

컴퓨터정보과 C# 프로그래밍

2021년도/1학기/13주차
장은미

제네릭 - 일반화 프로그래밍

- 클래스나 메소드에서 사용하는 내부 자료형을 특정하지 않고 **형식매개변수**를 사용하여 실행시마다 원하는 타입으로 동작할 수 있는 방식
 - 보통 형식매개변수는 T를 사용한다.
 - 형식 매개변수를 2개 이상 사용하면 T1, T2나 의미를 갖도록 다르게 정의한다.

```
namespace System.Collections.Generic
{
    ...public class List<T> : IList<T>, ICollection<T>, IEnumerable<T>,
    {
        ...public List();
        ...public List(int capacity);
        ...public List(IEnumerable<T> collection);

        ...public T this[int index] { get; }
        ...public int Count { get; }
        ...public int Capacity { get; set; }

        ...public void Add(T item);
        ...public void AddRange(IEnumerable<T> collection);
        ...public ReadOnlyCollection<T> AsReadOnly();
        ...public int BinarySearch(int index, int count, T item, ICompare
    }
```

```
namespace System.Collections.Generic
{
    ...public class Dictionary<TKey, TValue> : IDictionary<TKey, TValue>,
    {
        ...public Dictionary();
        ...public Dictionary(int capacity);
        ...public Dictionary(IEqualityComparer<TKey> comparer);
        ...public Dictionary(IDictionary<TKey, TValue> dictionary);
        ...public Dictionary(int capacity, IEqualityComparer<TKey> compare
        ...public Dictionary(IDictionary<TKey, TValue> dictionary, IEquali
        ...protected Dictionary(SerializationInfo info, StreamingContext c

        ...public TValue this[TKey key] { get; }
        ...public ICollection<TValue> Values { get; }
        ...public ICollection<TKey> Keys { get; }
        ...public int Count { get; }
    }
```

인덱서가
구현되어 있음을 알 수 있다.
(인덱서는 시험에 나오지 않음)

Generic 미적용

```
class ListInt
{
    int[] list;
    // 나머지 동일
}
```

```
class ListDouble
{
    double[] list;
    // 나머지 동일
}
```

```
ListInt score1 = new ListInt();
ListDouble score2 = new ListDouble();
```

*동일한 내용인데, list[]의 타입이 다르다는 이유만으로
클래스를 별도로 구성해야 함

Generic 적용

```
class ListGeneric<T>  
{  
    T[] list;  
}
```


```
ListGeneric<int> score1 = new ListGeneric<int>();  
ListGeneric<double> score2 = new ListGeneric<double>();
```

*형식매개변수를 이용하여 하나의 클래스로 다양한 값의 배열을 생성할 수 있다.

Dictionary<TKey, TValue>의 예

```
Dictionary<int, string> dics = new Dictionary<int, string>();  
dics[1] = "One";  
dics[2] = "Two";  
Console.WriteLine(dics[1]);  
Console.WriteLine(dics[0]); //0값의 key가 없음 - 에러발생
```

```
Dictionary<string, string> dics = new Dictionary<string, string>();  
Dics["1"] = "One";  
Dics["2"] = "Two";  
Console.WriteLine(dics[1]); //key 타입이 string이기 때문에 에러  
Console.WriteLine(dics["1"]);  
Console.WriteLine(dics["0"]); // "0"값의 key가 없음 - 에러발생
```



제네릭 예1 - 참고만 할 것

```
public class CTriggerValue_Struct<T>
    where T : struct
{
    T _value;

    public T Value
    {
        get
        {
            return _value;
        }

        set
        {
            if (!_value.Equals(value)) {
                _value = value;
                OnValueChanged();
            }
        }
    }

    public string PRINT_VALUE
    {
        get
        {
            return _value.ToString();
        }
    }

    public event System.EventHandler ValueChanged;
    protected virtual void OnValueChanged()
    {
        if (ValueChanged != null) {
            ValueChanged(this, EventArgs.Empty);
        }
    }
}
```

```
private CTriggerValue_Struct<bool> TempControllerUseSkip = new CTriggerValue_Struct<bool>();
```

```
TempControllerUseSkip.ValueChanged += TempControllerUseSkip_ValueChanged;
```

```
private void TempControllerUseSkip_ValueChanged(object sender, EventArgs e)
{
    #region Temp Controller

    if (STARTOPT.TEMP_CTRL) {
        if (TempControllerUseSkip.Value) {
            DoUseTempController();
        } else {
            DoSkipTempController();
        }
    }

    #endregion
}
```

```
TempControllerUseSkip.Value =
    (WholeCommand.Instance.ctrlCmd.CtrlTemp[(int)CTRLCmdContents.eCtrlTemp.USESKIP_HEATER_TEMP] == 1);
```

