변수 타입

자료형

자료형	상세	표현 범위
int	부호있는 4byte 범위 정수	-2147483648 ~ 2147483647 (-21억 ~ 21억)
char	2byte 유니코드 문자	0000 ~ FFFF
float	4byte 부동 소수점	
double	8byte 부동 소수점	
bool	1byte	true, false
string		문자 여러 개 표현가능

전체 자료형

- 정수 자료형

자료형	크기	범위
sbyte	부호있는 1byte 범위 정수	-128 ~ 127
byte	부호없는 1byte 범위 정수	0 ~ 255
short	부호있는 2byte 범위 정수	-32768 ~ 32767
ushort	부호없는 2byte 범위 정수	0 ~ 65535
int	부호있는 4byte 범위 정수	-2147483648 ~ 2147483647 (-21억 ~ 21억)
uint	부호없는 4byte 범위 정수	0 ~ 4294967295 (~ 42억)
long	부호있는 8byte 범위 정수	-9223372036854775808 ~ ~9223372036854775807
ulong	부호없는 8byte 범위 정수	0~18446744073709551615
char	2byte 유니코드 문자	0000~FFFF

- 실수 자료형

자료형	크기	범위
float	4byte 부동 소수점 수	±1.5×10^-45~±3.4×1038
double	8byte 부동 소수점 수 (float형 보다 정밀)	±5.0×10^-324±1.7×10308
decimal	16byte 부동 소수점 수 (double형 보다 정밀)	1.0×10^-28~약7.9×10^28

- 논리형

자료형	크기	범위
bool	1byte	true, false

- 문자형

자료형	크기	범위	표현방법
char	유니코드 2byte	0000 ~ FFFF	' ' (문자 1개만 표현 가능)
string			" " (문자 여러 개 표현 가능)

변수 변환

명시적

```
sbyte value1 = 64;
sbyte value2 = 64;
int value3 = (int)value1 + value2;
Console.WriteLine(value3);
```

암시적

```
sbyte value1 = 64;
sbyte value2 = 64;
int value3 = value1 + value2;
Console.WriteLine(value3);
```

연산자

연산자	사용법	
증가 연산자	num++	
감소 연산자	num--	
논리곱(&&, AND)	A && B	두 조건이 전부 참일경우, 참
논리합(||, OR)	A || B	하나라도 참일 경우, 참
논리 부정(!, NOT)	! C	True <=> False
비트 연산	<< >>	

조건, 반복문

조건 반복문

명령어	
IF	만약 조건문이 해당한다면
switch	변수가 해당하는 것에
삼항 연산자 (조건? A, B)	조건문에 해당, 해당하지 않을 경우
for	조건 만큼 반복
while	조건에 해당할때까지 반복
foreach	순서에 상관없이 for
break	중간에 종료
continue	중간에서 다음 반복문으로 넘김

IF

```
int num = 0;


if (num > 0)
  Console.WriteLine("양수");
else if (num < 0)
  Console.WriteLine("음수");
else
  Console.WriteLine("영");
```

SWITCH

```
int input = 11;

switch (input)
{
    case 12:
        Console.WriteLine("input의 값이 12입니다.");
        break;
    default:
        Console.WriteLine("해당하는 값이 없습니다.");
        break;
}
```

```
string day = "화요일";

switch (day)
{
    case "월요일":
    case "화요일":
    case "수요일":
    case "목요일":
    case "금요일":

Console.WriteLine("평일입니다.");
        break;
    case "토요일":
    case "일요일":

Console.WriteLine("휴일입니다.");
        break;
}
```

삼항 연산자

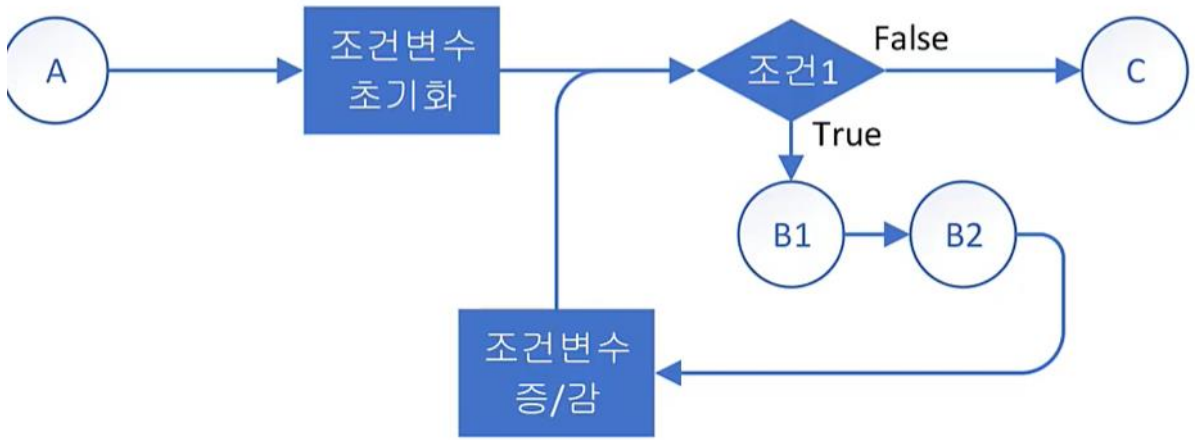
```
int number = 2;
bool isEven;

isEven = (number % 2 == 0) ? true : false;
Console.WriteLine(isEven);
```

