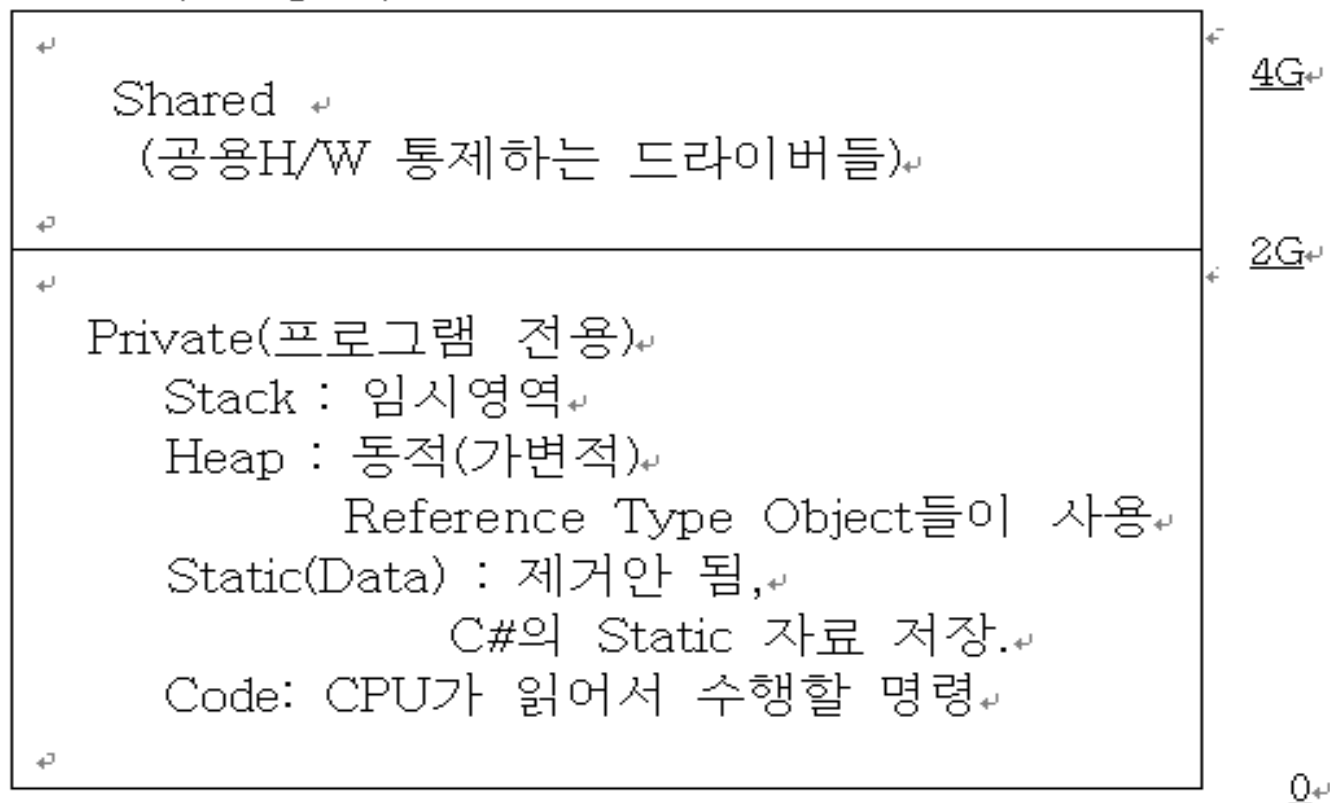


프로세스 할당 구조

** win32 Memory Map

프로세스 당 최고 => 4GB



정적함수 또는 정적변수는 프로세스가 실행되면서 메모리에 할당된다.

