C#프로그래밍의 이해

코드의 흐름 제어하기

01. 분기문 (6/7)

- switch (1/2)
  - switch문의 사용 형식

```
switch (조건식)
{
    case 상수1:
        // 실행할 코드
        break;
    case 상수2:
        // 실행할 코드
        break;
    case 상수N:
        // 실행할 코드
        break;
    default:
        //
        break;
}
```

01. 분기문 (7/7)

- switch (2/2)
  - switch문의 사용 예

```
int number = 1;

switch ( number )
{
    case 1:
        Console.WriteLine("하나");
        break;
    case 2:
        Console.WriteLine("둘");
        break;
    case 3:
        Console.WriteLine("셋");
        break;
    default:
        Console.WriteLine("제가 아는 숫자는 1, 2, 3 뿐입니다.");
        break;
}
```

02. 반복문 (1/8)

- 반복문(Loop Statement)
  - 특정 조건을 만족하는 동안 코드 또는 코드 블록을 계속 반복해서 실행하도록 하는 문장
  - C#은 다음 네 가지의 반복문을 제공
    - while
    - do while
    - for
    - foreach

02. 반복문 (2/8)

- while
  - 형식

```
while ( 조건식 )
    반복실행할_코드
```

  - 사용 예 1

```
int a = 10;
while ( a > 0 )
    Console.WriteLine( a-- );
```

  - 사용 예 2

```
int a = 10;
while ( a > 0 )
{
    Console.WriteLine ( a );
    a -= 2;
}
```

02. 반복문 (3/8)

- do while
  - while과 유사하지만 while문이 조건식을 평가한 후에 그 결과가 참일 때에만 코드를 실행하는 데 반해, do while 문은 조건식을 평가하기 전에 무조건 첫 1회는 코드를 실행
  - 형식

```
do
{
    반복실행할_코드_블록
}
while( 조건식 ) ;
```

이 코드 블록은 최초 한 번은 무조건 실행

do while문은 세미콜론을 붙여야 함

  - 사용 예

```
int a = 10;
do
{
    Console.WriteLine ( a ); // 이 문장은 첫 한번에만 실행
    a -= 2;
}
while ( a > 10 ); // 조건 평가 결과가 거짓이 되므로 반복 종료
```

02. 반복문 (4/8)

- for (1/2)
  - 형식

```
for( 초기화식; 조건식; 반복식; )
    반복실행할_코드;
```

  - 초기화식
    - 반복을 실행하기 전에 가장 먼저, 딱 한 번만 실행되는 코드
    - for 반복문에서 사용할 변수 등을 이곳에서 초기화
  - 조건식
    - 반복을 계속 수행할지 여부를 결정하는 식
    - 이 조건식의 결과가 false가 되면 반복을 중단
  - 반복식
    - 매 반복이 끝날 때마다 실행, 주로 여기에서 조건식에서 사용하는 변수의 값을 조정

02. 반복문 (5/8)

- for (2/2)
  - 사용 예

```
for ( int i=0; i<5; i++ )
    Console.WriteLine(i);
```

실행 결과

```
0
1
2
3
4
```

02. 반복문 (6/8)

- 중첩 for

```
for ( int i=0; i<5; i++ )
{
    for ( int j=0; j<i; j++ )
    {
        Console.Write("*");
    }
    Console.WriteLine();
}
```

실행 결과

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

02. 반복문 (7/8)

- foreach
  - 배열(또는 컬렉션)을 순회하며 각 데이터 요소에 차례대로 접근
  - 형식

```
foreach( 데이터형식 변수명 in 배열_또는_컬렉션)
    코드_또는_코드블럭
```

  - 사용 예제

```
int[] arr = new int[] {0, 1, 2, 3, 4}; // 배열 선언
foreach (int a in arr)
{
    Console.WriteLine(a);
}
```

실행 결과

```
0
1
2
3
4
```

02. 반복문 (8/8)

- for 또는 while을 이용한 무한 반복 코드

```
for( ;; )
    // 반복 실행할 코드 블록
```

VS 취향에 따라 결정

```
while( true )
    // 반복 실행할 코드 블록
```

03. 점프문 (1/3)

- break
  - break는 이름(영어로' 탈출하다',' 중단하다'라는 뜻) 그대로 현재 실행 중인 반복문이나 switch 문의 실행을 중단하고자 할 때 사용

```
int i = 0;  // i를 초기화하는 코드가 실행되고

while ( i >= 0) // 반복이 실행되다가
{
    if ( i == 10 )
        break;     // i가 10이 되면 while 문에서 탈출

    Console.WriteLine( i++ );
}

// 프로그램의 실행 위치는 while 블록 다음으로 이동
Console.WriteLine("Prison Break");
```