FOR

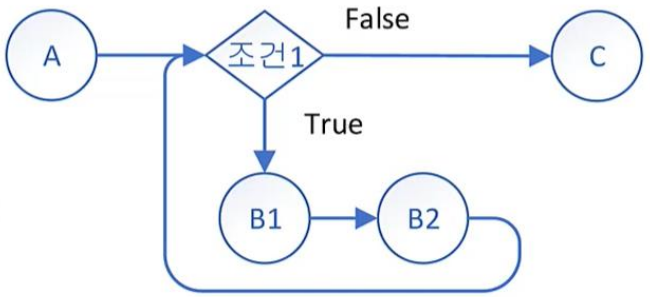
```
for (①초기화; ②조건식; ④반복식)
{
   // ③반복 코드
}
```

```
for(int i=1; i<11; i++)
  Console.WriteLine(i);
```

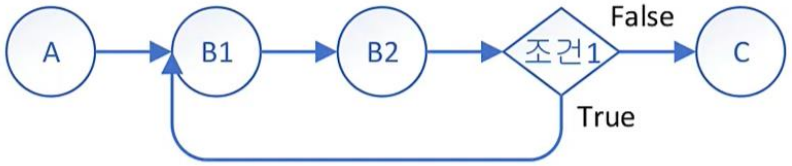


WHILE

```
int i=1;
while(i<11)
  Console.WriteLine(i++);
```




```
Console.WriteLine("A");
 do
{
      Console.WriteLine(i++);
      Console.WriteLine("B");
}
while (i<10)
Console.WriteLine("C");
```



FOREACH

```
string text = "abcde";
foreach (char a in text)
{
    Console.WriteLine(a);
}
```

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

foreach (int i in numbers)
{
    Console.WriteLine(numbers[i] + " ");

}
```

BREAK

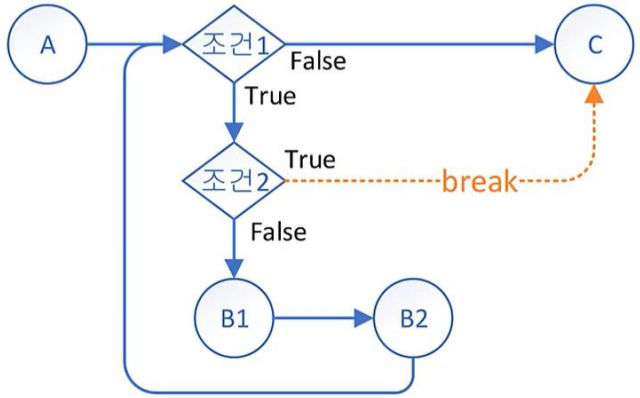
```
for (int i = 0; i < 10; i++)
{
    if (i == 5)
    {
        break;
    }
    Console.WriteLine(i);
}
```

```
Console.WriteLine("A");

while (조건1)
{
  if (조건2)
    break;

  Console.WriteLine("B1");
  Console.WriteLine("B2");
}

Console.WriteLine("C");
```



CONTINUE

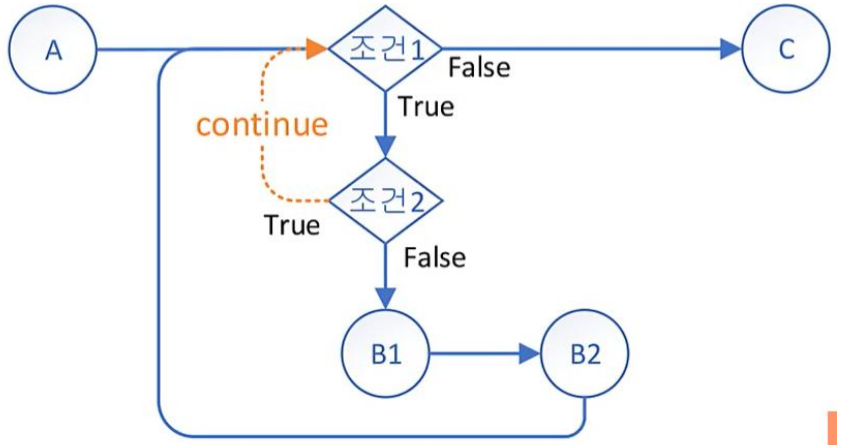
```
for (int i = 0; i < 10; i++)
{
    if (i % 2 == 0)
    {
        continue;
    }
    Console.WriteLine (i+" : 홀수");
}
```