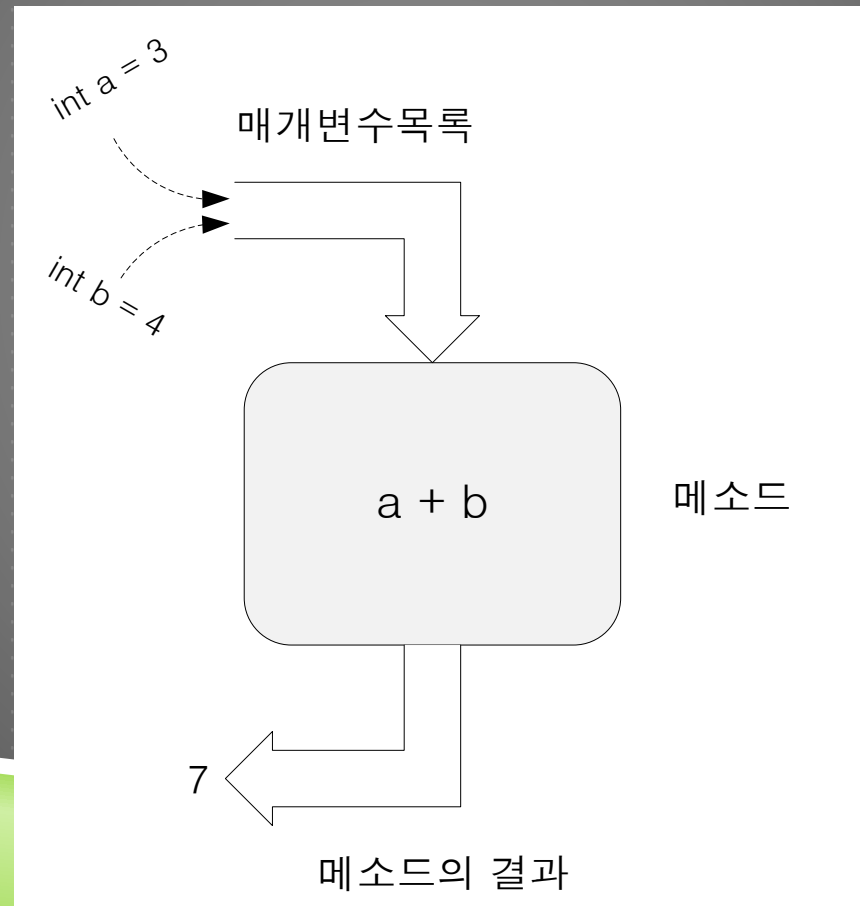
01. 메소드란? (1/4)

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

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

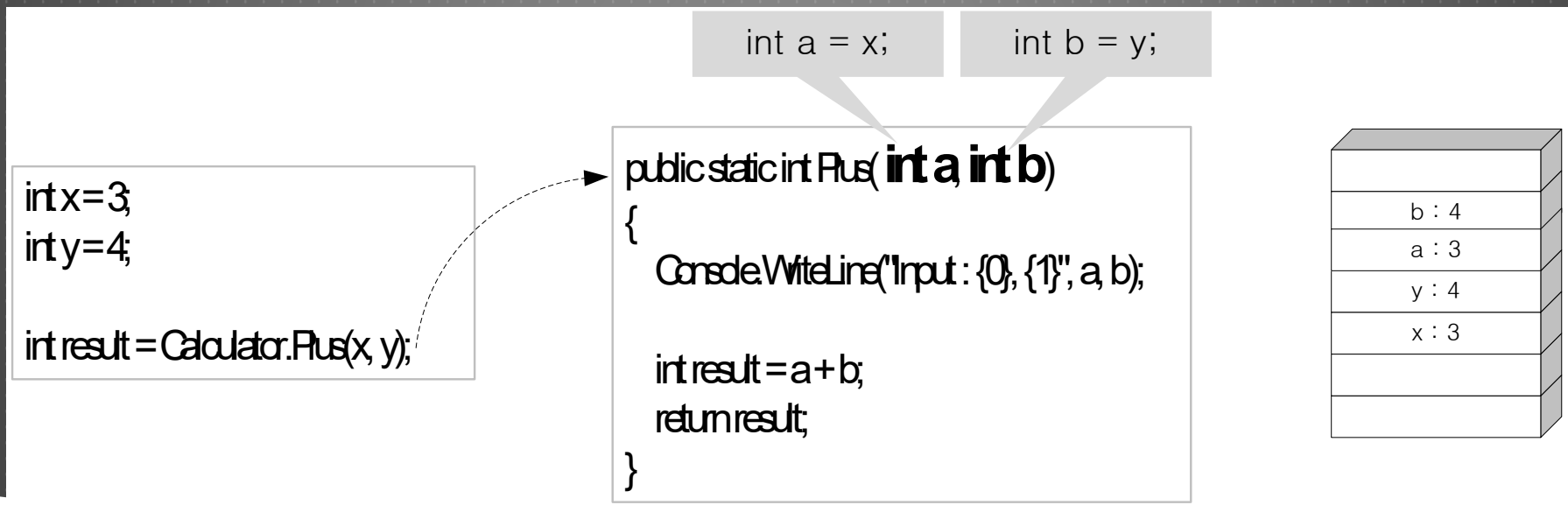
01. 메소드란? (2/4)

