

프로퍼티

UPDATE 2023

01. getter, setter

- 할당 연산자 =를 이용한 필드 setter, getter
- Get/Set 메소드를 사용한 필드 setter, getter

```
public string Name1;
```

```
private string Name2;
```

참조 1개

```
public string GetName()
```

```
{  
    return Name2;  
}
```

참조 1개

```
public void SetName(string value)
```

```
{  
    this.Name2 = value;  
}
```

```
MyClass myClass = new MyClass();
```

```
myClass.Name1 = "윤지훈";  
Console.WriteLine(myClass.Name1);
```

```
myClass.SetName("박지훈");  
Console.WriteLine(myClass.GetName());
```

02. 프로퍼티

- 프로퍼티를 이용한 필드 setter, getter

```
class 클래스이름
{
    데이터형식 필드이름;
    접근한정자 데이터형식 프로퍼티이름
    {
        get
        {
            return 필드이름;
        }
        set
        {
            필드이름 = value;
        }
    }
}
```

```
class MyClass
{
    private int myField;
    public int MyField
    {
        get
        {
            return myField;
        }
        set
        {
            myField = value;
        }
    }
}
```

```
MyClass obj = new MyClass();
obj.MyField = 3 ;
Console.WriteLine( obj.MyField );
```

03. 프로퍼티를 이용한 getter/setter

// property 방식

```
private string Name3;
```

참조 2개

```
public string name3 {  
    get { return this.Name3; }  
    set { this.Name3 = value; }  
}
```

// property 방식을 이용한 필드 방식

참조 2개

```
public string Name4 { get; set; }
```

```
MyClass myClass = new MyClass();
```

```
myClass.name3 = "최지훈";  
Console.WriteLine(myClass.name3);
```

```
myClass.Name4 = "이지훈";  
Console.WriteLine(myClass.Name4);
```

04. 프로퍼티를 이용한 유효성검사

```
class Namecard
{
    private string Name; // 2글자 이상
    private int Age;     // 음수 불가능

    참조 4개
    public string name
    {
        get { return this.Name; }
        set {
            if (value.Length >= 2) this.Name = value;
            else this.Name = "Unknown";
        }
    }
}
```

04. 프로퍼티를 이용한 유효성검사

```
public int age
{
    get { return this.Age; }
    set
    {
        if (value >= 0) this.Age = value;
        else this.Age = 0;
    }
}
```

04. 프로퍼티를 이용한 유효성검사

```
Console.WriteLine("=====");
Namecard myCard = new Namecard();
myCard.name = "오";
myCard.age = -99;
Console.WriteLine($"이름 : {myCard.name}");
Console.WriteLine($"나이 : {myCard.age}");

myCard.name = "공유";
myCard.age = 37;
Console.WriteLine($"이름 : {myCard.name}");
Console.WriteLine($"나이 : {myCard.age}");
```

이름 : Unknown
나이 : 0
이름 : 공유
나이 : 37

퀴즈

- Student 클래스의 필드가 아래의 조건에 맞추어 입력될 수 있게 선언하고 객체화 시켜 결과를 확인하여라.

```
class Student
{
    private int No;           // 3000 ~ 3999 오류시 0
    private int Grade;        // 1~4 오류시 0
    private char Gender;      // F 나 M 오류시 U
}
```


컬렉션

UPDATE 2023

01. 컬렉션 개요

- 컬렉션(Collection)이란 같은 성격의 데이터 모음을 담는 자료 구조로 .NET 프레임워크의 컬렉션 클래스에서 상속받아 사용된다.
 - ArrayList
 - Queue
 - Stack
 - Hashtable

[illegible]

01. 컬렉션 개요

- 배열과 컬렉션 비교
 - ✓ 배열: 정수형, 문자열 등 집합을 나타냄
 - ✓ 컬렉션: 개체의 집합을 나타내며 리스트, 집합, 맵, 사전도 컬렉션과 같은 개념으로 사용된다.
- 데이터를 그룹으로 묶어 관리할 때는 일반적으로 배열로 관리한다.
배열은 크기가 고정되어 있어 새로운 데이터를 추가할 수 없다는 단점이 있는데 이러한 단점을 제거한 것이 바로 컬렉션이다.