```
Console.WriteLine("A");

while (조건1)
{
  if (조건2)
    continue;

  Console.WriteLine("B1");
  Console.WriteLine("B2");
}

Console.WriteLine("C");
```



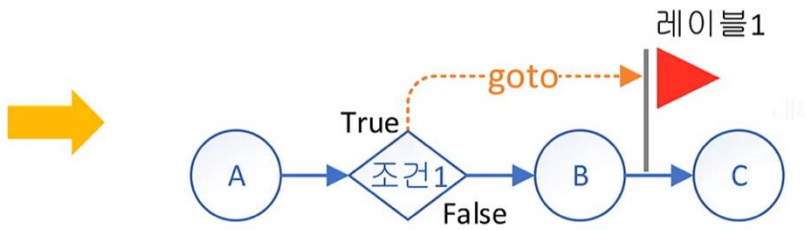
GOTO

```
Console.WriteLine ("A");
If (true)
        goto label1;
    Console.WriteLine ("B");
label1:
Console.WriteLine ("C");
```

```
Console.WriteLine("A");

if (조건1)
    goto 레이블1;
Console.WriteLine("B");

레이블1:
Console.WriteLine("C");
```



추가 문법

CONST 상수 선언

- const int MAX_INT_BIT = 32;

ENUM 열거형식

기본적으로 열거형 멤버의 연결된 상수 값은 int 형식입니다. 즉, 0으로 시작하고 정의 텍스트 순서에 따라 1씩 증가합니다.

```
enum Season
{
    Spring,
    Summer,
    Autumn,
    Winter
}
```

0
1
2
3

ENUM 열거형식 2

```
public enum Days
{
    None      = 0b_0000_0000,  // 0
    Monday    = 0b_0000_0001,  // 1
    Tuesday   = 0b_0000_0010,  // 2
    Wednesday = 0b_0000_0100,  // 4
    Thursday  = 0b_0000_1000,  // 8
    Friday    = 0b_0001_0000,  // 16
    Saturday  = 0b_0010_0000,  // 32
    Sunday    = 0b_0100_0000,  // 64
    Weekend   = Saturday | Sunday
}
```

```
Days meetingDays = Days.Monday | Days.Wednesday | Days.Friday;
        Console.WriteLine(meetingDays);
        // Output:
        // Monday, Wednesday, Friday

        Days workingFromHomeDays = Days.Thursday | Days.Friday;
        Console.WriteLine($"Join a meeting by phone on {meetingDays & workingFromHomeDays}");
        // Output:
        // Join a meeting by phone on Friday

        bool isMeetingOnTuesday = (meetingDays & Days.Tuesday) == Days.Tuesday;
        Console.WriteLine($"Is there a meeting on Tuesday: {isMeetingOnTuesday}");
        // Output:
        // Is there a meeting on Tuesday: False

        var a = (Days)37;
        Console.WriteLine(a);
        // Output:
        // Monday, Wednesday, Saturday
```

배열

변수 선언, 메모리 할당

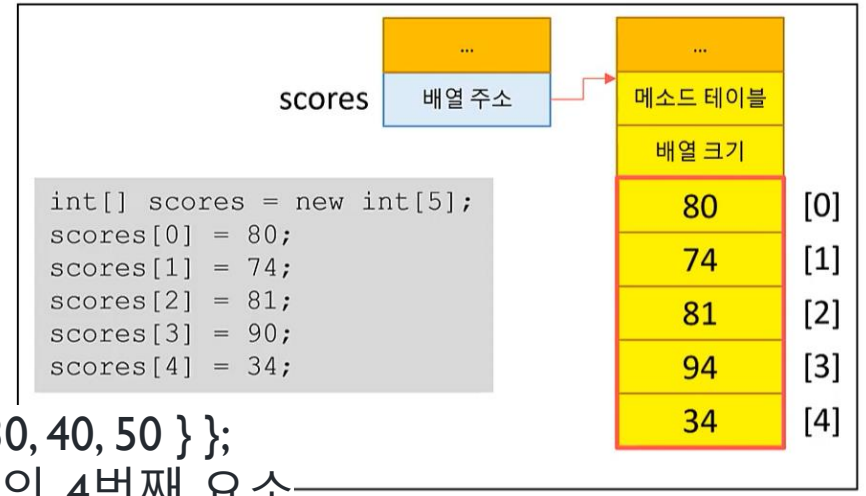
```
int[] arr = new int[2];
```

값 초기화