▶ 메소드 호출 과정



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.



04. 파라미터 전달 구분

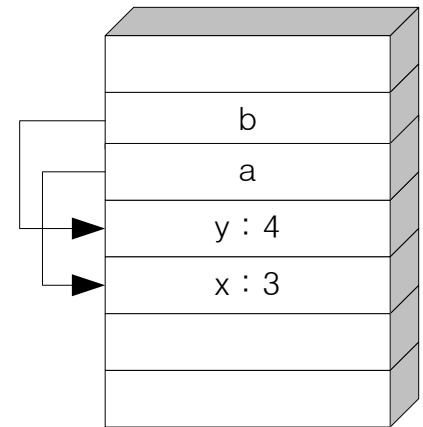
- ▶ **in**(pass by value) : 값을 전달, 값을 복사하여 전달. (기본값)
- ▶ **in, out** (pass by reference): 메모리 위치에 대한 참조(주소를 전달), **ref** 필요
실체는 하나인데 이름만(별명) 하나 덧 붙인 것, **alias** 이다.
(C++ 에 &인 참조연산자)
- ▶ **out** : 파라미터를 통해 결과를 얻음, 기존 값을 얻어올 수 없다.
out 필요
여러 개의 인자를 넘길 때 유용하다.
여러 개의 값을 얻어오고자 할 때 사용.

04. 참조에 의한 매개 변수 전달

- ▶ Swap() 메소드가 참조로 매개 변수를 전달할 때의 과정

```
int x=3;  
int y=4;  
  
Swap(ref x, ref y);
```

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



05. 출력 전용 매개 변수 (2/2)

▶ 출력 전용 매개 변수의 선언 및 사용 예

```
void Divide( int a, int b, out int quotient, out int remainder )  
{  
    quotient = a / b;  
    remainder = a % b;  
}
```

```
int a = 20;  
int b = 3;  
int c;  
int d;
```

```
Divide( a, b, out c, out d );  
Console.WriteLine("Quotient : {0}, Remainder {1}", c, d );
```



06. 메소드 오버로딩

- ▶ Overloading : 과적하다
- ▶ 메소드 오버로딩 : 하나의 메소드 이름에 여러 개의 구현을 올림

```
int Plus(int a, int b)
```

```
{  
    return a + b;  
}
```

```
double Plus(double a, double b)
```

```
{  
    return a + b;  
}
```

```
int result1 = Plus( 1, 2 );
```

```
double result2 = Plus( 3.1, 2.4 );
```

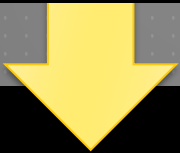
호출

호출

07. 가변길이 매개 변수 (1/2)

- ▶ 가변길이 매개변수 : 개수가 유연하게 변할 수 있는 매개 변수
- ▶ 다음 코드에서 호출하는 Sum() 메소드는 가변길이 매개 변수를 이용하여 단 하나만 구현(오버로딩을 이용하지 않음)

```
int total = 0;  
  
total = Sum( 1, 2 );  
total = Sum( 1, 2, 3 );  
total = Sum( 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 );
```



```
int Sum( params int[] args )  
{  
    int sum = 0;  
    for(int i=0; i<args.Length; i++)  
    {  
        sum += args[i];  
    }  
    return sum;  
}
```

