

# Chapter 06 메서드

01 메서드 기본 형태

02 매개변수와 반환

03 클래스 메서드

04 오버로딩

05 접근 제한자

06 생성자

07 소멸자

08 속성

09 값 복사와 참조 복사

10 함께하는 응용예제

11 원도 품: 원도 품에서 메서드 활용하기

요약

연습문제

# Section 01 메서드 기본 형태(1)

## ■ 메서드의 기본 형태

```
[접근제한자] [반환형] [메서드 이름]([매개변수])  
{  
    _____ 이후에 자세히 알아볼 것이므로 일단 public을 입력한다고 생각합시다.  
    [메서드 코드]  
}
```

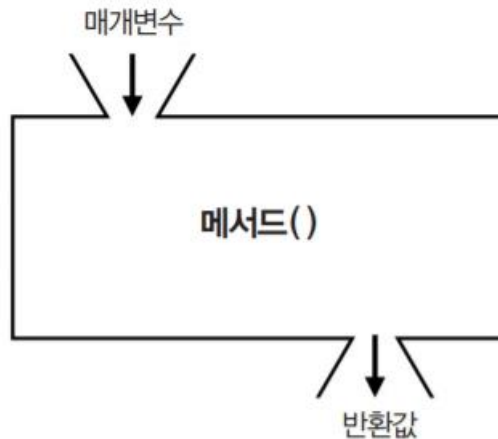


그림 6-1 함수 상자와 메서드

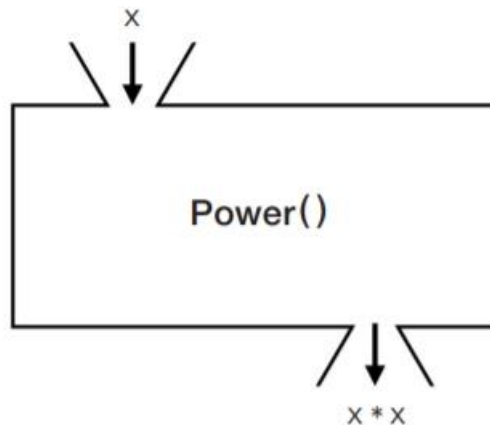


그림 6-2 Power() 메서드

# Section 01 메서드 기본 형태(2)

- 기본예제 6-1 인스턴스 메서드 생성과 사용(교재 266p)

/6장/InstanceMethod

코드 6-1

인스턴스 메서드 생성과 사용

```
01 class Program
02 {
03     class Test
04     {
05         public int Power(int x)
06         {
07             return x * x;
08         }
09     }
10
11     static void Main(string[] args)
12     {
13         Test test = new Test();
14         Console.WriteLine(test.Power(10));
15         Console.WriteLine(test.Power(20));
16     }
17 }
```

실행 결과

100

400

## Section 01 메서드 기본 형태(3)

- 메서드는 매개변수를 여러 개 가질 수 있음
- 예

코드 6-2

두 개의 매개변수를 갖는 메서드

/6장/Methods

```
01 class Program
02 {
03     class Test
04     {
05         public int Multi(int x, int y)
06         {
07             return x * y;
08         }
09     }
10
11     static void Main(string[] args)
12     {
13         Test test = new Test();
14         Console.WriteLine(test.Multi(52, 273));
15         Console.WriteLine(test.Multi(103, 32));
16     }
17 }
```

# Section 01 메서드 기본 형태(4)

코드 6-3

아무것도 반환하지 않는 메서드

/6장/Methods

```
01 class Program
02 {
03     class Test
04     {
05         public void Print()
06         {
07             Console.WriteLine("Print() 메서드가 호출되었습니다.");
08         }
09     }
10
11
12     static void Main(string[] args)
13     {
14         Test test = new Test();
15         test.Print();
16         test.Print();
17         test.Print();
18     }
19 }
```

## Section 02 매개 변수와 반환(1)

- 반환 메서드 형태

```
public 자료형 메서드(자료형 매개변수, 자료형 매개변수)
{
    자료형 output = 초깃값;

    // output에 값을 계산

    return output;
}
```

## Section 02 매개 변수와 반환(2)

### ■ 기본예제 6-2 매개 변수와 반환(1)(교재 270p)

/6장/SumMethod

```
class Program
{
    class Test
    {
        public int Sum(int min, int max)
        {
            int output = 0;
            for (int i = min; i <= max; i++)
            {
                output += i;
            }
            return output;
        }
    }

    static void Main(string[] args)
    {
        Test test = new Test();
        Console.WriteLine(test.Sum(1, 100));
    }
}
```

실행 결과

5050

## Section 02 매개 변수와 반환(2)

### ■ 기본예제 6-3 매개 변수와 반환(2)(교재 270p)

/6장/MultiplyMethod

```
class Program
{
    class Test
    {
        public int Multiply(int min, int max)
        {
            int output = 1;
            for (int i = min; i <= max; i++)
            {
                output *= i;
            }
            return output;
        }
    }

    static void Main(string[] args)
    {
        Test test = new Test();
        Console.WriteLine(test.Multiply(1, 10));
    }
}
```

실행 결과

3628800



# Section 03 클래스 메서드(1)

코드 6-6

Main() 메서드는 클래스 메서드

/6장/Methods

```
01 class Program
02 {
03     static void Main(string[] args)
04     {
05
06     }
07 }
```

## ■ 클래스 메서드 생성 방법

```
[접근 제한자] static [반환형] [메서드 이름]([매개변수])
{
}
}
```

## ■ 클래스 메서드 사용 방법

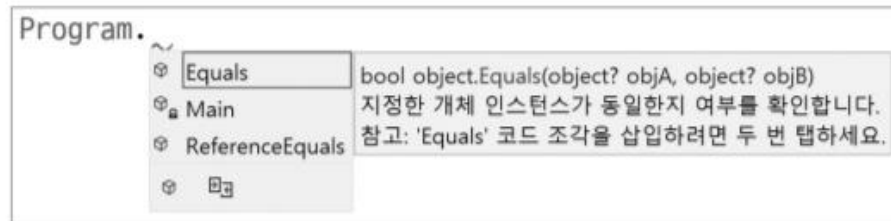


그림 6-5 Main() 메서드

## Section 03 클래스 메서드(2)

- 기본예제 6-4 클래스 메서드 생성과 사용(교재 274p)

/6장/ClassMethod

```
class Program
{
    class MyMath
    {
        public static int Abs(int input)
        {
            if (input < 0)
            {
                return -input;
            }
            else
            {
                return input;
            }
        }
    }

    static void Main(string[] args)
    {
        Console.WriteLine(MyMath.Abs(52));
        Console.WriteLine(MyMath.Abs(-273));
    }
}
```

실행 결과

52

273

## ■ 클래스 메서드에서 사용할 수 있는 것

- 클래스 메서드에서는 메모리에 올라가지 않은 인스턴스 변수, 인스턴스 메서드 사용 못 함

코드 6-8 클래스 메서드에서 인스턴스 변수 사용은 오류가 발생

/6장/ClassMethods

```
class Program
{
    public int instanceVariable = 10;

    static void Main(string[] args)
    {
        Console.WriteLine(instanceVariable);
    }
}
```

코드 6-9 클래스 메서드에서는 클래스 변수 사용만 사용 가능

/6장/ClassMethods

```
class Program
{
    public static int instanceVariable = 10;

    static void Main(string[] args)
    {
        Console.WriteLine(instanceVariable);
    }
}
```

추가해주었습니다.