02. ArrayList

- 배열 처럼 [] 연산자 이용, 특정 위치 요소에 데이터 할당이 가능이 가능하다.
- 용량을 미리 지정할 필요 없고 필요에 따라 용량 증가/감소가 가능하다.
- 데이터의 형식이 같지 않아도 된다.
- 대표적인 메소드 : Add(), RemoveAt(), Insert()
- 전체 길이 속성 : Count

```
ArrayList list = new ArrayList();  
list.Add( 10 );  
list.Add( 20 );  
list.Add( 30 );  
list.RemoveAt( 1 ); // 20을 삭제  
list.Insert( 1, 25 ); // 25를 1번 인덱스에 삽입. 즉, 10과 30 사이에 25를 삽입
```

02. ArrayList

using System.Collections; 으로 클래스 임포트가 필요하다.

```
> using System.Collections;
>
> ArrayList list = new ArrayList();
> list.Add("C#");
> list.Add("TypeScript");
>
> for (int i = 0; i < list.Count; i++)
. {
.     Console.WriteLine(list[i].ToString());
. }
C#
TypeScript
```

02. ArrayList

- ArrayList 클래스의 인스턴스인 list를 생성한 후 Add() 메서드로 문자열 등을 저장할 수 있다.
- for 문 등으로 list[i] 형태로 ArrayList에 저장된 값을 읽어 사용할 수 있다.
- ArrayList의 Add() 메서드는 매개변수로 object 형식을 받기에 문자열을 포함한 C#의 모든 데이터 형식을 저장하고 사용할 수 있다.
- Add()로 추가된 항목은 Remove() 같은 메서드로 제거할 수 있다.

02. ArrayList

```
ArrayList list = new ArrayList();  
for (int i = 0; i < 5; i++)  
{  
    list.Add(i);  
}
```

```
foreach (var item in list)  
{  
    Console.Write(item);  
}  
Console.WriteLine();
```

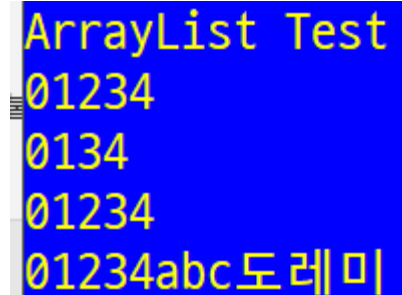
02. ArrayList

```
list.RemoveAt(2);  
foreach (var item in list)  
{  
    Console.Write(item);  
}  
Console.WriteLine();
```

```
list.Insert(2, 2);  
foreach (var item in list)  
{  
    Console.Write(item);  
}  
Console.WriteLine();
```


02. ArrayList

```
list.Add("abc");  
list.Add("도레미");  
for (int i = 0; i < list.Count; i++)  
{  
    Console.Write(list[i]);  
}  
Console.WriteLine();
```



```
ArrayList Test  
01234  
0134  
01234  
01234abc도레미
```

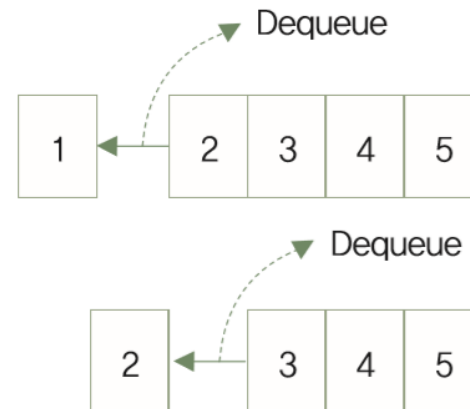
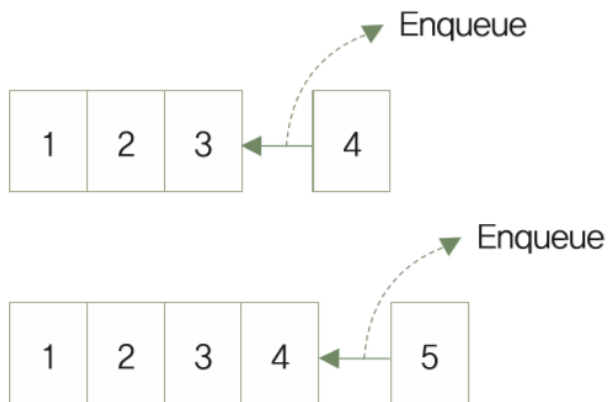
03. Queue

