# 메소드와 인자 & 배열

321190
2012년 가을학기
9/19/2012
박경신

## Overview

- 효과적인 메소드 호출의 방법
- 중첩 메소드와 재귀 메소드
- 메소드와 변수와의 관계
- 메소드에 인자를 전달하여 호출
- 메소드 오버로딩
- 배열의 의미와 필요성
- 1차원 배열, 다차원 배열
- 배열의 요소에 접근하는 방법에 대한 이해
- 배열에서 지원하는 속성과 메소드 활용

## Class/Method

- C# FCL에는 많은 클래스들이 정의되어 있음 - 예를 들어,
  - Console
  - MessageBox
  - Int32
  - Math
- 클래스의 정의는 메소드(Method)와 자료 속성(Data Properties)를 포함
  - Method 예 - Console.Write(), Console.WriteLine(), Int32.Parse()
  - Property 예 – Int32.MinValue, Int32.MaxValue, Math.PI, Math.E

## Method

- 메소드(method)란 정해진 작업을 수행하기 위해 그룹으로 묶어, 이름을 붙인 명령문의 집합

```
접근지정자 리턴형 함수이름(인자 리스트)
{
        명령1;  명령2;  .....     return 리턴 값;
}
```

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

class　method　(parameters)

dot

# Method Examples: Math Class

| Method | Description | Example |
|---|---|---|
| **Abs( x )** | absolute value of *x* | **Abs( 23.7 )** is **23.7**<br>**Abs( -23.7 )** is **23.7** |
| **Ceiling( x )** | rounds *x* to the smallest integer not less than *x* | **Ceiling( 9.2 )** is **10.0**<br>**Ceiling( -9.8 )** is **-9.0** |
| **Cos( x )** | trigonometric cosine of *x* (*x* in radians) | **Cos( 0.0 )** is **1.0** |
| **Exp( x )** | exponential method *ex* | **Exp( 1.0 )** is approximately **2.7182818284590451**<br>**Exp( 2.0 )** is approximately **7.3890560989306504** |
| **Floor( x )** | rounds *x* to the largest integer not greater than *x* | **Floor( 9.2 )** is **9.0**<br>**Floor( -9.8 )** is **-10.0** |
| **Log( x )** | natural logarithm of *x* (base *e*) | **Log( 2.7182818284590451 )** is approximately **1.0**<br>**Log( 7.3890560989306504 )** is approximately **2.0** |
| **Max( x, y )** | larger value of *x* and *y*<br>(also has versions for **float**, **int** and **long** values) | **Max( 2.3, 12.7 )** is **12.7**<br>**Max( -2.3, -12.7 )** is **-2.3** |
| **Min( x, y )** | smaller value of *x* and *y*<br>(also has versions for **float**, **int** and **long** values) | **Min( 2.3, 12.7 )** is **2.3**<br>**Min( -2.3, -12.7 )** is **-12.7** |
| **Pow( x, y )** | *x* raised to power *y* (xy) | **Pow( 2.0, 7.0 )** is **128.0**<br>**Pow( 9.0, .5 )** is **3.0** |
| **Sin( x )** | trigonometric sine of *x* (*x* in radians) | **Sin( 0.0 )** is **0.0** |
| **Sqrt( x )** | square root of *x* | **Sqrt( 900.0 )** is **30.0**<br>**Sqrt( 9.0 )** is **3.0** |
| **Tan( x )** | trigonometric tangent of *x* (*x* in radians) | **Tan( 0.0 )** is **0.0** |

# Method 정의와 호출

- 접근 지정자
  - 함수를 호출할 수 있는 범위를 지정한다.
- 반환형
  - 함수가 돌려주는 값의 형식을 지정한다.
- 함수이름
  - 함수를 호출할 때 사용하는 함수의 이름
- 인자 리스트
  - 인자 리스트란 함수에게 실행 전에 전달되는 변수를 의미하며, 변수의 형식과 실제 값이 전달된다.
- 명령문
  - 실제 함수가 실행하는 문장
- 반환 값
  - 함수가 종료될 때 되돌려주는 값으로 위에 선언한 리턴형에 맞는 형식의 값이어야 한다. 리턴 값은 없을 수도 있으며 이때의 리턴형은 void로 선언한다

# Method Declaration

- 메소드 선언

```
class MyClass
{
    ...
    static int SquareSum( int num1, int num2 )
    {
        int sum = num1 + num2;
        return sum * sum;
    }
```

properties / return type / method name / parameter list



# Method Call

- static method (정적 메소드) 호출

같은 클래스 내에서의 static method 호출 :

```
메소드명();
```

다른 클래스 내에서의 static method 호출 :

```
클래스명.메소드명();
```

## Method Call

- instance method (인스턴스 메소드) 호출

같은 클래스 내에서의 instance method 호출 :