03. 점프문 (2/3)

- continue
  - 반복문을 멈추게 하는 break와는 달리, continue 문은 한 회 건너 뛰어 반복을 계속 수행

```
for ( int i=0; i<5; i++ )
{
    if ( i == 3 )
        continue;
    Console.WriteLine( i );
}
```

가독성이 더 뛰어남

VS

```
for ( int i=0; i<5; i++ )
{
    if ( i != 3 )
    {
        Console.WriteLine( i );
    }
}
```

03. 점프문 (3/3)

- goto
  - 지정한 레이블로 바로 코드의 실행을 이동하는 점프문
  - 형식

```
goto 레이블;

레이블 :
   // 이어지는 코드
```

가독성이 더 뛰어남

  - 사용 예제

```
{
    Console.WriteLine( " 1 " );
    goto JUMP;
    Console.WriteLine( " 2 " );
    Console.WriteLine( " 3 " );

JUMP:
    Console.WriteLine( " 4" );
}
```

실행 결과 :

1
4

메소드

코드의 흐름 제어하기

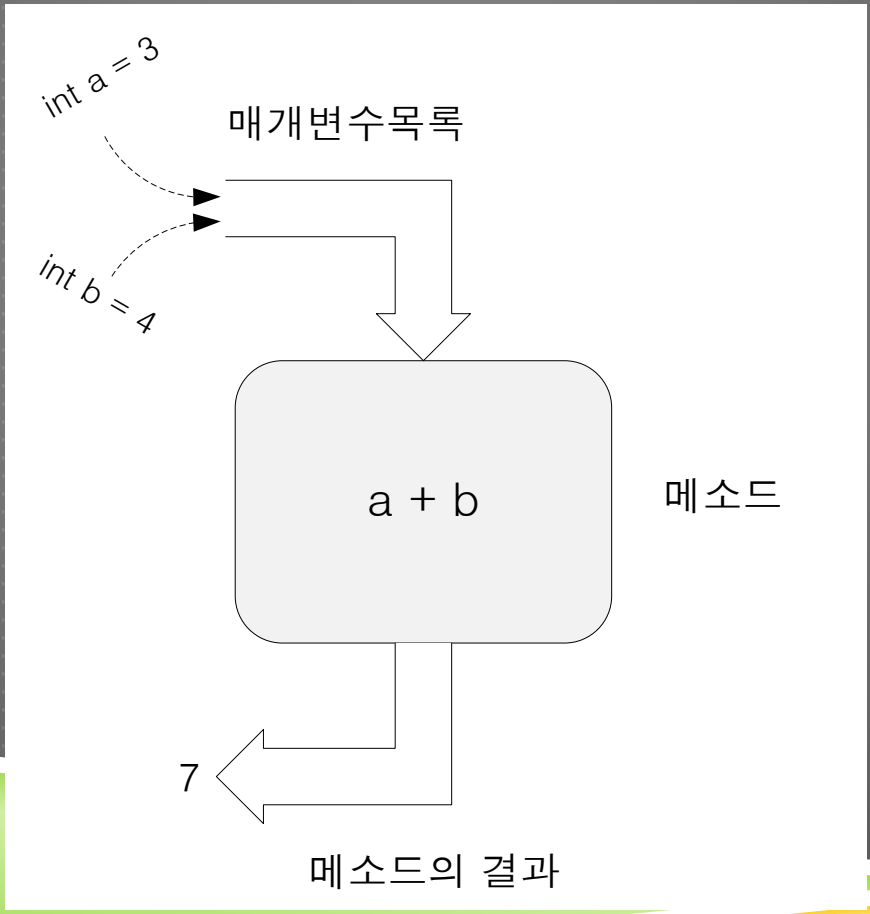
01. 메소드란? (1/4)

- 메소드
  - 일련의 코드를 하나의 이름 아래 묶은 것
  - OOP적인 의미에서는 객체의 데이터를 처리하는 방법을 추상화한 것
  - 묶은 코드는 메소드의 이름을 불러주는 것만으로 실행
  - C/C++에서는 함수(Function), 파스칼에서는 프로시져(Procedure) 등으로 부름
- 메소드 선언 형식

```
class 클래스의_이름
{
    한정자 반환_형식 메소드의_이름( 매개_변수_목록 )
    {
        // 실행하고자 하는 코드 1
        // 실행하고자 하는 코드 2
        // …
        // 실행하고자 하는 코드 n
        return 메소드의_결과;
    }
}
```

01. 메소드란? (2/4)

- 메소드 호출 과정



01. 메소드란? (3/4)