- Queue 클래스는 먼저 들어온 데이터가 먼저 나간다. 일반적으로 은행에서 먼저 온 사람을 처리하는 것처럼 큐(Queue)라는 단어는 기다림 통로(은행 줄서기) 또는 FIFO(First In First Out)(선입선출)로 표현되며, 먼저 들어온 것이 먼저 나가는 형태의 데이터를 다룰 때 사용한다.
- CPU의 작업, 프린터의 여러 문서 출력, 스트리밍 서비스에서 콘텐츠 버퍼링에도 큐의 예이다.
- 입력은 오직 뒤에서 출력은 앞에서만 이루어진다.

```
Queue que = new Queue();  
que.Enqueue( 1 );  
que.Enqueue( 2 );  
que.Enqueue( 3 );  
que.Enqueue( 4 );
```

03. Queue

- Enqueue() 메소드 – 데이터를 입력한다.
- Dequeue() 메소드 – 데이터를 출력한다.



03. Queue

```
> using System.Collections;
>
> var queue = new Queue(); //① Queue 클래스의 인스턴스 생성
>
> queue.Enqueue(10);        //② 큐(대기 행렬)에 데이터 입력: Enqueue()
> queue.Enqueue(20);
> queue.Enqueue(30);
>
> queue.Dequeue()           //③ 큐에서 데이터 출력: Dequeue()
10
```

03. Queue

```
> queue.Dequeue()
```

```
20
```

```
> queue.Dequeue()
```

```
30
```

```
> queue.Dequeue()
```

```
System.InvalidOperationException: 큐가 비어 있습니다.
```

```
+ System.Collections.Queue.Dequeue()
```

03. Queue

```
Queue que = new Queue();  
que.Enqueue("홍길동.");  
que.Enqueue("장나라.");  
que.Enqueue("김석진.");  
  
while (que.Count > 0)  
{  
    Console.WriteLine(que.Dequeue());  
}
```

04. Stack

- Stack 클래스는 단어 그대로 음식을 담는 접시처럼 아래에서 위로 데이터를 쌓는 형태의 자료 구조이다.
- LIFO(Last In First Out)(후입선출) : 나중에 들어온 데이터가 먼저 출력되는 자료 구조
- Count : 스택에 있는 데이터 개수 조회한다.
- Push() : 스택에 데이터를 저장한다.
- Pop() : 스택에서 데이터를 꺼낸다.
- 오버플로(Overflow) : 스택이 꽉 차는 것을 의미한다.
- 언더플로(Underflow) : 스택이 비어 있는 것을 의미한다.


04. Stack

> using System.Collections;

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


04. Stack

```
Stack stack = new Stack();  
stack.Push(100);  
stack.Push(true);  
stack.Push("Maria");  
while (stack.Count > 0)  
{  
    Console.WriteLine(stack.Pop());  
}
```



Maria
True
100

05. Hashtable

- 해시테이블(Hashtable) 클래스는 키와 값의 쌍으로 이루어진 데이터를 다룰 때 사용한다. 다른 언어에서는 딕셔너리라고 한다.
- 탐색 속도가 빠르며 정수 인덱스 및 문자열 인덱스를 사용할 수 있다.
- 배열과 비교한 장점
 - 데이터를 저장할 요소의 위치로 키를 사용한다.
 - 키로 사용할 수 있는 데이터 형식에 제한이 없다.
 - 키를 이용해 단번에 데이터 저장 위치인 컬렉션 내의 주소를 계산(해싱)한다.

05. Hashtable

```
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"] );
```

05. Hashtable

```
Console.WriteLine("Hashtable Test");  
Hashtable ht = new Hashtable();  
ht["하나"] = "one";  
ht["둘"] = "two";  
ht["셋"] = "three";  
Console.WriteLine(ht["하나"]);  
Console.WriteLine(ht["둘"]);  
Console.WriteLine(ht["셋"]);
```

06. 컬렉션 초기화

- 배열을 이용한 컬렉션 초기화
 - 배열 생성 후 배열 값으로 컬렉션을 초기화할 수 있다.

```
int[] arr = { 123, 456, 789 };  
  
ArrayList list = new ArrayList(arr); // 123, 456, 789  
Stack stack = new Stack(arr); // 789, 456, 123  
Queue queue = new Queue(arr); // 123, 456, 789
```

- ArrayList의 경우 중괄호에 값을 지정하여 직접 초기화할 수 있다.

```
ArrayList list2 = new ArrayList() { 11, 22, 33 };
```

컬렉션 초기자는 생성자를 호출할 때,
생성자 뒤에 {와} 사이에 컬렉션 요소
의 목록을 입력하여 사용합니다.