**메소드명();**

다른 클래스 내에서의 instance method 호출 :

**인스턴스명.메소드명();**



## Method 정의와 호출

- 메소드 호출 예제

```
namespace MethodExample {
        public class A {
                public void MethodC() {
                        Console.WriteLine("MethodC() in class A");
                }
                public static void MethodA() {
                        Console.WriteLine("MethodA() in class A");
                }
        }
        public class B {
                public static void Main(string[] args) {
                        A.MethodA();
                        A a = new A();
                        a.MethodC();
                }
        }
}
```

```
MethodA() in class A
MethodC() in class A
콘솔 창에 출력
```

## Nested Method

- 메소드를 호출할 경우, 호출된 메소드 내에서 또 다른 메소드를 계속 호출해서 사용하는 메소드
- 중첩 메소드를 사용할 때는 메소드끼리 서로를 계속 호출하여, 프로그램이 종료되지 않는 무한루프에 빠지지 않도록 주의할 것



## Nested Method

- 중첩 메소스 예제

```
using System;
namespace NestedMethodExample
{
        public class NestedMethod       {
                public static void MethodA() {
                        Console.WriteLine("MethodA.");
                }
                public static void MethodB()       {
                        MethodA();
                        Console.WriteLine("MethodB.");
                        MethodA();
                }
                public static void Main(string[] args) {
                        MethodB();
                        MethodA();
                }
        }
}
```

```
MethodA.
MethodB.
MethodA.
MethodA.
```

## Recursive Call

- 재귀 메소드
  - 자기 자신을 호출하는 메소드로서 같은 반복된 작업이 필요한 경우를 구현한 메소드

(예제) 재귀메소드를 이용하여 N!(factorial) 구하기

N-팩토리얼 = N * (N-1) * (N-2) * (N-3) * ..... * 1

## Nested Method

- 재귀 메소스 예제

```
using System;
namespace RecursiveCallExample
{
        public class RecursiveCall  {
                public static ulong Factorial(ulong number) {
                        if (number <= 1)
                                return 1;
                        else
                                return number * Factorial(number – 1);
                }
                public static void Main(string[] args) {
                        ulong nfact = Factorial(5);
                        Console.WriteLine("5 * 4 * 3 * 2 * 1 = " + nfact);
                }
        }
}
```

5 * 4 * 3 * 2 * 1 = 120

## Local Variables

- 지역변수란 메소스 내에서 선언된 변수
- 메소드가 실행될 때 변수를 저장하기 위한 메모리가 생성
- 선언된 메소드 내부에서만 사용이 가능
- 메소드의 실행이 종료될 때 메모리가 해제
- 변수를 선언한 후, 초기 값을 부여하는 초기화가 반드시 필요함



## Class Variables

- 클래스 변수란 클래스 내에 하나만 존재하며, 클래스 내의 모든 개체가 사용할 수 있는 변수
- 클래스가 로딩될 때 변수를 저장하기 위한 메모리를 생성
- 클래스 이름을 이용하여 접근함
- 모든 클래스 내의 인스턴스(개체)가 공유함
- 값을 지정하지 않은 경우에 자동으로 0으로 초기화함

## Variables

- 지역변수 (local variable)과 클래스변수 (class variable 또는 static variable)

```
class MyClass {
    public static int value = 10;
    public static int  SquareSum( int num1, int num2 )  {
        int sum = num1 + num2; // 지역변수 sum은 초기화해서 사용해야 함
        return sum * sum;
    }
}
class Program {
   public static void Main(string[] args) {
        int a = 0; // 지역변수 a는 초기화해서 사용해야 함
        a = MyClass.SquareSum(2, 3);
        Console.WriteLine("SquareSum = " + a);
        a += MyClass.value; // 클래스변수 value는 클래스명.변수명으로 사용
        Console.WriteLine("a = " + a);
   }
}
```



## Array

- 배열 (array)이란 같은 형식의 데이터를 그룹화해서 사용하거나 편집할 때 유용하게 사용할 수 있도록 하는 데이터 타입
  - System.Array라는 클래스에서 파생되었으며 배열 (array)의 모든 element 는 형(type)이 같아야 함
  - 구조체 (structure)란 서로 연관성이 있는 데이터이지만 형식 (type)이 다른 경우의 그룹화에 사용
  - 배열 (array)는 서로 연관성이 있고 형식도 같은 데이터를 그룹화 할 때 사용