```
int[] arr2 = new int[2] {1,2};
```

배열의 길이 만큼 for문

```
int[] arr = {1,2,3,4,5};
for(int i =0; i<arr.Length;i++)
{
    print(i);
}
Console.WriteLine(Array.IndexOf(arr, 3));
```



다차원 배열

```
int[,] array2 = { { 1, 2, 3, 4, 5 }, { 10, 20, 30, 40, 50 } };
print(array2[1, 3]); // 2번째 1차원 배열의, 4번째 요소
print(array2[0, 1]); // 1번째 1차원 배열의, 2번째 요소
```

삼차원 배열

```
int[,,] array3 = {
{ { 1, 2, 3, 4, 5 }, { 10, 20, 30, 40, 50 } },
{ { 1, 2, 3, 4, 5 }, { 10, 20, 30, 40, 50 } }
};
```

컬렉션 - ARRAYLIST 클래스

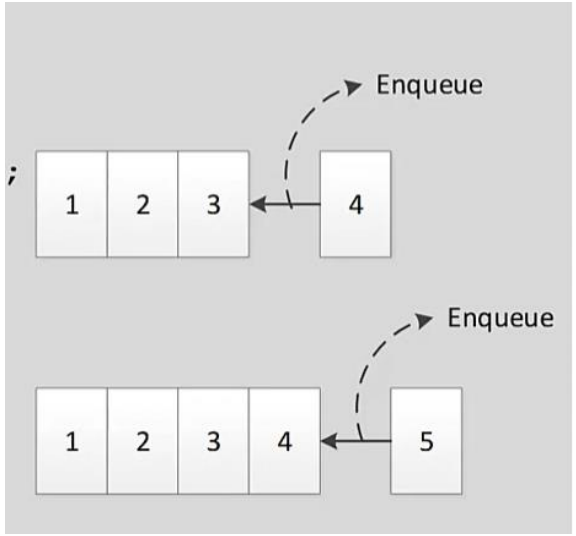
```
ArrayList al = new ArrayList();
   // Add 메소드를 통해 ArrayList에 아이템 추가
   al.Add(1);
   al.Add("Hello");
   al.Add(3.3);
   al.Add(true);
   foreach(var item in al) {
     Console.WriteLine(item);
   }
   Console.WriteLine();
   // Remove 메소드를 통해 ArrayList에서 아이템 삭제
   al.Remove("Hello");
   foreach(var item in al) {
     Console.WriteLine(item);
   }
```

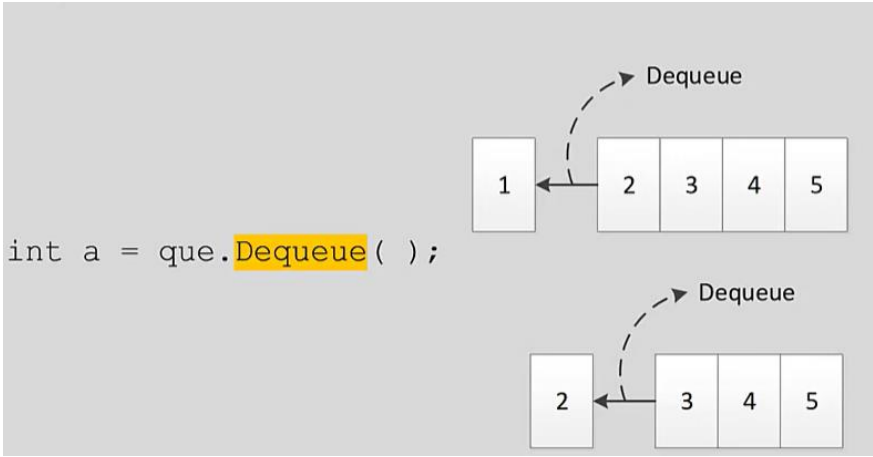
컬렉션 – QUEUE 클래스

```
Queue qu = new Queue();

// Enqueue 메소드를 통해 Queue에 아이템 추가
qu.Enqueue(1);
qu.Enqueue(2);
qu.Enqueue(3);