## Section 04 오버로딩(1)

- 오버로딩(Overloading) : 이름은 같고, 매개변수가 다른 메서드를 만드는 것

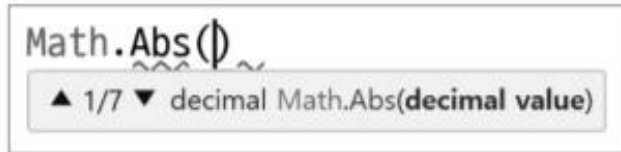


그림 6-6 Math.Abs() 메서드의 형태(1)

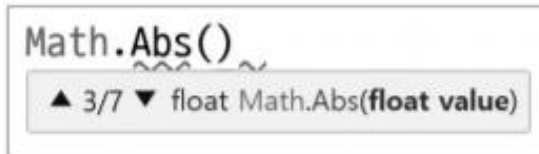


그림 6-7 Math.Abs() 메서드의 형태(2)

## Section 04 오버로딩(2)

### ■ 기본예제 6-5 메서드 오버로딩(교재 277p)

/6장/Overloading

```
class Program
{
    class MyMath
    {
        public static int Abs(int input)
        {
            if (input < 0) { return -input; }
            else { return input; }
        }

        public static double Abs(double input)
        {
            if (input < 0) { return -input; }
            else { return input; }
        }

        public static long Abs(long input)
        {
            if (input < 0) { return -input; }
            else { return input; }
        }
    }
}
```

```
static void Main(string[] args)
{
    // int
    Console.WriteLine(MyMath.Abs(52));
    Console.WriteLine(MyMath.Abs(-273));

    // double
    Console.WriteLine(MyMath.Abs(52.273));
    Console.WriteLine(MyMath.Abs(-32.103));

    // long
    Console.WriteLine(MyMath.Abs(21474836470));
    Console.WriteLine(MyMath.Abs(-21474836470));
}
```

실행 결과

```
52
273
52.273
32.103
21474836470
21474836470
```

## Section 04 오버로딩(3)

- 자동 완성 기능으로 출력 결과

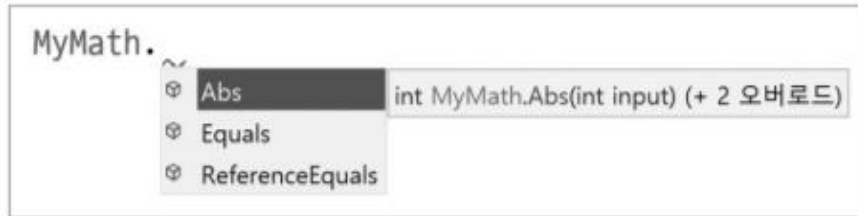


그림 6-8 오버로딩 용어 확인

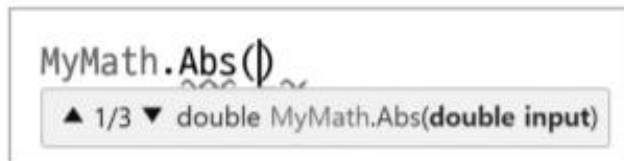


그림 6-9 3개의 메서드가 오버로딩됨

## Section 05 접근 제한자(1)

- 접근 제한자의 형태

```
[접근 제한자] [자료형] [변수 이름]
[접근 제한자] [반환형] [메서드 이름]([매개변수])
{
    [메서드 코드]
}
```

- 대표적 접근 제한자 : public, private

접근제한자	의미
private	클래스 내부에서만 접근 가능
public	모든 곳에서 해당 멤버로 접근 가능
protected	클래스 외부에서는 접근할 수 없으며 파생 클래스에서만 접근 가능
internal	같은 어셈블리에서만 public 권한으로 접근 가능

## Section 05 접근 제한자(2)

### ■ private 접근 제한자

- 접근 제한자를 입력하지 않으면 자동으로 private 접근 제한자로 설정
- 예

코드 6-12 Main() 메서드는 기본적으로 private 메서드

/6장/PrivateModifiers

```
01 static void Main(string[] args)
02 {
03
04 }
```

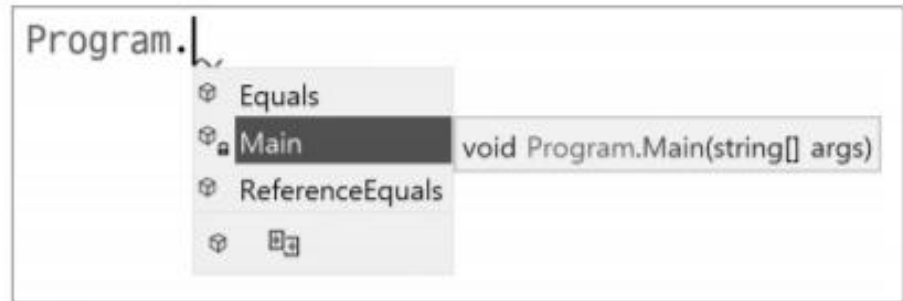


그림 6-11 자물쇠가 채워져 있는 Main() 메서드

- Main( ) 메서드의 그림 옆에 자물쇠(private 접근 제한자 적용 되었다는 의미)
- private 접근 제한자 적용 되면 : 자신의 클래스 내부에서만 해당 메서드 사용 가능



## Section 05 접근 제한자(3)

- 다른 클래스를 만들고 다른 클래스에서 Program 클래스의 Main( ) 메서드 호출

코드 6-13

외부 클래스에서의 접근

/6장/PrivateModifiers

```
01 class Test
02 {
03     public void TestMethod()
04     {
05         Program.Main(new string[] { "" });
06     }
07 }
08
09 class Program
10 {
11     static void Main(string[] args)
12     {
13
14     }
15 }
```

```
public void TestMethod()
{
    Program.Main(new string[] { "" });
}
```

void Program.Main(string[] args)

CS0122: '보호 수준 때문에 'Program.Main(string[])'에 액세스할 수 없습니다.

잠재적 수정 사항 표시 (Alt+Enter 또는 Ctrl+.)