## Array

- 배열의 특징
  - 같은 데이터형의 변수를 한꺼번에 여러 개 생성
  - 배열의 크기는 배열의 첨자로 결정
  - 첨자에 해당하는 만큼의 같은 데이터 형을 가진 메모리 생성
  - 배열의 메모리는 연속적으로 지정
  - 배열의 참조 값을 이용하여 핸들 할 수 있음
  - 배열의 이름은 연속된 변수들을 참조하기 위한 참조 값
  - 배열의 요소는 변수

## Array

- 배열 선언

```
type [ ] name;
```

  - type – 배열을 실제로 구성하는 요소의 형식(type)을 나타냄
  - [] – 배열의 차원 (rank)를 나타냄
  - name – 배열변수의 이름을 나타남

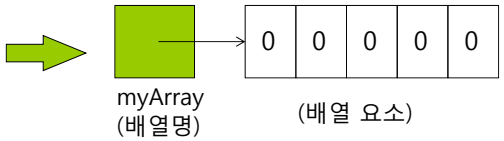
```
//선언
int[] myArray;

//초기화
myArray = new int[5];
```

## Array

- 배열의 초기화

| 타입 | 초기 값 |
|---|---|
| 숫자(int,long,float) 등 | 0 |
| 문자(char) | Null(빈 값을 의미) |
| 문자열(string) | Null(빈 값을 의미) |
| enum | 0 |
| 참조형(reference) | Null(빈 값을 의미) |

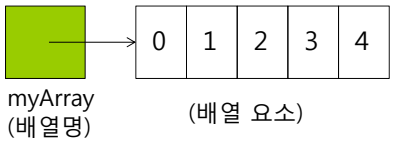
```
int[] myArray;
myArray = new int[5];
```




## Array

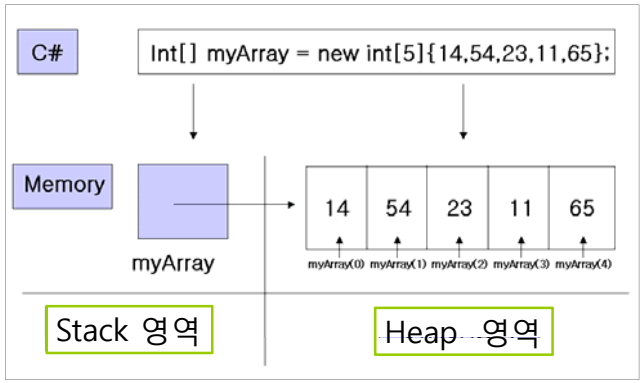
- 배열의 요소 값 지정하는 방법

```
int[] myArray = new int[5]{0,1,2,3,4};
int[] myArray = {0,1,2,3,4};
int[] myArray;
myArray = new int[5]{0,1,2,3,4};
```




## Array

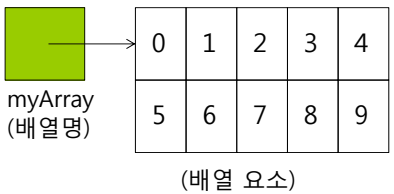
- 배열의 초기화 후의 모습
  - int형의 5개의 요소를 가진 배열 myArray가 정상적으로 초기화가 되면, 메모리에 배열이 저장되는데, 배열명은 Stack에, 각 요소들의 값은 Heap에 저장이 됨




## 다차원 Array

- 다차원 배열 선언

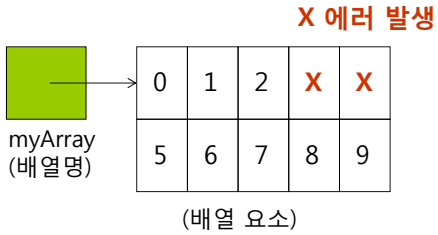
```
// 2차원 배열의 예

int[,] myArray = new int [2,5]{
        {0,1,2,3,4},
        {5,6,7,8,9}
};
```



## 다차원 Array

- 다차원 배열 선언시의 에러 예제

```
int[,] myArray = new int[2,5]{
        {0,1,2,},
        {5,6,7,8,9}
};
```

