14     {
15         Parent parent = new Parent();
16         Child child = new Child();
17
18         parent.Test();
19         child.Test();
20     }
21 }
```

abstract 클래스로 선언했습니다.

반드시 상속해서 쓰라.
해당 클래스 자체는 인스턴스를 만들 수 없다.

여기서 오류가 발생합니다.

그림 7-12 abstract 클래스 오류

Section 09 상속과 오버라이딩 제한(4)

- 메서드 오버라이딩 제한 오류 예

코드 7-34 abstract 메서드 /7장/Abstract

```
01 abstract class Parent
02 {
03     public abstract void Test();
04 }
05
06 class Child : Parent
07 {
08
09 }
```

abstract 메서드를 선언하려면 반드시 abstract 클래스가 되어야 합니다.

abstract 메서드로 선언했습니다.

여기에서 오류가 발생합니다.

그림 7-13 abstract 메서드 오류

Section 09 상속과 오버라이딩 제한(5)

- abstract 메서드와 관련된 오류 해결

코드 7-35 abstract 메서드와 관련된 오류 해결 /7장/Abstract

```
01 abstract class Parent
02 {
03     public abstract void Test();
04 }
05
06 class Child : Parent
07 {
08     public override void Test() { }
09 }
```

override 키워드를 사용해 오버라이딩해야 합니다.

■ 윈도 폼에서의 상속과 다형성

코드 7-36 Form1 클래스의 기본적인 형태

```
01 public partial class Form1 : Form ——— Form 클래스의 상속을 받습니다.
02 {
03     public Form1()
04     {
05         InitializeComponent();
06     }
07 }
```

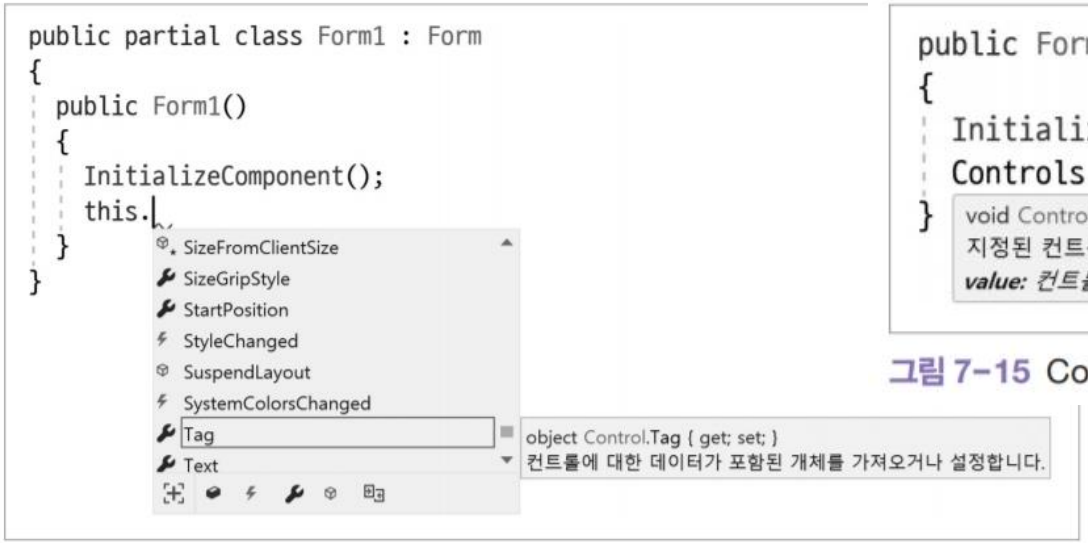


그림 7-14 Form 클래스로부터 상속받은 것들

그림 7-15 Controls.Add() 메서드의 매개변수

Section 10 윈도 폼: 윈도 폼에서 상속과 다형성 활용하기(2)

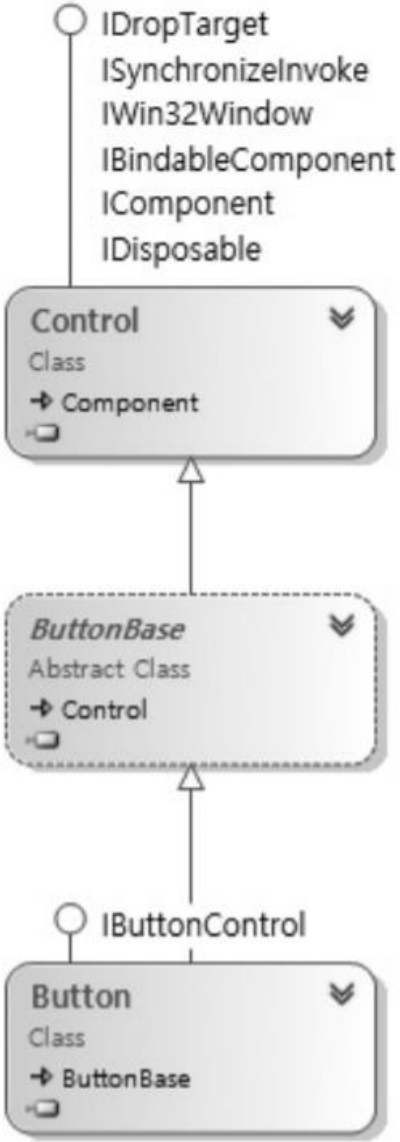


그림 7-16 Button 클래스의 상속 관계

그림 7-17 정의로 이동

■ 메시지 상자

▪ 새로운 화면을 띄우는 방법

- 화면 디자인

그림 7-18 화면 디자인

- 버튼 더블 클릭 또는 속성에서 Click 이벤트 오른쪽의 공간 더블 클릭해서 이벤트 메서드 연결

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.
}
```