그림 6-12 접근 제한자로 인한 접근 불가 상태

## Section 05 접근 제한자(4)

코드 6-14

내부 클래스에서의 접근

/6장/PrivateModifiers

```
01 class Program
02 {
03     class Test
04     {
05         public void TestMethod()
06         {
07             Program.Main(new string[] { "" }); —— 내부 클래스의 메서드에서 private 메서드 접근
08         }
09     }
10
11     public void TestMethod()
12     {
13         Program.Main(new string[] { "" }); —— 자신의 클래스가 가진 메서드에서 private 메서드 접근
14     }
15
16     static void Main(string[] args)
17     {
18
19     }
20 }
```

일단 private이 걸린 변수 또는 메서드는 외부에서 접근할 수 없다.

# Section 05 접근 제한자(5)

## ■ public 접근 제한자

- 다른 클래스에서 Main( ) 메서드 호출 가능

코드 6-15

public 접근 제한자를 붙인 Main() 메서드

/6장/PublicModifiers

```
01 class Test
02 {
03     public void TestMethod()
04     {
05         Program.Main(new string[] { "" });
06     }
07 }
08
09 class Program
10 {
11     public static void Main(string[] args)
12     {
13         접근 제한자를 추가했습니다.
14     }
15 }
```

- public 접근 제한자가 걸린 변수 또는 메서드는 모든 곳에서 접근 가능

# Section 06 생성자(1)

## ■ 생성자Constructor

- 무언가를 생성할 때 자동으로 호출되는 메서드
- 인스턴스 생성자의 생성 조건
  - 이름이 클래스 이름과 같아야 함
  - 접근 제한자는 public
  - 반환과 관련된 선언을 하지 않을 것
- 생성자의 형태

```
public [클래스 이름]([매개변수])  
{  
  
}
```

## Section 06 생성자(2)

- 생성자의 인스턴스 변수 초기화

코드 6-16

기본적인 생성자의 모습

/6장/Constructors

```
01 class Product
02 {
03     public string name;
04     public int price;
05
06     public Product(string name, int price)
07     {
08         this.name = name;
09         this.price = price;
10     }
11 }
```

this 키워드는 클래스 자기 자신을 나타냅니다.  
따라서 this.name은 자신의 name입니다.

this.price는 자신의 price입니다.

## Section 06 생성자(3)

### ■ 기본예제 6-6 인스턴스 생성 개수 확인(교재 285p)

/6장/ConstructorWithCounter

```
class Program
{
    class Product
    {
        public static int counter = 0;
        public int id;
        public string name;
        public int price;

        public Product(string name, int price)
        {
            Product.counter = counter + 1;
            this.id = counter;
            this.name = name;
            this.price = price;
        }
    }

    static void Main(string[] args)
    {
        Product productA = new Product("감자", 2000);
        Product productB = new Product("고구마", 3000);

        Console.WriteLine(productA.id + ":" + productA.name);
        Console.WriteLine(productB.id + ":" + productB.name);
        Console.WriteLine(Product.counter + "개 생성되었습니다.");
    }
}
```

#### 실행 결과

1:감자  
2:고구마  
2개 생성되었습니다.

## ■ private 생성자

- 생성자로 클래스의 인스턴스를 만들 수 없게 할 때는 private 생성자 사용
  - 수학적 메서드만 제공하는 Math 클래스의 경우 인스턴스를 만들어도 아무 효용성이 없음
    - 이런 클래스의 경우 인스턴스를 못 만들게 private 생성자를 정의
    - **C#의 Math 클래스는 인스턴스를 만들 수 있음!**

코드 6-18 private 생성자

/6장/Constructors

```
class Program
{
    class Hidden
    {
        private Hidden() { }
    }

    static void Main(string[] args)
    {
        Hidden hidden = new Hidden();
    }
}
```

오류가 발생합니다.

## ■ 정적 생성자

- 정적 요소를 초기화할 때에 사용
- 정적 생성자 사용의 제한
  - 접근 제한자 사용 못함.
  - 매개변수 사용 못함.

코드 6-19 정적 생성자

/6장/Constructors

```
class Sample
{
    public static int value;

    static Sample()
    {
        value = 10;
        Console.WriteLine("정적 생성자 호출");
    }
}
```



## NOTE(3)

- 정적 요소를 사용할 때, 또는 인스턴스를 생성하는 초기 시점에 한 번만 호출
- 해당 클래스와 관련된 요소를 처음 사용하는 시점에 자동 호출(별도 호출 불가능)

코드 6-20 정적 생성자의 호출 시점

/6장/Constructors

```
static void Main(string[] args)
{
    Console.WriteLine("첫 번째 위치");
    Console.WriteLine(Sample.value);
    Console.WriteLine("두 번째 위치");
    Sample sample = new Sample();
    Console.WriteLine("세 번째 위치");
}
```

```
static void Main(string[] args)
{
    Console.WriteLine("첫 번째 위치");
    Sample sample = new Sample();
    Console.WriteLine("두 번째 위치");
    Console.WriteLine(Sample.value);
    Console.WriteLine("세 번째 위치");
}
```

## Section 07 소멸자(1)

### ■ 소멸자Destructor : 인스턴스가 소멸될 때에 호출

- 변수가 더 이상 사용되지 않을 것이 확실할 때 객체 소멸시키며 소멸자 호출
- 소멸자 생성 규칙
  - 이름은 클래스 이름 앞에 ~ 기호 붙은 것
  - 접근 제한자 사용 없음
  - 반환과 관련된 선언 하지 않음
  - 매개변수와 관련된 선언 하지 않음
  - 하나의 클래스에는 하나의 소멸자만
- 소멸자의 형태

```
~[클래스 이름]()  
{  
  
}
```

## Section 07 소멸자(2)

### ■ 기본예제 6-7 소멸자 생성과 사용(교재 289p)

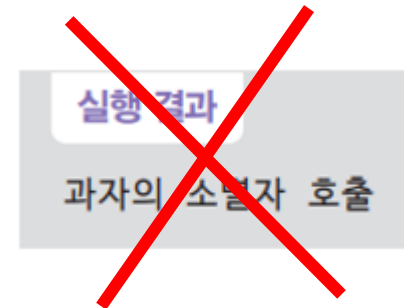
/6장/Destructor

```
class Program
{
    class Product
    {
        public string name;
        public int price;

        public Product(string name, int price)
        {
            this.name = name;
            this.price = price;
        }

        ~Product()
        {
            Console.WriteLine(this.name + "의 소멸자 호출");
        }
    }

    static void Main(string[] args)
    {
        Product product = new Product("과자", 1000);
    }
}
```



## Section 07 소멸자(2)

### C# 소멸자 명시적 호출

- C / C++의 경우 new / delete 키워드를 통해서 프로그래머가 메모리 관리
- C#의 경우 프로그래머 직접 관리 방식이 아닌 **Garbage Collector(GC)**를  
통해 객체의 생명주기 관리
- 소멸자를 명시적으로 호출한다 하더라도
  - .NET 5(.NET Core 포함) 이상 버전인 경우 애플리케이션이 종료될 때  
소멸자를 호출하지 않음
  - 콘솔창에서 소멸자가 호출되었는지 확인 불가능
- .NET 5(.NET Core 포함) 이상 버전에서 소멸자를 강제로 실행하기  
위해서는 GC.Collect() 호출

## Section 07 소멸자(2)

- 기본예제 6-7 소멸자 생성과 사용(교재 289p) → 코드 수정!