// Dequeue 메소드를 통해 Queue에서 아이템을 제거
while (qu.Count > 0)
{
    Console.WriteLine(qu.Dequeue());
}
```



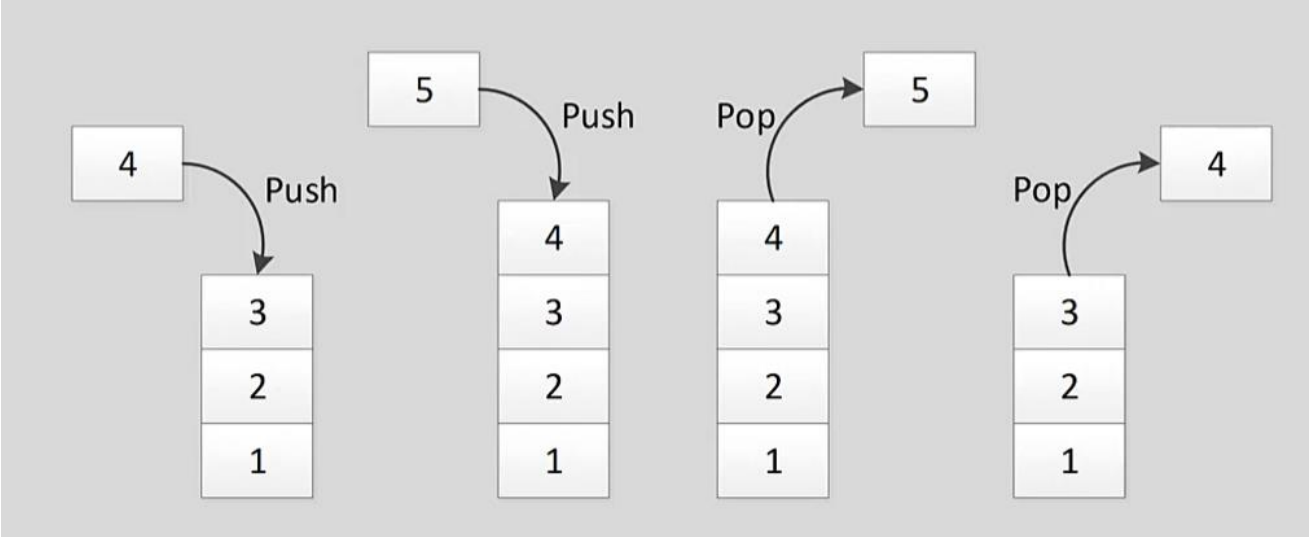


컬렉션 - STACK 클래스

```
Stack st = new Stack();

// Push 메소드를 통해 Stack에 아이템 추가
st.Push(1);
st.Push(2);
st.Push(3);

// Pop 메소드를 통해 Stack에서 아이템을 제거
while (st.Count > 0)
{
    Console.WriteLine(st.Pop());
}
```



컬렉션 - HASHTABLE 클래스

Hashtable 클래스는 키(Key)와 값(Value)으로 저장하는 자료 구조

```
Hashtable ht  = new Hashtable();

// 키와 값 형태로 Hashtable에 아이템 추가
ht["apple"] = "사과";
ht["banana"] = "바나나";
ht["orange"] = "오렌지";

// Hashtable에 저장된 키에 해당하는 값을 출력
Console.WriteLine(ht["apple"]);
Console.WriteLine(ht["banana"]);
Console.WriteLine(ht["orange"]);
```

실습 1. 계산기 만들기

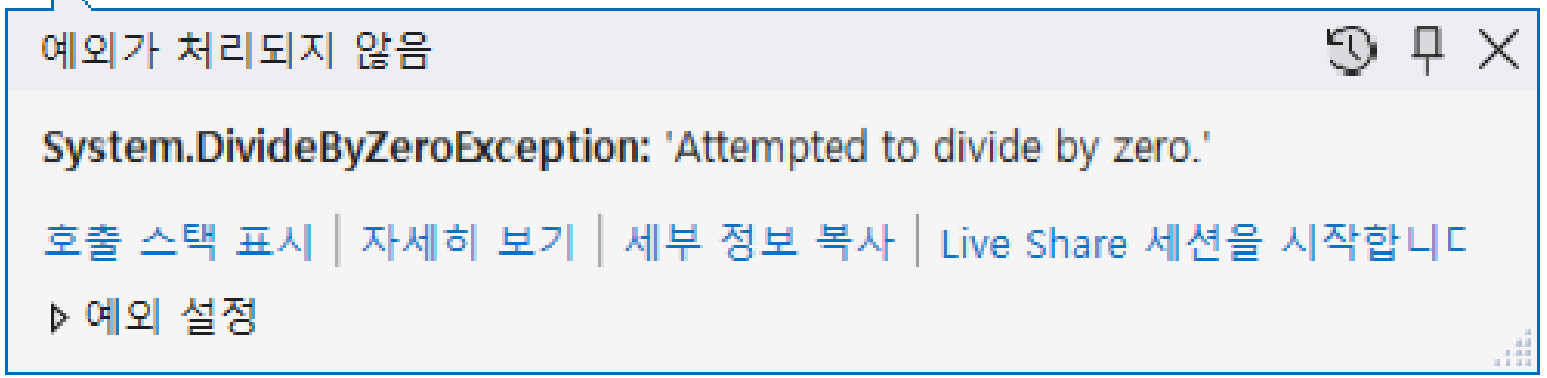
- 사칙연산이 가능한 console 계산기