Show
Equals
ReferenceEquals
Show

그림 7-19 MessageBox 클래스의 메서드

코드 7-37 MessageBox 클래스의 Show() 메서드

```
01 private void button1_Click(object sender, EventArgs e)
02 {
03     MessageBox.Show("내용");
04     MessageBox.Show("내용", "제목");
05     MessageBox.Show("내용", "제목", MessageBoxButtons.RetryCancel);
06 }
```

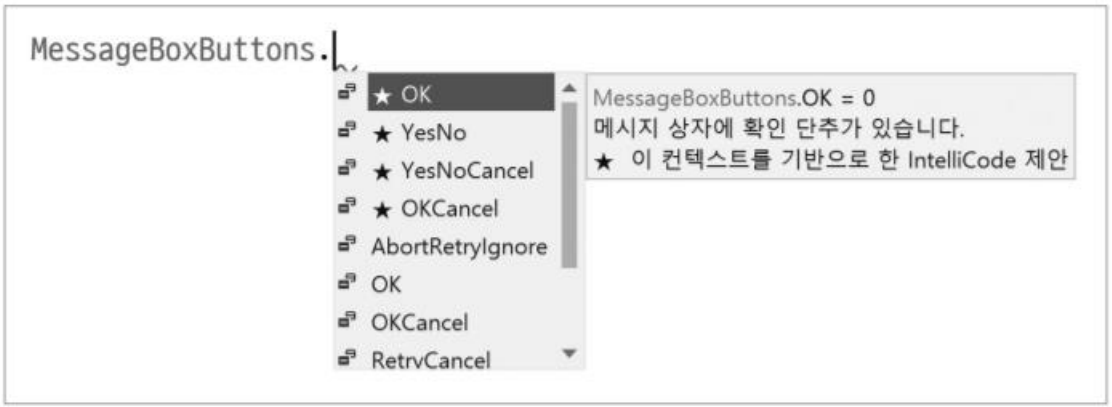


그림 7-20 MessageBoxButtons 열거자

Section 10 윈도 폼: 윈도 폼에서 상속과 다형성 활용하기(5)

