



# C#


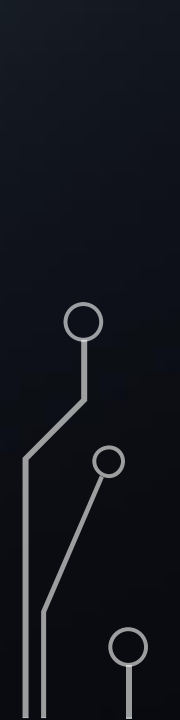
## -CHAPTER4-

SOUL SEEK



# 목차

---

1. 배열과 문자열
  2. 컬렉션
- 
- 

# 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];`

C#에서 배열은 .NET Framework의 **System.Array** 클래스에 대응되므로 배열로 선언하면 **System.Array** 클래스에서 제공하는 **Method**를 사용할 수 있다.

분류	이름	설명
정적 메소드	<b>Sort()</b>	배열을 정렬한다.
	<b>BinarySearch&lt;T&gt;()</b>	이진 탐색을 수행한다.
	<b>IndexOf()</b>	배열에서 찾고자 하는 특정 데이터의 인덱스를 반환한다.
	<b>TrueForAll&lt;T&gt;()</b>	배열의 모든 요소가 지정한 조건에 부합하는지의 여부를 반환한다.

## 2. 배열 & 문자열

분류	이름	설명
정적 메소드	<b>FindIndex&lt;T&gt;()</b>	배열에서 지정한 조건에 부합하는 첫 번째 요소의 인덱스를 반환한다. <b>indexOf()</b> 메소드가 특정 값을 찾는데 비해, <b>FindIndex</b> 는 지정한 조건을 바탕으로 값을 찾는다.
	<b>Resize&lt;T&gt;()</b>	배열의 크기를 재조정한다.
	<b>Clear()</b>	배열의 모든 요소를 초기화한다. 배열의 숫자 형식 기반이면 <b>0</b> 으로, 논리 형식 기반이면 <b>false</b> 로, 참조 형식 기반이면 <b>null</b> 로 초기화 한다.
	<b>ForEach&lt;T&gt;()</b>	배열의 모든 요소에 대해 동일한 작업을 수행하게 한다.
인스턴스 메소드	<b>GetLength()</b>	배열에서 지정한 차원의 길이를 반환한다. 이 메소드는 다차원 배열에 유용하다.
프로퍼티	<b>Length</b>	배열의 길이를 반환한다.
	<b>Rank</b>	배열의 차원을 반환한다.

## 2. 배열 & 문자열

### String

- **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

### **StringBuilder**클래스

- **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



# 3. 컬렉션

## C#에서 제공하는 자료구조

- `using System.Collection`

### ArrayList

- 인덱스로 요소 접근이 가능하다
- 배열과 같은 배열크기를 지정해줘야 하는 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 사용할 때 언박싱이 일어나게 된다. 즉, 입출력을 할때 마다 박싱과 언박싱이 일어난다

### Queue

- 선입선출(**FIFO**) → 먼저 들어온 데이터가 먼저 나간다. **(First In First Out)**
- **CPU**나 프린터에서 나타나는 작업대기열이 이것과 똑같은 형식으로 처리되는 것이다.
- **Enqueue**와 **Dequeue**로 입력, 출력을 담당한다.

### Stack

후입선출(**LIFO**) → 나중에 들어온 데이터가 먼저 나간다. **(Last In First Out)**  
계산기가 대표적인 경우이고 **Push()**, **Pop()**으로 입력, 출력을 담당한다.

### Hashtable

- **Key**와 **Value**의 쌍으로 이루어진 데이터를 다룰 때 사용한다.
- 사전이 가장 좋은 예가 될 수 있다.
- 데이터 덩어리를 인덱스로 관리하는 좋은 구조이다.