제네릭 예2 - 참고만 할 것

```
public static class EnumUtil<T>
{
    public static T EnumParse(string s)
    {
        return (T)Enum.Parse(typeof(T), s);
    }
}
```

```
public enum eFromSCREEN
{
    AutRun,
    Status,
    Manual,
    Tnit
}
```

```
public string screen;
```

```
public eFromSCREEN cmdScreen;
```

```
public string Content;
```

```
public string Before;
```

```
public string After;
```

```
public CLogUserParameterTupLeOld(DateTime Date, string Auth, string
{
```

```
    this.Date = Date;
```

```
    this.Auth = Auth;
```

```
    this.UserName = UserName;
```

```
    this.Screen = Screen;
```

```
    try {
```

```
        this.cmdScreen = EnumUtil<eFromSCREEN>.EnumParse(Screen);
```

```
    } catch {
```

```
        this.cmdScreen = eFromSCREEN.NORMAL;
```

```
    }
```

제너릭 예3- 참고만 할 것

```
public interface IMyControl
{
    #region >> Abstract Property

    string Name
    {
        get;
        set;
    }

    string DecimalPlaceFormat
    {
        get;
        set;
    }

    #endregion

    #region >> Abstract Method

    void SetMaxMIN<T>(T maxValue, T minValue) where T : struct;
    void MyControl_TextChanged(object sender, EventArgs e);

    #endregion
}
```

```
public class MyLabel : Label, IMyControl
{
    public decimal MAX...
    public decimal MIN...

    string _decimalPlaceFormat;
    public string DecimalPlaceFormat
    {
        get
        {
            return _decimalPlaceFormat;
        }
        set
        {
            _decimalPlaceFormat = value;
        }
    }

    System.Type TYPE;
    public System.Type MyType...

    bool _preventTextBoxEvents = false;

    public void SetMaxMIN<T>(T maxValue, T minValue) where T : struct
    {
        this.TYPE = typeof(T);
        MAX = (decimal)Convert.ChangeType(maxValue, typeof(decimal));
        MIN = (decimal)Convert.ChangeType(minValue, typeof(decimal));

        MyControl_TextChanged(this, EventArgs.Empty);
    }

    public void MyControl_TextChanged(object sender, EventArgs e)
    {
        if (_preventTextBoxEvents)
            return;
        else
            _preventTextBoxEvents = true;

        try {
            if (!string.IsNullOrEmpty(this.Text)) {
                decimal decValue = 0M;
                if (!Decimal.TryParse(this.Text, out decValue)) {
                    this.TYPE = typeof(string);
                    return;
                }

                if (decValue > MAX) {
                    this.Text = MAX.ToString(DecimalPlaceFormat);
                } else if (decValue < MIN) {
                    this.Text = MIN.ToString(DecimalPlaceFormat);
                } else {
                    this.Text = decValue.ToString(DecimalPlaceFormat);
                }
            }
        } catch (Exception ex) {
            throw ex;
        } finally {
            _preventTextBoxEvents = false;
        }
    }

    #region >> Constructor

    public MyLabel()...

    #endregion
}
```