- 프로젝트를 실행 결과

그림 7-21 메시지 상자

- 메시지 상자의 버튼 누르면 Show( ) 메서드의 반환으로 어떤 버튼을 눌렀는지 알 수 있음

코드 7-38 메시지 상자 활용

```
01 private void button1_Click(object sender, EventArgs e)
02 {
03     MessageBox.Show("내용");
04     MessageBox.Show("내용", "제목");
05
06     DialogResult result;
07     do
08     {
09         result = MessageBox.Show("내용", "제목", MessageBoxButtons.RetryCancel);
10     } while (result == DialogResult.Retry);
11 }
```

적어도 한 번은 실행하게 만들고자 do while 반복문을 사용했습니다.

NOTE

■ enum 자료형(열거자) : 숫자에 특정한 이름을 붙여주는 방법

- 자료형 비교를 통해 숫자를 조금 더 안전하게 입력
- 예

코드 7-39 열거자

```
enum OrderState { Ordered, Paymented, Prepared, Sended };

class Program
{
    static OrderState OrderCheck(int orderId)
    {
        return OrderState.Ordered;
    }

    static void Main(string[] args)
    {
        if (OrderCheck(12345) == OrderState.Ordered)
        {
            Console.WriteLine("주문 완료되었습니다.");
        }
    }
}
```

열거자는 클래스, 인터페이스 등과 같은 위치에 생성합니다.

메서드의 매개변수 또는 반환형에 자주 활용됩니다.

■ 간단한 형태의 폼 추가

- Form1 클래스 디자인

그림 7-22 화면 구성

Section 10 윈도 폼: 윈도 폼에서 상속과 다형성 활용하기(7)

- Form1 클래스 상속

코드 7-40 Form 클래스 상속

```
01 public partial class Form1 : Form
02 {
03     class CustomForm : Form ———— Form 클래스를 상속받습니다.
04     {
05
06     }
07
08     public Form1()
09     {
10         InitializeComponent();
11     }
12 }
```

Section 10 윈도 폼: 윈도 폼에서 상속과 다형성 활용하기(8)

- 폼의 Text 속성 지정해서 폼에 제목 지정

코드 7-41 새로 생성한 CustomForm 클래스의 모양 지정

```
01 class CustomForm : Form
02 {
03     public CustomForm()
04     {
05         Text = "제목 글자";
06     }
07 }
```

Section 10 윈도 폼: 윈도 폼에서 상속과 다형성 활용하기(9)

- 모달리스 : 새로운 화면이 열려도 기존에 있던 화면 조작 가능한 형태
  - 버튼에 이벤트 연결, CustomForm 클래스의 인스턴스 생성 후 Show ( ) 메서드 호출

코드 7-42 Show() 메서드

```
01 public partial class Form1 : Form
02 {
03     class CustomForm : Form
04     {
05
06     }
07
08     public Form1()
09     {
10         InitializeComponent();
11     }
12
13     private void button1_Click(object sender, EventArgs e)
14     {
15         CustomForm form = new CustomForm();
16         form.Show();
17     }
18 }
```

Form 클래스를 상속받습니다.

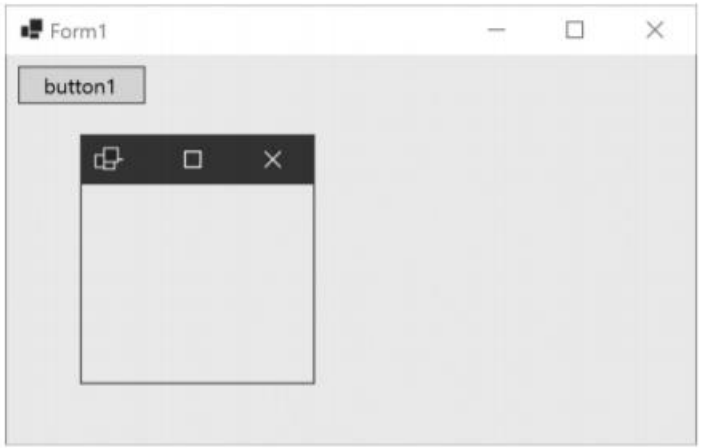


그림 7-23 Show() 메서드의 동작

MDI

- 하나의 화면 내부에 여러 개의 다른 화면을 띄우는 것

그림 7-24 MDI 프로그램

NOTE(2)