가변길이 매개변수는 **params**
키워드와 **배열**을 이용하여 선언

07. 가변길이 매개 변수 (2/2)

- ▶ 메소드 오버로딩 vs 가변길이 매개 변수
 - ▶ 메소드 오버로딩
 - 매개 변수의 개수가 유한하게 정해져 있을 때 사용
 - 매개 변수의 각 형식이 다를 때 사용
 - ▶ 가변길이 매개 변수
 - 형식은 같으나 매개 변수의 개수만 유연하게 달라질 수 있는 경우에 사용

09. 선택적 매개 변수

- ▶ 메소드 선언시 매개 변수에 기본 값을 할당함으로써, 해당 매개 변수에 명시적으로 값을 할당할지/않을지를 선택가능하게 하는 기능
- ▶ 선택적 매개 변수를 가지는 메소드 선언의 예

```
void MyMethod( int a = 0, int b = 0 )  
{  
    Console.WriteLine( "{0},{1}", a, b );  
}
```

b는 선택적 매개 변수

- ▶ 선택적 매개 변수의 호출 예

```
MyMethod(3);    // 매개변수 b 생략  
MyMethod(3, 4);
```

08. 명명된 매개 변수

- ▶ 메소드를 호출할 때 매개 변수 목록 중 어느 매개 변수에 데이터를 할당할 것인지를 지정하는 것은 "순서"
- ▶ 명명된 매개 변수(Named Parameter)는 메소드를 호출할 때 매개 변수의 이름을 명시함으로써 순서에 관계없이 매개 변수에 할당할 데이터를 바인드하는 기능

```
static void PrintProfile( string name, string phone)
{
    Console.WriteLine("Name:{0}, Phone:{1}", name, phone);
}
static void Main(string[] args)
{
    PrintProfile( name : "박찬호", phone : "010-123-1234");
}
```

01. 배열(1차원 배열)

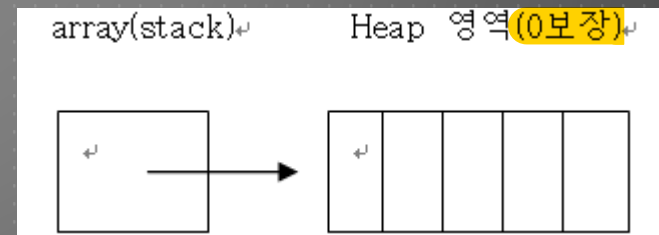
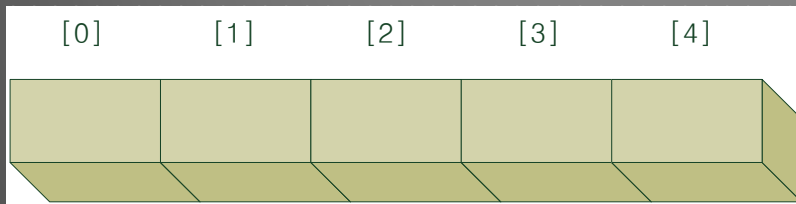
- ▶ 하나의 이름으로 참조되는 같은 자료형을 같은 데이터들의 집합

- ▶ 선언 형식

```
데이터형식[ ] 배열이름 = new 데이터형식[ 용량 ];
```

- ▶ 배열 선언 예

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



**** Array ,배열 **** System.Array 클래스의 파생 Class

heap에 저장된다. Array class 이다, 따라서 참조형.

Heap : reference type의 Object (class, interface, delegate)

Stack : 지역변수와 value type 인 struct, enum 할당

개요: 원소가 연속적으로 메모리에 할당된 구조, 모든 원소는 같은 타입의 집합.
구조체는 다른 타입의 원소를 가질 수 있다.

배열은 System.Array의 파생클래스이므로 속성,메서드,이벤트를 가질 수 있다.