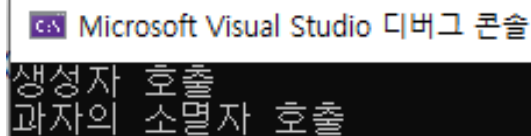
```
internal class Program
{
    class Product
    {
        public string name;
        public int price;

        public Product(string name, int price)
        {
            this.name = name;
            this.price = price;
            Console.WriteLine("생성자 호출");
        }

        ~Product()
        {
            Console.WriteLine(this.name + "의 소멸자 호출");
        }
    }
}
```

```
public static void Sample()
{
    Product product = new Product("과자", 1000);
}

private static void Main(string[] args)
{
    Sample();
    GC.Collect();
}
```



Microsoft Visual Studio 디버그 콘솔

```
생성자 호출
과자의 소멸자 호출
```

## ■ 상수

- 일반적인 변수는 값 계속 변경 가능하지만, 상수로 선언된 변수는 값 변경 불가능

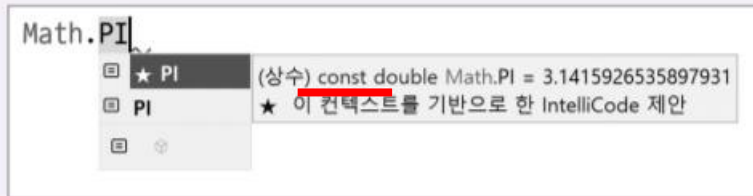


그림 6-14 상수

코드 6-22 상수 변경

/6장/Constants

```
static void Main(string[] args)
{
    Math.PI = 20;

    int r = 10;
    Console.WriteLine("원의 둘레: " + (2 * Math.PI * r));
    Console.WriteLine("원의 넓이: " + (Math.PI * r * r));
}
```

Math.PI = 20;

☐ (상수) const double Math.PI = 3.1415926535897931  
CS0131: 할당식의 왼쪽은 변수, 속성 또는 인덱서여야 합니다.

그림 6-15 상수를 변경할 때의 오류

# NOTE(2)

- 상수 만들기

코드 6-23 상수 생성

/6장/Constants

```
class MyMath
{
    public const double PI = 3.141592;
}
```

코드 6-24 메서드 내부에서 상수 사용

/6장/Constants

```
static void Main(string[] args)
{
    const int value = 10;

    Console.WriteLine(value);
}
```

## ■ readonly 키워드

- 읽기 전용 변수
- 변수 선언 시점과 생성자 메서드에서만 값 변경 가능(이 외에는 오류 발생)

코드 6-25 readonly 키워드

/6장/Constants

```
class Product
{
    private static int count;
    public readonly int id;
    public string name;
    public int price;
```

```
    public Product(string name, int price)
```

```
    {
        id = count++;
        this.name = name;
        this.price = price;
    }
}
```

생성자에서는 readonly 키워드를 적용한 변수를 변경할 수 있습니다.

```
Product product = new Product("이름", 100);
product.id = 10;
```

(지역 변수) Product product

CS0191: 읽기 전용 필드에는 할당할 수 없습니다. 단, 필드가 정의된 형식의 생성자 또는 초기값 전용 setter나 변수 이니셜라이저에서는 예외입니다.

그림 6-16 readonly 키워드를 적용한 변수를 생성자가 아닌 곳에서 변경할 때의 오류



## ■ 캡슐화

- 변수 width와 height에 음수가 들어갈 수 있는 위험성 원천 봉쇄

코드 6-27

변수 width와 height를 건드리지 못하게 수정

/6장/Boxes

```
01 class Box
02 {
03     private int width;
04     private int height;
05
06     public Box(int width, int height)
07     {
08         if (width > 0 || height > 0)
09         {
10             this.width = width;
11             this.height = height;
12         }
13         else
14         {
15             Console.WriteLine("너비와 높이는 자연수로 초기화해주세요!");
16         }
17     }
18
19     public int Area()
20     {
21         return this.width * this.height;
22     }
23 }
```

## ■ 캡슐화

- width와 height에 들어간 값 확인 불가능

코드 6-27

변수 width와 height를 건드리지 못하게 수정

/6장/Boxes

```
01 class Box
02 {
03     private int width;
04     private int height;
05
06     public Box(int width, int height)
07     {
08         if (width > 0 || height > 0)
09         {
10             this.width = width;
11             this.height = height;
12         }
13         else
14         {
15             Console.WriteLine("너비와 높이는 자연수로 초기화해주세요!");
16         }
17     }
18
19     public int Area()
20     {
21         return this.width * this.height;
22     }
23 }
```

### ■ 겐터와 셋터

- 변수를 바로 수정할 수는 없지만, 변수 변경 메서드를 만들고 메서드를 호출해 변경

코드 6-28

겐터와 셋터

/6장/Boxes

```
01 class Box
02 {
03     // 변수
04     private int width;
05     private int height;
06
07     // 생성자
08     public Box(int width, int height)
09     {
10         if (width > 0 || height > 0)
11         {
12             this.width = width;
13             this.height = height;
14         }
15         else { Console.WriteLine("너비와 높이는 자연수로 초기화해주세요!"); }
16     }
17
18     // 인스턴스 메서드
19     public int Area() { return this.width * this.height; }
20 }
```

## Section 08 속성(3)

```
21    // 겐터(Getter)
22    public int GetWidth() { return width; }
23    public int GetHeight() { return height; }
24
25    // 셋터(Setter)
26    public void SetWidth(int width)
27    {
28        if (width > 0) { this.width = width; }
29        else { Console.WriteLine("너비는 자연수를 입력해주세요"); }
30    }
31
32    public void SetHeight(int height)
33    {
34        if (height > 0) { this.height = height; }
35        else { Console.WriteLine("높이는 자연수를 입력해주세요"); }
36    }
37 }
```

쓸데 없이 코드가 길어짐!!!

