제 1 장 시작하기

컴퓨터가 특정한 작업을 수행하도록 지시하는 명령 모음을 **프로그램**(program)이라 한다. 일반적으로 하드 디스크와 같은 저장 매체에 이진 파일(binary file)로 저장되어 있다가 메모리에 적재되고 중앙 처리 장치(CPU, central processing unit)에 의하여 실행된다. 개발자가 작성한 코드와 이 코드를 컴파일하고 링크하여 얻어지는 실행 파일을 통칭하여 프로그램이라 한다.

프로그래밍 언어(programming language)는 프로그램을 만들기 위한 언어이다. 일반적으로 컴파일러가 이해하는 단어와 문법으로 이루어져 있다. 개발자는 프로그래밍 언어로 프로그램을 작성한 다음 컴파일하고 링크하여 실행 가능한 형태로 만든다. 컴퓨터는 프로그램에 주어진 명령을 실행하여 그 결과를 출력한다.

1.1 C# 개요

1999년 1월 Anders Hejlsberg는 새로운 프로그래밍 언어를 개발하기 위한 팀을 구성하였다. 당시에는 이 새로운 언어의 이름이 C-like Object Oriented Language의 줄임말인 Cool이었다. 그러나 상표권과 관련된 이유로 2000년 7월 C#으로 공표되었다.

2002년 1월 .NET Framework 1.0을 기반으로 한 C# 1.0이 발표되었다. 이후 계속 발전하면서 2012년 8월 .NET Framework 4.5을 기반으로 한 C# 5.0이 발표되었다.

1.2 콘솔 응용 프로그램

Visual Studio에서는 실행 프로그램(.exe)이나 동적 연결 라이브러리(.dll) 제작 단위를 **프로젝트**(project)라 한다. 프로그램 제작에 필요한 자료와 파일을 관리할 수 있으며 옵션을 설정할 수 있다. 응용 프로그램을 제작하는 데 필요한 항목을 모아놓은 것을 **솔루션**(solution)이라 한다. 프로젝트를 하나 이상 포함하고 있으며 포함된 모든 프로젝트에서 필요한 항목을 관리한다.

1.2.1 프로젝트 만들기. 프로젝트를 만드는 방법을 알아보자. Visual Studio를 실행하면 창 [Visual Studio 2019]가 출력될 것이다. 여기서 버튼 [새 프로젝트 만들기]를 클릭한다. 다른 작업을 하고 있었다면 메뉴에서

파일(F) -> 새로 만들기(N) -> 프로젝트(P)

를 클릭한다.

창 [새 프로젝트 만들기]가 출력될 것이다. 이 창에서 [C#], [Windows], [모든 프로젝트 형식(T)]와 [콘솔 앱(.NET Framework)]를 선택하고 버튼 [다음(N)]을 누른다(그림 1.2).

그림 1.1 새 프로젝트 만들기

창 [새 프로젝트 구성]이 열릴 것이다. 이 창에

프로젝트 이름(N): Hello World
위치(L): (솔루션 위치)
솔루션 이름(M): Ch01

을 입력하고 버튼 [만들기(C)]를 누른다(그림 1.2).

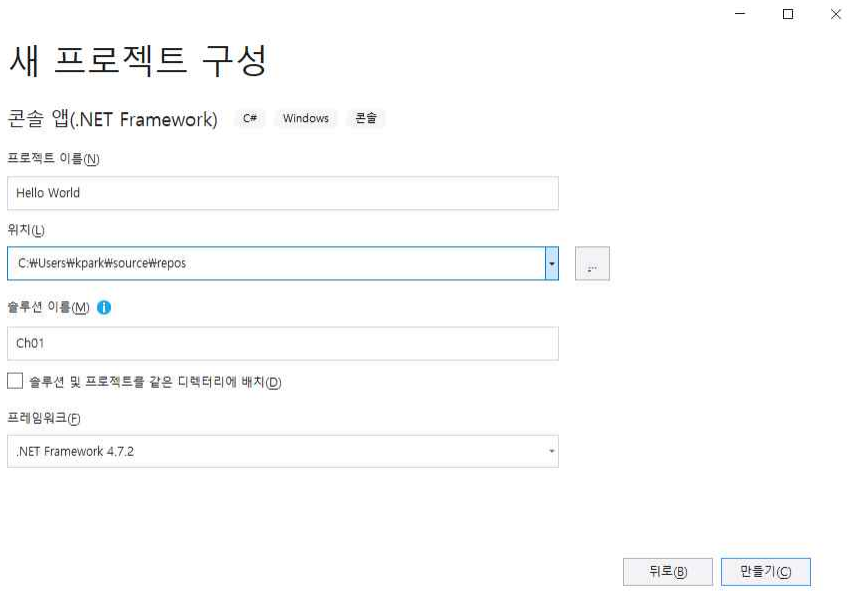


그림 1.2 새 프로젝트 구성

화면에서 솔루션 [Ch01], 프로젝트 [Hello World], 파일 [Program.cs]를 확인하기 바란다. 코드를 다음과 같이 수정한다.

예제 1.2.1 Hello World

```
using System;

namespace Hello_World
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

이 프로젝트를 실행하자.

1.2.2 빌드. 실행 파일이나 동적 연결 라이브러리를 만드는 것을 빌드(build)라 한다.

메뉴에서

빌드(B) -> 솔루션 빌드(B)

를 선택하면 프로젝트를 빌드하여 실행 파일을 만든다. 화면 아래 쪽의 탭 [출력]에는 그 결과가 출력된다(그림 1.3).

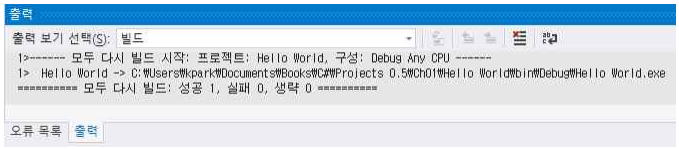


그림 1.3 솔루션 빌드 결과

오류 창이 나타나지 않으면 실행 파일이 만들어 진 것이다. 오류가 있다면 항목을 더블 클릭하여 오류가 발생한 곳으로 이동한다. 잘못된 부분을 수정한 후 다시 빌드한다.

◇ 빌드는 **컴파일**(compile)과 **링크**(link)의 두 단계로 나누어진다.

실행 파일은 앞에서 입력한 솔루션 위치의 하위 폴더에 저장된다.

```
...\Ch01\Hello World\Bin\Debug\Hello World.exe
```

를 찾아서 확인하자.

1.2.3 실행. 빌드로 만들어진 실행 파일을 실행하자. 메뉴에서

```
디버그(D) -> 디버깅 시작(S)
```

또는

```
디버그(D) -> 디버깅하지 않고 시작(H)
```

를 클릭한다. 출력되는 결과는

```
Hello World!
```

이다. 내 컴퓨터나 윈도우 탐색기에 실행 파일을 찾아 더블 클릭해도 같은 결과를 얻는다. 창이 너무 빨리 닫혀서 결과를 확인하기 어려울 것이다.

1.3 프로그램 구성

컴퓨터가 특정한 작업을 수행하도록 지시하는 명령 모음을 **컴퓨터 프로그램**(computer program)이라 한다. 단순히 프로그램이라고도 한다. 일반적으로 하드 디스크와 같은 저장 매체에 이진 파일(binary file)로 저장되어 있다가 메모리에 적재되고 중앙 처리 장치(central processing unit, CPU)에 의하여 실행된다.

초기 프로그램들은 중앙 처리 장치에 의하여 실행 가능한 기계의 언어로 이루어져 있었다. 그 코드는 이진수의 나열이었으며, 따라서 사람이 이해하기에는 어려움이 있었다. 그것을 극복하기 위하여 사람이 이해하기 쉬운, 곧 사람의 언어(단어와 기호)를 사용한 코드 체계가 개발되었다. 중앙 처리 장치는 사람의 언어를 이해할 수 없으므로 기계의 언어로 번역하는 번역기도 함께 개발해야만 했다. 현대의 프로그램은 대부분 사람의 언어로 작성되어 기계의 언어로 번역된다. 사람의 언어로 작성된 코드나 그것을 기계의 언어로 번역한 코드, 둘 다 프로그램이라 한다. 번역기는 컴파일러라 한다.

프로그래밍 언어(programming language)는 프로그램을 만들기 위한 언어이다. 사람의 언어로 이루어진 프로그래밍 언어는 번역기인 컴파일러가 이해하는 단어와 기호로 이루어져 있다. 프로그래밍 언어로 소스 코드(source code)를 작성한 다음 컴파일과 링크를 거쳐 실행 가능한 이진 파일로 변환한다.

프로그램을 구성하는 형식을 **문법**(grammar)이라 한다. 컴파일러가 이해하는 단어와 기호 그리고 그것들을 운용하는 규칙을 말한다. 프로그램은 사람이 사용하는 언어와 달리 문법을 엄격하게 지켜야만 한다. 조금의 오류도 인정되지 않는다.

1.3.1 문장. 완결된 실행 단위를 **문장**(statement)이라 한다. 문장에는 단문과 복문이 있다. 단문은 쌍반점(;)으로 끝나는 한 문장이다. 다음은 단문의 예이다.

```
int x = 1;
x++;
string str;
str = "C# 프로그램";
```

복문은 여러 문장이 중괄호({})로 묶여 있으며 주로 `if`나 `while` 같이 분기를 나타내는 키워드 뒤에 온다. 다음은 복문의 예이다.

```
if (z > 5) {
    z = z * 7;
    z -= 3;
}
```

내용이 없는 빈 문장도 가능하다. 단순히

```
;
```

으로 적으면 아무 일도 하지 않는 문장이 된다.

문장 안이나 문장 사이의 띄어쓰기는 칸의 수를 따지지 않고 한 칸으로 간주한다. 줄 바꿈과 탭도 띄어쓰기로 마찬가지이다. 다만 따옴표 사이의 띄어쓰기는 칸의 수를

따진다. 따라서 위의 두 코드는

```
int x = 1;    x++;    string
str; str = "C# 프로그램";
if (z > 5) {z = z * 7; z -= 3; }
```

과 같이 써도 된다.

1.3.2 주석. 코드를 이해하기 쉽도록 덧붙여주는 설명을 **주석**(comment)이라 한다. 띄어쓰기로 처리되어 실행되지 않는다. 주석을 삽입하는 방법은 두 가지이다. 첫째

```
// 이 줄은 주석이다
```

나

```
int x = 1; // 여기서부터 이 줄의 끝까지 주석이다.
```

처럼 //로 시작하여 그 줄의 끝까지 주석으로 지정하는 것이다. 둘째

```
int x = /* 주석 */ y;
```

나

```
/* 여기서부터 주석
   이 줄도 주석
   여기까지 주석 */
```

와 같이 /*로 시작하여 */로 끝내는 것이다.

1.3.3 구역. 0개 이상의 문장을 중괄호({})로 둘러싸서 만든 코드 부분을 구역(block)이라 한다. 문장이 없어도 되고 1개만 있어도 된다.

```
int z = 23;
if (z > 5) { // 구역 시작
    z = z * 7;
    z -= 3;
} // 구역 끝
```

위 코드를 종합하여 프로젝트를 만들어 보자. 솔루션 탐색기에서 [솔루션 Ch01]을 선택한다. 메뉴에서

파일(F) -> 추가(D) -> 새 프로젝트(N)

을 클릭한다. 창 [새 프로젝트 추가] 창에서 항목 [콘솔 앱(.NET Framework)]를 선택하고 버튼 [다음(N)]을 누른다. 창 [새 프로젝트 구성]에

프로젝트 이름(N): Statement Demo

를 입력하고 버튼 [만들기]를 누른다.

파일 Program.cs에 다음과 같이 입력한다.

예제 1.3.1 Statement Demo

```
using System;

class Program
{
    static void Main()
    {
        int x = 1;  // 여기서부터 이 줄의 끝까지 주석이다.
        x++;
        Console.WriteLine(x);

        // 이 줄은 주석이다.
        string a;
        a = "C# 프로그램";
        Console.WriteLine(a);

        /* 여기서부터 주석
           이 줄도 주석
           여기까지 주석 */

        int z = 23;
        if (z > 5) { // 구역 시작
            z = z * 7;
            z -= 3;
        } // 구역 끝
        Console.WriteLine(z);
    }
}
```

출력

```
2
C# 프로그램
158
```

창 [솔루션 탐색기]에서 프로젝트 Statement Demo를 오른 클릭하여

시작 프로젝트로 설정(A)

를 선택한 다음 빌드하고 실행하자.

◇ 예제 1.3.1에는

```
namespace Statement_Demo
```

가 없다. 이 코드를 제거해도 실행에는 문제가 없다.

1.3.4 실행 순서. 프로그램은 진입점으로부터 출발한다. C#의 진입점은 그 이름이 `Main()`이다. 프로젝트에는 `Main()`이 단 한 개만 정의되어야 한다. `Main()`의 첫째 줄부터 한 문장씩 실행되며 모든 문장이 실행되면 프로그램이 종료된다.

◇ 첫 번째 문장

```
class Program
```

은 `Main()`을 둘러싸고 있는 껍질과 같은 것으로 이해하자. 또, `static`과 `void` 역시 항상 `Main()`과 같이 붙어있는 단어 정도로 이해하면 된다.

1.3.5 키워드. 특정한 용도로만 사용하도록 제한한 단어를 키워드(keyword)라 한다. 정해진 용도로만 사용할 수 있으며 그 외에 다른 용도로는 사용할 수 없다. 예제 1.3.1에서 사용한 `int`, `if` 등이 그것이다. C#의 키워드는 다음과 같다.

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit	int	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	void
volatile	while			

1.4 변수와 상수

자료를 저장하기 위하여 필요한 크기만큼 메모리에 확보한 공간을 **변수**(variable)라 한다. 변수에는 대부분 이름을 붙인다. 변수를 만드는 것, 곧 메모리에 공간을 확보하고 이름을 붙이는 것을 **변수 선언**(variable declaration)이라 한다.

자료_형식 변수_이름

으로 선언한다. **자료_형식**은 자료의 형태와 저장 공간의 크기를 결정한다. 제 2 장에서 자세히 다룬다. **변수_이름**은 자료를 저장하거나 저장된 자료를 불러올 때 사용된다. 변수에 값을 저장하려면 등호를 사용한다.

변수_이름 = 값

값은 변수, 곧 메모리에 저장되는 자료를 나타낸다. 선언하면서 값을 설정하려면

자료_형식 변수_이름 = 값

으로 적는다.

자료 형식이 int이고 이름이 height인 변수는

```
int height;
```

로 선언한다. 이 변수에 14를 저장하려면

```
height = 14;
```

로 적는다. 둘을 결합하여

```
int height = 14;
```

로 적어도 된다. 변수를 선언하고 자료를 저장하는 두 문장을 줄여서 한 문장으로 적은 것이다.

변수 이름을 정할 때 다음 규칙을 지켜야 한다.

영문자, 한글, 숫자, 밑줄(_)로 구성할 수 있다.
특수 문자는 밑줄 이외에는 허용되지 않는다.
대문자와 소문자는 서로 다른 문자로 인식한다.
첫째 글자는 숫자로 쓸 수 없다.
키워드는 사용할 수 없다.

변수 이름을 정할 때 C#에서 권장하는 규칙이 있다.

변수 이름은 소문자로 시작한다.
한 단어로 이루어진 변수 이름은 모든 글자를 소문자로 쓴다.
두 단어 이상으로 이루어진 변수 이름에서 둘째 단어부터는 각 단어의 첫째 글자를 대문자로 쓴다.

이것을 **낙타 표기법** 또는 **카멜 표기법**(camel casing)이라 한다. 다음은 낙타 표기법으로 작성한 변수 이름이다.

```
name
myName
nameOfUncle
```

변하지 않는 값을 **상수**(constant)라 한다. 32, -12.67, "산은 높고"와 같이 어떤

상황에서도 변하지 않는 고정된 값을 말한다.

솔루션 Ch01에 다음 예제를 추가하자.

예제 1.4.1 Variable Demo

```
using System;

class Program
{
    static void Main(string[] args)
    {
        int height;
        height = 14;
        Console.WriteLine(height);

        int treeHeight = 5;
        Console.WriteLine(treeHeight);

        double weightOfApple = 0.2;
        string 과일 = "사과";
        Console.WriteLine(과일 + " 한 개의 무게는 " + weightOfApple +
                "kg이다");
    }
}
```

출력

```
14
5
사과 한 개의 무게는 0.2kg이다
```

예제 1.4.1의 int, double, string은 각각 정수, 실수, 문자열을 나타내는 자료 형식이다. height, treeHeight, weightOfApple, 과일은 변수이다. 각 변수에 값을 저장하고 저장된 값을 콘솔에 출력하였다.

◆ 연습문제 1.4.1. 이름이 네 단어로 이루어진 변수를 만드시오. 값을 저장하고 그 값을 출력하시오.

1.5 콘솔 출력

콘솔에 자료를 출력하려면

```
Console.Write(출력할_자료);
```

나

```
Console.WriteLine(출력할_자료);
```

와 같이 적는다. 출력할_자료에는 상수나 변수 또는 이들의 연산을 적는다. WriteLine()은 자료를 출력하고 줄을 바꾼다. Write()는 줄을 바꾸지 않는다.

예제 1.5.1 Console Output Demo

```
using System;

class Program
{
    static void Main(string[] args)
    {
        Console.Write(1.23);
        Console.WriteLine("문자열 출력");

        int x = 123;
        Console.WriteLine(x);
        Console.WriteLine(x + 1.23);
    }
}
```

```
출력
    1.23문자열 출력
    123
    124.23
```

1.6 명령줄 인수*

프로그램을 처음 접하는 독자라면 이 절을 이해하기 어려울 수도 있다. 천천히 읽어보고 이해하는 것은 뒤로 미루는 것이 좋을 것이다. 이 책을 학습하는 동안 필요하지도 않다.

진입점인 Main()은 string[] 형식 매개 변수를 갖는다.

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

args의 값은 프로그램을 실행할 때 사용자가 설정한다. 그 값을 **명령줄 인수**(command-line argument)라 한다.

예제를 실행하면서 자세히 알아본다.

예제 1.6.1 Argument Demo

```
using System;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine(args[0]);
        Console.WriteLine(args[1]);
    }
}
```

위 예제를 실행하면 오류 메시지가 출력될 것이다. args의 값이 설정되지 않았는데 출력하기 때문이다. 오류를 바로 잡아보자.

Visual Studio의 메뉴에서

디버그(D) -> Argument Demo 디버그 속성

를 클릭하면 실행하면 설정 창이 출력된다. 이 창에 '산은 높고'를 입력하고 저장한다(그림 1.4).

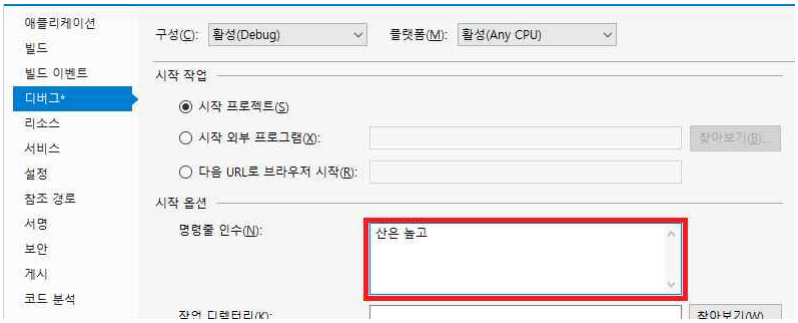


그림 1.4 명령줄 인수

이제 프로그램을 실행하면

```
산은
```

```
    높고
```

가 출력될 것이다. 그림 1.4에서는 두 명령줄 인수 '산은'과 '높고'를 설정한 것이다. 이 때 `args[0]`은 `"산은"`이고 `args[1]`은 `"높고"`이다. 명령줄 인수는 띄어쓰기로 분리된다.

명령줄 인수는 명령 프롬프트(cmd)에서 실행할 때 입력하도록 고안된 것이다(그림 1.5). 자세한 설명은 생략한다.

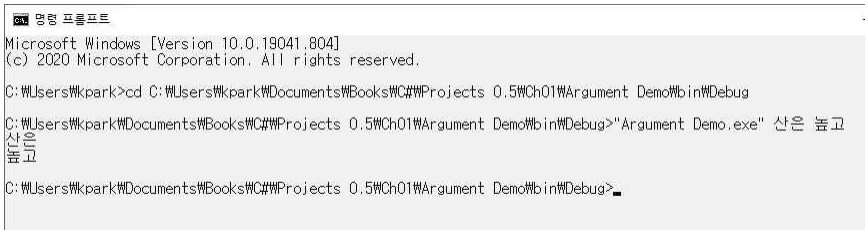


그림 1.5 명령 프롬프트에서 실행

제 2 장 자료 형식 I

프로그램에서 사용되는 자료는 기억 장치에 저장된다. 따라서 자료가 차지하는 공간의 크기를 정해야 한다. 또, 자료를 가공하려면 그것이 어떤 형태인지 알아야 한다. 이렇게 크기와 형태로 분류된 각 항목을 자료 형식이라 한다.

정수나 실수는 저장 공간의 크기에 따라 여러 자료 형식으로 나뉜다. 각 자료 형식에 정해진 크기만큼 저장 공간을 확보하고 값을 저장한다. 한 개 이상의 자료를 묶고 구조를 부여하여 새 자료 형식을 만들 수도 있다. 여러 자료를 묶어 한 자료처럼 다루므로 많은 자료를 처리할 때 편리하다. 또, 코드가 단순해지고 이해하기 쉬워진다.

이 장에서는 C#이 제공하는 다양한 자료 형식에 대하여 알아본다.

2.1 기본 형식

자료는 그 형태와 저장 공간의 크기에 따라 분류된다. 분류된 각 항목을 **자료 형식**(data type)이라 한다. 자료 형식은 프로그래밍 언어가 제공하는 것도 있고 사용자가 직접 설계하여 만드는 것도 있다. 언어가 제공하는 자료 형식 중에서 기본적인 것을 **기본 형식**(built-in data type)이라 한다. 부울, 문자, 정수, 실수, 문자열을 표현하는 자료 형식이 여기에 속한다.

◇ 마이크로소프트에서는 기본 형식 대신 기본 제공 형식이라 한다.

2.1.1 문자. **문자**(character)는 한 글자를 말한다. 프로그램에서는 '가' 또는 'A'와 같이 두 작은따옴표 사이에 적어서 표현한다. 문자를 저장할 변수는 키워드 char를 써서 선언한다.

```
char ch;
```

는 메모리에 문자 저장 공간을 확보하고 그 이름을 ch로 정한다. char 형식 변수라 불리며 문자의 유니코드를 저장한다. 이 변수에 값을 저장하려면 등호(=)를 사용한다.

```
ch = '가';
```

변수를 선언하면서 값을 저장할 수 있다.

```
char ch = '가';
```

와 같은 형식이다.

예제 2.1.1 Char Demo

```
using System;

class Program
{
    static void Main()
    {
        char ch = '가';
        Console.WriteLine(ch);
    }
}
```

출력

```
가
```

◆ 연습문제 2.1.1. 예제 2.1.1에 char 형식 변수를 추가하시오. 'A'를 저장하시오.

◇ 변수는 저장 공간이고 변수의 값은 그곳에 저장된 자료이다. 따라서 변수와 값은 구분되어야 한다. 그러나 이해하기 쉽고 단순한 용어를 사용하기 위하여 이 둘을 구분하지 않기로 한다. 예를 들어

```
char ch = 'A';
```

로 선언된 변수 ch를 '문자 ch'라 부르기도 한다. 또, 변수와 값을 동일시하여 'ch는 'A'이

다'와 같은 표현도 사용한다.

◇ 명령줄 인수가 필요 없다면 예제 2.1.1의

```
static void Main()
```

과 같이 Main()의 괄호 안을 비워도 된다.

2.1.2 문자열. 문자를 한 줄로 나열한 것을 **문자열**(string)이라 한다. 따옴표 안에 문자들을 적어서 표현한다. 문자열을 저장하는 변수는 키워드 string을 써서 선언한다.

```
string str = "이것이 문자열";
```

이 때 str을 string 형식 변수라 한다.

두 문자열을 이어 붙이려면 +를 사용한다.

```
string str1 = "산은 높";
string str2 = str1 + "고 물은 깊어";
```

str2는 두 문자열을 이어 붙여 만든 문자열 "산은 높고 물은 깊어"이다. 문자열에 문자를 이어 붙일 수 있다.

```
string str3 = "산은 높" + '고';
```

str3는 "산은 높고"가 된다.

예제 2.1.2 String Demo

```
using System;

class Program
{
    static void Main()
    {
        string str1 = "산은 높고";
        string str2 = "물은 깊어";
        Console.WriteLine(str1 + ' ' + str2);
    }
}
```

출력

```
산은 높고 물은 깊어
```

◆ 연습문제 2.1.2. 코드

```
char c = ' ';
string str = "산은 높고" + c + "물은 깊어";
```

가 실행된 후 str의 값은 무엇인가?

2.1.3 정수. 정수는 저장 공간의 크기에 따라 여러 자료 형식으로 나뉜다. 가장 자주 쓰는 형식은 키워드 int를 써서 표현하는 것이다.

예제 2.1.3 Int Demo

```
using System;

class Program
{
    static void Main()
    {
        int x = 12;
        int y = x;
        Console.WriteLine("x = " + x + ", y = " + y);
    }
}
```

출력

```
x = 12, y = 12
```

문자열에 int 형식을 이어 붙일 수 있다.

```
string s = "x: " + x;
```

를 실행하면 문자열 s의 값은 "x: 12"이다.

◆ 연습문제 2.1.3. 코드

```
int age = 21;
Console.WriteLine("내 나이는 " + age);
```

가 출력하는 것은 무엇인가?

◆ 연습문제 2.1.4. 값을 대입하는 기호 =는 어떤 역할을 할까? y = x는 무엇을 대입한다는 것일까? 코드

```
int x = 12;
int y = x;
x = 10;
Console.WriteLine(y);
```

는 어떤 결과를 출력할까? 추측해보고 난 후 실행해 보자.

◇ 십진수는 0부터 9까지 기호 열 개로 수를 표현한다. 각 기호들은 10보다 작은 정수를 나타낸다. 십육진수는 기호 열여섯 개가 필요하다. 0부터 9까지는 그대로 사용하고 10부터 15까지는 A부터 F까지 차례로 하나씩 대응시켜 표현한다. 프로그램에서 십육진수는 숫자 앞에 0x를 붙여 적는다. 0xC는 12이고 0x10은 16이다. 팔진수는 숫자 앞에 0을 붙여 적는다. 따라서 011은 9이다.

◆ 연습문제 2.1.5. 문장

```
int x = 0x000012AB;
```

를 실행하였을 때 x에 저장되는 값은 무엇인가?

2.1.4 실수. 실수도 저장 공간의 크기와 저장 방법에 따라 여러 자료 형식으로 나뉜다. 자주 쓰이는 방법은 double 형식 변수를 사용하는 것이다.

예제 2.1.4 Double Demo

```
using System;

class Program
{
    static void Main()
    {
        double d = 12.3;
        double e = d + 1;
        Console.WriteLine("d = {0}, e = {1}", d, e);
    }
}
```

출력

```
d = 12.3, e = 13.3
```

정수나 실수의 부호를 바꾸려면 - 를 사용한다. 상수나 변수 앞에 이 기호를 적어 주면 부호가 바뀐다.

◆ 연습문제 2.1.6. 변수 f를 만들어 2.7을 저장하고 -f를 출력하시오.

예제 2.1.4에서는 Console.WriteLine()의 따옴표 안에 {0}, {1}을 사용하였다. {0}은 따옴표 뒤에 오는 변수 중 첫째 변수의 값을, {1}은 둘째 변수의 값을 그 자리에 출력하라는 의미이다. 따라서

```
Console.WriteLine("d = {0}, e = {1}", d, e);
```

는

```
Console.WriteLine("d = " + d + ", e = " + e);
```

와 유사한 결과를 출력한다.

◆ 연습문제 2.1.7. 다음 문장을 중괄호를 써서 출력하는 문장으로 바꾸시오.

```
Console.WriteLine("x는 " + x + "입니다.");
```

◆ 연습문제 2.1.8. 다음 문장을 중괄호를 써서 출력하는 문장으로 바꾸시오.

```
Console.WriteLine("x = " + x + "y = " + y + "z = " + z);
```

◆ 연습문제 2.1.9. 두 변수

```
double a = 0.2;
double b = -1.3;
```

의 값을 교환하시오. a에는 -1.3이 저장되고 b에는 0.2가 저장되어야 한다. 상수를 직접 대입하면 안 된다.

2.1.5 산술 연산. 덧셈이나 뺄셈과 같이 수를 계산하는 것을 **산술 연산**(arithmetic operation)이라 하고, 산술 연산을 수행하는 연산자를 **산술 연산자**(arithmetic operator)라 한다. 산술 연산자는 표 2.1에 주어져 있다.

연산자	의미	기타
+	덧셈	
-	뺄셈	
*	곱셈	
/	나눗셈	정수를 정수로 나누는 경우 소수점 이하를 버림
%	나머지	

표 2.1 산술 연산

int 형식 변수나 상수 또는 double 형식 변수나 상수에는 산술 연산을 적용할 수 있다. 우리에게 익숙한 산술 연산은 나눗셈을 제외하고 그대로 적용된다. 산술 연산에서 두 항의 자료 형식이 같으면 결과도 같은 자료 형식이다. 곧, int 형식과 int 형식의 연산은 int 형식이며 double 형식과 double 형식의 연산은 double 형식이다. 따라서 정수를 정수로 나누면 그 결과가 정수이며 소수점 이하를 버린다.

◆ 연습문제 2.1.10. int 형식 변수를 int 형식 변수로 나눈 몫을 출력하는 코드를 작성하시오.

◆ 연습문제 2.1.11. int 형식 변수와 double 형식 변수의 산술 연산 결과는 어떤 자료 형식이 되는가?

◆ 연습문제 2.1.12. 두 변의 길이 width와 height로부터 직사각형의 넓이를 구하시오.

◆ 연습문제 2.1.13. 반지름 radius가 주어진 원의 넓이를 구하시오.

◆ 연습문제 2.1.14. 일정한 속도로 달리는 자동차가 있다. 달린 거리로부터 걸린 시간을 구하시오.

◆ 연습문제 2.1.15. 귤 m개를 n명이 똑같이 나누어 먹으려 한다. 나누어 먹고 남은 개수를 계산하시오.

특수한 경우에 연산을 줄여서 간단히 표현할 수 있다. 덧셈

```
x = x + 3;
```

에는 x가 두 번 나타난다. 이것을 줄여서

```
x += 3;
```

과 같이 적는다. 이것을 **연산자 축약**이라고 하고 +=를 **축약 연산자**라고 한다. 마찬가지 방법으로 -=, *=, /=, %=를 사용한다.

◆ 연습문제 2.1.16. 다음 코드는 무엇을 출력하는가?

```
int x = 12;
int y = 5;
x /= y;
Console.WriteLine(x);
```

◇ Microsoft의 Reference에서는 축약이란 용어를 사용하지 않는다. 의미가 약간 다르기 때문일 것이다.

정수 1을 더하는 것과 빼는 것은 더욱 간단하게 축약할 수 있다.

```
x = x + 1;
```

을 줄여서

```
x++;
```

또는

```
++x;
```

와 같이 적는다. 두 연산의 의미가 같지는 않지만 단독으로 사용될 경우에는 같은 것으로 간주해도 좋다. 마찬가지로 x--, --x도 x = x - 1 대신 사용한다.

◆ 연습문제 2.1.17. 코드

```
int x = 12;
x++;
Console.WriteLine(x);
```

가 출력하는 값은 무엇인가?

2.2 기본 형식 자세히 보기

지금까지 네 기본 형식 int, double, char, string에 대하여 알아보았다. 정수와 실수는 저장 공간의 크기에 따라 여러 자료 형식으로 나뉜다. 또, 참과 거짓을 표현하는 자료 형식도 있다. 기본 형식에 대하여 자세히 알아보자.

2.2.1 기본 형식 분류. 기본 형식은 부울, 정수, 실수, 문자열을 다양한 방식으로 표현한다. 각 범주에 속하는 자료 형식을 살펴보면 다음과 같다.

유형	자료 형식
부울	bool
정수	sbyte, byte, char, short, ushort, int, uint, long, ulong
실수	float, double, decimal
문자열	string

표 2.2 기본 형식 분류

bool 형식은 true(참)와 false(거짓)를 값으로 가진다.

```
bool b = true;
Console.WriteLine(b);
```

정수는 다양한 자료 형식으로 표현된다(표 2.3). char는 문자의 유니코드를 값으로 가진다. byte, ushort, uint, ulong은 0보다 크거나 같은 값을 가지며 sbyte, short, int, long은 양수와 음수를 모두 값으로 가진다. uint의 u는 unsigned를, sbyte의 s는 signed의 줄임말이다. 각 자료 형식은 메모리에서 차지하는 공간의 크기에 따라 값의 범위가 결정된다.

자료 형식	범위	크기
sbyte	$-2^7 \sim 2^7-1$	8비트
byte	$0 \sim 2^8-1$	8비트
char	$0 \sim 2^{16}-1$	16비트
short	$-2^{15} \sim 2^{15}-1$	16비트
ushort	$0 \sim 2^{16}-1$	16비트
int	$-2^{31} \sim 2^{31}-1$	32비트
uint	$0 \sim 2^{32}-1$	32비트
long	$-2^{63} \sim 2^{63}-1$	64비트
ulong	$0 \sim 2^{64}-1$	64비트

표 2.3 정수 자료 형식

백슬래시(\)로 시작하는 문자 표현을 **이스케이프 시퀀스**(escape sequence)라 한다. 따옴표와 같이 특수한 용도로 사용되는 문자를 나타내기 위하여 사용된다. 주요 이스케이프 시퀀스로는 \', \", \\, \t, \r, \n이 있으며 각각 작은따옴표, 따옴표, 백슬래

시, 탭, 캐리지 리턴(carriage return), 줄 바꿈을 나타낸다.

◆ 연습문제 2.2.1. 문장

```
Console.WriteLine("그가 \"안녕!\"이라고 말했다.");
```

가 출력하는 것은 무엇인가?

실수를 표현하는 자료 형식은 float, double, decimal이 있다(표 2.4).

자료 형식	범위	자리수	크기
float	±1.5e−45 ~ ±3.4e38	7자리	32비트
double	±5.0e−324 ~ ±1.7e308	15~16자리	64비트
decimal	-7.9e28 ~ 7.9e28	28~29자리	128비트

표 2.4 실수 자료 형식

기호 E나 e는 10의 거듭제곱을 나타낸다. 예를 들어 1.4E45와 1.4E-45는 각각 1.4×10^{45}과 1.4×10^{-45}을 나타낸다.

◆ 연습문제 2.2.2. 탄소(원소기호 C) 1g에 포함된 원자의 개수를 구하시오. 탄소의 표준 원자량은 12.01이고 아보가드로의 수는 6.02×10^{23}이다.

2.2.2 형식 변환. 자료 형식은 표현 범위가 제한되어 있다. 표 2.3, 2.4에서 보듯 그 범위는 자료 형식마다 다르다. 또, 같은 수를 저장하는 방법도 자료 형식마다 다르다. 예를 들어 1이라는 수는 int 형식, short 형식, double 형식으로 표현 가능하지만 저장 방법은 모두 다르다. 때때로 자료의 형식을 바꾸어 표현할 필요가 있다. int 형식을 double 형식으로 바꾸는 경우가 있으며 그 반대의 경우도 있다. 자료 형식을 바꾸는 것을 **형식 변환**(type casting)이라 한다.

우선 형식 변환과 관련된 문제점을 살펴보자. 코드

```
int x;
    ⋮
short s = x;
```

에서 int 형식인 x는 32 비트이고 short 형식인 s는 16 비트이다. int 형식이 표현할 수 있는 범위가 넓으므로 x의 값이 s에 그대로 대입될 수 없는 경우가 있을 것이다. double과 float에서도 유사한 문제가 발생한다. 연산과 관련된 문제도 있다. 다음은 정수를 정수로 나누어 실수를 구하는 코드이다.

```
int x, y;
    ⋮
double div = x / y;
```

그런데 정수를 정수로 나누면 소수점 이하를 잘라버린다. 일반적으로 올바른 결과를 기대할 수 없을 것이다.

형식 변환은 프로그래머가 일일이 명시하지 않아도 자동으로 이루어지는 때가 많다. 직관적으로 이해할 수 있는 경우가 그에 해당된다.

```
short s;
int x;
float f;
    ⋮
int y = s;
double d1 = x;
double d2 = f;
```

등과 같이 표현 범위가 좁은 형식에서 넓은 형식으로의 변환이 포함된다.

표현 범위가 넓은 형식에서 좁은 형식으로의 변환이나 나눗셈이 포함되는 경우,강제로 형식 변환을 하려면

(자료_형식) 변수

또는

(자료_형식) 상수

로 적는다. 위에서 문제점으로 지적한 코드는

```
short s = (short) x;
double div = (double) x / y;
```

로 고쳐 적으면 된다.

◇ (short) x와 같이 자료 형식을 밝혀 적는 형식 변환을 **명시적 형식 변환**(explicit type conversion)이라 한다. 또, 자료 형식을 명시하지 않는 형식 변환을 **암시적 형식 변환**(implicit type conversion)이라 한다. 그와 같이 표현하는 경우도 있지만 거의 사용하지 않는다. 기억할 필요는 없을 것이다.

double 형식을 float 형식으로 변환하려면 변수나 상수 앞에 (float)을 적는다. 다만 실수 상수를 float 형식으로 변환할 때는

```
float f= 1.1f;
```

와 같이 간단히 숫자 뒤에 f를 붙여 주면 된다. 실수 상수를 decimal 형식으로 변환하려면

```
decimal d = 1.1m;
```

와 같이 숫자 뒤에 m을 붙여 준다.

예제 2.2.1 Type Conversion Demo

```
using System;

class Program
{
    static void Main()
    {
        int x = 21;
        short s = (short)x; // 명시적 형식 변환
        x = s; // 암시적 형식 변환
        Console.WriteLine(x);
```

```
            x = 40000; // short의 범위를 벗어난다
            s = (short)x;
            Console.WriteLine(s); // x 값과 다르다

            float a = (float)1.1; // 소수점이 있는 상수는 double 형식이다
            a = 1.1f; // 숫자 뒤에 f를 써도 된다.
            Console.WriteLine(a);

            decimal d = 1.1m;
            Console.WriteLine(d);
        }
    }
```

```
출력
    21
    -25536
    1.1
    1.1
```

◆ 연습문제 2.2.3. a를 int 형식 변수에 대입하는 문장을 예제 2.2.1에 추가하시오.

◆ 연습문제 2.2.4. 예제 2.2.1에서 변수 s의 값이 -25536이다. 그 이유는 무엇인가? 이 문제와 관련된 내용은 이 책에서 언급하지 않는다. 검색하여 해결하기 바란다.

◆ 연습문제 2.2.5. 세 문장

```
Console.WriteLine("" + 'A');
Console.WriteLine("" + (int) 'A');
Console.WriteLine("" + (char) 65);
```

가 출력하는 것은 각각 무엇인가? 이유는 무엇인가?

◆ 연습문제 2.2.6. 다음 코드는 평균을 구한 것이다.

```
int numOfData = 3;
int a = 1, b = 1, c = 2;
Console.WriteLine((a + b + c) / numOfData);
```

실행하면 평균이 출력되는가? 잘못된 점은 무엇인가? 마지막 문장에 형식 변환을 표시하여 바른 값을 출력하시오.

2.2.3 문자열 변환. string.Format()은 부울, 정수, 실수를 string 형식으로 변환한다. 이 함수의 형식은 Console.WriteLine()과 같다(2.1.4).

```
bool b = false;
int x = 957;
double d = 12.87;
string str = string.Format("b={0}, x={1}, d={2}", b, x, d);
```

연산 +를 사용하여

```
string str = "b=" + b + ", x=" + x + ", d=" + d;
```

와 같이 쓸 수도 있다.

◆ 연습문제 2.2.7. 코드

```
string s = string.Format("{0}, {1}", 1.1f, 2.2m);
```

을 실행하여 얻는 s의 값은 무엇인가?

◇ string은 String의 별칭으로 둘은 같은 것으로 생각해도 된다.

string 형식을 각 자료 형식으로 변환하는 함수는 Parse()이다. 자료 형식 뒤에 Parse()를 붙여 적어 해당 자료 형식으로 변환할 수 있다. 예를 들어

```
int.Parse("3498");
double.Parse("3.498");
bool.Parse("false");
```

는 문자열을 각각 int 형식, double 형식, bool 형식으로 변환한다.

예제 2.2.2 String Conversion Demo

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("부울 입력: ");
        string line = Console.ReadLine();
        bool b = bool.Parse(line);
        Console.WriteLine("입력된 값: {0}", b);

        Console.Write("정수 입력: ");
        int n = int.Parse(Console.ReadLine());
        Console.WriteLine("입력된 값: {0}", n);

        Console.Write("실수 입력: ");
        Console.WriteLine("입력된 값: " + double.Parse(Console.ReadLine()));
    }
}
```

```
입력
    false
    239
    745.92
출력
    부울 입력:
    입력된 값: False
    정수 입력:
    입력된 값: 239
    실수 입력:
    입력된 값: 745.92
```

키보드로부터 입력을 받으려면 Console.ReadLine()을 사용한다. 이 함수는 키보드로 들어오는 문자열 한 줄을 읽는다. Console.Write()는 Console.WriteLine()과 마찬가지로 콘솔에 값을 출력한다. 그러나 출력한 후 줄을 바꾸지 않는다.

◆ 연습문제 2.2.8. decimal 형식을 읽어서 출력하시오.

◆ 연습문제 2.2.9. 두 값을 입력하여 그 평균을 출력하시오.

2.2.4 변수 적용 범위. 변수는 그것이 정의된 구역에서만 사용할 수 있다. 구역이 다르면 이름이 같은 변수라도 다른 것으로 취급한다.

```
static void Main()
{
    {
        int x = 11;
        Console.WriteLine(x);
    }
    {
        int x = -12;
        Console.WriteLine(x);
    }
}
```

두 구역에서 각각 선언된 x는 서로 다른 변수이다.

상위 구역에서 선언된 변수는 하위 구역에서 사용할 수 있다.

```
{
    // 상위 구역
    int x;
    {
        // 하위 구역
        x = 13;
    }
    // 상위 구역
    Console.WriteLine(x);
}
```

그러나 하위 구역에서 선언된 변수를 상위 구역에서 사용할 수는 없다.

◆ 연습문제 2.2.10. 다음 코드의 오류를 모두 찾으시오.

```
{
    int x = 13;
    int y = -9;
}
Console.WriteLine(x);
{
    Console.WriteLine(y);
}
```

2.3 배열

학생 700명의 수학 성적을 저장하려면 변수 700개가 필요하다. 이 변수들을 선언하고 사용하는 작업이 가능할까? 이론적으로는 가능하다하더라도 실제로는 거의 불가능한 일이다.

많은 변수들을 한꺼번에 다루기 위하여 고안된 것이 배열이다. 자료 형식이 같은 변수들을 이어서 모아 놓은 것을 **배열**(array)이라 한다. 배열에 포함되는 변수들을 **요소**(element)라 한다. 각 변수에 저장된 값도 요소라 부른다. 배열은

요소_자료_형식[] 배열_이름

과 같은 형식으로 선언한다. 요소_자료_형식은 배열의 요소, 곧 변수의 자료 형식이고 배열_이름은 배열을 나타내는 변수 이름이다.

◇ 한 배열에 포함된 변수들은 메모리에 연속적으로 위치한다.

요소의 자료 형식이 int인 배열 row는

```
int[] row;
```

로 선언한다. row의 자료 형식은 int[]이다. 'int의 배열'이라 읽으면 좋을 것이다. 배열을 생성하려면 키워드 new를 사용하며 요소의 개수를 명시한다.

```
row = new int[3];
```

세 요소를 가지는 배열을 생성하고 각 요소의 값을 0으로 설정한다. 두 문장을 통합하여

```
int[] row = new int[3];
```

으로 쓸 수 있다.

배열의 요소는 변수 뒤에 대괄호([])를 붙이고 그 안에 인덱스(index)를 적어서 가져온다. 인덱스로는 0부터 배열의 크기보다 1만큼 작은 정수까지 적을 수 있다. 배열 row의 크기가 3이므로 요소는 row[0], row[1], row[2]이다.

```
row[0] = 7;
Console.WriteLine(row[0]);
```

은 row의 첫째 요소의 값을 7로 설정하고 그 요소를 출력한다.

예제 2.3.1 Array Demo1

```
using System;

class Program
{
    static void Main()
    {
        double[] values = new double[4];
        values[0] = -1.3;
        values[1] = 3.2;
        values[2] = -1.5;
        values[3] = 2.7;
```

```
            Console.WriteLine(values[0] + values[1] + values[2] + values[3]);
        }
    }
```

출력

```
    3.1
```

◆ 연습문제 2.3.1. float 형식 변수 열 개를 요소로 갖는 배열을 생성하시오. 마지막 요소를 33.3f로 설정하시오.

배열을 선언하는 동시에 각 요소의 값을 설정할 수 있다.

```
int[] arr = new int[] { -9, 7, 8, -8 };
```

또는

```
int[] arr = { -9, 7, 8, -8 };
```

로 적는다. arr의 첫째 요소는 -9, 둘째 요소는 7이다. 배열의 크기는 요소의 개수에 의하여 결정된다. arr은 네 요소를 가지므로 크기는 4이다.

예제 2.3.2 Array Demo2

```
using System;

class Program
{
    static void Main()
    {
        int[] arr = new int[] { -9, 7, 8, -8 };
        int sum = arr[0] + arr[1] + arr[2] + arr[3];
        double mean = sum / (double)arr.Length;
        Console.WriteLine(mean);
    }
}
```

출력

```
    -0.5
```

배열의 크기는 위 예제의 arr.Length와 같이 배열 이름 뒤에 점을 찍고 Length를 붙여 적어 가져온다. int 형식이다.

◆ 연습문제 2.3.2. arr의 요소들의 분산을 출력하는 코드를 예제 2.3.2에 추가시오.

◆ 연습문제 2.3.3. 네 문자 'e', 'a', 's', 'y'를 요소로 갖는 배열을 만드시오.

◆ 연습문제 2.3.4. 다음은 한 라면 회사의 월별 판매량이다.

64, 65, 64, 63, 60, 59, 59, 60, 62, 61, 62, 63

이 자료를 저장하는 배열을 만드시오.

제 3 장 흐름 제어

지금까지 작성한 프로그램은 Main()의 첫 문장부터 순서대로 실행된다. 마지막 문장까지 실행되고 난 후 종료된다. 그러나 일이 항상 그렇게 간단하게 끝나지는 않는다. 상황에 따라 할 일이 추가되는 경우가 있으며 두 코드 중 하나를 선택하는 경우도 있다. 한 구역을 반복하여 실행할 수도 있다. 또 코드의 일부분을 함수로 독립시켜 호출할 수도 있다.

이 장에서는 분기, 반복, 함수를 사용하여 프로그램의 흐름을 바꾸는 방법을 알아본다.

3.1 분기

외출을 하려고 옷을 입는다고 하자. 기온이 높으면 얇은 옷을, 낮으면 두꺼운 옷을 선택할 것이다. 기온에 따라 그에 맞는 옷을 선택한다. 프로그램에서도 이와 같은 일이 빈번하게 발생한다. 조건에 따라 그에 해당되는 작업이 실행되는 것이다. 이것을 **분기**라 한다.

3.1.1 if. 조건의 참•거짓을 판별하여 뒤이어 실행할 문장을 결정할 때 키워드 if를 사용한다.

```
if (조건)
    선택
```

괄호 안의 조건이 참이면 선택을 실행하고 다음으로 진행한다. 거짓이면 실행하지 않고 다음으로 진행한다. if (조건)은 '조건이 만족되면' 또는 '조건이 참이면'이라는 뜻으로 해석한다. 괄호 안에는 bool 형식 변수나 상수 또는 연산이 대입된다. 3.1.2에서 자세히 살펴본다. 'input이 5보다 작으면'을 if를 써서 표현하면

```
if (input < 5)
```

이다.

다음 예제는 정수를 입력하여 5보다 작으면 2를 곱하여 출력하고 그렇지 않으면 그대로 출력하는 것이다.

예제 3.1.1 If Demo1

```
using System;

class Program
{
    static void Main()
    {
        int input = int.Parse(Console.ReadLine());
        if (input < 5)
            input *= 2;
        Console.WriteLine(input);
    }
}
```

```
입력
    3
출력
    6
```

◆ 연습문제 3.1.1. 실수 x를 읽으시오. x가 양수이면 부호를 바꾸시오. x를 출력하시오.

조건이 만족될 때 실행할 문장이 둘 이상이면

```
if (input < 5) {
    input *= 2;
    input++;
}
```

와 같이 문장들을 중괄호({})로 묶어 구역을 만든다. 조건이 참이면 이 구역 전체를 실행하고 그렇지 않으면 건너뛴다.

예제 3.1.1의 코드

```
if (input < 5)
```

와 같이 if () 다음에는 쌍반점(;)을 붙이지 않는다. 문장이 아니기 때문이다.

3.1.2 관계 연산. 연산자 <는 두 항을 비교한다. 앞 항이 뒷 항보다 작으면 결과가 참이고 그렇지 않으면 거짓이다. 예를 들어 2 < 3은 참이고 4 < 3은 거짓이다. 이와 같이 두 항의 크기를 비교하는 연산을 **관계 연산**(relational operation)이라 한다. 관계 연산을 수행하는 연산자를 **관계 연산자**(relational operator)라 한다. 관계 연산자는 표 3.1과 같이 주어진다.

연산자	연산 결과
==	두 항이 같으면 참
!=	두 항이 다르면 참
<	앞 항이 뒷 항보다 작으면 참
<=	앞 항이 뒷 항보다 작거나 같으면 참
>	앞 항이 뒷 항보다 크면 참
>=	앞 항이 뒷 항보다 크거나 같으면 참

표 3.1 관계 연산자

◇ 두 항이 같은지 비교하는 연산자는 등호 둘을 이어 붙인 ==이다. 값을 대입하는 등호와 다르니 주의하기 바란다.

◆ 연습문제 3.1.2. 두 정수를 읽으시오. 첫째 정수가 둘째 정수보다 크면 두 정수를 모두 출력하시오.

◆ 연습문제 3.1.3. string 형식 변수 sign을 "음수 아님"으로 정의하시오. 실수 num을 읽으시오. num이 0보다 작으면, sign을 "음수"로 설정하고 num을 0으로 바꾸시오. 두 변수 sign과 num을 출력하시오.

관계 연산의 결과는 bool 형식이다. 연산 결과가 참이면 그 값이 true이고 거짓이면 false이다. 3.1.1에서 if의 괄호 안에 bool 형식이 대입된다고 하였으므로 당연히 그럴 것이다.

```
bool b = 1 < 3; // true
if (b)
```

는

```
if (1 < 3)
```

과 동일하다.

문자는 유니코드로 변환하여 크기를 비교한다. 따라서 'a'는 'b'보다 작으며 대문

자는 소문자보다 작다. 다음은 모두 true이다.

```
'A' < 'B'
'Z' < 'a'
'a' < 'b'
65 <= 'A'  // 'A'의 유니코드는 65
'z' == 122 // 'z'의 유니코드는 122
```

다음 예제에서는 입력한 첫 문자가 아스키 문자인지 판별한다.

예제 3.1.2 If Demo2

```
using System;

class Program
{
    static void Main()
    {
        string str = Console.ReadLine();
        if (str[0] <= 127)
            Console.WriteLine("아스키 문자 " + str[0]);
    }
}
```

```
입력
    P
출력
    아스키 문자 P
```

문자열에서 한 문자를 가져오려면 대괄호([])를 붙이고 그 안에 인덱스를 적는다. 첫째 문자는 0, 둘째 문자는 1과 같이 뒤로 갈수록 1씩 증가한다. 배열의 인덱스와 같은 방식이다(2.3).

◆ 연습문제 3.1.4. 콘솔에서 한 줄을 읽어 첫 문자가 'A'이면 "OK"를 출력하시오.

◆ 연습문제 3.1.5. 입력된 문자열이 자신의 이름이면 나이를 출력하시오. 두 문자열이 같은지 알아보려면 함수 Equals()를 사용한다.

3.1.3 else. 조건에 따라 둘 중 하나를 선택하는 것도 가능하다. 이 경우에는 if와 else를 사용하며

```
if (조건)
    선택_1
else
    선택_2
```

와 같은 형식으로 적는다. 조건이 참이면 선택_1을 실행하고 거짓이면 선택_2를 실행한다. if ()와 마찬가지로 else 다음에도 세미콜론을 붙이지 않는다.

input이 짝수이면 문자열 "짝수"를 출력하고 그렇지 않으면 "홀수"를 출력하는 코드는 다음과 같다.

```
if (input % 2 == 0)
```

```
    Console.WriteLine("짝수");
else
    Console.WriteLine("홀수");
```

다음 예제는 사용자가 입력한 값을 조건에 따라 변형하여 출력한다.

예제 3.1.3 If Else Demo1

```
using System;

class Program
{
    static void Main()
    {
        int input = int.Parse(Console.ReadLine());
        int output;
        if (input < 5)
        {
            Console.WriteLine("5보다 작은 수");
            output = input * 2;
        }
        else
        {
            Console.WriteLine("5보다 크거나 같은 수");
            output = input / 2;
        }
        Console.WriteLine(output);
    }
}
```

```
입력
    31
출력
    15
```

◆ 연습문제 3.1.6. 두 정수 x, y를 읽으시오. y가 영이 아니면 x / y를 출력하고 그렇지 않으면 "분모가 0이다"를 출력하시오.

◆ 연습문제 3.1.7. 두 실수 x, y를 읽으시오. 두 수를 비교하여 큰 수를 a, 작은 수를 b라 하시오. a와 b를 출력하시오.

◆ 연습문제 3.1.8. 경로 A는 100km/h로 20km, 110km/h로 30km를 달려 목적지에 도착한다. 경로 B는 120km/h로 20km, 100km/h로 30km를 달려 목적지에 도착한다. 시간이 적게 걸리는 경로는 어느 것인가?

조건에 따라 여러 갈래로 갈라지는 상황에서는 if와 else를 연속하여 사용한다.

```
if (조건_1)
    선택_1
else if (조건_2)
    선택_2
else
    선택_3
```

실수의 부호를 나타내는 함수 $\mathrm{sign}(x)$는 x가 양수이면 1, 영이면 0, 음수이면 -1로 정의된다.

$$\mathrm{sign}(x) = \begin{cases} 1, & (x > 0) \\ 0, & (x = 0) \\ -1, & (x < 0) \end{cases}$$

실수 x를 입력하여 $\mathrm{sign}(x)$를 출력하는 프로그램은 다음과 같다.

예제 3.1.4 If Else Demo2

```
using System;

class Program
{
    static void Main()
    {
        double x = double.Parse(Console.ReadLine());
        int y;
        if (x > 0)
            y = 1;
        else if (x == 0)
            y = 0;
        else
            y = -1;
        Console.WriteLine("signum({0}) = {1}", x, y);
    }
}
```

```
입력
    -0.1
출력
    sign(-0.1) = -1
```

◆ 연습문제 3.1.9. 실수 volume을 읽으시오. volume이 0보다 작으면 0으로 바꾸고, 10보다 크면 10으로 바꾸시오. volume을 출력하시오.

◆ 연습문제 3.1.10. 두 정수를 읽으시오. 한 수가 다른 수의 배수이면 "good", 그렇지 않으면 "bad"를 출력하시오.

◆ 연습문제 3.1.11. 입력된 문자열이 "원"이면 원의 반지름을 읽어 그 넓이를 출력하시오. "직사각형"이면 직사각형의 두 변의 길이를 읽어 그 넓이를 출력하시오. "원"도 아니고 "직사각형"도 아니면 "잘못된 입력"을 출력하시오. 문자열 비교는 연습문제 3.1.5를 참고하기 바란다.

조건을 만족할 때 if를 하나 더 사용하여 다른 조건을 판별하는 것도 가능하다. 예를 들어 예제 3.1.4의 변수 y의 값은

```
if (x >= 0) {
    if (x > 0)
        y = 1;
    else
        y = 0;
```

```
    } else
        y = -1;
```

로 얻을 수 있다.

◆ 연습문제 3.1.12. 동전을 한 개 던져 앞면이 나오면 동전을 한 개 더 던진다. 나온 앞면의 개수를 출력하시오. 동전의 앞면과 뒷면은 각각 F와 T로 입력한다.

◆ 연습문제 3.1.13. 점수가 90 이상이면 학점이 A, 80 이상이면 B, 70 이상이면 C, 60 이상이면 D, 60 미만이면 F이다. 점수를 입력하여 학점을 출력하시오. 어떤 값을 입력하여도 관계 연산이 세 번 이하 수행되도록 하시오.

3.1.4 논리 연산. true와 false로 계산하는 연산을 **논리 연산**(logical operation)이라 한다. 논리 연산을 수행하는 연산자를 **논리 연산자**(logical operator)라 한다. C#는 세 논리 연산자를 제공한다.

연산자	이름	연산 결과
&&	논리곱	양쪽의 조건이 모두 참일 때만 연산 결과가 참
||	논리합	양쪽의 조건 중 하나라도 참이면 연산 결과가 참
!	부정	오른쪽 조건이 거짓일 때만 연산 결과가 참

표 3.2 논리 연산자

다음 프로그램에서는 한 문장을 읽어 첫째 문자가 숫자인지 아닌지 판별한다.

예제 3.1.5 Logic Demo

```
using System;

class Program
{
    static void Main()
    {
        string line = Console.ReadLine();
        if ('0' <= line[0] && line[0] <= '9')
            Console.WriteLine("숫자 " + line[0]);
        else
            Console.WriteLine("숫자 아님");
    }
}
```

```
입력
    23
출력
    숫자 2
```

논리 연산자는 bool 형식 변수 또는 상수에 대하여 연산을 수행하며 그 결과도 bool 형식이다. 따라서

```
bool b1 = x < 0;
bool b2 = !b1 && x <= 10;
```

과 같은 연산이 가능하다.

◆ 연습문제 3.1.14. 입력된 값이 0보다 작거나 10보다 크면 "부적합"을 출력하고, 그렇지 않으면 "적합"을 출력하시오.

◆ 연습문제 3.1.15. 입력된 문자열이 "Q"도 아니고 "q"도 아니면 "잘못된 입력"을 출력하시오.

◆ 연습문제 3.1.16. 한 문자를 읽어 숫자, 대문자, 소문자, 기타로 구분하시오.

◆ 연습문제 3.1.17. 키워드 if와 else를 각각 한 번씩만 써서

```
if (x >= 0) {
    if (y >= 0)
        Console.WriteLine("같은 부호");
    else
        Console.WriteLine("다른 부호");
} else {
    if (y >= 0)
        Console.WriteLine("다른 부호");
    else
        Console.WriteLine("같은 부호");
}
```

와 같은 결과를 출력하는 코드를 작성하시오.

◆ 연습문제 3.1.18. 에스프레소의 가격이 10000 원 이하이거나 카푸치노 가격이 15000 원 이하이고, 좌석이 10 석 이상 20 석 미만인 커피 판매점을 찾으려 한다. 에스프레소 가격, 카푸치노 가격, 좌석 수를 읽어 적당한 장소인지 판별하시오.

◆ 연습문제 3.1.19. 삼각형의 세 변의 길이를 읽어 정삼각형인지 또는 이등변삼각형인지 판별하시오.

◆ 연습문제 3.1.20. 삼각형의 세 변의 길이를 읽어 예각 삼각형, 직각 삼각형, 둔각 삼각형 중 어느 것인지 판별하시오.

3.1.5 다중 분기. 발생할 수 있는 여러 경우 중 하나를 선택할 때 사용되는 키워드가 switch이다.

```
switch(식)
{
    case 값_1:
        선택_1
        break;
     ⋮
    case 값_n:
        선택_n
        break;
    default:
        선택_n+1
        break;
}
```

괄호 안의 식을 계산하여 해당되는 선택을 실행한다. 계산 결과가 값_1과 같으면

```
case 값_1:
```

다음에 오는 선택_1을 실행한다. 값_n과 같으면 선택_n을 실행한다. 해당되는 값이 없으면

```
default:
```

다음에 오는 선택_n+1을 실행한다.

```
break;
```

는 거기까지 실행하고 switch 구역을 탈출하라는 뜻이다.

예제 3.1.6 Switch Demo

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("방향을 입력하시오(0:←, 1:→, 2:↑, 3:↓): ");
        int num = int.Parse(Console.ReadLine());
        string dir;
        switch (num)
        {
            case 0:
                dir = "왼쪽";
                break;
            case 1:
                dir = "오른쪽";
                break;
            case 2:
                dir = "위쪽";
                break;
            default:
                dir = "아래쪽";
                break;
        }
        Console.WriteLine(dir + "으로 이동");
    }
}
```

```
입력
    1
출력
    방향을 입력하시오(0:←, 1:→, 2:↑, 3:↓):
    오른쪽으로 이동
```

◆ 연습문제 3.1.21. 입력한 정수를 7로 나눈 나머지에 따라 "일", "월", "화", "수", "목", "금", "토"를 출력하는 프로그램을 작성하시오.

C#에서는 여러 경우에 동일한 선택을 실행하는 형식을 허용한다.

```
switch(n)
{
```

```
        case 1:
        case 2:
            x = 2 * n;
            break;
        case 3:
            x = 3 * n;
            break;
    }
```

◆ 연습문제 3.1.22. 입력된 문자가 '토', '일'이면 "휴일", '월', '수', '금'이면 "주간", '화', '목'이면 "야간"을 출력하시오.

3.2 반복

크기 100인 정수의 배열에 저장된 값들을 모두 더한다고 하자. 어떤 방법이 있을까? 하나씩 차례로 더하는 방법이 있다. 권장할 수 없지만 지금까지 학습한 내용으로는 다른 방법이 없다. 이 절에서는 다음과 같은 알고리즘을 코드로 구현한다.

```
arr을 크기 100인 정수의 배열이라 하자
int s = 0, i = 0;
i가 100보다 작은 동안 다음을 반복한다
    s += arr[i];
    i++;
s를 출력한다
```

변수 i를 증가시키면서 s에 배열의 값을 차례로 더한다. 출력되는 값은 배열 요소 전체의 합일 것이다. 여기서 중요한 것은 같은 코드가 여러 번 반복되는 것이다.

같은 문장이나 구역을 여러 번 실행하는 것을 **반복**(iteration)이라 한다. 일반적으로 어떤 조건이 만족되는 동안 반복하여 실행하다가 조건이 만족되지 않으면 실행을 중단한다. 원하는 회수만큼 반복될 때까지 조건이 만족되도록 유지하는 것이 중요하다. 반복을 프로그램으로 구현한 것을 **루프**(loop)라 한다. 반복문(iteration) 또는 순환문이라 부르기도 한다. 순환하면서 문장을 실행하는 것처럼 보이기 때문에 이와 같이 부른다.

3.2.1 while 루프. 반복을 구현하는 키워드 중 하나가 while이다. while을 사용한 반복은

```
while (조건)
    반복_문장
```

으로 구성되며 이를 while **루프**(while-loop)라 한다. **조건**은 if에서 사용한 것과 마찬가지로 bool 형식이다. **반복_문장**은 여러 문장으로 이루어질 수도 있다. 두 개 이상이면 중괄호로 묶어서 구역을 만들어야 한다. 실행 순서를 살펴보자. 먼저 괄호 안의 **조건**이 참인지 거짓인지 판별한다. 거짓이면 루프를 종료하고 다음으로 이동한다. 참이면 **반복_문장**을 실행한다. 다시 **조건**을 판별하여 **반복_문장**을 실행하거나 다음으로 이동한다. 이 과정은 조건이 참인 동안 계속된다. while 루프가 종료되려면 **조건**이 거짓이어야 한다. 따라서 **반복_문장**에는 **조건**을 변화시키는 코드가 포함되는 것이 일반적이다.

1부터 10까지 자연수를 출력하는 while 루프는 다음과 같다.