사용법 : `int[] age= new int[10];` //배열 크기는 new 할 때 주어야 한다.
`int[] x;` //배열인 []는 항상 왼쪽에 와야 한다고 C#이 규정
`int x[3];` -> error이다.
`ArrayList list= new ArrayList();`

**** 배열 선언과 할당**

`int[] age ;` //1차원배열 선언, 배열을 할당하기 위한 참조변수만 할당,
`age =new int[5];` //heap 에 메모리 할당(초기값, 0을 보장,
C#의 CLR이 0을 채움)

`int[,] age ;` //2차원배열 선언
`age = new int[3,4];`

02. 배열을 초기화하는 방법 세 가지

```
int[] x = new int[5];  
    //x 는 생성된 intance 의 선두주소를 갖는다.  
int[] y = new int[-5];  
    //error , 배열의 크기는 반드시 양수  
int[] z = new int[];  
    //error , 생성할 때는 반드시 크기를 지정
```

```
string[] array1 = new string[3]{ "안녕", "Hello", "Halo" };
```

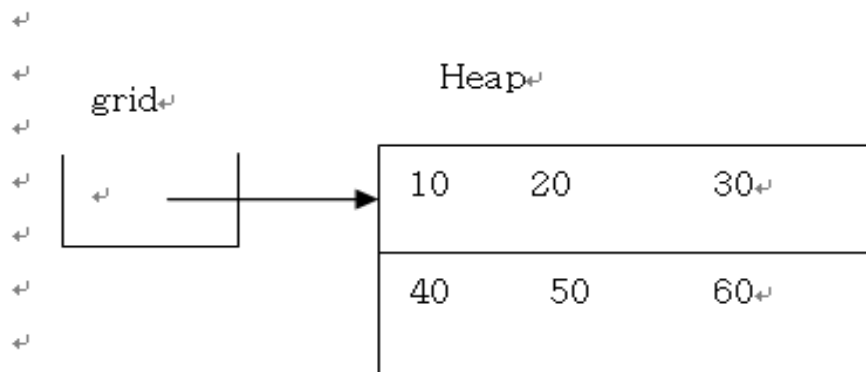
```
string[] array2 = new string[] { "안녕", "Hello", "Halo" };
```

```
string[] array3 = { "안녕", "Hello", "Halo" };
```

04. 다차원 배열

** 다차원 배열 ↵

```
int grid[,] = new int [2,3];
```



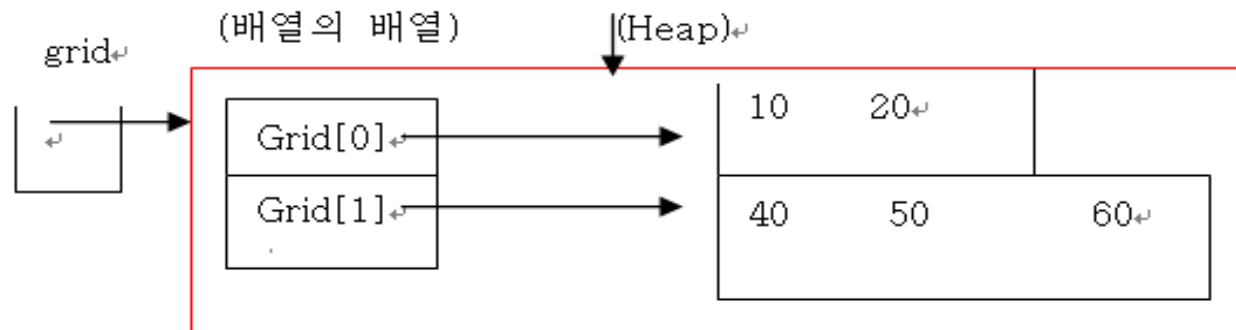
배열 변수는 Stack에 할당, 변수가 가리키는 할당된 영역은 heap이다.