## Section 08 속성(4)

### ■ 일반적인 속성 생성 방법

- 속성 : getter와 setter를 쉽게 만드는 방법
- 속성의 일반적인 형태

```
private int [변수 이름];  
public int [속성 이름]           속성 이름은 대문자로 시작! (개발자들의 약속)  
{  
    get { return [변수 이름]; }  
    set { [변수 이름] = value; }  
}
```

- 속성의 사용 방법

```
[인스턴스 이름].[속성 이름]           // getter 호출  
[인스턴스 이름].[속성 이름] = [값]     // setter 호출
```

## Section 08 속성(5)

### ■ 기본예제 6-8 일반적인 속성(교재 297p)

/6장/BoxProperty

```
class Program
{
    class Box
    {
        // 변수와 속성
        private int width;
        public int Width
        {
            get { return width; }
            set
            {
                if (value > 0) { width = value; }
                else { Console.WriteLine("너비는 자연수를 입력해주세요"); }
            }
        }

        private int height;
        public int Height
        {
            get { return height; }
            set
            {
                if (value > 0) { height = value; }
                else { Console.WriteLine("높이는 자연수를 입력해주세요"); }
            }
        }
    }
}

// 생성자
public Box(int width, int height)
{
    Width = width;
    Height = height;
}

// 인스턴스 메서드
public int Area() { return this.width * this.height; }

static void Main(string[] args)
{
    Box box = new Box(-10, -20);

    box.Width = -200;
    box.Height = -100;
}
```

## Section 08 속성(5)

- 기본예제 6-8 일반적인 속성(교재 297p)      /6장/BoxProperty

### 실행 결과

너비는 자연수를 입력해주세요  
높이는 자연수를 입력해주세요  
너비는 자연수를 입력해주세요  
높이는 자연수를 입력해주세요



그림 6-17 속성 기호

## Section 09 값 복사와 참조 복사(1)

- C#의 모든 자료형은 값<sup>Value</sup> 또는 참조<sup>Reference</sup> 두 가지로 분류
  - int/float 등 기본 자료형<sup>Primitive Type</sup>은 값
  - 클래스로 만들어진 인스턴스(객체)는 참조
  - 메서드의 매개변수로 값과 참조 전달 시 큰 차이 발생

```
static void Main(string[] args)
{
    int a = 10;
}
```

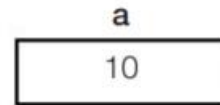


그림 6-19 기본 자료형 선언

```
static void Main(string[] args)
{
    Random a = new Random();
}
```

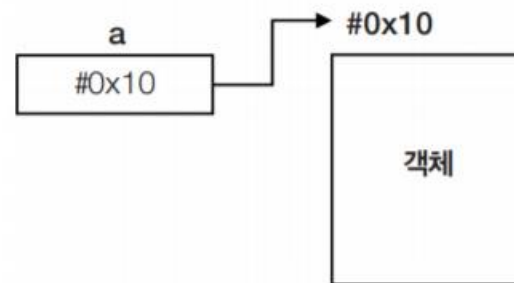


그림 6-20 객체 자료형 선언



## Section 09 값 복사와 참조 복사(2)

### ■ 값 복사의 경우

#### ■ 예

코드 6-30 값 복사 예

```
01 class Program
02 {
03     static void Change(int input)
04     {
05         input = 20;
06     }
07
08     static void Main(string[] args)
09     {
10         int a = 10;
11         Change(a);
12         Console.WriteLine(a);
13     }
14 }
```

실행 결과

10

## Section 09 값 복사와 참조 복사(3)



그림 6-21 값 복사의 과정



그림 6-22 값 복사된 변수를 변경

### 코드 6-31 간단한 값 복사 예

```
01 static void Main(string[] args)
02 {
03     int a = 10;
04     int input = a;
05     input = 20;
06     Console.WriteLine(a);
07 }
```

## Section 09 값 복사와 참조 복사(4)

### ■ 참조 복사의 경우

#### ■ 예

코드 6-32 참조 복사 예

```
class Program
{
    class Test
    {
        public int value = 10;
    }

    static void Change(Test input)
    {
        input.value = 20;
    }

    static void Main(string[] args)
    {
        Test test = new Test();
        test.value = 10;
        Change(test);

        Console.WriteLine(test.value);
    }
}
```

실행 결과

20

## Section 09 값 복사와 참조 복사(5)

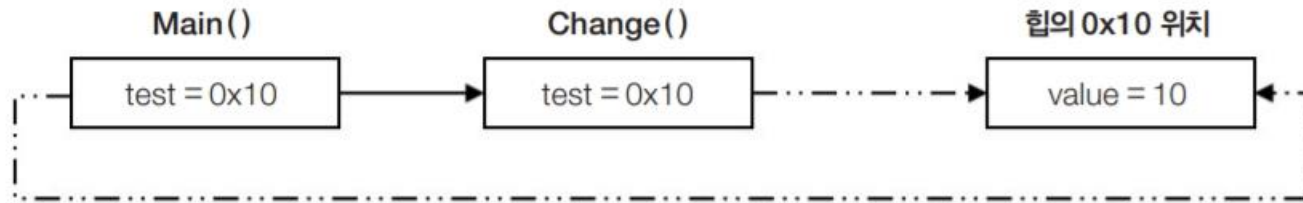


그림 6-23 참조 복사의 과정

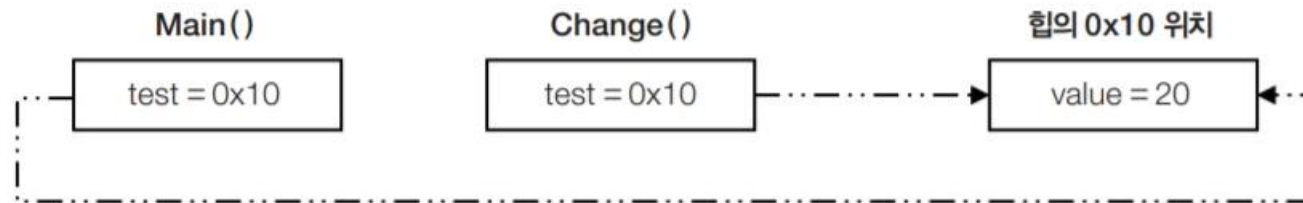


그림 6-24 참조 복사된 객체를 변경

### 코드 6-33 간단한 참조 복사 예

```
01 static void Main(string[] args)
02 {
03     Test testA = new Test();
04     Test testB = testA;
05     testA.value = 10;
06     testB.value = 20;
07     Console.WriteLine(testA.value);
08 }
```

## Section 10 함께하는 응용예제(1)

- 응용예제 6-1 재귀 메서드(교재 306p)

/6장/Recursion

- 예

코드 6-34 재귀 메서드

```
01 class Program
02 {
03     static void Main(string[] args)
04     {
05         Main(new string[0]); —— 자기 자신을 호출합니다.
06     }
07 }
```

- 항상 자기 자신을 탈출할 수 있는 종료 조건이 포함되어야 함

## Section 10 함께하는 응용예제(2)

### ■ 피보나치 수로 재귀 메서드 만들기

코드 6-35 재귀 메서드를 사용한 피보나치 인스턴스 메서드

```
01 class Fibonacci
02 {
03     public long Get(int i)
04     {
05         if (i < 0) { return 0; }
06         if (i == 1) { return 1; }
07         return Get(i - 2) + Get(i - 1);
08     }
09 }
10
11 class Program
12 {
13     static void Main(string[] args)
14     {
15         Fibonacci fibo = new Fibonacci();
16         Console.WriteLine(fibo.Get(1));
17         Console.WriteLine(fibo.Get(2));
18         Console.WriteLine(fibo.Get(3));
19         Console.WriteLine(fibo.Get(4));
20         Console.WriteLine(fibo.Get(5));
21     }
22 }
```

피보나치 수는 기하급수적으로 커지므로 long 자료형을 사용하게 했습니다.