- 메소드의 선언 예

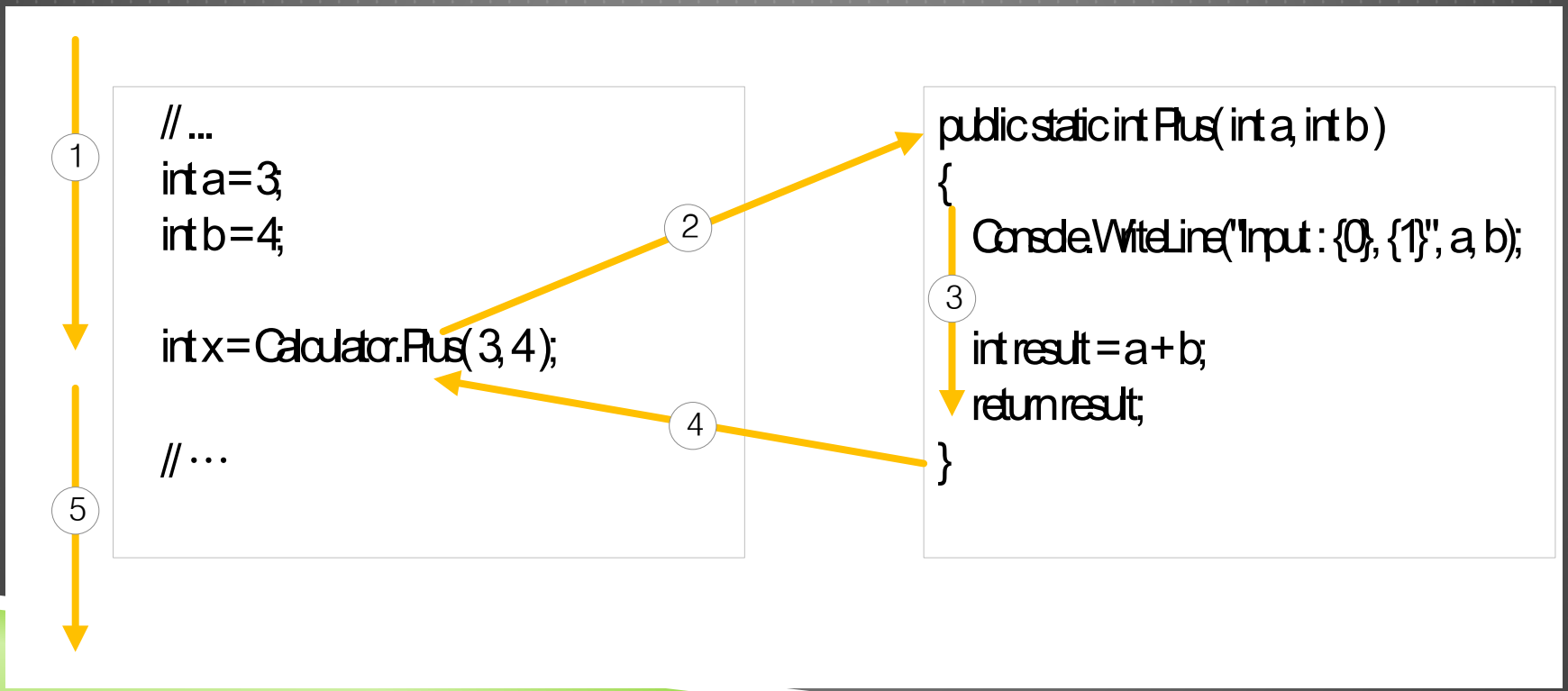
```
class Calculator
{
    public static int Plus( int a, int b )
    {
        Console.WriteLine("Input : {0}, {1}", a, b);
        int result = a + b;
        return result;
    }
}
```

- 메소드 호출의 예

```
int x = Calculator.Plus( 3, 4 ); //x는 7
int y = Calculator.Plus( 5, 6 ); //y는 11
int z = Calculator.Plus( 7, 8 ); //y는 15
```

01. 메소드란? (4/4)

- 메소드 호출시에 일어나는 프로그램 흐름의 변화



02. RETURN에 대하여 (1/2)

- return 문 (1/2)
  - 점프문의 한 종류
  - 프로그램의 흐름을 갑자기 호출자에게로 돌림
  - return 문은 언제든지 메소드 중간에 호출되어 메소드를 종결시키고 프로그램의 흐름을 호출자에게 돌려줄 수 있음

```
int Fibonacci( int n )
{
    if (n < 2)
        return n;
    else
        return Fibonacci(n-1) + Fibonacci(n-2);
}
```