```
x = Convert.ToInt32(Console.ReadLine());
y = Convert.ToInt32(Console.ReadLine());

result = add(x, y);
```

예외 처리 :TRY ~ CATCH 문

```
Console.Write("나눌 숫자를 입력하세요 : ");
int divider = int.Parse(Console.ReadLine());
Console.WriteLine(10 / divider);
```



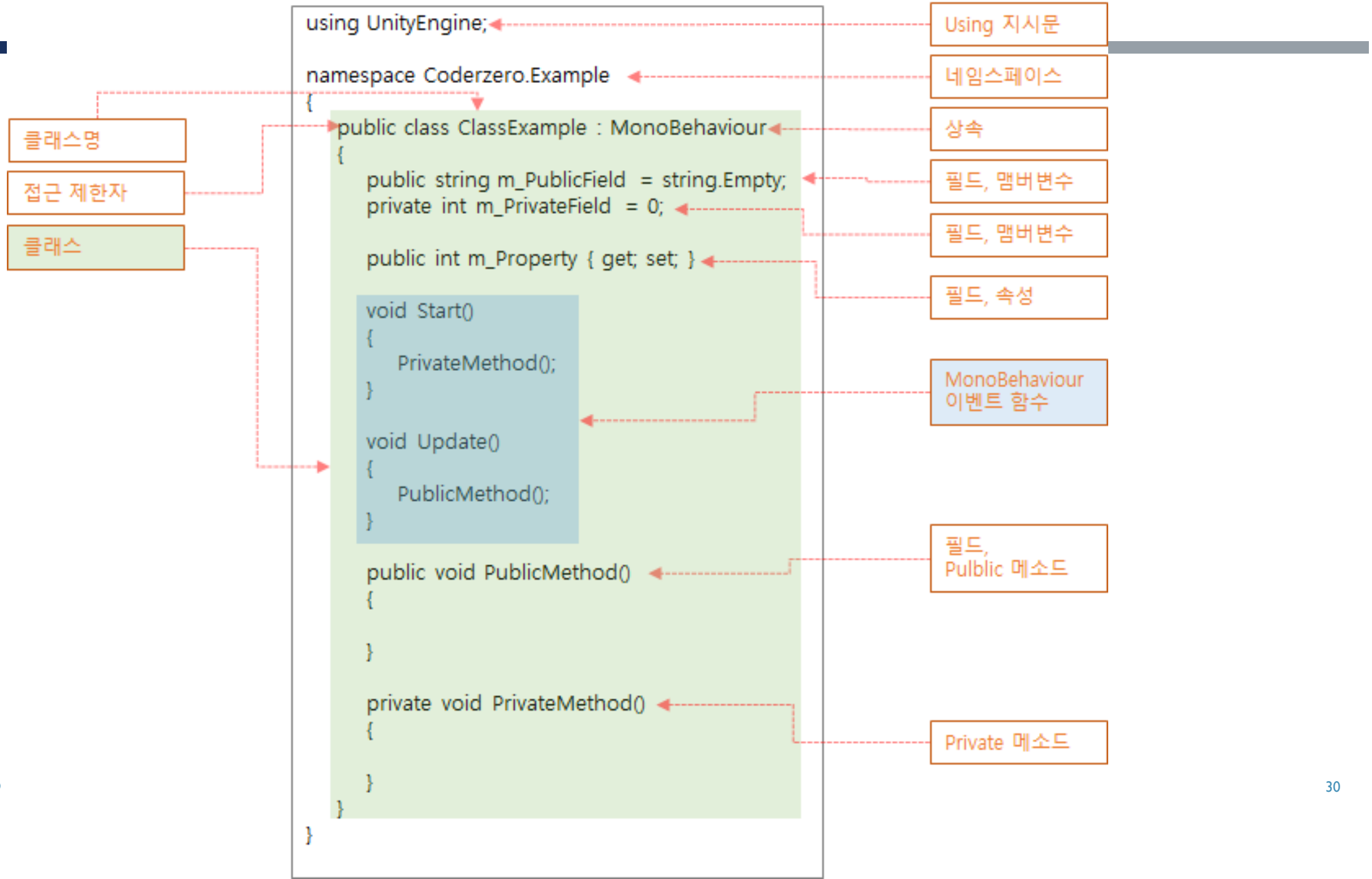
예외 처리 :TRY ~ CATCH 문