```
int i = 1;
while (i <= 10) {
    Console.WriteLine(i);
    i++;
}
```

첫째 문장에 의하여 i가 1이 되므로 조건 i <= 10이 참이다. 따라서 두 문장

```
Console.WriteLine(i);
i++;
```

를 실행한다. i <= 10인 동안 이 문장들은 반복하여 실행되고 그 때마다 i는 1씩 증가한다. 변수 i가 11이 되면 i <= 10이 거짓이므로 루프를 중단하고 다음으로 이동한다.

다음 예제에서는 1부터 100까지 자연수의 합을 구한다.

예제 3.2.1 While Demo1

```
using System;

class Program
{
    static void Main()
    {
        int sum = 0;
        int i = 1;
        while (i <= 100)
        {
            sum += i;
            i++;
        }
        Console.WriteLine(sum);
    }
}
```

출력

```
5050
```

변수 i는 1부터 100까지 1씩 증가하고 차례로 sum에 더해진다. 결국 sum은 1부터 100까지 정수의 합이 된다.

◆ 연습문제 3.2.1. 구구단 7단을 출력하는 while 루프를 작성하시오.

◆ 연습문제 3.2.2. 1부터 100까지 홀수의 합을 구하는 while 루프를 작성하시오.

◆ 연습문제 3.2.3. 두 정수 m, n을 읽어 m부터 n까지 정수의 곱을 출력하시오.

◆ 연습문제 3.2.4. 두 정수 m, n을 읽어 m부터 n까지 3의 배수의 합을 출력하시오.

◆ 연습문제 3.2.5. 240의 약수를 큰 수부터 출력하시오.

while 루프는 그것을 반복할 조건이 거짓이 되면 종료된다. 따라서 다양한 형태의 반복을 구현할 수 있다. 특별히 사용자가 특정한 값을 입력하면 종료되도록 할 수 있다. 증가하는 값이 다음 예제는 입력된 문자열을 모아 한 줄로 출력한다. 입력된 문자열이 "quit"이면 프로그램을 종료한다.

예제 3.2.2 While Demo2

```
using System;

class Program
{
```

```
    static void Main()
    {
        string output = "";
        string input = Console.ReadLine();
        while (!input.Equals("quit"))
        {
            output += input + ' ';
            input = Console.ReadLine();
        }
        Console.WriteLine(output);
    }
}
```

```
입력
    산은 높고
    물은 깊어
    quit
출력
    산은 높고 물은 깊어
```

◆ 연습문제 3.2.6. 입력된 실수의 합을 계산하는 while 루프를 작성하시오. 음수가 입력되면 루프가 종료되도록 하시오.

◆ 연습문제 3.2.7. 1부터 n까지 자연수의 곱 $1 \cdot 2 \cdot \cdots \cdot n$이 백만을 넘는 가장 작은 n을 구하시오.

◆ 연습문제 3.2.8. 한 문장을 읽어 첫째 단어의 글자 수를 출력하시오.

3.2.2 do-while. 위에서 소개한 while 루프는 조건을 검사한 다음 반복 문장을 실행한다. 상황에 따라 반복 문장을 먼저 실행하고 조건을 검사하는 것이 편리한 경우도 있다. 이 때 사용하는 것이 do-while **루프**이다. do-while 루프는

```
do
    반복_문장
while (조건);
```

으로 구성한다. 반복 문장이 여럿이면 중괄호({})로 둘러싼다.

```
int x = 1;
do {
    Console.WriteLine(x);
    x++;
} while (x <= 10);
```

는 1부터 10까지 정수를 출력한다.

다음 예제는 콘솔에서 문자열 한 줄을 읽어 한 문자씩 출력한다. 문자가 '산'이면 출력을 멈추고 루프를 종료한다.

예제 3.2.3 Do-While Demo

```
using System;

class Program
```

```
{
    static void Main()
    {
        string line = Console.ReadLine();
        int i = 0;
        do
        {
            Console.Write(line[i]);
            i++;
        } while (line[i] != '산');
        Console.WriteLine();
    }
}
```

```
입력
    아름다운 강산
출력
    아름다운 강
```

◆ 연습문제 3.2.9. 연습문제 3.2.2를 do-while 루프로 해결하시오.

◆ 연습문제 3.2.10. 연습문제 3.2.7을 do-while 루프로 해결하시오.

3.2.3 for 루프. 반복을 구현하는 또 다른 방법은 키워드 for를 사용하는 것이다. for를 사용한 반복은

for (초기화; 조건; 변화)
반복_문장

형식이며 이것을 for **루프**(for loop)라 한다. 반복되는 문장이 둘 이상이면 문장들을 중괄호로 둘러싸서 구역을 만든다. 실행 순서를 살펴보자. 먼저 **초기화**를 실행한 후 **조건**의 참·거짓을 판별한다. **조건**이 거짓이면 루프를 종료하고 다음 문장으로 이동한다. 참이면 **반복_문장**을 실행한 후 **변화**를 실행한다. 다시 **조건**을 판별하여 **반복_문장**을 실행하거나 루프를 종료한다. 이 과정은 조건이 참인 동안 반복된다.

1부터 10까지 자연수를 출력하는 for 루프는 다음과 같다.

```
for (int i = 1; i <= 10; i++)
    Console.WriteLine(i);
```

먼저

```
int i = 1;
```

로 변수 i를 선언하고 1을 대입한다. 이어서 조건 i <= 10의 참·거짓을 판별한다. 이 조건이 참이므로 반복 문장

```
Console.WriteLine(i);
```

를 실행한다. 그리고 i++를 실행하여 i를 1만큼 증가시킨다. 여전히 조건이 참이므로 반복 문장을 실행한다. 이 과정이 계속되어 i가 11이 되면 조건은 거짓이 되고 루프가 종료된다.

다음 예제는 1부터 100까지 자연수의 합을 for 루프로 구한 것이다. 예제 3.2.1의

while 루프와 비교해 보기 바란다.

예제 3.2.4 For Demo

```
using System;

class Program
{
    static void Main()
    {
        int sum = 0;
        for (int i = 1; i <= 100; i++)
            sum += i;
        Console.WriteLine(sum);
    }
}
```

출력

```
5050
```

예제 3.2.4는 예제 3.2.1의 문장들을 위치만 바꾸어 놓은 것임을 알 수 있다. for 루프는 많은 경우 이와 같은 방법으로 얻을 수 있다.

◆ 연습문제 3.2.11. 연습문제 3.2.1 ~ 3.2.5를 for 루프로 해결하시오.

배열의 인덱스는 0부터 1씩 증가한다. 따라서 배열은 for 루프와 함께 사용되는 경우가 많다.

```
int[] arr = { 1, -7, 9, 4};
for(int i = 0; i < 4; i++)
    Console.WriteLine(arr[i]);
```

와 같이 인덱스를 증가시켜 요소의 값을 설정하거나 가져온다.

◆ 연습문제 3.2.12. 크기 5인 실수의 배열을 만드시오. 실수 5개를 읽어 배열을 채우시오. for 루프를 사용하시오.

◆ 연습문제 3.2.13. 영문자와 띄어쓰기로만 이루어진 문자열을 입력하여 대문자와 소문자가 각각 몇 개씩 포함되어 있는지 조사하시오. 입력된 문자열에서 문자를 하나씩 모두 가져오려면 그것의 길이를 알아야 한다. 이 때 함수 Length()를 사용한다. 예를 들어 문자열 str의 길이는 str.Length()로 가져온다.

연습문제 3.2.14 ~ 3.2.17은 while, do-while, for 루프 중 어느 것을 사용하든지 상관없다.

◆ 연습문제 3.2.14. 점화식

$$a_1 = 2,\ a_n = 2\,a_{n-1} - 1$$

으로 표현된 수열 a_n의 제 30 항 a_{30}을 구하시오. 배열을 사용하는 코드와 사용하지 않는 코드를 각각 작성하시오.

◆ 연습문제 3.2.15. 피보나치(fibonacci) 수열은 점화식

$$a_1 = 1, \quad a_2 = 1, \; a_n = a_{n-2} + a_{n-1}$$

으로 표현된다. a_{30}을 구하시오. 배열을 사용하는 코드와 사용하지 않는 코드를 각각 작성하시오.

◆ 연습문제 3.2.16. 연이율 4% 복리로 매년 120 만원씩 30 년간 예금하려한다. 30 년 후 원리금을 계산하시오.

◆ 연습문제 3.2.17. $2^{74207281} - 1$은 소수이다. 이 수의 마지막 다섯 자리를 구하시오.

3.2.4 foreach 루프. 배열의 요소나 문자열의 문자를 가져오는 반복을 구현할 때 foreach를 사용할 수 있다. **foreach 루프**는

```
foreach(요소_자료형 요소_변수 in 배열_변수)
    반복_문장
```

과 같은 형식이다. 요소_변수는 배열의 각 요소 값을 앞에서부터 하나씩 차례로 가져온다. double[] 형식 변수 scores에 대하여

```
foreach (double d in scores)
    Console.WriteLine(d);
```

는 scores의 모든 요소를 출력한다. 변수 d는 scores[0], scores[1], ...을 순서대로 하나씩 값으로 가진다.

다음 예제에서는 배열에 저장된 값의 평균과 분산을 출력한다.

예제 3.2.5 Foreach Demo

```
using System;

class Program
{
    static void Main()
    {
        double[] scores = { 78, 72.7, 81, 92, 88.2 };
        double sum = 0;
        double squareSum = 0;
        foreach (double d in scores)
        {
            sum += d;
            squareSum += d * d;
        }
        double mean = sum / scores.Length;
        double variance = squareSum / scores.Length - mean * mean;
        Console.WriteLine("평균: {0}, 분산: {1}", mean, variance);
    }
}
```

출력

```
평균: 82.38, 분산: 48.2416000000012
```

◆ 연습문제 3.2.18. 예제 3.2.5에 배열 scores의 요소 중 가장 큰 것을 구하는 코드를 추가하시오.

foreach 루프는 문자열의 각 문자를 앞에서부터 순서대로 얻을 때도 사용할 수 있다.

```
string str = "산은 높고";
foreach (char ch in str)
    Console.WriteLine(c);
```

에서 ch에는 str의 각 문자가 앞에서부터 순서대로 할당된다.

◆ 연습문제 3.2.19. 문자열 "더워지는 지구"에 포함된 각 문자의 유니코드를 순서대로 출력하는 foreach 루프를 작성하시오.

◇ 다음 코드를 보자.

```
double[] list = new double[10];
for (double el in list)
    el = 0.99;
```

배열 list의 모든 요소가 0.99일 것이라 기대를 해볼 만하다. 실제로는 어떨까? list의 각 요소를 출력하여 확인하기 바란다. 변수 el은 list의 요소를 가져오는 것이 아니고 요소의 값을 가져온다. 따라서 el을 변경하여도 요소의 값은 바뀌지 않는다.

3.2.5 반복 제어. 1부터 100까지 정수 중 5의 배수를 제외한 수의 합을 구하려면 어떻게 해야 할까? 다양한 해결 방법이 있을 것이다.

◆ 연습문제 3.2.20. 1부터 100까지 정수 중 5의 배수를 제외한 수를 모두 합하시오.

루프 중간에서 나머지 부분을 실행하지 않고 건너뛰려면 키워드 continue를 사용한다. 다음은 1부터 100까지 정수 중 5의 배수를 제외한 수를 합하는 코드이다.

```
int sum = 0;
for (int i = 1; i <= 100; i++)
{
    if (i % 5 == 0)
        continue;
    sum += i;
}
```

루프를 실행하다가 i가 5의 배수가 될 때마다

```
continue;
```

를 만나므로

```
sum += i;
```

를 건너뛴다. i가 5의 배수가 아니면 이 문장을 실행한다.

키워드 break는 아예 루프를 끝낸다.

```
int sum = 0;
int i = 1;
while(i < 1000)
{
    sum += i;
```

```
        if(sum > 100)
            break;
        i++;
    }
```

이 코드의 while 루프는 계속 실행되다가 sum이 100보다 크면 break를 만나서 종료된다.

예제 3.2.6 Continue Break Demo

```
using System;

class Program
{
    static void Main()
    {
        int i = 0;
        while (true)
        {
            i++;
            if (i % 3 == 0)
                continue;
            if (i == 10)
                break;
            Console.Write(i + " ");
        }
        Console.WriteLine();
    }
}
```

출력

```
1 2 4 5 7 8
```

while 루프의 조건

```
while (true)
```

는 루프가 무한히 반복될 것임을 말하고 있다. 따라서 적당한 종료 조건이 추가되어야만 한다. 변수 i가 10이면 종료되도록 break를 삽입하였다.

◆ 연습문제 3.2.21. 예제 3.2.6의 코드

```
if (i == 10)
```

을

```
if (i == 12)
```

로 바꾸면 어떤 문제가 발생하는가?

◆ 연습문제 3.2.22. 정수의 배열

```
int[] arr = { 1, -1, 21, 13, 5, -14, 7 };
```

의 요소 중 양수들의 평균과 분산을 구하시오. 키워드 continue를 사용하시오.

◆ 연습문제 3.2.23. 정수들을 입력하여 그 합을 계산하시오. 무한히 반복되는 while 루프를 작

성하고 0이 입력되면 키워드 break로 루프를 종료하시오. 계산한 합을 출력하시오.

3.2.6 루프 안에 루프 두기. 문자 '*'로 이루어진 4×4 격자

```
* * * *
* * * *
* * * *
* * * *
```

를 출력하려면 어떻게 해야 할까? 일반적으로 자연수 n에 대하여 $n\times n$인 경우도 고려하면서 프로그램을 작성해보자. 한 번에 한 줄씩 출력하는 루프를 작성하여 네 번 실행하면 4×4 격자가 출력된다.

```
for (int i = 1; i <= 4; i++) {
    // * * * * 출력
}
```

와 같은 형태가 될 것이다. 한 줄은 한 번에 출력할 수도 있지만, 일반적인 경우를 고려하면 루프를 사용하는 것이 좋다. 한 줄을 출력하는 코드는

```
for (int j = 1; j <= 4; j++)
    Console.Write("* ");
```

가 될 것이다. 결과적으로 격자는

```
for (int i = 1; i <= 4; i++) {
    for (int j = 1; j <= 4; j++)
        Console.Write("* ");
    Console.WriteLine();
}
```

으로 출력하게 된다.

다음 예제에서는 문자 '*'를 삼각형으로 배치한다. 각 줄의 길이가 다르므로 격자로 배치하는 것보다 어려울 수 밖에 없다.

예제 3.2.7 For-For Demo

```
using System;

class Program
{
    static void Main()
    {
        for (int i = 1; i <= 3; i++)
        {
            for (int j = 1; j <= i; j++)
                Console.Write("* ");
            Console.WriteLine();
        }
    }
}
```

출력

```
*
```

```
* *
* * *
```

◇ 예제 3.2.7의 for 루프는 while 루프로 바꿀 수 있다. 둘 중 하나만 바꿀 수도 있고 둘 다 바꿀 수도 있다.

◆ 연습문제 3.2.24. 다음을 출력하시오.

```
1
1 2
1 2 3
```

◆ 연습문제 3.2.25. 다음을 출력하시오.

```
  *
 * *
* * *
```

◆ 연습문제 3.2.26. 정수들의 합

$$1,\ 2+3,\ 4+5+6,\ \ldots,\ 46+\cdots+55$$

를 계산하시오.

3.3 함수

특정한 역할을 하는 코드를 독립시켜 놓은 것을 **함수**(function)라 한다. **메서드**(method)라고도 한다. 코드를 이해하기 쉽게 하거나 중복되는 부분을 한 번만 적을 때 사용할 수 있다. 함수는 자료를 받아들여 처리하거나 계산 결과를 되돌려 준다. 받아들이는 자료는 변수에 저장되며 그 변수를 **매개 변수**(parameter)라 한다. 매개 변수에 저장되는 값을 **인수**(argument)라 한다. 또, 인수를 처리하여 얻은 결과를 호출한 곳으로 전달하는 것을 **반환**(return)이라 하고 전달되는 값을 **반환 값**(return value)이라 한다.

◇ 함수 $y=f(x)$에서 x는 매개 변수이다. 이 매개 변수에 대입하는 값이 인수이다. $f(7)$에서 7이 인수이다. 또, 함숫값, 곧 계산의 결과 $f(7)$은 반환 값이다.

함수의 정의는 형식과 본문으로 구성된다. 형식은 한정자, 반환 값의 자료 형식, 함수 이름, 매개 변수로 구성된다. 본문에는 기능 곧, 하는 일을 적는다.

```
한정자 반환_형식 함수_이름(자료_형식_1, 매개_변수_1, ...) {
    함수_본문
}
```

한정자는 함수에 대한 제한 조건을 설정한다. 다양한 한정자가 존재하며 뒤에서 차차 다루게 된다. 반환_형식은 반환 값의 자료 형식이다. 함수 이름은 모든 단어의 첫 글자를 대문자로 쓸 것을 권장한다. 이것을 **파스칼 표기법**(pascal casing)이라 한다.

3.3.1 정의와 호출.

C#의 함수는 **정적 메서드**(static method)와 **인스턴스 메서드**(instance method)로 나뉜다. 인스턴스 메서드는 4.1에서 다루기로 하고 여기서는 정적 메서드에 대하여 알아보자.

실수에 1.0을 더하여 반환하는 함수는 다음과 같다.

```
static double Foo(double x) {
    return x + 1.0;
}
```

static은 정적 메서드를 나타내는 한정자이다. 매개 변수 x는 double 형식이고 반환 값도 double 형식이다. 반환하는 값은 키워드 return 다음에 적는다.

```
return 식;
```

함수를 사용하는 것을 **호출**(call)이라 한다. 함수 Foo()는

```
double y = Foo(32.0);
```

와 같이 괄호 안에 인수를 적어서 호출한다. 이 문장에서 함수 Foo()의 인수는 32.0이다.

예제 3.3.1 Method Demo1

```
using System;
```

```
class Program
{
    static double AddOne(double x)
    {
        return x + 1;
    }
    static void Main()
    {
        double a = 11.2;
        double b = AddOne(a);
        Console.WriteLine(b);
    }
}
```

출력

```
12.2
```

함수 AddOne()은 매개 변수에 1을 더한 값을 반환한다. 따라서 b는 11.2에 1을 더한 값, 곧 12.2이다.

◆ 연습문제 3.3.1. 원의 넓이를 구하는 함수를 작성하시오. 반지름을 매개 변수로 받아들여 넓이를 반환하도록 하시오.

◆ 연습문제 3.3.2. 한 문자를 매개 변수로 받아들여 그 문자가 숫자이면 true를 반환하고 그렇지 않으면 false를 반환하는 함수 IsNumeric()을 작성하시오.

◆ 연습문제 3.3.3. 섭씨를 화씨로 변환하는 함수를 작성하시오.

매개 변수가 둘 이상이면 반점(,)으로 분리한다. 매개 변수가 둘인 함수는

```
static int Bar(double x, double y)
```

와 같이 정의한다. 호출할 때도 두 인수를 대입한다.

```
int c = Bar(a, b);
```

다음 예제는 배열에서 특정한 값보다 큰 수의 개수를 출력한다.

예제 3.3.2 MethodDemo2

```
using System;

class Program
{
    static int CountOverBasis(int[] arr, int basis)
    {
        int count = 0;
        foreach (int d in arr)
        {
            if (d > basis)
                count++;
        }
        return count;
    }
    static void Main()
    {
```

```
            int[] values = { 8, 12, 9, 8, 11, 14, 13, 10 };
            Console.WriteLine(CountOverBasis(values, 10));
        }
    }
    출력
        4
```

함수 CountOverBasis()는 정수의 배열 arr과 정수 basis를 받아들여 arr의 요소 중 basis보다 큰 것을 세어 그 개수를 반환한다.

◆ 연습문제 3.3.4. 직사각형의 넓이를 구하는 함수를 작성하시오. 폭과 높이를 받아들여 넓이를 반환하도록 하시오.

◆ 연습문제 3.3.5. 두 실수 중 큰 수를 구하는 함수를 작성하시오.

◆ 연습문제 3.3.6. 세 실수 중 가장 큰 수를 구하는 함수를 작성하시오.

매개 변수가 없는 함수나 값을 반환하지 않는 함수도 정의할 수 있다. 매개 변수가 없는 함수는 선언할 때 매개 변수가 들어갈 부분에 아무것도 적지 않는다.

```
static int Foo() {
```

값을 반환하지 않는 함수는 반환 형식을 void로 설정한다.

```
static void Bar(double a) {
```

함수

```
static void Qux() {
    Console.WriteLine("매개 변수가 없고 값을 반환하지 않는 함수");
}
```

는 반환 값이 없으므로

```
Qux();
```

와 같이 단독으로 호출된다. 값을 반환하지 않는 함수에서도 return을 사용할 수 있다. 함수 실행 도중

```
return;
```

을 만나면 실행이 종료된다. 곧, 함수 실행이 종료되어 함수를 호출한 곳으로 돌아간다. 예를 들어

```
static void PrintIfPositive(int x) {
    if (x < 0)
        return;
    Console.WriteLine(x);
}
```

는 매개 변수 x가 음수이면 return을 만나 종료되고 그렇지 않으면 x를 출력한 후 종료된다.

◆ 연습문제 3.3.7. 문자열에 포함된 문자 중 숫자의 개수를 출력하는 함수를 작성하시오.

◆ 연습문제 3.3.8. 실수들을 읽어 합을 출력하는 함수를 작성하시오. 음수가 입력되면 합을 출

력하고 종료된다.

다음 예제에서는 50보다 작은 소수와 953663의 가장 큰 소인수를 구한다.

예제 3.3.3 MethodDemo3

```
using System;
class Program
{
    static bool IsPrime(int n)
    {
        if (n < 2)
            return false;
        if (n % 2 == 0)
            return n == 2;
        for (int i = 3; i * i <= n; i += 2)
            if (n % i == 0)
                return false;
        return true;
    }

    static void BiggestPrimeFactor(int n)
    {
        if (n < 2)
            return;
        int i = n;
        while (true)
        {
            if (n % i == 0 && IsPrime(i))
            {
                Console.WriteLine(i);
                return;
            }
            i--;
        }
    }

    static void Main()
    {
        for (int i = 1; i < 50; i++)
            if (IsPrime(i))
                Console.Write(i + " ");
        Console.WriteLine();
        BiggestPrimeFactor(953663);
    }
}
```

출력

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
4993
```

함수 IsPrime()은 정수를 매개 변수로 받아들여 그것이 소수이면 true, 그렇지 않으면 false를 반환한다. 함수 BiggestPrimeFactor()에서는 IsPrime()을 사용하여 가장 큰 소인수를 구한다.

◆ 연습문제 3.3.9. 정수 n에 대하여

$$1^2 - 2^2 + 3^2 - 4^2 + \cdots + (-1)^n n^2$$

을 반환하는 함수를 작성하시오.

◆ 연습문제 3.3.10. 각각 평균과 분산을 반환하는 두 함수를 작성하시오. 두 함수의 매개 변수는 실수의 배열이다.

◆ 연습문제 3.3.11. 문자열의 배열에 포함된 문자열을 모두 순서대로 이어 붙여 만든 문자열을 반환하는 함수를 작성하시오.

◆ 연습문제 3.3.12. 자연수를 받아들여 그 수의 약수 전체로 이루어진 배열을 반환하는 함수를 작성하시오.

◆ 연습문제 3.3.13. 두 수의 최소 공배수를 반환하는 함수를 작성하시오.

◆ 연습문제 3.3.14. 두 수의 최대 공약수를 반환하는 함수를 작성하시오.

◆ 연습문제 3.3.15. 배열에 포함된 요소를 크기순으로 정렬하는 함수를 작성하시오.

3.3.2 변수 적용 범위. 함수의 매개 변수와 함수에서 선언한 변수는 그 함수에서만 사용할 수 있다. 또, 함수 본문에서 매개 변수의 값을 바꾸어도 호출한 곳에서는 영향을 받지 않는다.

```
static void Foo(int x) {
    int a = 33;
    Console.WriteLine(a + x);
}

static void Bar(int x) {
    int a = 44;
    Console.WriteLine(a - x);
}
```

함수 Foo()에서 선언한 변수 a와 매개 변수 x는 함수 Foo()에서만 사용할 수 있다. 함수 Bar()에서도 마찬가지이다. 따라서 Foo()와 Bar()에서 각각 선언한 두 a는 이름만 같을 뿐 서로 다른 변수이다. 매개 변수에도 같은 규칙이 적용된다.

함수의 매개 변수로 전달되는 것은 변수가 아니라 그 변수에 저장된 값이다. 이와 같이 동작하는 함수 호출을 **값에 의한 호출**(call by value)이라 한다. C#은 값에 의한 호출만 허용한다.

```
static void Qux(int x) {
    x++;
}
```

에 대하여

```
int num = 11;
Qux(num);
```

을 실행한 후 num의 값은 12일까? 예상과 달리 num은 여전히 11이다. 함수의 매개 변수를 함수 본문에서 바꾸어도 호출한 부분에서는 바뀌지 않는다. 매개 변수 x는 num의 값을 복사하기 때문이다. 이것이 값에 의한 호출이다.

3.3.3 함수 오버로딩. 두 정수의 합을 반환하는 함수와 두 실수의 합을 반환하는 함수를 선언해 보자.

```
static int AddInt(int x, int y) {
    return x + y;
}

static double AddDouble(double x, double y) {
    return x + y;
}
```

두 함수는 모두 두 매개 변수를 합하여 반환한다. 만약 허용된다면 같은 이름으로 정의하는 것이 편리할 것이다. 실제로 그것이 허용된다.

```
int Add(int x, int y) {
    return x + y;
}

double Add(double x, double y) {
    return x + y;
}
```

함수를 호출할 때 대입되는 인수에 따라 둘 중 하나가 선택된다. 이름이 같은 함수를 여럿 정의하는 것을 **함수 오버로딩**(overloading)이라 한다. 오버로딩된 함수는 인수의 자료 형식에 따라 호출되는 함수가 달라진다.

예제 3.3.4 Overloading Demo

```
using System;

class Program
{

    static int Add(int x, int y)
    {
        Console.WriteLine("Add(int, int)");
        return x + y;
    }

    static double Add(double x, double y)
    {
        Console.WriteLine("Add(double, double)");
        return x + y;
    }

    static void Main()
    {
        Console.WriteLine(Add(1, 2));
        Console.WriteLine(Add(1.2, 2.3));
    }
}
```

출력

```
Add(int, int)
```

```
3
Add(double, double)
3.5
```

◆ 연습문제 3.3.16. 반지름으로부터 원의 넓이를 구하는 함수 Area()와 밑변과 높이로부터 직사각형의 넓이를 구하는 함수 Area()를 예제 3.3.4에 추가하시오.

함수 선언에서 함수 이름과 매개 변수의 자료 형식을 순서대로 나열한 것을 그 함수의 **시그니처**(signature)라 한다. 함수

```
int Foo(float x, double y);
```

의 시그니처는

```
Foo(float, double)
```

이다. 시그니처는 오버로딩된 함수를 구별할 때 사용된다.

◆ 연습문제 3.3.17. 예제 3.3.4에서 정의한 두 함수 Add()의 시그니처는 각각 무엇인가?

3.3.4 재귀 함수. 함수에서 함수 자신을 호출하는 것을 **재귀 호출**(recrusion)이라 하고 그 함수를 **재귀 함수**(recursive function)라 한다.

1부터 n까지 정수의 합을 계산하는 함수 sum()을 재귀 함수

```
static int Sum(int n) {
    if(n == 1)
        return 1;
    return Sum(n - 1) + n;
}
```

으로 정의할 수 있다. 함수 Sum()에서 함수 자신을 호출하고 있다. 예를 들어, Sum(10)을 실행하면

```
return Sum(9) + 10;
```

이 실행된다. Sum(9)는 다시 동일한 함수를 호출하여 Sum(8) + 9를 계산한다. 이와 같이 인수를 계속 감소시키면 결국 Sum(1)에 도달한다. 이 때

```
return 1;
```

을 만나 호출이 종료된다. 정리하면

```
Sum(10) = Sum(9) + 10
        = Sum(8) + 9 + 10
         ⋮
        = Sum(1) + 2 + ⋯ + 10
        = 1 + 2 + ⋯ + 10
```

이 된다.

◇ 재귀 함수가 자신을 계속 호출하면 프로그램이 종료되지 않는다. 따라서 호출을 멈추는 조건이 포함되어야 한다.

◆ 연습문제 3.3.18. 1부터 n까지 정수의 곱을 반환하는 재귀 함수를 작성하시오.

◆ 연습문제 3.3.19. 두 수의 최대 공약수를 반환하는 재귀 함수를 작성하시오.

3.3.5 대리자. C#에서는 함수를 값으로 갖는 변수를 허용한다. 함수를 변수에 대입할 수 있으며 이 변수는 함수처럼 사용할 수 있다. 함수를 연결하는 형식을 **대리자**(delegate)라 한다. 대리자는 키워드 delegate를 붙여 선언한다.

delegate 반환_자료형 대리자_이름(매개_변수_자료형 매개_변수)

매개_변수는 개수에 제한이 없으며 없을 수도 있다.

대리자 Operation을

```
delegate double Operation(double d);
```

로 정의하면 Operation형 변수

```
Operation op;
```

는 함수를 값으로 갖는다. Operation과 같은 형식의 함수를 대입할 수 있다. 함수

```
static double AddOne(double d)
{
}
```

는 Operation과 같은 형식이므로 변수 op에 대입할 수 있다.

```
op = AddOne;
```

op는 함수처럼 호출할 수 있으며 AddOne과 똑같이 작용한다. 따라서

```
Console.WriteLine(op(3.4));
```

로 출력되는 값은 AddOne(3.4)이다.

예제 3.3.5 Delegate Demo

```
using System;

delegate double Operation(double d);

class Program
{
    static double AddOne(double d)
    {
        return d + 1;
    }

    static void Main()
    {
        Operation op = AddOne;
        double x = double.Parse(Console.ReadLine());
        Console.WriteLine(op(x));
    }
}
```

입력

```
3.4
```

```
출력
    4.4
```

◆ 연습문제 3.3.20. int 형식을 매개 변수로 하고 double 형식을 반환하는 대리자를 정의하시오.

◆ 연습문제 3.3.21. 다음 함수를 값으로 가질 수 있는 대리자를 정의하시오.

```
static void Display()
{
}
```

3.4 연산

수를 기반으로 하는 프로그램에서 사칙 연산은 필수적이다. 흐름을 제어하려면 관계 연산과 논리 연산이 필요하다. 이와 같이 일정한 규칙에 따라 변수의 값을 변화시키거나 하나 이상의 자료로부터 값을 얻는 것을 **연산**(operation)이라 한다. 연산을 표시하는 기호를 **연산자**(operator)라 한다. 예를 들어 x + y를 계산하는 것은 연산이고 +는 연산자이다.

3.4.1 연산자 분류. 연산은 항의 개수에 따라 단항 연산, 이항 연산, 삼항 연산으로 나뉜다. 부호를 바꾸는 -x나 값을 증가시키는 x++와 같은 연산은 항이 x 하나뿐이므로 **단항 연산**(unary operation)이라 한다. 이때 -와 ++를 **단항 연산자**(unary operator)라 한다. x + y와 같이 항이 둘인 연산을 **이항 연산**(binary operation)이라 하고 +를 **이항 연산자**(binary operator)라 한다. 항이 세 개인 **삼항 연산**(ternary operation)의 연산자를 **삼항 연산자**(ternary operator)라 한다. C#가 제공하는 연산자는 표 3.3과 같다.

종류	연산자	분류
부호	+ -	단항
산술	+ - * / %	이항
논리	! ~	단항
	&& || & | ^	이항
문자열 연결	+	이항
증가, 감소	++ --	단항
비트 이동	<< >>	이항
관계	== != < > <= >=	이항
대입	= += -= *= /= %= |= ^= <<= >>=	이항
조건	?:	삼항

표 3.3 연산자 분류

삼항 연산자 ?:는

조건 ? 조건이_참일_때_값 : 조건이_거짓일_때_값

으로 정의된다. 예를 들어

```
int y = x > 0 ? 1 : -1
```

을 실행한 후 y의 값은 x가 영보다 크면 1, 그렇지 않으면 -1이다. 따라서

```
int x = 23;
int y = x > 0 ? 1 : -1;
```

을 실행한 후 y의 값은 1이다.

◆ 연습문제 3.4.1. 변수 x가 짝수이면 연산 결과가 문자열 "짝수", 그렇지 않으면 "홀수"인 삼항 연산을 작성하시오.

3.4.2 복합 연산. 여러 연산자를 혼합하여 사용하면 한 연산을 수행하여 얻는 값을 다

른 연산에 사용한다. 이들이 어떻게 결합되는지 알아보자.

단항 연산자 ++는 변수의 값을 1만큼 증가시킨다. 따라서

```
int x = 23;
x++;
Console.WriteLine(x);
```

는 24를 출력한다. 이 경우에는 ++x처럼 ++가 변수의 앞에 와도 상관없다. 이 연산자를

```
x = 23;
int y = x++;
```

와 같이 다른 연산자와 혼합하여 사용할 수 있다. 이 경우 y는 23과 24 중 어느 것일까? 연산자 ++가 변수 뒤에 오면 변수의 값을 적용한 후 1만큼 증가시킨다. 변수 앞에 오면 변수를 1만큼 증가시킨 후 적용한다. 따라서 위 코드를 실행한 후 y의 값은 23이고 x의 값은 24이다. 코드

```
x = 23;
int y = ++x;
```

를 실행하면 x와 y의 값은 모두 24이다. 단항 연산자 --는 변수의 값을 1만큼 감소시키며 복합 연산에서는 연산자 ++와 같은 규칙이 적용된다.

◆ 연습문제 3.4.2. 다음 코드가 출력하는 값은 무엇인가?

```
int x = 7;
Console.WriteLine(-x--);
Console.WriteLine(x++ * x);
```

결과를 추측해보고 실행하여 확인하시오.

한 문장에서 변수에 값을 대입하고 그 변수의 값을 사용할 수 있다.

```
x = y = 1;
a = (b = 1) + 2;
```

첫째 문장에서는 y에 1을 대입하고 y의 값을 x에 대입한다. 둘째 문장에서는 b에 1을 대입하고 b와 2를 더하여 a에 대입한다. 따라서 a는 3이다. 마찬가지로 식

```
if((input += 3) > 0)
```

는 input에 3을 더하고 input이 0보다 큰지 조사한다. 따라서

```
int input = -1;
if ((input += 3) > 0)
    Console.WriteLine(input);
```

은 2를 출력한다.

다음 예제는 입력된 값이 양수인 동안 반복되는 while 루프를 포함하고 있다. 입력된 정수 중 약수가 네 개 이상인 것의 개수를 출력한다.

예제 3.4.1 Operation Demo1

```
using System;

class Program
{
    static void Main()
    {
        int count = 0;
        int input;
        while ((input = int.Parse(Console.ReadLine())) > 0)
        {
            if (NumberOfFactors(input) >= 4)
                count++;
        }
        Console.WriteLine(count);
    }

    static int NumberOfFactors(int n)
    {
        int nFactors = 0;
        for (int i = 1; i <= n; i++)
            if (n % i == 0)
                nFactors++;
        return nFactors;
    }
}
```

```
입력
    9
    12
    15
    0
출력
    2
```

◆ 연습문제 3.4.3. 문자열을 한 줄씩 읽어 출력하시오. 입력된 문자열이 "quit"이면 종료하시오.

3.4.2 비트 연산. 비트 연산은 int 형식과 long 형식에 대하여 각 비트별로 연산을 수행한다. 비트의 값이 0이면 false, 1이면 true로 보고 계산한다.

연산자	이름	연산 결과
&	비트 논리곱	두 비트가 모두 1이면 1
|	비트 논리합	두 비트 중 하나라도 1이면 1
^	비트 배타적 논리합	두 비트가 서로 다르면 1
~	비트 부정	비트가 0이면 1

표 3.4 비트 논리 연산자

두 변수 x, y를 다음과 같이 정의하자.

```
int x = 10;
int y = 12;
```

변수 x의 오른쪽 네 비트는 1010이고 나머지 왼쪽 28 비트는 0이다(표 3.5). 또, y의

오른쪽 네 비트는 1100이고 나머지 28 비트는 0이다. x, y에 대하여 비트 논리 연산을 수행하면 표 3.5와 같은 결과를 얻는다.

식	비트	값
x	0 … 01010	10
y	0 … 01100	12
x & y	0 … 01000	8
x ¦ y	0 … 01110	14
x ^ y	0 … 00110	6
~x	1 … 10101	-11
~y	1 … 10011	-13

표 3.5 비트 논리 연산

따라서

```
int z = x & y;
Console.WriteLine(z);
```

는 8을 출력한다.

◆ 연습문제 3.4.4. 두 수 21과 11의 비트 연산을 모두 계산하시오.

◇ 2의 보수를 사용하는 C#에서 비트 논리 부정 연산 ~x는 -(x+1)과 같다.

비트 이동 연산자(shift operator) <<와 >>는 int, uint, long, ulong 형식에 적용할 수 있다.

```
x << n
```

은 x의 비트를 왼쪽으로 n 만큼 이동시킨다. x의 왼쪽 n 비트는 무시되고 오른쪽 n 비트는 0으로 채워진다. 이동 비트 수 n은 x가 int 형식, uint 형식인 경우 0부터 31까지 유효하며 long형, ulong형인 경우 0부터 63가지 유효하다.

```
x >> n
```

은 x의 비트를 오른쪽으로 n 만큼 이동시킨다. x가 int 형식이거나 long형이면 왼쪽 n 비트는 부호 비트로 채워진다. x가 uint 형식이거나 ulong형이면 왼쪽 n 비트는 0으로 채워진다. 이동 비트 수 n은 x가 int 형식, uint 형식인 경우 0부터 31까지 유효하며 long형, ulong형인 경우 0부터 63가지 유효하다.

두 int 형식 변수 a, b에 대하여 비트 이동 연산을 수행하면 다음 표와 같은 결과를 얻는다.

식	비트	값
a	0 ··· 0001010	10
a << 2	0 ··· 0101000	40
a >> 2	0 ··· 0000010	2
b	1 ··· 1110110	-10
b << 2	1 ··· 1011000	-40
b >> 2	1 ··· 1111101	-3

표 3.5 비트 이동 연산

◆ 연습문제 3.4.5. 비트 이동 1 << 2는 얼마인가?

◇ 비트 이동 연산자의 이동 비트 수는 위 정의와 약간 다르다. 정확한 정의를 찾아보자.

산술 연산과 마찬가지로 비트 연산도 축약할 수 있다. 변수 x가 두 번 나오는 연산

```
x = x & y;
```

는 축약하여

```
x &= y;
```

로 쓸 수 있다. |, ^, <<, >>도 마찬가지로 |=, ^=, <<=, >>=로 축약할 수 있다.

◆ 연습문제 3.4.6. 다음 코드가 출력하는 값은 무엇인가?

```
int x = 9;
x >>= 2;
Console.WriteLine(x);
```

3.4.3 연산자 우선순위. 연산자가 여럿 포함된 식을 계산하려면 연산의 순서를 정해야 한다. 순서를 정하는 기준을 **연산자 우선순위**(operator precedence)라 한다.

식을 계산할 때는 우선순위가 높은 연산부터 차례로 수행한다. x + y * z에서 곱셈이 덧셈보다 우선순위가 높으므로 y * z부터 계산한다. 우선순위가 같은 두 연산은 왼쪽부터 계산한다. 연산자 우선순위는 표 3.7과 같이 주어진다.

우선순위	연산자	결합	기타
높음	() [] . ++ --	오른쪽	
	++ -- + - ~ !	왼쪽	단항 연산자
	* / %		
	+ -		이항 연산자
	<< >>		
	< > <= >=		
	== !=		
	&		
	^		
	|		
	&&		
	||		
	?:		
낮음	= += -= *= /= %= &= ^= |= <<= >>=		

표 3.7 연산자 우선순위

표 3.7에서 결합은 단항 연산자를 변수의 왼쪽과 오른쪽 중 어느 곳에 붙이는가를 나타낸다. 예를 들어

```
x++
```

는 연산자가 오른쪽에 붙으므로 오른쪽 결합이다.

◆ 연습문제 3.4.7. 두 식

```
1 - 2 + 3
"" + 1 + 2
```

에서 연산의 순서를 각각 말하시오.

◆ 연습문제 3.4.8. 문장

```
bool b = 2 * 3 == 6;
```

에서 연산의 순서를 말하시오.

다음 예제는 2의 16 제곱을 출력한다. 일반적으로 정수 x의 n 제곱을 계산하는 함수를 정의하여 그 함수로부터 값을 얻는다.

예제 3.4.2 Operation Demo2

```
using System;

class Program
{
    static int Power(int x, int n)
    {
        int value = 1;
        while (n > 0)
        {
            if ((n & 1) == 1)
```

```
                value *= x;
            x *= x;
            n >>= 1;
        }
        return value;
    }
    static void Main()
    {
        Console.WriteLine(Power(2, 16));
    }
}
```

출력

```
65536
```

식 (n & 1) == 1에서 &의 우선순위가 ==보다 낮으므로 괄호로 묶어 먼저 계산하도록 하였다.

◇ 괄호와 대괄호를 연산자에 포함시킬 수도 있다. 그 때 이들의 우선순위는 가장 높다.

지금까지 공부한 내용을 정리하는 의미로 달력을 출력하는 프로그램을 작성한다.

예제 3.4.3 Calendar

```
using System;

class Program
{
    static bool IsLeapYear(int year)
    {
        return year % 400 == 0 || (year % 100 != 0 && year % 4 == 0);
    }

    static int DatesSinceFirstYear(int year)
    {
        return --year * 365 + year / 4 - year / 100 + year / 400;
    }

    static int NumberOfDates(int year, int month)
    {
        switch (month)
        {
            case 4:
            case 6:
            case 9:
            case 11:
                return 30;
            case 2:
                return IsLeapYear(year) ? 29 : 28;
            default:
                return 31;
        }
    }

    static int FirstDayOfWeek(int year, int month)
    {
```

```
            int total = DatesSinceFirstYear(year);
            for (int i = 1; i < month; i++)
                total += NumberOfDates(year, i);
            return (total + 1) % 7;
        }

        static void Print(int year, int month)
        {
            Console.WriteLine("    {0}년 {1}월", year, month);

            string[] days = { "일", "월", "화", "수", "목", "금", "토" };
            foreach (string str in days)
                Console.Write(str + " ");
            Console.WriteLine();

            int pos = FirstDayOfWeek(year, month) - 1;
            for (int i = 0; i <= pos; i++)
                Console.Write("   ");
            int last = NumberOfDates(year, month);
            for (int i = 1; i <= last; i++)
            {
                if (++pos == 7)
                {
                    Console.WriteLine();
                    pos = 0;
                }
                Console.Write(i < 10 ? " {0} " : "{0} ", i);
            }
            Console.WriteLine();
        }
        static void Main()
        {
            Print(9999, 12);
        }
    }
```

출력

```
    9999년 12월
일 월 화 수 목 금 토
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

제 4 장 객체 지향 프로그래밍 I

프로그램에서 다루는 대상은 정수와 같이 한 요소로 이루어진 것도 있고 직사각형과 같이 여러 요소들로 이루어진 것도 있다. 후자의 경우 각 요소들을 기본 형식으로 표현한다. 지금까지는 대상을 구성하는 각 요소들을 독립된 변수로 나타내어 작업을 수행하였다. 작업의 단위는 대상이 아니라 요소들이었다. 대상은 프로그램 작성자의 생각에는 있지만 코드에는 명시되지 않는다.

객체 지향 프로그래밍은 여러 요소로 이루어진 대상도 작업의 단위로 설정할 수 있는 프로그래밍 기법이다. 대상의 요소들로 구성된 설계도인 클래스로 작업의 단위를 설정한다. 대상을 표현하는 객체는 설계도에 주어진 대로 만들어지며 작업은 객체 단위로 수행된다.

이 장에서는 객체를 규정하는 클래스에 대하여 알아본다. 또 객체 지향 프로그램의 세 가지 주요 특성인 상속, 캡슐화, 다형성에 대하여 설명한다.

4.1 객체와 클래스

폭과 높이로부터 직사각형의 넓이를 구하는 프로그램은 다음과 같다.

```
class Program
{
    static double Area(double width, double height)
    {
        return width * height;
    }

    static void Main()
    {
        double width = 1.3;
        double height = 2.4;
        Console.WriteLine("폭: " + width + ", 높이: " + height);
        Console.WriteLine("넓이: " + Area(width, height));
    }
}
```

이 프로그램에서 다루는 대상은 직사각형이다. 폭 width와 높이 height는 직사각형을 구성하는 요소이고 함수 Area()는 직사각형의 넓이를 계산한다. 따라서 세 요소 width, height, Area()는 직사각형이라는 테두리로 한데 묶을 수 있다. 그러나 코드에는 직접적인 관련성이 드러나지 않는다. 각 단어의 뜻과 함수의 계산 방법으로부터 직사각형을 유추할 수는 있다. 만약, 복잡한 코드 속에 세 요소가 포함되어 있다면 직사각형과의 관련성이 바로 보이지는 않을 것이다. 여기에 고민이 있다. 직사각형이 눈에 보이도록 작성할 수는 없을까? 그것은 설계도인 클래스를 작성하여 해결할 수 있다. 세 요소를 묶어 클래스를 만들어 보자.

4.1.1. 객체. 앞에서 설명한 직사각형이 어떻게 구성되고 활용되는지 알아보자.

예제 4.1.1 Class Demo

```
using System;

class Rectangle
{
    public double width;
    public double height;

    public double Area()
    {
        return width * height;
    }
}

class Program
{
    static void Main()
    {
        Rectangle rect = new Rectangle();
        rect.width = 1.3;
        rect.height = 2.4;
```

```
            Console.WriteLine("폭: " + rect.width + ", 높이: " + rect.height);
            Console.WriteLine("넓이: " + rect.Area());
        }
    }
```

```
출력
    폭: 1.3, 높이: 2.4
    넓이: 3.12
```

예제 4.1.1에서 Rectangle은 직사각형을 나타내는 클래스이다. 각각 폭과 높이를 나타내는 두 double 형식 변수 width와 height가 선언되어 있으며 넓이를 반환하는 함수 Area()가 정의되어 있다. new Rectangle()은 직사각형을 생성한다. 이 직사각형의 폭과 높이를 설정하고 넓이를 계산하여 출력한다. 앞에서 작성했던 코드와 비교해 보기 바란다. public은 4.3에서 자세히 설명한다.

프로그램에서 다루는 대상 중 기본 형식이 아닌 것을 **객체**(object)라 한다. 객체는 자료와 함수로 이루어진다. 변수들과 함수들을 묶어 그들의 관련성을 겉으로 드러낸다. 여러 요소들을 묶어 변수 하나로 표현함으로써 복잡한 것을 단순화시키는 장점도 있다. 예제 4.1.1에서 클래스 Rectangle은 대상이 직사각형임을 명확히 드러낸다. 뒤에서 설명하겠지만 변수 rect는 한 직사각형을 나타낸다. 직사각형 객체라 할 수 있다. 그 객체는 폭 width와 높이 height라는 자료, 그리고 넓이를 계산하는 함수 Area()로 구성된다.

객체 지향 프로그래밍(object oriented programming, OOP)은 객체들의 상호 작용을 기술하는 프로그래밍 기법을 말한다. 다루고자 하는 대상을 분석하여 설계도인 클래스를 작성한다. 프로그램에서는 설계도대로 만들어진 객체들을 적절히 운용하여 원하는 결과를 얻는다.

◇ 사람에 따라 '객체' 대신 '개체'라는 표현을 쓰기도 한다. 사실 후자가 더 적합한 표현이다. 그러나 관습적으로 전자를 써왔기 때문에 어느 것을 쓸지 망설여진다. 이 책에서는 전자를 선택하였다.

4.1.2 클래스. 객체를 규정하는 틀(template)을 **클래스**(class)라 한다. 객체에 대한 설계도이다. 클래스에는 자료를 저장하는 변수와 객체에 적용되는 함수가 포함된다. 클래스에 명시된 것들은 모두 사용할 수 있으며 그것들만 사용할 수 있다. 객체는 그것을 규정하는 클래스의 **인스턴스**(instance)라 한다. 인스턴스는 틀인 클래스에 설계된 대로 구성되고 메모리 공간이 할당된다. 객체를 구성하는 자료를 **필드**(field)라 한다. **인스턴스 필드**(instance field)나 **멤버 변수**(member variable)라 부르기도 한다. 객체에 적용되는 함수를 **메서드**(method)라 한다. 객체 자신이나 다른 객체에 대한 행위를 기술한다. **인스턴스 메서드**(instance method)나 **멤버 함수**(member function)라 부르기도 한다. 예제 4.1.1의 width와 height가 필드이고 Area()가 메서드이다. new Rectangle()은 클래스 Rectangle의 인스턴스, 곧 Rectangle 객체를 생성한다.

◇ 필드는 객체를 구성하는 자료를 일컬어 말하기도 하고 그것을 클래스에서 표현한 변수를 일

컫기도 한다.

클래스는 키워드 class와 그 이름을 적어서 정의한다. 클래스 Rectangle은

```
class Rectangle {
```

로 정의한다. 이 클래스에 각각 폭과 높이를 나타내는 두 필드 width와 height, 그리고 넓이를 반환하는 메서드 Area()를 추가하면

```
class Rectangle
{
    public double width;
    public double height;

    public double Area()
    {
        return width * height;
    }
}
```

와 같은 형태가 된다. 필드는 변수이므로 자료 형식을 명시해야 한다. 기본 형식, 배열, 클래스 등 어떤 자료 형식이든지 상관없다. 메서드는 3.3.1의 정적 메서드와 유사한 형식으로 정의된다. 필요에 따라 반환 형식과 매개 변수를 적어 준다. 정적 메서드와 달리 static은 붙이지 않는다. 클래스 이름에는 모든 단어의 첫 글자를 대문자로 쓰도록 권장한다. 곧, 파스칼 표기법을 권장한다(3.3).

```
class MyRoom
class CityInAsia
```

필드는 낙타 표기법, 메서드는 파스칼 표기법으로 적는다.

클래스를 틀로 하는 객체를 만들려면 키워드 new를 사용한다.

```
Rectangle rect = new Rectangle();
```

이 문장에서 new Rectangle()이 객체를 생성한다. 이 객체를 Rectangle 객체 또는 Rectangle의 인스턴스(instance)라 한다. 변수 rect는 Rectangle 객체를 참조한다. 객체를 메모리에 저장하고 그 주소를 변수 rect에 저장하는 것으로 이해하면 된다. rect는 Rectangle 객체를 참조하는 변수라는 뜻으로 Rectangle 형식 변수라 한다. 다시 말하지만 객체를 변수에 저장하는 것이 아니다. 객체는 따로 메모리에 저장하고 그 주소를 변수에 저장한다. 일반적으로 객체를 참조하는 변수를 **참조 변수**(reference variable)라 한다.

◇ 객체를 참조하는 변수와 객체는 당연히 서로 다르다.

```
Rectangle rect = new Rectangle();
```

에서 변수 rect는 new Rectangle()로 생성된 객체를 참조한다. 따라서 '객체 rect'라는 표현은 바르지 않다. 'rect가 참조하는 객체'가 바른 표현이다. 그러나 용어가 길어지는 단점을 피하기 위하여 때때로 '객체 rect'로 쓰기도 한다.

필드는 객체를 구성하고 메서드는 객체에 적용되는 함수이다. 따라서 필드를 사용

하려면 그들이 어떤 객체를 구성하는지 밝혀야 한다. 메서드는 어떤 객체의 적용되는지 밝혀야 한다. 필드가 선언된 클래스 밖에서 그 필드에 접근하려면 객체를 명시해야 한다. 어떤 객체의 필드인지 밝혀야 되기 때문이다. 메서드도 마찬가지이다. 예제 4.1.1의 문장

```
rect.width = 1.3;
```

의 rect.width는 객체 rect의 필드임을 나타낸다. Rectangle 객체가 여럿인 상황을 생각해보면 이해가 좀 더 빠를 것이다. 또 다른 Rectangle 객체

```
Rectangle rect1 = new Rectangle();
```

을 만들어 보자. 이와 같은 상황에서 단순히 width라고 하면 rect와 rect1 중 어느 객체의 폭인지 알 수 없다. 따라서 rect.width, rect1.width와 같이 어느 객체의 폭인지 밝혀 적는다.

필드가 선언된 클래스 내부에서 그 필드에 접근하려면 이름만 적는다. 예제 4.1.1의 문장

```
return width * height;
```

의 width나 height와 같은 형태이다. 이 문장은 Main()의

```
Console.WriteLine("넓이: " + rect.Area());
```

와 같이 메서드 Area()를 호출하면 실행된다. 이 때 width와 height는 Area()를 호출하는 객체, 곧 rect를 구성하는 필드이다.

- ◆ 연습문제 4.1.1. 예제 4.1.1의 Main()에 Rectangle 객체를 추가하시오. 두 필드 width와 height를 설정하고 넓이를 출력하시오.
- ◆ 연습문제 4.1.2. 반지름으로부터 원둘레를 계산하는 프로그램을 작성하시오. 원을 나타내는 클래스 Circle을 만들어 사용하시오.
- ◆ 연습문제 4.1.3. 정수의 절댓값을 구하는 프로그램을 작성하시오. 정수를 나타내는 클래스 Number를 만드시오. 이 클래스에 절댓값을 반환하는 메서드 Abs()를 포함시키시오.
- ◆ 연습문제 4.1.4. 책을 나타내는 클래스 Book을 만드시오. 제목을 나타내는 필드 title과 제목을 출력하는 메서드 PrintTitle()이 포함되도록 구성하시오.

년과 월을 지정하여 그 달의 날짜 수를 계산하는 프로그램을 작성해보자. 클래스는 어떻게 구성하는 것이 좋을까? 년과 월을 모두 필드로 가지도록 할 수 있다. 또, 년만 필드로 가지도록 할 수도 있다. 선택은 용도에 의하여 결정된다. 여기서는 어느 것이든 상관없지만 후자를 선택한다. 다음 예제를 예제 3.4.3과 비교해보기 바란다.

예제 4.1.2 Year

```
using System;

public class Year
```

```
{
    public int n;

    public bool IsLeap()
    {
        return n % 400 == 0 || (n % 100 != 0 && n % 4 == 0);
    }

    public int NumberOfDates(int month)
    {
        switch (month)
        {
            case 4:
            case 6:
            case 9:
            case 11:
                return 30;
            case 2:
                return IsLeap() ? 29 : 28;
            default:
                return 31;
        }
    }

    static void Main()
    {
        Year year = new Year();
        year.n = int.Parse(Console.ReadLine());
        Console.WriteLine(year.IsLeap());
        int month = int.Parse(Console.ReadLine());
        Console.WriteLine(year.NumberOfDates(month));
    }
}
```

```
입력
    9900
    2
출력
    false
    28
```

메서드 Main()에서는 메서드 IsLeap() 앞에 year를 붙여서 호출한다. 어느 객체가 IsLeap()을 호출하는지 알아야 하기 때문이다. 반면 메서드 NumberOfDates()에서는 객체를 붙이지 않고 호출한다. 이 때에는 NumberOfDates()를 호출하는 객체가 IsLeap()을 호출한다. 예를 들어, Main()에서

```
Console.WriteLine(year.NumberOfDates(month));
```

를 호출하면 year가 IsLeap()을 호출한다.

◆ 연습문제 4.1.5. 예제 3.4.3과 같은 달력을 출력하는 메서드를 예제 4.1.2의 클래스 Year에 추가하시오.

◆ 연습문제 4.1.6. 년과 월을 필드로 가지는 클래스로 예제 4.1.2를 다시 작성하시오.

◆ 연습문제 4.1.7. 세 수를 읽어 맨 나중에 읽은 수가 나머지 두 수 중 어느 수에 가까운지 판

별하시오. 먼저 읽은 두 수로 객체를 구성하시오.

◆ 연습문제 4.1.8. 두 수를 필드로 가지는 클래스를 작성하여 최소 공배수를 구하시오.

Main()은 프로그램 진입점으로 동작할 뿐 그 외에 다른 역할은 없다. 그것이 속한 클래스에도 영향을 주지 않는다. 따라서 어디든지 편리한 클래스에 포함시키면 된다. 두 예제 4.1.1과 예제 4.1.2를 비교해보기 바란다.

4.1.3 생성자. 예제 4.1.1에서

```
Rectangle rect = new Rectangle();
```

로 Rectangle 객체를 만들었다. 객체를 생성할 때는 new Rectangle()과 같이 클래스와 이름이 같은 함수에 new를 붙여 호출한다. **생성자**(constructor)는 클래스와 이름이 같은 함수이다. 메서드와 비슷하게 정의하지만 반환하는 값이 없으며, 따라서 반환 형식도 적지 않는다. 생성자에서는 객체 및 그것의 필드를 저장할 공간을 확보한다. 또, 필드의 값과 같은 객체의 속성을 설정한다.

클래스 Rectangle의 생성자를 다음과 같이 정의할 수 있다.

```
public Rectangle(double w, double h)
{
    width = r;
    height = h;
}
```

이 생성자는 new를 붙이고 인수를 대입하여 호출한다.

```
Rectangle rect = new Rectangle(1.3, 2.4);
```

이 때 new Rectangle(1.3, 2.4)은 Rectangle 객체를 생성하고 변수 rect는 이 객체를 참조한다. 생성된 객체의 두 필드 width와 height는 각각 1.3과 2.4가 된다. 따라서

```
Rectangle rect = new Rectangle();
rect.width = 1.3;
rect.height = 2.4;
```

와 같은 효과가 있다. 생성자를 정의하면 객체를 생성하는 코드가 단순해진다. 전체 코드도 이해하기 쉬워진다.

다음은 예제 4.1.1의 클래스 Rectangle에 생성자를 추가한 것이다.

예제 4.1.3 ClassDemo 수정

```
using System;

class Rectangle
{
    public double width;
    public double height;
```

```
        public Rectangle(double w, double h)
        {
            width = w;
            height = h;
        }

        public double Area()
        {
            return width * height;
        }
    }

    class Program
    {
        static void Main()
        {
            Rectangle rect = new Rectangle(1.3, 2.4);
            // Rectangle rect = new Rectangle();
            // rect.width = 1.3;
            // rect.height = 2.4;
            Console.WriteLine("폭: " + rect.width + ", 높이: " + rect.height);
            Console.WriteLine("넓이: " + rect.Area());
        }
    }
```

객체를 생성하려면 반드시 키워드 new와 생성자를 적어야 한다. 따라서 모든 클래스는 생성자를 가져야 한다. 그런데 예제 4.1.1이나 예제 4.1.2의 코드에는 생성자의 정의가 보이지 않는다. 코드에 생성자가 없으면 컴파일러는 자동으로 매개 변수가 없는 생성자를 추가한다. 예를 들어 예제 4.1.1의 클래스 Rectangle에는

```
Rectangle()
{
}
```

이 추가된다. Main()의

```
Rectangle rect = new Rectangle();
```

은 이 생성자를 호출한 것이다. 예제 4.1.2의 Year에도 유사한 형태의 생성자가 추가된다.

매개 변수가 없는 생성자를 **기본 생성자**(default constructor)라 한다. 클래스에 생성자를 정의하지 않으면 컴파일러는 기본 생성자를 추가한다. 생성자가 하나라도 정의하면 추가하지 않는다. 클래스에 기본 생성자를 정의해도 되며 이 경우에도 컴파일러는 기본 생성자를 추가하지 않는다.

생성자에서는 모든 필드의 값을 설정한다. 코드에서 명시적으로 값을 설정하지 않으면 컴파일러에 의하여 자동으로 설정된다. 자동으로 설정되는 값은 필드가 정수이면 0, 실수이면 0.0, bool 형식이면 false, 참조 변수이면 null이다. null은 참조하는 객체가 없음을 나타낸다. 4.4.1에서 설명한다.

◆ 연습문제 4.1.9. 코드

```
class A
{
    public int x;

    public A(int t)
    {
        x = t;
    }

    static void Main()
    {
        A a = new A();
    }
}
```

에서 잘못된 점은 무엇인가? 이유는 무엇인가?

◆ 연습문제 4.1.10. 코드

```
class B
{
    public int x;

    static void Main()
    {
        B b = new B();
        Console.WriteLine(b.x);
    }
}
```

를 실행하면 어떤 값이 출력되는가? 이유는 무엇인가?

◆ 연습문제 4.1.11. 예제 4.1.2에서 정의한 클래스 Year에 생성자를 추가하시오.

4.1.4 this. 코드는 그 의미를 빠르고 정확하게 파악할 수 있도록 작성하여야 한다. 따라서 변수 이름은 뜻이 잘 전달되는 것이 좋다. 앞에서 작성한 프로그램

```
class Rectangle
{
    public double width;
    public double height;

    public Rectangle(double w, double h)
    {
        width = w;
        height = h;
    }
```

에서 w와 h는 좋은 변수 이름이라 할 수 없다. 그 의미를 파악하기 어렵기 때문이다. 가급적 width와 height처럼 뜻이 일치하는 이름을 사용하고 싶다. 그러나 그렇게 하면 필드와 이름이 겹쳐 구별할 수 없다. 이 때 객체 자신을 가리키는 키워드 this를 사용한다.

```
public Rectangle(double width, double height)
{
    this.width = width;
```

```
        this.height = height;
    }
```

this.width는 필드를, this가 없는 width는 매개 변수를 나타낸다.

◆ 연습문제 4.1.12. 예제 4.1.2에서 정의한 클래스 Year에 생성자를 추가하고 this를 써서 필드의 값을 설정하시오.

◆ 연습문제 4.1.13. 키워드 this는 객체 자신을 가리킨다고 하였다. 여기서 '객체 자신'의 뜻은 무엇인가? 직사각형 객체를 예로 설명하시오.

4.2 상속

생물 분류에서 곤충과 개미를 생각해 보자. 계통상 하위에 위치하는 개미는 곤충이 만족해야 할 조건을 모두 만족한다. 곤충의 속성은 모두 개미의 속성이 된다.

객체 지향 프로그래밍에서는 객체를 분류하여 계통을 만들고 각 항목을 클래스로 나타낸다. 계통의 하위에 위치하는 클래스는 상위에 위치하는 클래스의 필드와 메서드를 모두 포함한다. 상위에 위치하는 클래스의 속성을 모두 포함한다.

4.2.1 파생 클래스. 클래스 B의 인스턴스가 모두 클래스 A의 인스턴스이면 A를 B의 **기본 클래스(**base class)라 하고 B를 A의 **파생 클래스**(derived class)라 한다. B는 A에서 파생된 클래스라고 표현하기도 한다. 예를 들어 곤충과 개미의 관계에서 곤충은 기본 클래스이고 개미는 파생 클래스이다.

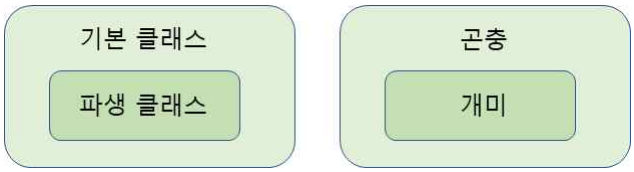


그림 4.1 기본 클래스와 파생 클래스

곤충의 속성이 모두 개미의 속성인 것처럼, 파생 클래스는 기본 클래스의 필드와 메서드를 모두 포함한다. 파생 클래스가 기본 클래스를 **상속**(inheritance)한다고 한다. 일반적으로 파생 클래스에서는 기본 클래스의 필드를 자신의 필드처럼 사용한다. 메서드도 마찬가지이다.

◇ 기본 클래스는 **상위 클래스**(superclass), **부모 클래스**(parent class)라고도 하며 파생 클래스는 **하위 클래스**(subclass), **자식 클래스**(child class)라고도 한다.

사람과 학생을 클래스로 정의해 보자. 학생은 사람이므로 사람이 기본 클래스, 학생이 파생 클래스가 될 것이다. 여기서는 클래스 Person이 사람을 표현하고 필드 name과 메서드 PrintName()을 가지도록 한다. name은 이름이고 PrintName()은 이름을 출력한다.