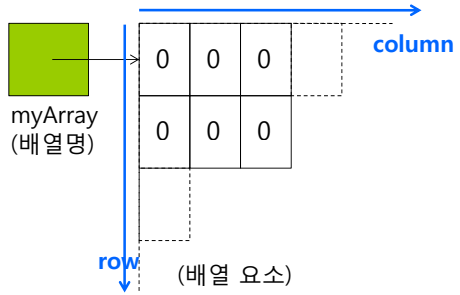



## 다차원 Array

- 2차원 배열의 크기

```
int[,] myArray = new int[rows, columns];
```

  - Rows * columns 만큼의 배열 크기가 생성됨
  - 실제 사용할 배열의 크기보다 훨씬 큰 배열을 선언하고 사용한다면, 메모리 낭비임
  - 반대로 사용할 배열보다 선언한 배열의 크기가 작다면 배열의 크기가 가변적이지 않기 때문에 다시 다른 변수명으로 배열을 새로 선언하고 사용해야 함




## 불규칙적 Array

- 다차원 배열 vs. 불규칙적 배열
  - 다차원 배열의 경우는 반드시 값을 채워줘야 함
  - 불규칙적인 배열(Jagged Array, a.k.a Array of Arrays)은 [][] 형태로 해주어야 함

```
int[,] myArray = new int[2,5]{
        {0,1,2,}, // 에러 발생
        {5,6,7,8,9}
};
```

```
int[][] myArray = new int[2][];
myArray[0] = new int[] {0,1,2,};
myArray[1] = new int[] {5,6,7,8,9};
```

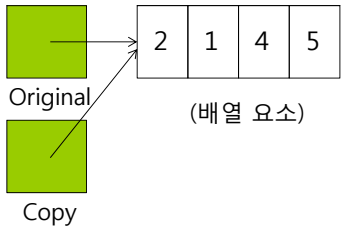
## Array의 복사

- 배열의 복사

```
class CopyArray
{
        static void Main (string [] args )
        {
                long [] Original = new long[4] { 2, 1, 4, 5};
                long [] Copy = Original;
                Console.WriteLine("1 : " + Copy[3]); // 1 : 5

                Original[3] = Original[0] + Original[2];
                Console.WriteLine("2 : " + CopyValue); // 2 : 6
        }
};
```

## Array의 복사

- 배열의 복사
  - 배열의 복사가 이루어지면, 복사된 배열명은 새로 배열을 생성하는 것이 아니라, 원래 있는 배열의 값들을 참조하고 있는 것
  - 그렇기 때문에 원래의 배열의 요소 값에 변화가 생기면 당연히 복사된 배열에서도 바뀐 값을 참조하는 것임




## Array의 활용

- 배열의 차원 (rank)

```
class ArrayTest
{
        static void Main(string[] args)
        {
                int[] Array1 = new int[4];
                int[,] Array2 = new int[2,3];
                int[,,] Array3 = new int[2,4,3];
                // 배열의 차원은 배열 선언시의 각 요소의 갯수
                // Array1의 차원 : 1
                Console.WriteLine("Array1의 차원 : " + Array1.Rank);
                // Array2의 차원 : 2
                Console.WriteLine("Array2의 차원 : " + Array2.Rank);
                // Array3의 차원 : 3
                Console.WriteLine("Array3의 차원 : " + Array3.Rank);
        }
}
```

## Array의 활용

- 배열의 크기 (length)

```
class ArrayTest
{
        static void Main(string[] args)
        {
                int[] Array1 = new int[4];
                int[,] Array2 = new int[2,3];
                int[,,] Array3 = new int[2,4,3];
                // 배열의 크기는 배열의 각 요소 크기의 곱셈
                // Array1의 크기 : 4
                Console.WriteLine("Array1의 크기 : " + Array1.Length);
                // Array2의 크기 : 6
                Console.WriteLine("Array2의 크기 : " + Array2.Length);
                // Array3의 크기 : 24
                Console.WriteLine("Array3의 크기 : " + Array3.Length);
        }
}
```

## Array의 활용

- 배열의 인덱스 (index)
  - 배열명[index]라고 쓰면 그 배열의 index 순서에 있는 element를 뜻하며, 첫번째 element 의 index는 0 임
  - 배열의 유효 index 범위를 넘는 인덱스를 사용하면 IndexOutOfRangeException 예외가 발생함

```
class ArrayTest
{
        static void Main(string[] args)
        {
                int[] Array1 = new int[4] {1, 2, 3, 4};
                for (int i = 0; i < Array1.Length; i++)
                {
                        Console.WriteLine("Array1[{0}]={1}", i, Array1[i]);
                }
        }
}
```