```
Console.WriteLine("10을 0으로 나눕니다. ");
int divider = 0;
try
{
    Console.WriteLine(10 / divider);
    Console.WriteLine("0으로 나눌 수 있습니다.");
}
catch
{
    Console.WriteLine("0으로 나눌 수 없습니다.");
}
```

```
int divider = 0;

try
{
    Console.WriteLine(10 / divider);
    Console.WriteLine("0으로 나눌 수 있습니다.");
}
catch (Exception e)
{
    Console.WriteLine("예외 상황 : " + e.Message);
      //    예외 상황 : Attempted to divide by zero.
}
```

객체 지향 언어


using UnityEngine;
namespace Coderzero.Example
{
public class ClassExample : MonoBehaviour
{
public string m_PublicField = string.Empty;
private int m_PrivateField = 0;
public int m_Property { get; set; }
void Start()
{
PrivateMethod();
}
void Update()
{
PublicMethod();
}
public void PublicMethod()
{
}
private void PrivateMethod()
{
}
}
}
클래스명
접근 제한자
클래스
Using 지시문
네임스페이스
상속
필드, 맴버변수
필드, 맴버변수
필드, 속성
MonoBehaviour 이벤트 함수
필드, Pulblic 메소드
Private 메소드

클래스와 인스턴스

- Factory 에서 모형 틀 = 클래스
- 모형틀에 찍힌 빵 = 인스턴스

인스턴스

클래스

```
Person p1;
p1 = new Person();
// 인스턴스 변수를 통해 속성과 메서드 호출
p1.Name = "서준";
p1.Eat();
class Person
{
    // 속성 변수의 값을 null로 초기화
    public string Name = null;
    public string Birthday = null;
    public string Gender = null;

    // 메서드 구현
    public void Eat()
    {
        Console.WriteLine(Name + "이(가) 아침을 먹습니다.");
    }
    public void Walk()
    {
        Console.WriteLine(Name + "이(가) 걷습니다.");
    }
    public void Run()
    {
        Console.WriteLine(Name + "이(가) 뜁니다.");
    }
}
```

클래스 소멸 주기

생성자(Constructor)

인스턴스 생성시 호출
생성자가 오버로딩 되어있어
다중 생성자가 있을경우,
호출한 생성자만 실행.

소멸자(Destructor)
== 종료자(Finalizer)

소멸자도 호출되는 시점을 개발자가
결정하는게 아닌 시스템의 가비지
컬렉터(Garbage Collector)라는
소프트웨어가 결정

생성자와 소멸자(종료자)

```
static void Test()
{
    Cat myCatA = new Cat();
    Cat myCatB = new Cat("하루");
    Cat myCatC = new Cat("코코");
    Cat myCatD = new Cat("몰리", 3);

    // Cat myCatE = new Cat("몰리", 3.5F);
}
Test();
GC.Collect();
Console.ReadLine();
```

```
class Cat
{
    public string Name;
    public int Age;

    // 생성자
    public Cat()
    {
        Name = "길고양이";
        Console.WriteLine("길고양이 생성자가 호출되었습니다.");