06. 컬렉션 초기화

- 해시테이블의 경우 중괄호안에 [키]=값 형태로 초기화할 수 있다.

```
Hashtable ht = new Hashtable()
{
    ["하나"] = 1, // ;가 아니라 ,를 이용하여 항목을 구분합니다.
    ["둘"] = 2,
    ["셋"] = 3
};
```

06. 컬렉션 초기화

```
// ArrayList 초기화
ArrayList mylist = new ArrayList(arr);
foreach (var item in mylist)
{
    Console.Write($" {item} ");
}
Console.WriteLine();
```

06. 컬렉션 초기화

```
// Stack 초기화
Stack mystack = new Stack(arr);
foreach (var item in mylist)
{
    Console.Write($" {item} ");
}
Console.WriteLine();
```


06. 컬렉션 초기화

```
// Queue 초기화
Queue myqueue = new Queue(arr);
foreach (var item in myqueue)
{
    Console.Write($" {item} ");
}
Console.WriteLine();
```

06. 컬렉션 초기화

```
// ArrayList를 {}를 이용하여 초기화
ArrayList mylist2 = new ArrayList { 100, 200, 300 };
foreach (var item in mylist2)
{
    Console.Write($" {item} ");
}
Console.WriteLine();
```

06. 컬렉션 초기화

```
// Hashtable 초기화1
Hashtable myhashtable1 = new Hashtable()
{
    ["a"] = "africa",
    ["b"] = "banana",
    ["c"] = "cat"
};
Console.WriteLine(myhashtable1["a"]);
Console.WriteLine(myhashtable1["b"]);
Console.WriteLine(myhashtable1["c"]);
```

06. 컬렉션 초기화

```
// Hashtable 초기화2
Hashtable myhashtable2 = new Hashtable()
{
    {"c", "can" },
    {"d", "dress" }
};
Console.WriteLine(myhashtable2["c"]);
Console.WriteLine(myhashtable2["d"]);
```

퀴즈

- 입력 받은 데이터를 이용하여 ArrayList를 생성하고 출력하여라.

좋아하는 음식은? 초밥

최근 본 영화는? 알라딘

좋아하는 가수는? BTS

좋아하는 숫자? 10

최근 여행지? 부산

당신에 관한 리스트 : ['초밥', '알라딘', 'BTS', 10, '부산']

퀴즈

- 아래와 같이 어레이 리스트를 정의하고 다음과 같이 출력하여라.

{'사과','망고','치즈케이크','주스'}

우리집 냉장고에는? ['사과', '망고', '치즈케이크', '주스']

동생이 사과를 먹었다

우리집 냉장고에는? ['망고', '치즈케이크', '주스']

이모가 수박을 사오셨다.

우리집 냉장고에는? ['망고', '치즈케이크', '주스', '수박']

동생 친구가 치즈케이크,수박을 먹었다.

우리집 냉장고에는? ['망고', '주스']

퀴즈

- stack을 이용하여 문자열을 구성하는 낱개 문자를 각각 저장하고 다음과 같이 출력하여라
- Before : 도레미파솔라시
- After : 시 = 라 = 솔 = 파 = 미 = 레 = 도 =

퀴즈

- 3의 배수로 구성된 길이 15의 배열을 만들어 Queue 형태의 저장소에 삽입하고 결과를 확인하여라

$^3 \ ^6 \ ^9 \ ^{12} \ ^{15} \ ^{18} \ ^{21} \ ^{24} \ ^{27} \ ^{30} \ ^{33} \ ^{36} \ ^{39} \ ^{42} \ ^{45}$

퀴즈

- 공백으로 구성되어 있는 문자열 단어를 분리시켜 HashTable 형태로 저장하고 출력하여라.
- 키는 각 단어의 첫 글자로 한다.

```
string myWord = "banana fruit orange apple mango";
```