```
class Person
{
    public string name;

    public void PrintName()
    {
    }
}
```

파생 클래스 Student는 학번을 나타내는 필드 id와 학생의 정보를 반환하는 메서드 GetInfo()를 갖는다. 파생 클래스를 정의하려면 쌍점(:)을 사용한다.

```
class Student : Person
{
```

```
    public int id;

    public string GetInfo()
    {
    }
}
```

Student는 id와 GetInfo() 뿐만 아니라 name과 PrintName()까지 포함한다. Person에서 파생된 클래스이기 때문이다.

예제 4.2.1 Inheritance Demo

```
using System;

class Person {
    public string name;

    public void PrintName()
    {
        Console.WriteLine(name);
    }
}

class Student : Person
{
    public int id;

    public string GetInfo()
    {
        return "이름: " + name + ", 학번: " + id;
    }
}

class Program
{
    static void Main()
    {
        Student s = new Student();
        s.name = "장 길산";
        s.id = 1019;
        s.PrintName();
        Console.WriteLine(s.GetInfo());
    }
}
```

출력

```
장 길산
이름: 장 길산, 학번: 1019
```

필드 name은 Person에 포함된 필드이다. 그런데 마치 Student에 속한 것처럼 사용되고 있다. 메서드 PrintName()도 마찬가지이다.

◆ 연습문제 4.2.1. 직사각형 유리를 나타내는 클래스 Glass를 작성하시오. 색상을 나타내는 필드를 추가하여 파생 클래스 ColoredGlass를 작성하시오.

◆ 연습문제 4.2.2. 각각 애완동물과 고양이를 나타내는 기본 클래스와 파생 클래스를 정의하시

오.

◆ 연습문제 4.2.3. 은하에 대하여 조사하여 클래스 Galaxy를 만드시오. 타원 은하와 나선 은하를 각각 Galaxy의 파생 클래스로 만드시오.

클래스는 둘 이상의 클래스에서 직접 파생될 수 없다. 다시 말하면

```
class Foo extends Bar, Qux {  // 오류
```

와 같이 두 기본 클래스를 가질 수 없다.

◇ 한 클래스가 여러 클래스에서 직접 파생되는 것을 **다중 상속**(multiple inheritance)이라 한다. C#는 다중 상속을 허용하지 않는다. C++는 다중 상속을 허용한다.

◇ 한 파일에는 한 클래스만 두는 것이 좋다. 이것은 권장 사항이다. 여러 클래스를 한 파일에 모아 놓아도 되지만 여러모로 불편하다. 이 책의 예제들은 편의상 한 파일에 여러 클래스를 두고 있지만 실제 프로젝트를 수행할 경우에는 분리하는 것이 좋다. 예를 들어, 예제 4.2.1은 세 파일로 나누는 것이 좋다.

파생 클래스의 파생 클래스를 만들 수 있다.

```
class A {}
class B : A {}
class C : B {}
```

와 같은 형식이다. 이 때 클래스 C는 B의 파생 클래스이기도 하고 A의 파생 클래스이기도 하다. 역으로 A는 B의 기본 클래스이기도 하고 C의 기본 클래스이기도 하다. 세 클래스 A, B, C의 관계는 동물, 곤충, 개미의 관계와 유사하다(그림 4.2).

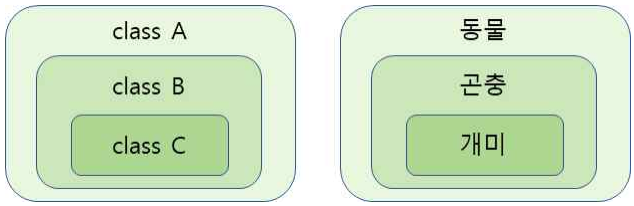


그림 4.2 파생 클래스의 파생클래스

4.2.2 생성자. 파생 클래스를 초기화하려면 우선 기본 클래스를 초기화해야 한다. 기본 클래스의 필드를 초기화한 후 파생 클래스의 필드를 초기화하는 것이다. 따라서 파생 클래스의 생성자는 가장 먼저 기본 클래스의 생성자를 실행한다. 기본 클래스의 생성자는 함수 base()로 호출한다. 클래스

```
class Foo
{
    public Foo( … )
    {
```

에서 파생된 클래스는

```
class Bar extends Foo
{
```

```
    public Bar( … ) : base( … )  // Foo의 생성자 Foo() 호출
    {
```

와 같은 형태이다.

◇ 키워드 base는 base class(기본 클래스)에서 따온 것이다.

다음 예제는 예제 4.2.1에 생성자를 추가한 것이다.

예제 4.2.2 InheritanceDemo 수정

```
using System;

class Person
{
    public string name;

    public Person(string name)
    {
        this.name = name;
    }

    public void PrintName()
    {
        Console.WriteLine(name);
    }
}

class Student : Person
{
    public int id;

    public Student(string name, int id) : base(name)
    {
        this.id = id;
    }

    public string GetInfo()
    {
        return "이름: " + name + ", 학번: " + id;
    }
}

class Program
{
    static void Main()
    {
        // Student s = new Student();
        // s.name = "장 길산";
        // s.id = 1019;
        Student s = new Student("장 길산", 1019);
        s.PrintName();
        Console.WriteLine(s.GetInfo());
    }
}
```

◆ 연습문제 4.2.4. 연습문제 4.2.1에서 작성한 두 클래스 Glass와 ColoredGlass에 매개 변수

가 있는 생성자를 각각 추가하시오.

예제 4.2.1을 다시 살펴보자. 기본 클래스 Person과 파생 클래스 Student에 기본 생성자를 추가하면 다음과 같다.

```
class Person
{
    public string name;

    public Person()
    {
    }
        ⋮
}

class Student : Person
{
    public int id;

    public Student()
    {
    }
        ⋮
```

이 코드는 잘 동작한다. 그런데 Student의 생성자에 base()가 보이지 않는다. 어떻게 된 일일까? 생성자에 base()를 적지 않으면, 컴파일러가

```
public Student() : base()
```

와 같이 base()를 삽입한다. 이 코드는 기본 생성자를 호출한다.

◆ 연습문제 4.2.6. 예제 4.2.2에 정의된 클래스 Student의 생성자를

```
public Student(string name, int id)
{
    this.name = name;
    this.id = id;
}
```

로 바꾸면 오류가 발생한다. 그 이유는 무엇인가?

4.3 캡슐화

운전하는 사람이 자동차의 구조를 정확히 알 필요는 없다. 단지 조작법을 익혀 그대로 시행할 줄 알면 된다. 자동차의 구조를 알아야 운전할 수 있다면 그것은 얼마나 큰 고통이겠는가.

객체 지향 프로그래밍에는 **캡슐화**(encapsulation)라는 기법이 있다. 객체를 사용하는 입장에서는 알 필요가 없는 것을 감추는 기법이다. 캡슐화의 구현에는 두 가지 측면이 있다. 하나는 객체를 구성하는 자료를 외부에 밝히지 않는 것이다. 다른 하나는 객체의 용도와 사용법을 밝히는 것이다. 자동차가 어떤 부품으로 구성되어 있는지 알 필요는 없다. 어떤 연료를 주입하고 어떻게 주행하는지 알면 된다.

구체적인 예를 하나 들어보자. 온도를 나타내는 클래스 Temperature가 다음과 같이 사용된다고 하자.

```
Temperature temp = new Temperature(36.5);  // 섭씨 온도 설정
Console.WriteLine(temp.getCelsius());     // 섭씨 온도 출력
Console.WriteLine(temp.getFahrenheit());  // 화씨 온도 출력
Console.WriteLine(temp.getKelvin());      // 절대 온도 출력
```

객체 temp(클래스 Temperature라 해도 좋다)는 분명 온도를 필드로 가지고 있을 것이다. 그러나 필드가 섭씨, 화씨, 절대 온도 중 어느 것일까? 어느 것이든 상관없고 어느 것인지 알 필요도 없을 것이다. 필드를 굳이 드러내지 않고 get...()과 같은 메서드만 사용하면 된다. 이것이 캡슐화이다.

◇ 생성자의 인수가 섭씨 온도라 해서 필드가 섭씨 온도일 필요는 없다.

◆ 연습문제 4.3.1. 위에서 사용한 클래스 Temperature를 작성하시오. 필드를 절대 온도로 설정하시오. main()을 작성하여 위 코드를 포함시키고 실행하시오.

캡슐화는 접근 제한으로 구현한다. 클래스의 필드나 생성자, 메서드에 대한 외부의 접근을 제한할 수 있다. 이 때 사용하는 키워드를 **접근 한정자**(access modifier)라 한다. 사용 가능한 접근 한정자는 다음과 같다.

private
: 선언된 클래스에서만 접근할 수 있다.

protected
: 선언된 클래스, 파생 클래스에서만 접근할 수 있다.

public
: 어디서든 접근할 수 있다.

접근 한정자는 필드, 생성자, 메서드의 선언 부분에 적는다. 다음 코드는 앞에서 사용한 Temperature를 일부 적은 것이다.

```
class Temperature
{
    private double kelvin;

    public Temperature(double celsius)
    {
        kelvin = celsius + 273.15;
```

```
        }

        public double getCelsius()
        {
            return kelvin  -  273.15;
        }
    }
```

필드 kelvin의 접근 한정자는 private이다. 따라서 그것이 선언된 클래스 Temperature에서만 사용할 수 있고 외부에서는 사용할 수 없다. 접근 한정자가 public인 생성자 Temperature()와 메서드 getCelsius()는 클래스 Temperature의 내부, 외부 가릴 것 없이 어디서든 사용 가능하다.

◆ 연습문제 4.3.2. 클래스 Temperature에 두 메서드 getFahrenheit()와 getKelvin()을 추가하시오. 접근 한정자 public을 붙이시오.

외부에서 클래스의 멤버에 접근하는 것을 제한하여 보호하는 것을 **캡슐화**(encapsulation)라 한다. 필드나 메서드를 용도에 맞게 사용하도록 제한하며 그 외에 다른 용도로 사용하는 것을 방지한다. 여러 사람이 프로그램을 개발하는 경우 클래스를 작성자의 의도대로 사용하도록 제한할 수 있다.

필드나 메서드를 선언할 때 이름 앞에 **접근 한정자**(access modifier)를 붙여서 접근을 제한한다. C#에서 사용할 수 있는 접근 한정자는 다음과 같다.

private
: 선언된 클래스에서만 접근할 수 있다.

protected
: 선언된 클래스와 파생 클래스에서만 접근할 수 있다.

public
: 어디서든 접근할 수 있다.

4.3.1 public. 접근 한정자 public을 붙여 선언한 필드, 생성자, 메서드는 어디서든 접근할 수 있다.

일반적으로 필드는 private으로 선언하여 외부에 공개하지 않는다. 또, 생성자와 메서드는 보통 public으로 선언한다. 용도에 따라서 다른 접근 한정자를 붙이기도 하지만 특별한 경우이다. 예제 4.1.1의 클래스 Rectangle은

```
    class Rectangle
    {
        private double width;
        private double height;

        public Rectangle(double width, double height)
        {
        }

        public double Area()
        {
        }
    }
```

로 정의하는 것이 일반적이다.

4.3.2 private과 프로퍼티. 접근 한정자 private을 붙여 선언한 필드, 생성자, 메서드는 그것이 선언된 클래스에서만 접근할 수 있다. 다른 클래스에서는 직접 접근할 수 없다. 클래스

```
class Person {
    private string name;
}
```

의 필드 name은 다른 클래스에서 접근할 수 없다. Person에서만 사용할 수 있다.

```
class Demo {
    static void Main() {
        Person p = new Person();
        p.name = "장 길산";     // 오류: private 필드에 접근
```

필드는 private으로 선언하는 것이 일반적이다. 그렇다면 필드의 값을 바꾸거나 필드에 설정된 값을 불러오려면 어떻게 해야 할까. 이 때 **프로퍼티**(property)를 사용한다. 클래스 Person의 필드 name에 접근하는 통로로 사용할 프로퍼티 Name은

```
class Person
{
    private string name; // 필드
    public string Name   // 프로퍼티
    {
        set { name = value; } // 셋터
        get { return name; }  // 겟터
    }
}
```

와 같은 형식으로 정의한다. 설명은 잠시 미루고 이 프로퍼티를 사용해보자.

```
Person p = new Person();
p.Name = "장길산";
```

을 실행하면 set 구역의 코드가 실행되어 name의 값이 설정된다. 이 때 value의 값은 "장길산"이며 따라서 name 역시 같은 값이다. 필드 name에 저장된 값을 출력하려면

```
Console.WriteLine(p.Name);
```

으로 입력한다. 이 때에는 get 구역의 코드가 실행된다. Name의 자료 형식은

```
return name;
```

이 반환하는 자료의 형식이면서

```
name = value;
```

의 value의 자료 형식이어야 한다. 따라서 name과 같아야 한다.

프로퍼티는 set 구역인 셋터(setter)와 get 구역인 겟터(getter)로 이루어진다. 셋터는 필드의 값을 설정하고 겟터는 설정된 값을 가져온다. 필요에 따라 둘 중 하나만

정의해도 된다. 프로퍼티의 이름은 파스칼 표기법으로 적을 것을 권장한다(3.3). 곧, 각 단어의 첫 글자를 대문자로 적을 것을 권장한다. 5.2에서 프로퍼티에 대하여 자세히 설명한다.

예제 4.3.1 Property Demo

```
using System;

class Person
{
    private string name;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
}

class Program
{
    static void Main()
    {
        Person p = new Person();
        p.Name = "장길산";
        Console.WriteLine(p.Name);
    }
}
```

출력

```
장길산
```

◆ 연습문제 4.3.3. 예제 4.3.1의 코드

```
public string Name
```

을

```
private string Name
```

으로 바꾸면 어떤 문제가 발생하는가?

◆ 연습문제 4.3.4. 클래스

```
class Rectangle
{
    public double width;
    public double height;
}
```

의 두 필드 width와 height에 대응되는 두 프로퍼티를 각각 작성하시오.

4.3.3 protected. 접근 한정자 protected를 붙여 선언한 필드, 생성자, 메서드는 선언된 클래스, 파생 클래스, 같은 패키지에 있는 클래스에서만 직접 접근할 수 있다. private보다 접근 범위가 넓으며 public보다 좁다. 클래스 Car

```
class Car
{
    protected int speed
}
```

의 필드 speed는 protected로 선언되어 있다. 이 필드는 Car의 파생 클래스

```
class SportsCar : Car
{
    public void accelerate()
    {
        speed += 100;
    }
}
```

에서 직접 접근할 수 있다.

예제 4.3.2 Protected Demo

```
using System;

class Car
{
    protected int speed;
    public int Speed
    {
        get { return speed; }
    }

    public Car()
    {
        speed = 0;
    }
}

class SportsCar : Car
{
    private bool turbo;
    public bool Turbo
    {
        set { turbo = value; }
    }

    public SportsCar()
    {
        turbo = false;
    }

    public void Accelerate()
    {
        if (turbo)
            speed += 100;
        else
            speed += 50;
    }
}
```

```
class Program
{
    static void Main()
    {
        SportsCar sc = new SportsCar();
        sc.Turbo = true;
        sc.Accelerate();
        Console.WriteLine(sc.Speed);
    }
}
```

출력

```
100
```

클래스 Car의 필드 speed에 붙은 접근 한정자가 protected이므로 파생 클래스 SportsCar에서 접근할 수 있다. 메서드 Accelerate()에서 그 값을 변화시킬 수 있는 것이다.

◆ 연습문제 4.3.5. 예제 4.3.2에서 정의한 클래스 Car의 프로퍼티 Speed에 set 구역을 추가하시오. 또, SportsCar의 Turbo에 get 구역을 추가하시오.

4.4 다형성

개미 한 마리는 개미이기도 하지만 곤충이기도 하고 동물이기도 하다. 동사 '가다' 역시 다양한 형태로 표현된다. 걸어서 가는 것도 나타내고 날아서 가는 것도 나타낸다. 한 요소가 다양한 형태로 표현되는 것이다. 객체 지향 프로그래밍에서 객체나 메서드는 상황에 따라 다른 형식으로 표현된다. 이처럼 하나의 대상이 여러 형식으로 표현되는 것을 **다형성**(polymorphism)이라 한다.

4.4.1 객체의 형식. 파생 클래스의 인스턴스는 기본 클래스의 인스턴스이기도 하다. 따라서 파생 클래스의 인스턴스는 기본 클래스 형식의 변수에 의하여 참조될 수 있다. 곤충을 나타내는 클래스

```
class Insect
```

와 개미를 나타내는 파생 클래스

```
class Ant extends Insect
```

에 대하여 Ant 객체는 Insect 객체이기도 하다. 따라서

```
Insect insect = new Ant();
```

와 같이 대입할 수 있다. '이 개미는 곤충이다'를 코드로 표현한 것이다.

예제 4.4.1 Reference Demo

```
using System;

class Insect
{
    private bool winged;
    public bool Winged
    {
        get { return winged; }
    }

    public Insect(bool winged)
    {
        this.winged = winged;
    }
}

class Ant : Insect
{
    private int size;
    public int Size
    {
        get { return size; }
    }

    public Ant(bool winged, int size) : base(winged)
    {
        this.size = size;
```

```
        }
    }

    class HoneyBee : Insect
    {
        public HoneyBee() : base(true)
        {
        }
    }

    class ReferenceDemo
    {
        static void Main()
        {
            Insect in1 = new Ant(false, 11);
            Console.WriteLine(in1.Winged);

            Ant ant = (Ant)in1;
            Console.WriteLine(ant.Size);

            Insect in2 = new HoneyBee();
            Console.WriteLine(in2.Winged);
        }
    }
```

```
출력
    False
    11
    True
```

파생 클래스의 인스턴스를 기본 클래스 형식의 변수에 대입하는 것이 가능하다. 역으로 기본 클래스 형식의 변수가 참조하는 객체를 파생 클래스 형식 변수에 대입하려면 형식 변환이 필요하다. 예제 4.4.1에서 Insect 형식 변수 in1은 Ant 객체를 참조하므로

```
Ant ant = (Ant) in1;
```

로 대입할 수 있다. Insect 형식 변수 in1이 Ant 객체를 참조하지만 Ant의 메서드나 프로퍼티를 호출할 수는 없다.

```
Console.WriteLine(in1.Size); // 오류
```

Insect의 메서드나 프로퍼티만 호출할 수 있다. 참조하는 객체가 아니라 변수의 자료형식에 따라 접근할 수 있는 필드와 메서드, 프로퍼티가 결정된다.

키워드 instanceof로 객체가 특정 클래스의 인스턴스인지 알아볼 수 있다.

```
x is ClassA
```

는 x가 ClassA의 인스턴스이면 true이고 그렇지 않으면 false이다. 예제 4.4.1에

```
if (in1 is Ant)
    Console.WriteLine("Ant의 인스턴스");
```

를 추가하여 결과를 살펴보기 바란다.

◆ 연습문제 4.4.1. 예제 4.4.1에 in2가 Ant의 인스턴스인지 판별하는 코드를 추가시오.

참조하는 객체가 없다는 것을 표현하기 위하여 키워드 null을 사용한다.

```
Foo f = null;
```

변수 f는 참조하는 객체가 없다. 따라서 필드에 접근하거나 메서드를 호출할 수 없다.

예제 4.4.2 Null Demo

```
using System;

class NullDemo
{
    static void Print(string str)
    {
        if (str == null)
            Console.WriteLine("null");
        else if ("".Equals(str))
            Console.WriteLine("빈 문자열");
        else
            Console.WriteLine(str);
    }

    static void Main()
    {
        Print(null);
        Print("");
        Print("산은 높고 물은 깊어");

        NullDemo demo = null;
        Console.WriteLine(demo is NullDemo);
    }
}
```

```
출력
    null
    빈 문자열
    산은 높고 물은 깊어
    False
```

클래스 string은 메서드 Equals()를 포함하고 있다. 빈 문자열 ""도 string 객체이므로 이 메서드를 호출할 수 있다.

4.4.2 메서드 오버로딩. 24 비트 RGB 색상은 세 성분(red, greed, blue)으로 표현된다. 각 성분은 빛의 세기를 나타내며 0 이상 255 이하의 정수를 값으로 가진다. 각 성분을 255로 나누어 0 이상 1 이하의 실수 성분으로 표현하기도 한다. 이와 같은 정보를 바탕으로 클래스 Color를 만들어보자. 세 int 형식 필드 red, green, blue를 포함시키자. 빛의 세기가 정수인 경우와 실수인 경우가 있으므로 두 메서드 setRGBInt()와 setRGBDouble()을 포함시키자. 코드로 적으면 다음과 같다.

```
class Color
```

```
{
    private int red, green, blue;

    public void setRGBInt(int red, int green, int blue)
    {
    }

    public void setRGBDouble(double red, double green, double blue)
    {
    }
}
```

이 클래스의 두 메서드는 기능이 같다. 따라서 이들을 같은 이름으로 정의할 수 있으면 좋을 것이다. 각각

```
public void setRGB(int red, int green, int blue)
```

와

```
public void setRGB(double red, double green, double blue)
```

로 정의하고 싶은 것이다. 실제로 이것이 허용된다.

한 클래스에 이름이 같은 메서드가 둘 이상 존재하는 것을 **메서드 오버로딩**(method overloading)이라 한다. 일반적으로 기능이 같은 메서드를 같은 이름으로 정의한다. 오버로드딩된 메서드는 매개 변수의 개수와 자료 형식으로 구별한다. 호출할 때 대입되는 인수에 따라 가장 적절한 것이 선택된다.

예제 4.4.3 Color

```
using System;

public class Color
{
    private int red, green, blue;
    public int Red
    {
        get { return red; }
    }

    public void setRGB(int red, int green, int blue)
    {
        this.red = red;
        this.green = green;
        this.blue = blue;
    }

    public void setRGB(double red, double green, double blue)
    {
        this.red = (int)(255 * red);
        this.green = (int)(255 * green);
        this.blue = (int)(255 * blue);
    }

    static void Main()
    {
```

```
            Color color = new Color();
            color.setRGB(0.5, 0.0, 0.25);
            Console.WriteLine(color.Red);
            color.setRGB(127, 0, 63);
            Console.WriteLine(color.Red);
        }
    }
```

```
출력
    127
    127
```

위 예제에서

```
color.setRGB(0.5, 0.0, 0.25);
```

는 매개 변수가 double 형식인 메서드를 호출하고

```
color.setRGB(127, 0, 63);
```

는 매개 변수가 int 형식인 메서드를 호출한다.

◆ 연습문제 4.4.3. 항해 일지를 나타내는 클래스

```
class Log
{
    private string time; // yyyy-mm-dd 형식
}
```

에 각각 다음과 같이 호출하는 오버로딩된 두 메서드 setTime()을 추가하시오.

```
Log log = new Log();
log.setTime("1990-12-08");
log.setTime("1990", "12", "08");
```

프로퍼티 Time을 추가하고 Main()에서 time을 출력하여 결과를 확인하시오.

◆ 연습문제 4.4.4. 어느 모임의 회원 명부를 나타내는 클래스

```
class MemberList
{
    private int[] id;
    private string[] name;
    private string[] phone;

    public MemberList()
    {
        id = new int[] { 101, 102, 103 };
        name = new string[] { "장길산", "홍길동", "임꺽정" };
        phone = new string[] { "100-0001", "101-1234", "123-0123" };
    }
}
```

에 한 회원의 전화번호를 출력하는 오버로딩된 두 메서드를 추가하시오. 각각 아이디와 이름을 대입하도록 작성하시오.

정적 메서드도 오버로딩할 수 있다. 한 클래스에 이름이 같은 정적 메서드를 둘 이상 정의하는 것을 말한다. 3.3.3에서 설명하였다.

◆ 연습문제 4.4.5. 클래스

```
class Math
```

에 두 수 중 큰 수를 반환하는 정적 메서드 max()를 자료 형식에 따라 둘 이상 작성하시오.

메서드 오버로딩은 매우 유용하다. C#이 제공하는 메서드 중 많은 수가 오버로딩되어 있다. 예를 들어, 메서드 Console.WriteLine()은

```
Console.WriteLine();
Console.WriteLine(1);
Console.WriteLine("산은 높고");
```

와 같이 매개 변수에 따라 여럿으로 오버로딩되어 있다.

4.4.3 생성자 오버로딩. 한 클래스가 생성자를 둘 이상 가지는 것을 **생성자 오버로딩** (constructor overloading)이라 한다. 메서드와 마찬가지로 매개 변수의 개수나 자료 형식이 달라야 한다.

다음 예제는 연습문제 4.4.3의 클래스 Log에 생성자와 메서드를 추가한 것이다.

예제 4.4.4 Log

```
using System;

public class Log
{
    private string time;
    public string Time
    {
        get { return time; }
    }

    public Log(string time)
    {
        this.time = time;
    }

    public Log(int year, int month, int date)
    {
        time = string.Format("{0,4:0000}-{1,2:00}-{2,2:00}",
                year, month, date);
    }

    static void Main()
    {
        Log log = new Log("1900-12-09");
        Console.WriteLine(log.Time);
        log = new Log(1900, 12, 9);
        Console.WriteLine(log.Time);
    }
}
```

출력

```
1900-12-09
```

```
        1900-12-09
```

string.Format()에 출력되는 숫자의 자릿수를 지정할 수 있다. 예를 들어,

```
    string str = string.Format("{0,5}", 123);
```

는 123을 다섯 자리에 맞춘다. 앞에 띄어쓰기 둘을 추가하여 " 123"이 된다. 띄어쓰기를 0으로 채우려면

```
    string str = string.Format("{0,5:00000}", 123);
```

으로 적는다. 따라서 str은 "00123"이 된다.

◆ 연습문제 4.4.6. 예제 4.4.4의 클래스 Log에 기본 생성자를 추가하시오.

◆ 연습문제 4.4.7. 예제 4.4.3의 클래스 Color에 세 생성자를 추가하시오.

4.4.3 메서드 오버라이딩. 기본 클래스에 정의된 메서드를 파생 클래스에서 다시 정의하는 것을 **메서드 오버라이딩**(overriding)이라 한다. 반환 형식, 메서드 이름, 매개 변수의 개수와 자료 형식이 일치하는 경우만 해당된다.

클래스

```
    class Car
    {
        public void Accelerate()
        {
        }
    }
```

과 파생 클래스

```
    class SportsCar : Car
    {
        public new void Accelerate()
        {
        }
    }
```

에는 메서드 Accelerate()이 정의되어 있다. 기본 클래스에 정의된 메서드가 파생 클래스에도 정의된 것이다. 이 때, new는 없어도 되지만 적어주는 것을 권장한다. SportsCar 객체

```
    SportsCar sc;
```

에 대하여

```
    sc.Accelerate();
```

를 호출하면 SportsCar의 accelerate()가 호출된다.

예제 4.4.5 Overriding Demo1

```
    using System;
```

```
class Car
{
    protected int speed;
    public int Speed
    {
        get { return speed; }
    }

    public Car()
    {
        speed = 0;
    }

    public void Accelerate()
    {
        speed += 50;
    }
}

class SportsCar : Car
{
    private bool turbo;

    public SportsCar(bool turbo)
    {
        this.turbo = turbo;
    }
    public new void Accelerate()
    {
        if (turbo)
            speed += 100;
        else
            base.Accelerate();
    }
}

class Program
{
    static void Main()
    {
        SportsCar sc = new SportsCar(true);
        sc.Accelerate();
        Console.WriteLine(sc.Speed);
    }
}
```

출력

```
100
```

키워드 base는 기본 클래스의 메서드를 호출할 때도 사용된다. 위 예제에서

```
base.Accelerate();
```

는 Car의 메서드 Accelerate()를 호출한다.

◆ 연습문제 4.4.6. 예제 4.4.5의

```
SportsCar sc = new SportsCar(true);
```

에 true 대신 false를 대입하여 실행하시오.

◆ 연습문제 4.4.7. 클래스

```
class Foo
{
    public string GetClassName()
    {
        return "Foo";
    }
}
```

를 상속하는 클래스를 만들고 메서드 GetClassName()을 오버라이딩하시오. 메서드에서는 클래스 이름인 "Bar"를 반환하시오.

◆ 연습문제 4.4.8. 기본 클래스 Glass와 파생 클래스 ColoredGlass를 작성하고 두 클래스에 메서드 GetPrice()를 정의하시오. ColoredGlass의 GetPrice()가 Glass의 메서드를 오버라이딩하도록 하시오.

◆ 연습문제 4.4.9. 예제 4.4.3의 Main()에

```
Car c = sc;
c.Accelerate();
Console.WriteLine(c.Speed);
```

를 추가하여 실행하고 어떤 메서드가 호출되는지 확인하시오.

메서드 오버라이딩은 여러 파생 클래스를 동시에 다룰 때 그 위력을 발휘한다. 다시 한번 곤충을 생각해보자. 클래스 Insect를 기본 클래스로 하고 두 파생 클래스 Ant와 HoneyBee를 만들었다고 하자. Insect에 이동하는 메서드 Move()를 정의하고 Ant와 HoneyBee에서 각각 오버라이딩하였다. 개미는 기어가고 꿀벌은 날아가도록 정의하였다고 하자. 배열

```
Insect[] insects = { new Ant(), new HoneyBee() };
```

을 정의하고 insects[0].Move()를 호출하였다. insects[0]가 Ant 객체이므로 기어가는 것이 온당하다. 그런데 예제 4.4.5와 같이 오버라이딩하면 Insect의 Move()가 호출된다. 누구든 이것을 기대하지는 않았을 것이다.

여기서 두 키워드 virtual과 override를 사용한다. 기본 클래스의 Move()에는 virtual을, 파생 클래스의 Move()에는 override를 붙인다.

```
class Insect
{
    public void virtual Move()
    {
    }
}

class Ant
{
    public void override Move()
    {
```

```
    }
}

class HoneyBee
{
    public void override Move()
    {
    }
}
```

이렇게 정의하고 insects[0].Move()를 호출하면 Ant의 Move()가 실행된다.

다음 예제는 게임에 등장하는 유닛을 어떻게 표현하는지 보여준다.

예제 4.4.4 Overriding Demo2

```
using System;

class Unit
{
    public virtual void Attack()
    {
        Console.WriteLine("공격 방법을 알 수 없다.");
    }
}

class Fighter : Unit
{
    public override void Attack()
    {
        Console.WriteLine("무기로 공격한다.");
    }
}

class Wizard : Unit
{
    public override void Attack()
    {
        Console.WriteLine("마법으로 공격한다.");
    }
}

class Program
{
    static void Main()
    {
        Unit[] units = new Unit[3];
        units[0] = new Unit();
        units[1] = new Fighter();
        units[2] = new Wizard();

        foreach (Unit unit in units)
            unit.Attack();
    }
}
```

출력

```
공격 방법을 알 수 없다.
```

```
        무기로 공격한다.
        마법으로 공격한다.
```

units은 클래스 Unit의 배열이지만 실제로는 Fighter와 Wizard를 포함하고 있다. Fighter와 Wizard는 Unit의 파생 클래스이므로 이렇게 대입해도 오류는 없다.

```
    unit.Attack();
```

으로 Unit에 정의된 메서드를 호출하면 어떤 메서드가 실행되느냐 하는 것이 문제이다. 위 예제와 같이 두 키워드 virtual과 override를 사용하면 unit의 실제 클래스의 메서드 Attack()이 실행된다. 곧 unit이 Unit 객체이면 Unit의 Attack(), Fighter 객체이면 Fighter의 Attack(), Wizard 객체이면 Wizard의 Attack()이 실행된다. virtual과 override 중 하나라도 빠지면 위와 같은 결과가 나오지 않는다. 만약 virtual만 빠진다면 오류가 출력된다.

◆ 연습문제 4.4.10. 예제 4.4.4에 궁수 클래스 Archer를 추가하고 배열 units의 요소로 사용하시오.

4.4.4 Object. C#의 클래스는 System.Object를 상속한다. 클래스의 기본 클래스를 추적하면 마지막에는 Object에 도달한다. 또 클래스를 정의할 때 기본 클래스를 명시하지 않으면 자동적으로 Object가 기본 클래스가 된다.

클래스 Object의 메서드 ToString()은 반환 형식이 string이다. 객체를 string 형식으로 변환할 때 자동으로 호출된다. 예를 들어

```
    Food food = new Food();
    string str = "객체 " + food;
```

를 실행할 때 food.ToString()이 호출되어

```
    string str = "객체 " + food.ToString();
```

과 동일한 결과를 얻는다. 또

```
    Console.WriteLine(food);
```

와 같이 객체를 출력할 때도 호출된다.

메서드 ToString()을 오버라이딩하면 객체를 문자열로 변환할 때 오버라이딩된 메서드가 호출된다. 이 때 다음 코드와 같이 ToString()에 키워드 override를 붙여야 한다.

예제 4.4.5 Object Demo

```
class Person
{
    private string name;

    public Person(string name)
    {
```

```
        this.name = name;
    }

    public override string ToString()
    {
        return "이름: " + name;
    }

    static void Main()
    {
        Person p = new Person("장길산");
        Console.WriteLine(p);
    }
}
```

출력
```
이름: 장길산
```

◆ 연습문제 4.4.11. 예제 4.4.5의

```
public override string ToString()
```

에서 override를 제거하고 결과를 확인하시오.

◆ 연습문제 4.4.12. 예제 4.4.5의 클래스 Person에 필드 age를 추가하시오. name과 age를 모두 포함하는 문자열을 반환하도록 메서드 ToString()을 수정하시오.

◆ 연습문제 4.4.13. 예제 4.4.4에서 정의한 클래스에 모두 ToString()을 오버라이딩하여 추가하시오.

제 5 장 객체 지향 프로그래밍 II

인스턴스 필드와 인스턴스 메서드는 객체의 특성을 기술한다. 클래스에는 그 외에도 클래스의 특성을 기술하는 정적 필드와 정적 메서드가 포함될 수 있다.

제 4 장에서 필드에 접근하는 통로로 프로퍼티를 정의하였다. 프로퍼티는 다양한 기능을 하도록 정의할 수 있다. 객체를 배열처럼 사용할 수 있는 도구로 인덱서를 제공한다.

C#에서는 연산자를 오버로딩할 수 있다. 예를 들어 객체의 덧셈을 정의할 수 있다.

광범위한 대상들은 한 클래스로 나타낼 수 없다. 이 때 추상 클래스를 도입하여 모호하게 표현하는 기법이 가능하다. 인터페이스는 객체의 한 면을 표현하는 기법이다.

5.1 정적 멤버

객체, 곧 클래스의 인스턴스를 구성하는 자료는 필드에 저장된다. 메서드는 객체의 행위를 기술한다. 객체를 강조하기 위하여 이들을 각각 인스턴스 필드와 인스턴스 메서드라 부르기도 한다. 객체를 구성하지는 않지만 클래스와 관련된 자료가 있을 수 있다. 객체에 적용되지는 않지만 클래스와 관련된 메서드 역시 있을 수 있다. 이들을 정의하는 도구로 정적 필드와 정적 메서드를 제공한다. 이 둘을 **정적 멤버**(static member)라 한다.

5.1.1 정적 필드. 벽돌 깨기라는 오래된 게임이 있다. 크기가 모두 같은 벽돌이 여럿 배치된 상태로 시작된다. 이 게임에서 벽돌을 클래스로 나타낸다고 하자. 한 모서리의 x 좌표, y 좌표, 폭 그리고 높이가 필드로 선언될 것 같다. 그런데 여기서 다시 생각해 볼 점이 있다. 벽돌의 폭이나 높이는 모두 같다. 따라서 모든 객체가 그것을 필드로 가지고 있을 필요가 없을 것이다.

보기를 하나 더 보자. 카드 게임에서 카드 객체는 무늬와 숫자로 구성된다. 카드를 클래스로 정의하면 이들을 필드로 가질 것이다. 그런데 생성된 카드의 개수가 필요하다면 어떻게 해야 할까? 카드 개수가 카드 객체의 필드일까?

필드는 인스턴스 필드와 정적 필드로 나뉜다. 인스턴스 필드는 객체, 곧 인스턴스를 구성하는 자료를 저장하며, 일반적으로 각 객체마다 다른 값을 가진다. 지금까지 사용했던 필드는 모두 인스턴스 필드이다.

클래스 자신과 관련된 자료를 저장하는 필드(변수)를 **정적 필드**(static field)라 한다. **클래스 변수**(class variable)라고도 한다. 객체의 구성 요소는 아니지만 클래스의 특성을 나타내는 자료가 저장된다. 예를 들어 벽돌의 폭이나 높이는 모든 객체가 동일한 값으로 공유하므로 객체의 특성이 아니라 클래스의 특성이다. 생성된 카드 객체의 개수도 카드라는 클래스의 특성이다. 이들을 정적 필드로 정의한다. 정적 필드는 키워드 static을 붙여서 선언한다.

```
class Brick
{
    static int width;
```

정적 필드에 접근하려면 클래스 이름에 점을 찍고 필드 이름을 붙여 적는다.

```
Brick.width = 20;
```

정의된 클래스에서 사용할 때는 클래스 이름을 붙이지 않아도 된다.

예제 5.1.1 Static Field Demo

```
using System;

class Brick
{
    public static int width;
```

```
        public static int height;

        private int x;
        private int y;

        public Brick(int x, int y)
        {
            this.x = x;
            this.y = y;
        }

        public void Draw()
        {
            Console.WriteLine("위치: ({0}, {1}), 폭: {2}, 높이: {3}",
                x, y, width, height);
        }
    }

    class Program
    {
        static void Main()
        {
            Brick.width = 20;
            Brick.height = 12;
            Brick brick = new Brick(0, 0);
            brick.Draw();
        }
    }
```

출력

```
위치: (0, 0), 폭: 20, 높이: 12
```

◆ 연습문제 5.1.1. 예제 5.1.1의 Main()에 Brick의 width를 출력하는 문장을 추가하시오.

값을 수정할 수 없는, 따라서 상수와 같은 역할을 하는, 정적 필드를 **상수 필드** (constant field)라 한다. const를 붙여 정의한다. 예제 5.1.1에서 width나 height가 변하지 않는다면 상수 필드로 정의해도 된다.

```
public const int Width = 20;
```

상수 필드의 이름은 파스칼 표기법으로 적을 것을 권장한다. 곧, 모든 단어의 첫 글자를 대문자로 적는다. 정적 필드와 마찬가지로 클래스 이름을 붙여 적는다.

```
Console.WriteLine(Brick.Width);
```

◆ 연습문제 5.1.2. 빛을 나타내는 클래스 Light는 가시광선의 최소 파장과 최대 파장을 상수 필드로 가진다. 클래스 Light를 작성하시오.

◇ 클래스 Math에는 원주율을 나타내는 상수 필드 PI가 정의되어 있다.

```
Console.WriteLine(Math.PI * radius * radius); // 원의 넓이
```

와 같이 원의 넓이를 출력할 수 있다.

5.1.2 정적 생성자. 예제 5.1.1의 클래스 Brick에 선언된 두 정적 필드 width와

height는 Main()에서 초기화된다. 그러나 규모가 큰 프로그램의 Main()에서 정적 필드를 초기화하는 것은 무모한 일이다. 클래스와 관련된 설정은 클래스에서 처리하는 것이 온당하다. 클래스를 초기화하는 기능이 필요한 것이다.

정적 생성자(static constructor)는 static을 붙여 정의한 생성자를 말한다.

```
class Brick
{
    static Brick()
    {
```

static 이외에 다른 한정자를 붙일 수 없으며 매개 변수도 가질 수 없다. 클래스의 생성자나 정적 메서드가 호출되기 전에 자동으로 실행되며 명시적으로 호출할 수 없다. 주로 정적 필드와 같은 클래스의 특성을 초기화한다. 텍스트 파일이나 그림 파일을 읽을 수 있으며 그 외에도 클래스와 관련된 일은 무엇이든 할 수 있다. 객체를 초기화하는 것이 생성자라면 클래스를 초기화하는 것은 정적 구역이라 하겠다.

예제 5.1.2 Static Block Demo

```
using System;

class Rice
{
    public const double CaloriePerGram = 1.5;
    public static double unit;

    static Rice() // 정적 생성자
    {
        unit = 210;
    }

    private double weight;

    public Rice(double weight)
    {
        this.weight = weight;
    }

    public double GetCalorie()
    {
        return CaloriePerGram * weight;
    }
}

class Program
{
    static void Main()
    {
        Rice rice = new Rice(Rice.unit);
        Console.WriteLine(rice.GetCalorie() + " kcal");
    }
}
```

출력

```
315.0 kcal
```

◆ 연습문제 5.1.3. 정적 생성자와 보통의 생성자 중 어느 것이 먼저 실행될까? 예제 5.1.2의 클래스 Rice의 생성자와 정적 생성자에 각각 문자열을 출력하는 문장을 삽입하여 어느 것이 먼저 실행되는지 확인하시오.

◆ 연습문제 5.1.4. 예제 5.1.2의 클래스 Rice의 정적 생성자에

```
weight = 210;
```

을 추가하면 오류가 발생한다. 이유는 무엇인가?

5.1.3 정적 메서드. 필드가 인스턴스 필드와 정적 필드로 나뉘듯, 메서드 역시 인스턴스 메서드와 정적 메서드로 나뉜다. 인스턴스 메서드는 그것을 호출하는 객체에 적용되는 함수이다. **정적 메서드**(static method)는 클래스에 적용되는 함수를 말한다. **클래스 메서드**(class method)라고도 한다. 클래스와 관련된 기능을 한다. 예를 들어, 정적 필드의 값을 변경할 수 있다. 일반적으로 객체와 직접적인 연관이 없다. 정적 메서드는 static을 붙여 정의하며 클래스 이름을 붙여 호출한다.

온도를 나타내는 클래스 Temperature에 섭씨 온도를 화씨 온도로 변환하는 정적 메서드를 정의해 보자.

```
class Temperature
{
    public static double CelsiusToFahrenheit(double celsius) {
        return celsius * 1.8 + 32;
    }
```

섭씨온도 36.5를 화씨온도로 변환하여 출력하려면

```
Console.WriteLine(Temperature.celsiusToFahrenheit(36.5));
```

로 적는다. 이 메서드는 Temperature 객체와 별로 연관이 없다. 단지 섭씨 온도를 화씨 온도로 바꾸어 반환할 뿐이다.

◇ 클래스 java.lang.Math에는 삼각함수, 지수함수, 로그함수 등이 정적 메서드로 정의되어 있다.

```
Console.WriteLine(Math.Cos(Math.PI/3));
```

와 같이 사용할 수 있다.

정적 메서드는 3.3에서 소개한 바 있다. 그 때에는 함수를 클래스와 상관없이 작성하였다. 함수 Main()도 정적 메서드이다. 프로그램 진입점이라는 것 외에 보통 정적 메서드와 다를 것이 없다.

예제 5.1.3 Static Member Demo

```
using System;

class Card
```

```
{
    public const int MaxNum = 13;
    private static int count = 0;
    private static Random rand;

    static Card()
    {
        rand = new Random();
    }

    public static int GetCount()
    {
        return count;
    }

    private int num;

    public Card()
    {
        count++;
        num = rand.Next(MaxNum) + 1;
    }

    public override string ToString()
    {
        return "" + num;
    }
}

class Program
{
    static void Main()
    {
        Card[] cards = { new Card(), new Card(), new Card() };

        Console.WriteLine("Card.MaxNum = " + Card.MaxNum);
        Console.WriteLine("Card.count = " + Card.GetCount());

        foreach (Card card in cards)
            Console.Write(card + " ");
    }
}
```

출력

```
Card.MaxNum = 13
Card.count = 3
4 5 10
```

정적 필드 count에는 생성된 Card 객체의 개수가 저장된다. 따라서 생성자에서만 수정할 수 있도록 private으로 정의하였다. 클래스 Random은 난수를 제공한다. 이 클래스의 메서드 Next()는 정수를 반환한다. Next(n)과 같이 매개변수를 적어주면 0 이상이고 n 미만인 정수를 반환한다.

◆ 연습문제 5.1.5. 예제 5.1.3의 클래스 Card에 정적 필드 max를 추가하시오. 생성된 카드 객체들의 필드 num의 최댓값을 max에 저장하시오. Main()에서 max를 출력하시오.

객체를 상수로 설정하려면 const 대신 readonly를 사용한다.

```
public readonly Card MyCard = new Card();
```

MyCard의 값은 변경할 수 없다.

◆ 연습문제 5.1.6. 클래스

```
public class Number {
    public readonly Number Zero = new Number(0.0);

    private double x;

    public Number(double x) {
        this.x = x;
    }

    public static Number add(Number n1, Number n2) {
        return new Number(n1.x + n2.x);
    }
}
```

는 실수를 표현한다. 실수 1을 나타내는 상수 필드 One을 추가하시오. 또, 뺄셈과 곱셈을 나타내는 정적 메서드를 각각 정의하시오.

5.2 프로퍼티

클래스의 필드를 private으로 정의하면 그 클래스의 외부에서 접근할 수 없고 protected로 정의하면 파생 클래스에서만 접근할 수 있다. **프로퍼티**(property)는 이 필드에 접근할 수 있는 통로를 제공한다(4.3.2).

프로퍼티는 필드처럼 사용되지만 필드가 아니며 메서드와 형태는 다르지만 기능은 같다. 클래스 Foo의 필드 bar에 접근하는 프로퍼티 Bar는

```
class Foo
{
    private int bar;
    public int Bar
    {
        set { bar = value; }
        get { return bar; }
    }
```

와 같은 형태이다. 셋터인 set 구역은 필드에 값을 대입하는 기능을 하며

```
Foo foo = new Foo();
foo.Bar = 23;
```

과 같이 사용된다. 단순히 형태만 보면 필드에 값을 대입하는 것처럼 보인다. 그러나 Bar는 필드가 아니다. 대입된 값 23은 **암시적 매개 변수** value로 전달되며 다시 필드 bar에 대입된다. 겟터인 get 구역은 값을 가져오는 기능을 하며

```
Console.WriteLine(foo.Bar)
```

와 같이 사용된다.

프로퍼티를 반드시 필드에 대응하여 만들어야 하는 것은 아니다. 필드 없이

```
class Foo
{
    public int Qux { get; set; }
```

와 같이 정의한 프로퍼티 Qux는 마치 변수 qux가 있는 것처럼 사용한다.

```
foo.Qux = 199;
Console.WriteLine(foo.Qux);
```

여러 필드의 값을 설정하거나 가져오는 프로퍼티도 정의할 수 있다. 색상을 나타내는 클래스 Color가 세 필드 red, green, blue를 가진다고 하자. 이 클래스에 세 필드의 값을 한 번에 설정하는 프로퍼티를 만들 수 있다.

```
class Color
{
    private int red, green, blue;
    public int[] RGB
    {
        set
        {
            red = value[0];
            green = value[1];
            blue = value[2];
```

```
            }
        }
```

이 프로퍼티는

```
    Color col = new Color();
    col.RGB = new int[] { 255, 255, 255 };
```

와 같이 사용한다.

◆ 연습문제 5.2.1. 위에서 정의한 프로퍼티 RGB에 get 구역을 추가하시오.

프로퍼티는 접근 한정자를 붙여 사용을 제한할 수 있다.

```
    private string Prop
    {
        get; set;
    }
```

와 같이 전체의 접근을 제한할 수 있으며

```
    public string Prop
    {
        get; // Prop의 접근 한정자인 public이 적용된다.
        private set;
    }
```

과 같이 get 구역과 set 구역에 대한 접근을 다르게 제한할 수도 있다. 이 때 get이나 set의 접근 한정자는 프로퍼티의 접근 한정자보다 제한적이어야 한다. 프로퍼티에 접근 한정자를 적어 주지 않으면 private으로 간주한다.

◆ 연습문제 5.2.2. 다음 코드는 어떤 문제가 있는가?

```
    private string Prop
    {
        public get;
    }
```

프로퍼티는 오버로딩할 수 있으며 오버라이딩할 수 있다. 메서드와 같이 키워드 abstract, virtual, override를 사용하여 속성을 변경할 수 있다.

```
    class Person
    {
        private string name;

        public virtual string Info
        {
            get { return "이름: " + name; }
        }
    }

    class Student : Person
    {
        private int id;

        public override string Info
        {
            get { return base.Info + " id: " + id; }
```

```
        }
    }
```

지금까지 소개한 프로퍼티들은 객체에 작용한다. 그런 의미에서 **인스턴스 프로퍼티**라고도 한다. 클래스에 작용하는 프로퍼티를 **정적 프로퍼티**(static property)라 한다. 정적 프로퍼티는 키워드 static을 붙여서 정의한다. 형태로 보면 정적 필드와 같으며 기능으로 보면 정적 메서드와 같다. 클래스 이름 뒤에 점을 찍고 프로퍼티 이름을 적어서 사용한다.

예제 5.2.1 Property Demo

```
using System;

class Stopwatch
{
    private static int timeout;

    private int minute;
    private int second;

    public Stopwatch(int minute, int second)
    {
        this.minute = minute;
        this.second = second;
    }

    public int TimeBySecond
    {
        set
        {
            minute = value / 60;
            second = value % 60;
        }
        get
        {
            return minute * 60 + second;
        }
    }

    public override string ToString()
    {
        return string.Format("{0}분 {1}초", minute, second);
    }

    public static int Timeout
    {
        set { timeout = value; }
        get { return timeout; }
    }
}

class Program
{
    static void Main()
    {
```

```
            Stopwatch.Timeout = 120;
            Console.WriteLine(Stopwatch.Timeout);

            Stopwatch watch = new Stopwatch(2, 18);
            Console.WriteLine(watch.TimeBySecond);

            watch.TimeBySecond = 102;
            Console.WriteLine(watch);
        }
    }
```

```
출력
    120
    138
    1분 42초
```

위 예제에서 Timeout은 정적 프로퍼티이다.

◆ 연습문제 5.2.3. 예제 5.1.1의 정적 필드 count의 값을 설정하고 불러오는 정적 프로퍼티를 작성하시오.

5.3 인덱서

문자열 str에 대하여 str[0]은 첫째 문자를 가져온다. 어떻게 이것이 가능할까? **인덱서**(indexer)는 객체를 배열처럼 사용할 수 있도록 한다. 대괄호([])에 인덱스를 대입하여 값을 가져올 수 있다. 인덱서를 사용하면 간결하고 이해하기 쉬운 코드를 작성할 수 있다.

인덱서는 키워드 this로 구현한다.

```
class List
{
    private string[] names;

    public string this[int n]
    {
        get { return names[n]; }
        set { names[n] = value; }
    }
}
```

에서

```
public string this[int n]
```

은 List가 인덱서로 정의되었음을 나타낸다. 인덱서에 접근하는 것은 형태 상으로 배열과 유사하나, 값을 대입할 때는 셋터가 호출되며 저장된 값을 불러올 때는 겟터가 호출된다.

```
List list = new List();
list[3] = "장길산";
```

과 같이 값을 대입한다. 또

```
Consol.WriteLine(list[3]);
```

과 같이 값을 불러온다.

예제 5.3.1 Indexer Demo1

```
using System;

class List
{
    private string[] names;

    public List(int size)
    {
        names = new string[size];
    }

    public string this[int n]        // 인덱서 정의
    {
        get { return names[n]; }
        set { names[n] = value; }
    }
```

```
        static void Main()
        {
            List list = new List(4);
            list[0] = "장길산";
            list[1] = "임꺽정";

            for (int i = 0; i < 2; i++)
                Console.WriteLine("list[{0}] = {1}", i, list[i]);
        }
    }
```

```
출력
list[0] = 장길산
list[1] = 임꺽정
```

인덱서의 대괄호 안에는 정수 뿐만 아니라 다른 자료형도 사용할 수 있다. 곧,

```
public int this[double x]
```

와 같이 정의하는 것도 가능하다. 인덱서는 오버로딩할 수 있다.

```
public double this[int x]
{
}
public int this[string s]
{
}
```

와 같이 두 인덱서를 동시에 정의할 수 있다.

◆ 연습문제 5.3.1. 회원 목록에 인덱서를 활용할 수 있을 것 같다. 이름, 아이디, 주소, 전화번호 등이 회원의 정보일 것이다. 적당한 상황을 설정하고 인덱서 오버로딩을 구현하시오.

5.4 연산자 오버로딩

두 객체의 덧셈이 필요한 프로그램을 작성한다고 하자. 덧셈은 메서드로 구현될 것이다. 그런데 메서드 대신 연산자 +를 쓰면 읽고 이해하기 쉬울 것이다. C#에서는 이것을 허용한다. 클래스의 메서드를 연산자로 표현하는 기법을 **연산자 오버로딩**(operator overloading)이라 한다. 기존의 연산자에 다른 의미를 부여하기 때문에 오버로딩이라는 표현을 쓴다.

5.4.1 연산자. 2 차원 벡터(vector)를 나타내는 클래스 Vector를 생각하자. 두 벡터의 덧셈을

```
class Vector
{
    public static Vector Add(Vector v, Vector w)
    {
```

와 같이 메서드로 정의하자. 세 벡터 u, v, w의 덧셈 u + v + w는

```
Vector sum = Vector.Add(Vector.Add(u, v), w)
```

와 같이 계산해야 한다. 코드가 복잡해 보이고 이해하는 것도 시간이 걸린다. 원래의 식 u + v + w를 그대로 쓸 수 있다면 좋을 것이다. 연산자를 오버로딩하면 그것이 가능하다.

```
class Vector
{
    public static Vector operator +(Vector v, Vector w)
    {
    }
}
```

와 같이 정의한 연산자 +를 사용한 코드는 다음과 같다.

```
Vector sum = u + v + w
```

매우 간단해졌다. 이것이 연산자 오버로딩이다.

연산자 오버로딩은 키워드 operator를 써서 정의한다. 정의하고자 하는 연산자 앞에 이 키워드를 써 준다. 단항 연산은 매개 변수가 하나여야 하고 이항연산은 매개 변수가 둘이어야 한다. 다음 예제는 두 연산자 +와 *를 오버로딩한다.

예제 5.4.1 Operator Overloading Demo

```
using System;

class Vector
{
    private double x;
    private double y;

    public Vector(double x, double y)
    {
```

```
            this.x = x;
            this.y = y;
        }

        public static Vector operator -(Vector v)
        {
            return new Vector(-v.x, -v.y);
        }

        public static Vector operator +(Vector v, Vector w)
        {
            return new Vector(v.x + w.x, v.y + w.y);
        }

        public static Vector operator *(double d, Vector v)
        {
            return new Vector(d * v.x, d * v.y);
        }

        public static double operator *(Vector v, Vector w)
        {
            return v.x * w.x + v.y * w.y;
        }

        public override string ToString()
        {
            return string.Format("({0}, {1})", x, y);
        }
    }

    class Program
    {
        static void Main()
        {
            Vector u = new Vector(1, -3);
            Vector v = new Vector(1, 0);
            Vector w = u + v;
            Console.WriteLine(w);
            Console.WriteLine(-w);
            Console.WriteLine(5.2 * u);
            Console.WriteLine(u * v);
        }
    }
```

출력

```
(2, -3)
(-2, 3)
(5.2, -15.6)
1
```

연산자 *는 두 가지 방법으로 오버로딩되었음을 알 수 있다. 피연산자(매개 변수)에 따라 둘 중 하나가 선택된다.

오버로딩 가능한 연산자는 다음과 같다.

종류	연산자
단항 연산자	+ - ! ~ ++ -- true false
산술 연산자	+ - * / %
관계 연산자	== != < > <= >=
비트 연산자	& ¦ ^ << >>

표 5.1 오버로딩 가능한 연산자.

오버로딩한 연산자의 사용 규칙 및 우선순위는 그 연산자의 고유 규칙 및 우선순위와 같다. ++ 연산자는 본래 규칙에 따라 ++x, x++와 같이 사용할 수 있다. 그러나 x~는 규칙에 없으므로 사용할 수는 없다.

◆ 연습문제 5.4.1. 뺄셈 연산자 - 를 오버로딩하여 Vector의 뺄셈을 정의하시오.

◆ 연습문제 5.4.2. 단항 연산자 ++를 오버로딩하시오. 객체 obj에 대하여 ++obj와 obj++의 차이점은 무엇인가?

5.4.2 형식 변환. 종이로 상자를 만든다고 하자. 클래스 Paper는 직사각형 종이를 나타낸다.

```
class Paper
{
```

상자는 클래스 Box는 상자를 나타낸다. 이 클래스에는 Paper 객체를 매개 변수로 가지는 생성자가 포함된다.

```
class Box
{
    public Box(Paper paper)
    {
```

두 클래스를 사용하여 코드

```
Paper paper = new Paper();
Box box = new Box(paper);
```

를 작성할 수 있다. 이 때 Box의 생성자를 사용하지 않고

```
Box box = (Box)paper;
```

와 같이 형식 변환으로 적을 수 있다면 편리할 것이다. 클래스 Box에

```
class Box
{
    public static explicit operator Box(Paper paper)
    {
```

와 같이 연산자를 만들어 주면 그것이 가능하다.

형식 변환을 직접 적어 주는 것을 **명시적 형식 변환**(explicit conversion)이라 한다. 객체의 명시적 형식 변환은 다음과 같은 꼴로 정의한다.

static explicit operator 클래스_이름(자료_형식 매개_변수)
명시적 형식 변환을 정의한다.

static explicit operator 구조체_이름(자료_형식 매개_변수)
명시적 형식 변환을 정의한다.

구조체는 제 6 장에서 다룬다.

예제 5.4.2 Explicit Conversion Demo

```
using System;

class Paper
{
    public int Width {get; set;}
    public int Height { get; set; }

    public Paper(int width, int height)
    {
        Width = width;
        Height = height;
    }
}

class Box
{
    public int Width { get; set; } // 가로
    public int Depth { get; set; } // 세로
    public int Height { get; set; } // 높이

    public Box(int width, int depth, int height)
    {
        Width = width;
        Depth = depth;
        Height = height;
    }

    public static explicit operator Box(Paper paper)
    {
        int height = paper.Width / 3;
        int width = paper.Width - 2 * height;
        int depth = paper.Height - 2 * height;
        return new Box(width, depth, height);
    }
}

class Program
{
    static void Main(string[] args)
    {
        Paper paper = new Paper(30, 40);
        Box box = (Box)paper;
        Console.WriteLine(box.Width);
    }
}
```

출력

```
10
```

예제 5.4.2의 형식 변환에서 (Box)를 제거할 수 있다.

```
Box box = paper;
```

이 때에는 클래스 Box에

```
class Box
{
    public static implicit operator Box(Paper paper)
    {
```

와 같이 연산자를 만들어 주어야 한다.

필요한 곳에서 자동으로 형식 변환을 수행하는 것을 **암시적 형식 변환**(implicit conversion)이라 한다. 형식 변환을 명시하지 않고 사용할 수 있다. 객체의 암시적 형식 변환은 다음과 같은 꼴로 정의한다.

static implicit operator 클래스_이름(매개_변수_자료형 매개_변수)
암시적 형식 변환을 정의한다.

static implicit operator 구조체_이름(매개_변수_자료형 매개_변수)
암시적 형식 변환을 정의한다.

◆ 연습문제 5.4.3. 예제 5.4.2의 명시적 형식 변환을 암시적 형식 변환으로 바꾸시오.

5.5. 추상 클래스

곤충을 표현하는 클래스를 작성한다고 하자. 대상마다 특성이 각기 달라 한 클래스로 표현할 수 없을 것 같다. 곤충들이 공통으로 가지는 필드는 일부에 불과하며 기술할 수 없는 메서드도 존재한다. 범위를 좁혀야 비로소 대상을 정확히 표현하는 클래스를 정의할 수 있다.

곤충과 같이 확정되지 않은 클래스를 **추상 클래스**(abstract class)라 한다. 대상들이 공통으로 가지는 요소들로 구성된다. 불완전한 설계도라고 보면 될 것이다. 추상 클래스는 키워드 abstract를 붙여 정의한다.

```
abstract class Insect
```

추상 클래스에는 아래에 설명할 추상 메서드나 추상 프로퍼티 또는 추상 인덱서가 포함되는 것이 일반적이다.

선언만 있고 본문이 없는 메서드를 **추상 메서드**(abstract method)라 한다. 메서드가 있다는 것은 알지만 기술할 수는 없을 때 사용한다. 예를 들어, 곤충을 그리는 메서드는 구체적으로 기술할 수 없다. 어떤 곤충을 그릴 것인가? 곤충을 표현하는 클래스에 대상을 그리는 메서드를 포함시킨다면 추상 메서드가 될 것이다. 추상 메서드 역시 키워드 abstract를 붙여 정의한다.

```
public abstract void Draw();
```

추상 메서드를 포함하는 클래스는 반드시 추상 클래스이어야 한다. 추상 프로퍼티나 추상 인덱서도 유사하게 정의된다. 추상 클래스를 상속하는 클래스에서 추상 메서드, 추상 프로퍼티 또는 추상 인덱서를 정의하려면 키워드 override를 사용해야 한다.

추상 클래스의 인스턴스는 직접 생성할 수 없다. new Insect()를 사용할 수 없는 것이다. 불완전한 설계도로 물건을 만들 수 없는 것과 같은 이치이다. 예를 들어 곤충 객체를 생성한다고 하자. 정보가 충분하지 않아 어떤 곤충을 생성할지 문제가 될 것이다. 추상 클래스의 인스턴스를 생성하려면 반드시 파생 클래스를 작성하고 그것의 인스턴스를 생성해야 한다. 파생 클래스에서는 추상 메서드, 추상 프로퍼티, 추상 인덱서를 오버라이딩하여 본문을 작성해야 한다.

```
class HoneyBee extends Insect
{
    public override void Draw()
    {
```

설계도를 완성하는 것이다.

예제 5.5.1 AbstractClassDemo1

```
using System;

abstract class Insect
{
    public double Length { get; }
```

```
        public Insect(double length)
        {
            Length = length;
        }

        public abstract bool Winged { get; }
        public abstract void Draw();
    }

    class HoneyBee : Insect
    {
        public HoneyBee(double length) : base(length)
        {
        }

        public override bool Winged
        {
            get { return true; }
        }

        public override void Draw()
        {
            Console.WriteLine("꿀벌: 날개 {0}, 몸 길이 {1}",
                    Winged ? "있음" : "없음",  Length);
        }
    }

    class Program
    {
        static void Main()
        {
            HoneyBee bee = new HoneyBee(1.2);
            bee.Draw();
        }
    }
```

출력

```
꿀벌: 날개 있음, 몸 길이 1.2
```

◆ 연습문제 5.5.1. 도형을 나타내는 추상 클래스 Shape은 각각 둘레와 넓이를 반환하는 두 추상 메서드 circumference()와 area()를 포함한다.