2. 배열의 배열

단점 : 문법 복잡하고 속도가 느리다. ⁴

장점 : 배열의 열의 길이가 다른 크기를 할당하고 메모리를 절약할 수 있다.

```
int[][] grid = new int [2]
grid = new int[2];
grid = new int[3];
```



03. 알아 두면 삶이 윤택해지는 SYSTEM.ARRAY

▶ System.Array 클래스의 주요 메소드

분류	이름	설명
정적 메소드	Sort()	배열을 정렬
	BinarySearch<T>()	이진 탐색을 수행
	IndexOf()	배열에서 찾고자 하는 특정 데이터의 인덱스를 반환
	TrueForAll<T>()	배열의 모든 요소가 지정한 조건에 부합하는지의 여부를 반환
	FindIndex<T>()	배열에서 지정한 조건에 부합하는 첫 번째 요소의 인덱스를 반환. IndexOf() 메소드가 특정 값을 찾는 데 비해, FindIndex<T>() 메소드는 지정한 조건에 바탕하여 값을 찾음.
	Resize<T>()	배열의 크기를 재조정
	Clear()	배열의 모든 요소를 초기화
인스턴스 메소드	ForEach<T>()	배열의 모든 요소에 대해 동일한 작업을 수행하게 함
	GetLength()	배열에서 지정한 차원의 길이를 반환. 다차원 배열에서 유용하게 사용
프로퍼티	Length	배열의 길이를 반환
	Rank	배열의 차원을 반환

07. 컬렉션(COLLECTION)

- ▶ 컬렉션이란? (**Collection(Data를 모아서 처리하는 것)**)

- ▶ 같은 성격을 띄는 데이터의 모음을 담는 자료 구조
- ▶ 배열도 .NET 프레임워크가 제공하는 컬렉션 자료구조 중 하나

- ▶ ArrayList

추가,제거가 자유롭고 자료형이 달라도 된다. 갯수의 제한이없다.
ObjectType으로 변환하여 사용하므로 속도 느리다.

Array 은 고정된 크기를 갖는다.

```
ArrayList list = new ArrayList();  
list.Add( 10 );  
list.Add( 20 );  
list.Add( 30 );
```

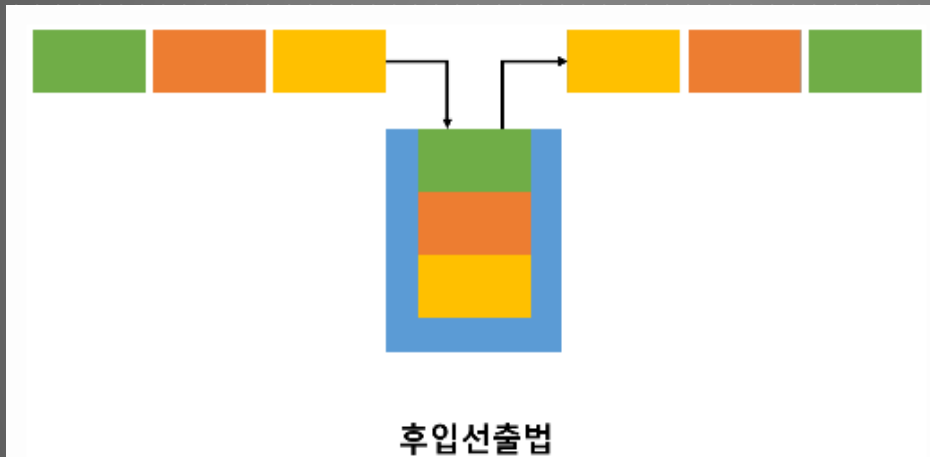
```
list.RemoveAt( 1 ); // 20을 삭제
```

```
list.Insert( 25, 1 ); // 25를 1번 인덱스에 삽입. 즉, 10 과 30 사이에 25를 삽입
```

07. 컬렉션 맛보기 (4/5)