## Array의 활용

- 정렬 (Sort) 메소드 – System.Array.Sort()
  - 정렬 메소드는 배열의 요소값들을 크기의 순서대로 작은 순서부터 큰 순서대로 정렬을 해주고 이를 배열에 반영해주는 메소드
- 초기화 (Clear) 메소드 – System.Array.Clear()
  - 배열의 각 요소들의 값을 초기화하는 메소드
- 복제 (Clone) 메소드 – System.Array.Clone()
  - 배열의 크기와 요소값을 모두 같게하여 새로운 배열 생성하는 메소드
- 색인 (IndexOf) 메소드 – System.Array.IndexOf()
  - 찾으려는 값이 배열의 몇 번째 요소인지를 반환하는 메소드

## Array의 활용

- 복제 (Clone), 색인 (IndexOf), 정렬 (Sort) 메소드

```
int[] one = new int[] {2, 1, 4, 5};          // one {2, 1, 4, 5}
int[] clone = (int[])one.Clone();            // clone {2, 1, 4, 5}

one[3] = one[0] + one[2];                    // one {2, 1, 4, 6} clone {2, 1, 4, 5}
foreach(int i in clone) {
      Console.Write("{0} ", i);              // 2 1 4 5
}

int where = Array.IndexOf(clone, 4);         // 4 is located in 2
Console.WriteLine("₩n4 is located in {0}", where);

Array.Sort(clone);
Console.Write("After sort: ");
foreach(int i in clone) {
      Console.Write("{0} ", i);              // After sort: 1 2 4 5
}
```

## Array의 활용

- 메소드의 리턴 값으로의 배열

```
class ArrayReturn
{
    static void Main(string[] args)
    {
        int[] MyArray = CreateIntArray(10);
        // MyArray의 크기 : 10
        Console.WriteLine("MyArray의 크기 : " + MyArray.Length);
    }

    static int[] CreateIntArray(int size)
    {
        int[] intArray = new int[size];
        return intArray;
    }
}
```

## Array의 활용

- 메소드의 인자로서의 배열

```
class ArrayParam
{
        static void Main(string[] args)
        {
                int[] MyArray = {2,6,5,4,1};
                MyMethod(MyArray);
                // 3, 7, 6, 5, 2
                for(int i=0;i<MyArray.Length;i++)
                        Console.WriteLine (MyArray[i]);
        }
        static void MyMethod(int[] parameter)
        {
                for(int j=0;j<parameter.Length;j++)
                        parameter[j]++;
        }
}
```

## Array의 예제

```
int[] Array1 = new int[4] {2, 1, 4, 5};          // one-dimensional array
for (int i = 0; i < Array1.Length; i++)
    Console.WriteLine("Array1[{0}]={1}", i, Array1[i]);

int[,] Array2 = new int[2,3];                    // two-dimensional array
int a = 1;
for (int i = 0; i < Array2.GetLength(0); i++)
    for (int j = 0; j < Array2.GetLength(1); j++)
        Array2[i, j] = a++;
Array2[1, 2] = 7;

int[, ,] Array3 = new int[2, 4, 3] {             // three-dimensional array
                  { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 }, { 10, 11, 12 } },
                  { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 }, { 10, 11, 12 } }
                  };
for (int i = 0; i < Array3.GetLength(0); i++)
        for (int j = 0; j < Array3.GetLength(1); j++)
            for (int k = 0; k < Array3.GetLength(2); k++)
                Console.WriteLine("Array3[{0},{1},{2}]={3}", i, j, k, Array3[i, j, k]);
```

2 1 4 5

1 2 3
4 5 7

1 2 3
4 5 6
7 8 9
10 11 12

## Array의 예제

```
int[][] Array4 = new int[2][];                   // jagged array
Array4[0] = new int[] { 0, 1, 2 };
Array4[1] = new int[] { 5, 6, 7, 8, 9 };

for (int i = 0; i < Array4.GetLength(0); i++)
        for (int j = 0; j < Array4[i].GetLength(0); j++)
            Console.WriteLine("Array4[{0}][{1}]={2}", i, j, Array4[i][j]);

int[][] Array5 = new int[][] {                   // jagged array
                 new int[] {1, 2},
                 new int[] {3, 4, 5},
                 new int[] {6, 7, 8, 9},
                 new int[] {10, 11, 12}
                 };

for (int i = 0; i < Array5.Length; i++)
    for (int j = 0; j < Array5[i].Length; j++)
        Console.WriteLine("Array5[{0}][{1}]={2}", i, j, Array5[i][j]);
```

0 1 2
5 6 7 8 9

1 2
3 4 5
6 7 8 9
10 11 12