```
abstract class Shape {
    public abstract double circumference();
    public abstract double area();
}
```

원을 나타내는 파생 클래스 Circle을 만드시오.

다양한 유닛들이 등장하는 게임에서 유닛들의 기본 클래스 Unit을 정의한다고 하자. 이 클래스는 상대를 공격하는 메서드 Attack()을 가질 수 있다. 그러나 이 메서드를 정확히 기술할 수는 없다. 공격 방법이 유닛마다 다르기 때문이다. 따라서 Attack()은 추상 메서드가 되고 Unit은 추상 클래스가 된다.

다음 예제에서는 Unit에서 파생된 두 클래스 Fighter와 Wizard를 만든다. 두 클래스에서 Attack()의 본문을 각 클래스에 어울리도록 작성한다.

예제 5.5.2 Abstract Class Demo2

```
using System;

abstract class Unit
{
    public int Hp { get; set; }

    public Unit(int hp)
    {
        Hp = hp;
    }

    public abstract void Attack(Unit target);
}

class Fighter : Unit
{
    public int Ap { get; set; }

    public Fighter(int hp, int ap) : base(hp)
    {
        Ap = ap;
    }

    public override void Attack(Unit target)
    {
        target.Hp -= Ap;
    }
}

class Wizard : Unit
{
    public int Mp { get; set; }

    public Wizard(int hp, int mp) : base(hp)
    {
        Mp = mp;
    }

    public override void Attack(Unit target)
    {
        target.Hp -= Mp;
    }
}

class Program
{
    static void Main()
    {
        Unit[] units = new Unit[3];
        units[0] = new Fighter(1100, 90);
        units[1] = new Wizard(420, 200);
        units[2] = new Wizard(410, 200);

        Unit target = new Fighter(1200, 85);
        foreach (Unit u in units)
```

```
            u.Attack(target);
            Console.WriteLine(target.Hp);
        }
    }
```

출력

```
710
```

◆ 연습문제 5.5.2. 예제 5.5.2에 궁수를 나타내는 클래스 Archer를 추가하시오.

◆ 연습문제 5.5.3. 덧셈을 나타내는 클래스 Addition은 다음과 같다.

```
class Addition extends Operation {
    public Addition() {
        super("덧셈", '+');
    }

    public override double Compute(double a, double b) {
        return a + b;
    }
}
```

추상 클래스 Operation을 작성하시오. 또, 곱셈을 나타내는 클래스 Multiplication을 Operation의 파생 클래스로 정의하시오.

파생 클래스에서 정의되지 않은 추상 메서드는 파생 클래스에서도 여전히 추상 메서드로 남는다. 이 경우 파생 클래스도 추상 클래스가 되어야 한다. 설계도가 완성되지 않았기 때문이다.

◇ 예제 5.5.2에서 Unit을 추상 클래스로 만드는 이유는 무엇일까? 다음과 같이 abstract를 모두 빼고 Attack()을 아무 내용이 없이 작성하면 어떨까?

```
class Unit {
    public virtual void Attack(Unit target) {
    }
```

이 클래스를 기본 클래스로만 사용할 때는 추상 클래스와 다른 점이 없다. 다만 아무 의미도 없는 Unit 객체를 생성할 수 있다는 것이다. 추상 클래스는 이것을 막아 준다.

5.6 인터페이스

인터페이스는 접촉 통로 또는 창구라는 의미로 사용된다. 키보드, 마우스, 모니터와 같은 장치는 컴퓨터와 사람이 접촉하는 인터페이스이다. 그래픽 유저 인터페이스(GUI)와 같이 장치가 아니라 소프트웨어적 기능을 의미하는 경우도 있다.

자동차를 생각해 보자. 그것을 구입할 때는 가격이 중요한 요소가 된다. 그러나 운전을 할 때는 연료의 양이 중요하다. 가격에는 전혀 관심이 없다. 클래스로 표현한다면 구입할 때는 가격을 반환하는 메서드를 사용한다. 반면 운전을 할 때는 연료의 양을 반환하는 메서드가 필요하다. 객체의 일부 기능만 필요할 때 C#에서는 그것을 규정하는 방법으로 인터페이스를 사용한다.

클래스는 자료와 기능으로 객체를 제한한다. 어떤 자료로 구성되어 있는지 어떤 기능이 있는지 밝혀 적는다. 추상 클래스도 마찬가지이다. 반면 인터페이스는 특정 기능으로 객체를 제한한다. 필드는 상관하지 않고, 특정 메서드를 가지는지만 관심을 갖는다.

5.6.1 정의와 구현. '인쇄할 수 있는 것'을 생각해보자. 그것은 프린터로 출력할 수 있다. 무엇으로 이루어져 있든지 상관없이 인쇄할 수 있으면 된다. 코드로 구현했을 때 페이지를 반환하는 메서드 GetPage()만 포함하고 있으면 된다고 하자. 출력하는 것이 그림이라면 GetPage()는 그림으로 쪽을 구성하여 반환할 것이다. 문서라면 그에 맞는 것을 반환할 것이다. 이 메서드를 포함하는 것들을 Printable 객체라 하자. 그것은 다음과 같이 사용된다.

```
public void PrintPage(Printable p, int n)
{
    p.GetPage(n)을 프린터로 출력
}
```

여기서 메서드 GetPage()는 인쇄할 대상으로부터 각 쪽의 정보를 가져오는 통로이다. Printable은 메서드 GetPage()만 포함하고 있으면 된다. 이것을 표현해 보자.

인터페이스(interface)는 기능(메서드, 프로퍼티, 인덱서)만으로 객체를 규정하는 도구이다. 인스턴스 필드 없이 추상 메서드, 추상 프로퍼티 또는 추상 인덱서를 모아 놓은 것이다. 키워드 interface를 써서 정의한다. '인쇄할 수 있는 것'은

```
interface Printable
{
    String GetPage(int n);
```

와 같다. GetPage()는 n 쪽을 반환해야 하지만 편의상 문자열을 반환하도록 하였다. GetPage()는 본문이 없는 추상 메서드이다. 인터페이스에 포함되는 추상 메서드, 추상 프로퍼티, 추상 인덱서에는 키워드 abstract를 붙이지 않아도 된다. 인터페이스 이름은 클래스 이름과 마찬가지로 파스칼 표기법으로 적는다. 일반적으로 인터페이스는 이름 앞에 I를 붙여 인터페이스임을 드러낸다. 따라서 위에서 언급한 Printable도 IPrintable로 적기로 하자.

인터페이스는 추상 클래스와 마찬가지로 추상 메서드, 추상 프로퍼티 또는 추상 인덱서를 포함한다. 따라서 객체를 직접 생성할 수 없으며 그것을 상속하는 클래스의 인스턴스를 생성해야 한다. 그 클래스에서는 인터페이스에 선언된 메서드를 정의해야 한다. 다시 말하면 오버라이딩하여 본문을 작성해야 한다. 이것을 '인터페이스를 **구현**한다'고 한다. 인터페이스의 메서드를 클래스에서 오버라이딩하여 작성할 때는 키워드 override를 적지 않는다.

문서는 출력할 수 있다. 그것을 표현하기 위하여 IPrintable을 구현하는 클래스 Document를 정의해보자.

```
class Document : IPrintable
{
    private String[] pages;

    public String GetPage(int n)
    {
        // n 쪽 반환
    }
```

과 같이 정의한다. 메서드 GetPage(int n)에서는 n 쪽을 구성하여 반환해야 한다. 클래스 Printer를 작성하여 IPrintable 객체를 출력해보자.

예제 5.6.1 Interface Demo1

```
using System;

interface IPrintable
{
    String GetPage(int n);
}

class Document : IPrintable
{
    private String[] pages;

    public Document(String[] pages)
    {
        this.pages = pages;
    }

    public String GetPage(int n)
    {
        return n <= pages.Length ? pages[n - 1] : null;
    }
}

class Printer
{
    public void Print(IPrintable p)
    {
        String str;
        int i = 1;
        while ((str = p.GetPage(i++)) != null)
            Console.WriteLine(str);
```

```
        }
    }

    class Program
    {
        static void Main()
        {
            Document doc = new Document(new String[] { "1쪽", "2쪽" });
            new Printer().Print(doc);
        }
    }
```

```
출력
    1쪽
    2쪽
```

메서드 Print()에서는 GetPage()로 각 쪽을 가져와 인쇄한다. 따라서 인쇄하는 객체가 무엇인지 알 필요 없다. 접촉 통로인 메서드 GetPage()가 정의되어 있으면 된다. IPrintable을 구현하는 클래스의 인스턴스이기만 하면 되는 것이다.

◆ 연습문제 5.6.1. 예제 5.6.1에 그림을 나타내는 클래스 Figure를 추가하시오. 이 클래스에서 인터페이스 IPrintable을 구현하시오.

◆ 연습문제 5.6.2. 인터페이스

```
interface IAveragable
{
    double Average();
}
```

의 Average()는 평균을 반환한다. 이 인터페이스를 구현하는 두 클래스를 작성하시오. 한 클래스는 double의 배열을 필드를 가지며 Average()는 배열 요소들의 평균을 반환한다. 다른 클래스는 두 실수를 필드로 가지며 Average()는 두 실수의 평균을 반환한다.

C#의 클래스는 기본 클래스를 하나만 가질 수 있다. 둘 이상의 클래스에서 파생될 수 없으므로 상당한 제약이 따른다. 이 점을 보완하기 위한 수단으로 인터페이스를 사용할 수 있다.

```
interface IContainer { }

class Box { }

class Load : Box, IContainer
```

클래스 Load는 Box에서 파생된 클래스이며 인터페이스 IContainer를 구현한다. 물론 IContainer에 선언된 메서드를 모두 정의한다.

예제 5.6.2 InterfaceDemo2

```
using System;

interface IContainer
{
```

```
    int Room { get; } // 여유 공간
    void Add(int size); // size만큼 화물 추가
}

class Box
{
    public int Capacity { get; set; } // 전체 용량

    public Box(int capacity)
    {
        Capacity = capacity;
    }
}

class Load : Box, IContainer {
    public int NumPackets { get; set; } // packet 개수

    public Load(int capacity) : base(capacity)
    {
        NumPackets = 0;
    }

    public int Room
    {
        get { return Capacity - NumPackets; }
    }

    public void Add(int size)
    {
        NumPackets += size;
    }
}

class Program
{
    static void Main()
    {
        Load load = new Load(10);
        load.Add(7);
        Console.WriteLine("전체 공간: " + load.Capacity);
        Console.WriteLine("packet 개수: " + load.NumPackets);
        Console.WriteLine("여유 공간: " + load.Room);
    }
}
```

```
출력
    전체 공간: 10
    packet 개수: 7
    여유 공간: 3
```

◆ 연습문제 5.6.3. 예제 5.6.2의 IContainer에 메서드 Remove()를 추가하시오. Load에서 패킷을 제거하도록 이 메서드를 구현하시오.

◆ 연습문제 5.6.4. 일차 함수를 나타내는 클래스 LinearFunction은

```
class LinearFunction : IFunction
{
    private double a, b;
```

```
    public LinearFunction(double a, double b)
    {
        this.a = a;
        this.b = b;
    }

    public double Evaluate(double x) // IFunction에 선언된 메서드
    {
        return a * x + b;
    }
}
```

으로 정의된다. 메서드 Evaluate()가 선언된 인터페이스 IFunction을 작성하시오. 또, 이차함수를 나타내는 클래스 QuadraticFunction을 IFunction을 구현하여 작성하시오.

◆ 연습문제 5.6.5. 인터페이스

```
interface IEnumerator
{
    bool MoveNext();
    int Next { get; }
}
```

를 구현하는 클래스 ArrayIterator를 작성하시오. 정수의 배열 arr에 대하여

```
ArrayIterator ai = new ArrayIterator(arr);
while (ai.MoveNext())
    Console.WriteLine(ai.Next);
```

가 arr의 모든 요소를 순서대로 출력하도록 하시오.

5.6.2 상속. 인터페이스를 상속하는 클래스에서 메서드 전체 또는 일부를 정의하지 않을 수도 있다. 이 경우 그것은 추상 클래스로 선언되어야 한다. 추상 메서드가 포함되므로 당연한 일일 것이다.

인터페이스를 상속하는 인터페이스를 만들 수 있다. 이것을 **파생 인터페이스**라 한다. 파생 인터페이스는 **기본 인터페이스**의 모든 메서드를 상속한다. 따라서 파생 인터페이스를 구현하는 클래스는 기본 인터페이스의 메서드까지 정의해야 한다.

클래스는 여러 인터페이스를 구현할 수 있다.

```
interface Interf1 { }

interface Interf2 : Interf1 { }

interface Interf3 : Interf1 { }

class MyClass1 { }

class MyClass2 : MyClass1, Interf2, Interf3
```

클래스 MyClass2는 세 인터페이스 Interf1, Interf2, Interf3가 포함하는 메서드를 모두 정의해야 한다. 세 인터페이스에 같은 메서드가 포함될 수 있으며 그럴 경우 그 메서드를 한 번만 정의하면 된다.

제 6 장 자료 형식 II

제 2 장에서는 C#이 제공하는 기본 형식에 대하여 알아보았다. 이들은 부울, 정수, 실수, 문자 또는 문자열을 표현한다. 기본 형식의 변수들을 모아 구조를 부여하면 클래스를 만들 수 있다. 그것은 제 4 장에서 소개하였다.

C#의 자료 형식은 값 형식과 참조 형식으로 분류된다. 이 장에서 분류 방법을 설명한다. 또, 각 자료 형식이 어떤 범주에 포함되는지 알아본다. 이 장에서는 문자열에 적용되는 메서드에 대하여 알아본다. 제 2 장에서 학습한 배열을 다양한 형식으로 정의한다. 또, 클래스와 유사한 구조체와 열거형을 소개한다.

6.1 값 형식과 참조 형식

C#의 자료 형식은 크게 값 형식과 참조 형식으로 나뉜다. 값 형식은 변수에 값이 직접 저장된다. 참조 형식은 객체를 따로 저장하고 변수에는 그 객체의 주소를 저장한다. 값 형식과 참조 형식이 프로그램에서 어떻게 적용되는지, 차이점과 공통점은 무엇인지 알아보자.

6.1.1 값 형식. 값 형식(value type)은 변수에 값이 직접 저장된다. 정수, 실수, 부울, 열거형, 구조체가 여기에 속한다. 기본 형식 중에는 string을 제외한 나머지가 값 형식에 포함된다.

값 형식 변수에 다른 변수를 대입하면 저장된 값을 복사한다.

```
int x = 3;
int y = x;
y = 5;
```

와 같이 대입한 후 y 값을 바꾸어도 x 값은 변하지 않는다. 변수 x의 값이 y에 복사되기 때문이다.

◆ 연습문제 6.1.1. double 형식이 값 형식임을 확인하시오.

메서드의 매개 변수로 사용될 때도 값을 복사한다.

```
void f(int a)
{
    a++;
}
```

와 같이 정의된 함수 f를

```
f(x);
```

와 같이 호출하여도 x의 값은 변하지 않는다.

6.1.2 참조 형식. 참조 형식(reference type)은 객체의 주소를 변수에 저장한다. 객체는 메모리의 다른 영역에 두고 변수는 그 주소를 값으로 가진다. 따라서 대입 연산은 주소를 복사하며 동일한 객체를 나타낸다. 클래스, 대리자, dynamic이 참조 형식에 속한다.

◇ 변수에 주소를 저장한다는 표현보다 객체를 참조한다는 표현이 올바르다. 그러나 이해를 돕기 위하여 전자를 사용하였다.

대표적인 참조 형식인 클래스를 예로 들어 보자.

```
class Foo
{
    public int n;
    public Foo(int n)
    {
        this.n = n;
    }
```

```
}
```

에 대하여

```
Foo foo1 = new Foo(3);
```

이라 하면 foo1에는 new Foo(3)에 의하여 생성된 객체의 주소가 저장된다.

```
Foo foo2 = foo1;
```

과 같이 대입하면 foo2에는 foo1의 값이 복사된다. foo1에 Foo 객체의 주소가 저장되어 있으므로 foo2에도 같은 객체의 주소가 저장된다. 따라서

```
foo2.n = 5;
```

와 같이 foo2의 필드 값을 바꾸면 foo1.n도 5로 바뀐다.

메서드의 매개 변수로 Foo를 사용해 보자.

```
public static void AddOne(Foo foo)
{
    foo.n++;
}
```

에서 foo는 Foo 객체의 주소이다. 그 주소에 저장된 객체의 필드 n을 1만큼 증가시킨다. 이 메서드를

```
AddOne(foo1);
```

과 같이 호출하면 foo1의 필드 n이 1만큼 증가한다.

예제 6.1.1 Type Demo

```
using System;

struct StructPerson
{
    public string Name { get; set; }

    public StructPerson(string name)
    {
        Name = name;
    }
}

class ClassPerson
{
    public string Name { get; set; }

    public ClassPerson(string name)
    {
        Name = name;
    }
}

class Program
{
```

```
        static void Rename(StructPerson p)
        {
            p.Name = "장길산";
        }

        static void Rename(ClassPerson p)
        {
            p.Name = "장길산";
        }

        static void Main(string[] args)
        {
            StructPerson sp = new StructPerson("임꺽정");
            Rename(sp);
            Console.WriteLine(sp.Name);

            ClassPerson cp = new ClassPerson("임꺽정");
            Rename(cp);
            Console.WriteLine(cp.Name);
        }
    }
```

```
출력
    임꺽정
    장길산
```

코드

```
struct StructPerson
```

은 구조체(struct type)를 정의한다. 구조체는 6.3에서 설명한다. 여기서는 클래스와 다를 것이 없다고 보면 된다. 다만 구조체는 값 형식이고 클래스는 참조 형식이라는 차이가 있다. 위 예제를 보면 StructPerson과 ClassPerson의 코드는 크게 다르지 않다. 그러나 출력된 결과는 다르다. 이유는

```
Rename(sp);
```

을 실행하면 sp의 사본이 전달되고

```
Rename(cp);
```

는 객체의 주소가 전달되기 때문이다. 이것이 값 형식과 참조 형식의 차이점이다.

◆ 연습문제 6.1.2. 예제 6.1.1에

```
StructPerson sp1 = sp;
sp1.Name = "홍길동";
Console.WriteLine(sp.Name);
```

를 추가하여 실행하시오. cp에 대하여 동일한 과정을 수행한 다음 결과를 비교하시오.

참조 형식은 객체의 주소를 값으로 가지므로 값이 없을 수도 있다. 어떤 객체도 참조하지 않는 경우이다. 이 때 값을 null로 설정한다.

```
Foo foo = null;
```

foo는 참조하는 객체가 없으며 foo.n으로 필드를 사용할 수도 없다.

◇ 값 형식은 메모리의 스택(stack)에 저장되며 참조 형식은 힙(heap)에 저장된다.

◇ 스택에 저장되는 값은 실행 구역을 벗어나면 제거된다. 예를 들어 메서드에서 사용되는 값 형식은 메서드가 종료되면 제거된다. 반면 힙에 저장되는 값은 더 이상 사용될 일이 없다고 판단될 때 제거된다.

객체가 값 형식인지 아니면 참조 형식인지 알아 보려면 어떻게 해야 할까? 답은 조금 뒤로 미루고, 먼저 클래스 Type에 대하여 알아보자. 이 클래스는 클래스나 구조체의 정보를 포함하고 있다. 객체 foo가

```
Foo foo = new Foo();
```

와 같이 정의되었다고 하자. 이 때 Foo에 대한 정보를 알려면

```
Type type = typeof(Foo); // 자료 형식으로부터
```

나

```
Type type = foo.getType(); // 객체로부터
```

으로 Type 객체를 생성한다. type에는 Foo에 대한 정보가 포함되어 있다. 이제 Foo가 값 형식인지 알아보는 것은 간단하다.

```
type.isValueType
```

이 true이면 값 형식이고 false이면 참조 형식이다. 다른 정보도 알 수 있다. 예를 들어, Foo가 클래스라면

```
type.isClass
```

가 true이다.

◆ 연습문제 6.1.3. int 형식이 값 형식임을 확인하시오.

◆ 연습문제 6.1.4. int가 클래스인지 확인하시오.

6.2 문자열

문자열은 문자들을 한 줄로 나열한 것으로 클래스 String의 인스턴스이다. 배열과 같이 0부터 시작되는 인덱스를 써서 각 문자를 나타낼 수 있다.

string은 String의 별칭이며 둘 중 어느 것을 써도 상관없다. 클래스 String의 주요 멤버는 다음과 같다.

public sealed class String
- **public int Length { get; }**
  문자열의 길이 곧 문자 수를 반환한다.
- **public char this[int index] { get; }**
  인덱스 index인 문자를 반환한다.
- **public int IndexOf(char value, int startIndex)**
  startIndex 이후 맨 처음 발견되는 문자 value의 인덱스를 반환한다.
- **public string Substring(int startIndex, int length)**
  startIndex부터 length 개를 문자열로 반환한다.
- **public string[] Split(params char[] separator)**
  separator를 경계로 분할하여 분할된 부분 문자열의 배열을 반환한다.
- **public string ToLower()**
  모든 문자를 소문자로 바꾸어 반환한다.
- **public string ToUpper()**
  모든 문자를 대문자로 바꾸어 반환한다.
- **public static string Format(string format, object arg0)**
  format의 복사본을 만들고 arg0를 지정한 위치에 삽입한다.

메서드 Split()은 문자열을 여럿으로 분할하여 배열로 반환한다. 매개 변수 separator는 분할의 경계가 되는 문자를 나타낸다. 키워드 params는 자료 형식이 char인 매개 변수를 여럿 써도 된다는 뜻이다.

문자열

```
string line = "산은 높고 물은 깊어";
```

에 대하여

```
string[] substrings = line.Split(' ');
```

는 line을 분할한다. ' '이 분할의 경계가 된다. 따라서 "산이", "높고", "물이", "깊어"와 같이 넷으로 분할된다.

예제 6.2.1 String Demo

```
using System;

class Program
{
    static void Main()
    {
        string line = "산은 높고 물은 깊어";
        Console.WriteLine(line.IndexOf('은'));
        Console.WriteLine(line.Substring(1, 6));
```

```
            string[] substrings = line.Split(' ');
            foreach (string str in substrings)
                Console.Write(str + ';');
        }
    }
```

```
출력
    1
    은 높고 물
    산은;높고;물은;깊어;
```

◆ 연습문제 6.2.1. Type 객체를 써서 string이 클래스임을 확인하시오.

◆ 연습문제 6.2.2. 예제 6.2.1에

```
string[] ss = "산은 높고, 물은 깊어".Split(' ', ',');
```

를 추가하여 실행하시오. 결과를 설명하시오.

◆ 연습문제 6.2.3. 문자열의 첫 두 문자를 맨 뒤로 옮겨 만든 새 문자열을 반환하는 함수를 작성하시오.

정적 메서드 Format()은 서식에 맞춰 문자열을 구성한다. 콘솔 출력을 위한 메서드 Console.Write()와 같은 형식을 사용한다(9.1.3).

```
int x = 4;
double d = 5.3;
string str = String.Format("x={0}, d={1}", x, d);
```

{0}과 {1}은 각각 x, d로 치환된다. 객체는 메서드 ToString()을 호출하여 반환된 문자열로 치환된다(4.4.4).

```
object obj = new object();
string str = String.Format("obj={1}", obj);
```

에서 {1}은 obj.ToString()으로 치환된다.

◆ 연습문제 6.2.4. 클래스

```
class Rectangle
{
    double width, height;
}
```

에 메서드 ToString()을 오버라이딩하여 작성하시오. String.Format()으로 문자열 구성하여 반환하시오.

6.3 구조체

클래스는 필드와 메서드를 묶어서 만든 참조 형식의 설계도이다. 구성은 클래스와 같지만 값 형식인 설계도를 **구조체**(struct type)라 한다. 클래스와 마찬가지로 필드와 메서드로 구성된다. 클래스는 키워드 class를 사용하여 정의하지만 구조체는 키워드 struct를 써서 정의한다.

원을 나타내는 구조체 Circle은

```
struct Circle
{
    public double radius;

    public double Area()
    {
        return Math.PI * radius * radius;
    }
}
```

와 같이 정의할 수 있다. radius는 필드이며 Area()는 메서드이다. 이 구조체의 인스턴스를 생성하여 사용해 보자.

```
static void Main()
{
    Circle circ;
    circ.radius = 5.6;

    Console.WriteLine(circ.Area());
}
```

구조체는 생성자를 사용하지 않아도 객체를 만들 수 있다. 생성자를 반드시 사용해야 하는 클래스와는 대조적이다.

예제 6.3.1 Struct Demo

```
using System;

struct Circle
{
    public double radius;

    public double Area()
    {
        return Math.PI * radius * radius;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Circle circ;
        circ.radius = 5.6;
        Console.WriteLine(circ.Area());
    }
}
```

```
출력
    98.5203456165759
```

◆ 연습문제 6.3.1. Type 객체를 써서 예제 6.3.1의 Circle이 값 형식임을 확인하시오.

◆ 연습문제 6.3.2. 직사각형을 나타내는 구조체 Rectangle을 만드시오. 각각 폭과 높이를 나타내는 필드 width와 height, 그리고 넓이를 반환하는 메서드 Area()를 가진다.

◇ 구조체는 생성자를 호출하지 않아도 객체를 생성할 수 있다. 왜 그럴까? 그 이유는 메모리 사용 방법에서 찾을 수 있다. 값 형식이든 참조 형식이든 변수를 선언하면 메모리의 스택(stack)에 필요한 만큼 공간을 확보한다. 공간의 크기는 컴파일 단계에서 결정된다.

```
Circle circ;
```

는 값 형식이기 때문에 Circle 객체를 저장할 공간을 확보한다. 그렇다면 클래스의 경우는 어떨까? 클래스 Foo에 대하여

```
Foo foo;
```

는 주소를 저장할 공간만 확보한다. circ는 객체가 들어갈 공간을 확보한 반면 foo는 그렇지 못하다. 따라서 foo가 참조할 객체는 따로 생성해야 한다. 그것이 생성자의 필요 여부를 결정한다.

구조체도 생성자를 가질 수 있다. 구조체와 같은 이름으로 정의된다. 한 가지 주의할 점은 '생성자에서는 모든 필드를 초기화해야 한다'는 것이다. 구조체 Circle의 생성자는 다음과 같이 정의할 수 있다.

```
struct Circle
{
    public int radius;

    public Circle(int radius)
    {
        this.radius = radius;
    }
}
```

이 생성자는

```
Circle circ = new Circle(5.6)
```

과 같이 new를 붙여서 호출한다.

◆ 연습문제 6.3.3. 예제 6.3.1에 Circle의 생성자를 추가하시오. 생성자를 사용하여 Circle 객체를 생성하시오.

◆ 연습문제 6.3.4. 구조체

```
struct Color
{
    public int red, green, blue;
}
```

의 생성자를 작성하시오.

매개 변수가 없는 **기본 생성자**는 내부적으로 항상 만들어지며 프로그래머가 명시적으로 만들 수 없다. 이 생성자는

```
Circle circ = new Circle();
```

와 같이 호출할 수 있다. 기본 생성자에 의하여 생성된 객체의 필드는 모두 기본값으로 초기화된다. 정수는 0, 실수는 0.0, bool은 false, 참조 형식은 null로 초기화된다.

◆ 연습문제 6.3.5. 기본 생성자를 사용하여 Circle 객체를 생성하는 코드를 예제 6.3.1에 추가하시오.

구조체 객체의 선언

```
Circle circ;
```

는 메모리에 공간을 확보할 뿐 필드는 초기화하지 않는다. 따라서 모든 필드를 초기화하는 코드를 작성해야 한다.

◆ 연습문제 6.3.6. 예제 6.3.1에서

```
circ.radius = 5.6;
```

을 제거하시오. 오류를 확인하시오.

구조체는 항상 System.Object에서 파생된다. 그 외에 다른 클래스나 구조체로부터 파생될 수 없다. 또, 구조체나 클래스로 상속될 수 없다. 따라서 구조체로부터 파생된 구조체나 클래스를 만들 수 없다.

구조체의 필드와 메서드에 접근 한정자 private이나 public을 붙여 접근을 제한할 수 있다. 다만 상속될 수 없기 때문에 protected는 사용할 수 없다. 접근 한정자를 적어주지 않으면 private으로 간주한다.

구조체는 값 형식이다. 따라서 메서드의 매개 변수로 사용하여 값을 바꾸더라도 메서드를 호출한 곳에서는 영향을 받지 않는다.

클래스와 마찬가지로 구조체도 정적 필드와 정적 메서드를 가질 수 있다. 또 구조체에서도 인덱서를 정의할 수 있다. 클래스와 같은 형식으로 인덱서를 정의하고 사용한다.

기본 형식 중 string을 제외한 나머지는 구조체로 정의되어 있다. 이들은 모두 정식 이름과 사용하기 쉬운 별칭을 가진다(표 6.1). 예를 들어 32 비트 정수를 표현하는 구조체는 Int32이며 이 이름 대신 int를 별칭으로 사용한다.

구조체	별칭
Boolean	bool
SByte	sbyte
Byte	byte
Char	char
Int16	short
UInt16	ushort
Int32	int
UInt32	uint
Int64	long
UInt64	ulong
Single	float
Double	double
Decimal	decimal

표 6.1 기본 형식과 별칭

본래 이름과 별칭은 어떤 상황에서도 서로 호환된다. 예를 들어

```
Int32 x;
float f = Single.Parse("12.3");
```

은

```
int x;
Single f = float.Parse("12.3");
```

과 일치하는 문장들이다.

표 6.1의 구조체들은 각각 최소값과 최대값을 나타내는 상수 MinValue와 MaxValue를 포함하고 있다. int의 최대값은

```
int m = int.MaxValue;
```

로 얻는다.

struct System.Int32
    public const int MaxValue
        Int32의 최대값을 나타내는 상수이다.
    public const int MinValue
        Int32의 최소값을 나타내는 상수이다.

◆ 연습문제 6.3.7. double의 최대값과 최소값을 출력하시오.

6.4 열거형

정수 상수들에 이름을 부여하고 이들을 모아 자료 형식으로 나타낸 것을 **열거형 형식** (enum type) 또는 **열거형**이라 한다. 열거형에 포함되는 상수를 **열거형 멤버**라 한다. 열거형을 사용하면 상수들을 숫자로 표현하는 것보다 이해하기 쉽다.

네 멤버 East, West, South, North를 포함하는 열거형 Direction은 다음과 같이 정의한다.

```
enum Direction
{
    East,
    West,
    South,
    North
}
```

멤버는 내부적으로 int 형식으로 정의된다. East의 값은 0이고 그 뒤로 순서대로 1씩 증가한다. 멤버 값을 명시적으로 지정할 수도 있다.

```
enum Direction
{
    East = 11
    West = 3,
    South,
    North
}
```

지정되지 않은 멤버 값은 바로 앞 멤버 값에 1을 더한 것이다. 따라서 South와 North의 값은 각각 4, 5이다.

앞에서 살펴본 봐와 같이 지정하지 않은 열거형의 멤버는 int 형식이다. 특별히 멤버의 자료 형식을 지정할 수도 있다.

```
enum Color : short
{
    Red = 4,
    Green = 3,
    Blue = 7
}
```

Color의 각 멤버는 short 형식이다. 가능한 자료 형식은 byte, sbyte, short, ushort, int, uint, long, ulong이다.

열거형 변수에 정수를 대입하려면

```
Color color = (Color)4;
```

와 같이 형식 변환을 한다. 이 때 (Color)4는 Color.Red와 같다. 문장

```
Console.WriteLine(color);
```

는 color에 저장된 멤버의 이름을 출력한다. 멤버의 값을 출력하려면

```
Console.WriteLine("{0:D}", direc);
```

와 같이 **형식 지정 문자** D를 사용한다.

다음 예제에서는 정수를 입력하여 대응되는 멤버를 찾는다.

예제 6.4.1 Enum Demo1

```
using System;

public enum Color : byte
{
    Red = 4,
    Green = 3,
    Blue = 7
}

class Program
{
    static void Main()
    {
        Console.Write(@"4 - 빨강
3 - 녹색
7 - 파랑
색상을 입력하시오: ");
        Color color = (Color)Byte.Parse(Console.ReadLine());
        switch (color)
        {
            case Color.Red:
                Console.WriteLine("빨강");
                break;
            case Color.Green:
                Console.WriteLine("녹색");
                break;
            case Color.Blue:
                Console.WriteLine("파랑");
                break;
            default:
                Console.WriteLine("해당 없음");
                break;
        }
    }
}
```

```
입력
    3
출력
    4 - 빨강
    3 - 녹색
    7 - 파랑
    색상을 입력하시오:
    녹색
```

◇ Console.Write이나 Console.WriteLine에 @을 사용하면 코드에 적은 대로 줄을 바꾼다.

◆ 연습문제 6.4.1. 예제 6.4.1의 열거형 Color가 값 형식임을 확인하시오.

◆ 연습문제 6.4.2. 사계절을 멤버로 하는 열거형을 만드시오.

열거형의 변수 값이 반드시 멤버 중 하나이어야 할 필요는 없다. 예제 6.4.1의 Color에 대하여

```
Color color = (Color)95;
```

도 가능하며, 비트 연산

```
Color color = Color.Red | Color.Green
```

도 대입할 수 있다.

6.5 배열

배열은 자료 형식이 같은 변수들을 이어서 모아 놓은 것을 말한다. 배열은 그것을 구성하는 방법에 따라 일차원 배열, 다차원 배열, 배열의 배열로 분류할 수 있다.

6.5.1 일차원 배열. 일차원 배열(single-dimensional array)은 변수들을 한 줄로 세워 놓았다는 뜻이다. 2.3에서 이미 다루었다. 통상적으로 배열이라고 하면 일차원 배열을 의미한다.

◆ 연습문제 6.5.1. 배열이 참조 형식임을 확인하시오.

◆ 연습문제 6.5.2. 구조체

```
struct Circle
{
    public double radius;
}
```

의 배열을 만드시오.

배열은 추상 클래스 Array의 파생 클래스이다.

abstract class System.Array

public int Length { get; }
배열의 길이를 반환한다.

public void CopyTo(Array array, int index)
모든 요소를 array에 복사한다. array의 인덱스 index부터 시작하여 길이만큼 복사한다.

public static void Copy(Array src, Array dst, int length)
src의 요소 length 개를 dst로 복사한다.

public static void Copy(Array src, int srcIndex, Array dst, int dstIndex, long length)
src의 요소 length 개를 dst로 복사한다. src의 인덱스 srcIndex부터 length 개를 dst의 인덱스 dstIndex부터 시작하여 복사한다.

배열을 복사하려면 메서드 CopyTo()를 사용한다.

```
int[] src = new int[10];
int[] dst = new int[20];
src.CopyTo(dst, 7);
```

는 src에 저장된 요소를 dst의 인덱스 7부터 시작하여 모두 복사한다. 다시 말하면 src[0]을 dst[7]에, src[1]을 dst[8]에 복사한다. 배열의 일부를 복사하려면 정적 메서드 Copy()를 사용한다.

```
Array.Copy(src, 2, dst, 12, 5);
```

는 src의 요소 중 인덱스 2부터 5개를 dst의 인덱스 12부터 시작하여 순서대로 복사한다. 다시 말하면 src[2]를 dst[12]에, src[3]을 dst[13]에 복사한다.

다음 예제는 열거형의 배열을 복사하고 함수의 인자로 사용한 예이다.

예제 6.5.1 Single Array Demo

```
using System;

enum Lamp
{
    R = 4,
    G = 2,
    B = 1
}

class Program
{
    static void TurnOn(Lamp[] lamps, Lamp color)
    {
        for (int i = 0; i < lamps.Length; i++)
            lamps[i] |= color;
    }

    static void Display(Lamp lamp)
    {
        Console.Write("R: {0}, ", (lamp & Lamp.R) != 0 ? "on" : "off");
        Console.Write("G: {0}, ", (lamp & Lamp.G) != 0 ? "on" : "off");
        Console.Write("B: {0}", (lamp & Lamp.B) != 0 ? "on" : "off");
        Console.WriteLine();
    }

    static void Main()
    {
        Lamp[] src = new Lamp[] { Lamp.R | Lamp.B, Lamp.G };
        Lamp[] dst = new Lamp[2];
        src.CopyTo(dst, 0);
        TurnOn(dst, Lamp.R);
        foreach (Lamp lamp in dst)
            Display(lamp);
    }
}
```

출력

```
R: on, G: off, B: on
R: on, G: on, B: off
```

배열은 참조 형식이므로 메서드에서 요소의 값을 바꾸면 호출한 곳에도 값이 바뀐다. 메서드 TurnOn()에서 바꾼 값은 Main()에서도 그대로 적용된다.

◆ 연습문제 6.5.3. 예제 6.5.1의 CopyTo()를 Array의 정적 메서드 Copy()로 바꾸시오.

6.5.2 다차원 배열. 다차원 배열(multidimensional array)은 둘 이상의 차원을 갖는 배열을 말한다. 인덱스의 개수에 따라 이차원 배열, 삼차원 배열과 같이 부른다.

이차원 배열은

```
int[,] table = new int[2, 3];
```

와 같이 선언하고 생성할 수 있다. 첫째 차원의 요소는 2 개이며 둘째 차원의 요소는 3 개이다. 이 배열에는 크기가 2×3인 표를 저장할 수 있다. 이차원 배열을 선언하면

서 초기화하는 방법도 있다.

```
int[,] table = new int[,] { { 4, 2, 1 }, { 3, 1, 1 } };
```

요소 3 개인 배열 2 개로 이루어져 있다.

삼차원 배열은 차원이 3인 배열이다. 유사한 방법으로 선언하고 생성한다.

```
int[,,] cube = new int[3, 2, 4];
```

다음은 예제에서는 이차원 배열을 초기화하고 그 값을 출력한다.

예제 6.5.2 Multiarray Demo1

```
using System;

class Program
{
    static void Main()
    {
    int[,] table = new int[,] { { 4, 2, 1 }, { 3, 1, 1 } };
        for (int i = 0; i < 2; i++)
        {
            for (int j = 0; j < 3; j++)
                Console.Write("table[{0}, {1}] = {2}\t", i, j, table[i, j]);
            Console.WriteLine();
        }
    }
}
```

출력

```
    table[0, 0] = 4 table[0, 1] = 2 table[0, 2] = 1
    table[1, 0] = 3 table[1, 1] = 1 table[1, 2] = 1
```

◆ 연습문제 6.5.4. 예제 6.5.2의 이차원 배열 table에

```
foreach (int x in table)
    Console.WriteLine(x);
```

를 추가하여 실행하시오.

◆ 연습문제 6.5.5. 장기판의 각 칸에 동전을 쌓아 놓은 모양을 생각해 보자. 각 더미에 쌓인 동전의 개수를 나타내는 배열을 구현해 보자.

다차원 배열도 추상 클래스 Array의 파생 클래스이다. 다차원 배열에 적용할 수 있는 멤버에 대하여 알아보자.

abstract class System.Array

public int Length { get; }
모든 차원의 요소 총 수를 반환한다.

public int Rank { get; }
배열의 차원을 반환한다.

public int GetLength(int dimension)
지정한 차원의 요소 수를 반환한다.

public object GetValue(int index1, int index2)
지정한 위치의 요소를 반환한다.

public void SetValue(object value, int index1, int index2)
지정한 위치의 요소 값을 설정한다.

이차원 배열

```
double[,] table = new double[10,15];
```

에 대하여

```
int dim0 = table.GetLength(0)
```

은 table의 첫째 차원의 요소 수, 곧 10이다. 또,

```
double val = table.GetValue(i, j)
```

는 인덱스가 (i,j)인 요소 값이다.

다음은 예제에서는 2 차원 배열의 각 행의 합을 출력한다.

예제 6.5.3 MultiArray Demo2

```
using System;

class Program
{
    static double[] Sum(double[,] arr)
    {
        double[] total = new double[arr.GetLength(0)];
        for (int i = 0; i < arr.GetLength(0); i++)
            for (int j = 0; j < arr.GetLength(1); j++)
                total[i] += arr[i, j];
        return total;

    }
    static void Main()
    {
        double[,] arr = { { 1.0, 1.1, 1.5 }, { -0.8, 2.2, 1.8 } };
        Console.WriteLine(string.Join(", ", Sum(arr)));
    }
}
```

출력

```
3.6, 3.2
```

클래스 string의 정적 메서드 Join()은 배열의 요소들을 이어 붙여 문자열을 만든다.

```
Console.WriteLine(string.Join(", ", Sum(arr)));
```

에서 ", "는 요소들 사이에 들어갈 문자열이다.

◆ 연습문제 6.5.6. 배열의 각 행의 평균과 분산을 각각 반환하는 두 메서드를 예제 6.5.3에 작성하시오. 두 메서드를 호출하여 결과를 출력하시오.

◆ 연습문제 6.5.7. 배열의 Rank를 출력하는 문장을 예제 6.5.3의 Main()에 삽입하시오.

6.5.3 배열의 배열. 배열의 배열(jagged array)은 각 요소가 배열인 배열이다.

```
int[][] foo = new int[3][];
```

는 배열의 배열 foo를 정의한다. foo는 3 개의 배열로 구성된다. foo의 요소는

```
foo[0] = new int[2];
foo[1] = new int[4];
foo[2] = new int[3];
```

와 같이 배열로 정의한다.

```
foo[1][3]
```

은 둘째 요소 foo[1]의 넷째 요소이다. 차원이 큰 배열의 배열도 정의할 수 있다.

```
int[,][,,] big = new int[5,3][,,];
```

는 삼차원 배열을 요소로 갖는 이차원 배열을 정의한다.

예제 6.5.4 Jagged Array Demo

```
using System;

class Program
{
    static void Main()
    {
        int[][] foo = new int[][] { new int[] {1, 2},
                                    new int[] {-3, -2, -1, -0},
                                    new int[] {2, 2, 1} };
        for (int i = 0; i < foo.Length; i++)
        {
            for (int j = 0; j < foo[i].Length; j++)
                Console.Write("foo[{0}][{1}] = {2}\t", i, j, foo[i][j]);
            Console.WriteLine();
        }
    }
}
```

출력

```
foo[0][0] = 1   foo[0][1] = 2
foo[1][0] = -3  foo[1][1] = -2  foo[1][2] = -1  foo[1][3] = 0
foo[2][0] = 2   foo[2][1] = 2   foo[2][2] = 1
```

◆ 연습문제 6.5.8. 예제 6.5.4의 foo를 콘솔 입력으로 초기화하시오.

◆ 연습문제 6.5.9. 예제 6.5.3의 이차원 배열을 배열의 배열로 바꾸어 실행하시오.

이차원 배열은 배열의 배열로 바꿀 수 있다. 그러나 배열의 배열은 주소를 여러 번 참조하므로 더 큰 저장 공간이 필요하고 실행 속도도 느리다.

◇ 길이가 3인 배열 int[]의 배열은 int[][3]과 같이 정의하는 것이 일관성이 있을 것이다. 그러나 배열의 값을 참조할 때는 다시 뒤집어 문법처럼 사용하는 것이 올바르다. 이것은 혼란을 가져올 것이다. 따라서 선언 단계에서 일관성을 버리고 int[3][]과 같은 뒤집힌 문법을 적용한다.

제 7 장 윈도우즈 폼 프로그래밍

GUI(graphical user interface)는 아이콘과 같은 그래픽 요소로 사용자와 소통하는 방법을 말한다. GUI 프로그램은 키보드와 마우스로 입력을 받고 윈도우 화면에 글자, 그림, 아이콘 등을 출력한다.

대비되는 개념으로 TUI(text-based user interface)가 있다. TUI 프로그램은 문자들을 입력받고 문자열을 출력한다.

C#은 GUI 프로그램을 제작하기 위한 도구를 제공한다. 윈도우즈 폼은 최상위 윈도우를 출력한다. 그 안에 컨트롤을 배치하여 화면을 구성할 수 있다. 이벤트 처리 함수는 컨트롤에서 받아들인 사용자 입력을 처리하는 과정을 포함한다.

7.1 윈도우즈 폼 개요

윈도우즈 폼(windows form)은 윈도우즈 프로그램의 기본 단위이다. 단순히 **폼**(form)이라고도 한다. 개발자는 폼에 메뉴와 컨트롤을 배치하여 사용자 인터페이스를 만든다. 마우스나 키보드로 사용자 입력을 받아들이는 이벤트를 작성하고 이를 처리한 결과를 출력한다.

7.1.1 폼 만들기.

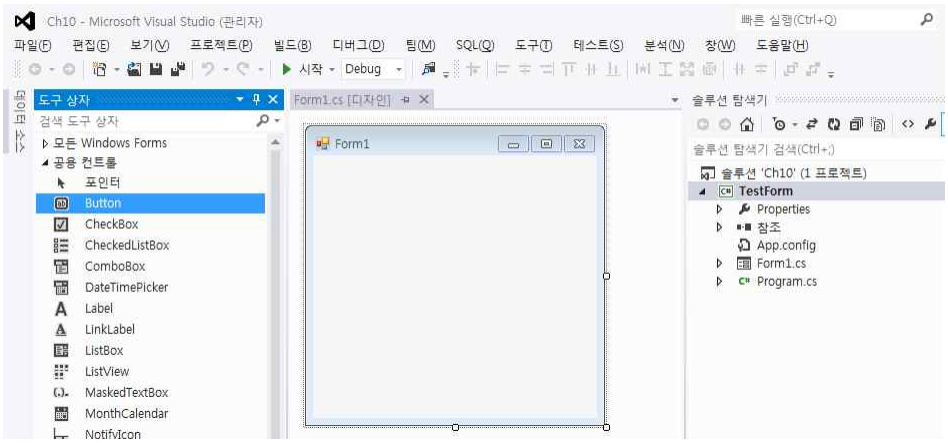
폼을 만드는 방법을 알아보자. 메뉴에서

파일(F) -> 새로 만들기(N) -> 프로젝트(P)

를 선택한다. 새 프로젝트 화면에서 Windows Forms 응용 프로그램을 선택한다.

이름(N): TestForm
위치(L): 작업할 디렉토리
솔루션(S): Ch6

을 입력하고 확인을 누른다. Form1.cs [디자인] 탭이 선택되어 있고 빈 폼이 나타난다. 도구 상자를 열고 Button을 끌어 폼의 중앙에 배치한다.



폼의 크기를 조절하여 보기 좋게 만든다. 버튼(button)은 button1이라는 제목으로

화면 중앙에 배치되어 있다. 이 버튼을 더블 클릭한다. Form1.cs 탭이 선택되고 코드가 나타난다.

프로젝트 TestForm

Form1.cs

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("안녕!!");
}
```

와 같이 한 줄을 추가한다. 프로그램을 실행하여 버튼 button1을 눌러 보자.

이 프로그램의 소스 코드는 세 부분으로 이루어져 있다.

Program.cs
메인 쓰레드를 구성한다.

Form1.cs
클래스 Form1을 구성한다. 더블 클릭하면 디자인이 보인다. 마우스 오른쪽 버튼을 클릭하여 코드 보기(C)를 선택하면 코드를 볼 수 있다.

Form1.Designer.cs
Form1의 디자인을 코드로 볼 수 있다.

Program.cs에서 중요한 문장은

```
Application.Run(new Form1());
```

이다. 클래스 Application의 정적 메서드 Run()은 키보드 입력, 마우스 클릭과 같은 사용자 이벤트를 처리하기 위하여 호출된다. 이 작업은 동일한 과정을 반복하는 것으로 이루어지며 **메시지 루프**(message loop)라 불린다.

sealed class System.Windows.Forms.Application
public static void Run(Form mainForm)
메시지 루프의 실행을 시작하고 지정된 폼을 표시한다.

클래스 Form1은 Form의 파생 클래스이며 앞에서 만든 폼은 Form1 객체이다. 이 클래스에 대한 정보는 Form1.cs와 Form1.Designer.cs에 나뉘어져 기록되어 있다. Form1.Designer.cs를 열어

```
Windows Form 디자이너에서 생성한 코드
```

부분을 확장하자. 다음과 같은 주석

```
/// 이 메서드의 내용을 코드 편집기로 수정하지 마십시오.
```

는 이 부분을 수정하지 말 것을 권고하고 있다.

폼과 관련된 코드를 살펴보자.

```
this.ClientSize = new System.Drawing.Size(165, 117);
```

는 폼의 크기를 결정하는 문장이다. 이 폼의 크기는 가로 165 픽셀(pixel), 세로 117 픽셀이다. 폼의 크기를 바꾸면서 이 부분을 확인해 보자.

```
this.Text = "Form1";
```

은 프로그램을 실행할 때 나타나는 윈도우 곧 폼의 제목이다.

버튼 button1과 관련된 부분을 살펴보자.

```
this.button1.Location = new System.Drawing.Point(42, 46);
```

은 버튼을 그릴 위치를 나타낸다.

```
this.button1.Name = "button1";
```

은 버튼의 이름이다. 화면에 보이는 버튼의 제목이 아니라 코드에서 사용되는 이름이다.

```
this.button1.Size = new System.Drawing.Size(75, 23);
```

은 버튼의 크기를 나타낸다.

```
this.button1.Text = "button1";
```

은 화면에 보이는 버튼의 제목이다.

```
this.button1.Click += new System.EventHandler(this.button1_Click);
```

은 버튼을 클릭했을 때 이벤트를 처리하는 메서드이다. 버튼 button1을 클릭하면 메서드 button1_Click()이 실행된다. 이와 같은 코드는 Windows Form 디자이너가 자동으로 생성한다. 프로그래머가 직접 작성해도 되지만 폼 디자이너를 사용하려면 수정하지 않는 것이 좋다.

7.1.2 좌표계. 윈도우즈 폼의 **좌표계**(coordinate system)는 일반적으로 픽셀(pixel)을 단위로 한다. 한 점은 정수의 쌍 (x, y)로 표현된다. 폼 내부 영역의 왼쪽 위 모서리의 좌표는 (0, 0)이다. x는 이 점에서 오른쪽으로 이동한 단위만큼 증가하고 y는 아래로 이동한 단위만큼 증가한다.

```
this.button1.Location = new System.Drawing.Point(21, 23);
```

은 버튼 button1의 왼쪽 위 모서리의 좌표가 (21, 23)임을 나타낸다. 다시 말하면 폼 내부 영역의 왼쪽 위 모서리에서 오른쪽으로 21, 아래로 23 만큼 이동한 점에 버튼의 왼쪽 위 모서리를 위치시킨다.

◆ 연습문제 7.1.1. 위 button1의 왼쪽 위 모서리의 좌표를

```
this.button1.Location = new System.Drawing.Point(0, 0);
```

으로 수정하고 실행해 보자.

◆ 연습문제 7.1.2. button1의 크기를

```
this.button1.Size = new System.Drawing.Size(100, 50);
```

으로 수정하고 실행해 보자. 또 디자이너에서 버튼의 크기를 바꾸어 실행해 보자.

구조체 Point는 점을 표현한다.

struct System.Drawing.Point
public Point(int x, int y)
좌표가 (x, y)인 Point 객체를 생성한다.
public int X { get; set; }
x 좌표를 설정하거나 가져온다.
public int Y { get; set; }
y 좌표를 설정하거나 가져온다.

두 프로퍼티 X, Y가 좌표를 나타낸다.

```
Point p = new Point(21, 23);
```

이라 하면 p.X는 21이고 p.Y는 23이다.

구조체 Size는 직사각형의 크기를 표현한다.

struct System.Drawing.Size
public Size(int width, int height)
폭이 width이고 높이가 height인 Size 객체를 생성한다.
public int Width { get; set; }
폭을 설정하거나 가져온다.
public int Height { get; set; }
높이를 설정하거나 가져온다.

두 프로퍼티 Width, Height가 각각 폭과 높이를 나타낸다.

```
Size s = new Size(75, 23);
```

이라 하면 s.Width는 75이고 s.Height는 23이다.

7.2 컨트롤

컨트롤(control)은 윈도우에 배치되어 있는 단위 요소를 말한다. 사용자의 입력을 받아들이는 것도 있고 작업을 수행한 결과를 보여주는 창구 역할을 하는 것도 있다. 버튼(button), 텍스트 박스(text box), 콤보 박스(combo box) 등과 같이 윈도우 화면에 보이는 많은 요소가 컨트롤이다. 윈도우즈 폼도 컨트롤이다.

각 컨트롤은 클래스에 대응되며 이 클래스는 클래스 Control로부터 파생된다. 또 Control은 클래스 Component로부터 파생된다.

class System.Windows.Forms.Component
class System.Windows.Forms.Control

폼에 컨트롤을 배치하여 사용하는 방법을 알아보자.

7.2.1 레이블. 클래스 Label에 대응되는 **레이블**(label)은 주로 변하지 않는 문자열을 표시하는 역할을 한다.

```
class Sysetm.Windows.Forms.Label
```

7.2.2 버튼. 클래스 Button에 대응되는 **버튼**(button)은 그것을 눌러 정보를 전달한다. 매우 자주 쓰이는 컨트롤이다.

```
class System.Windows.Forms.Button
    public event EventHandler Click
        버튼을 누르면 발생한다. 이 이벤트는 클래스 Control에 정의되어 있다.
```

7.2.3 텍스트 박스. 문자열을 입력할 때 주로 사용하는 컨트롤이 **텍스트 박스**(text box)이다. 이 컨트롤은 클래스 TextBox에 대응된다.

```
class System.Windows.Forms.TextBox
    public override string Text { get; set; }
        텍스트 박스의 문자열을 설정하거나 가져온다.
    public event EventHandler TextChanged
        Text의 값이 변경되면 발생한다. 이 이벤트는 클래스 Control에 정의되어
        있다.
    public event EventHandler Click
        텍스트 박스를 클릭하면 발생한다. 이 이벤트는 클래스 Control에 정의되
        어 있다.
```

텍스트 박스에 출력될 문자열을 지정하려면 프로퍼티 Text를 사용한다. TextBox 객체 tb에 대하여

```
tb.Text = "아름다운 강산";
```

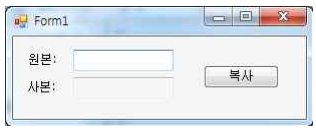
은 tb에 문자열 아름다운 강산을 출력한다.

```
tb.Text = "";
```

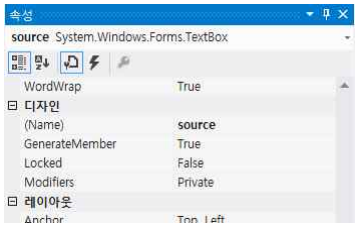
은 tb에 출력된 문자열을 지운다.

새 프로젝트 TestControl1을 추가하고 다음과 같이 폼에 컨트롤을 배치하자. 도구 상자에서 컨트롤을 끌어다 놓고 크기를 조절한다. 적당한 위치로 옮기고 정렬한다. 레이아웃 툴바의 아이콘을 사용하면 편리하게 정렬할 수 있다.

컨트롤	속성	이벤트
Label	Text: 원본:	
Label	Text: 사본:	
TextBox	(Name): source	
TextBox	(Name): target ReadOnly: True	
Button	(Name): copy Text: 복사	Click



이제 속성 창에서 속성을 수정하자. 속성 창은 다음 그림과 같다. 필요한 항목을 찾아서 수정하면 된다.



이 프로젝트에서는 텍스트 박스 source에 문자열을 쓰고 버튼 copy를 눌러서 target에 복사하려 한다. 파일 Form1.Designer.cs를 열어서 다음 다섯 문장을 확인하자.

```
private System.Windows.Forms.Label label1;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.TextBox source;
private System.Windows.Forms.TextBox target;
private System.Windows.Forms.Button Copy;
```

앞에서 만든 다섯 컨트롤이 정의되어 있다.

버튼 copy의 Click 이벤트를 등록하자. Form1.cs [디자인] 창에서 버튼을 더블 클릭하면 파일 Form1.cs가 열리고 커서가

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

에 위치한다. 여기에 다음과 같이 입력한다.

프로젝트 TestControl1

Form1.cs

```
private void copy_Click(object sender, EventArgs e)
{
    target.Text = source.Text;
}
```

프로그램을 실행하고 결과를 확인하자. 버튼을 클릭하면 source의 문자열이 target으로 복사된다.

파일 Form1.Designer.cs의 문장

```
this.copy.Click += new System.EventHandler(this.copy_Click);
```

은 메서드 copy_Click을 버튼 copy의 클릭 이벤트 처리 함수로 등록한다. 연산자 += 가 사용된다.

이벤트 처리 함수는 Form1.cs [디자인] 창에서 컨트롤의 속성을 변경하여 등록한다. 속성 창 위 부분에 있는 이벤트 아이콘(⚡)을 클릭한다. 출력된 창에서 처리할 이벤트를 더블 클릭하면 자동으로 등록된다. 각 컨트롤의 기본 이벤트는 해당 컨트롤을 더블 클릭하여 등록할 수도 있다. 예를 들어 버튼의 Click 이벤트는 버튼을 더블 클릭하여 등록할 수 있다.

텍스트 박스 source를 클릭하면 거기에 쓰인 문자열을 지우도록 이벤트 처리 함수를 작성하고 등록해 보자. source를 선택하고 속성 창을 연다. 속성 창 위 부분에 있는 이벤트 아이콘을 클릭한다. Click 항목을 더블 클릭한다. 열려진 코드 창에 다음과 같이 입력한다.

Form1.cs

```
private void source_Click(object sender, EventArgs e)
{
    source.Text = "";
}
```

프로그램을 실행하고 결과를 확인하자.

◇ 이벤트 처리 함수의 매개 변수 sender는 이벤트가 발생한 객체를 나타낸다. 위 코드에서는 source이다. 따라서

```
source.Text = "";
```

는

```
((TextBox)sender).Text = "";
```

와 같이 써도 된다.

◆ 연습문제 7.2.1. 사용자가 텍스트 박스에 문자를 쓰면 TextChanged 이벤트가 발생한다. source에 문자를 쓰면 target에 복사되도록 이벤트 처리 함수를 작성하시오.

◇ 폼 프로그램에서도 콘솔에 문자열을 출력할 수 있다. 프로젝트 TestControl1을

```
private void copy_Click(object sender, EventArgs e)
{
    target.Text = source.Text;
    Console.WriteLine(source.Text);
}
```

와 같이 수정하자. 메뉴에서

디버그(D) -> 디버깅 시작(S)

을 눌러 프로그램을 실행하자. 메뉴에서

보기(V) -> 출력(O)

을 선택하면 화면 아래쪽에 **출력** 창이 나타난다. 텍스트 박스 source에 문자열을 입력하고 버튼 copy를 누르면 출력 창에 입력한 문자열이 출력된다.

System.Diagnostics.Debug.WriteLine()을 써도 같은 결과를 얻을 수 있다.

◆ 연습문제 7.2.2. System.Diagnostics.Debug.WriteLine()을 써서 **출력** 창에 문자열을 출력하시오.

◆ 연습문제 7.2.3. 사칙연산 계산기를 만드시오.

7.2.4 컨테이너. 컨트롤들을 한 그룹으로 묶어 담는 그릇과 같은 컨트롤을 **컨테이너**(container)라 한다. 그룹 박스와 패널이 대표적인 컨테이너이다.

클래스 GroupBox로 나타내는 **그룹 박스**(group box)는 제목이 있는 컨테이너이다. 그룹의 제목을 표시할 때 사용한다.

```
class Sysetm.Windows.Forms.GroupBox
```

패널(panel)은 제목이 없는 컨테이너이다. 제목이 없이 단순한 묶음만 표시한다. 클래스 Panel에 대응된다.

```
class Sysetm.Windows.Forms.Panel
```

7.2.5 라디오 버튼. 라디오 버튼(radio button)은 여럿 중 하나를 선택할 때 사용된다. 이들을 그룹 박스나 패널 안에 배치시켜 한 묶음으로 만든다. 하나를 선택하면 묶음 안의 나머지 라디오 버튼은 자동으로 선택이 해제된다. 라디오 버튼은 클래스 RadioButton에 대응된다.

```
class Sysetm.Windows.Forms.RadioButton
    public bool Checked { get; set; }
        라디오 버튼의 선택 여부를 가져오거나 설정한다.
    public event EventHandler CheckedChanged;
        Checked의 값이 변경되었을 때 발생한다.
```

라디오 버튼이 선택되었는지 아닌지는 프로퍼티 Checked의 값으로 알 수 있다. 선택되었으면 Checked의 값이 true이고 선택이 해제되었으면 false이다. 라디오 버튼을 클릭하면 CheckedChanged 이벤트가 발생한다. 따라서

```
private void radioButton_CheckedChanged(object sender, EventArgs e)
{
    if (radioButton.Checked)
        라디오 버튼 radioButton이 선택된 상태일 때 할 일
    else
        라디오 버튼 radioButton이 선택되지 않은 상태일 때 할 일
```

```
    }
```

과 같이 활용할 수 있다.

7.2.6 체크 박스. 체크할 수 있는 상자에 이름을 붙여놓은 것을 **체크 박스**(check box)라 한다. 라디오 버튼과 달리 단독으로 사용되어 선택된 상태나 선택되지 않은 상태를 나타낸다. 클래스 CheckBox가 체크 박스에 대응된다.

```
class Sysetm.Windows.Forms.CheckBox
    public bool Checked { get; set; }
        체크 박스의 선택 여부를 가져오거나 설정한다.
    public event EventHandler CheckedChanged;
        Checked의 값이 변경되었을 때 발생한다.
    public CheckState CheckState { get; set; }
        체크 박스의 선택 여부를 가져오거나 설정한다.
    public event EventHandler CheckStateChanged;
        CheckState의 값이 변경되었을 때 발생한다.
```

체크 박스는 두 상태 또는 세 상태를 표현할 수 있다. 속성에서 ThreeState를 False로 설정하면 두 상태를 표현하고 True로 설정하면 세 상태를 표현한다.

두 상태 체크 박스의 상태는 프로퍼티 Checked의 값으로 알 수 있다. 체크된 상태이면 Checked는 true이고 그렇지 않으면 false이다. 두 상태 체크 박스에서 발생하는 이벤트는 CheckedChanged이다. 따라서

```
private void checkBox_CheckedChanged(object sender, EventArgs e)
{
    if (checkBox.Checked)
        체크 박스 checkBox가 체크된 상태일 때 할 일
    else
        체크 박스 checkBox가 체크되지 않은 상태일 때 할 일
}
```

와 같은 방법으로 활용할 수 있다.

세 상태 체크 박스의 상태는 프로퍼티 CheckState의 값으로 알 수 있다. 프로퍼티 CheckState는 열거형 형식 CheckState의 요소를 값으로 가진다. 프로퍼티와 열거형 형식의 이름이 같다.

```
enum Sysetm.Windows.Forms.CheckState
    Unchecked
        선택되지 않은 상태를 나타낸다.
    Checked
        선택된 상태를 나타낸다.
    Indeterminate
        결정되지 않은 상태를 나타낸다.
```

세 상태 체크 박스에서 발생하는 이벤트는 CheckStateChanged이다. 따라서

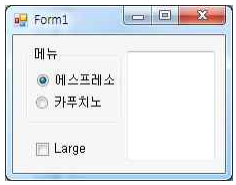
```
private void checkBox_CheckStateChanged(object sender, EventArgs e)
{
    if (checkBox.CheckState == CheckState.Checked)
```

```
            체크 박스 checkBox가 체크된 상태일 때 할 일
        else if (checkBox.CheckState == CheckState.Indeterminate)
            체크 박스 checkBox의 상태가 결정되지 않았을 때 할 일
        else
            체크 박스 checkBox가 체크되지 않은 상태일 때 할 일
    }
```

와 같은 방법으로 활용할 수 있다.