종료 조건입니다.

#### 실행 결과

1  
1  
2  
3  
5

## Section 10 함께하는 응용예제(3)

코드 6-36

재귀 메서드를 사용한 피보나치 클래스 메서드

```
01 class Fibonacci
02 {
03     public static long Get(int i)
04     {
05         if (i < 0) { return 0; }
06         if (i == 1) { return 1; }
07         if (i == 2) { return 1; }
08         return Get(i - 2) + Get(i - 1);
09     }
10 }
11
12 class Program
13 {
14     static void Main(string[] args)
15     {
16         Console.WriteLine(Fibonacci.Get(1));
17         Console.WriteLine(Fibonacci.Get(2));
18         Console.WriteLine(Fibonacci.Get(3));
19         Console.WriteLine(Fibonacci.Get(4));
20         Console.WriteLine(Fibonacci.Get(5));
21     }
22 }
```

클래스 메서드로 선언했습니다.

## Section 10 함께하는 응용예제(4)

- 응용예제 6-2 메모화(교재 295p) /6장/Memorize
- 메모화 : 한 번 계산했던 값 저장해두는 것

코드 6-37 메모화

```
class Fibonacci
{
    private static Dictionary<int, long> memo = new Dictionary<int, long>();
    public static long Get(int i)
    {
        // 기본 값
        if (i < 0) { return 0; }
        if (i == 1) { return 1; }

        // 이미 계산했던 값인지 확인
        if (memo.ContainsKey(i))
        {
            return memo[i];
        }
        else
        {
            계산한 피보나치 수를 저장하는 Dictionary 객체를 만듭니다.
        }
    }
}
```



## Section 10 함께하는 응용예제(5)

```
        // 계산하지 않았다면 계산
        long value = Get(i - 2) + Get(i - 1);
        memo[i] = value;
        return value;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine(Fibonacci.Get(40));
        Console.WriteLine(Fibonacci.Get(100));
    }
}
```

### 실행 결과

102334155

3736710778780434371

# 실습 문제

1. 다음 main() 메소드를 실행하였을 때 다음과 같이 출력되도록 TV 클래스를 작성하라.

```
private static void Main(string[] args)
{
    TV myTV = new TV("LG", 2017, 32); // LG에서 만든 2017년 32인치
    myTV.show();
}
```

LG에서 만든 2017년형 32인치 TV

2. Grade 클래스를 작성하라. 3과목의 점수를 입력받아 Grade 객체를 생성하고, 성적 평균(소수점이하 2자리)을 출력한다.

Hint) Grade 클래스에 int 타입의 math, science, english 필드를 private으로 선언, 생성자와 세 과목의 평균(실수)을 리턴하는 average() 메소드 작성

# Section 11 윈도 폼: 윈도 폼에서 메서드 활용하기(1)

## ■ 이벤트 정적 연결(디자인에서 연결)

- 도구 상자에서 버튼(Button 클래스), 텍스트 박스(TextBox 클래스), 레이블(Label 클래스) 드래그



그림 6-27 화면 디자인

### 코드 6-38 Form1.Designer.cs 파일에 자동 생성된 요소

```
01 private System.Windows.Forms.Button button1;  
02 private System.Windows.Forms.TextBox textBox1;  
03 private System.Windows.Forms.Label label1;
```

## Section 11 윈도 폼: 윈도 폼에서 메서드 활용하기(2)

- 디자인 화면에서 생성된 버튼 클릭, 속성 화면의 번개 모양 아이콘(이벤트) 클릭

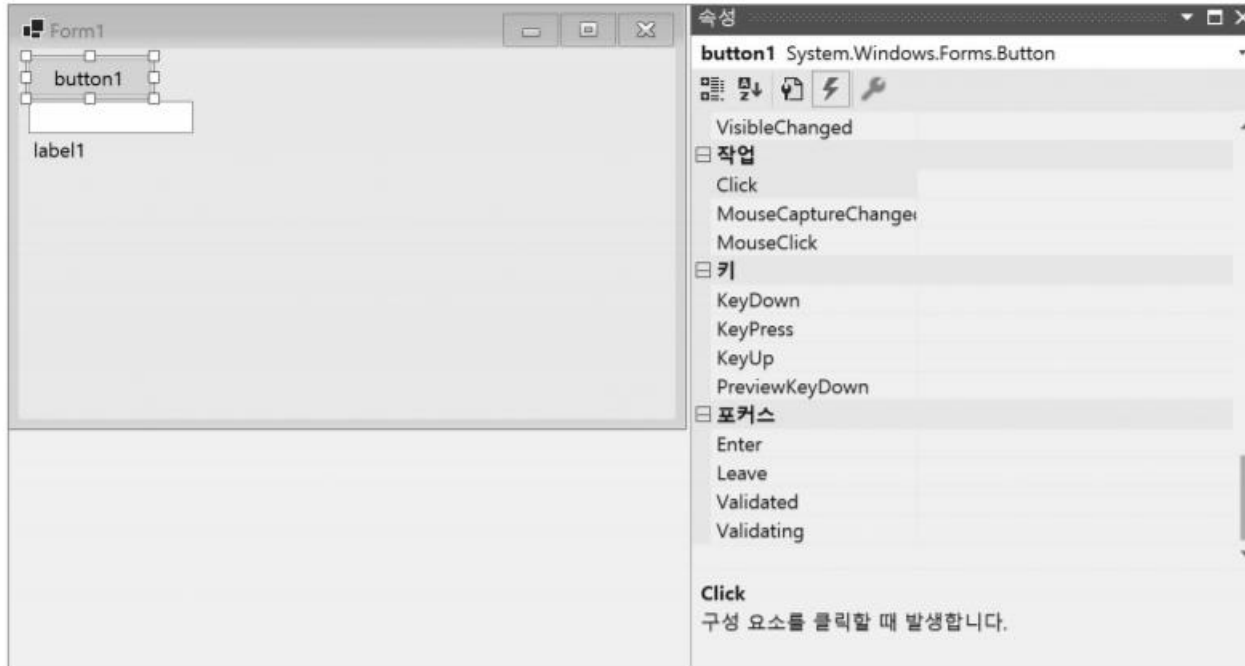


그림 6-28 버튼에 적용할 수 있는 이벤트

- Click 옆 빈 공간을 잡고 더블 클릭

# Section 11 윈도 폼: 윈도 폼에서 메서드 활용하기(3)

코드 6-39 자동으로 생성된 이벤트 메서드

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

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

— 자동으로 생성된 이벤트 메서드

- 디자인 화면의 속성 화면
  - Click 부분에 자동 생성된 button1\_Click ( )메서드 연결 확인



그림 6-29 연결된 이벤트

## Section 11 윈도 폼: 윈도 폼에서 메서드 활용하기(4)

코드 6-40

button1\_Click() 메서드

```
01 private void button1_Click(object sender, EventArgs e)
02 {
03     textBox1.Text += "+";
04     label1.Text += "+";
05 }
```



