C#
-CAHPTER2-

SOUL SEEK

목차

클래스와 프로퍼티

1. 클래스와 프로퍼티

클래스

- 구조체는 **Value Type,** 클래스는 **Reference Type**이다.
- **C#**에서는 특정 값 타입을 반환하는 클래스틑 사용 불가능하다.

프로퍼티

- **C#**에서는 **private**와 **public**을 **C++**처럼 선언하지 않고 각 변수마다 **public int a**라는 형식으로 직접적으로 선언해줘야 한다.
- 구조체의 경우 **struct**가 **public** 이라고 암묵적으로 구성변수도 **public** 으로 되는 것이 아니다
- 이루고 있는 변수들도 같이 **public** 으로 다 선언해줘야 한다.

get, set

- 접근자라고 하며 접근제어로 감추어진 값을 대입하거나 얻어오게 하는 역할을 한다.
- 직접적인 참조나 대입이 아닌 간접적인 허락에 의한 행동.
- **set**사용하면 **private**로 선언한 의미가 너무 퇴색된다 다른 곳에 선언해서 함부로 사용되는 데이터를 보호하기 위해 사용되는 것인데 **set**을 통해 저장하게 되면 자기가 원하는 순간에 대입해서 사용하려 하는 의미가 줄어들게 되는 것이다. 그래서 **set**이 존재하지만 따로 대입 함수를 만들어서 사용하는걸 추천한다.

배열 & 문자열

2. 배열 & 문자열

일반 배열 초기화 타입

- Int[] a = new int[]{1, 2, 3};
- Int[] a = new int[3]{1, 2, 3};
- Int[] a = {1, 2, 3};

객체 배열 초기화

- Animal[] a = new Animal[3];
- A[0] = new Animal();

다차원 배열

- Int[,] a = new int[4,3];

2. 배열 & 문자열

- **System.String** 클래스와 같은 것이다.
  - 그래서 **string**으로 선언하면 **reference type**이 되는 것
- **Immutable Type**
  - 한번 그값이 설정되면 다시 변경할 수 없는 타입이 됩니다. 예제) **s = "C#";** 이라고 한후 **s = "F#"**이라고 실행하면 값이 바뀌는 것이 아니라 새로운 **string**객체를 생성해서 **F#**이라는 데이터로 초기화 한 후 할당해버린다. **s**는 내부적으로 전혀 다른 메모리를 갖는 객체를 가르키는 것이 된다.

- 문자의 집합체 인덱스로 문자요소에 접근 가능.

- **Substring**
  - 문자열의 위치를 이용하여 문자열을 컨트롤 한다.
- **Split**
  - 지정된 문자를 기준으로 문자열을 분리한다.
- **IndexOf, LastIndexOf**
  - 문자열에서 특정문자의 위치를 찾는다.
  - **indexOf :** 찾은 문자열의 처음 문자열의 위치를 알려준다.
  - **lastIndexOf :** 찾은 문자열의 마지막 문자열의 위치를 알려준다.
- **Replace**
  - 문자열을 변경한다
- **ToUpper, ToLower**
  - 대소문자 변경
- **Trim**
  - 문자열의 공백문자를 제거한다.
- 정수형과 문자열간의 변경.
  - 문자열--->숫자, 숫자--->문자열

2. 배열 & 문자열

StringBuilder클래스

- **System.Text.StringBuilder** 클래스
- 루프 안에서 문자열을 추가 변경 시 **String** 대신 사용
- **String** 과 달리 문자열의 갱신이 많이 필요한 곳에 사용된다**.** 그 이유는 메모리를 생성**,** 소멸하지 않고 일정한 버퍼를 갖고 문자열을 갱신을 효율적으로 처리하기 때문이다**.**

컬렉션 & 제너릭

3. 컬렉션 & 제너릭

C#에서 제공하는 자료구조

- **C++ std** 템플릿 **Collection**
- **using System.Collection**

ArrayList

- 큐, 스택, 헤쉬테이블, 배열처럼 인덱스로 요소 접근이 가능하다
- 배열과 같은 배열크기를 지정해줘야하는 일이 없고 추가 삭제에 따라 자동으로 크기를 늘였다 줄였다 한다.**(STL vector)**
  모든 타입의 변수를 담을 수 있다.
- 어떠한 데이터 타입이든 전부 담을 수 있는 것은 장점이지만 역시 **object** 형으로 박싱되어 저장되기 때문에 사용할때 언박싱이 일어나게 된다. 즉, 입출력을 할때마다 박싱과 언박싱이 일어난다

Generic - using System.Generic

- **Queue, Stack, List, Dictionary(Generic Collection)** 성능상의 문제로 형식을 지정해서 사용하는 **Generic Collection**으로 대처한다.

List<T>, Dictionary<T>

- **T**에 타입을 지정하고 지정한 타입외에는 사용 할수 없다.

- **List : ArrayList**랑 사용법이 같다.**(**예제**)**
- **Dictionary :** 데이터에 접근하기 위한 키와 데이터값인 벨류로 이루어진다. 일반적인 순차적인 인덱스 구성이라고 볼수 없다. 데이터 구조에서 특정 데이터값을 매칭시키는 경우에 많이 사용한다.
- 예를 들면 게임 데이터 테이블 같은경우 데이터 **ID**와 데이터를 매칭하는데 이것을 키와 벨류로 구분해서 가지고 있는 경우가 많다.