새 프로젝트 TestControl2를 추가하자. 폼에 다음과 같이 컨트롤을 배치하고 속성을 수정하자.

컨트롤	속성	이벤트
GroupBox	Text: 메뉴	
RadioButton	(Name): espresso Checked: True Text: 에스프레소	CheckedChanged
RadioButton	(Name): cappuccino Text: 카푸치노	CheckedChanged
CheckBox	(Name): large Text: Large	CheckedChanged
TextBox	(Name): info Multiline: True	



이 프로젝트에서는 라디오 버튼과 체크 박스의 상태를 텍스트 박스 info에 출력한다. 라디오 버튼 espresso, cappuccino와 체크 박스 large를 더블 클릭하여 이벤트 처리 함수를 등록하고 Form1.cs를 다음과 같이 수정한다.

프로젝트 TestControl2

Form1.cs

```
private void DisplayInfo()
{
    string str;
    if (espresso.Checked)
        str = "에스프레소\r\n";
    else
        str = "카푸치노\r\n";
    str += large.Checked ? "  Large" : "  Small";
    info.Text = str;
}

private void espresso_CheckedChanged(object sender, EventArgs e)
{
    DisplayInfo();
}

private void cappuccino_CheckedChanged(object sender, EventArgs e)
{
    DisplayInfo();
}

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

```
        DisplayInfo();
    }
```

◆ 연습문제 7.2.4. 음료를 나타내는 라디오 버튼을 추가하시오.

◆ 연습문제 7.2.5. 각 음료에 가격을 매기고 텍스트 박스 info에 가격을 출력하는 코드를 추가하시오.

◆ 연습문제 7.2.6. 체크 박스 large가 세 상태를 표현하도록 수정하시오.

7.2.7 글꼴. 윈도우즈가 제공하는 글꼴은 클래스 Font로 얻을 수 있다.

```
sealed class System.Drawing.Font
    public Font(Font prototype, FontStyle newStyle)
        글꼴 prototype을 새 스타일 newStyle로 변형하는 글꼴을 생성한다.
    public Font(string familyName, float emSize)
        크기가 emSize인 글꼴을 생성한다. 크기의 단위는 em-size(포인트)이다.
    public Font(string familyName, float emSize, FontStyle style)
        크기 emSize, 스타일 style인 글꼴을 생성한다. 크기의 단위는 em-size
        (포인트)이다.
```

매개 변수 familyName은 굴림, 굴림체, 돋움, 돋움체와 같은 글꼴의 이름이다. emSize는 글꼴의 크기를 포인트(pt) 단위로 나타낸 것이다. 열거형 형식 FontStyle은 글꼴의 굵기, 기울임, 밑줄 등을 결정한다.

```
enum System.Drawing.FontStyle
    Regular
    Bold
    Italic
    Underline
    Strikeout
```

새로운 글꼴은 다음과 같이 생성할 수 있다.

```
Font font = new Font("궁서체", 12.0f, FontStyle.Underline);
```

생성된 글꼴 font는 궁서체 12포인트이고 밑줄이 있다. 이 폰트를 TextBox tb의 글꼴로 설정하려면

```
tb.Font = font;
```

와 같이 적는다.

7.2.8 색상. 컨트롤의 바탕색이나 글자색을 설정하려면 구조체 Color를 사용한다. 색상은 네 요소 RGBA로 구성되며 각 요소의 크기는 한 바이트씩이다. 주요 색상은 정적 프로퍼티로 얻을 수 있다.

```
struct System.Drawing.Color
    public byte R { get; }
```

```
        색상의 빨강 구성 요소 값을 가져온다.
    public byte G { get; }
        색상의 녹색 구성 요소 값을 가져온다.
    public byte B { get; }
        색상의 파랑 구성 요소 값을 가져온다.
    public byte A { get; }
        색상의 알파 구성 요소 값을 가져온다.
    public static Color Black { get; }
        검정색 Color 객체를 가져온다.
    public static Color White { get; }
        흰색을 Color 객체를 가져온다.
    public static Color Red { get; }
        빨간색 Color 객체를 가져온다.
    public static Color Blue { get; }
        파란색 Color 객체를 가져온다.
    public static Color FromArgb(int red, int green, int blue)
        RGB 색상을 갖는 Color 객체를 가져온다.
```

클래스 Control의 프로퍼티 ForeColor는 문자열의 색을 가져오거나 설정할 때 사용된다. 또, 프로퍼티 BackColor는 배경색을 가져오거나 설정할 때 사용된다. 클래스 TextBox에서는 이 두 프로퍼티를 오버라이딩한다.

```
class System.Windows.Forms.TextBox
    public override Color ForeColor { get; set; }
        문자열의 색상을 설정하거나 가져온다.
    public override Color BackColor{ get; set; }
        배경색을 설정하거나 가져온다.
```

TextBox 객체 tb에 대하여

```
tb.BackColor = Color.Blue;
```

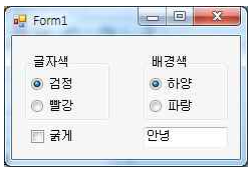
는 tb의 배경색을 파랑으로 설정하며

```
tb.ForeColor = Color.Red;
```

는 tb의 글자색을 빨강으로 설정한다.

새 프로젝트 TestControl3을 추가하자. 폼에 다음과 같이 컨트롤을 배치하고 속성을 수정하자.

컨트롤	속성	이벤트
GroupBox	Text: 글자색	
RadioButton	(Name): blackText Checked: True Text: 검정	CheckedChanged
RadioButton	(Name): redText Text: 빨강	CheckedChanged
GroupBox	Text: 배경색	
RadioButton	(Name): whiteBack Checked: True Text: 하양	CheckedChanged
RadioButton	(Name): blueBack Text: 파랑	
CheckBox	(Name): bold Text: 굵게	CheckedChanged
TextBox	(Name): display Text: 안녕	



이 프로젝트에서는 라디오 버튼과 체크 박스의 상태에 따라 텍스트 박스 display의 배경색과 거기에 출력되는 문자열의 색상 및 폰트를 다르게 설정한다. 네 라디오 버튼 blackText, redText, whiteBack, blueBack과 체크 박스 bold를 더블 클릭하여 이벤트 처리 함수를 등록하고 Form1.cs를 다음과 같이 수정한다.

프로젝트 TestControl3

Form1.cs

```
private void SetTextColor()
{
    if (blackText.Checked)
        display.ForeColor = Color.Black;
    else
        display.ForeColor = Color.Red;
}

private void blackText_CheckedChanged(object sender, EventArgs e)
{
    SetTextColor();
}

private void redText_CheckedChanged(object sender, EventArgs e)
{
    SetTextColor();
}

private void bold_CheckedChanged(object sender, EventArgs e)
{
    if (bold.Checked)
        display.Font = new Font(display.Font, FontStyle.Bold);
    else
        display.Font = new Font(display.Font, FontStyle.Regular);
}

private void SetBackColor()
```

```
{
    if (whiteBack.Checked)
        display.BackColor = Color.White;
    else
        display.BackColor = Color.Blue;
}

private void whiteBack_CheckedChanged(object sender, EventArgs e)
{
    SetBackColor();
}

private void blueBack_CheckedChanged(object sender, EventArgs e)
{
    SetBackColor();
}
```

◆ 연습문제 7.2.7. 라디오 버튼을 추가하여 색상 수를 늘리시오.

◆ 연습문제 7.2.8. 이탤릭 글꼴을 사용할 수 있는 체크 박스를 추가하시오.

7.2.9 리스트 박스. 클래스 ListBox에 대응되는 **리스트 박스**(list box)는 요소를 나열하여 사용자가 선택할 수 있도록 한다.

class System.Windows.Forms.ListBox

public ListBox.ObjectCollection Items { get; }
리스트 박스의 요소들을 포함하는 집합체를 가져온다. 이 객체에 요소를 추가하면 리스트 박스에 추가된다. 또 이 객체에서 요소를 제거하면 리스트 박스에서 제거된다.

public override int SelectedIndex { get; set; }
선택된 항목의 인덱스를 가져오거나 항목을 새로 선택한다.

public object SelectedItem { get; set; }
선택된 항목을 가져오거나 항목을 새로 선택한다.

public ListBox.SelectedObjectCollection SelectedItems { get; }
선택된 항목을 가져온다. 다중 선택이 가능한 경우 선택된 모든 항목을 가져온다. 선택된 항목을 제거하거나 추가로 선택할 수 있다.

public event EventHandler SelectedIndexChanged
SelectedIndex가 변경되었을 때 발생한다.

클래스 ObjectCollection은 ListBox의 내부 클래스로 리스트 박스의 항목 전체로 구성된다. 리스트 박스에 항목을 추가하거나 제거하려면 이 클래스의 인스턴스를 사용해야 한다.

class System.Windows.Forms.ListBox.ObjectCollection

public int Count { get; }
요소의 개수를 가져온다.

public int Add(object item)
목록에 요소를 추가한다. 추가된 항목의 인덱스를 반환한다.

public void Remove(Object value)
목록에서 요소를 제거한다.

public void RemoveAt(int index)

인덱스가 index인 항목을 제거한다.

리스트 박스 listBox에 대하여

```
ListBox.ObjectCollection items = listBox.Items;
items.Add("Mango");
```

는 Mango를 listBox의 항목으로 추가하며

```
items.Remove("Mango");
```

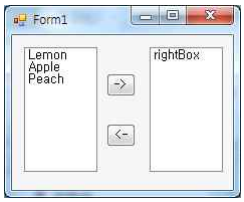
는 listBox의 항목에서 Mango를 제거한다.

클래스 SelectedObjectCollection은 ListBox의 내부 클래스로 선택된 항목 전체로 구성된다.

class System.Windows.Forms.ListBox.SelectedObjectCollection
public int Count { get; }
요소의 개수를 가져온다.
public void CopyTo(Array destination, int index)
요소를 배열의 인덱스 index부터 시작하여 복사한다.

새 프로젝트 TestControl4를 추가하자. 폼에 다음과 같이 컨트롤을 배치하고 속성을 수정하자.

컨트롤	속성	이벤트
ListBox	(Name): leftBox Items: Lemon Apple Peach SelectionMode: MultiSimple	
ListBox	(Name): rightBox SelectionMode: MultiSimple	
Button	(Name): toRight Text: ->	Click
Button	(Name): toLeft Text: <-	Click



리스트 박스 leftBox의 속성을 열고 Items 항목에서 (컬렉션)을 클릭한다. 문자열 컬렉션 편집기에

Lemon
Apple
Peach

를 입력한다. 확인을 눌러 입력을 종료한다. 두 버튼 toRight와 toLeft에 Click 이벤트 처리 함수를 등록한다. 파일 Form1.cs를 열어서 다음과 같이 수정한다.

프로젝트 TestControl4

Form1.cs

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

```
{
    ListBox.ObjectCollection itemsLeft = leftBox.Items;
    ListBox.ObjectCollection itemsRight = rightBox.Items;
    object[] sel = new object[leftBox.SelectedItems.Count];
    // 선택된 항목을 배열에 복사한다.
    leftBox.SelectedItems.CopyTo(sel, 0);
    foreach (object obj in sel)
    {
        // rightBox에 추가한다.
        itemsRight.Add(obj);
        // leftBox에서 삭제한다.
        itemsLeft.Remove(obj);
    }
}
```

◆ 연습문제 7.2.9. 리스트 박스 rightBox의 속성 Items에 항목을 추가하시오.

◆ 연습문제 7.2.10. 메서드 toLeft_Click()을 채워 완성하시오.

◆ 연습문제 7.2.11. 리스트 박스의 속성 SelectionMode의 값은 MultiExtended나 One으로도 설정할 수 있다. 이들의 차이점을 찾으시오.

◆ 연습문제 7.2.12. 텍스트 박스와 버튼을 추가하고 버튼을 눌러 텍스트 박스에 쓰인 문자열을 리스트 박스에 추가하는 코드를 작성하시오.

7.2.10 이미지. 주로 이차원 그림을 말하는 **이미지**(image)는 파일을 읽어 생성할 수도 있고 픽셀에 색상 정보를 대입하여 얻을 수도 있다. 클래스는 Bitmap은 이미지의 정보 및 픽셀 자료를 포함하고 있다.

```
sealed class System.Drawing.Bitmap
    public Bitmap(string filename)
        비트맵 파일을 불러온다.
    public Bitmap(int width, int height)
        가로 width 픽셀, 세로 height 픽셀인 비트맵을 생성한다.
    public Color GetPixel(int x, int y)
        픽셀의 색을 가져온다.
    public void Save(string filename)
        비트맵을 파일에 저장한다.
    public void SetPixel(int x, int y, Color color)
        픽셀의 색을 설정한다.
```

파일로부터 이미지를 불러오려면

```
Bitmap bitmap = new Bitmap("C:\\MyFolder\\image.jpg");
```

와 같이 생성자에 매개 변수로 파일 경로를 적어주면 된다.

Bitmap의 기본 클래스는 추상 클래스 Image이다. 이미지의 정보를 포함하고 있다.

```
abstract class System.Drawing.Image
    public int Width { get; }
        이미지의 폭을 가져온다.
```

public int Height { set;}
이미지의 높이를 가져온다.
public Size Size { size; }
이미지의 크기를 가져온다.
public void Save(string filename)
이미지를 파일에 저장한다.

7.2.11 콤보 박스. 항목을 감추어 두고 선택할 때 열어볼 수 있는 컨트롤이 **콤보 박스**(combo box)이다. 클래스 ComboBox로 구현한다.

class System.Windows.Forms.ComboBox
public ComboBox.ObjectCollection Items { get; }
리스트 박스의 요소들을 포함하는 집합체를 가져온다. 이 객체에 요소를 추가하면 리스트 박스에 추가된다. 또 이 객체에서 요소를 제거하면 리스트 박스에서 제거된다.
public override int SelectedIndex { get; set; }
선택된 항목의 인덱스를 가져오거나 항목을 새로 선택한다.
public object SelectedItem { get; set; }
선택된 항목을 가져오거나 항목을 새로 선택한다.
public event EventHandler SelectedIndexChanged
SelectedIndex가 변경되었을 때 발생한다.

콤보 박스의 항목을 모두 포함하는 객체는 프로퍼티 Items로 얻을 수 있다. 이 객체는 ComboBox의 내부 클래스인 ObjectCollection의 인스턴스이다.

class System.Windows.Forms.ComboBox.ObjectCollection
public int Count { get; }
요소의 개수를 가져온다.
public int Add(object item);
목록에 요소를 추가한다. 추가된 항목의 인덱스를 반환한다.

콤보 박스 comboBox에 대하여

```
comboBox.ObjectCollection items = comboBox.Items;
items.Add("figure");
```

는 figure를 comboBox에 항목으로 추가하며

```
items.Remove("figure");
```

는 comboBox의 항목에서 figure를 제거한다.

7.2.12 픽쳐 박스. 그림 파일을 읽어서 출력하려면 **픽쳐 박스**(picture box)를 사용한다. 클래스 PictureBox에 대응된다.

class System.Windows.Forms.PictureBox
public Image Image { get; set; }
픽쳐 박스에 표시되는 이미지를 가져오거나 설정한다.
public PictureBoxSizeMode SizeMode { get; set; }
픽쳐 박스에 그림을 그리는 방법을 가져오거나 설정한다.

픽쳐 박스에 그림을 출력하려면

```
pictureBox.Image = new Bitmap("C:\\MyFolder\\image.jpg");
```

와 같이 프로퍼티 Image를 설정하면 된다.

프로퍼티 SizeMode는 픽쳐 박스에 그릴 그림의 위치, 확대·축소 등을 결정한다. 열거형 형식 PictureBoxSizeMode의 요소를 값으로 가진다. 기본 값은 이미지를 왼쪽 위 모퉁이에 배치하는 Normal이다.

public enum System.Windows.Forms.PictureBoxSizeMode

Normal
이미지를 왼쪽 위 모퉁이에 배치한다. 기본 설정이다.

StretchImage
이미지를 픽쳐 박스의 크기에 맞게 확대하거나 축소한다.

AutoSize
픽쳐 박스의 크기를 이미지에 맞게 조정한다.다.

CenterImage
이미지와 픽쳐 박스의 중심을 일치시킨다.

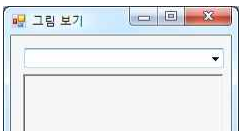
Zoom
가로 세로 비율이 유지되도록 이미지를 확대하거나 축소하여 픽쳐 박스에 포함시킨다.

그림 파일을 읽어서 픽쳐 박스에 출력해 보자. 새 프로젝트 TestControl5를 추가하자. 폼의 속성에서 윈도우 제목을 수정하자.

Text: 그림 보기

폼에 다음과 같이 컨트롤을 배치하고 속성을 수정하자.

컨트롤	속성	이벤트
Form	Text: 그림 보기	Load
ComboBox	(Name): fileList	SelectedIndexChanged
PictureBox	(Name): picture BorderStyle: Fixed3D	



디렉토리를 지정하여 거기에 포함된 모든 그림 파일들의 이름으로 콤보 박스를 구성하려 한다. 폼을 더블 클릭하여 Load 이벤트 처리 함수를 등록한다. 콤보 박스 fileList를 더블 클릭하여 SelectedIndexChanged 이벤트 처리 함수를 등록한다. 파일 Form1.cs를 열어서 다음과 같이 수정한다.

프로젝트 TestControl5

Form1.cs

```
public partial class Form1 : Form
{
    // 그림 파일이 들어있는 디렉토리를 설정한다.
    private string path = @"C:\...\Pictures";
    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        string[] files = Directory.GetFiles(path);
        ComboBox.ObjectCollection items = fileList.Items;
        foreach (string name in files)
        {
            FileInfo info = new FileInfo(name);
            if (info.Extension == ".bmp" || info.Extension == ".jpg")
                items.Add(info.Name);
        }

         picture.SizeMode = PictureBoxSizeMode.StretchImage;
    }

    private void fileList_SelectedIndexChanged(object sender, EventArgs e)
    {
        picture.Image = new Bitmap(path + '\\' + fileList.SelectedItem);
    }
}
```

프로그램을 실행하면 콤보 박스 fileList에 확장자가 bmp, jpg인 파일들이 나열된다. 파일 이름을 클릭하여 그림이 그려지는 것을 확인하자.

◆ 연습문제 7.2.13. PictureBoxSizeMode.CenterImage를 사용하도록 코드를 수정하시오.

◆ 연습문제 7.2.14. 비트맵을 좌우로 대칭시켜 그리시오. 기존 비트맵과 크기가 같은 새로운 비트맵을 생성하고 클래스 Bitmap의 메서드 SetPixel()로 각 픽셀의 색상을 설정한다.

윈도우즈 폼에서도 여러 가지 이벤트가 발생한다. 폼의 속성 창에서 확인할 수 있다. 화면에 처음 표시될 때 이벤트 Load가 발생한다.

```
class System.Windows.Forms.Form
    public event EventHandler Load
        폼이 처음으로 표시되기 전에 발생한다.
```

디렉토리 내용을 불러오려면 클래스 Directory를 사용한다.

```
static class System.IO.Directory
    public static string[] GetFiles(string path)
        디렉토리에 있는 파일의 이름을 반환한다.
```

클래스 FileInfo는 문자열로부터 디렉토리 이름, 파일 이름, 확장자 등의 정보를 분

리하여 제공한다. 실제 경로가 아니어도 상관없으며 단지 경로를 나타내는 문자열을 대입하면 된다.

```
sealed class System.IO.FileInfo
    public string DirectoryName { get; }
        디렉토리 이름을 가져온다.
    public string Name { get; }
        파일 이름을 가져온다.
    public string Extension { get; }
        확장자를 가져온다.
```

7.3 대화 상자

대화 상자(dialog)는 프로그램 실행 정보를 출력하거나 사용자 입력을 받기 위해 별도로 보여주는 창이다. 경고 메시지, 파일 선택, 색상 선택, 글꼴 선택과 같이 내부적으로 제공되는 것도 있고 개발자가 직접 만들어 제공하는 것도 있다.

내부적으로 제공되는 대화 상자의 기본 클래스는 CommonDialog이다.

```
abstract class System.Windows.Forms.CommonDialog
    public DialogResult ShowDialog()
        대화 상자에서 사용자의 선택을 반환한다. 확인 버튼을 누르면 OK를 반환
        하고 취소 버튼을 누르면 Cancel을 반환한다.
```

열거형 형식 DialogResult는 대화 상자에서 사용자가 선택한 버튼에 대응되는 값을 요소로 갖는다. 여러 값이 있으므로 주요 값만 알아본다.

```
enum System.Windows.Forms.DialogResult
    OK
        확인 버튼을 누르면 반환하는 값이다.
    Cancel
        취소 버튼을 누르면 반환하는 값이다.
```

CommonDialog는 추상 클래스이므로 인스턴스를 직접 생성할 수 없다. 파생 클래스인 ColorDialog, FontDialog, OpenFileDialog, SaveFileDialog, PrintDialog 등을 객체로 만들어 사용할 수 있다.

7.3.1 ColorDialog.

색상을 선택할 수 있는 대화 상자는 클래스 ColorDialog에 대응된다.

```
class System.Windows.Forms.ColorDialog
    public Color Color { get; set; }
        색상을 가져오거나 설정한다.
```

ColorDialog는 다음과 같이 생성한다.

```
ColorDialog dlg = new ColorDialog();
```

객체를 만들었다고 바로 대화 상자가 화면에 출력되는 것은 아니다. 화면에 출력하려면

```
dlg.ShowDialog();
```

메서드를 호출해야 한다. 다음 그림과 같은 대화 상자가 출력된다.

사용자는 이 대화 상자에서 색상을 선택하고 확인 버튼을 누르거나 취소 버튼을 누른다. 이 결과를 어떻게 받아야 할까? 사용자 선택은 메서드 ShowDialog()가 열거형 형식 DialogResult의 요소로 반환한다.

```
DialogResult result = dlg.ShowDialog();
```

사용자가 확인을 선택하면 result의 값은 DialogResult.OK가 되고 취소를 선택하면 DialogResult.Cancel이 된다.

사용자가 선택한 값은 프로퍼티 Color로 얻을 수 있다.

```
if (result == DialogResult.OK)
{
    Color color = dlg.Color;
```

이 프로퍼티는 대화 상자의 초깃값을 설정할 때도 사용된다. 메서드 ShowDialog()를 호출하기 전에

```
dlg.Color = Color.Red
```

와 같이 대화 상자의 초기 색상을 설정할 수 있다.

7.3.2 FontDialog.

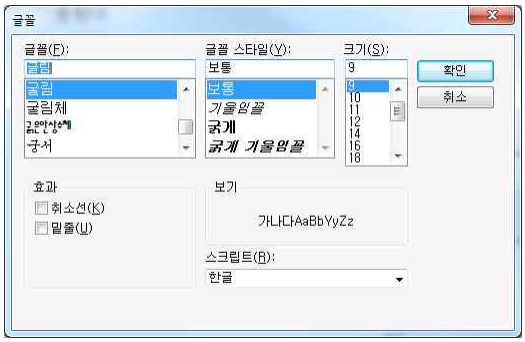
글꼴을 선택할 수 있는 대화 상자는 클래스 FontDialog에 대응된다.

```
class System.Windows.Forms.FontDialog
    public Font Font { get; set; }
        글꼴을 가져오거나 설정한다.
```

FontDialog의 사용법은 ColorDialog와 거의 같다.

```
FontDialog dlg = new FontDialog();
if (dlg.ShowDialog() == DialogResult.OK)
{
    Font font = dlg.Font();
```

와 같이 사용자가 선택한 font를 얻는다.



새 프로젝트 TestColorFontDialog를 추가하자. 폼에 다음과 같이 컨트롤을 배치하고 속성을 수정하자.

컨트롤	속성	이벤트
Button	(Name): colorButton Text: 바탕색	Click
Button	(Name): fontButton Text: 폰트	Click

파일 Form1.cs를 다음과 같이 수정한다.

프로젝트 TestColorFontDialog

Form1.cs

```
private void colorButton_Click(object sender, EventArgs e)
{
    ColorDialog dlg = new ColorDialog();
    dlg.Color = BackColor;
    if (dlg.ShowDialog() == DialogResult.OK)
        BackColor = dlg.Color;
}

private void fontButton_Click(object sender, EventArgs e)
{
    FontDialog dlg = new FontDialog();
    dlg.Font = fontButton.Font;
    if (dlg.ShowDialog() == DialogResult.OK)
        fontButton.Font = dlg.Font;
}
```

두 문장

```
dlg.Color = BackColor;
```

와

```
dlg.Font = fontButton.Font;
```

는 대화 상자의 초깃값을 설정한다.

7.3.3 FileDialog. 읽거나 쓰기 위한 파일을 선택하는 대화 상자가 **파일 대화 상자** (file dialog)이다. 추상 클래스 FileDialog에서 파생된 클래스를 사용한다.

```
abstract class System.Windows.Forms.FileDialog
    public string FileName { get; set; }
        선택된 파일 이름을 가져오거나 설정한다.
    public string Filter { get; set; }
        파일 필터링 형식을 가져오거나 설정한다.
    public int FilterIndex{ get; set; }
        현재 선택된 필터의 인덱스를 가져오거나 설정한다. 기본 값은 1이다.
    public string InitialDirectory { get; set; }
        이 대화 상자의 초기 경로를 설정한다.
    public string Title { get; set; }
        대화 상자의 제목을 설정하거나 가져온다.
    public DialogResult ShowDialog()
        대화 상자에서 사용자의 선택을 반환한다. 확인 버튼을 누르면 OK를 반환
        하고 취소 버튼을 누르면 Cancel을 반환한다.
```

C#은 읽기 위한 파일을 선택하기 위하여 클래스 OpenFileDialog를 제공한다. 쓰기 위한 파일을 선택하기 위하여 SaveFileDialog를 제공한다. 두 클래스 모두 추상 클래스 FileDialog에서 파생된다.

```
sealed class System.Windows.Forms.OpenFileDialog
sealed class System.Windows.Forms.SaveFileDialog
```

OpenFileDialog는

```
OpenFileDialog dlg = new OpenFileDialog();
dlg.Title = "파일 열기";
dlg.Filter = "텍스트 파일 (*.txt)|*.txt|기타 파일 (*.bat,*.ini)|*.bat;*.ini";
dlg.InitialDirectory = @"C:\";
if (dlg.ShowDialog() == DialogResult.OK)
{
```

와 같이 대화 상자 제목, 파일 경로, 파일 종류를 설정하여 출력할 수 있다.
SaveFileDialog도 거의 같은 방법으로 출력할 수 있다.

7.3.4 RichTextBox. 클래스 RichTextBox는 폰트, 글자색 등을 설정하여 서식이 있는 문서를 작성할 수 있도록 다양한 도구를 제공한다.

```
class System.Windows.Forms.RichTextBox
    public Color SelectionColor { get; set; }
        색상을 가져오거나 설정한다. 선택된 문자열의 색상도 바꾼다.
    public Color SelectionBackColor { get; set; }
        배경색을 가져오거나 설정한다. 선택된 문자열의 배경색도 바꾼다.
    public Font SelectionFont { get; set; }
        글꼴을 가져오거나 설정한다. 선택된 문자의 글꼴도 바꾼다.
    public string Text { get; set;}
        문자열을 가져오거나 설정한다.
```

문자열을 설정하거나 추가하려면 프로퍼티 Text를 사용한다.

```
richTextBox.Text = "설정할 문자열";
```

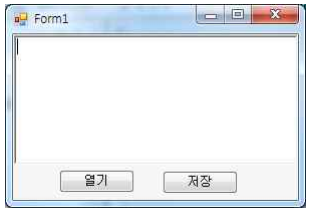
는 문자열을 설정한다. 여기에 문자열을 추가할 수 있다.

```
richTextBox.Text += "추가할 문자열";
```

는 추가할 문자열을 기존 문자열 뒤에 추가한다.

문자 파일을 읽어서 RichTextBox에 출력하는 프로그램을 작성해 보자. 새 프로젝트 TestFileDialog를 추가하자. 폼에 다음과 같이 컨트롤을 배치하고 속성을 수정한다.

컨트롤	속성	이벤트
RichTextBox	(Name): content	
Button	(Name): open Text: 열기	Click
Button	(Name): save Text: 저장	Click



Form1.cs를 다음과 같이 수정한다.

프로젝트 TestFileDialog

Form1.cs

```
private void open_Click(object sender, EventArgs e)
{
    OpenFileDialog dlg = new OpenFileDialog();
    dlg.Title = "파일 열기";
    dlg.Filter =
"모든 파일|*.*|텍스트 파일 (*.txt)|*.txt|기타 파일 (*.bat,*.ini)|*.bat;*.ini";
    dlg.InitialDirectory = @"C:\";
    if (dlg.ShowDialog() == DialogResult.OK)
        content.Text = File.ReadAllText(dlg.FileName);
}
private void save_Click(object sender, EventArgs e)
{
}
```

◆ 연습문제 7.3.1. 위 프로젝트의 버튼 저장을 클릭하면 content의 문자열이 파일에 저장되도록 메서드 save_Click()을 완성하시오.

7.4 UI 구성

윈도우즈 어플리케이션은 대부분 메뉴(menu)를 갖는다. 폼의 상단에 위치하는 **메인 메뉴**(main menu)는 어플리케이션으로 할 수 있는 모든 작업을 포함하는 것이 일반적이다. 팝업 창으로 활성화되는 **상황에 맞는 메뉴**(context menu)는 각 컨트롤에 대하여 자주 사용되는 항목으로 구성된다. **도구 모음** 또는 **툴 바**(tool strip)은 메뉴에서 자주 사용하는 항목을 아이콘으로 표시한다. **상태 표시줄**(status strip)은 주목할 만한

현재 상황을 표시한다. 이들을 구성하고 사용하는 방법에 대하여 알아보자.

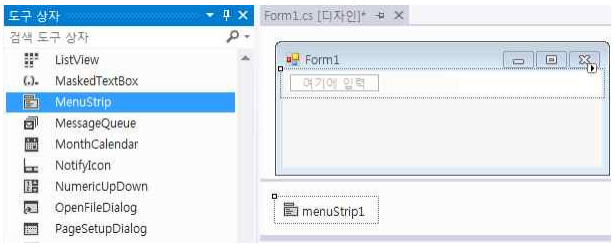
7.4.1 메인 메뉴. 메인 메뉴는 클래스 MenuStrip에 대응된다.

```
class System.Windows.Forms.MenuStrip
```

메인 메뉴에 포함되는 메뉴 항목은 클래스 ToolStripMenuItem에 대응된다.

```
class System.Windows.Forms.ToolStripMenuItem
    public event EventHandler Click
        메뉴 항목을 클릭할 때 발생한다.
```

메인 메뉴를 만들려면 도구 상자에서 MenuStrip을 끌어다 폼에 놓으면 된다. 자동으로 폼의 상단에 위치한다. 편집 창 아래쪽에 메뉴를 나타내는 menuStrip1 버튼이 생성된다.

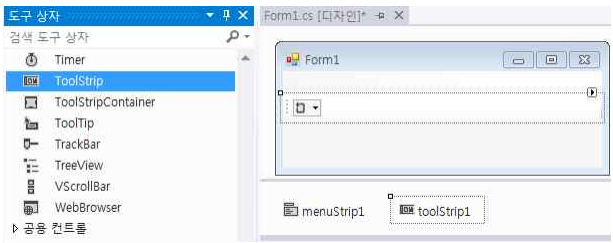


메인 메뉴의 입력란을 클릭하여 메뉴를 추가할 수 있다.

7.4.2 도구 모음. 메인 메뉴에서 자주 사용하는 메뉴를 골라 도구 모음을 만든다. 각 항목은 구분하기 쉽게 아이콘을 사용한다. 클래스 ToolStrip은 도구 모음을 나타낸다.

```
class System.Windows.Forms.ToolStrip
```

도구 상자에서 ToolStrip을 끌어 폼에 놓으면 도구 모음이 생성된다. 자동으로 메뉴 다음에 위치하며 편집 창 아래쪽에 toolStrip1 버튼이 생성된다.



도구 모음에는 아이콘으로 출력되는 버튼이 배치된다. 클래스 ToolStripButton으로 표현되며 일반적인 버튼과 마찬가지로 Click 이벤트를 발생시킨다.

```
class System.Windows.Forms.ToolStripButton
    public event EventHandler Click
        도구 모음의 아이콘을 클릭할 때 발생한다.
```

7.4.3 웹 브라우저. 클래스 WebBrowser는 웹 탐색을 지원하는 컨트롤을 나타낸다. 인터넷에서 HTML 파일을 받아 태그를 해석하고 화면에 출력한다.

```
class System.Windows.Forms.WebBrowser
    pulic Uri Url { get; set; }
        현재 문서의 URL을 가져오거나 설정한다.
    public void GoHome()
        사용자 홈페이지를 탐색한다.
    public bool GoBack()
        탐색 기록에서 이전 페이지를 탐색한다. 성공적으로 탐색하면 true, 그렇
        지 않으면 false를 반환한다.
    public bool GoForward()
        탐색 기록에서 다음 페이지를 탐색한다. 성공적으로 탐색하면 true, 그렇
        지 않으면 false를 반환한다.
    public void Navigate(string urlString)
        지정된 URL 문서를 로드한다.
    public event WebBrowserNavigatedEventHandler Navigated
        문서를 로드하기 시작했을 때 발생한다.
```

메서드 Navigate()는 주소를 매개 변수로 받아 들여 그 주소에 존재하는 파일을 읽어 출력한다.

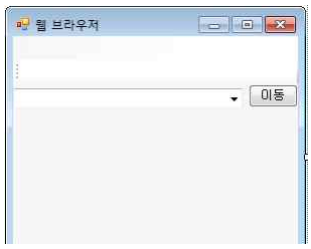
```
webBrowser.Navigate(new Uri("http://www.myhome.net"));
```

URL(uniform resource locator)는 자원이 있는 곳을 나타내는 문자열이다. 흔히 웹 주소와 동일한 것으로 취급된다. URI(uniform resource identifier)는 웹에 존재하는 자원을 나타내기 위한 문자열이며 URL보다 상위 개념이다. 클래스 Uri는 URI를 표현하기 위한 클래스이다.

```
class System.Uri
    public Uri(string uriString);
        해당 주소를 갖는 Uri를 생성한다.
```

새 프로젝트 TestMenu를 추가하자. 도구 상자에서 MenuStrip과 ToolStrip을 끌어 폼에 놓는다. 다음과 같이 컨트롤을 배치하고 속성을 수정하자.

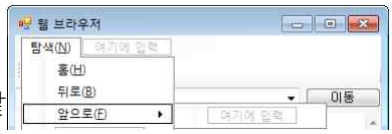
컨트롤	속성	이벤트
Form	Text: 웹 브라우저	
MenuStrip	(Name): menuStrip	
ToolStrip	(Name): toolStrip	
ComboBox	(Name): address	
Button	(Name): go Text: 이동	Click
WebBrowser	(Name): webBrowser	



메뉴를

탐색(&N)
홈(&H)

뒤로(&B)
앞으로(&F)



와 같이 구성하고 속성을 다음과 같이 수정한다.

메뉴	속성	이벤트
탐색(&N)	(Name): itemNavigate	
홈(&H)	(Name): itemHome	Click
뒤로(&B)	(Name): itemBack	Click
앞으로(&F)	(Name): itemForward	Click

도구 모음에서 ToolStripButton 추가를 눌러 버튼을 세 개 추가하고 이름을 각각

buttonHome
buttonBack
buttonForward

라 한다.

콤보 박스 address의 속성을 열고 Items 항목에서 (컬렉션)을 클릭한다. 문자열 컬렉션 편집기에

http://msdn.microsoft.com
http://www.daum.net

을 입력한다.

메뉴 아이템 홈(&H), 뒤로(&B), 앞으로(&H)를 하나씩 더블 클릭하여 이벤트 처리 함수를 등록한다. 또 도구 모음의 버튼을 하나씩 더블 클릭하여 이벤트 처리 함수를 등록한다. 버튼 go도 마찬가지로 이벤트 처리 함수를 등록한다. Form1.cs를 다음과 같이 수정한다.

프로젝트 TestMenu

Form1.cs

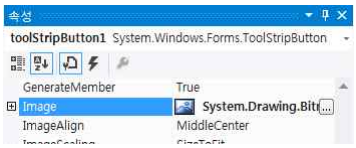
```
private void itemHome_Click(object sender, EventArgs e)
{
    webBrowser.GoHome();
}
private void itemBack_Click(object sender, EventArgs e)
{
    webBrowser.GoBack();
}
private void itemForward_Click(object sender, EventArgs e)
{
    webBrowser.GoForward();
}
private void buttonHome_Click(object sender, EventArgs e)
{
    webBrowser.GoHome();
}
private void buttonBack_Click(object sender, EventArgs e)
{
```

```
        webBrowser.GoBack();
    }
    private void buttonForward_Click(object sender, EventArgs e)
    {
        webBrowser.GoForward();
    }
    private void go_Click(object sender, EventArgs e)
    {
        webBrowser.Navigate(new Uri(address.Text));
    }
```

◆ 연습문제 7.4.1. 메뉴 항목을 추가하시오. 도구 모음에 대응되는 아이콘을 추가하시오.

◆ 연습문제 7.4.2. 도구 모음의 버튼의 속성에서 ToolTipText를 변경하시오. 프로그램을 실행하여 마우스 포인터를 도구 모음의 버튼에 올려놓고 기다려 출력되는 설명을 확인하시오.

도구 모음에 포함된 버튼의 아이콘은 **속성** 창에서 Image 항목으로 변경할 수 있다.



로컬 리소스(L)

이나

프로젝트 리소스 파일(P)

을 선택해야 한다. 로컬 리소스를 사용하면 해당 아이콘 만 바뀌고 프로젝트 리소스 파일을 사용하면 그림 파일이 폴더 Resource에 복사된다.

◆ 연습문제 7.4.3. 도구 모음의 버튼의 속성에서 이미지를 변경하시오.

◆ 연습문제 7.4.4. 콤보 박스의 이벤트 SelectedIndexChanged는 항목을 선택할 때 발생하는 이벤트이다. address의 이벤트 SelectedIndexChanged를 처리하는 함수를 작성하시오.

◆ 연습문제 7.4.5. 메뉴에서 홈(H), 뒤로(B), 앞으로(F) 등을 선택하여도 콤보 박스의 주소가 바뀌지 않는다. 클래스 WebBrowser의 프로퍼티 Url은 현재 문서의 주소를 포함하는 Uri 객체를 반환한다. 클래스 Uri의 메서드 ToString()은 주소를 문자열로 반환한다. 웹 브라우저의 이벤트 Navigated는 새 문서를 로드하기 시작하면 발생한다. 이 때 프로퍼티 Url의 값이 바뀐다. 콤보 박스에 주소가 나오도록 webBrowser의 이벤트 Navigated를 처리하는 함수를 작성하시오.

프로젝트 TestMenu를 실행하여 창의 크기를 변경해도 컨트롤들의 위치나 크기가 변하지 않는다. 이 문제점을 해결하기 위하여 Form1의 이벤트 Resize를 처리하는 함수

를 등록하자.

class System.Windows.Forms.Form
public event EventHandler Resize
폼의 크기를 변경하면 발생한다. 이 이벤트는 Control에 정의되어 있다.

코드를 열어 다음과 같이 수정하자.

```
private void Form1_Resize(object sender, EventArgs e)
{
    Point loc = go.Location;
    loc.X = Width - go.Width - 8;
    go.Location = loc;
    address.Width = loc.X - 6;
}
```

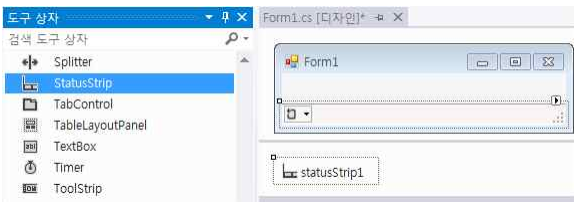
프로그램을 실행하고 폼의 크기를 변경하면 콤보 박스 address와 버튼 go의 크기와 위치가 달라짐을 확인할 수 있다. webBrowser의 크기를 바꾸는 것은 연습문제로 남긴다.

◆ 연습문제 7.4.6. 위 코드에 webBrowser의 크기를 바꾸는 문장을 추가하시오.

7.4.4 상태 표시줄. Num Lock 키가 눌린 것과 같은 프로그램 실행 정보를 제공할 때 **상태 표시줄**(status strip)을 사용한다. 실행 창의 아래쪽에 위치한다. 클래스 StatusStrip은 상태 표시줄에 나타낸다.

class System.Windows.Forms.StatusStrip

도구 상자에서 StatusStrip을 끌어 폼에 놓으면 자동으로 하단에 배치된다. 편집 창 아래쪽에 statusStrip1 버튼이 생성된다.



상태 표시줄에는 상태를 기술하는 문자열을 표시하는 StatusLabel, 작업의 진행 정도를 표시하는 ProgressBar와 두 종류의 버튼 DropDownButton, SplitButton이 포함될 수 있다.

StatusLabel은 클래스 ToolStripStatusLabel에 대응된다.

class System.Windows.Forms.ToolStripStatusLabel
public virtual string Text { get; set; }
출력될 문자열을 설정하거나 출력된 문자열을 가져온다.

상태 표시줄에 포함되는 DropDownButton은 클래스 ToolStripDropDownButton에 대응된다.

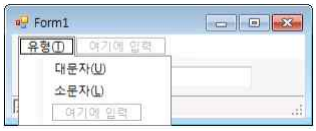
```
class System.Windows.Forms.ToolStripDropDownButton
    public event EventHandler Click
        항목을 클릭할 때 발생한다.
```

새 프로젝트를 TestStatusStrip이라는 이름으로 추가하자. 폼에 StatusStrip을 끌어다 놓는다. 다음과 같이 컨트롤을 배치하고 속성을 수정하자.

컨트롤	속성	이벤트
MenuStrip		
TextBox	(Name): textBox	TextChanged
StatusStrip		

메뉴를

유형(&T)
대문자(&U)
소문자(&L)



와 같이 구성하고 속성을 수정한다.

메뉴	속성	이벤트
유형(&T)	(Name): itemType	
대문자(&U)	(Name): itemUpper	Click
소문자(&L)	(Name): itemLower	Click

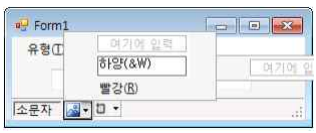
상태 표시줄을 다음과 같이 구성한다.

컨트롤	속성	이벤트
StatusLabel	AutoSize: False (Name): statusLabel Size: 50, 17 BorderSides: All BorderStyle: Sunken Text: 소문자 TextAlign: MiddleLeft	
DropDownButton	(Name): colorButton	

DropDownButton colorButton을 클릭하여

하양(&W)
빨강(&R)

을 배치하고 속성을 수정한다.



메뉴	속성	이벤트
하양(&W)	(Name): itemWhite	Click
빨강(&R)	(Name): itemRed	Click

메뉴의 대문자(&U)와 소문자(&L), 텍스트 박스 textBox, DropDownButton의 두 메뉴 아이템 하양(&W)와 빨강(&R)을 각각 더블 클릭하여 이벤트를 등록한다. 코드를 다음

과 같이 수정한다.

프로젝트 TestStatusStrip

Form1.cs

```
enum TextType
{
    Uppercase,
    Lowercase
}

public partial class Form1 : Form
{
    private TextType textType;

    public Form1()
    {
        InitializeComponent();
        textType = TextType.Lowercase;
    }

    private void ConvertText()
    {
        if (textType == TextType.Lowercase)
            textBox.Text = textBox.Text.ToLower();
        else
            textBox.Text = textBox.Text.ToUpper();
        textBox.SelectionStart = textBox.Text.Length;
    }

    private void textBox_TextChanged(object sender, EventArgs e)
    {
        ConvertText();
    }

    private void itemUpper_Click(object sender, EventArgs e)
    {
        statusLabel.Text = "대문자";
        textType = TextType.Uppercase;
        ConvertText();
    }

    private void itemLower_Click(object sender, EventArgs e)
    {
        statusLabel.Text = "소문자";
        textType = TextType.Lowercase;
        ConvertText();
    }

    private void itemWhite_Click(object sender, EventArgs e)
    {
        textBox.BackColor = Color.White;
    }

    private void itemRed_Click(object sender, EventArgs e)
    {
        textBox.BackColor = Color.Red;
```

```
        }
    }
```

메뉴를 선택하여 statusLabel의 변화를 살펴보자. 또 하양(&W)와 빨강(&R)을 눌러서 textBox의 색상이 바뀜을 확인하자.

다음 문장은 텍스트 박스에서 커서(cursor)의 위치를 줄의 끝으로 이동시킨다.

```
textBox.SelectionStart = textBox.Text.Length;
```

이 문장을 생략하면 커서의 위치가 앞으로 이동할 때가 있다.

◆ 연습문제 7.4.7. 상태 표시줄에 DropDownButton을 추가하시오.

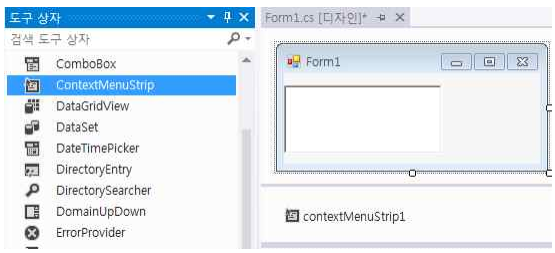
◆ 연습문제 7.4.8. 상태 표시줄에서 SplitButton을 사용해 보자. DropDownButton과 다른 점은 무엇인가?

7.4.5 상황에 맞는 메뉴. 각 컨트롤마다 자주 사용되는 항목을 메뉴로 구성한 것이 상황에 맞는 메뉴이다. 컨트롤에서 마우스를 오른 클릭하면 그 컨트롤에 설정된 상황에 맞는 메뉴가 출력된다. 상황에 맞는 메뉴는 클래스 ContextMenuStrip에 대응된다.

class System.Windows.Forms.ContextMenuStrip

상황에 맞는 메뉴에 포함되는 메뉴 항목은 클래스 ToolStripMenuItem에 대응된다. 메인 메뉴에 포함되는 메뉴 항목과 같다.

도구 상자에서 ContextMenuStrip을 끌어 폼에 놓으면 자동으로 배치된다. 편집 창 아래쪽에 contextMenuStrip1 버튼이 생성된다.



상황에 맞는 메뉴는 실행 화면에서 바로 보이는 것이 아니다. 컨트롤을 오른 클릭해야 출력된다. 따라서 상황에 맞는 메뉴는 컨트롤에 연결되어야 한다. 컨트롤의 속성에서 ContextMenuStrip을 설정하면 연결된다.

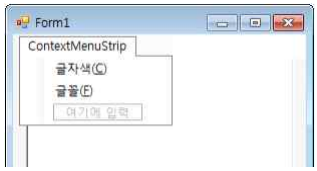
새 프로젝트 TestContextMenu를 추가하자. 폼에 다음과 같이 컨트롤을 배치하고 속성을 수정하자.

컨트롤	속성	이벤트
ContextMenuStrip	(Name): editorMenu	
RichTextBox	(Name): editor ContextMenuStrip: editorMenu	

상황에 맞는 메뉴 editorMenu를

글자색(&C)
글꼴(&F)

와 같이 구성하고 속성을 수정한다.



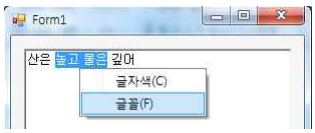
메뉴	속성	이벤트
글자색(&C)	(Name): itemColor	Click
글꼴(&F)	(Name): itemFont	Click

Form1.cs를 다음과 같이 수정한다.

프로젝트 TestContextMenu

Form1.cs

```
private void itemColor_Click(object sender, EventArgs e)
{
    ColorDialog dlg = new ColorDialog();
    dlg.Color = editor.SelectionColor;
    if (dlg.ShowDialog() == DialogResult.OK)
        editor.SelectionColor = dlg.Color;
}
private void itemFont_Click(object sender, EventArgs e)
{
    FontDialog dlg = new FontDialog();
    dlg.Font = editor.SelectionFont;
    if (dlg.ShowDialog() == DialogResult.OK)
        editor.SelectionFont = dlg.Font;
}
```



◆ 연습문제 7.4.9. editorMenu에 배경색을 바꾸는 메뉴를 추가하고 이벤트를 처리하시오.

제 8 장 윈도우즈 폼 그래픽스

도형 그리기, 그림 편집, 애니메이션, 게임과 같은 프로그램을 작성하려면 폼이나 컨트롤의 Graphics 객체에 대한 이해가 필수적이다. 선의 굵기, 색상 등을 결정해야 하며 글꼴을 선택할 수 있어야 한다. 이미지를 자르거나 합쳐서 새로운 그림을 만들어야 한다.

이 장에서는 클래스 Graphics 및 그와 관련된 클래스의 사용법에 대하여 알아본다.

8.1 그래픽스

폼에 문자열을 쓰고 도형을 그릴 수 있다. 그림 파일을 읽어서 출력하거나 배경으로 사용할 수 있다. 폼에 패널이나 픽쳐 박스를 배치하고 비슷한 작업을 수행할 수 있다.

폼이나 컨트롤을 그릴 때 이벤트 Paint가 발생한다. 이벤트 처리 함수에서 처음 그리거나 그린 것을 지우고 다시 그릴 때 컨트롤에 출력할 내용을 설정한다. 클래스 PaintEventArgs의 인스턴스를 매개 변수로 받는다.

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
```

클래스 PaintEventArgs의 프로퍼티 Graphics는 컨트롤의 그래픽스 정보를 갖고 있는 Graphics 객체를 반환한다.

class System.Windows.Forms.PaintEventArgs
    public Graphics Graphics { get; }
        Graphics 객체를 가져온다.

다음과 같이 Graphics 객체를 가져온다.

```
Graphics graphics = e.Graphics;
```

클래스 Graphics는 문자열을 출력하거나 도형을 그리는 메서드를 제공한다. 이 절에서 자세히 알아보자.

8.1.1 펜. 직선, 직사각형, 타원과 같이 선분 또는 곡선으로 이루어진 도형을 그리는 도구가 **펜**(pen)이다. 클래스 Pen에 대응된다. 색상과 굵기를 지정할 수 있다. 색상을 나타내는 클래스 Color는 6.2.8을 참조하자.

sealed class System.Drawing.Pen
    public Pen(Color color)
        주어진 색상으로 Pen 객체를 생성한다.
    public Pen(Color color, float width)
        주어진 색상과 굵기로 Pen 객체를 생성한다.
    public Color Color { get; set; }
        색상을 가져오거나 설정한다.
    public float Width { get; set; }
        굵기를 가져오거나 설정한다.

C#은 Pen 객체 생성을 도와주는 클래스 Pens를 제공한다. 다양한 Pen 객체가 클래

스 Pens에 정의되어 있다.

sealed class System.Drawing.Pens
public static Pen Red { get; }
폭이 1인 빨간색 Pen 객체를 가져온다.
public static Pen Green { get; }
폭이 1인 녹색 Pen 객체를 가져온다.
public static Pen Blue { get; }
폭이 1인 파란색 Pen 객체를 가져온다.

Pen 객체는 생성자로부터 얻을 수 있다.

```
Pen pen = new Pen(Color.Red);
```

클래스 Pens의 프로퍼티를 사용하면

```
Pen pen = Pens.Red;
```

같은 역할을 하는 객체를 얻는다.

선분 또는 곡선으로 이루어진 도형을 그리려면 클래스 Graphics의 인스턴스가 필요하다. 그리는 방법으로 다음과 같은 메서드를 제공한다.

sealed class System.Drawing.Graphics
public void DrawLine(Pen pen, float x1, float y1, float x2, float y2)
두 점 (x1, y1)과 (x2, y2)를 잇는 선분을 그린다.
public void DrawLine(Pen pen, Point p1, Point p2)
두 점 p1과 p2를 잇는 선분을 그린다.
public void DrawRectangle(Pen pen, float x, float y, float width, float height)
왼쪽 위 꼭짓점이 (x, y)이고 폭 width, 높이 height인 직사각형을 그린다.
public void DrawRectangle(Pen pen, Rectangle rect)
직사각형 rect를 그린다.
public void DrawEllipse(Pen pen, float x, float y, float width, float height)
왼쪽 위 꼭짓점이 (x, y)이고 폭 width, 높이 height인 직사각형에 내접하는 타원을 그린다.
public void DrawEllipse(Pen pen, Rectangle rect)
직사각형 rect에 내접하는 타원을 그린다.
public void DrawPolygon(Brush brush, Point[] points)
points를 꼭짓점으로 하는 다각형을 그린다.

선분은

```
graphics.DrawLine(new Pen(Color.Red, 1), 0, 0, 80, 100);
```

또는

```
graphics.DrawLine(Pens.Red, 0, 0, 80, 100);
```

으로 그릴 수 있다.

구조체 Rectangle은 직사각형을 나타낸다.

struct System.Drawing.Rectangle
public Rectangle(int x, int y, int width, int height)
왼쪽 위 꼭짓점이 (x, y)이고 폭 width, 높이 height인 Rectangle을 생성한다.
public int X { get; set; }
왼쪽 위 꼭짓점의 x 좌표를 가져오거나 설정한다.
public int Y { get; set; }
왼쪽 위 꼭짓점의 y 좌표를 가져오거나 설정한다.
public int Width { get; set; }
폭을 가져오거나 설정한다.
public int Height { get; set; }
높이를 가져오거나 설정한다.

모서리와 크기를 지정하여 직사각형으로 그리는

```
graphics.DrawRectangle(Pens.Red, x, y, width, height);
```

는 Rectangle 객체를 생성하여 그리는

```
graphics.DrawLine(Pens.Red, new Rectangle(x, y, width, height));
```

와 같은 결과를 출력한다.

8.1.2 **브러쉬**. 사각형이나 타원과 같이 영역을 갖는 도형의 내부를 칠할 때 사용되는 도구가 **브러쉬**(brush)이다. 추상 클래스 Brush에 대응된다.

abstract class System.Drawing.Brush

도형의 내부를 한 색상으로 칠하려면 클래스 SolidBrush를 사용한다. Brush의 파생 클래스이다.

sealed class System.Drawing.SolidBrush
public SolidBrush(Color color);
주어진 색상으로 SolidBrush 객체를 생성한다.
public Color Color { get; set; }
색상을 가져오거나 설정한다.

다양한 SolidBrush 객체가 클래스 Brushes에 정의되어 있다.

sealed class System.Drawing.Brushes
public static Brush Red { get; }
빨간색 SolidBrush 객체를 가져온다.
public static Brush Green { get; }
녹색 SolidBrush 객체를 가져온다.
public static Brush Blue { get; }
파란색 SolidBrush 객체를 가져온다.

Pen에서와 같이

```
Brush brush = new Brush(Color.red);
```

와

```
Brush brush = Brushes.Red;
```

는 같은 역할을 하는 SolidBrush 객체를 생성한다.

일정한 이미지를 반복하여 도형의 내부를 칠하려면 클래스 TextureBrush를 사용한다. Brush의 파생 클래스이다.

sealed class System.Drawing.TextureBrush
public TextureBrush(Image image)
주어진 이미지로 TextureBrush 객체를 생성한다.
public Image Image { get; }
이미지를 가져온다.

클래스 Graphics는 직사각형과 타원의 내부를 칠하는 메서드를 제공한다.

sealed class System.Drawing.Graphics
public void FillRectangle(Brush brush, float x, float y, float width, float height)
왼쪽 위 꼭짓점이 (x, y)이고 폭 width, 높이 height인 직사각형의 내부를 채운다.
public void FillRectangle(Brush brush, Rectangle rect)
직사각형 rect의 내부를 채운다.
public void FillEllipse(Brush brush, float x, float y, float width, float height)
왼쪽 위 꼭짓점이 (x, y)이고 폭 width, 높이 height인 직사각형에 내접하는 타원의 내부를 채운다.
public void FillEllipse(Brush brush, Rectangle rect)
직사각형 rect에 내접하는 타원의 내부를 채운다.
public void FillPolygon(Brush brush, Point[] points)
points를 꼭짓점으로 하는 다각형의 내부를 칠한다.

왼쪽 위 모서리가 (10, 10)이고 폭이 90 높이가 110인 직사각형의 내부를 파란색으로 칠하려면

```
graphics.FillRectangle(Brushes.Blue, 10, 10, 90, 110);
```

와 같이 적는다.

8.1.3 문자열. 클래스 Graphics는 문자열을 출력하는 메서드를 제공한다. 글꼴과 색상을 설정해야 하며 위치를 지정해야 한다.

sealed class System.Drawing.Graphics
public void DrawString(string s, Font font, Brush brush, float x, float y)
점 (x, y)에 문자열을 출력한다. font는 글꼴을, brush는 색상을 결정한다.

글꼴은 6.2.7에서 설명한 바와 같이

```
Font font = new Font("돋움", 12);
```

로 생성한다. 이 글꼴로 점 (30, 10)에 빨간색 문자열 "a string"을 쓰려면

```
g.DrawString("a string", font, Brushes.Red, 30, 10);
```

와 같이 적는다.

폼에 도형을 그리고, 문자열을 출력해보자. 새 프로젝트 TestGraphics1을 추가하자. 폼의 속성에서 BackColor를 Window로 수정하자.

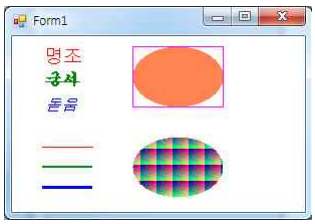
컨트롤	속성	이벤트
Form	BackColor: Window	Paint

Form1에 이벤트 Paint를 처리하는 함수를 등록하고 Form1.cs를 다음과 같이 수정하자.

프로젝트 TestGraphics1

Form1.cs

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;

    Font font = new Font("명조", 12);
    g.DrawString("명조", font, Brushes.Red, 30, 10);

    font = new Font("궁서", 12, FontStyle.Bold ¦ FontStyle.Strikeout);
    g.DrawString("궁서", font, Brushes.Green, 30, 35);

    font = new Font("돋움", 12, FontStyle.Italic);
    g.DrawString("돋움", font, Brushes.Blue, 30, 60);

    g.DrawLine(new Pen(Color.Red, 1), 30, 110, 80, 110);
    g.DrawLine(new Pen(Color.Green, 2), 30, 130, 80, 130);
    g.DrawLine(new Pen(Color.Blue, 3), 30, 150, 80, 150);

    Rectangle rect = new Rectangle(120, 10, 90, 60);
    g.DrawRectangle(Pens.Magenta, rect);
    g.FillEllipse(new SolidBrush(Color.Coral), rect);

    Bitmap bitmap = new Bitmap(16, 16);
    Random rand = new Random();
    for(int i=0; i<16; i++)
        for (int j = 0; j < 16; j++)
        {
            Color color = Color.FromArgb(16 * i, 16 * j, rand.Next(256));
            bitmap.SetPixel(i, j, color);
        }
    g.FillEllipse(new TextureBrush(bitmap), 120, 100, 90, 60);
}
```



위 프로젝트에서는 가로 16픽셀, 세로 16픽셀 비트맵을 TextureBrush의 패턴으로 사용하였다. 비트맵은 6.2.10에서 설명한 클래스 Bitmap으로 생성한다.

```
Bitmap bitmap = new Bitmap(16, 16);
```

각 픽셀의 색상을

```
bitmap.SetPixel(i, j, color);
```

와 같이 설정하였다. 위 프로젝트에서 출력된 타원의 내부는 반복되는 문양으로 채워졌음을 알 수 있다.

RGB 값을 갖는 Color 객체는 정적 메서드 FromArgb()로 생성한다.

struct System.Drawing.Color
public static Color FromArgb(int red, int green, int blue)
주어진 빨강, 녹색, 파랑 요소의 값을 갖는 Color 객체를 반환한다. 각 요소의 값은 0부터 255까지 이다.

◆ 연습문제 8.1.1. 위 프로젝트를 수정하여 출력된 돋움에 밑줄을 그으시오.

◆ 연습문제 8.1.2. 위 프로젝트에 두 점 (0, 0), (210, 160)을 잇는 선분을 그리는 코드를 추가하시오.

◆ 연습문제 8.1.3. 위 프로젝트에 세 점 (100, 100), (200, 100), (100, 200)을 잇는 삼각형의 내부를 녹색으로 칠하는 코드를 추가하시오. 또, 이 삼각형을 빨간색으로 그리는 코드를 추가시오.

◆ 연습문제 8.1.4. 새 프로젝트를 만들어 폼에 픽쳐 박스를 추가하고 픽쳐 박스에 프로젝트 TestGraphics1과 같은 결과를 출력하시오.

◆ 연습문제 8.1.5. 프로젝트 TestGraphics의 배경에 그림이 들어가도록 Form1의 속성에서 BackgroundImage를 설정하시오.

컴포넌트를 다시 그리려면 메서드 Invalidate()를 사용한다. 사용자 입력으로 그래픽스 환경이 바뀌면 이 메서드를 호출한다. PictureBox pb를 다시 그리려면

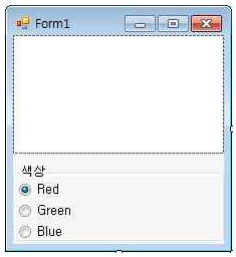
```
pb.Invalidate();
```

와 같이 입력한다.

새 프로젝트 TestGraphics2를 추가하자. 이 프로젝트에서는 사용자 입력을 받아 설정을 바꾸는 방법을 알아본다. PictureBox와 GroupBox를 배치하자. GroupBox에 세 라

디오 버튼을 추가하자.

컨트롤	속성	이벤트
PixturePox	(Name): pictureBox BackColor: Window	Paint
GroupBox	Text: 색상	
RadioButton	(Name): red Checked: True Text: Red	CheckChanged
RadioButton	(Name): green Text: Green	CheckChanged
RadioButton	(Name): blue Text: Blue	CheckChanged



속성에서 세 라디오 버튼의 이벤트 CheckChanged를 처리하는 함수를 등록한다.

프로젝트 TestGraphics2

Form1.cs

```
public partial class Form1 : Form
{
    private Color color;
    public Form1()
    {
        InitializeComponent();
        color = Color.Red;
    }

    private void pictureBox_Paint(object sender, PaintEventArgs e)
    {
        e.Graphics.FillRectangle(new SolidBrush(color), 20, 20,
            pictureBox.Width - 40, pictureBox.Height - 40);
    }

    private void red_CheckedChanged(object sender, EventArgs e)
    {
        if (red.Checked)
        {
            color = Color.Red;
            pictureBox.Invalidate();
        }
    }

    private void green_CheckedChanged(object sender, EventArgs e)
    {
        if (green.Checked)
        {
            color = Color.Green;
            pictureBox.Invalidate();
        }
    }

    private void blue_CheckedChanged(object sender, EventArgs e)
    {
        if (blue.Checked)
```

```
            {
                color = Color.Blue;
                pictureBox.Invalidate();
            }
        }
    }
```

◆ 연습문제 8.1.6. 위 프로젝트에 라디오 버튼 yellow를 추가하고 이벤트 처리함수를 작성하시오.

◆ 연습문제 8.1.7. 위 프로젝트에 직사각형의 외곽선을 그리는 코드를 추가하시오. 외곽선의 색상을 지정하는 라디오 버튼을 셋 배치하고 이벤트를 처리하시오.

◆ 연습문제 8.1.8. 새 프로젝트를 만들어 폼에 패널을 추가하고 이 패널에 프로젝트 TestGraphics2와 같은 결과를 출력하시오.

8.1.4 마우스. 마우스는 이동시키거나 버튼을 눌러 이벤트를 발생시킨다. 폼이나 컨트롤에서 발생하는 마우스 이벤트는 MouseClick, MouseDown, MouseUp, MouseMove 등이 있다.

class System.Windows.Forms.Form

public event EventHandler MouseClick
왼쪽 버튼을 클릭하면 발생한다. Form의 기본 클래스 Control에 정의되어 있다.

public event EventHandler MouseDown
버튼을 누르면 발생한다. Form의 기본 클래스 Control에 정의되어 있다.

public event EventHandler MouseUp
버튼 누름을 해제하면 발생한다. Form의 기본 클래스 Control에 정의되어 있다.

public event EventHandler MouseMove
마우스가 이동하면 발생한다. Form의 기본 클래스 Control에 정의되어 있다.

public void Invalidate()
화면을 지우고 다시 그린다. Form의 기본 클래스 Control에 정의되어 있다.

눌린 버튼, 클릭 회수 등 마우스에 관련된 정보는 클래스 MouseEventArgs의 인스턴스로 이벤트 처리 함수에 전달한다.

class System.Windows.Forms.MouseEventArgs

public MouseButtons Button { get; }
누른 버튼을 가져온다.

public Point Location { get; }
좌표를 가져온다.

public int X { get; }
좌표의 x 값을 가져온다. Location.X와 같다.

public int Y { get; }
좌표의 y 값을 가져온다. Location.Y와 같다.

사용자가 누른 마우스 버튼은 열거형 형식 MouseButtons로 표현된다.

enum System.Windows.Forms.MouseButtons

None
버튼을 누르지 않았음을 나타낸다.

Left
왼쪽 버튼을 눌렀음을 나타낸다.

Right
오른쪽 버튼을 눌렀음을 나타낸다.

Middle
가운데 버튼을 눌렀음을 나타낸다.

끌기(drag)는 마우스 왼쪽 버튼을 누르고 이동하는 것이므로 이벤트 MouseMove에서 확인한다.