그림 6-30 버튼을 여러 번 클릭했을 때의 화면

# Section 11 윈도 폼: 윈도 폼에서 메서드 활용하기(5)

## ■ 이벤트 동적 연결(코드에서 연결)

코드 6-41 Form1.Designer.cs 파일에 자동 생성된 코드

```
01 this.button1.Location = new System.Drawing.Point(12, 12);
02 this.button1.Name = "button1";
03 this.button1.Size = new System.Drawing.Size(75, 23);
04 this.button1.TabIndex = 0;
05 this.button1.Text = "button1";
06 this.button1.UseVisualStyleBackColor = true;
07 this.button1.Click += new System.EventHandler(this.button1_Click);
```

여기가 바로 이벤트를 연결하는 부분입니다.

## ■ Form1.cs 파일에서 button1에 Click 이벤트 연결

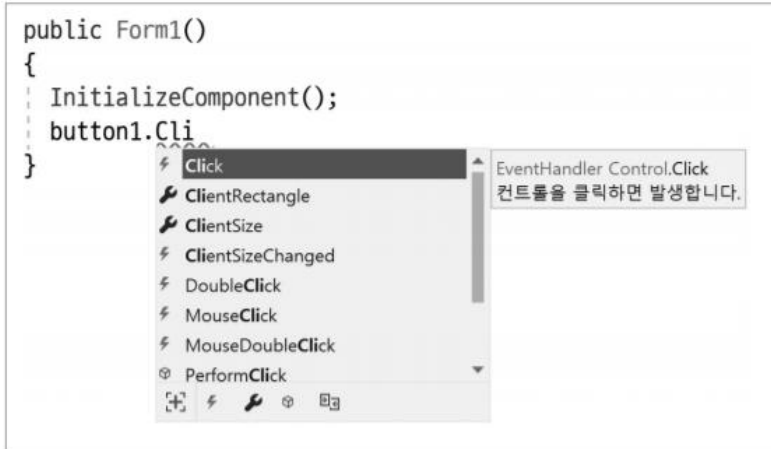


그림 6-31 번개 모양의 이벤트

## Section 11 윈도 폼: 윈도 폼에서 메서드 활용하기(6)

- 이벤트 메서드 자동 작성, 이벤트 뒤에 + = 기호 입력

```
public Form1()
{
    InitializeComponent();
    button1.Click +=
}
```

Button1\_Click; (삽입하려면 <Tab> 키 누름)

그림 6-32 삽입하려면 **Tab** 키를 누름



## Section 11 윈도 폼: 윈도 폼에서 메서드 활용하기(7)

코드 6-42 자동 생성된 이벤트 메서드

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();

        button1.Click += Button1_Click;
    }
}
```

```
private void Button1_Click(object sender, EventArgs e)
```

```
{
```

```
    throw new NotImplementedException();
```

```
}
```

구현되지 않았으니 구현해달라고 강제로 오류를 띄우는 코드입니다. 지금 버튼을 클릭하면 강제로 예외가 발생합니다. 이와 관련된 내용은 10장에서 다루지만 지금 기억해두면 그때 도움이 될 것입니다.

```
private void button1_Click(object sender, EventArgs e)
```

```
{
```

```
    textBox1.Text += "+";
```

```
    label1.Text += "+";
```

```
}
```

```
}
```

# Section 11 윈도 폼: 윈도 폼에서 메서드 활용하기(8)

## ■ 이벤트 메서드의 매개변수

- 이벤트 메서드의 형태

```
private void Button1_Click(object sender, EventArgs e)
{

}
}
```

- sender 객체 : 이벤트를 발생시킨 자기 자신을 나타냄
  - 예

### 코드 6-43 sender 객체 활용

```
01 private void Button1_Click(object sender, EventArgs e)
02 {
03     Button self = (Button)sender;
04     self.Text = "저를 클릭했습니다!";
05 }
```

## Section 11 윈도 폼: 윈도 폼에서 메서드 활용하기(9)

- 이벤트 정보 객체 : 이벤트와 관련된 추가 정보를 알려주는 객체
  - 예

코드 6-44 FormClosed 이벤트