    } public Cat(string name)
    {
        Name = name;
        Console.WriteLine("고양이의 이름은 " + Name + "입니다.");
    } public Cat(string name, int age)
    {
        Name = name;
        Age = age;
        Console.WriteLine("고양이의 이름은 " + Name + "이며, 나이는 " + Age +
"kg입니다.");
    }
    // 소멸자(종료자)
    ~Cat()
    {
        Console.WriteLine(Name + "가 사라집니다.");
    }
}
```

소멸자(DESTRUCTOR) == 종료자(FINALIZER)

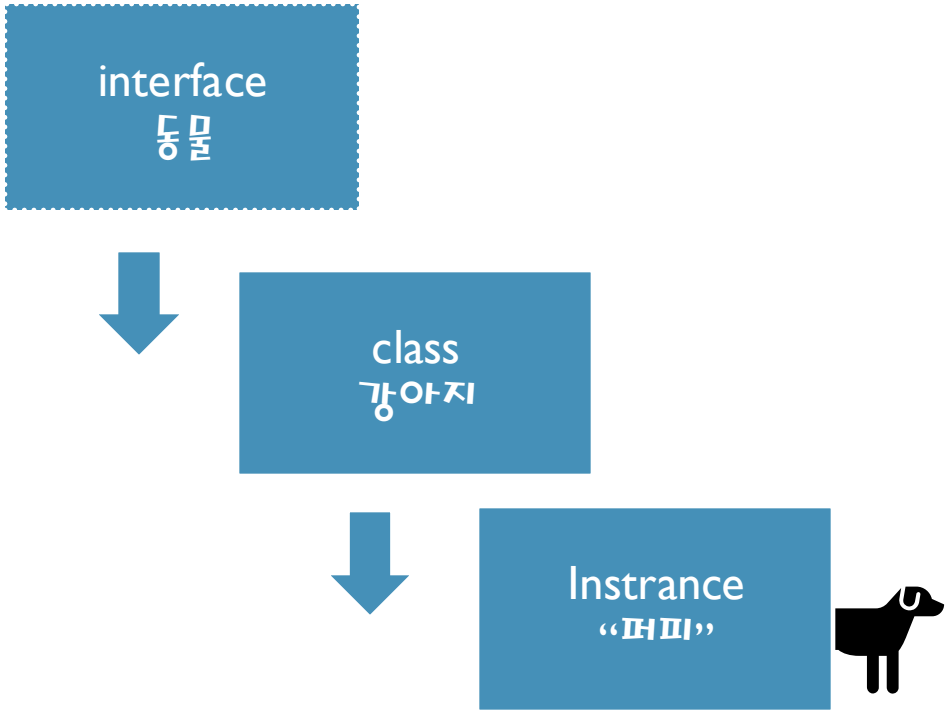
소멸자는 .NET 5(.NET Core 포함) 이상 버전인 경우 애플리케이션이 종료될 때 소멸자를 호출하지 않습니다. 그러므로 콘솔 창에서 소멸자가 호출되었는지 확인할 수 없습니다. .NET 5(.NET Core 포함) 이상 버전에서 소멸자를 강제로 실행하기 위해서는 GC.Collect()를 호출해야 합니다.

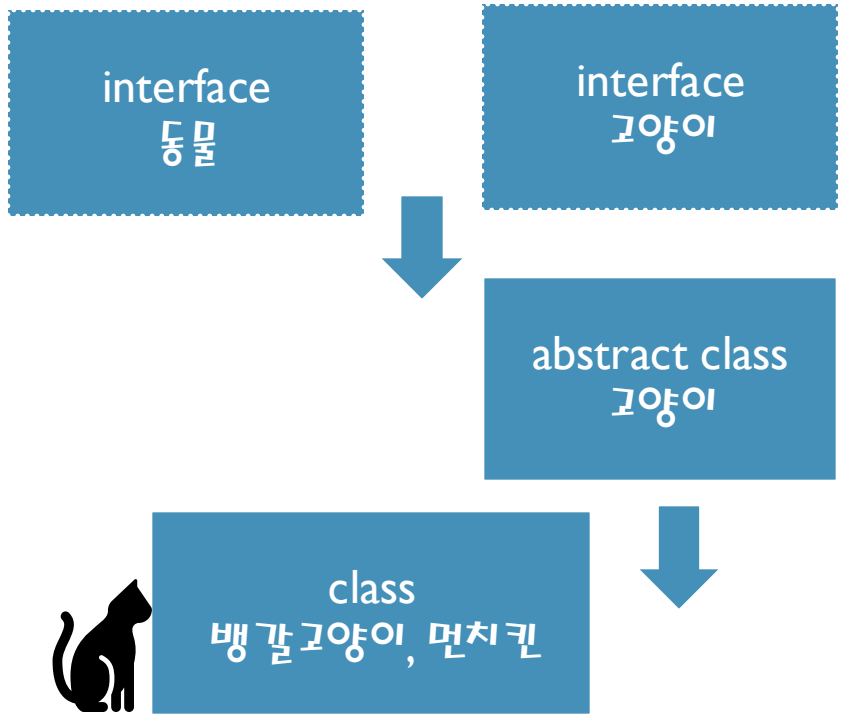
인터페이스와 추상 클래스

C# 인터페이스와 추상클래스의 차이점

	Interface	Abstract Class
접근 지정자	- 함수에 대한 접근 지정자를 가질수 없습니다. - 기본적으로 public 입니다.	- 함수에 대한 접근 지정자를 가질 수 있습니다.
구현	- 구현이 아닌 서명만 가질 수 있습니다.	- 구현을 제공할 수 있습니다.
속도	- 인터페이스가 상대적으로 느립니다.	- 추상 클래스가 빠릅니다.
인스턴스화	- 인터페이스는 추상적이며 인스턴스화 할 수 없습니다.	- 추상클래스는 인스턴스화 할 수 없습니다.
필드	- 인터페이스는 필드를 가질 수 없습니다.	- 추상클래스는 필드와 상수를 정의 할 수 있습니다.
메소드	- 인터페이스에는 추상메소드만 있습니다.	- 추상클래스에는 비추상메소드가 있을 수 있습니다.

인터페이스와 상속





C#은 클래스의 다중 상속을 허용하지 않습니다.
다만, 인터페이스의 다중 상속은 허용합니다.