```
private void Form1_MouseMove(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)
    {
    }
}
```

는 폼에서 발생하는 마우스 끌기 이벤트를 처리한다.

마우스를 끌어 도형을 그려보자. 새 프로젝트 TestMouse를 추가하자. 폼의 속성에서 세 이벤트 MouseDown, MouseMove, Paint를 처리하는 함수를 각각 등록하자.

컨트롤	속성	이벤트
Form		MouseDown MouseMove Paint

Form1.cs를 다음과 같이 수정하자.

프로젝트 TestMouse

Form1.cs

```
public partial class Form1 : Form
{
    private Point p1, p2;

    public Form1()
    {
        InitializeComponent();
        p1 = new Point(0, 0);
        p2 = new Point(0, 0);
    }

    private void Form1_MouseDown(object sender, MouseEventArgs e)
    {
        if (e.Button == MouseButtons.Left)
        {
            p1 = new Point(e.X, e.Y);
```

```
                p2 = p1;
                Invalidate();
            }
        }

        private void Form1_MouseMove(object sender, MouseEventArgs e)
        {
            if (e.Button == MouseButtons.Left)
            {
                p2 = new Point(e.X, e.Y);
                Invalidate();
            }
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            e.Graphics.DrawLine(Pens.Red, p1, p2);
        }
    }
```

폼이나 컨트롤의 메서드 Invalidate()는 컨트롤을 다시 그리도록 한다.

위 프로젝트에서는 마우스 왼쪽 버튼을 누르면 p1을 설정하고 화면을 다시 그린다.

```
p1 = new Point(e.X, e.Y);
Invalidate();
```

마우스를 끌면 p2를 설정하고 화면을 다시 그린다.

```
p2 = new Point(e.X, e.Y);
Invalidate();
```

이전 화면은 지워지고 두 점 p1과 p2를 잇는 선분이 그려진다.

◆ 연습문제 8.1.7. 위 프로젝트를 수정하여 두 점 p1과 p2를 모서리로 갖는 직사각형을 그리시오.

◆ 연습문제 8.1.8. 위 프로젝트를 수정하여 두 점 p1과 p2를 모서리로 갖는 내부가 채워진 직사각형을 그리시오.

◆ 연습문제 8.1.9. 선분, 직사각형, 타원을 선택하여 그릴 수 있도록 위 프로젝트에 메인 메뉴와 툴바를 추가하시오.

◆ 연습문제 8.1.10. 마우스로 한 점을 클릭하고 이동하여 다른 점을 클릭하면 두 점을 이어 선분을 그리는 프로젝트를 작성하시오.

◆ 연습문제 8.1.11. 프로젝트 TestMouse에서는 도형을 한 개만 그린다. 두 개 이상의 도형을 그리는 프로젝트를 작성하시오.

8.1.5 리소스. 프로그램에서 사용할 문자열, 아이콘, 이미지, 파일과 같은 자원을 **리소스**(resource)라 한다.

리소스를 등록하고 불러오는 방법을 알아보자. 새 프로젝트 TestResource을 추가하

자. 이 프로젝트에 리소스를 추가하려면 메뉴에서

프로젝트(P) -> TestResource 속성(P)

를 클릭하여 속성 창을 연다.

리소스 -> 리소스 추가(R) -> 기존 파일 추가(E)

를 클릭한다. 리소스에 기존 파일 추가 창에서 추가할 그림 파일을 선택한다. 솔루션 탐색기의 TestResource에 Resources 폴더가 생성되고 그림 파일이 복사된다. 하드디스크의 프로젝트 TestResource의 실제 디렉토리에도 폴더가 생성되고 파일이 복사된다.

여기서는 그림 파일의 이름을 MyPicture.bmp로 하였다. 확장자가 png, jpg, gif인 파일도 같은 방법으로 불러올 수 있다. 이 리소스는

```
Bitmap bitmap = Properties.Resources.MyPicture;
```

와 같이 불러온다. Properties는 네임스페이스이며 Resources는 클래스이다.

이 비트맵을 폼에 출력하려면 클래스 Graphics의 인스턴스가 필요하다.

sealed class System.Drawing.Graphics
public void DrawImage(Image image, float x, float y)
이미지의 왼쪽 위 꼭짓점이 컨트롤의 점 (x, y)에 오도록 그린다.
public void DrawImage(Image image, Point point)
이미지의 왼쪽 위 꼭짓점이 컨트롤의 점 point에 오도록 그린다.

Graphics의 메서드

```
graphics.DrawImage(bitmap, 0, 0);
```

는 컨트롤의 왼쪽 위 모서리에 맞춰 그림의 왼쪽 위 모서리가 오도록 그려준다.

폼의 속성에서 BackColor를 Window로 수정하고 Paint 이벤트를 처리하는 함수를 등록하자.

컨트롤	속성	이벤트
Form	BackColor: Window	Paint

Form1.cs를 다음과 같이 수정한다.

프로젝트 TestResource

Form1.cs

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Bitmap bitmap = Properties.Resources.MyPicture;
    Graphics g = e.Graphics;
    g.DrawImage(bitmap, 0, 0);
}
```

◆ 연습문제 8.1.12. 위 프로젝트의 코드

```
g.DrawImage(bitmap, 0, 0);
```

에서 꼭짓점의 좌표 (0,0)을 Point 객체로 바꾸시오.

◆ 연습문제 8.1.13. 그림의 왼쪽 위 모서리와 폼의 점 (100, 100)이 일치하도록 코드를 수정하시오.

◆ 연습문제 8.1.14. 폼의 왼쪽 위 모서리와 그림의 점 (100, 100)이 일치하도록 코드를 수정하시오.

◆ 연습문제 8.1.15. 그림을 마우스로 끌어 이동시키는 프로젝트를 작성하시오.

◆ 연습문제 8.1.16. 새 프로젝트를 만들어 폼에 패널을 배치하고, 패널의 Paint 이벤트를 처리하는 함수를 위 프로젝트와 같이 적어 그림을 그리시오.

◆ 연습문제 8.1.17. 새 프로젝트를 만들어 폼에 픽쳐 박스를 배치하고, 픽쳐 박스의 Paint 이벤트를 처리하는 함수를 위 프로젝트와 같이 적어 그림을 그리시오.

8.1.6 그림 파일. 그림 파일을 리소스에 등록하지 않고 직접 읽어 오려면 6.2.10에서 설명한 클래스 Bitmap의 생성자를 사용한다.

```
Bitmap bitmap = new Bitmap("그림 파일 경로");
```

새 프로젝트 TestFileLoad를 추가하자. 폼에 다음과 같이 컨트롤을 배치하고 속성을 수정하자.

컨트롤	속성	이벤트
Button	(Name): open Text: 열기	Click
Panel	(Name): panel BackColor: Window	Paint

이 프로그램은 그림 파일을 열어 패널에 그리도록 작성될 것이다. 버튼 open의 이벤트 Click, 패널 panel의 이벤트 Paint를 처리하는 함수를 각각 등록하고 Form1.cs를 다음과 같이 수정한다.

프로젝트 TestFileLoad

Form1.cs

```
public partial class Form1 : Form
{
    private Bitmap bitmap;

    public Form1()
    {
        InitializeComponent();
        bitmap = null;
```

```
        }

        private void open_Click(object sender, EventArgs e)
        {
            OpenFileDialog dlg = new OpenFileDialog();
            dlg.Title = "그림 파일 열기";
            dlg.Filter =
        "모든 파일 (*.*)|*.*|BMP (*.bmp)|*.bmp|JPEG (*.jpg,*.jpeg)|*.jpg;*.jpeg";
            dlg.FilterIndex = 3;
            if (dlg.ShowDialog() == DialogResult.OK)
            {
                bitmap = new Bitmap(dlg.FileName);
                panel.Invalidate();
            }
        }

        private void panel_Paint(object sender, PaintEventArgs e)
        {
            if (bitmap != null)
                e.Graphics.DrawImage(bitmap, 0, 0);
        }
    }
```

읽기 위한 파일 선택은 6.3.3에서 설명한 클래스 OpenFileDialog를 사용한다.

```
dlg.FilterIndex = 3;
```

는 대화 상자를 열 때 선택되는 필터를 JPEG로 설정한다.

◆ 연습문제 8.1.18. 대화 상자 클래스 SaveFileDialog와 클래스 Bitmap의 메서드 Save()를 써서 그림을 파일에 저장하시오.

◆ 연습문제 8.1.19. 위 프로젝트에서 패널을 픽쳐 박스로 바꾸어 동일한 작업을 수행하시오.

8.2 애니메이션

그림 여러 장을 차례로 출력하여 움직이는 것처럼 보이도록 만든 것을 **애니메이션**(animation)이라 한다. 기본적으로 애니메이션은 그림을 출력하는 것과 같은 방법으로 출력된다.

애니메이션을 출력할 때 쓰레드를 사용하는 경우가 많다. 이 쓰레드에서 사용할 함수는

```
private void Run()
{
    while (애니메이션을 출력할 조건)
    {
        그림 선택;
        화면 다시 그리기;
        Thread.Sleep(대기 시간);
    }
}
```

과 같은 형식으로 정의된다. 이 함수를 매개 변수로 하여 쓰레드를 생성하고

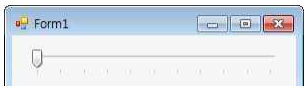
```
new Thread(Run).Start();
```

와 같이 애니메이션이 시작된다.

먼저 다음 프로젝트에서 애니메이션의 속도를 조절할 때 사용할 트랙 표시줄을 소개한다.

8.2.1 트랙 표시줄. **트랙 표시줄**(track bar)은 값의 변화를 쉽게 볼 수 있도록 일정한 간격으로 점을 찍어 표시한 것이다. 클래스 TrackBar에 대응된다.

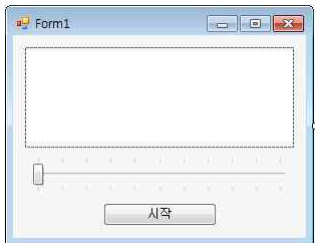
```
class System.Windows.Forms.TrackBar
    public int Minimum { get; set; }
        범위의 최대값을 가져오거나 설정한다.
    public int Maximum { get; set; }
        범위의 최소값을 가져오거나 설정한다.
    public int Value { get; set; }
        값을 가져오거나 설정한다.
    public event EventHandler Scroll
        마우스나 키보드를 조작하여 값을 변화시킬 때 발생한다.
```



TrackBar 객체의 범위와 간격을 따로 설정하지 않으면 가질 수 있는 값은 정수 0부터 10까지이다.

새 프로젝트 TestAnimation을 추가하자. 폼에 다음과 같이 컨트롤을 배치하고 속성을 수정하자.

컨트롤	속성	이벤트
Form		FormClosing
Panel	(Name): panel BackColor: Window	Paint
TrackBar	(Name): trackBar TickStyle: Both	Scroll
Button	(Name): start Text: 시작	Click



다음과 같이 연속 동작이 그려진 그림을 한 장 준비하고 리소스에 등록하자. 그림 파일 이름을 Figure.bmp라 하자.

폼의 이벤트 FormClosing을 처리하는 함수를 등록하자. 또 panel의 이벤트 Paint,

trackBar의 이벤트 Scroll, start의 이벤트 Click을 처리하는 함수를 등록하자. 코드를 다음과 같이 수정한다.

프로젝트 TestAnimation

Form1.cs

```
public partial class Form1 : Form
{
    private int width;
    private int height;
    private int numberOfFrames;
    private double shift;
    private int index;
    private bool isRunning;
    private int delay;

    public Form1()
    {
        InitializeComponent();
        numberOfFrames = 8;
        width = Properties.Resources.Figure.Width / numberOfFrames;
        height = Properties.Resources.Figure.Height;
        shift = (panel.Width - width) / (numberOfFrames - 1.0);
        index = 0;
        isRunning = false;
        delay = 11;
    }

    private void Run()
    {
        while (isRunning)
        {
            if (++index == 8)
                index = 0;
            panel.Invalidate();
            Thread.Sleep(delay * 100);
        }
    }

    private void panel_Paint(object sender, PaintEventArgs e)
    {
        Rectangle srcRect = new Rectangle(index * width, 0, width, height);
        Rectangle dstRect = new Rectangle((int)(shift*index), 0,
            width, height);
        Graphics g = e.Graphics;
        g.DrawImage(Properties.Resources.Figure, dstRect, srcRect,
            GraphicsUnit.Pixel);
    }

    private void trackBar_Scroll(object sender, EventArgs e)
    {
        delay = 11 - trackBar.Value;
    }

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

```
            if (isRunning)
            {
                start.Text = "시작";
                isRunning = false;
            }
            else
            {
                start.Text = "멈춤";
                isRunning = true;
                new Thread(Run).Start();
            }
        }

        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            isRunning = false;
        }

    }
```

이벤트 FormClosing은 폼을 닫을 때 발생한다. 폼을 닫으면 메인 쓰레드는 종료되지만

```
    new Thread(Run).Start();
```

로 생성된 쓰레드는 종료되지 않고 살아있게 된다. 따라서

```
    isRunning = false;
```

로 이 쓰레드를 종료시킨다.

◆ 연습문제 8.2.1. 화면을 클릭하면 애니메이션이 시작되고 다시 클릭하면 종료되도록 위 프로젝트에 코드를 추가하시오.

◆ 연습문제 8.2.2. 두 장의 그림을 번갈아 가면서 보여주는 애니메이션을 만드시오.

◆ 연습문제 8.2.3. 그림 여러 장을 무작위로 보여주는 애니메이션을 만드시오.

◆ 연습문제 8.2.4. 트랙 표시줄의 범위를 20으로 늘리시오.

◆ 연습문제 8.2.5. 폼의 두 이벤트 FormClosing과 FormClosed의 차이점을 말하시오.

8.2.2 더블 버퍼링. 프로젝트 TestAnimation을 자세히 보면 깜빡임이 있어 부자연스럽다. 연습문제 8.1.13과 같이 그림을 마우스로 그림을 끌어 움직여도 깜빡임이 있어 보기 좋지 않다. 폼에 패널을 추가하고 이 패널에서 같은 동작을 해도 마찬가지 현상이 나타난다. 그러나 픽쳐 박스에서는 깜빡임이 없이 매끄럽게 움직이는 것을 볼 수 있다.

컨트롤의 메서드 Invalidate()를 호출하면 먼저 메서드 OnPaintBackground()가 호출되어 화면을 지운다(실제로는 BackColor로 칠한다). 다음으로 Paint 이벤트를 처리하는 함수가 호출되어 그림을 그린다. 이 과정에서 깜빡임이 발생할 수 있다. 여기서는 OnPaintBackground()가 자주 호출되어 발생한다.

깜빡임을 없애는 대표적인 방법은 **더블 버퍼링**(double buffering)을 사용하는 것이다. 더블 버퍼링은 출력할 폼 또는 컨트롤과 크기가 같은 메모리 공간을 만든 다음 그 공간에 그림을 그리고 그림이 완성되면 이 공간을 화면에 보여주는 것을 말한다. 이와 같은 메모리 공간을 **후면 버퍼**(back buffer)라 한다.

더블 버퍼링을 구현하는 방법을 알아보자. 폼에 더블 버퍼링을 구현하려면 Form1.cs를 다음과 같이 수정한다.

Form1.cs

```
public Form1()
{
    InitializeComponent();

    SetStyle(System.Windows.Forms.ControlStyles.DoubleBuffer ¦
        System.Windows.Forms.ControlStyles.UserPaint ¦
        System.Windows.Forms.ControlStyles.AllPaintingInWmPaint,
        true);
}
```

◆ 연습문제 8.2.6. 연습문제 8.1.13의 폼에 더블 버퍼링을 구현하시오.

패널에 더블 버퍼링을 구현하는 방법은 약간 복잡하다. Form1.Designer.cs를 열어 namespace 구역 첫 부분에 다음과 같이 새로운 클래스를 정의한다.

Form1.Designer.cs

```
class DoubleBufferedPanel : System.Windows.Forms.Panel
{
    public DoubleBufferedPanel()
    {
        SetStyle(System.Windows.Forms.ControlStyles.DoubleBuffer ¦
            System.Windows.Forms.ControlStyles.UserPaint ¦
            System.Windows.Forms.ControlStyles.AllPaintingInWmPaint,
            true);
        UpdateStyles();
    }
}
```

클래스 Panel을 기본 클래스로 하는 파생 클래스 DoubleBufferedPanel을 만들고 스타일을 설정하였다.

Windows Form 디자이너에서 생성한 코드

를 확장하고

```
panel = new System.Windows.Forms.Panel();
```

을

```
panel = new DoubleBufferedPanel();
```

로 바꾼다.

위와 같이 패널에 더블 버퍼링을 구현하는 프로그램을 작성해 보자. 새 프로젝트 TestDoubleBuffering을 추가하자. 이 프로젝트에서는 마우스로 끌어 직사각형과 타원을 그리려 한다. 두 도형을 클래스로 작성하기 위하여 새로운 파일을 만든다.

메뉴에서

프로젝트(P) -> 클래스 추가(C)

를 클릭한다. 새 항목 추가 창에서

클래스

를 선택하고

이름(N): Shape.cs

를 적은 다음 추가 버튼을 클릭한다. 추가된 Shapes.cs에 다음과 같이 코드를 작성한다.

프로젝트 TestDoubleBuffering

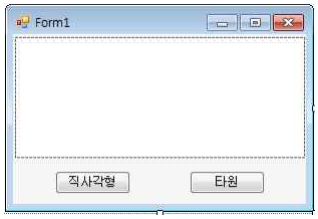
Shape.cs

```
using System.Drawing;

namespace TestDoubleBuffering
{
    abstract class Shape
    {
        protected Point p1;
        public Point P1
        {
            set { p1 = value; }
        }

        protected Point p2;
        public Point P2
        {
            set { p2 = value; }
        }

        public abstract void Draw(Graphics g);
    }

    class Rect : Shape
    {
        public override void Draw(Graphics g)
        {
            g.FillRectangle(Brushes.Red, p1.X, p1.Y,
                p2.X - p1.X, p2.Y - p1.Y);
        }
    }

    class Ellipse : Shape
    {
```

```
        public override void Draw(Graphics g)
        {
            g.FillEllipse(Brushes.Red, p1.X, p1.Y, p2.X - p1.X, p2.Y - p1.Y);
        }
    }
}
```

폼에 다음과 같이 컨트롤을 배치하고 속성을 수정하자.

컨트롤	속성	이벤트
Panel	(Name): panel BackColor: Window	Paint
Button	(Name): rect Text: 직사각형	Click
Button	(Name): ellipse Text: 타원	Click



패널 panel의 이벤트 Paint, 두 버튼 rect, ellipse의 이벤트 Click을 처리하는 함수를 등록하고 Form1.cs를 다음과 같이 수정하자.

Form1.cs

```
public partial class Form1 : Form
{
    private Shape shape;

    public Form1()
    {
        InitializeComponent();

        shape = new Rect();
    }

    private void rect_Click(object sender, EventArgs e)
    {
        shape = new Rect();
        panel.Focus();
    }

    private void ellipse_Click(object sender, EventArgs e)
    {
        shape = new Ellipse();
    }

    private void panel_Paint(object sender, PaintEventArgs e)
    {
        shape.Draw(e.Graphics);
    }

    private void panel_MouseDown(object sender, MouseEventArgs e)
    {
        shape.P1 = e.Location;
        shape.P2 = e.Location;
        panel.Invalidate();
    }
```

```
        private void panel_MouseMove(object sender, MouseEventArgs e)
        {
            if (e.Button == MouseButtons.Left)
            {
                shape.P2 = e.Location;
                panel.Invalidate();
            }
        }
    }
```

프로그램을 이대로 실행하면 도형을 그릴 때 깜빡임이 발생한다. 더블 버퍼링을 구현해 보자. Form1.Designer.cs를 다음과 같이 수정한다.

Form1.Designer.cs

```
namespace TestDoubleBuffering
{
    class DoubleBufferedPanel : System.Windows.Forms.Panel
    {
        public DoubleBufferedPanel()
        {
            SetStyle(System.Windows.Forms.ControlStyles.DoubleBuffer ¦
                System.Windows.Forms.ControlStyles.UserPaint ¦
                System.Windows.Forms.ControlStyles.AllPaintingInWmPaint,
                true);
            UpdateStyles();
        }
    }

    partial class Form1
    {
        private void InitializeComponent()
        {
            //this.panel = new System.Windows.Forms.Panel();
            this.panel = new DoubleBufferedPanel();
                ...
```

◇ 디자인을 수정하면 코드가 자동으로 수정되어 위와 다를 수 있다. 그러나 동작에는 문제가 없다.

◆ 연습문제 8.2.7. 프로젝트 TestAnimation에 더블 버퍼링을 추가하시오.

제 9 장 입출력

시간이 지남에 따라 순서대로 접근이 가능하도록 자료가 한 줄로 나열되어 있는 흐름을 **스트림**(stream)이라 한다. 시간에 지나면 처리할 대상이 도착하는 벨트 컨베이어와 유사한 개념이다.

키보드 입력은 스트림을 통하여 이루어진다. 사용자가 입력한 순서대로 자료에 접근할 수 있다. 개념적으로 스트림이 만들어 지는 것이다.

화면 출력, 파일에서 자료 읽기, 파일에 쓰기, 네트워크를 통한 자료 전송 등도 스트림을 통해 이루어진다. 오디오 재생과 비디오 재생도 역시 마찬가지이다.

스트림은 접근 단위에 따라 **비트 스트림**(bit stream), **바이트 스트림**(byte stream), **문자 스트림**(character stream)으로 나눌 수 있다. 단위에 제한을 두지 않는 스트림은 **파이프라인**(pipeline)이라 한다.

9.1 콘솔 입출력

콘솔(console)은 여러 가지 의미로 사용되는 단어이다. 여기서는 사용자가 키보드로 문자를 입력하고 컴퓨터 프로그램이 모니터에 문자를 출력하여 의사소통하는 방식을 말한다. **명령 줄 인터페이스**(CLI, command line interface)라고도 한다.
클래스 Console은 콘솔을 통한 표준 입력, 표준 출력 스트림을 나타낸다.

9.1.1 문장 읽기. 클래스 Console에는 문장을 읽는 두 정적 메서드 Read()와 ReadLine()이 정의되어 있다.

```
static class System.Console
    public static int Read()
        키보드로 입력한 내용을 문장 단위로 받아들인다. 한 문자씩 대응되는 유
        니코드로 변환한 다음 int형으로 반환한다.
    public static string ReadLine()
        키보드로 입력한 문장을 문자열로 반환한다. 문자열 끝의 줄 바꿈 문자
        ("\r\n")은 제거한다.
```

코드

```
int c = Console.Read();
```

는 한 문자를 읽어 c에 저장한다.

다음 예제는 문자를 읽어서 출력한다. 'Q' 또는 'q'를 입력하면 프로그램을 종료한다.

프로젝트 TestRead

```
using System;

class Program
{
    static void Main()
```

```
        {
            while (true)
            {
                int c = Console.Read();
                if (c == 'Q' || c == 'q')
                    break;
                Console.Write("{0} ", c);
            }
        }
    }
```

```
입력
    a1B !@
    q
출력
    97 49 66 32 33 64 13 10
```

출력에서 13과 10은 엔터키에 대응되는 두 문자를 나타낸다.

◆ 연습문제 9.1.1. 읽은 문장을 그대로 출력하는 코드를 추가하시오.

9.1.2 키 읽기. 클래스 Console의 정적 메서드 ReadKey()는 사용자가 누른 키의 정보를 반환한다.

class System.Console
public static ConsoleKeyInfo ReadKey(bool intercept)
키에 대한 정보가 구조체 ConsoleKeyInfo 객체로 반환된다. 매개 변수 intercept가 false이면 입력한 문자를 화면에 출력하고 true이면 출력하지 않는다.
public static ConsoleKeyInfo ReadKey()
ReadKey(false)와 같다.

키의 정보를 전달하는 구조체 ConsoleKeyInfo는 세 프로퍼티를 갖는다.

struct System.ConsoleKeyInfo
char KeyChar
입력된 문자를 반환한다.
ConsoleKey Key
키보드에서 눌린 키를 열거형 형식 ConsoleKey의 요소로 반환한다.
ConsoleModifiers Modifiers
Alt, Shift, Ctrl 키를 눌렀는지 여부를 열거형 형식 ConsoleModifiers의 요소로 반환한다.

열거형 형식 System.ConsoleKey는 키보드에 있는 모든 키를 요소로 갖는다. 여기서 나열하기에는 너무 많은 요소가 있다. 예를 들어 A에 대응되는 요소는 A이고 숫자 1에 대응되는 요소는 D1이다.

열거형 형식 System.Modifiers는 세 요소 Alt, Shift, Control로 구성되어 있다. 각각 Alt 키, Shift 키, Ctrl 키가 눌렸음을 나타낸다. 둘 이상의 키가 눌리면 비트 논리합 연산을 수행한다.

Alt 키가 눌렸는지 알아보려면

```
ConsoleKeyInfo info = Console.ReadKey();
bool altPressed = (info.Modifiers & ConsoleModifiers.Alt) != 0
```

와 같이 적어준다.

다음 예제는 입력한 키에 따라 다른 문자열을 출력한다. Esc 키를 누르면 프로그램을 종료한다.

프로젝트 TestReadKey

```
using System;

class Program
{
    static void Main()
    {
        while (true)
        {
            ConsoleKeyInfo info = Console.ReadKey(true);
            string str = "";
            switch (info.Key)
            {
                case ConsoleKey.A:
                    str = "apple";
                    break;
                case ConsoleKey.B:
                    str = "bee";
                    break;
                case ConsoleKey.C:
                    str = "crony";
                    break;
                case ConsoleKey.Escape:
                    Environment.Exit(0);
                    break;
                default:
                    str = "garbage";
                    break;
            }
            if ((info.Modifiers & ConsoleModifiers.Shift) != 0)
                Console.WriteLine(str.ToUpper());
            else if ((info.Modifiers & ConsoleModifiers.Alt) != 0)
                Console.WriteLine(Char.ToUpper(str[0]) + str.Substring(1));
            else
                Console.WriteLine(str);
        }

    }
}
```

```
입력
    a
    Alt + b
    Shift + c
    Esc
출력
    apple
    Bee
```

```
        CRONY
```

◆ 연습문제 9.1.2. 위 프로젝트에서

```
    info = Console.ReadKey(true);
```

를

```
    info = Console.ReadKey();
```

로 바꾸어 실행하고 결과를 비교하시오.

◆ 연습문제 9.1.3. 위 프로젝트에 info.KeyChar를 출력하는 문장을 추가하시오.

9.1.3 출력. 콘솔에 출력하려면 클래스 Console의 메서드 Write()나 WriteLine()을 사용한다. WriteLine()은 출력한 후 줄을 바꾸고 Write()는 줄을 바꾸지 않는 것 외에 다를 것이 없다. 이들은 여러 형태로 오버라이딩되어 있다. 대표적인 것만 알아보자.

```
class System.Console
    public static void Write(object value)
        value를 출력한다.
    public static void Write(string format, object arg0)
        format을 출력한다. arg0는 지정된 형식으로 format에 대입한다.
    public static void WriteLine()
        줄을 바꾼다.
    public static void WriteLine(object value)
        value를 출력하고 줄을 바꾼다.
    public static void WriteLine(string format, object arg0)
        format을 출력하고 줄을 바꾼다. arg0는 지정된 형식으로 format에 대입
        한다.
```

형식을 지정하지 않고 출력하려면

```
int x = 12;
Console.WriteLine(x);
```

와 같이 변수를 직접 써 준다. 결과적으로 x의 값이 출력된다.

```
Object obj;
Console.WriteLine(obj);
```

와 같이 객체를 출력하면 obj.ToString()이 호출되어 obj를 문자열로 변환한 다음 출력한다.

출력할 문자열을 써 주고 그 안에 출력 형식을 지정할 수 있다.

```
int x = 12;
Console.WriteLine("x의 값은 {0}", x);
```

중괄호({}) 안에 인덱스를 적으면 따옴표 뒤의 변수로 치환된다. 앞에서부터 순서대로 {0}, {1}, {2}, ...에 대응된다. 중괄호 안에는 출력 형식을 추가로 지정할 수 있다.

{0:C}와 같이 쌍점(:) 뒤에 특정한 기호를 적는다.

프로젝트 TestWriteLine

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine(
            "(C) 통화:. . . . . {0:C}\n" +
            "(E) 지수:. . . . . {1:E}\n" +
            "(G) 일반:. . . . . {0:G}\n" +
            "    (기본):. . . . . {0} (기본 = 'G')\n" +
            "(P) 백분율:. . . . . {1:P}\n" +
            "(X) 16진수:. . . . . {0:X}\n",
            123, 123.45);
    }
}
```

출력

```
    (C) 통화:. . . . . \123
    (E) 지수:. . . . . 1.234500E+002
    (G) 일반:. . . . . 123
        (기본):. . . . . 123 (기본 = 'G')
    (P) 백분율:. . . . . 12,345.00 %
    (X) 16진수:. . . . . 7B
```

정수 출력의 기본형은 형식을 G로 지정한 것과 같음을 알 수 있다.
정수의 출력 형식을 지정하는 문자는 다음 표와 같다.

문자	형식	보기
C/c	통화	\123
D/d	10진법	123
G/g	일반형	123
X	16진법(대문자)	7B
x	16진법(소문자)	7b

표 9.1 정수 출력 형식

자릿수를 지정하려면

```
Console.WriteLine("{0,10}", 123);
Console.WriteLine("{0,10:X}", 123);
```

과 같이 반점(,)을 찍고 자리수를 적는다. 앞에 맞추어 출력하려면 음수를 쓰고 뒤에 맞추어 출력하려면 양수를 쓴다. 자릿수를 적은 다음 출력 형식을 지정할 수 있다.

◆ 연습문제 9.1.4. 다음 코드를 실행하여 결과를 확인하시오

```
Console.WriteLine("{0,-10:x}", 123);
```

실수의 출력 형식을 지정하는 문자는 다음 표와 같다.

문자	형식	보기
E/e	지수	1.234500E+002
F/f	고정 소수점	123.45
G/g	일반형	123.45
N/n	숫자	123.45
P/p	백분율	12,345.00%

표 9.2 실수 출력 형식

자릿수를 지정하는 방법은 정수와 같다. 소수점 아래 자릿수는 출력 형식 다음에 지정한다.

```
Console.WriteLine("{0,10}", 123.45);
Console.WriteLine("{0,-10:F5}", 123.45);
```

첫째 문장은 123.45를 출력하고 둘째 문장은 123.45000을 출력한다.

◆ 연습문제 9.1.5. 실수 출력 형식 F와 G의 차이점을 설명하시오.

날짜와 시간을 나타내는 클래스는 System.DateTime이다. DateTime 객체는

```
DateTime dt = new DateTime(2030, 9, 17, 19, 8, 15);
```

와 같이 생성한다. dt를 출력하려면

```
Console.WriteLine("{0:F}", dt);
```

와 같이 적는다. 날짜의 출력 형식을 지정하는 문자는 다음 표와 같다.

문자	형식	보기
d	날짜	2030-09-17
D	날짜 요일	2030년 9월 17일 화요일
t	시분	오후 7:08
T	시분초	오후 7:08:15
f	날짜 시분	2030년 9월 17일 화요일 오후 7:08
F	날짜 시분초	2030년 9월 17일 화요일 오후 7:08:15
g	날짜 시분	2030-09-17 오후 7:08
G	날짜 시분초	2030-09-17 오후 7:08:15
M	월일	9월 17일
U	표준시	2030년 9월 17일 화요일 오전 10:08:15
Y	년월	2030년 9월

표 9.3 날짜 출력 형식

◆ 연습문제 9.1.6. 현재 시간을 출력하시오.

9.2 문자 파일

자료를 파일에 쓰는 방법과 파일에 저장된 자료를 읽어 오는 방법에 대하여 알아본다. 파일은 문자로 이루어진 자료를 저장한 **문자 파일**(text file)과 이진수로 이루어진 자료를 저장한 **이진 파일**(binary file)로 나눌 수 있다.

CPU로부터 파일에 대한 읽기/쓰기 권한을 받는 것을 **열기**(open)이라고 한다. 파일을 열면 스트림이 만들어지고 비로소 자료를 읽거나 쓸 수 있다. 파일을 열어서 원하는 작업을 수행한 다음 받은 권한을 반납하는 것을 **닫기**(close)라 한다. 파일을 닫으면 스트림이 닫힌다.

파일에서 자료를 읽을 위치를 **현재 위치**(current position)이라 한다. 파일을 열면 현재 위치는 0이다. 이 파일에서 크기가 n 바이트인 자료를 읽으면 현재 위치는 n으로 이동한다. 파일의 자료를 모두 읽으면 현재 위치는 파일의 크기와 같은 값을 가진다.

마찬가지로 파일을 열고 크기가 n 바이트인 자료를 파일에 쓰면 현재 위치는 n이 된다.

9.2.1 파일 전체 읽기. 클래스 File은 파일 만들기, 복사, 삭제, 이동 및 열기를 지원한다. 이 클래스에는 명시적으로 파일을 열지 않고 내용만 읽어오는 정적 메서드들이 정의되어 있다. 또 문자열을 파일에 저장하는 메서드도 있다.

```
class System.IO.File
    public static string ReadAllText(string path)
        파일  내용을 문자열로 반환한다. 반환된 문자열의 끝에는 줄 바꿈 문자
        ("\r\n")이 포함된다. path는 파일의 경로이다.
    public static string[] ReadAllLines(string path)
        파일의 각 줄이 요소인 문자열의 배열을 반환한다. 각 줄에서 줄 바꿈 문
        자는 제거된다. path는 파일의 경로이다.
    public static void WriteAllText(string path, string contents)
        경로가 path인 새 파일을 만들고 contents를 파일에 쓴다. 파일이 존재하
        면 덮어쓴다.
    public static void WriteAllLines(string path, string[] contents)
        경로가 path인 새 파일을 만들고 contents를 파일에 쓴다. 파일이 존재하
        면 덮어쓴다. contents의 각 요소는 파일의 한 줄이 된다.
```

파일 경로를 지정할 때

```
string text = File.ReadAllText("..\\..\\TestTextFile.cs");
```

와 같이 **디렉토리 분리 문자**(directory separator character, '\')는 \를 두 번 써서 표현한다. 이와 같은 번거로움을 피하려면

```
string text = File.ReadAllText(@"..\..\TestTextFile.cs");
```

와 같이 @을 문자열 앞에 써 주면 된다.

다음 예제는 파일을 읽어서 그대로 쓴다.

프로젝트 TestTextFile

```
using System;
using System.IO;
```

```
class Program
{
    static void Main()
    {
        string text = File.ReadAllText(@"..\..\TestTextFile.cs");
        File.WriteAllText(@"..\..\TextCopy.txt", text);

        string[] lines = File.ReadAllLines(@"..\..\TestTextFile.cs");
        File.WriteAllLines(@"..\..\LineCopy.txt", lines);
    }
}
```

두 파일 TextCopy.txt와 LineCopy.txt를 열어서 원본과 비교해 보자.

◆ 연습문제 9.2.1. 문자 파일을 읽어서 줄의 순서를 거꾸로 하여 다른 파일에 저장하시오.

◆ 연습문제 9.2.2. 문자 파일을 읽어서 각 줄의 문자의 순서를 거꾸로 하여 다른 파일에 저장하시오.

9.2.2 파일 일부분 읽기. 용량이 큰 파일을 모두 읽어서 메모리에 저장하면 공간도 많이 차지하고 처리 속도도 느려진다. 따라서 파일을 조금씩 읽어 들이는 방법이 필요하다.

클래스 StreamReader는 파일의 일부분을 읽어오는 메서드를 제공한다.

class File.IO.StreamReader

public StreamReader(string path)
StreamReader 객체를 생성한다.

public int Read()
문자를 읽어서 유니코드를 반환한다. 파일 끝에 도달하면 -1을 반환한다.

public int Read(char[] buffer, int index, int count)
문자 count개를 읽어서 buffer의 index에서 index+count-1까지 요소에 저장한다. 읽은 문자의 개수를 반환한다. 파일 끝에 도달하여 읽은 문자가 없으면 0을 반환한다.

public int ReadBlock(char[] buffer, int index, int count)
문자 count개를 읽어서 buffer의 index에서 index+count-1까지 요소에 저장한다. 읽은 문자의 개수를 반환한다. 파일 끝에 도달하여 읽은 문자가 없으면 0을 반환한다.

public string ReadLine()
한 문장을 읽는다. 읽은 문장을 반환한다. 파일 끝에 도달하여 읽은 문장이 없으면 null을 반환한다.

public string ReadToEnd()
파일 끝까지 읽는다. 파일 끝에 도달하여 읽은 문장이 없으면 빈 문자열("")을 반환한다.

public void Close()
파일을 닫는다.

클래스 StreamReader 객체는 읽기 전용 스트림이며

```
StreamReader src = new StreamReader(path);
```

와 같이 생성한다. path는 읽을 파일의 경로를 나타내는 문자열이다. 이 스트림에서 한 줄을 읽으려면

```
string line = src.ReadLine()
```

과 같이 적는다.

클래스 StreamWriter는 파일을 열어 놓고 자료를 여러 차례 저장할 수 있도록 한다. 문자로 된 자료를 파일에 저장하기 위한 두 메서드 Write()와 WriteLine()를 제공한다. 두 메서드는 여러 형태로 오버라이딩되어 있으며 Console의 메서드 Write(), WriteLine()과 사용법이 같다.

class File.IO.StreamWriter
- **public StreamWriter(string path)**
  StreamWriter 객체를 생성한다. 이미 파일이 있으면 덮어쓴다.
- **public StreamWriter(string path, bool append)**
  StreamWriter 객체를 생성한다. append가 true이면 파일 뒤에 이어 쓰며 false이면 파일에 덮어쓴다.
- **public void Write(object value)**
  스트림에 value를 쓴다.
- **public void Write(char[] buffer, int index, int count)**
  buffer의 요소 중 index부터 index + count - 1까지 쓴다.
- **public void Write(string format, object arg0)**
  format을 쓴다. arg0는 지정된 형식으로 format에 대입한다.
- **public void WriteLine()**
  줄을 바꾼다.
- **public void WriteLine(object value)**
  value를 쓰고 줄을 바꾼다.
- **public void WriteLine(char[] buffer, int index, int count)**
  buffer의 요소 중 index부터 index + count - 1까지 쓰고 줄을 바꾼다.
- **public void WriteLine(string format, object arg0)**
  format을 쓰고 줄을 바꾼다. arg0는 지정된 형식으로 format에 대입한다.
- **public void Close()**
  파일을 닫는다.

클래스 StreamWriter 객체는 쓰기 전용 스트림이며

```
StreamWriter dst = new StreamWriter(path);
```

와 같이 생성한다. path는 저장할 파일의 경로를 나타내는 문자열이다.

프로젝트 TestStream

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        int count = 0;
```

```
            string line;

            StreamReader src = new StreamReader(@"..\..\TestStream.cs");
            StreamWriter dst = new StreamWriter(@"..\..\Copy.txt");
            while ((line = src.ReadLine()) != null)
            {
                dst.WriteLine(line);
                count++;
            }
            dst.Close();
            src.Close();

            Console.WriteLine("{0}줄 복사", count);
        }
    }
```

```
출력
    23줄 복사
```

위 예제는 파일에서 문자열 자료를 한 문장씩 읽어서 처리하는 표준적인 방법을 제공한다.

```
while ((line = src.ReadLine()) != null)
```

은 src.ReadLine()에서 반환한 문자열을 line에 대입하고 그것이 null이 아니면 while 루프를 계속하라는 뜻이다.

◆ 연습문제 9.2.3. 파일을 읽어서 각 줄에 번호를 붙여 저장하시오.

◆ 연습문제 9.2.4. 파일에 포함된 단어의 개수를 출력하시오.

9.3 이진 파일

이진 파일을 CPU가 처리할 때는 문자 파일과 다르지 않다. 그러나 프로그램에서는 입출력 단위가 다르다. 문자 파일은 문자 단위로 입출력이 이루어지지만 이진 파일은 바이트 단위로 이루어진다. 또 이진 파일에는 줄의 끝을 나타내는 문자가 없다.

9.3.1 입출력. 일반적으로 파일을 열려면 여는 방법을 지정해야 한다. 있는 파일을 열 수도 있고 새로 만들 수도 있다. 열거형 형식 FileMode은 파일을 여는 방법을 요소로 갖는다.

```
enum System.IO.FileMode
    Append
        파일을 열어서 끝으로 이동한다. 파일이 없으면 새로 만든다.
    Create
        파일을 새로 만든다. 파일이 있으면 덮어 쓴다.
    CreateNew
        파일을 새로 만든다. 파일이 있으면 에러가 발생한다.
    Open
        파일을 연다. 파일이 없으면 에러가 발생한다.
    OpenOrCreate
```

파일이 있으면 열고 없으면 새로 만든다.

Truncate
파일을 연다. 이미 열려 있으면 내용을 모두 삭제한다.

파일을 열 때 파일을 읽기만 할 것인지 아니면 쓰기만 할 것인지도 지정해야 한다. 열거형 형식 FileAccess는 읽고 쓰는 방법을 요소로 갖는다.

enum System.IO.FileAccess

Read
읽기만 허용한다.

ReadWrite
읽기와 쓰기를 모두 허용한다.

Write
쓰기만 허용한다.

이진 파일에 접근하기 위한 클래스로 FileStream이 있다. 파일을 생성하거나 기존 파일을 열어 스트림을 생성한다. 자료를 읽거나 쓰는 다양한 방법을 제시한다.

class System.IO.FileStream

public FileStream(string path, FileMode mode)
경로가 path인 파일을 열어 읽기와 쓰기를 허용하는 스트림을 반환한다.

public FileStream(string path, FileMode mode, FileAccess access)
경로가 path인 파일을 열어 스트림을 반환한다.

public int Read()
현재 위치(current position)에서 한 바이트를 읽는다. 읽은 값을 int형으로 반환한다. 파일 끝에 도달하여 자료를 읽지 못하면 -1을 반환한다.

public int Read(byte[] array, int offset, int count)
현재 위치에서 최대 count 바이트를 읽어서 array[offset]부터 차례로 저장한다. 읽은 바이트 수를 반환한다. 파일의 끝에 도달하여 자료를 읽지 못하면 0을 반환한다.

public void WriteByte(byte value)
현재 위치에 value를 쓴다.

public void Write(byte[] array, int offset, int count)
현재 위치에 array[offset]부터 array[offset+count-1]까지 쓴다.

public long Seek(long offset, SeekOrigin origin)
origin으로부터 offset 바이트만큼 증가한 지점을 현재 위치로 지정한다.

public void Close()
파일을 닫는다.

읽기 위한 파일은

```
FileStream fs = new FileStream(path, FileMode.Open, FileAccess.Read);
```

와 같이 연다. 이 스트림에서 한 바이트를 읽으려면

```
int n = fs.Read();
```

와 같이 적는다.

새로운 파일을 열어서 한 바이트를 쓰려면

```
FileStream fs = new FileStream(path, FileMode.Create, FileAccess.Write);
fs.WriteByte(b)
```

와 같이 적는다.

FileStream의 생성자

```
FileStream fs = new FileStream(path, mode);
```

로부터 얻는 객체 fs는 읽기와 쓰기를 모두 허용한다. 따라서

```
FileStream fs = File.Open(path, mode, FileAccess.ReadWrite);
```

와 동일하게 동작한다.

FileStream의 메서드 Seek()는 origin으로부터 offset 바이트 이동한 곳으로 현재 위치를 지정한다. 둘째 인자 origin은 열거형 형식 SeekOrigin의 요소이다. 이 열거형 형식은 다음과 같은 요소로 구성되어 있다.

enum SeekOrigin

Begin
스트림의 맨 앞을 지정한다.

Current
스트림의 현재 위치를 지정한다.

End
스트림의 맨 뒤를 지정한다.

스트림 fs에 대하여

```
fs.Seek(-100, SeekOrigin.Current);
```

는 스트림의 현재 위치에서 100바이트 앞으로 이동한다. 이후 스트림에서 읽으면 그 위치에서 읽는다. 스트림에 쓰면 그 위치에 쓴다.

프로젝트 TestFileStream

```
using System;
using System.IO;
using System.Text;

class Program
{
    static void Main()
    {
        UTF8Encoding encoding = new UTF8Encoding(true);
        string path = Path.GetTempFileName();

        FileStream fs = new FileStream(path, FileMode.Create);
        byte[] data = encoding.GetBytes("산은 높고 물은 깊어");
        fs.Write(data, 0, data.Length);

        fs.Seek(0, SeekOrigin.Begin);
        byte[] b = new byte[1024];
        int count;
        while ((count = fs.Read(b, 0, b.Length)) > 0)
            Console.WriteLine(encoding.GetString(b, 0, count));
        fs.Close();
    }
```

```
}
```

```
출력
    산은 높고 물은 깊어
```

클래스 Path는 파일이나 디렉토리 경로를 나타내는 문자열에 대한 작업을 지원한다.

class System.IO.Path
public static string GetTempFileName()
디스크에 크기가 0바이트인 임시 파일을 만들고 해당 파일의 전체 경로를 반환한다.

◆ 연습문제 9.3.1. 위 예제의 path를 출력하여 임시 파일 이름을 확인하시오.

클래스 UTF8Encoding은 유니코드 UTF8과 관련된 작업을 지원한다. 문자열을 UTF8로 인코딩하거나 UTF8 바이트 배열을 문자열로 디코딩을 할 때 사용한다.

class System.text.UTF8Encoding
public byte[] GetBytes(string s)
문자열을 UTF8로 인코딩하여 반환한다.
public string GetString(byte[] bytes, int index, int count)
byte의 배열을 UTF8로 디코딩하여 반환한다.

◆ 연습문제 9.3.2. UTF8 이외의 인코딩 방법을 알아보자.

◆ 연습문제 9.3.3. 메서드 Seek()를 사용하여 파일의 둘째 바이트에 저장된 값을 읽으시오.

9.3.2 직렬화. 객체를 파일에 저장하려면 바이트의 배열로 변환하여야 한다. 이것을 **직렬화**(serialization)라 한다. 역으로 바이트의 배열을 객체로 변환하는 것을 **역직렬화**(deserialization)라 한다. 클래스 BinaryFormatter는 직렬화와 역직렬화를 지원한다.

class System.Runtime.Serialization.Formatters.Binary.BinaryFormatter
public void Serialize(Stream stream, Object obj)
obj를 직렬화하여 stream에 저장한다.
public object Deserialize(Stream stream)
stream으로부터 바이트의 배열을 읽어 객체로 역직렬화한다. stream에는 직렬화된 객체가 저장되어 있어야 한다.

Stream 객체 stream과 객체 obj에 대하여

```
BinaryFormatter bf = new BinaryFormatter();
bf.Serialize(stream, obj);
```

라 하면 obj는 바이트의 배열로 직렬화되어 stream에 저장된다.

클래스의 인스턴스를 직렬화하려면 어트리뷰트 Serializable을 사용하여야 한다.

```
[Serializable]
```

```
class Person
```

이와 같이 정의된 클래스 Person의 인스턴스는 BinaryFormater로 직렬화할 수 있다.

프로젝트 TestSerialization

```
using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

[Serializable]
class Person
{
    private string name;
    public string Name
    {
        get { return name; }
    }
    private int age;
    public int Age
    {
        get { return age; }
    }
    public Person(string name, int age)
    {
        this.name = name;
        this.age = age;
    }
}

class Program
{
    public static void Write(Stream stream, object obj)
    {
        BinaryFormatter bf = new BinaryFormatter();
        bf.Serialize(stream, obj);
    }

    public static object Read(Stream stream)
    {
        BinaryFormatter bf = new BinaryFormatter();
        return bf.Deserialize(stream);
    }

    static void Main(string[] args)
    {
        string path = Path.GetTempFileName();
        FileStream fs = new FileStream(path, FileMode.Create);
        Write(fs, 3);
        Write(fs, new Person("장길산", 21));
        Write(fs, 4.7);

        fs.Seek(0, SeekOrigin.Begin);
        while (fs.Position < fs.Length)
        {
            object obj = Read(fs);
            if (obj.GetType().Name == "Person")
```

```
            {
                Person p = (Person)obj;
                Console.WriteLine("{0}, {1}", p.Name, p.Age);
            }
            else
                Console.WriteLine(obj);
        }
        fs.Close();
    }
}
```

```
출력
    3
    장길산, 21
    4.7
```

위 예제는 객체를 직렬화하여 파일에 직접 저장한다. 직렬화한 바이트의 배열을 얻고 싶으면 스트림에서 읽어야 한다. 그러나 FileStream은 파일을 다루기 때문에 불필요한 자원을 사용하게 된다. 클래스 MemoryStream은 메모리를 스트림의 저장 공간으로 사용한다.

◆ 연습문제 9.3.4. 클래스 MemoryStream을 사용하여 객체를 직렬화하는 함수를 작성하시오.

◆ 연습문제 9.3.5. 클래스 MemoryStream을 사용하여 바이트의 배열을 객체로 역직렬화하는 함수를 작성하시오.

제 10 장 집합체

요소 수를 동적으로 변화시키면서 자료를 그룹으로 관리할 수 있도록 구조화한 것을 **집합체**(collection)라 한다. 용량을 늘릴 수 있으므로 자료 수가 늘어나도 개수에 상관없이 모두 수용할 수 있다.

C#은 여러 가지 집합체를 클래스로 제공한다. 단순히 자료의 목록을 만들 수도 있고 순서에 따라 정렬할 수도 있다. 자료를 삽입한 순서대로 꺼낼 수 있는 큐와 반대로 꺼내는 스택도 사용할 수 있다. 또 키와 값으로 이루어진 쌍을 요소로 가지며 키를 대입하여 값을 얻는 구조도 제공된다.

10.1 비정형 집합체

자료형을 결정하지 않고 모든 자료형을 요소로 가질 수 있는 집합체를 **비정형 집합체**(non-generic collection)라 한다. 요소의 자료형이 일정하지 않으므로 저장된 값을 object 객체로 불러오게 된다. 모든 자료형을 포괄할 수 있어야 하므로 어쩔 수 없지만 이것이 프로그램의 실행 속도를 느리게 한다. 이 범주에 포함되는 클래스는 모두 네임스페이스 System.Collections에 포함된다.

10.1.1 ArrayList. 클래스 ArrayList는 배열과 유사한 집합체를 만들어 준다. 생성과 동시에 요소 수를 정해야 하는 배열과 달리 ArrayList의 요소 수는 가변적이다. 자료 수를 확정할 수 없는 경우 충분한 공간을 잡아야 하는 배열에 비해 저장 공간을 효율적으로 사용하지만 실행 속도는 느리다.

```
class System.Collections.ArrayList
    public virtual int Capacity { get; set; }
        용량을 설정하거나 가져온다.
    public virtual int Count { get; }
        요소 수를 가져온다.
    public virtual object this[int index] { get; set; }
        인덱스가 index인 요소를 가져오거나 설정한다.
    public virtual int Add(object value)
        끝에 value를 요소로 추가한다. value가 추가된 인덱스를 반환한다.
    public virtual void RemoveAt(int index)
        인덱스가 index인 요소를 제거한다.
    public virtual IEnumerator GetEnumerator()
        열거자를 반환한다.
```

ArrayList를 생성하면 용량이 0이다.

```
ArrayList al = new ArrayList();
```

요소를 추가하려면 메서드 Add()를 사용한다.

```
al.Add("산은 높고");
```

al의 용량은 자동으로 늘어나고 첫째 요소로 문자열 "산은 높고"가 추가된다. 요소

값을 불러내려면 인덱서를 호출한다. 배열과 같이 대괄호([])를 사용한다.

```
Console.WriteLine(al[0]);
```

두 프로퍼티 Capacity와 Count는 각각 용량과 요소 수를 반환한다. 요소 수는 용량보다 작거나 같다. 요소가 추가되어 용량을 넘어서면 자동적으로 용량이 늘어난다.

ArrayList에 포함된 요소를 지우려면 메서드 RemoveAt()을 사용한다.

```
al.RemoveAt(1);
```

는 al의 인덱스 1인 요소(둘째 요소)를 제거하고 뒤에 있는 요소들을 앞으로 이동시킨다.

프로젝트 TestArrayList

```
using System;
using System.Collections;

class Program
{
    static void Main()
    {
        ArrayList al = new ArrayList();
        al.Add("산은 높고");
        al.Add(23);
        al.Add("물은 깊어");

        Console.WriteLine(al[1]);
        al.RemoveAt(1);

        Console.WriteLine("요소 수: {0}", al.Count);
        foreach(Object obj in al)
            Console.Write("{0} ", obj);
        Console.WriteLine();
    }
}
```

출력

```
23
요소 수: 2
산은 높고 물은 깊어
```

◆ 연습문제 10.1.1. 위 프로젝트의 al에 새 객체를 추가하시오.

◆ 연습문제 10.1.2. 위 프로젝트에서 요소를 추가할 때마다

```
Console.WriteLine(al.Capacity);
```

를 삽입하여 용량의 변화를 알아보자.

ArrayList 객체를 생성할 때 초기 용량을 설정할 수 있다.

```
ArrayList al = new ArrayList(3);
```

요소 수보다 작은 인덱스 n에 대하여

```
al[n] = 129;
```

와 같이 요소 값을 바꿀 수 있다.

◆ 연습문제 10.1.3. 다음 코드에서 잘못된 곳을 찾아 고치시오.

```
ArrayList list = new ArrayList();
list.Add(129);
list[1] = 111;
```

10.1.2 큐. 저장된 요소 중 맨 앞에 위치한 요소에만 접근할 수 있도록 제한하는 것이 좋을 때가 있다. 이와 같이 먼저 삽입된 요소에만 접근할 수 있도록 제한하는 방식을 **선입 선출**(FIFO, first-in first-out)이라 한다. 일반적으로 **큐**(queue)는 선입 선출 방식으로 동작하는 구조를 말한다.

클래스 Queue는 큐를 구현한다.

class System.Collections.Queue
public virtual int Count { get; }
요소 수를 가져온다.
public virtual void Enqueue(object value)
끝에 value를 요소로 추가한다.
public virtual object Dequeue()
첫째 요소를 제거한다. 제거한 요소를 반환한다.
public virtual object Peek()
첫째 요소를 반환한다.
public virtual IEnumerator GetEnumerator()
열거자를 반환한다.

새로운 큐는

```
Queue q = new Queue();
```

와 같이 생성한다. 문자열 "산은 높고"를 요소로 추가하려면

```
q.Enqueue("산은 높고");
```

와 같이 적는다. 저장된 자료를 꺼내려면 Dequeue()를 사용한다.

```
string str = (String) q.Dequeue();
```

이 메서드는 맨 처음 삽입된 요소를 반환하고 큐에서 이 요소를 제거한다. 요소를 제거하지 않고 단지 읽기만 하려면 메서드 Peek()를 사용한다.

프로젝트 TestQueue

```
using System;
using System.Collections;

class Program
{
    static void Main()
    {
```

```
            Queue q = new Queue();
            q.Enqueue(Console.ReadLine());
            Console.WriteLine(q.Peek());

            q.Enqueue(Console.ReadLine());
            Console.WriteLine(q.Peek());

            while(q.Count > 0)
                Console.Write("{0} ", q.Dequeue());
            Console.WriteLine();
            Console.WriteLine("q의 요소 수: {0}", q.Count);
        }
    }
```

```
입력
    산은 높고
    물은 깊어
출력
    산은 높고
    산은 높고
    산은 높고 물은 깊어
    q의 요소 수: 0
```

10.1.3 스택. 마지막으로 삽입된 요소에만 접근할 수 있도록 제한하는 방식을 **후입 선출**(LIFO, last-in first-out)이라 한다. 일반적으로 **스택**(stack)은 후입 선출 방식으로 동작하는 구조를 말한다.

클래스 Stack은 스택을 구현한다.

public class System.Collections.Stack
public virtual int Count { get; }
요소 수를 가져온다.
public virtual void Push(object value)
끝에 value를 요소로 추가한다.
public virtual object Pop()
마지막 요소를 제거한다. 제거한 요소를 반환한다.
public virtual object Peek()
마지막 요소를 반환한다.
public virtual IEnumerator GetEnumerator()
열거자를 반환한다.

스택에 자료를 삽입하려면 메서드 Push()를 사용한다.

```
Stack s = new Stack();
s.Push("산은 높고");
```

저장된 자료를 꺼내려면 Pop()을 사용한다.

```
string str = s.Pop();
```

이 메서드는 맨 나중에 삽입된 요소를 반환하고 스택에서 이 요소를 제거한다. 요소를 제거하지 않고 단지 읽기만 하려면 메서드 Peek()를 사용한다.

프로젝트 TestStack

```
using System;
using System.Collections;

class Program
{
    static void Main()
    {
        Stack s = new Stack();
        s.Push("산은 높고");
        s.Push("물은 깊어");
        s.Push(23);

        Console.WriteLine(s.Pop());
        Console.WriteLine("s의 요소 수: {0}", s.Count);
        Console.WriteLine(s.Peek());

        while(s.Count > 0)
            Console.Write("{0} ", s.Pop());
        Console.WriteLine();
        Console.WriteLine("s의 요소 수: {0}", s.Count);
    }
}
```

```
출력
    23
    s의 요소 수: 2
    물은 깊어
    물은 깊어 산은 높고
    s의 요소 수: 0
```

10.1.4 열거자. 세 클래스 ArrayList, Queue, Stack의 인스턴스로부터 요소를 나열하는 **열거자**(enumerator)를 얻을 수 있다. 열거자는 메서드 GetEnumerator()가 반환하는 인터페이스 IEnumerator 객체이다.

interface System.Collections.IEnumerator

object Current { get; }
현재 요소를 반환한다.

bool MoveNext()
다음 요소로 이동한다. 반환 값은 요소가 있으면 true, 없으면 false이다. IEnumerator 객체를 생성하여 Current로 첫째 요소를 얻으려면 MoveNext()를 한번 실행해야 한다.

void Reset()
첫째 요소 앞의 초기 위치로 이동한다. Current로 첫째 요소를 얻으려면 MoveNext()를 한번 실행해야 한다.

ArrayList 객체

```
ArrayList al = new ArrayList();
al.Add(12);
al.Add(34);
```

al로부터 열거자를 얻으려면

```
IEnumerator ie = al.GetEnumerator();
```

와 같이 적는다. ie의 요소를 가져오려면 먼저 MoveNext()를 실행해야 한다. 다음 요소가 있으면 그 요소로 이동하고 true를 반환한다. 만약 다음 요소가 없으면 false를 반환한다.

```
if (ie.MoveNext())
    Object obj = ie.Current;
```

프로퍼티 Current는 해당 요소를 반환한다. MoveNext()를 실행할 때마다 다음 요소로 이동하며 맨 처음 실행하면 첫 요소로 이동한다.

프로젝트 TestEnumerator

```
using System;
using System.Collections;

class Program
{
    static void ArrayListEnumerator()
    {
        Console.Write("ArrayList: ");
        ArrayList al = new ArrayList();
        al.Add(12);
        al.Add(34);
        IEnumerator ie = al.GetEnumerator();
        while (ie.MoveNext())
            Console.Write(" {0}", ie.Current);
        Console.WriteLine();
    }

    static void QueueEnumerator()
    {
        Console.Write("Queue: ");
        Queue q = new Queue();
        q.Enqueue(12);
        q.Enqueue(34);
        IEnumerator ie = q.GetEnumerator();
        while (ie.MoveNext())
            Console.Write(" {0}", ie.Current);
        Console.WriteLine();

    }

    static void StackEnumerator()
    {
        Console.Write("Stack: ");
        Stack s = new Stack();
        s.Push(12);
        s.Push(34);
        IEnumerator ie = s.GetEnumerator();
        while (ie.MoveNext())
            Console.Write(" {0}", ie.Current);
        Console.WriteLine();
    }

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

```
            ArrayListEnumerator();
            QueueEnumerator();
            StackEnumerator();
        }
    }
```

출력

```
ArrayList:  12 34
Queue:  12 34
Stack:  34 12
```

열거자로 값을 불러올 수는 있지만 값을 대입할 수는 없다. 또, 원래 집합체에 값을 대입하거나 추가하면 기존의 열거자는 유효하지 않으며 다시 집합체로부터 다시 얻어야 한다.

키워드 foreach는 내부적으로 메서드 GetEnumerator()를 사용한다. 따라서 인터페이스 IEnumerator를 사용하는 것과 마찬가지이다.

◆ 연습문제 10.1.4. 위 프로젝트의 열거자를 모두 foreach로 바꾸시오.

10.1.5 Hashtable. 클래스 Hashtable은 **키와 값으로 이루어진 쌍**(key/value pair)을 요소로 갖는 집합체이다. 키는 각 요소들을 구분하는 고유의 객체이며 모두 달라야 한다. 이름과 같은 역할을 하는 것이다.

```
class System.Collections.Hashtable
    public virtual int Count { get; }
        요소 수를 가져온다.
    public virtual object this[object key] { get; set; }
        키 key에 대응되는 값을 불러오거나 설정한다.
    public virtual ICollection Keys { get; }
        키를 모두 포함하는 ICollection 객체를 불러온다.
    public virtual void Add(object key, object value)
        키 key와 값 value로 이루어진 쌍을 요소로 추가한다.
    public virtual void Remove(object key)
        키 key를 가지는 요소를 제거한다.
```

Hashtable에 요소를 추가하려면 키와 값을 순서대로 써 준다.

```
Hashtable table = new Hashtable();
table.Add(key1, value1);
```

키가 key1이고 값이 value1인 요소가 추가된다.

값을 불러오려면 인덱서를 사용한다.

```
Object obj = table[key1];
```

와 같이 키를 인덱서의 매개 변수로 적어준다. 이때 obj의 값은 value1이 된다.