재귀호출 : 메소드 내부에서 스스로를 다시 호출

02. RETURN에 대하여 (2/2)

- return 문 (2/2)
  - 메소드가 반환할 것이 아무것도 없는 경우(즉, 반환 형식이 void인 경우)에도 return 문 사용 가능
  - 이 경우 return문은 어떤 값도 반환하지 않고 메소드만 종료시킴

```
void PrintProfile(string name, string phone)
{
    if (name == "")
    {
        Console.WriteLine("이름을 입력해주세요.");
        return;
    }
    Console.WriteLine( "Name:{0}, Phone:{1}", name, phone );
}
```

03. 매개 변수에 대하여 (1/3)

- 메소드를 호출할 때 넘기는 매개 변수는 그대로 메소드 안으로 넘겨지는 것일까? : 답은 "아니다."

메소드 선언부의 매개 변수

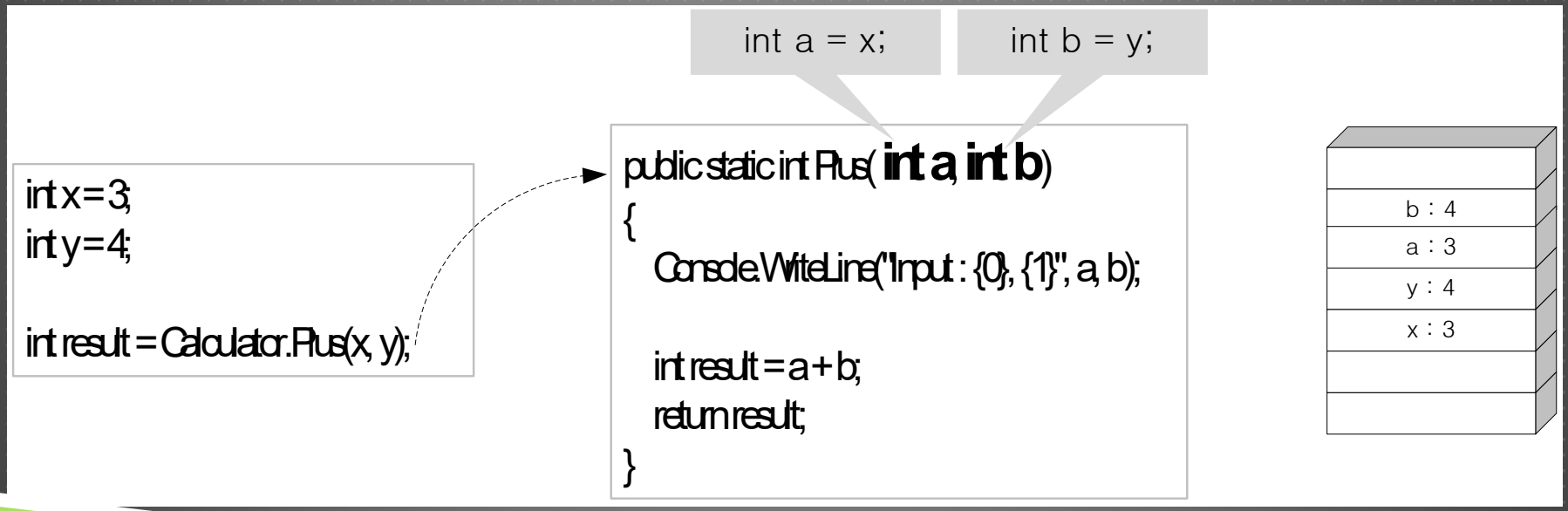
```
class Calculator
{
    public static int Plus( int a, int b )
    {
        Console.WriteLine("Input : {0}, {1}", a, b);
        int result = a + b;
        return result;
    }
    // …
}
```

메소드 호출부의 매개 변수

```
class MainApp
{
    public static void Main()
    {
        int x = 3;
        int y = 4;
        int result = Calculator.Plus( x, y );
        // …
    }
}
```

03. 매개 변수에 대하여 (2/3)

- 매개 변수도 메소드 외부에서 메소드 내부로 데이터를 전달하는 매개체 역할을 할 뿐, 근본적으로는 "변수"
  - 한 변수를 또 다른 변수에 할당하면 변수가 담고 있는 데이터만 복사됨.



03. 매개 변수에 대하여 (3/3)

- 예제 : 두 매개 변수의 값을 교환하는 Swap() 메소드

```
public static void Swap(int a, int b)
{
    int temp = b;
    b = a;
    a = temp;
}
```

```
static void Main(string[] args)
{
    int x = 3;
    int y = 4;

    Swap(x, y);
}
```

Swap() 함수를 호출하고난 뒤에도 x와 y의 값은 변하지 않는다.