- 부모 폼에 IsMdiContainer 속성, 자식 폼에 MdiParent 속성 지정
- 예

코드 7-43 MDI 사용

```
public partial class Form1 : Form
{
    class CustomForm : Form { }

    public Form1()
    {
        InitializeComponent();
        IsMdiContainer = true;
    }

    private void button1_Click(object sender, EventArgs e)
    {
        CustomForm form = new CustomForm();
        form.MdiParent = this;
        form.Show();
    }
}
```

자기 자신을 Mdi 컨테이너로 만듭니다. 코드가 아니라 디자인 화면의 속성에서 지정해도 됩니다. 이 책에서는 간단하게 코드로 지정하겠습니다.

MdiParent 속성으로 부모를 자기 자신(Form1 객체)으로 지정해줍니다. 이를 다른 입력 양식으로 지정하면 A라는 폼에서 버튼을 클릭해 B라는 입력 양식에도 띄울 수 있답니다.

NOTE(3)

그림 7-25 MDI 실행 결과

Section 10 윈도 폼: 윈도 폼에서 상속과 다형성 활용하기(10)

- 모달 : 모달리스와 반대로 새로운 화면을 띄웠을 때 기존 화면 조작불가능
  - Show ( ) 메서드를 ShowDialog ( ) 메서드로 전환
  - 예

코드 7-44 ShowDialog() 메서드

```
01 private void button1_Click(object sender, EventArgs e)
02 {
03     CustomForm form = new CustomForm();
04     form.ShowDialog(); —— 모달 대화상자를 생성합니다.
05 }
```