프로젝트 TestHashtable

```
using System;
```

```
using System.Collections;

class Program
{
    static void Main()
    {
        Hashtable h = new Hashtable(5);
        h.Add(0, "산은 높고");
        h.Add("dummy", 23);
        h.Add(1, "물은 깊어");

        Console.WriteLine(h["dummy"]);
        h.Remove("dummy");

        Console.WriteLine("키-값 쌍의 수: {0}", h.Count);
        foreach (Object key in h.Keys)
            Console.WriteLine("{0} : \t{1}", key, h[key]);
    }
}
```

```
출력
23
키-값 쌍의 수: 2
1 :     물은 깊어
0 :     산은 높고
```

10.1.6. SortedList. 클래스 SortedList는 Hashtable과 마찬가지로 키와 값으로 이루어진 쌍(key/value pair)을 요소로 갖는 집합체이다. 다만 키의 순서에 따라 오름차순으로 정렬하는 것이 다른 점이다.

class System.Collections.SortedList

public virtual int Count { get; }
요소 수를 가져온다.

public virtual int Capacity { get; set; }
용량을 설정하거나 가져온다.

public virtual object this[object key] { get; set; }
키 key에 대응되는 값을 불러오거나 설정한다.

public virtual ICollection Keys { get; }
키를 모두 포함하는 ICollection 객체를 불러온다.

public virtual void Add(object key, object value)
키 key와 값 value로 이루어진 쌍을 요소로 추가한다.

public virtual void Remove(object key)
키 key를 가지는 요소를 제거한다.

키의 순서는 인터페이스 IComparer로 결정한다.

interface System.Collectons.IComparer

int Compare(object x, object y)
x가 y보다 작으면 음수를 반환한다. 같으면 0을 반환한다. x가 y보다 크면 양수를 반환한다.

이 인터페이스를 구현하는 클래스를 만들어 메서드 Compare()를 오버라이딩해야 한다.

```
class KeyComparer : IComparer
{
    public int Compare(Object a, Object b)
    {
        // a가 b보다 크면 음수, 같으면 0, 크면 양수를 반환한다.
    }
}
```

클래스 KeyComparer를 **비교자**(comparer)라 한다. SortedList의 생성자를 호출할 때

```
SortedList sl = new SortedList(new KeyComparer());
```

와 같이 IComparer 객체를 매개 변수로 넘겨준다.

프로젝트 TestSortedList

```
using System;
using System.Collections;

class KeyComparer : IComparer
{
    public int Compare(Object a, Object b)
    {
        int ia = (int)a;
        int ib = (int)b;
        if (ia < ib)
            return -1;
        else if (ia == ib)
            return 0;
        else
            return 1;
    }
}

class Program
{
    static void Main()
    {
        SortedList sl = new SortedList(new KeyComparer());
        sl.Add(-1, "높고");
        sl.Add(-3, "산은");
        sl.Add(0, "dummy");
        sl.Add(3, "깊어");
        sl.Add(1, "물은");

        Console.WriteLine(sl[0]);
        sl.Remove(0);
        Console.WriteLine(sl.Count);

        foreach (int x in sl.Keys)
            Console.Write("{0} ", sl[x]);
        Console.WriteLine();
    }
}
```

출력

```
dummy
4
산은 높고 물은 깊어
```

클래스 SortedList의 생성자에

```
SortedList sl = new SortedList();
```

와 같이 IComparer 객체를 넘겨주지 않으면 내부적으로 정해진 기본 비교 객체를 사용한다.

◆ 연습문제 10.1.5. 기본 비교 객체를 사용하도록 위 프로그램을 수정하고 출력된 결과를 비교하시오.

◆ 연습문제 10.1.6. 프로젝트 TestSortedList의 집합체 sl의 정렬 방식이 주어진 방식과 반대가 되도록 KeyComparer를 수정하시오.

10.2 정형 집합체

요소로 가질 수 있는 자료형을 결정하여 제한한 집합체를 **정형 집합체**(generic collection)라 한다. 자료형이 정해져 있으므로 형 변환을 할 필요가 없으며 비정형 집합체에 비하여 실행 속도가 빠르다. 이 범주에 포함되는 클래스는 모두 네임스페이스 System.Collections.Generic에 포함된다.

10.2.1 HashSet<T>. 집합을 표현하는 정형 집합체가 HashSet<T>이다. T는 이 집합의 요소가 될 수 있는 자료형을 나타낸다. HashSet<T>는 집합을 표현하므로 동일한 요소가 여러 번 추가되어도 하나만 포함한다.

```
class System.Collections.Generic.HashSet<T>
    public int Count { get; }
        요소 수를 가져온다.
    public bool Add(T item)
        item을 요소로 추가한다. item이 이미 요소로 포함되어 있으면 false, 그
        렇지 않으면 true를 반환한다.
    public bool Remove(T item)
        item을 제거한다. 제거되었으면 true, 그렇지 않으면 false를 반환한다.
    public bool Contains(T item)
        item이 들어있는지 확인한다. 들어있으면 true, 그렇지 않으면 false를
        반환한다.
    public HashSet<T>.Enumerator GetEnumerator()
        열거자를 반환한다. HashSet<T>.Enumerator는 HashSet<T>의 내부 형식이
        며 열거자 IEnumerator를 구현한다.
```

요소의 자료형이 string인 HashSet은

```
HashSet<string> set = new HashSet<string>();
```

와 같이 정의하고 생성한다. set에 요소로 추가할 수 있는 객체는 모두 string형이어

야 한다. 요소는

```
set.Add("산은 높고");
```

와 같이 추가한다.

프로젝트 TestGenericHashSet

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main(string[] args)
    {
        HashSet<int> set = new HashSet<int>();
        set.Add(1);
        set.Add(2);
        set.Add(1);

        Console.WriteLine(set.Contains(2));

        foreach (int x in set)
            Console.WriteLine(x);
    }
}
```

출력

```
True
1
2
```

◆ 연습문제 10.2.1. 위 프로젝트의 set에 5를 추가하시오.

◆ 연습문제 10.2.2. 위 프로젝트의 set에서 1을 제거하시오.

10.2.2 List<T>. 클래스 List<T>는 비정형 집합체 ArrayList와 유사하다. 다만 T 자리에 자료형을 써야 하고 삽입할 수 있는 객체가 그 자료형이어야 한다는 것이 다르다.

class System.Collections.Generic.List<T>

public int Capacity { get; set; }
용량을 설정하거나 가져온다.

public int Count { get; }
요소 수를 가져온다.

public T this[int index] { get; set; }
인덱스가 index인 요소를 가져오거나 설정한다.

public int Add(T value)
끝에 value를 요소로 추가한다. value가 추가된 인덱스를 반환한다.

public void RemoveAt(int index)
인덱스가 index인 요소를 제거한다.

public List<T>.Enumerator GetEnumerator()
열거자를 반환한다. List<T>.Enumerator는 List<T>의 내부 형식이며 열거

자 IEnumerator를 구현한다.

T 자리에 string을 대입하여

```
List<string> list = new List<string>();
```

와 같이 list를 생성하면 삽입할 수 있는 객체는 모두 string 객체이어야 한다.

프로젝트 TestGenericList

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        List<string> list = new List<string>();
        list.Add("산은 높고");
        list.Add("The");
        list.Add("물은 깊어");

        Console.WriteLine(list[1]);
        list.RemoveAt(1);

        foreach (string str in list)
            Console.Write("{0} ", str);
        Console.WriteLine();
    }
}
```

출력

```
The
산은 높고 물은 깊어
```

◆ 연습문제 10.2.3. 위 예제의 list에 새 객체를 삽입하시오.

◆ 연습문제 10.2.4. 열거자로 list의 요소를 나열하시오.

◆ 연습문제 10.2.5. 삽입되는 자료가 int형인 List<T> 객체를 만드시오.

10.2.3 Queue<T>와 Stack<T>. 두 클래스 Queue<T>, Stack<T>는 각각 비정형 집합체 Queue, Stack과 유사하다. 다음 예제에서는 Stack<T>의 파생 클래스를 작성하여 사용한다.

프로젝트 TestGenericStack

```
using System;
using System.Collections.Generic;

struct Baggage
{
    private string name;
    private double volume;
```

```
        public double Volume
        {
            get { return volume; }
        }

        public Baggage(string name, double volume)
        {
            this.name = name;
            this.volume = volume;
        }

        public override string ToString()
        {
            return string.Format("{0}, {1}", name, volume);
        }
    }

    class Warehouse : Stack<Baggage>
    {
        private double maxVolume;
        private double remainder;

        public Warehouse(double maxVolume)
        {
            this.maxVolume = maxVolume;
            remainder = maxVolume;
        }

        public new void Push(Baggage baggage)
        {
            if (baggage.Volume < remainder)
            {
                base.Push(baggage);
                remainder -= baggage.Volume;
            }
        }
        public new Baggage Pop()
        {
            Baggage baggage = base.Pop();
            remainder += baggage.Volume;
            return baggage;
        }
    }

    class Program
    {
        static void Main()
        {
            Warehouse w = new Warehouse(10000);
            w.Push(new Baggage("공짜 짐", 30));
            w.Push(new Baggage("비싼 짐", 1200));
            w.Push(new Baggage("무거운 짐", 900));

            foreach(Baggage b in w)
                Console.WriteLine(b);
        }
    }
```

출력

```
무거운 짐, 900
비싼 짐, 1200
공짜 짐, 30
```

◆ 연습문제 10.2.6. 위 프로젝트의 w에 새 객체를 삽입하시오.

◆ 연습문제 10.2.7. 위 프로젝트의 클래스 Warehouse를 Queue<Baggage>의 파생 클래스로 바꾸시오.

10.2.4 SortedList<TKey, TValue>. 정형 집합체 SortedList<TKey, TValue>는 키와 값으로 이루어진 쌍을 저장하며 키의 순서에 따라 오름차순으로 정렬한다. 키의 순서를 결정하는 두 가지 방법을 소개한다.

첫째, 인터페이스 System.Collections.Generic.IComparer<T>를 구현하는 **비교자**(comparer)를 작성하는 것이다. 이 때 T는 TKey와 같거나 TKey의 기본 클래스들 중 하나여야 한다. 이 비교자는 IComparer<T>의 메서드 Compare()를 정의하여야 한다. 10.1.6의 클래스 KeyComparer와 유사하지만 두 매개변수가 T형이라는 점이 다르다.

```
class KeyComparer : IComparer<MyKey>
{
    public int Compare(MyKey a, MyKey b)
    {
    }
}
```

a가 b보다 작으면 음수, 같으면 영, 작으면 양수를 반환하도록 한다. 생성자를 호출할 때

```
SortedList<MyKey, MyValue> sl
    = new SortedList<MyKey, MyValue>(new KeyComparer());
```

와 같이 IComparer<MyKey> 객체를 매개 변수로 넘겨준다.

둘째, 내부적으로 정의된 기본 비교자를 사용하는 것이다. 대신 키의 클래스 정의에서 System.Collections.Generic.IComparable<T>를 구현한다.

```
class MyKey : IComparable<MyKey>
{
    public int CompareTo(MyKey key)
    {
    }
}
```

해당 객체(this)가 key보다 작으면 음수, 같으면 영, 크면 양수를 반환하도록 한다. 생성자를 호출할 때는 매개변수가 없어도 된다.

```
SortedList<MyKey, string> sl = new SortedList<MyKey, string>();
```

다음 예제는 둘째 방법을 구현한 것이다.

프로그램 TestGenericSortedList

```
using System;
using System.Collections.Generic;

class MyKey : IComparable<MyKey>
{
    private int x;
    public int X
    {
        get { return x; }
    }

    public MyKey(int x)
    {
        this.x = x;
    }

    public override string ToString()
    {
        return string.Format("{0}", x);
    }

    public int CompareTo(MyKey key)
    {
        if (x < key.x)
            return -1;
        else if (x == key.x)
            return 0;
        else
            return 1;
    }
}

class Program
{
    static void Main()
    {
        SortedList<MyKey, string> sl = new SortedList<MyKey, string>();
        sl.Add(new MyKey(3), "산은");
        sl.Add(new MyKey(5), "물은");
        sl.Add(new MyKey(6), "깊어");
        sl.Add(new MyKey(4), "높고");

        foreach (MyKey key in sl.Keys)
            Console.WriteLine("sl[{0}] = {1}", key, sl[key]);

    }
}
```

```
출력
    sl[3] = 산은
    sl[4] = 높고
    sl[5] = 물은
    sl[6] = 깊어
```

◆ 연습문제 10.2.8. IComparer<T>를 사용하도록 위 프로그램을 수정하시오.

◆ 연습문제 10.2.9. IComparer<T>와 IComparable<T>를 모두 사용하면 어느 비교자가 선택되는가?

정형 집합체 Dictionary<TKey, TValue>은 비정형 집합체 Hashtable과 유사하다.

◆ 연습문제 10.2.10. Dictionary<TKey, TValue>를 사용한 프로그램을 작성하시오.

◆ 연습문제 10.2.11. 클래스 Dictionary<TKey, TValue>는 두 키가 같은 지 판별할 때 인터페이스 System.Collections.Generic.IEqualityComparer<T> 객체를 사용한다. 이 인터페이스를 정의하여 사용하는 프로그램을 작성하시오.

10.2.5 LinkedList<T>. **연결 리스트**(linked list)는 **노드**(node)가 서로 연결된 구조를 이루고 있다. 노드는 자료와 함께 인접한 노드가 어느 것인지 알 수 있는 정보를 포함하고 있다. 연결 리스트는 **단순 연결 리스트**(singly linked list), **이중 연결 리스트**(doubly linked list), **원형 연결 리스트**(circularly linked list) 등이 있다.

LinkedListNode<T>는 노드를 나타내는 클래스이다.

```
sealed class System.Collections.Generic.LinkedListNode<T>
    public T Value { get; set; }
        노드에 포함된 값을 가져오거나 설정한다.
    public LinkedListNode<T> Previous { get; }
        이전 노드를 가져온다.
    public LinkedListNode<T> Next { get; }
        다음 노드를 가져온다.
```

클래스 LinkedList<T>는 이중 연결 리스트를 구현한다.

```
class System.Collections.Generic.LinkedList<T>
    public int Count { get; }
        포함된 노드 수를 반환한다.
    public LinkedListNode<T> First { get; }
        첫째 노드를 반환한다.
    public LinkedListNode<T> Last { get; }
        마지막 노드를 가져온다.
    public LinkedListNode<T> AddAfter(LinkedListNode<T> node, T value)
        node 다음에 새 노드를 삽입하고 값을 value로 설정한다. 삽입된 노드를
        반환한다.
    public LinkedListNode<T> AddBefore(LinkedListNode<T> node, T value)
        node 앞에 새 노드를 삽입하고 값을 value로 설정한다. 삽입된 노드를 반
        환한다.
    public LinkedListNode<T> AddFirst(T value)
        맨 앞에 새 노드를 삽입하고 값을 value로 설정한다. 삽입된 노드를 반환
        한다.
    public LinkedListNode<T> AddLast(T value)
        맨 뒤에 새 노드를 추가하고 값을 value로 설정한다. 삽입된 노드를 반환
        한다.
    public void Remove(LinkedListNode<T> node)
        노드를 제거한다.
    public bool Remove(T value);
        포함된 값이 value인 첫 노드를 제거한다.
    public void RemoveFirst();
```

첫 노드를 제거한다.
public void RemoveLast();
마지막 노드를 제거한다.

노드에 포함되는 자료가 int형인 연결 리스트는

```
LinkedList<int> list = new LinkedList<int>();
```

와 같이 생성한다. 노드를 추가하려면

```
list.AddFirst(3);
```

또는

```
list.AddLast(3);
```

과 같이 적어준다. 전자는 3을 포함하는 노드를 list의 맨 앞에 추가하고 후자는 맨 뒤에 추가한다.

list에 포함된 노드 중 맨 앞에 있는 것을 얻으려면

```
LinkedListNode<int> node = list.First;
```

와 같이 적어준다. 또

```
LinkedListNode<int> node = list.Last;
```

로 맨 뒤 노드를 얻는다.

다음 예제는 연결 리스트에 자료를 추가하면서 정렬한다.

프로젝트 TestLinkedList

```
using System;
using System.Collections.Generic;

class SortedLinkedList : LinkedList<int>
{
    public void Add(int n)
    {
        LinkedListNode<int> node = First;
        while (node != null && node.Value < n)
            node = node.Next;
        if (node == null)
            AddLast(n);
        else
            AddBefore(node, n);
    }
}

class ProgramLinkedList
{
    static void Main(string[] args)
    {
        SortedLinkedList list = new SortedLinkedList();
        list.Add(1);
        list.Add(-1);
        list.Add(2);
```

```
            list.Add(-2);

            foreach (int i in list)
                Console.WriteLine(i);
        }
    }
```

출력

```
    -2 -1 1 2
```

◆ 연습문제 10.2.12. 위 프로젝트의 AddBefore()를 AddAfter()로 수정하고 실행하여 결과를 비교하시오.

◆ 연습문제 10.2.13. LinkedList<double> 객체를 생성하고 자료를 추가하시오.

제 11 장 예외

프로그램 실행 중에 예기치 않은 상황이나 오류가 발생할 수 있다. 파일을 열어 읽으려 할 때 파일이 존재하지 않을 수 있다. 네트워크에 접속하려 할 때 연결되지 않거나 연결이 끊어질 수도 있다. 이와 같은 문제를 해결하기 위한 예외 처리 기능을 제공한다.

11.1 예외 발생 상황

프로그램 실행 중에 발생하는 예기치 않은 상황이나 오류를 **예외**(exception)라 한다.

예외가 발생하는 상황을 살펴보기 위하여 다음 코드를 실행해 보자.

```
int x = 1;
int y = 0;
Console.WriteLine(x / y);
```

다음과 같은 메시지가 콘솔에 출력되고 프로그램이 중단된다.

```
처리되지 않은 예외: System.DivideByZeroException: 0으로 나누려 했습니다.
   위치: Test.Main(String[] args) 파일 C:\...\MyFile.cs:줄 9
```

이 메시지는 예외 System.DivideByZeroException이 발생하였음과 예외에 관한 정보 및 발생한 위치를 알려준다. 마지막 내용 파일 C:\...\MyFile.cs:줄 9는 예외가 파일 MyFile.cs의 9번째 줄에서 발생하였음을 말해준다.

◆ 연습문제 11.1.1. 키워드 if를 써서 예외가 발생하지 않도록 처리하시오.

존재하지 않는 파일을 열어 보자.

```
System.IO.StreamReader reader = new System.IO.StreamReader("없는 파일");
```

상당히 긴 문장이 출력되고 프로그램이 중단된다.

```
처리되지 않은 예외: System.IO.FileNotFoundException: 'C:...
   위치: System.IO.__Error.WinIOError(Int32 errorCode, String maybeFullPath)
   ...
   위치: System.IO.StreamReader..ctor(String path)
   위치: Test.Main(String[] args) 파일 c:\...\MyFile.cs:줄 7
```

파일 C:\...\MyFile.cs의 7 번째 줄에서 예외 System.IO.FileNotFoundException이 발생하였음을 알 수 있다.

◆ 연습문제 11.1.2. 클래스 System.IO.FileInfo의 프로퍼티 Exists를 이용하여 예외가 발생하지 않도록 처리하시오.

예외는 사용자가 입력한 내용이나 프로그램 자체의 오류로부터 발생한다. 예외가 발생하면 프로그램 실행이 중단되며 사용자가 작업한 내용은 모두 삭제되고 만다. 따라서 예외가 발생하지 않도록 방지해야 하며 사용자가 잘못된 자료를 입력하지 않도록 유도해야 한다.

11.2 예외 처리

프로그래머는 예외가 발생하는 상황을 예상하고 이를 예방할 수 있다. 키워드 if를 써서 잘못된 입력에 대처할 수 있다. C#에서는 예외를 처리하는 또 다른 방법을 제공한다.

키워드 try는 코드를 실행하면서 예외를 검사한다.

```
try
{
    Console.WriteLine(x / y);
}
```

와 같이 적으면 x / y를 출력하면서 예외가 발생하는 지 검사한다. 예외가 발생하는 경우에 이를 처리하기 위하여 키워드 catch를 사용한다.

```
catch(DivideByZeroException e)
{
    Console.WriteLine("0으로 나눔");
}
```

예외 DivideByZeroException이 발생하면 catch 구역이 실행된다. 발생하지 않으면 무시된다. catch의 괄호 안에는 처리할 예외를 적어야 한다.

다음 프로젝트에서는 예외 DivideByZeroException을 처리하고 프로그램을 종료한다.

프로젝트 TestTryCatch1

```
using System;

class Program
{
    static void Main(string[] args)
    {
        int x = 1;
        int y = 0;
        try
        {
            Console.WriteLine(x / y);
        }
        catch (DivideByZeroException e)
        {
            Console.WriteLine(e.Message);
        }
        Console.WriteLine("실행 종료");
    }
}
```

```
출력
    0으로 나누려 했습니다.
    실행 종료
```

예외를 처리하지 않으면 프로그램 실행이 중단되지만 위와 같이 예외를 처리하면 프로그램은 정상적으로 실행된다.

실제로 예외는 클래스로 표현된다. 앞에서 사용한 DivideByZeroException, FileNotFoundException은 클래스의 이름이다. 예외를 나타내는 클래스는 이름이 Exception으로 끝나며 예외의 기본 클래스 Exception에서 파생된다. 클래스 Exception의 주요 멤버는 다음과 같다.

public class Exception
public virtual string Message { get; }
예외를 설명하는 메시지를 가져온다.
public virtual string StackTrace { get; }
호출 스택에 대한 문자열 표현을 가져온다.

프로퍼티 Message, StackTrace 또는 예외 객체를 출력하면 예외의 정보를 볼 수 있다. Message는 간단한 메시지를 반환하며 StackTrace는 예외가 발생한 경로를 자세히 반환한다. 예외가 발생한 경우

```
catch(DivideByZeroException e)
{
    Console.WriteLine(e.Message);
}
```

는 메시지를 출력한다.

◆ 연습문제 11.2.1. 위 프로젝트의 문장

```
Console.WriteLine(e.Message);
```

를

```
Console.WriteLine(e);
```

로 바꾸어 실행하여 예외 정보를 출력하시오.

◆ 연습문제 11.2.2. 클래스 System.IO.StreamReader의 생성자는 존재하지 않는 파일 경로를 주면 예외 FileNotFoundException을 발생시킨다. 이것을 처리하는 프로그램을 작성하시오.

둘 이상의 예외를 처리하려면

```
try { }
catch() { }
catch() { }
```

와 같이 catch를 필요한 만큼 연달아 쓴다. 개수에 제한은 없으며 catch의 괄호에는 예외 클래스를 지정해야 한다.

문자열을 int형으로 변환하는 메서드는 Int32.Parse()이다. 이 메서드의 매개 변수에 문자열을 대입하면 int형을 반환한다. 만약 문자열을 정수로 변환할 수 없으면 System.FormatException이 발생한다. 또 값이 너무 크거나 작아서 int형의 범위를 벗어나면 System.OverflowException이 발생한다.

프로젝트 TestTryCatch2

```
using System;

class Program
{
    static int ReadInt()
    {
        int? x = null;
        while (!x.HasValue)
        {
            try
            {
                x = Int32.Parse(Console.ReadLine());
            }
            catch (FormatException e)
            {
                Console.WriteLine("정수가 아닙니다.");
            }
            catch (OverflowException e)
            {
                Console.WriteLine("범위를 벗어났습니다.");
            }
        }
        return x.Value;
    }

    static void Main(string[] args)
    {
        int x = ReadInt();
        int y = ReadInt();
        try
        {
            Console.WriteLine(x / y);
        }
        catch (DivideByZeroException e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

```
입력
    afsd
    345
    9876543210
    0
출력
    정수가 아닙니다.
    범위를 벗어났습니다.
    0으로 나누려 했습니다.
```

위 프로젝트에서

```
catch (FormatException e)
```

의 다음 구역에서 매개 변수 e를 사용하지 않는다. 이 경우 e를 생략하고

```
catch (FormatException)
```

과 같이 적어도 된다.

◇ int?는 int 또는 null을 값으로 갖는 형식으로 Nullable<int>와 동일한 형식이다. 이와 같은 것을 nullable 형식이라 한다. 자세한 것은 13.7.2를 참조하기 바란다.

앞에서 설명한 바와 같이 예외는 모두 Exception의 파생 클래스이다. 따라서 여러 번 사용되는 catch를 하나로 해결할 수도 있다. 위 프로젝트의 코드

```
catch (FormatException e)
{
    Console.WriteLine("정수가 아닙니다.");
}
catch (OverflowException e)
{
    Console.WriteLine("범위를 벗어났습니다.");
}
```

는

```
catch (Exception e)
{
    Console.WriteLine("예외 발생");
}
```

과 같이 줄일 수 있다. 그러나 발생한 예외마다 처리하는 방법이 다르면 catch를 여럿 두는 것이 좋다.

◆ 연습문제 11.2.3. 메서드 Int32.Parse()는 매개 변수로 넘어오는 값이 null일 경우 예외 System.ArgumentNullException을 발생시킨다. 위 프로젝트에 이 예외를 처리하는 catch() 구역을 작성하시오.

예외가 발생하지 않고 정상적으로 실행되든지 아니면 예외가 발생하여 그것을 처리하든지 상관없이 마무리할 작업이 있으면 키워드 finally 다음에 적어 준다. 일반적인 형태는

```
try { }
catch() { }
finally { }
```

이다. try 구역을 시도하여 예외가 발생하면 catch 구역을 실행한 다음 finally 구역을 실행한다. 예외가 발생하지 않으면 catch 구역은 무시하고 finally 구역을 실행한다.

finally 구역에서는 사용자가 작업한 내용을 저장할 수도 있으며 열린 파일을 닫을 수도 있다.

다음 프로젝트는 파일을 열어 한 줄을 읽는다.

프로젝트 TestTryCatchFinally

```
using System;
using System.IO;
```

```
class Program
{
    public static void ReadFile(string path)
    {
        string line;
        StreamReader reader = null;
        try
        {
            reader = new StreamReader(path);
            line = reader.ReadLine();
        }
        catch (IOException)
        {
            Console.WriteLine("파일 {0}을(를) 읽을 수 없습니다.", path);
            return;
        }
        finally
        {
            if (reader != null)
                reader.Close();
        }
        Console.WriteLine(line);
    }

    static void Main(string[] args)
    {
        ReadFile(@"..\..\TestTryCatchFinally.cs");
    }
}
```

출력

```
using System;
```

메서드 ReadFile()은 파일을 열어 한 줄을 읽는다. 만약 파일이 없거나 파일 시스템에 문제가 생기면 예외 IOException이 발생하고 catch 구역을 실행한다.

```
return;
```

을 만나면 ReadFile()의 실행이 종료되고 Main()으로 되돌아간다. 그러나 이때에도 finally 구역이 실행된다. 예외가 발생하든 그렇지 않든 finally 구역의 코드

```
if (reader != null)
    reader.Close();
```

를 실행하여 reader를 닫는다.

◆ 연습문제 11.2.4. 위 프로젝트에서 IOException이 발생하는 경우를 찾아보고 설명하시오.

키워드 finally는 예외가 발생하든 그렇지 않든 해야 할 일이 있는 경우 사용한다. catch 구역에서 처리하지 않는 예외가 발생하여 프로그램이 종료되는 경우에도 finally 구역은 실행된다.

```
try
```

```
    {
        Console.WriteLine(Int32.Parse(Console.ReadLine());
    }
    catch(FormatException e)
    {
    }
    Console.WriteLine(1234);
```

를 실행하여 9876543210를 입력하면 OverflowException이 발생하여 마지막 줄은 실행되지 않고 프로그램이 종료된다. catch에서는 FormatException만 처리하기 때문이다. 그러나

```
    try
    {
        Console.WriteLine(Int32.Parse(Console.ReadLine());
    }
    catch(FormatException e)
    {
    }
    finally
    {
        Console.WriteLine(1234);
    }
```

를 실행하여 9876543210를 입력하면 OverflowException이 발생하여 프로그램이 종료되지만 그 전에 1234가 출력된다.

예외를 처리하는 catch 구역 없이 finally 구역만 있을 수도 있다.

```
    try
    {
    }
    finally
    {
    }
```

이 경우 예외가 발생하면 프로그램이 비정상적으로 종료되지만 어쨌든 finally 구역은 실행된다.

11.3 제공되는 예외

예외가 발생할 수 있는 상황은 다양하다. 각 상황에 대처할 수 있도록 Java는 발생할 수 있는 예외를 클래스로 제공한다. 그러나 예외들을 모두 파악하고 기억하는 것은 불가능에 가까우며 필요하지도 않다.

프로그래머는 자신의 코드에서 발생할 수 있는 예외를 직관적으로 이해하고 그것이 발생하지 않도록 미리 예방하게 된다. 따라서 예외 처리를 하지 않는 것이 일반적이다.

다음 코드는 예외를 발생시킬 가능성이 있다.

```
    void SetX(int n, double d)
    {
        x[n] = d;
    }
```

n 값이 0보다 작거나 x.length보다 크거나 같으면 예외가 발생할 것이다. 이 메서드를 사용하는 프로그래머는 n을 x의 인덱스 범위에 속하도록 설정할 것이다. 이와 같은 상황은 잘 통제되므로 예외가 발생할 일은 거의 없으며 예외 처리도 하지 않는다.

n 값이 x의 인덱스 범위를 벗어나면 어떤 예외가 발생하는지 알 수 있을까? 이 문제에 대한 시원한 답은 없다. 예외를 발생시켜 보거나 Java 홈페이지에서 찾아야 한다. 둘 다 번거로운 일이다.

◆ 연습문제 11.3.1. 위 코드를 포함하는 프로그램을 만드시오. 매개 변수 n의 값을 바꾸면서 실행해 보고 어떤 예외가 발생하는 지 확인하시오.

C#이 제공하는 클래스의 메서드, 프로퍼티, 인덱서가 발생시키는 예외는 비교적 찾기 쉽다. 메서드를 오른 클릭하여

정의로 이동(G)

을 선택하면 메타 데이터(부가 정보)를 보여준다. 숨겨진 코드를 열면 주석 처리된 부분에 예외가 기술되어 있다. C# 홈페이지에서도 비교적 쉽게 찾을 수 있다.
다음은 Console.WriteLine()의 정의로 이동하여 얻은 코드이다.

```
// 요약:
//     현재 줄 종결자를 표준 출력 스트림에 씁니다.
//
// 예외:
//   System.IO.IOException:
//     I/O 오류가 발생하는 경우
public static void WriteLine();
```

이 메서드에서는 IOException이 발생할 수 있음을 알 수 있다. 발생하는 상황에 대한 설명도 있다.

◆ 연습문제 11.3.2. 배열의 프로퍼티 Length에서 발생할 수 있는 예외를 확인시오.

◆ 연습문제 11.3.3. 클래스 String의 인덱서에서 발생할 수 있는 예외를 확인시오.

프로그래머가 예측할 수 없는 예외는 처리하는 것이 좋다. 네트워크나 파일을 사용하는 경우에는 어떤 일이 일어날지 알 수 없다. 네트워크에 연결되지 않았을 수 있으며 외부 저장소가 제거되었을 수도 있다. 따라서 파일이나 네트워크에서 발생할 수 있는 예외는 처리하는 것이 보통이다.

11.4 사용자 정의 예외

개발자는 필요에 따라 직접 예외를 정의하여 사용할 수 있다. 예외의 기본 클래스인 Exception 또는 그것으로부터 파생된 클래스에서 파생된 클래스를 만든다.

```
class System.Exception
    public Exception(string message)
```

클래스 Exception의 생성자에 message를 설정할 수 있다. 이 생성자를 호출하여 메

시지를 설정하며 프로퍼티 Message는 이것을 가져온다.

```
class MyException : Exception
{
    MyException()
        : base("Message of MyException")
    {
    }
    MyException(string message)
        :base(message)
    {
    }
}
```

메서드, 프로퍼티, 인덱서 등에서 예측했던 예외적인 상황이 되면 MyException 객체를 생성하고 키워드 throw 뒤에 적는다.

```
throw new MyException();
```

이 문장을 만나면 해당 메서드의 다음 부분은 실행을 중지하고 호출한 곳으로 되돌아간다. 호출한 부분에서는 예외 MyException을 처리하고 프로그램은 계속 실행하거나 예외를 출력하고 실행을 중지한다.

프로젝트 TestUserDefinedException

```
using System;

enum Fruit
{
    Apple,
    Pear,
    Peach
}

class FruitRangeOutOfBoundsException : Exception
{
    public FruitRangeOutOfBoundsException()
        : base("Fruit의 범위를 벗어났습니다.")
    {
    }
}

class Foo
{
    private Fruit fruit;
    public Fruit Fruit
    {
        set
        {
            if (Fruit.Apple <= value && value <= Fruit.Peach)
                fruit = value;
            else
                throw new FruitRangeOutOfBoundsException();
        }
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Foo foo = new Foo();
        try
        {
            foo.Fruit = (Fruit)5;
        }
        catch (FruitRangeOutOfBoundsException e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

출력

```
Fruit의 범위를 벗어났습니다.
```

◆ 연습문제 11.4.1. 다음 함수에서 사용하는 예외 클래스 UnknownOperationException울 작성하시오.

```
public static double Operate(char op, double x, double y)
{
    switch (op)
    {
        case '+':
            return x + y;
        case '-':
            return x - y;
        default:
            throw new UnknownOperationException("알 수 없는 연산: " + op);
    }
}
```

◆ 연습문제 11.4.2. 예외를 발생시키는 인덱서를 작성하시오.

C#에서 제공하는 예외도 키워드 throw로 발생시킬 수 있다. 그 방법은 사용자가 정의하는 예외와 같다.

```
public int GetX(int n)
{
    if (0 <= n && n < x.Length)
        return x[n];
    else
        throw new IndexOutOfRangeException();
}
```

예외 클래스 IndexOutOfRangeException은 인덱스가 범위를 벗어났을 때 발생하는 예외이다.

제 12 장 쓰레드

프로그램을 작성하여 실행하면 **프로세스**(process)가 생성된다. 프로세스에서는 프로그래머가 지시한 작업이 수행된다. 상황에 따라 한 작업이 연속적으로 수행될 수도 있고 여러 작업이 병렬로 수행될 수도 있다.

독립적으로 수행되는 가장 작은 작업 단위를 **쓰레드**(thread)라 한다. 프로세스에는 한 개 이상의 쓰레드가 존재한다. 엉성하게 표현하면 프로세스는 쓰레드로 구성된다고 할 수 있다. 프로그래머는 쓰레드를 생성할 수 있으며 프로그래머가 직접 요구하지 않아도 CPU에 의하여 내부적으로 생성될 수도 있다.

일반적으로 프로그램이 실행되면 프로세스가 하나 생성된다. 이 프로세스에는 메서드 Main()과 운명을 같이 하는 **메인 쓰레드**(main thread)가 하나 생성된다.

한 프로세스는 적어도 한 쓰레드를 가지며 여러 쓰레드를 동시에 가질 수도 있다. 여러 쓰레드가 동시에 수행되는 것을 **멀티쓰레딩**(multithreading)이라 한다.

12.1 쓰레드의 시작과 끝

프로그래머가 생성과 종료를 명시하지 않는 메인 쓰레드는 자동으로 생성되며 메서드 main()의 문장들을 처리하고 종료된다. 일반적으로 메인 쓰레드가 종료되면 프로그램이 종료된다.

프로그래머가 생성하는 쓰레드는 클래스 Thread의 인스턴스이다. 이 클래스는 쓰레드를 운용하기 위한 메서드를 제공한다.

```
sealed class System.Threading.Thread
    public static void Sleep(int milliseconds)
        실행 중인 쓰레드를 milliseconds 동안 일시 중단한다. milliseconds의
        단위는 1/1000초이다.
    public Thread(ThreadStart start)
        start를 실행할 쓰레드 객체를 생성한다.
    public bool IsAlive { get; }
        쓰레드 상태를 반환한다. 쓰레드가 시작되었으며 종료되지 않았으면 true
        그렇지 않으면 false를 반환한다.
    public void Start()
        쓰레드를 시작한다.
    public void Join()
        쓰레드가 종료될 때까지 기다린다.
```

대리자 ThreadStart는 쓰레드에서 실행할 메서드의 형식을 지정한다(3.3.3). 매개변수와 반환값이 없다.

```
delegate void System.Threading.ThreadStart()
    쓰레드에서 실행할 메서드의 형식을 지정한다.
```

12.1.1 쓰레드 생성. 쓰레드를 만들려면 그것이 실행할 인스턴스 메서드나 정적 메서드가 필요하다. 이 메서드는 매개 변수와 반환 값이 없어야 한다.

```
class MultiplicationTable
{
    public void Print()
    {
    }
}
```

쓰레드는 클래스 Thread의 인스턴스를 생성하면서 만들어진다. 생성자는 메서드를 매개 변수로 가져야 한다. 위에서 만든 메서드 Print()를 매개 변수로 하는 쓰레드를 만든다.

```
MultiplicationTable table = new MultiplicationTable();
Thread th = new Thread(table.Print);
```

쓰레드 객체 th가 만들어졌다.

12.1.2 쓰레드 시작.

이 쓰레드는

```
th.Start();
```

와 같이 메서드 Start()를 호출하여 시작된다.

프로젝트 TestStart

```
using System;
using System.Threading;

class MultiplicationTable
{
    private int n;
    public int N
    {
        set { n = value; }
    }
    public void Print()
    {
        for (int i = 1; i <= 9; i++)
            Console.WriteLine("{0} x {1} = {2}", n, i, n * i);
    }
    static void Main()
    {
        MultiplicationTable table = new MultiplicationTable();
        table.N = Int32.Parse(Console.ReadLine());

        Thread th = new Thread(table.Print);
        th.Start();
    }
}
```

```
입력
    8
출력
    8 x 1 = 8
    8 x 2 = 16
        ...
```

메인 쓰레드는

```
th.Start();
```

를 실행하여 쓰레드 th를 시작하고 종료된다. 잠깐 동안 두 쓰레드가 실행된다. 쓰레드 th는 메서드 Print()를 실행한 후 자동으로 종료된다.

겉으로만 보면

```
Thread th = new Thread(table.Print);
th.Start();
```

는

```
table.Print();
```

와 차이가 없다. 그러나 내부적으로 보면 쓰레드가 따로 만들어져 실행된다.

◆ 연습문제 12.1.1. 구구단을 모두 출력하는 메서드 PrintAll()을 작성하고 쓰레드에 대입하여 실행하시오.

12.1.3 쓰레드 종료. 매개 변수인 메서드가 종료되면 해당 쓰레드가 종료된다. 프로젝트 TestStart의 쓰레드는 Print()를 모두 실행한 후 종료된다.

한 쓰레드가 작업을 마칠 때까지 기다려서 다음 작업을 할 필요가 있으면 메서드 Join()을 사용한다. 이 메서드는 해당 쓰레드가 종료될 때까지 현재 쓰레드를 대기하도록 한다. 항상 필요하지는 않지만 때때로 다음 작업을 위하여 호출되기도 한다.

프로젝트 TestStartEnd

```
using System;
using System.Threading;

class Worker
{
    private int n;
    private bool working;
    public void Work()
    {
        n = 0;
        working = true;
        while(working)
            Console.WriteLine("Worker is working: {0}", ++n);
    }

    public void RequestStop()
    {
        working = false;
    }

    static void Main()
    {
        Worker wo = new Worker();
        Thread th = new Thread(wo.Work);
```

```
            Console.WriteLine("Worker 쓰레드 시작");
            th.Start();

            // Worker 쓰레드가 시작될 때까지 기다린다.
            while (!th.IsAlive) ;

            // 메인 쓰레드를 1/1000초간 쉬게 한다.
            Thread.Sleep(1);

            // Worker 쓰레드를 중단하도록 working을 false로 설정한다.
            wo.RequestStop();

            // Worker 쓰레드가 종료될 때까지 기다린다.
            th.Join();

            Console.WriteLine("Worker 쓰레드 종료");
        }
    }
```

```
출력
    Worker 쓰레드 시작
    Worker is working: 1
    Worker is working: 2
    Worker is working: 3
    Worker is working: 4
        (중간 생략)
    Worker 쓰레드 종료
```

Worker 쓰레드는 working의 값이 true인 동안 계속 실행된다. 따라서 쓰레드가 시작되면 while 구역이 반복 실행된다. 메서드 Work()에는 working의 값을 변화시키는 부분이 없지만 쓰레드가 실행되는 동안 RequestStop()이 호출되면 working의 값이 false로 바뀌어 while 구역에서 벗어난다.

```
wo.RequestStop();
```

은 working의 값을 false로 설정하여 쓰레드를 종료하도록 한다.

일반적으로 쓰레드가 생성되기도 전에 프로그램이 종료되거나 쓰레드를 실행한 후에 진행되어야 할 작업이 먼저 진행될 수 있다. 따라서 쓰레드가 생성될 때까지 기다리도록

```
while (!th.IsAlive) ;
```

을 삽입한다.

클래스 Thread의 정적 메서드 Sleep()은 해당 쓰레드의 진행을 멈추도록 한다. 매개 변수는 1/1000초 단위이다.

```
Thread.Sleep(1);
```

은 1/1000초 동안 메인 쓰레드가 진행되지 않도록 한다.

◆ 연습문제 12.1.2. 프로젝트 TestStartEnd의

```
Thread.Sleep(1);
```

를 수정하여 메인 쓰레드가 1초 동안 진행을 멈추도록 하시오.

◆ 연습문제 12.1.3. 메서드 Work()의 출력이 1/10초에 한 번씩만 이루어지도록 수정하시오.

◆ 연습문제 12.1.4. 사용자 입력을 받아 출력하는 쓰레드를 만드시오. 입력한 문자가 'x'이면 쓰레드를 종료하도록 while문을 사용하시오.

12.2 쓰레드 동기화

멀티쓰레딩은 한 프로세스에서 두 개 이상의 쓰레드가 동시에 실행되는 것을 말한다. 쓰레드들이 서로 다른 자료를 사용하면 대부분 문제가 없다. 그러나 자료를 공유하면 상황이 복잡해진다.

은행의 계좌에서 예금을 인출하는 쓰레드와 입금하는 쓰레드가 동시에 실행되는 상황을 생각해 보자. 편의상 계좌의 잔고가 10000원이라 하자. 인출하는 쓰레드가 먼저 계좌 잔고를 읽어서 10000원임을 인식한 뒤 입금하는 쓰레드가 다시 잔고를 확인하였다. 인출하는 쓰레드는 계좌에서 5000원을 인출한 뒤 잔고를 5000원으로 설정하였다. 그 뒤 입금하는 쓰레드가 계좌에 10000원을 입금하였다. 입금하는 쓰레드는 출금하는 쓰레드가 잔고를 5000원으로 변경한 것을 알 수 없으므로 잔고를 20000원으로 설정하였다. 두 쓰레드는 제대로 동작하였지만 결과는 엉망이 되었다.

쓰레드들이 자료를 공유하여 서로에게 영향을 주거나 쓰레드들의 실행 순서를 조정할 필요가 있을 때 이들 간에 질서를 바로 잡아주는 것을 **쓰레드 동기화**(synchronization)라 한다. 쓰레드들이 서로에게 영향을 주지 않고 독립적으로 실행될 때 **비동기적**(asynchronous)으로 실행된다고 한다.

프로젝트 TestStartEnd에서 메인 쓰레드와 쓰레드 th는 비동기적으로 실행된다. 다음 연습문제는 비동기적으로 실행되는 두 쓰레드를 만드는 것이다.

◆ 연습문제 12.2.1. 두 쓰레드를 만들어 구구단의 서로 다른 두 단을 동시에 출력하시오.

12.2.1 동기화 이벤트 객체. 쓰레드 동기화 방법 중 자료를 공유하지 않고 단순히 실행 순서를 조절하는 경우에 대하여 알아보자.

쓰레드의 실행 순서를 조절하기 위하여 **동기화 이벤트 객체**를 사용한다. 이 객체는 신호를 받은 상태와 신호를 받지 않은 상태 둘 중 하나의 상태로 존재한다. 신호를 받은 상태에서 메서드 WaitOne()을 만나면 해당 쓰레드를 계속 진행한다. 신호를 받지 않은 상태에서 WaitOne()을 만나면 신호를 받을 때까지 해당 쓰레드를 진행하지 않고 대기한다.

클래스 EventWaitHandle은 동기화 이벤트 객체의 생성 및 운용에 필요한 메서드를 제공한다.

class System.Threading.EventWaitHandle

public EventWaitHandle(bool initialState, EventResetMode mode)
동기화 이벤트 객체를 생성한다. initialState가 true이면 초기 상태를 신호 받음으로 설정하고 false이면 신호 받지 않음으로 설정한다. mode에 따라 상태가 자동으로 전환되거나 수동으로 전환된다.

public bool Set()
상태를 신호 받음으로 설정한다. 상태 설정을 성공적으로 마쳤으면 true, 실패했으면 fase를 반환한다.

public bool Reset()
상태를 신호 받지 않음으로 설정한다. 상태 설정을 성공적으로 마쳤으면 true, 실패했으면 fase를 반환한다.

public virtual bool WaitOne()
신호를 받을 때까지 해당 쓰레드의 실행을 일시 중단한다. 신호를 받으면 true를 반환하고 쓰레드를 계속 실행한다. 해당 동기화 이벤트 객체의 상태는 생성자에서 설정한 mode에 따라 신호 받음으로 유지되거나 신호 받지 않음으로 전환된다.

메서드 Waitone()을 실행하고 난 후 동기화 이벤트 객체의 상태는 열거형 형식 EventResetMode의 요소로 결정된다.

enum System.Threading.EventResetMode

AutoReset
메서드 Waitone()을 실행하고 난 후 동기화 이벤트 객체의 상태를 신호 받지 않음으로 자동 전환한다.

ManualReset
메서드 Waitone()을 실행하고 난 후 동기화 이벤트 객체의 상태를 전환하지 않는다.

클래스 EventWaitHandle의 파생 클래스 AutoResetEvent는 mode를 AutoReset으로 설정한 것이다.

sealed class System.Threading.AutoResetEvent

public AutoResetEvent(bool initialState)
동기화 이벤트 객체를 생성한다. initialState가 true이면 초기 상태를 신호 받음으로 설정하고 false이면 신호 받지 않음으로 설정한다.

AutoResetEvent 객체가 신호를 받은 상태에서 WaitOne()을 만나면 쓰레드는 계속 실행되고 상태는 신호 받지 않음으로 자동 전환(auto reset)된다. 신호를 받지 않은 상태에서 WaitOne()을 만나면 신호를 받을 때까지 해당 쓰레드는 실행이 일시 중단된다. 신호를 받으면 쓰레드는 다시 실행되고 상태는 신호 받지 않음으로 자동 전환된다.

프로젝트 TestAutoResetEvent

```
using System;
using System.Threading;

class Worker
{
```

```
        private EventWaitHandle waitHandle;
        public EventWaitHandle WaitHandle
        {
            get { return waitHandle; }
        }

        public Worker()
        {
            // 신호를 받지 않은 상태
            waitHandle = new AutoResetEvent(false);
        }

        public void Work()
        {
            Console.WriteLine("Wait...");

            // 신호를 받은 상태라면 무시하고 진행한다.
            // 신호를 받지 않은 상태라면 신호를 기다린다.
            waitHandle.WaitOne();

            // waitHandle은 AutoResetEvent이므로 신호를 받지 않은 상태로
            // 자동 전환 된다.

            Console.WriteLine("Worker is working");
        }
    }

    class Program
    {
        static void Main()
        {
            Worker wo = new Worker();
            Thread th = new Thread(wo.Work);
            th.Start();
            Thread.Sleep(1000);

            // 신호를 보낸다.
            wo.WaitHandle.Set();
        }
    }
```

```
출력
    Wait...
    Worker is working
```

쓰레드 th가

```
th.Start();
```

로 시작되면 메인 쓰레드는

```
Thread.Sleep(1000);
```

으로 1초간 실행이 중단된다. 동시에 쓰레드 th는

```
Console.WriteLine("Wait...");
```

를 출력한 후

```
waitHandle.WaitOne();
```

신호를 받기 위하여 대기한다. 중단된 지 1초가 지나면 메인 쓰레드는

```
wo.WaitHandle.Set();
```

신호를 보내고 종료된다. 신호를 받은 waitHandle은 쓰레드 th의 실행을 재개한다. th는

```
Console.WriteLine("Worker is working");
```

을 출력하고 종료된다.

◆ 연습문제 12.2.2. 위 프로젝트에서 AutoResetEvent를 EventWaitHandle로 바꾸고 같은 결과가 나오도록 수정하시오.

◆ 연습문제 12.2.3. 프로젝트 TestAutoResetEvent에서

```
waitHandle = new AutoResetEvent(false);
```

를

```
waitHandle = new AutoResetEvent(true);
```

로 바꾸어 실행하고 결과를 비교하시오.

EventWaitHandle의 파생 클래스 ManualResetEvent는 mode를 ManualReset으로 설정한 것이다.

sealed class ManualResetEvent
public ManualResetEvent(bool initialState)
동기화 이벤트 객체를 생성한다. initialState가 true이면 상태를 신호 받음으로 설정하고 false이면 신호 받지 않음으로 설정한다.

ManuelResetEvent 객체가 신호를 받은 상태에서 WaitOne()을 만나면 쓰레드는 계속 실행되고 신호를 받은 상태로 유지된다. 신호를 받지 않은 상태에서 WaitOne()을 만나면 신호를 받을 때까지 해당 쓰레드는 실행이 일시 중단된다. 신호를 받으면 쓰레드는 다시 실행되고 상태는 신호 받은 상태로 유지된다. 메서드 Reset()을 써서 신호를 받지 않은 상태로 전환할 수 있다. 위에서 설명한 AutoResetEvent와 다른 점이 이것이다.

◆ 연습문제 12.2.4. 프로젝트 TestAutoResetEvent에서 AutoResetEvent를 사용하고 있다. 이것을 ManualResetEvent로 바꾸면 어떻게 될까? 예상해 보고 실행하여 확인하자.

다음 프로젝트의 두 쓰레드는 신호를 서로 주고받아 번갈아 가면서 실행된다.

프로젝트 TestManualResetEvent

```
using System;
using System.Threading;
```

```
    class Worker
    {
        private string name;
        private EventWaitHandle waitHandle;

        private Worker theOtherMan;
        public Worker TheOtherMan
        {
            set { theOtherMan = value; }
        }

        public Worker(string name, bool initialState)
        {
            this.name = name;
            waitHandle = new ManualResetEvent(initialState);
        }

        public void Work()
        {
            for (int i = 0; i < 3; i++)
            {
                // 신호를 기다린다.
                waitHandle.WaitOne();

                Console.WriteLine("{0} is working: {1}", name, i);

                // 신호 받지 않음으로 전환
                waitHandle.Reset();

                // 다른 Worker에게 신호를 보낸다.
                theOtherMan.SetSignal();
            }
        }

        public void SetSignal()
        {
            waitHandle.Set();
        }
    }

    class Program
    {
        static void Main()
        {
            Worker wo1 = new Worker("Worker 1", false);
            Worker wo2 = new Worker("Worker 2", false);
            wo1.TheOtherMan = wo2;
            wo2.TheOtherMan = wo1;

            Thread th1 = new Thread(wo1.Work);
            Thread th2 = new Thread(wo2.Work);
            th1.Start();
            th2.Start();
            Thread.Sleep(200);

            // Worker 2에 신호를 보낸다.
            wo2.SetSignal();
        }
```

```
    }
```

```
출력
    Worker 2 is working: 0
    Worker 1 is working: 0
    Worker 2 is working: 1
    Worker 1 is working: 1
    Worker 2 is working: 2
    Worker 1 is working: 2
```

위 예제에서

```
Worker wo1 = new Worker("Worker 1", false);
```

는 동기화 이벤트 객체 waitHandle을 신호를 받지 않은 상태로 설정한다. wo2도 마찬가지이다.

```
th1.Start();
th2.Start();
```

는 두 쓰레드 th1과 th2가 시작되도록 하지만

```
waitHandle.WaitOne();
```

때문에 둘 다 대기 상태가 된다. 여기서 0.2초간 멈춘다.

```
Thread.Sleep(200);
```

0.2초 후

```
wo2.SetSignal();
```

에 의하여 wo2의 동기화 이벤트 객체가 신호를 받은 상태로 바뀐다. 따라서 보류되었던 코드

```
Console.WriteLine("{0} is working: {1}", name, i);
```

가 실행된다.

```
waitHandle.Reset();
```

에 의하여 wo2의 waitHandle은 신호를 받지 않은 상태로 전환된다. 그렇다고 바로 해당 쓰레드의 실행이 보류되는 것은 아니다. 평소와 마찬가지로 for 구역을 계속 실행하다가 WaitOne()을 만나야 비로소 실행이 보류된다.

한편

```
theOtherMan.SetSignal();
```

은 wo1의 동기화 이벤트 객체에 신호를 보낸다. 실행이 보류되었던 wo1의 메서드 Work()를 이어서 실행한다. 출력하고 동기화 이벤트 객체를 신호를 받지 않은 상태로 전환하고. wo2에 신호를 보낸다. 이와 같은 작업을 세 번 반복한다.

◆ 연습문제 12.2.5. 프로젝트 TestManualResetEvent에서

```
waitHandle.Reset();
```

를 제거하고 실행하시오. 결과에 대하여 설명하시오.

◇ 연습문제 12.2.6. 프로젝트 TestManualResetEvent에서

```
Worker wo2 = new Worker("Worker 2", false);
```

를

```
Worker wo2 = new Worker("Worker 2", true);
```

로 바꾸고

```
wo2.SetSignal();
```

을 삭제한 후 실행하여 결과를 비교하시오.

◆ 연습문제 12.2.7. ManualResetEvent를 AutoResetEvent로 바꾸어 같은 결과가 나오도록 프로젝트 TestManualResetEvent를 수정하시오.

◆ 연습문제 12.2.8. 동전을 던져 앞면이 나오면 Worker 1이 일을 하고 뒷면이 나오면 Worker 2가 일을 하도록 프로젝트 TestManualResetEvent를 수정하시오.

12.2.2 객체 잠금. 쓰레드들이 공유하는 자료가 있는 경우에 대하여 알아보자. 쓰레드는 일정 구역을 실행하는 동안 한 객체에 대한 배타적 권한을 설정할 수 있다. 이것을 **객체 잠금**(object locking) 또는 **잠금**이라 한다. 쓰레드가 객체에 대한 **모니터**(monitor)를 소유한다고도 한다. 객체를 잠그는 구역을 **동기화된 구역**(synchronized block)이라 한다.

직접적인 어떤 권한이 있는 것이 아니라 추상적으로 권한을 갖는다고 말한다. 한 쓰레드가 객체를 잠그면 다른 쓰레드는 그 객체를 잠글 수 없다. 객체를 잠그더라도 다른 쓰레드가 객체를 사용할 수 없는 것은 아니다. 다만 두 쓰레드가 동시에 같은 객체를 잠글 수는 없다.

두 쓰레드가 동시에 같은 객체를 잠글 수 없다는 것이 중요하다. 다른 쓰레드의 영향을 받지 않고 작업을 수행할 수 있기 때문이다.

키워드 lock은 구역을 설정하여 그 구역을 실행하는 동안 객체를 잠근다.

```
lock (obj)
{
    // 동기화된 구역
}
```

다른 쓰레드가 객체 obj를 잠그려면 이 객체의 잠금이 풀릴 때까지 기다려야 한다. 곧 동기화된 구역이 모두 실행될 때까지 기다려야 한다.

프로젝트 TestLock1

```
using System;
using System.Threading;
```

```
class Program
{
    static object obj = new object();
    static void foo()
    {
        lock (obj)
        {
            Console.Write("obj 잠금 ... ");
            Thread.Sleep(10);
            Console.WriteLine("잠금 풀림");
        }
    }

    static void Main()
    {
        new Thread(foo).Start();
        new Thread(foo).Start();
    }
}
```

```
출력
obj 잠금 ... 잠금 풀림
obj 잠금 ... 잠금 풀림
```

두 쓰레드 중

```
lock (obj)
```

에 먼저 도달하는 쪽이 객체 obj를 잠근다. 나중에 도달한 쓰레드는 obj를 잠그기 위하여 잠금이 풀릴 때까지 기다린다. 결국 먼저 도달한 쓰레드가 동기화된 구역을 벗어나야 비로소 늦게 도달한 쓰레드의 실행이 재개된다.

앞에서도 설명하였지만 뒤에 도달한 쓰레드는 객체를 사용하기 위하여 기다리는 것이 아니고 잠그기 위하여 기다린다. 따라서 잠금을 필요로 하지 않는 작업은 바로 수행할 수 있다.

◆ 연습문제 12.2.9. 프로젝트 TestLock1에서

```
lock (obj)
```

를 제거하고 실행하여 결과를 비교하시오.

◆ 연습문제 12.2.10. 클래스 TestLock1에서 코드

```
new Thread(foo).Start();
```

를 하나만 제거하고 그 자리에 메인 쓰레드가 객체 obj를 잠그는 코드를 추가하시오.

조금 복잡한 예를 하나 더 소개한다.

프로젝트 TestLock2

```
using System;
```

```
using System.Threading;

class Account
{
    public int Valance { get; set; }
}

class Manager
{
    private Account account;
    private int amount;
    public Manager(Account account, int amount)
    {
        this.account = account;
        this.amount = amount;
    }
    public void Deposit()
    {
        lock (account)
        {
            Console.WriteLine("Deposit: account 잠금");
            int valance = account.Valance;
            Thread.Sleep(100);
            account.Valance = valance + amount;
            Console.WriteLine(account.Valance);
            Console.WriteLine("Deposit: account 잠금 풀림");
        }
    }
    public void Withdraw()
    {
        lock (account)
        {
            Console.WriteLine("Withdraw: account 잠금");
            int valance = account.Valance;
            Thread.Sleep(100);
            account.Valance = valance - amount;
            Console.WriteLine(account.Valance);
            Console.WriteLine("Withdraw: account 잠금 풀림");
        }
    }
}

class Program
{
    static void Main()
    {
        Account account = new Account();
        account.Valance = 10000;
        Manager depositManager = new Manager(account, 2000);
        Manager withdrawManager = new Manager(account, 1000);

        new Thread(depositManager.Deposit).Start();
        new Thread(withdrawManager.Withdraw).Start();
    }
}
```

출력

```
Withdraw: account 잠금
9000
```

```
Withdraw: account 잠금 풀림
Deposit: account 잠금
11000
Deposit: account 잠금 풀림
```

◆ 연습문제 12.2.11. 프로젝트 TestLock2에서

```
lock (account)
```

를 하나만 제거하고 실행하여 결과를 비교하시오. 한 쓰레드가 객체를 잠가도 다른 쓰레드가 그 객체를 사용할 수 있음을 설명하시오.

◆ 연습문제 12.2.12. 프로젝트 TestLock2의 출력은 다음과 같을 수도 있다.

```
Deposit: account 잠금
12000
Deposit: account 잠금 풀림
Withdraw: account 잠금
11000
Withdraw: account 잠금 풀림
```

그 이유를 설명하시오.

lock으로 잠그는 객체는 참조 형식이어야 한다. 따라서 값 형식인 구조체는 잠글 수 없다.

◇ 객체 잠금은 System.Threading.Moniter.Enter()로도 설정할 수 있다.

◇ 잠그는 객체가 반드시 어떤 자료를 포함하고 있어야 하는 것은 아니다. 단순히 object 객체를 잠글 수도 있다. 그러나 의미가 없는 객체를 잠그는 일은 별로 없을 것이다.

12.3 쓰레드 풀

여러 쓰레드를 한꺼번에 관리할 수 있도록 모아 놓은 것을 **쓰레드 풀**(thread pool)이라 한다. 다양한 작업들을 모아 쓰레드 풀을 형성하고 백그라운드 작업을 할 수 있다.

클래스 ThreadPool은 쓰레드 풀을 지원한다.

static class System.Threading.ThreadPool

public static bool QueueUserWorkItem(WaitCallback callBack)
실행을 위해 메서드 callback을 큐에 등록시킨다. 성공적으로 큐에 등록시키면 true를 반환한다.

public static bool QueueUserWorkItem(WaitCallback callBack, object state)
실행을 위해 메서드 callback을 큐에 등록시키고 메서드에서 사용할 객체를 지정한다. 성공적으로 큐에 등록시키면 true를 반환한다.

대리자 WaitCallback은 쓰레드 풀에서 사용할 메서드의 형식을 지정한다. 매개 변수가 object 객체 하나이고 값은 반환하지 않는다.

delegate void WaitCallback(object state)
쓰레드 풀에서 사용할 메서드의 형식을 지정한다.

12.3.1 쓰레드 등록. 쓰레드를 만들 때와 마찬가지로 쓰레드 풀에서 실행할 인스턴스 메서드나 정적 메서드가 필요하다.

```
public static void Add(object obj)
```

이 메서드는

```
ThreadPool.QueueUserWorkItem(Add, 10);
```

과 같이 매개 변수를 적어서 큐에 등록시킨다. 10은 boxing되어 Add()의 매개 변수 obj에 할당된다. 큐에 등록시킨 쓰레드는 CPU의 계획에 따라 자동으로 실행된다. 매개 변수를 사용하지 않으려면

```
ThreadPool.QueueUserWorkItem(Add);
```

와 같이 둘째 매개 변수를 적지 않는다.

프로젝트 TestThreadPool1

```
using System;
using System.Threading;

class Program
{
    public static void Add(object obj)
    {
        int n = (int)obj;
        for (int i = 1; i < 3; i++)
            Console.WriteLine("Add {0} -> {1}", i, (n + i));
    }

    public static void Multiply(object obj)
    {
        int n = (int)obj;
        for (int i = 1; i < 3; i++)
            Console.WriteLine("Multiply {0} -> {1}", i, n * i);
    }

    public static void AddByOne(object obj)
    {
        for (int i = 1; i < 3; i++)
            Console.WriteLine("AddByOne {0} -> {1}", i, i + 1);
    }

    static void Main()
    {
        ThreadPool.QueueUserWorkItem(Add, 10);
        ThreadPool.QueueUserWorkItem(Multiply, 5);
        ThreadPool.QueueUserWorkItem(AddByOne);
        Thread.Sleep(1000);
    }
}
```

출력

```
    AddByOne 1 -> 2
```

```
AddByOne 2 -> 3
Multiply 1 -> 5
Add 1 -> 11
Add 2 -> 12
Multiply 2 -> 10
```

메서드 AddByOne()은 매개 변수 obj를 사용하지 않는다. 그러나 대리자 WaitCallback() 형식에 맞추기 위하여 필요 없는 매개 변수를 적어주었다.

위 프로젝트에서는 큐에 등록된 쓰레드가 모두 실행될 때까지 기다리기 위하여

```
Thread.Sleep(1000);
```

를 추가하였다.

◆ 연습문제 12.3.1. 위 프로젝트에서

```
Thread.Sleep(1000);
```

을 제거하시오. 프로그램을 실행하고 결과에 대하여 설명하시오.

◆ 연습문제 12.3.2. 위 프로젝트에 새로운 매소드를 추가하고 쓰레드 풀에 등록하시오.

12.3.2 쓰레드 상태. 쓰레드에서 동기화 이벤트 객체의 상태를 변화시켜 쓰레드가 종료되었음을 알릴 수 있다. 쓰레드 풀에 등록된 모든 쓰레드에 이 방법을 적용하여 작업이 종료되었는지 알 수 있다.