▶ Stack

- ▶ 선입후출(Firs In Last Out) 구조의 자료구조

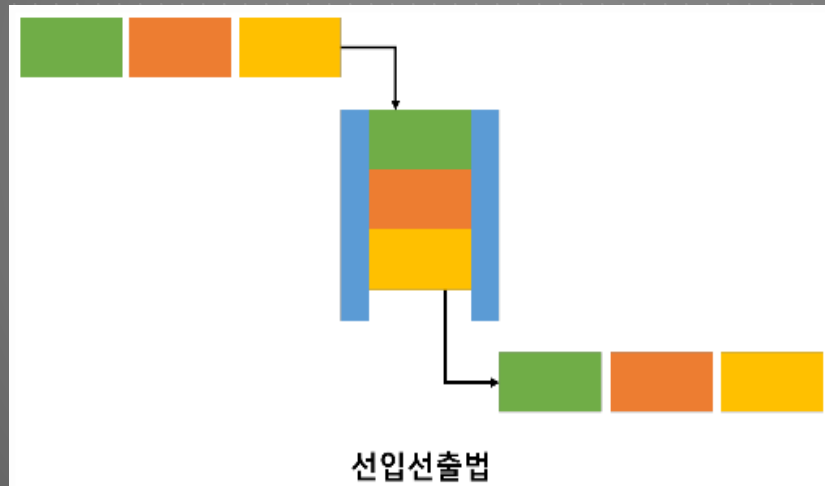


```
Stack stack = new Stack();  
stack.Push( 1 );// 최상위 데이터는 1  
stack.Push( 2 );// 최상위 데이터는 2  
stack.Push( 3 );// 최상위 데이터는 3  
  
int a = Push.Pop();// 최상위 데이터는 다시 2
```

07. 컬렉션 맛보기 (2/5)

▶ Queue (1/2)

- ▶ 대기열, 선입선출(First In First Out) 구조의 자료 구조
- ▶ Enqueue : 대기행렬에 데이터를 입력하는 연산
- ▶ Dequeue : 대기행렬에서 데이터를 출력하는 연산



```
Queue que = new Queue();  
que.Enqueue( 1 );  
que.Enqueue( 2 );  
int a = que.Dequeue( );
```

07. 컬렉션 맛보기 (5/5)

▶ Hashtable

- ▶ 키(Key)와 값(Value)으로 이루어진 데이터를 다룰 때 사용.
- ▶ 키를 해싱(Hashing)을 통해 테이블 내의 주소를 계산.
- ▶ 다루기 간편하고 탐색속도도 빠름

```
Hashtable ht = new Hashtable();  
ht["book"] = "책";  
ht["cook"] = "요리사";  
ht["tweet"] = "지저귀다";  
  
Console.WriteLine( ht["book"] );  
Console.WriteLine( ht["cook"] );  
Console.WriteLine( ht["tweet"] );
```

STRING 클래스

- ▶ 문자열을 저장하기 위한 클래스이다. `System.String` 클래스로 `System.Object` 의 하위 클래스이다. 자주 사용되는 문자열 메서드를 확인한다.

1. System.String 클래스의 인스턴스 메서드

1. Clone	클래스 참조 반환
CompareTo	특정 객체와 비교
CopyTo	객체 복사
EndsWith	특정 문자열로 끝나는지를 확인
Equals	비교 연산
GetEnumerator	IEnumerator 인터페이스 반환
GetHashCode	해시 코드 반환
GetType	형식 정보 반환

GetTypeCode	TypeCode 반환
IndexOf	문자열 검색
IndexOfAny	유니코드 문자열에서 먼저 나오는 문자 반환
Insert	문자열 삽입
LastIndexOf	IndexOf를 뒤에서부터 수행
LastIndexOfAny	IndexOfAny를 뒤에서부터 수행
PadLeft	문자열에서 남아있는 왼쪽을 빈 공백으로 채움
PadRight	문자열에서 남아있는 오른쪽을 빈 공백으로 채움
Remove	지정 개수의 문자 제거
Replace	문자열 치환

STRING 클래스

- ▶ 문자열을 저장하기 위한 클래스이다. `System.String` 클래스로 `System.Object` 의 하위 클래스이다. 자주 사용되는 문자열 메서드를 확인한다.

<code>Split</code>	문자열 분리하여 배열로 반환
<code>StartsWith</code>	특정 문자로 시작하는지를 확인
<code>Substring</code>	문자열 추출
<code>ToCharArray</code>	문자 배열로 변환
<code>ToLower</code>	소문자로 변환
<code>ToString</code>	객체를 나타내는 문자열 반환
<code>ToUpper</code>	대문자로 변환
<code>Trim</code>	양쪽 공백 없앰
<code>TrimEnd</code>	문자열 끝 부분의 공백 없앰
<code>TrimStart</code>	문자열 시작 부분의