■ 일반적인 형태의 폼 추가

- 디자인 화면과 함께 폼 생성
  - 프로젝트에서 마우스 오른쪽 버튼 클릭 [추가] - [Windows Form] 선택

    (또는 새 항목 선택 후 다음 화면에서 [Windows Form] 선택

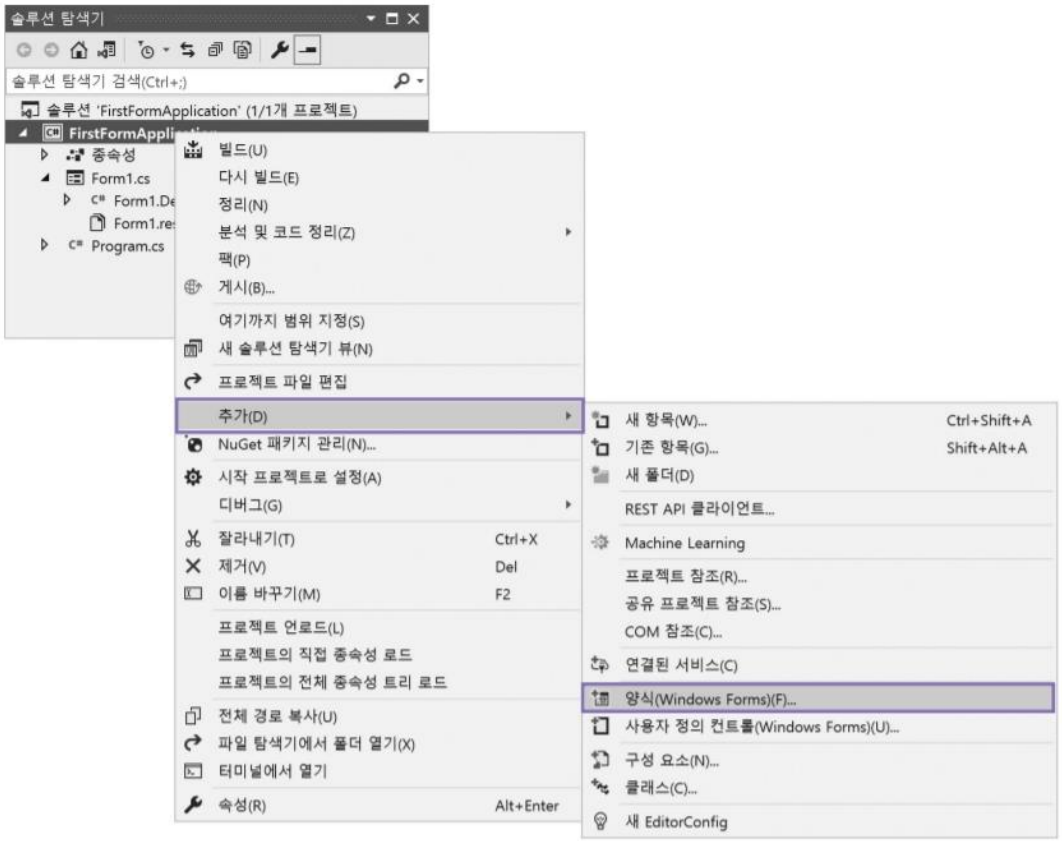


그림 7-26 [추가] – [Windows Form]으로 새로운 폼 생성

Section 10 윈도 폼: 윈도 폼에서 상속과 다형성 활용하기(12)