클래스 WaitHandle은 EventWaitHandle의 기본 클래스이다.

abstract class System.Threading.WaitHandle
public static bool WaitAll(WaitHandle[] waitHandles)
배열의 모든 요소가 신호를 받을 때까지 기다린다. 모든 요소가 신호를 받으면 true를 반환한다.

쓰레드 풀에 등록된 쓰레드마다 EventWaitHandle 객체를 하나씩 할당하고 이들을 배열을 만든다. 메서드 WaitAll()은 모든 객체가 신호를 받을 때까지 해당 쓰레드의 실행을 보류하고 기다린다.

프로젝트 TestThreadPool2

```
using System;
using System.Threading;

class Fibonacci
{
    private int n;
    public int N
    {
        get { return n; }
    }
```

```
        private int result;
        public int Result
        {
            get { return result; }
        }

        private EventWaitHandle doneEvent;
        public Fibonacci(int n, EventWaitHandle doneEvent)
        {
            this.n = n;
            this.doneEvent = doneEvent;
        }
        public void Calculate(Object context)
        {
            int index = (int)context;
            Console.WriteLine("쓰레드 {0} 계산 시작", index);
            result = ValueAt(n);
            Console.WriteLine("쓰레드 {0} 계산 종료", index);
            doneEvent.Set();
        }
        public static int ValueAt(int n)
        {
            if (n <= 1)
                return n;
            return ValueAt(n - 1) + ValueAt(n - 2);
        }
    }

    class Program
    {
        static void Main()
        {
            Fibonacci[] fib = new Fibonacci[3];
            EventWaitHandle[] doneEvents = new EventWaitHandle[3];
            Random rand = new Random();
            for (int i = 0; i < 3; i++)
            {
                doneEvents[i] = new AutoResetEvent(false);
                fib[i] = new Fibonacci(rand.Next(20,40), doneEvents[i]);
                ThreadPool.QueueUserWorkItem(fib[i].Calculate, i);
            }

            WaitHandle.WaitAll(doneEvents);
            Console.WriteLine("모든 계산 완료");

            for (int i = 0; i < 3; i++)
                Console.WriteLine("F({0}) = {1}", fib[i].N, fib[i].Result);
        }
    }
```

출력

```
쓰레드 0 계산 시작
쓰레드 1 계산 시작
쓰레드 2 계산 시작
쓰레드 2 계산 종료
쓰레드 0 계산 종료
쓰레드 1 계산 종료
모든 계산 완료
F(23) = 28657
```

```
F(28) = 317811
F(20) = 6765
```

클래스 Random의 메서드

```
rand.Next(20,40)
```

는 20보다 크거나 같고 40보다 작은 정수를 난수로 발생시킨다.

◆ 연습문제 12.3.3. 프로젝트 TestThreadPool2에서

```
doneEvents[i] = new AutoResetEvent(false);
```

를

```
doneEvents[i] = new AutoResetEvent(true);
```

로 바꾸면 어떻게 될까? 결과를 예측해 보고 실행하여 확인하시오.

◆ 연습문제 12.3.4. 매개 변수가 한 개인 메서드 QueueUserWorkItem()을 쓰도록 프로젝트 TestThreadPool2를 수정하시오.

◆ 연습문제 12.3.5. 메서드 WaitOne()을 여러 번 써서 WaitAll()과 비슷한 역할을 하도록 프로젝트 TestThreadPool2를 수정하시오. 두 방법의 차이점을 말하시오.

12.4 교착 상태

둘 이상의 쓰레드가 다른 쓰레드의 작업이 끝나기를 한없이 기다려야 하는 것을 **교착 상태**(deadlock)라 한다. 교착 상태가 발생하면 프로그램은 대기 상태에서 아무 일도 하지 못한다. 멀티쓰레딩을 구현할 때는 이런 상황이 발생하지 않도록 주의해야 한다.

다음은 교착 상태의 한 예이다. 한 쓰레드가 obj1을 잠그고 obj2를 잠그려고 한다.

```
lock (obj1)
{
    lock (obj2)
    {
    }
}
```

동시에 다른 쓰레드는 obj2를 잠그고 obj1을 잠그려고 기다린다.

```
lock (obj2)
{
    lock (obj1)
    {
    }
}
```

두 쓰레드는 한없이 기다리기만 할 뿐 할 수 있는 일이 없다.

다음은 교착 상태를 프로그램으로 구현한 것이다.

프로젝트 TestDeadLock

```
using System;
using System.Threading;

class Worker
{
    private Object obj0 = new Object();
    private Object obj1 = new Object();

    public void Zero2One()
    {
        lock (obj0)
        {
            Thread.Sleep(100);
            Console.WriteLine("Zero2One started");
            lock (obj1)
            {
                obj1 = obj0;
            }
        }
    }
    public void One2Zero()
    {
        lock (obj1)
        {
            Thread.Sleep(100);
            Console.WriteLine("One2Zero started");
            lock (obj0)
            {
                obj0 = obj1;
            }
        }
    }
}

class Program
{
    static void Main()
    {
        Worker wo = new Worker();

        new Thread(wo.Zero2One).Start();
        new Thread(wo.One2Zero).Start();
    }
}
```

◆ 연습문제 12.4.1. 위 상황을 해결할 수 있는 방법을 제시하시오.

제 13 장 추가로 살펴볼 것들

C#에는 초보자가 익히기 어려운 내용들이 많이 포함되어 있다. 다른 언어를 배워서 다양한 프로그램을 작성해 본 프로그래머는 그 내용들이 필요하다는 것을 경험적으로 알고 있지만 그렇지 않은 이들은 혼란스러울 수밖에 없다.

이 장은 초보자가 배우기 어려우나 사용하면 편리한 내용들로 이루어져 있다. 처음부터 순서대로 학습할 필요는 없으며 원하는 내용을 찾아서 보면 좋을 것이다. 뒤에 나오는 내용을 앞에서 인용하기도 하여 순서가 맞지도 않는다.

13.1 코드 분할

대규모 프로젝트를 진행하는 경우 서로 연관이 있는 요소들을 여러 파일로 나누어 개별적으로 작업을 하면 효율적일 것이다. 그러나 이들을 하나로 합쳐 파일을 만드는 것은 번거로울 뿐 아니라 오류가 발생할 확률을 높인다. 따라서 개별 파일들을 독립적으로 유지한 채 그룹을 만드는 방법이 필요하다.

13.1.1 네임스페이스. 서로 연관된 요소들을 묶어 이름을 붙인 것을 **네임스페이스**(namespace)라 한다. 키워드 namespace로 정의하며 클래스, 인터페이스, 구조체, 열거형 형식, 대리자, 네임스페이스를 포함할 수 있다. 특정한 용도로 사용되는 것들을 묶어 그룹으로 관리할 때 유용하다.

```
namespace SampleNamespace
{
    class SampleClass { }
    interface SampleInterface { }
    struct SampleStruct { }
    enum SampleEnum { a, b }
    delegate void SampleDelegate(int i);
    namespace Nested
    {
        class SampleClass2 { }
    }
}
```

네임스페이스에 포함된 요소를 사용하려면

```
SampleNamespace.SampleClass sc;
SampleNamespace.Nested.SampleClass2 sc2;
```

와 같이 네임스페이스의 이름에 점을 찍고 요소 이름을 적는다.

다음 프로젝트는 네임스페이스를 사용한다.

프로젝트 TestNamespace1

```
using System;

namespace SampleNamespace
{
    class SampleClass
```

```
    {
        public void print()
        {
            Console.WriteLine("SampleNamespace의 SampleClass");
        }
    }
}

class Program
{
    static void Main()
    {
        SampleNamespace.SampleClass sc = new SampleNamespace.SampleClass();
        sc.print();
    }
}
```

```
출력
    SampleNamespace의 SampleClass
```

네임스페이스의 이름을 계속 써야 한다면 작업이 번거롭고 코드가 지저분해진다. 이 때 키워드 using으로 시작하는 문장을 써서 특정한 네임스페이스를 사용함을 명시할 수 있다.

```
SampleNamespace.SampleClass sc;
```

는

```
SampleClass sc;
```

와 같이 간단히 쓸 수 있다. 대신 파일의 앞부분에

```
using SampleNamespace;
```

와 같이 네임스페이스를 사용함을 명시해야 한다.

네임스페이스는 네임스페이스를 포함할 수 있다. 이것을 **네임스페이스의 중첩**이라 한다. 중첩된 네임스페이스는 다음 세 형식 중 하나를 써서 정의할 수 있다.

```
namespace Figure
{
    namespace Image
    {
    }
}
```

```
namespace Figure
{
    namespace Figure.Image
    {
    }
}
```

```
namespace Figure
{
}
```

```
namespace Figure.Image
{
}
```

중첩된 네임스페이스를 사용하려면

```
using Figure.Image;
```

와 같이 using으로 시작하는 문장에 네임스페이스의 이름을 연속하여 적어주면 된다.

다음과 같은 프로젝트 TestNameSpace2를 만들어 보자.

프로젝트 TestNamespace2

```
using Figure;
using Figure.Image;
using System;

namespace Figure
{
    class Circle
    {
        public static void print()
        {
            Console.WriteLine("Figure.Circle");
        }
    }

    namespace Image
    {
        class Bitmap
        {
            public static void print()
            {
                Console.WriteLine("Figure.Image.Bitmap");
            }
        }
    }

}

namespace Figure.Image
{
    class JPEG
    {
        public static void print()
        {
            Console.WriteLine("Figure.Image.JPEG");
        }
    }
}

class Program
{
    static void Main()
    {
        Circle.print();
        Bitmap.print();
```

```
            JPEG.print();
        }
    }
```

```
출력
    Figure.Circle
    Figure.Image.Bitmap
    Figure.Image.JPEG
```

서로 다른 네임스페이스에 포함된 같은 이름을 가진 항목은 서로 다른 것으로 취급한다.

```
namespace A
{
    class Foo { }
}
namespace B
{
    class Foo { }
}
```

와 같이 정의된 두 클래스 Foo는 각각 A.Foo와 B.Foo로 구별하여 사용한다.

13.1.2 부분 형식. 클래스나 구조체, 인터페이스 또는 메서드를 여러 조각으로 나누어 정의하고 키워드 partial을 붙이면 하나로 인식한다. 각 조각은 **부분 클래스**(partial class), **부분 구조체**(partial struct), **부분 인터페이스**(partial interface), **부분 메서드**(partial method)라 한다. 이들은 한 파일에 있어도 되고 여러 파일에 나뉘어져 있어도 된다.

클래스를 두 조각으로 나누려면

```
partial class Foo
{
    public Foo() { }
}

partial class Foo
{
    public void Print() { }
}
```

와 같이 정의한다. 두 클래스 Foo는 하나로 정의된 것과 같다.

클래스를 여러 조각으로 분할하려면 모든 조각에 partial을 붙여서 정의한다. 한 조각이라도 추상 클래스로 선언되면 전체가 추상 클래스가 된다. 한 조각이라도 클래스나 인터페이스를 상속하면 전체가 상속하게 된다. 또 한 조각이라도 키워드 sealed를 붙여 봉인하면 전체를 봉인한 것과 같다.

다음 예제에서는 클래스를 두 조각으로 나누어 정의한다.

프로젝트 TestPartialClass

```
using System;

sealed partial class Point
{
    private double x;
    public double X
    {
        get { return x; }
    }
    private double y;
    public double Y
    {
        get { return y; }
    }
    public Point(double x, double y)
    {
        this.x = x;
        this.y = y;
    }
    public override string ToString()
    {
        return string.Format("({0}, {1})", x, y);
    }
}

partial class Point
{
    public void Move(double a, double b)
    {
        x += a;
        y += b;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Point p = new Point(1, -1);
        p.Move(3, 4);
        Console.WriteLine(p);
    }
}
```

출력

```
(4,3)
```

키워드 partial은 구조체나 인터페이스에서도 클래스에서와 유사하게 사용된다.

◆ 연습문제 13.1.1. 두 메서드를 갖는 인터페이스

```
interface IAccessible
{
    void Print();
    void Read();
}
```

를 두 조각으로 나누시오.

메서드의 본문은 두 조각으로 나눌 수 없다. 따라서 키워드 partial이 적용되는 방식도 클래스와 다르다.

메서드에 partial을 적용하여 부분 메서드를 만들려면 반환하는 값이 없어야 한다. 접근 한정자 적어주지 않지만 private으로 간주한다. 부분 메서드는 선언 부분과 구현 부분으로 나뉜다.

선언 부분

```
partial void Print(int x);
```

에는 본문이 없다. 본문은 구현 부분

```
partial void Print(int x)
{
    Console.WriteLine(x);
}
```

에 작성한다.

프로젝트 TestPartialMethod

```
using System;

partial class Foo
{
    private int n;
    public int N
    {
        get { return n; }
    }
    public Foo(int n)
    {
        this.n = n;
    }
    partial void Add(int x);
}

partial class Foo
{
    partial void Add(int x)
    {
        n += x;
    }
    public void Add(string str)
    {
        Add(Int32.Parse(str));
    }
}

class Program
{
    static void Main(string[] args)
    {
        Foo foo = new Foo(99);
        foo.Add("123");
```

```
            Console.WriteLine(foo.N);
        }
    }
```

부분 메서드의 매개 변수에 ref를 붙일 수는 있지만 out은 붙일 수 없다.

부분 메서드의 구현 부분이 없으면 메서드는 무시된다. 메서드를 호출하여도 에러가 발생하지 않으며 호출하는 코드도 무시된다.

◆ 연습문제 13.1.2. 위 프로젝트에서 부분 메서드 Add()의 구현 부분을 제거하고 실행하시오.

13.2 어트리뷰트

클래스, 구조체 또는 그것들의 멤버에 추가적인 정보를 설정하기 위하여 **어트리뷰트**(attribute)를 사용한다. 대괄호 안에 이름과 매개 변수를 적어서 설정한다.

[어트리뷰트_이름(위치_매개_변수, 명명된_매개_변수=값, ...)]

첫째 매개 변수인 위치_매개_변수는 필수적인 요소이며 문자열이어야 한다. 명명된_매개_변수는 해당 어트리뷰트에 정의된 것이며 없을 수도 있고 여럿일 수도 있다.

13.2.1 예약된 어트리뷰트. C#이 기본으로 제공하는 어트리뷰트를 **예약된 어트리뷰트**(reserved attribute)라 한다. AttributeUsage, Conditional, Obsolete 등이 있다.

어트리뷰트 Conditional은 System.Diagnostics에 정의되어 있다. 다음 정의는 13.2.2를 학습하고 나면 이해할 수 있다.

```
[AttributeUsage(AttributeTargets.Method, AllowMultiple = true)]
sealed class System.Diagnostics.ConditionalAttribute: Attribute
    public ConditionalAttribute(string conditionString)
        ConditionalAttribute 인스턴스를 생성한다.
    public string ConditionString { get; }
        컴파일할 조건을 정해주는 문자열을 가져온다.
```

Conditional은 컴파일러에게 해당 메서드를 컴파일할 것인지 말 것인지 알려준다. 다음 예제를 보자.

프로젝트 TestConditionalAttribute

```
#define SESSION

using System;
using System.Diagnostics;

class Program
{
    [Conditional("SESSION")]
    static void print()
    {
```

```
            Console.WriteLine("SESSION is defined");
        }

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

출력

```
    SESSION is defined
```

#define은 단어가 정의되었음을 컴파일러에게 알린다. 어트리뷰트 Conditional은 SESSION이 정의되어 있는 지 확인하여 메서드 print()를 컴파일할 것인지 말 것인지 결정하도록 한다. 여기서는

```
#define SESSION
```

으로 SESSION을 정의하였으므로 print()가 컴파일되며 결과가 출력된다. 이 문장을 제거하면 출력이 없다.

◆ 연습문제 13.2.1. 위 프로젝트에서

```
#define SESSION
```

을

```
//#define SESSION
```

으로 수정하고 실행하여 결과를 확인하시오.

◆ 연습문제 13.2.2. 위 프로젝트에

```
#define NEW_SESSION
```

을 삽입하시오.

```
[Conditional("SESSION")]
```

을

```
[Conditional("SESSION"), Conditional("NEW_SESSION")]
```

으로 수정하고 프로그램을 실행하여 결과를 확인하시오.

어트리뷰트 Obsolete는 해당 항목을 더 이상 사용하지 않음을 나타낸다. 이전 버전과 호환성을 위하여 남겨두지만 안전하지 않으므로 사용하지 말 것을 권장하는 것이다. 이 어트리뷰트는 클래스, 구조체 및 이것들의 멤버에 적용할 수 있다.

```
[Obsolete("B로 대체되었습니다")]
class A { }
class B { }
class Program
{
    static void Main()
    {
```

```
            A a = new A();       // Warning
        }
    }
```

◆ 연습문제 13.2.3. 위 코드를 포함하는 프로젝트를 작성하시오. 컴파일하여 경고를 확인하시오.

13.2.2 어트리뷰트 형식. 어트리뷰트의 이름에는 맨 뒤에 Attribute가 붙으며 생략할 수도 있다. 따라서 Conditional과 Obsolete는 각각 ConditionalAttribute와 ObsoleteAttribute로 써도 된다.

실제로 어트리뷰트는 클래스이며 클래스 System.Attribute에서 파생된다. 앞에서 살펴본 Conditional이나 Obsolete와 같이 예약된 어트리뷰트도 모두 마찬가지이다. Conditional의 정의

```
public class ConditionalAttribute: Attribute
```

에서 확인할 수 있다.

사용자가 정의한 어트리뷰트를 **사용자 지정 어트리뷰트**(custom attribute)라 한다. 어트리뷰트를 정의하려면 AttributeUsage를 적어준다. AttributeUsage도 어트리뷰트이다.

[AttributeUsage(적용할_항목, 옵션)]

적용할_항목에는 어트리뷰트를 적용할 수 있는 항목을 적어준다. 여러 항목에 적용 가능하도록 설정하려면 비트 논리합(|)을 사용한다. 어트리뷰트를 메서드에 적용하려면 AttributeTargets.Method를 적어 준다.

```
[AttributeUsage(AttributeTargets.Method)]
class MyMethodAttribute : Attribute
{
}
```

는 어트리뷰트 MyMethod를 정의한다. MyMethodAttribute 대신 MyMethod를 써도 된다.

다음 예제는 클래스와 구조체에 적용할 수 있는 어트리뷰트를 정의하고 클래스에 적용한 것이다.

프로젝트 TestCustomAttribute

```
using System;

[AttributeUsage(AttributeTargets.Class |
                AttributeTargets.Struct)
]
class Creator : Attribute
{
    private string author;
    public int date;
```

```
        public Creator(string author)
        {
            this.author = author;
        }

        public override string ToString()
        {
            return string.Format("Creator: Author={0}, Date={1}", author, date);
        }
    }

    [Creator("장길산", date = 20140815)]
    class Sample
    {
    }

    class Program
    {
        static void Main(string[] args)
        {
            Type type = typeof(Sample);
            object[] attr = type.GetCustomAttributes(true);
            foreach (object obj in attr)
                Console.WriteLine(obj);
        }
    }
```

```
출력
    Creator: Author=장길산, Date=20140815
```

AttributeUsage에서 어트리뷰트를 적용할 수 있는 항목은 열거형 형식 AttributeTargets에서 확인할 수 있다. 또 옵션에는 AllowMultiple과 Inherited를 사용할 수 있다.

sealed class System.Runtime.InteropServices.AttributeUsage : Attribute

public bool AllowMultiple { get; set; }

하나의 프로그램 요소에 이 어트리뷰트를 여럿 지정할 수 있는 지 여부를 나타내는 부울 값을 가져오거나 설정한다.

public bool Inherited { get; set; }

파생된 클래스 또는 오버라이딩하는 멤버에 표시된 특성이 상속되는지 여부를 나타내는 부울 값을 가져오거나 설정한다.

◆ 연습문제 13.2.4. 다음 코드를 포함하는 프로젝트를 만들어 실행하시오.

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple=true)]
class AttrA : Attribute
{
    public string str;
}
[AttributeUsage(AttributeTargets.Class, AllowMultiple=false)]
class AttrB : Attribute
{
}

[AttrA(str="1st"), AttrA(str="2nd"), AttrB()]
class AttrTest
```

```
    {
    }
```

AttrB를 중복하여 사용하면 어떻게 되는 지 확인하시오.

13.3 대리자

메서드를 값으로 갖는 자료형을 **대리자**(delegate)라 한다. 대리자의 정의 형식은 다음과 같다.

delegate 반환_자료형 대리자_이름(매개_변수_자료형 매개_변수)

매개_변수는 형식적으로 적어 주는 것으로 아무런 역할도 하지 않는다. 매개_변수는 여러 개일 수도 있고 없을 수도 있다.

대리자 Del은

```
delegate int Del(int x);
```

와 같이 선언한다.

대리자의 사용법은 일반적인 자료형과 유사하다. 구조체나 클래스의 필드로 사용할 수 있으며 메서드의 매개 변수 자료형이나 반환 자료형이 될 수 있다. 또 프로퍼티나 인덱서의 자료형이 될 수 있다.

```
class A
{
    private Del myDel;
    public Del MyDel {}
    public Del this[Del del] {}
    public Del MyMethod(Del del) {}
}
```

13.3.1 메서드 연결. 대리자 변수의 값은 메서드 이름을 대입하여 설정한다. 대입하는 메서드는 대리자와 같은 형식이어야 한다.

```
class B
{
     public static int Foo(int x) {}
   public int Bar(int x) {}
}
```

에 대하여

```
Del op1 = B.Foo;
B b = new B();
Del op2 = b.Bar;
```

와 같이 설정한다. 인스턴스 메서드이든 정적 메서드이든 상관없다.

다음 프로젝트에서는 대리자를 함수의 매개 변수로 사용한다.

프로젝트 TestDelegate1

```
using System;
```

```
delegate Sample Del(Sample s);

struct Sample
{
    public int x;
    public Sample(int x)
    {
        this.x = x;
    }
    public Sample Add(Sample s)
    {
        return new Sample(x + s.x);
    }
    public Sample Evaluate(Del op, Sample s)
    {
        return op(s);
    }
}

class Program
{
    static void Main(string[] args)
    {
        Sample s1 = new Sample(133);
        Sample s2 = new Sample(-43);
        Del op = s2.Add;
        Console.WriteLine(op(s1).x);
        Console.WriteLine(s1.Evaluate(s1.Add, s2).x);
    }
}
```

```
출력
    90
    90
```

◆ 연습문제 13.3.1. 메서드 Multiply를 추가하여 대리자와 연결하시오.

◆ 연습문제 13.3.2. double형 변수를 매개 변수로 하여 double형 값을 반환하는 대리자를 정의하시오.

13.3.2 이름 없는 메서드 연결. 대리자에 연결하는 메서드를 해당 코드에서 직접 작성할 수 있다. 대리자 Del에 대하여 **이름 없는 메서드**(anonymous method)를 대입할 수 있다.

```
Del del = delegate(int x) { return x * x; };
```

등호의 오른편이 이름 없는 메서드이다. 이 식은

```
Del del = x => x * x;
```

와 같이 쓸 수도 있다. 등호의 오른편은 x를 대입하여 x * x를 반환한다는 의미이다. 이와 같은 형식을 람다 식이라 한다(13.2.3 참고).

프로젝트 TestDelegate2

```
using System;

class Program
{
    delegate int Del(int x);

    static void Main(string[] args)
    {
        Del del = delegate(int x) { return x * x; };
        Console.WriteLine(del(14));

        del = x => x * x;
        Console.WriteLine(del(23));
    }
}
```

출력

```
196
529
```

◆ 연습문제 13.3.3. double형 변수를 매개 변수로 하고 그 값을 콘솔에 출력하는 이름 없는 메서드를 연결하는 대리자를 작성하시오.

13.3.3 람다 식. 키워드 => 왼쪽에 매개 변수를 표시하고 오른쪽에 함수 본문을 적은 것을 **람다 식**(lambda expression)이라 한다. 13.2.2에서 본 바와 같이 대리자에 이름 없는 메서드를 연결할 때 사용하면 편리하다.

람다 식은

(매개_변수) => 본문

의 형태로 정의된다. 매개 변수는 없을 수도 있으며 여럿일 수도 있다. 본문은 값을 반환할 수도 있으며 그렇지 않을 수도 있다. 다음은 사용 가능한 람다 식들이다.

() => 1;
매개 변수는 없고 1을 반환하는 함수.

() => { return 1; }
매개 변수는 없고 1을 반환하는 함수. 문장을 포함하면 중괄호로 둘러싸야 한다.

x => x + 1;
매개 변수가 한 개이면 괄호를 생략할 수 있다.

(x, y) => { Console.WriteLine(x + y); }
매개 변수가 두 개인 경우.

위 문장들은 직관적으로 이해할 수 있을 것이다.

프로젝트 TestLambdaExpression

```
using System;
```

```
delegate int GetInt();
delegate int Operation(int x, int y);

class Foo
{
    private GetInt getInt;
    public GetInt GetInt
    {
        get { return getInt; }
    }

    public Foo(GetInt getInt)
    {
        this.getInt = getInt;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Random random = new Random();
        Foo foo = new Foo(() => random.Next(200));
        Console.WriteLine(foo.GetInt());

        Operation mod = (x, y) =>
        {
            if (x >= 0)
                return x % y;
            else
                return x % y + y;
        };
        Console.WriteLine(mod(-10, 3));
    }
}
```

```
출력
    39
    2
```

람다 식에서 매개 변수의 자료형은 대리자의 형식으로부터 유추한다. 그러나 자료형을 유추할 수 없는 경우도 있다. 이 경우

```
(string s, int n) => s.Length > n;
```

와 같이 매개 변수의 자료형을 적어줄 수도 있다.

13.4 메서드의 매개 변수

C#에서는 메서드의 매개 변수에 대하여 융통성을 부여하고 있다. 매개 변수 개수를 확정하지 않고 호출할 때 결정할 수 있다. 또 값 형식 매개 변수의 값을 변경할 수 있다.

13.4.1 params. 매개 변수의 개수는 메서드의 정의에서 결정된다. 호출할 때는 정의

된 개수만큼 매개 변수를 넘겨주어야 한다. 그러나 개수를 확정하지 않고 호출할 때마다 다르게 지정하는 방법이 있다.

메서드의 매개 변수에 키워드 params를 붙이면 여러 매개 변수의 배열로 인식한다.

```
int Sum(params int[] args)
{
}
```

와 같이 정의된 메서드 Sum은 int형 매개 변수를 개수 제한 없이 받아들인다. 따라서

```
int x = Sum(1, 2);
x = Sum(1, 2, 3);
```

와 같은 호출이 가능하다.

프로젝트 TestParams

```
using System;

class Program
{
    static int Sum(params int[] args)
    {
        int sum = 0;
        foreach (int x in args)
            sum += x;
        return sum;
    }

    static void Main(string[] args)
    {
        Console.WriteLine(Sum(1, 2));
        Console.WriteLine(Sum(1, 2, 3));
    }
}
```

출력

```
3
6
```

매개 변수가 여럿일 때도 params를 사용할 수 있다. 그러나 params가 붙은 매개 변수는 오직 하나여야 하며 마지막 매개 변수가 되어야 한다.

```
void Foo(string str, params object[] args)
{
}
```

와 같이 params가 붙지 않은 매개 변수를 먼저 적어야 한다.

◆ 연습문제 13.4.1. 매개 변수를 모두 출력하는 메서드를 작성하시오.

◆ 연습문제 13.4.2. double형 매개 변수를 여럿 입력하여 최대값을 구하는 메서드 Max()를 작성하시오.

13.4.2 참조에 의한 호출. C#의 메서드는 값에 의한 호출을 기본으로 한다. 따라서 메서드에서 매개 변수의 값을 변경해도 호출한 곳에서는 영향을 받지 않는다.

키워드 ref와 out은 **참조에 의한 호출**(call by reference)을 지원한다.

```
void AddOne(ref int x)
{
    ++x;
}
```

로 정의한 메서드 AddOne()에서 매개 변수 x의 값을 변경하면 호출한 곳에서도 값이 바뀐다.

```
int a = 10;
AddOne(ref a);
```

를 실행하면 a의 값이 11이 된다. 메서드 정의와 호출하는 곳 양쪽에 ref를 적어주어야 한다. 또 ref를 사용하려면 a의 값이 반드시 초기화되어야 한다.

키워드 out은 ref와 사용법이 동일하지만 호출하는 곳에서 변수를 초기화하지 않아도 된다.

```
void Read(out int x)
{
    x = Int32.Parse(Console.ReadLine());
}
```

으로 정의한 메서드 Read()는

```
int a;
AddOne(out a);
```

와 같이 호출한다. a의 값은 초기화하지 않아도 된다.

다음 프로젝트에서는 int형 변수에 out와 ref를 적용한다. 또 out을 써서 클래스의 인스턴스를 초기화한다.

프로젝트 TestRefOut

```
using System;

class Sample
{
    public int x;
}

class Program
{
    static void AddOne(ref int x)
    {
        ++x;
    }

    static void Read(out int x)
    {
        x = Int32.Parse(Console.ReadLine());
```

```
        }

        static void Read(out Sample sample)
        {
            sample = new Sample();
            sample.x = Int32.Parse(Console.ReadLine());
        }

        static void Main()
        {
            int x;
            Read(out x);
            AddOne(ref x);
            Console.WriteLine(x);

            Sample s;
            Read(out s);
            Console.WriteLine(s.x);
        }
    }
```

```
입력
    34
    21
출력
    35
    21
```

◆ 연습문제 13.4.3. 두 매개 변수의 값을 바꾸는 메서드 Swap()을 작성하시오.

13.5 제네릭

상황에 맞는 자료형을 적용할 수 있도록 융통성을 부여하는 방식을 **제네릭**(generics)이라 한다. 정의에 **형식 매개 변수**(type parameter)를 추가하여 자료형을 나중에 결정할 수 있도록 한다.

13.5.1 제네릭 클래스. 이름에 형식 매개 변수를 붙여 적은 클래스를 **제네릭 클래스**(generic class)라 한다. 형식 매개 변수를 T로 하는 제네릭 클래스 A는

```
class A<T>
{
    T x;
}
```

와 같이 정의한다. 클래스 A<T>는 형식 매개 변수 T에 따라 달라지며 필드 x의 자료형은 T가 int이면 int형, double이면 double형이 된다.

```
A<int> a = new A<int>();
```

와 같이 정의를 적용하는 곳에서 형식 매개 변수를 치환할 자료형을 적어준다. 이 때 T는 모두 int로 치환된다.

프로젝트 TestGenerics1

```
using System;

class GenList<T>
{
    private int length;
    private T[] contents;

    public GenList(int maxLength)
    {
        contents = new T[maxLength];
    }

    public void Add(T element)
    {
        contents[length++] = element;
    }

    public override string ToString()
    {
        string str = "";
        for (int i = 0; i < length; i++)
            str += contents[i] + " ";
        return str;
    }
}

class Program
{
    class Sample { }

    static void Main(string[] args)
    {
        GenList<int> gl1 = new GenList<int>(4);
        gl1.Add(1);
        gl1.Add(-1);
        Console.WriteLine("gl1: " + gl1);

        GenList<Sample> gl2 = new GenList<Sample>(5);
        gl2.Add(new Sample());
        gl2.Add(new Sample());
        Console.WriteLine("gl2: " + gl2);
    }
}
```

```
gl1: 1 -1
gl2: Sample Sample
```

클래스 Sample은 클래스 Test 내부에서 정의한 클래스이다. 이와 같이 클래스 내부에서 정의한 자료형을 내부 형식이라 한다.

◆ 연습문제 13.5.1. 위 프로젝트에서 GenList<double> 객체를 만들고 사용하시오.

형식 매개 변수를 둘 이상 사용하려면 반점으로 분리한다.

```
class B<T, U>
{
```

```
        private T t;
        private U u;
    }
```

는 두 형식 매개변수 T와 U를 사용한다.

13.5.2 제네릭 메서드. 메서드에 형식 매개 변수를 붙여 적은 것을 **제네릭 메서드**(generic method)라 한다.

```
    void swap<T>(ref T x, ref T y)
    {
        T temp = x;
        x = y;
        y = temp;
    }
```

에서 x, y의 자료형은 T에 따라 달라진다.

```
    int x = 3;
    int y = -5;
    swap<int>(ref x, ref y);
```

와 같이 호출하면 x와 y의 값이 교환된다.

◆ 연습문제 13.5.2. 위 코드를 모으고 수정하여 프로젝트를 만드시오.

메서드를 호출할 때 형식 매개 변수의 자료형을 유추할 수 있으면 그것을 생략할 수 있다. 위에서 호출한 swap의 경우 x나 y의 자료형으로부터 T를 유추할 수 있으므로

```
    swap<int>(ref x, ref y);
```

대신

```
    swap(ref x, ref y);
```

와 같이 호출할 수 있다.

13.5.3 형식 제한. 형식 매개 변수에 조건을 붙여 사용할 수 있는 형식을 제한할 수 있다. 이 때 사용하는 키워드가 where이다.

```
    class Sample<T> where T : struct
```

는 T가 구조체임을 나타낸다. where 뒤에 붙일 수 있는 제한 조건은 다음 표와 같다.

제한	설명
where T : class	T는 참조 형식이다
where T : struct	T는 값 형식이다
where T : new()	T는 기본 생성자를 갖는다
where T : 인터페이스_이름	T는 주어진 인터페이스를 구현한다
where T : 클래스_이름	T는 주어진 클래스의 파생 클래스이다
where T : U	T는 U의 파생 클래스이다.

표 13.40 제네릭 형식 제한

제한 조건을 둘 이상 사용하려면 반점으로 분리한다. 형식 매개 변수를 둘 이상 사용하는 경우에는 where를 따로 적어준다.

```
class Foo<T> where T : class, new()
{ }
class Bar<T, U>
    where T : U
    where U : Foo
{ }
```

다음 예제는 제네릭 메서드에 대하여 형식을 제한한 것이다.

프로젝트 TestGenerics2

```
using System;
using System.Collections;

class Program
{
    static void Enumerate<T>(T src) where T : class, IEnumerable
    {
        foreach (object o in src)
            Console.WriteLine(o);
    }

    static void Main(string[] args)
    {
        ArrayList al = new ArrayList();
        al.Add("산은 높고");
        al.Add(256);
        al.Add("물은 깊어");

        Enumerate(al);
    }
}
```

```
출력
    산은 높고
    256
    물은 깊어
```

Enumerate(al)을 호출할 때 형식 매개 변수 T가 ArrayList임을 유추할 수 있으므로 <ArrayList>를 생략하였다.

13.6 내부 형식

클래스나 구조체 내부에 정의된 형식을 **내부 형식**(inner type) 또는 **중첩 형식**(nested type)이라 한다. 열거형 형식, 클래스, 구조체 등이 내부 형식이 될 수 있다.

내부 형식은 다음과 같이 정의한다.

```
class A
{
    public enum InnerEnum {}
    public struct InnerStruct {}
    public class InnerClass {}
}
```

정의된 클래스 밖에서 내부 형식을 사용하려면 클래스 이름을 붙여 적어야 한다.

```
A.InnerEnum e;
A.InnerStruct s = new A.InnerStruct();
A.InnerClass c = new A.InnerClass();
```

클래스 A 내부에서는 A를 붙이지 않고 이름만 적어도 된다.

다음 프로젝트에서는 내부 형식을 정의하고 사용한다.

프로젝트 TestInnerType

```
using System;

class Fruit
{
    public enum Kind
    {
        Apple,
        Pear,
        Peach
    }

    public struct Info
    {
        public int Weight { get; set; }
    }

    public class Color
    {
        public string Name { get; set; }
    }

    public Kind K { get; set; }
    public Info I { get; set; }
    public Color C { get; set; }
}

class Program
{
    static void Main(string[] args)
    {
        Fruit fruit = new Fruit();
```

```
            fruit.K = Fruit.Kind.Apple;

            Fruit.Info info = new Fruit.Info();
            info.Weight = 100;
            fruit.I = info;

            Fruit.Color color = new Fruit.Color();
            color.Name = "Red";
            fruit.C = color;

            Console.WriteLine("{0}, {1}, {2}",
                fruit.K, fruit.I.Weight, fruit.C.Name);
        }
    }
```

출력

```
Apple, 100, Red
```

◆ 연습문제 13.6.1. 내부 형식 Info에 인덱서를 추가하시오.

◆ 연습문제 13.6.2. 내부 형식 Color에 메서드를 추가하시오.

◆ 연습문제 13.6.3. 구조체를 만들고 그 구조체의 내부 형식을 정의하시오.

13.7 형식의 확장과 판별

이 절에서는 값 형식을 참조 형식으로 변환하는 방법과 null을 대입할 수 있도록 확장하는 방법에 대하여 알아본다. 또 형식을 판별하는 방법에 대하여 알아본다.

13.7.1 값 형식 참조. 값 형식을 참조 형식으로 변환하는 것을 boxing이라 한다. 변환할 수 있는 형식은 클래스 object나 인터페이스이다.

```
object obj = 3;
```

과 같이 object 객체를 생성하면 int형 값 3의 boxing이 이루어진다. boxing을 푸는 것을 unboxing이라 한다.

```
int x = (int)obj;
```

와 같이 형 변환을 하면 unboxing이 이루어진다.

프로젝트 TestBoxing

```
using System;

class Program
{
    static void Main(string[] args)
    {
        object obj = 3;
        Console.WriteLine(obj);

        int x = (int)obj;
```

```
            Console.WriteLine(x);
        }
    }
```

```
출력
    3
    3
```

◆ 연습문제 13.7.1. 위 프로젝트에 double형에 대한 boxing과 unboxing을 추가하시오.

◇ boxing과 unboxing은 내부적으로 상당한 계산을 필요로 한다.

13.7.2 값 형식 확장. 정수 나눗셈을 하는 경우 분모가 0이면 결과가 없으며 이것을 정수로 나타낼 수는 없다. 이와 같이 변수에 값을 대입할 수 없는 경우에 사용할 수 있는 것이 nullable 형식이다.

키워드 null은 값이 없음을 나타낸다. 값 형식을 확장하여 null을 값으로 가질 수 있도록 만든 것을 **nullable 형식**이라 한다. 이 형식으로 정의되는 객체는 구조체 System.Runtime.Nullable<T>의 인스턴스이다. 주어진 자료형을 nullable 형식으로 확장한 것은

Nullable<자료형>

또는 이것을 줄여서

자료형?

과 같이 적는다. 두 표현은 같은 것이며 어느 것을 써도 상관없다.

◇ 참조 형식은 null을 값으로 가질 수 있으므로 nullable 형식으로 확장할 수 없다.

int형을 nullable 형식으로 확장하여 변수를 정의하려면

```
Nullable<int> x = null;
```

또는

```
int? x = null;
```

로 적는다. x는 int형 변수가 가질 수 있는 값 또는 null을 값으로 가진다.
구조체 Nullable<T>에는 두 프로퍼티 HasValue와 Value가 있다.

struct System.Runtime.Nullable<T> where T : struct
public bool HasValue { get; }
객체에 값이 있는지 여부를 가져온다. 값이 있으면 true, null이면 false를 반환한다.
public T Value { get; }
저장된 값이 null이 아니면 그 값을 가져온다. 저장된 값이 null이면 System.InvalidOperationException을 반환한다.

nullable 형식 변수의 값을 가져오려면

```
if(x.HasValue)
    Console.WriteLine(x.Value);
```

와 같이 먼저 HasValue로 null인지 아닌지 판단하고 그 다음 프로퍼티 Value를 사용하는 것이 일반적이다. x.Value 대신 x를 써도 된다.

두 int?형 변수의 연산에 대하여 알아보자. 두 변수 모두 정수 값을 가지면 int형과 마찬가지로 연산을 수행한다. 둘 중 하나가 null이면 연산 결과는 null이다.

```
int? x = 123;
int? y = null;
int? z = x + y;
```

를 실행하면 z의 값은 null이다. 다른 자료형에 대한 연산도 마찬가지이다.

bool?형 변수의 논리 연산에 대하여 알아보자. 두 bool?형 변수 b, c의 논리합과 논리곱은 각각 b | c, b & c로 적는다. 두 변수 모두 bool형 값을 가지면 연산 결과는 bool형과 같다. 둘 중 하나가 null이면 연산 결과는 상황에 따라 다르다. 다음 표를 참고하자.

x	y	x & y	x | y
true	true	true	true
true	false	false	true
true	null	null	true
false	true	false	true
false	false	false	false
false	null	false	null
null	true	null	true
null	false	false	null
null	null	null	null

표 13.2 bool?형의 논리 연산

nullable 형식 변수를 nullable이 아닌 변수에 대입할 때 null을 배제하는 방법으로 연산자 ??를 사용할 수 있다.

변수_값 ?? 대체_값

형식으로 적용된다. 연산 결과는 변수의 값이 null이면 대체_값이고 그렇지 않으면 변수_값이다.

```
int? x = null;
int y = x ?? -1;
```

에서 x 값이 null이므로 y 값은 -1이다.

다음 예제는 위에서 설명한 내용을 포함하고 있다.

프로젝트 TestNullableType

```
using System;
```

```
class Program
{
    static void Print<T>(string str, T? x) where T : struct
    {
        Console.Write(str + " = ");
        if (x.HasValue)
            Console.WriteLine(x.Value);
        else
            Console.WriteLine("값 없음");
    }

    static void Main(string[] args)
    {
        int? x = null;
        Print("x", x);

        int? y = 235;
        Print("y", y);

        Print("x + y", x + y);

        x++;
        Print("x++", x);

        if (x < y)
            Console.WriteLine("작다");
        else if (x > y)
            Console.WriteLine("크다");
        else if(x == y)
            Console.WriteLine("같다");
        else
            Console.WriteLine("항상 거짓");

        bool? a = true;
        Print("a", a);

        bool? b = null;
        Print("a & b", a & b);
        Print("a | b", a | b);

        double? u = null;
        double? v = u ?? 12.34;
        Print("v", v);
    }
}
```

출력

```
x = 값 없음
y = 235
x + y = 값 없음
x++ = 값 없음
항상 거짓
a = True
a & b = 값 없음
a | b = True
v = 12.34
```

값이 null인 nullable 형식 변수는 콘솔에 ""로 출력된다.

◆ 연습문제 13.7.2. 다음 문장을 실행하여 결과를 확인하시오.

```
int? x = null;
Console.WriteLine(x);
```

◆ 연습문제 13.7.3. bool?형 변수에 대한 부정 !를 사용해 보고 결과를 말하시오.

13.7.3 형식 판별. 메서드의 매개 변수가 Object형과 같이 일반적인 자료형일 때 그 변수의 정확한 자료형을 구하는 방법을 알아보자.

키워드 is는 변수가 해당 자료형에 포함되는지 판별할 때 사용한다.

변수 is 자료형

과 같이 적용한다. 결과는 bool형이며 변수가 자료형에 포함되면 참, 그렇지 않으면 거짓이다.

```
int x = 32;
bool b = x is double;
```

에서 x가 double이 아니므로 b 값은 false가 된다.

프로젝트 TestIs

```
using System;
class Foo
{
    private int x;
    public int X
    {
        get { return x; }
        set { x = value; }
    }
}

class Program
{
    static void PrintFoo(object obj)
    {
        if (obj is Foo)
            Console.WriteLine(((Foo)obj).X);
        else
            Console.WriteLine("Foo가 아님");
    }

    static void Main()
    {
        Foo f = new Foo();
        f.X = 299;
        PrintFoo(f);

        int x = 4;
        PrintFoo(x);
```

```
            Console.WriteLine(f is Object);
        }
    }
```

```
299
Foo가 아님
True
```

◆ 연습문제 13.7.4. 다음 코드를 실행하여 결과를 확인하시오.

```
object obj = 32;
Console.WriteLine(obj is Int32);
Console.WriteLine(obj is Double);
```

클래스 Type은 자료형에 대한 정보를 가져올 때 사용한다. 필드, 프로퍼티, 메서드, 인터페이스, 기본 클래스 등 여러 가지 정보가 포함되어 있다. Type 객체를 얻으려면

typeof(자료형)

또는

변수.GetType()

을 사용한다. 클래스 Type의 정적 메서드 GetType()으로도 얻을 수 있다.

Type.GetType("자료형")

예를 들어

```
Type type = typeof(int);
```

는 int형에 대한 Type 객체를 반환한다.

abstract class System.Type
public static Type GetType(string typeName)
이름이 typeName인 type 객체를 가져온다.
public abstract string Name { get; }
이름을 가져온다.
public abstract string FullName { get; }
네임스페이스를 포함한 이름 전체를 가져온다.
public ConstructorInfo GetConstructor(Type[] types)
지정된 배열의 형식과 일치하는 매개 변수를 가진 public인 인스턴스 생성자를 반환한다. 매개 변수가 없으면 null을 대입한다. 형식과 일치하는 생성자가 없으면 null을 반환한다.

클래스 ConstructorInfo는 생성자에 대한 정보를 포함하고 있으며 생성자를 호출하여 객체를 생성한다.

abstract class System.Reflecton.ConstructorInfo
public object Invoke(object[] parameters)
해당 생성자를 호출하여 인스턴스를 생성한다. 매개 변수가 없으면 null을 대입한다.

다음 프로젝트는 Type을 사용하여 객체를 생성하고 판별한다.

프로젝트 TestType

```
using System;
using System.Reflection;
using TestType;

namespace TestType
{
    class Foo
    {
        private int x;
        public int X
        {
            get { return x; }
        }

        public Foo(int x)
        {
            this.x = x;
        }

        public class Bar
        {
            private int y;
            public int Y
            {
                get { return y; }
            }

            public Bar(int y)
            {
                this.y = y;
            }
        }
    }
}

class Program
{
    static void Print(object obj)
    {
        Type type = obj.GetType();
        if (type.Name == "Foo")
            Console.WriteLine(((Foo)obj).X);
        else if(type.Name == "Bar")
            Console.WriteLine(((Foo.Bar)obj).Y);
        else
            Console.WriteLine("해당 없음");
    }

    static object GetInstance(string typeName, int x)
    {
        Type type = Type.GetType(typeName);
        ConstructorInfo info =
            type.GetConstructor(new Type[] { typeof(int) });
        return info.Invoke(new object[] {x});
```

```
        }

        static void Main()
        {
            Print(3);

            Type type = typeof(Foo);
            Console.WriteLine(type.FullName);

            Foo f = (Foo)GetInstance("TestType.Foo", 123);
            Print(f);

            type = typeof(Foo.Bar);
            Console.WriteLine(type.FullName);

            Foo.Bar b = (Foo.Bar)GetInstance("TestType.Foo+Bar", -215);
            Print(b);
        }
    }
```

```
    해당 없음
    TestType.Foo
    123
    TestType.Foo+Bar
    -215
```

내부 클래스 Bar의 FullName은 TestType.Foo+Bar이며 Type.GetType()에도 이것이 사용됨을 알 수 있다.

◆ 연습문제 13.7.5. 클래스 Type의 메서드 GetProperties()를 사용하는 프로그램을 작성하시오.

13.8 상수와 읽기 전용 필드

가로와 세로로 칸을 나누고 그 개수를 사용하는 보드 게임을 만든다고 가정해 보자. 칸의 개수를 숫자로 사용하는 것보다 그 값을 변수 또는 필드에 대입하여 사용하는 것이 여러모로 편리하다. 프로그램을 만들다가 칸의 개수를 바꾸려면 숫자를 모두 찾아서 바꾸는 것보다 변수 값을 바꾸는 것이 편이 나을 것이다. 기억하기 쉬운 것도 장점이다.

한 번 값을 설정하면 프로그램 실행 도중에 바꿀 일이 없는 변수 또는 필드를 설정하는 방법을 알아보자.

13.8.1 상수 변수. 선언할 때 값을 설정하고 변경할 수 없는 변수를 **상수 변수** (constant local)라 한다. 상수와 같은 역할을 한다. 키워드 const는 지역 변수를 상수 변수로 지정한다. 선언할 때 반드시 값을 초기화해야 하며 나중에는 값을 변경할 수 없다.

```
    const double gravitationalAcceleration = 9.80665;
```

와 같이 선언된 gravitationalAcceleration은 상수 변수이다.

13.8.2 상수 필드. 선언 문장에서 값을 설정하고 그 이후에는 변경할 수 없는 필드를 **상수 필드**(constant field)라 한다. 키워드 const를 붙여서 선언한다.

```
class Foo
{
    public const int maximum = 10;
}
```

와 같이 정의된 상수 필드 maximum은 정적 필드와 마찬가지로

```
Console.WriteLine(Foo.maximum);
```

과 같이 클래스 이름 뒤에 이름을 적어서 접근한다.

두 키워드 const와 static을 동시에 붙일 수 없다. 다시 말하면 상수 필드를 정적 필드로 선언할 수 없다.

13.8.3 읽기 전용 필드. 선언 문장 또는 생성자에서 값을 설정하고 그 이후에는 변경할 수 없는 필드를 **읽기 전용 필드**라 한다. 선언 문장에서 초기화했더라도 생성자에서 값을 변경할 수는 있다. 상수 필드가 선언할 때 초기화되어야 하는 점과 다르다. 키워드 readonly를 붙여서 선언한다.

```
class Bar
{
    public readonly double minimum = 1.7;
}
```

와 같이 정의된 읽기 전용 필드는

```
Bar bar = new Bar()
double d = bar.minimum + 1.2;
```

와 같이 객체 뒤에 이름을 적어서 접근한다.

정적 읽기 전용 필드는 키워드 static을 붙여서 선언한다.

```
class Bar
{
    public static readonly double bound = 2.7;
}
```

와 같이 정의된 읽기 전용 필드는

```
double d = Bar.minimum + 0.2;
```

와 같이 객체 뒤에 이름을 적어서 접근한다. 정적 읽기 전용 필드 역시 선언 문장이나 정적 생성자에서 초기화할 수 있다.

다음 예제에서는 상수 필드와 읽기 전용 필드를 사용한다.

프로젝트 TestReadonly

```
using System;

class Sample
{
    public const bool turbo = true;

    public static readonly int countMax;

    static Sample()
    {
        countMax = 63;
    }

    private readonly double maximum = 10;
    public double Maximum
    {
        get { return maximum; }
    }

    private readonly double minimum = 0;
    public double Minimum
    {
        get { return minimum; }
    }

    public Sample(double minimum)
    {
        this.minimum = minimum;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine(Sample.turbo);
        Console.WriteLine(Sample.countMax);

        Sample s = new Sample(1.2);
        Console.WriteLine(s.Maximum);
        Console.WriteLine(s.Minimum);
    }
}
```

출력

```
True
63
10
1.2
```

13.9 상속 제한

클래스를 상속하여 파생 클래스를 만드는 것을 방지할 수 있다. 클래스를 의도한 용도로만 사용되도록 제한하는 것이다. 멤버를 오버라이딩하는 것도 제한할 수 있다.

13.9.1 봉인 클래스. 클래스를 상속하지 못하도록 키워드 sealed를 붙여 제한한 것을

봉인 클래스(sealed class)라 한다. 곧 봉인 클래스는 파생 클래스를 만들 수 없다.

```
sealed class A
{
}
```

이와 같이 정의된 클래스 A의 파생 클래스는 만들 수 없다.

예를 들어 System.Drawing.Font는 상속이 제한된 봉인 클래스이다.

13.9.2 봉인 멤버. 멤버에 키워드 sealed를 붙여 오버라이딩할 수 없도록 제한한 것을 **봉인 멤버**라 한다. 기본 클래스에서 virtual로 정의된 멤버는 파생 클래스에서 override를 붙여 오버라이딩한다. 이 때 sealed를 함께 붙이면 파생 클래스에서 다시 오버라이딩할 수 없다.

```
class A
{
    public virtual int this[int n] { get{} set{} }

    public virtual int Prop { get{} set{} }

    public virtual void F1() { }
    public virtual void F2() { }
}
class B : A
{
    public sealed override int this[int n] { get{} set{} }
    public sealed override int Prop { get{} set{} }

    public override void F1() { }
    public sealed override void F2() { }
}
class C : B
{
    // 인덱서 this는 override를 붙여 정의할 수 없다.
    // Prop는 override를 붙여 정의할 수 없다.
    public override void F1() {}
    // F2는 override를 붙여 정의할 수 없다.
}
```

클래스 C에서는 인덱서 int this[int n], 프로퍼티 Prop, 메서드 F2()를 오버라이딩할 수 없다.

◆ 연습문제 13.9.1. 위 코드를 실행 가능하도록 수정하고 프로젝트에 포함시켜 실행하시오.

13.10 정적 클래스

키워드 static을 붙여 정의한 클래스를 **정적 클래스**(static class)라 한다. 상수 필드와 정적 멤버로 구성된다. 매개 변수에 따라 동작할 뿐 인스턴스 필드를 필요로 하지 않는 정적 메서드를 모아 관리할 수 있다.

정적 클래스는 인스턴스 생성자를 포함할 수 없으며 인스턴스화할 수 없다. 파생

클래스를 만들 수 없으며 기본 클래스는 반드시 System.Object이어야 한다.

다음 예제는 정적 클래스를 정의하고 사용한다.

프로젝트 TestStaticClass

```
using System;

static class Conversion
{
    public const double cmPerInch = 2.54;
    public const double inchPerCm = 1 / cmPerInch;

    public static double Cm2Inch(double cm)
    {
        return cm / inchPerCm;
    }
    public static double Inch2Cm(double inch)
    {
        return inch * cmPerInch;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine(Conversion.Inch2Cm(2.5));
    }
}
```

출력

```
6.35
```

◆ 연습문제 13.10.1. 마일과 km을 변환하는 정적 메서드를 추가하시오.

정적 클래스 Math는 수학에서 사용되는 상수와 함수들을 모아놓은 것이다.

static class System.Math

public const double PI = 3.14159
원주율을 나타낸다.

public static double Cos(double u)
$\cos u$를 반환한다.

public static double Sin(double u)
$\sin u$를 반환한다.

위에서 나열한 것 외에도 많은 정적 메서드가 정의되어 있다.

다음 예제는 원의 넓이와 원 위의 점을 계산한다.

프로젝트 TestMath

```
using System;

struct Point
{
```

```
        private double x;
        public double X
        {
            get { return x; }
            set { x = value; }
        }

        private double y;
        public double Y
        {
            get { return y; }
            set { y = value; }
        }

        public Point(double x, double y)
        {
            this.x = x;
            this.y = y;
        }
    }

    class Circle
    {
        private Point center;
        private double radius;

        public Circle(Point center, double radius)
        {
            this.center = center;
            this.radius = radius;
        }

        public double GetArea()
        {
            return Math.PI * radius * radius;
        }

        public Point GetPoint(double angle)
        {
            return  new Point(center.X + radius * Math.Cos(angle),
                              center.Y + radius * Math.Sin(angle));
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Point center = new Point(0,0);
            Circle circle = new Circle(center, 1);
            Point p = circle.GetPoint(Math.PI / 4);

            Console.WriteLine(p.X + ", " + p.Y);
        }
    }
```

출력

```
3.14159265358979
0.707106781186548, 0.707106781186547
```

◆ 연습문제 13.10.2. Math.Abs를 사용하는 프로젝트를 만드시오.

◆ 연습문제 13.10.3. 직선과 x축이 이루는 각의 크기를 계산하는 메서드를 만드시오.

13.11 이벤트

윈도우에 포함된 객체는 특정한 사건이 일어났을 때 **이벤트**(event)를 발생시켜 다른 객체에 사건이 일어났음을 알린다. 이벤트를 발생시키는 객체를 **게시자**(publisher)라 하고 이벤트를 받는 객체를 **구독자**(subscriber)라 한다.

사용자가 버튼을 누르면 이 버튼은 게시자가 되어 이벤트를 발생시킨다. 이벤트를 받은 구독자는 상황에 맞는 적절한 처리 과정을 수행한다.

일반적으로 구독자는 이벤트를 받아 처리하는 메서드를 포함하고 있다. 이 메서드를 **이벤트 처리 함수**(event dispatcher, event handler) 또는 **이벤트 처리기**라 한다.

이벤트를 게시하려면 사건 정보를 담는 클래스가 필요하다. 이 클래스는 System.EventArgs의 파생 클래스이어야 한다.

```
class MyEventArgs : EventArgs
{
}
```

이벤트 처리 함수를 등록해야 게시자가 그 함수에 정보를 전달할 수 있다. 등록할 장소는 게시자 클래스의 대리자 필드이며 키워드 event를 붙여 선언해야 한다. 필드 이름은 파스칼 표기법으로 쓰도록 권장한다.

```
class Publisher
{
    // 등록 장소
    public event EventHandler<MyEventArgs> MyEvent;
}
```

등록 장소인 대리자는

```
public delegate void EventHandler<TEventArgs>(object sender, TEventArgs e)
```

형식이며 TEventArgs는 위에서 정의한 MyEventArgs이다.

이벤트 처리 함수는 구독자 클래스에 정의된다.

```
class Subscriber
{
    public void Subscribe(object sender, MyEventArgs args)
    {
    }
}
```

sender는 게시자 객체이며 args는 이벤트 정보를 갖는 객체이다. 이 메서드를 등록하려면 +=를 사용한다.

```
Publisher pub = new Publisher();
```

```
Subscriber sub = new Subscriber();
pub.MyEvent += sub.Subscribe;
```

sub.Subscribe는 pub에서 발생한 이벤트 처리 함수로 등록된다.

다음은 이벤트를 발생시키고 처리하는 과정으로 콘솔 프로그램으로 작성한 것이다.

프로젝트 TestEvent

```
using System;

class MyEventArgs : EventArgs
{
    private string message;
    public string Message
    {
        get { return message; }
    }

    public MyEventArgs(string message)
    {
        this.message = message;
    }
}

class Publisher
{
    public event EventHandler<MyEventArgs> MyEvent;

    public void Publish()
    {
        if (MyEvent != null)
            MyEvent(this, new MyEventArgs("이벤트 발생"));
    }
}

class Subscriber
{
    public void Subscribe(object sender, MyEventArgs args)
    {
        Console.WriteLine(args.Message);
    }
}

class Program
{
    static void Main(string[] args)
    {
        Publisher pub = new Publisher();
        Subscriber sub = new Subscriber();
        pub.MyEvent += sub.Subscribe;

        pub.Publish();
    }
}
```

출력

```
이벤트 발생
```

코드

```
pub.Publish();
```

가 호출되면

```
MyEvent(this, new MyEventArgs("이벤트 발생"));
```

이 실행되고 이벤트를 발생시켜 등록된 처리 함수에게 전달한다.

◆ 연습문제 13.11.1. 위 프로젝트의 구독자 메서드에 게시자를 출력하는 코드를 추가하시오.

◆ 연습문제 13.11.2. 위 프로젝트에 구독자 객체

```
Subscriber sub2 = new Subscriber();
```

를 추가하고 이 객체의 메서드를 이벤트 처리 함수로 등록하시오.

◆ 연습문제 13.11.3. EventArgs의 파생 클래스와 이벤트 게시자를 하나 더 추가하시오.

구독자 메서드를 등록할 때

```
pub.MyEvent += sub.Subscribe;
```

대신

```
pub.MyEvent += new EventHandler<MyEventArgs>(sub.Subscribe);
```

를 써도 된다.

◇ 이벤트는 주로 폼 프로그램에서 사용한다.

13.12 콘솔과 윈도우즈 폼

윈도우즈 폼 프로그램은 콘솔 창을 기본으로 제공하지 않는다. 그러나 콘솔이 필요하면 사용할 수 있다.

13.12.1 폼에서 콘솔 사용하기. 윈도우즈 폼 프로그램에서 콘솔을 사용하는 방법을 알아보자.

새 프로젝트 TestFormConsole을 Windows Forms 응용 프로그램으로 추가하자. 폼의 중앙에 버튼을 배치하고 더블 클릭하여 이벤트 Click을 처리하는 함수를 등록한다. Form1.cs를 다음과 같이 수성하자.

프로젝트 TestFormConsole

Form1.cs

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

```
{
    Console.WriteLine("콘솔 출력");
    MessageBox.Show("메시지 박스 출력");
}
```

메뉴에서

프로젝트(P) -> TestFormConsole 속성(P)

를 선택한다. 응용 프로그램 탭에서

출력 형식(U)

를

콘솔 응용 프로그램

으로 바꾼다. 프로그램을 실행하면 화면에 콘솔 창이 나타난다. 프로그램을 실행하고 버튼 button1을 눌러 콘솔에 표시된 출력 내용을 확인하자.

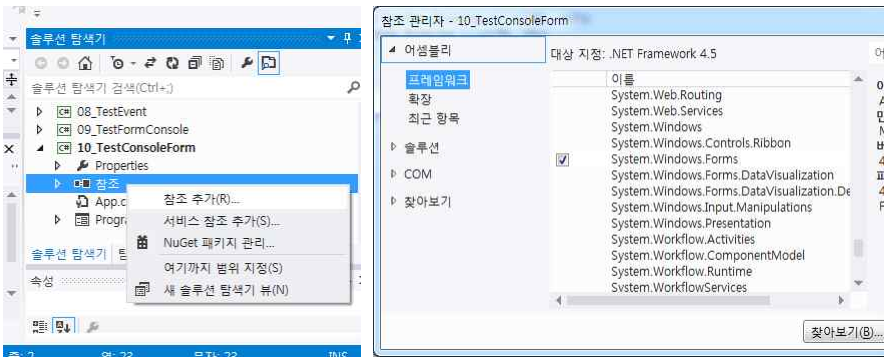
13.12.2 콘솔에서 폼 사용하기. 콘솔 프로그램에서 윈도우즈 폼을 사용하는 방법도 있다. 새 프로젝트 TestConsoleForm을 콘솔 응용 프로그램으로 추가하자. 솔루션 탐색기에서 프로젝트 TestConsoleForm의

참조

항목을 오른 클릭하여

참조 추가(R)

을 선택한다. 참조 관리자 창에서 System.Windows.Forms 항목을 선택하고 확인을 누른다. 같은 방법으로 System.Drawing을 추가한다.



Program.cs를 다음과 같이 수정한다

프로젝트 TestConsoleForm

```
using System;
```

```
using System.Windows.Forms;
using System.Drawing;

class Program : Form
{
    public Program()
    {
        Button b = new Button();
        b.Text = "버튼1";
        b.Location = new System.Drawing.Point(99, 117);
        b.Size = new System.Drawing.Size(75, 23);
        Controls.Add(b);
    }
    static void Main(string[] args)
    {
        Application.Run(new Program());
    }
}
```

Program.cs를 더블 클릭하면 Program.cs [디자인]이 나타난다. 코드를 보려면 마우스 오른쪽 버튼을 눌러

코드 보기(C)

를 선택해야 한다.

◇ 메뉴에서

프로젝트(P) -> TestConsole 속성(P)

를 선택하여

출력 형식(U)

를

Windows 응용 프로그램

으로 바꾸면 실행할 때 콘솔 창이 나타나지 않는다.

◇ 메인 메서드의 코드

```
Application.Run(new Program());
```

은

```
new Program().Show();
Application.Run();
```

로 바꾸어도 된다.

전자는 폼과 메시지 루프인 Application이 결합되지만 후자는 분리된다. 전자는 폼을 닫으면 Application이 종료되지만 후자는 그렇지 않다. 따라서 후자의 경우 폼을 닫을 때 Application을 종료하는 코드를 추가해야 한다.

◇ 폼을 닫을 때 Application을 종료하려면 Form1의 이벤트 FormClosed를 처리하는

함수를 작성해야 한다. 이 함수에

```
Application.Exit();
```

을 적어주면 폼을 닫을 때 Application을 종료한다.

◆ 연습문제 13.12.1. 이벤트 FormClosed를 처리하는 함수를 등록하고 위 코드를 적으시오.

◆ 연습문제 13.12.2. 프로젝트 TestConsole에 버튼 클릭 이벤트를 추가해보자.