```
01 public partial class Form1 : Form
02 {
03     public Form1()
04     {
05         InitializeComponent();
06         button1.Click += Button1_Click;
07         FormClosed += Form1_FormClosed;
08     }
09
10
11     private void Form1_FormClosed(object sender, FormClosedEventArgs e)
12     {
13         throw new NotImplementedException();
14     }
15
16     private void Button1_Click(object sender, EventArgs e) { /* 생략 */ }
17     private void button1_Click(object sender, EventArgs e) { /* 생략 */ }
18 }
```

자기 자신(Form1 클래스의 인스턴스)의 FormClosed 이벤트입니다.  
이는 부모에서 상속받은 이벤트인데요. 상속은 다음 장에서 알아보겠습니다.

이벤트 정보 객체의 형식이 이전과 다릅니다.

## Section 11 윈도 폼: 윈도 폼에서 메서드 활용하기(10)



그림 6-33 이벤트 정보 객체의 멤버

## ■ 백 그라운드 요소의 이벤트



그림 6-35 도구 상자의 구성 요소

# Section 11 윈도 폼: 윈도 폼에서 메서드 활용하기(12)

- 폼의 백그라운드에서 작동하는 요소들

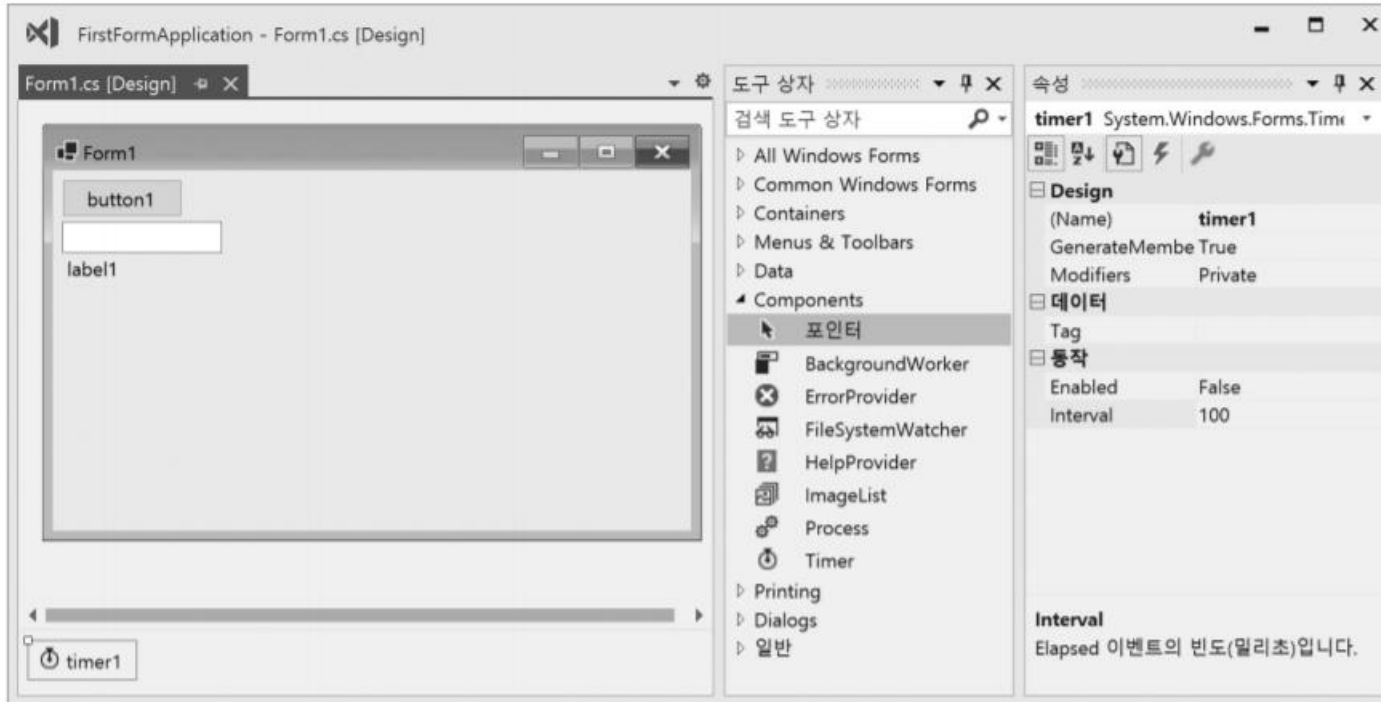


그림 6-36 타이머(Timer 클래스)

- 타이머 : 특정 시간 간격마다 특정한 코드를 호출해주는 기능

## Section 11 윈도 폼: 윈도 폼에서 메서드 활용하기(13)

- 버튼을 누르면 버튼의 Enabled 속성을 true로 바꿔서 타이머를 작동시키고, 프로그램을 실행한 지 몇 초나 지난는지 확인하는 프로그램

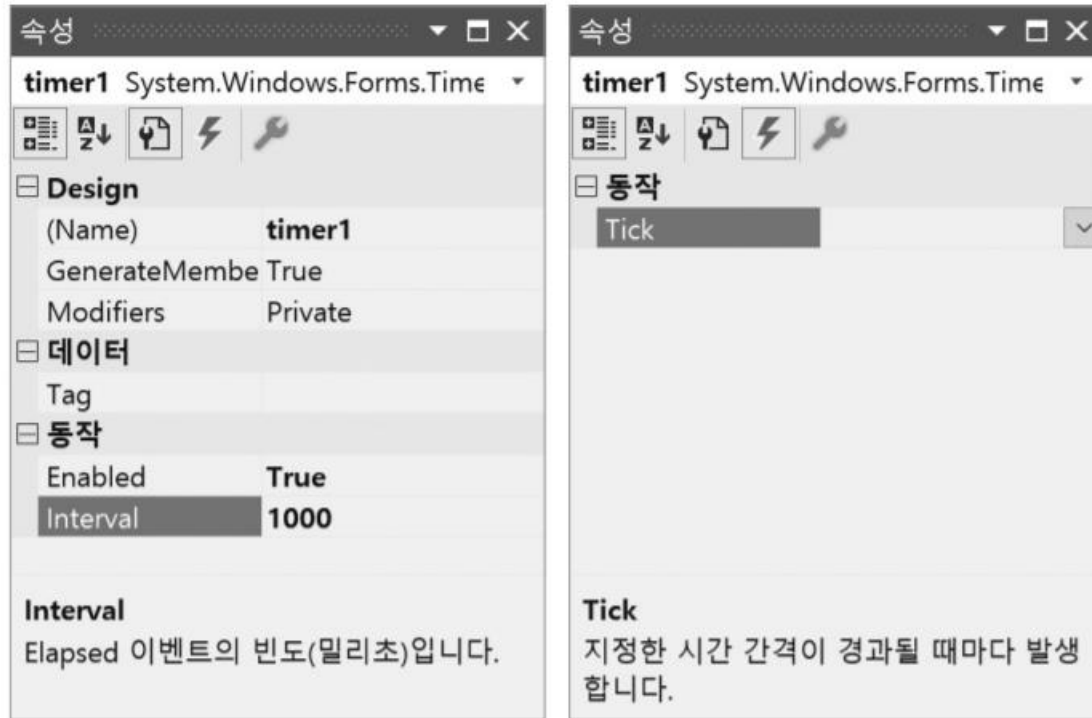


그림 6-37 타이머 객체의 속성 지정

### 코드 6-45 Tick 이벤트의 이벤트 메서드

```
01 private void timer1_Tick(object sender, EventArgs e)
02 {
03
04 }
```

## Section 11 윈도 폼: 윈도 폼에서 메서드 활용하기(14)

코드 6-46

시간 경과 확인 코드

```
01 private int elapsedTime = 0;
02 private void timer1_Tick(object sender, EventArgs e)
03 {
04     elapsedTime++;
05     textBox1.Text = elapsedTime + "초 경과";
06     label1.Text = elapsedTime + "초 경과";
07 }
```



그림 6-38 프로그램이 시작된 지 몇 초가 지났는지 출력



# Section 11 : 단위 환산

Form1

단위환산

inch 입력 :

inch to cm 결과 :

10초 경과

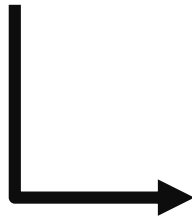
1. 본인의 학번과 이름을 제목으로 추가
2. 글자폰트 사이즈 : 15
3. 소수점이하 2자리 출력

※ 과제 제출 방법

- 과제 제출은 실행 전 화면과 실행 후 화면을 캡처하여 제출
- 코드 동작은 수업시간에 확인!

Form1

실행결과 화면



저를 클릭했습니다!

inch 입력 :

inch to cm 결과 : 6.35 cm

120초 경과