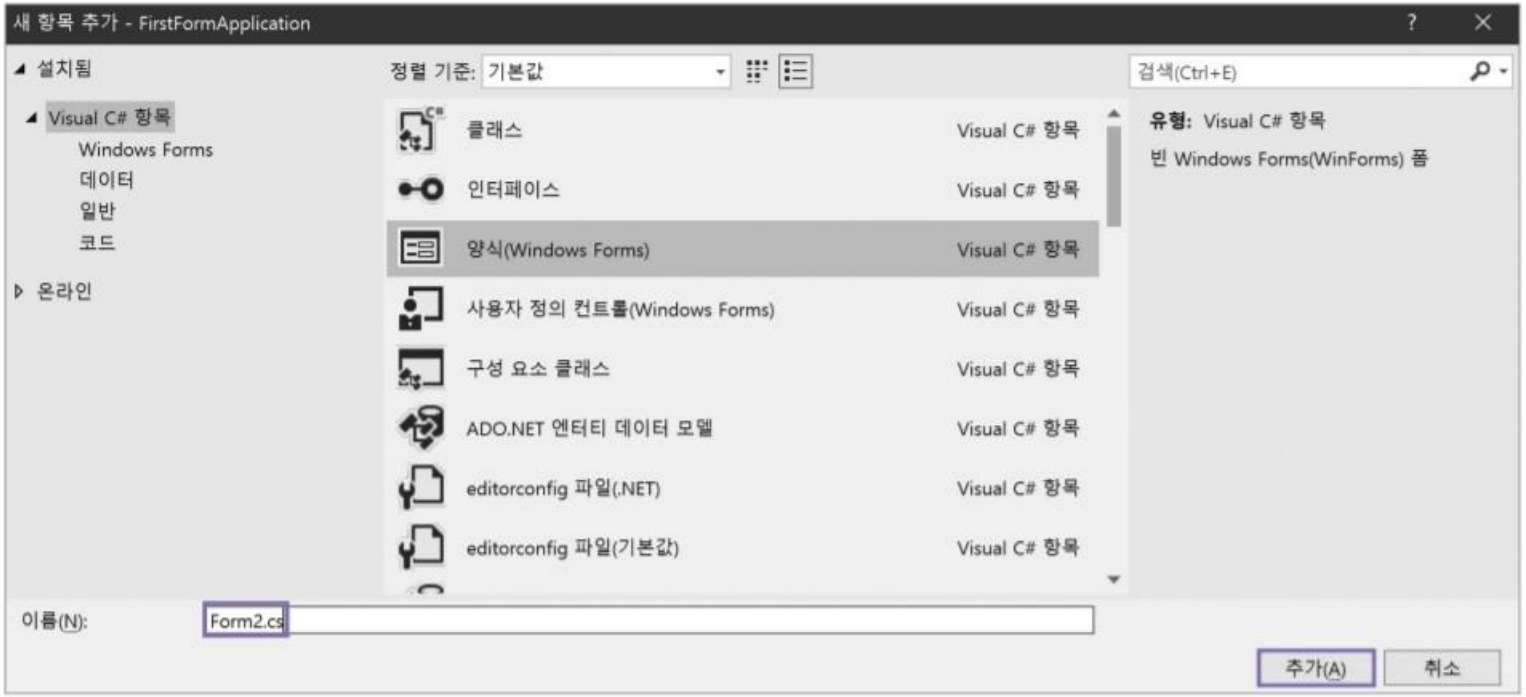


그림 7-27 새 항목 추가 대화상자

Section 10 윈도 폼: 윈도 폼에서 상속과 다형성 활용하기(13)

- 디자인 화면에서 드래그해서 요소를 화면에 올려 놓기

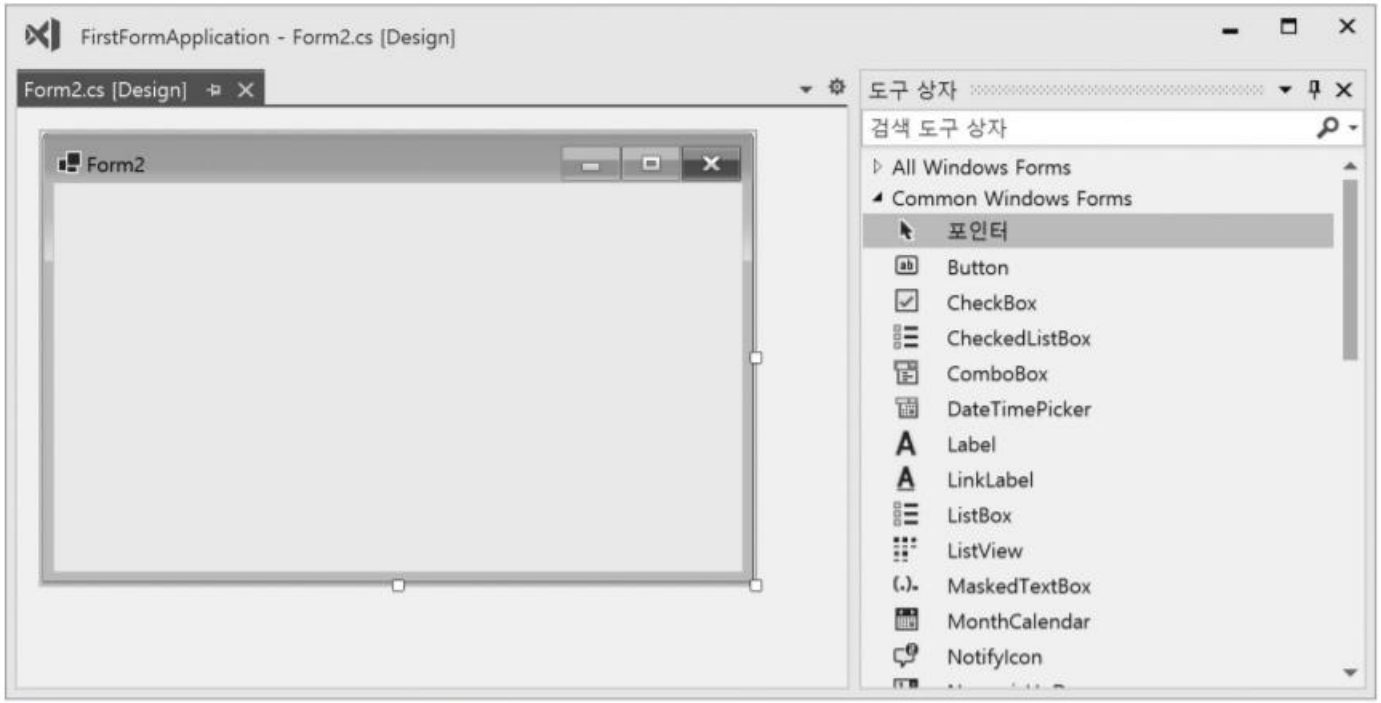


그림 7-28 Form2 클래스의 디자인 화면

Section 10 윈도 폼: 윈도 폼에서 상속과 다형성 활용하기(14)

- 디자인 화면에서 드래그해서 요소를 화면에 올려 놓기

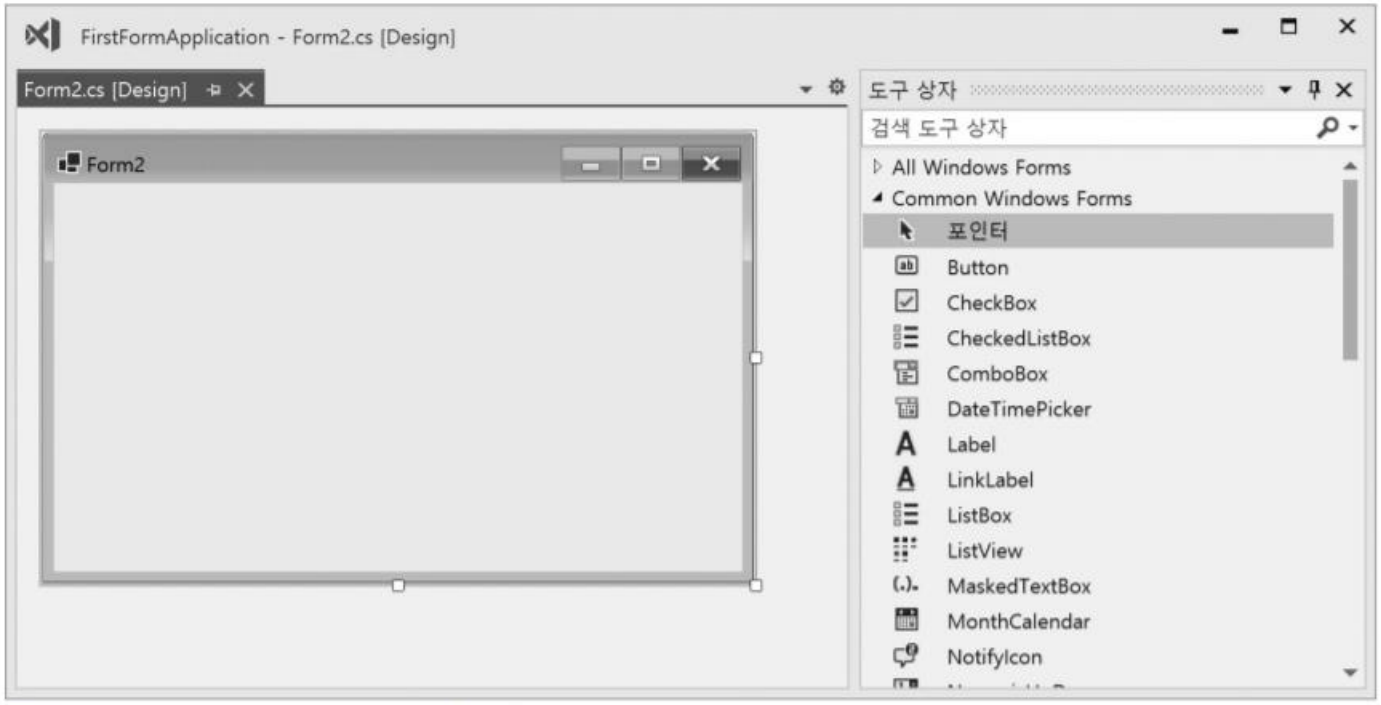


그림 7-28 Form2 클래스의 디자인 화면