제네릭 예4 - 12주차의 Subject class와 비교할 것

```
class Subject<T> where T : struct, IComparable<T>
{
    public static T MAX = (T)Convert.ChangeType(100, typeof(T));
    public static T MIN = (T)Convert.ChangeType(0, typeof(T));
    public const int WEEK = 8;

    string _title;
    public string Title
    {
        get
        {
            return _title;
        }
    }

    T _score;
    public T Score
    {
        get
        {
            return _score;
        }

        set
        {
            if (Comparer<T>.Default.Compare(value, MIN) >= 0 //value >= min
                && Comparer<T>.Default.Compare(value, MAX) <= 0 //value <= max
            )
            {
                _score = value;
            }
        }
    }

    ATTEND_TYPE[] _attendanceBook;

    //out 예제
    public bool GetAttendBook(int week, out ATTEND_TYPE state)
    {
        if (week < 1 || week > WEEK)
        {
            state = ATTEND_TYPE.EMPTY;
            return false;
        }
        else
        {
            state = _attendanceBook[week - 1];
            return true;
        }
    }
}
```

_score의 자료형을 다양하게
하고 싶을 때
(예, 정수형 vs. 실수형)

```
public bool SetAttendBook(int week, ATTEND_TYPE state)
{
    if (week < 1 || week > WEEK)
    {
        return false;
    }
    else
    {
        _attendanceBook[week - 1] = state;
        return true;
    }
}

public void AttendState(out int empty, out int attend, out int absence, out int late)
{
    empty = 0;
    attend = 0;
    absence = 0;
    late = 0;

    for (int i = 1; i <= Subject.WEEK; i++)
    {
        GetAttendBook(i, out ATTEND_TYPE state);
        if (state == ATTEND_TYPE.ATTEND)
            attend++;
        else if (state == ATTEND_TYPE.ABSENCE)
            absence++;
        else if (state == ATTEND_TYPE.LATE)
            late++;
        else
            empty++;
    }
}

public Subject(string title)
{
    this._title = title;
    this._score = MIN;
    this._attendanceBook = new ATTEND_TYPE[WEEK];
}
```

nullable 타입

```
7 namespace ConsoleApp18
8 {
9     class Subject
10    {
11        const double MAX = 100.0;
12        const double MIN = 0.0;
13
14        public string Title { get; set; }
15        private double? _score;
16        public double? Score
17        {
18            get
19            {
20                return _score;
21            }
22
23            set
24            {
25                if (value < MIN || MAX < value)
26                {
27                    _score = null;
28                }
29                else
30                {
31                    _score = value;
32                }
33            }
34        }
35
36        public Subject(string title)
37        {
38            Title = title;
39            _score = null;
40        }
41    }
42 }
```

자동구현 프로퍼티
(private 변수를 자동으로 생성하고,
get, set의 기본 내용 또한 자동으로
구현해준다.)

nullable 타입
(null값을 넣고, 비교할 수 있음)
double외에 null 값을 가질 수 있음

value == null ||
수식을 조건에 추가하는 것이 좋을 듯.
(첨부한 수업시간 코드에 있으니 참고)

Indexer / out

```

43 class RegSubjectList
44 {
45     List<Subject> subjects = new List<Subject>();
46
47     public Subject this[int index]
48     {
49         get
50         {
51             if (index < subjects.Count)
52             {
53                 return subjects[index];
54             }
55             else
56             {
57                 return null;
58             }
59         }
60     }
61
62     public int Count
63     {
64         get
65         {
66             return subjects.Count;
67         }
68     }
69
70     public bool RegSubject(string title)
71     {
72         subjects.Add(new Subject(title));
73         return true;
74     }
75

```

생성된 인스턴스 자체가 []를 사용할 수 있게 해줌.

```

76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114

```

```

public bool Avg(out double value)
{
    double sum = 0;
    int count = 0;
    bool result = false;
    value = .0;

    foreach (var sub in subjects)
    {
        if (sub.Score != null)
        {
            sum += (double)sub.Score;
            count++;
        }
    }

    if (count != 0)
    {
        value = sum / count;
        result = true;
    }

    return result;
}

public bool SetScore(string title, double score)
{
    bool result = false;
    for (int i = 0; i < subjects.Count; i++)
    {
        if (subjects[i].Title == title)
        {
            subjects[i].Score = score;
            result = true;
        }
        break;
    }
    return result;
}

```

출력 전용 매개 변수
반드시, 해당 메소드는
출력 전용 매개 변수에
값을 넣어주어야 함.

nullable 타입 체크

둘의 차이점은?

out / indexer / null 병합연산자

```

116 class Student
117 {
118     public string Name;
119     public RegSubjectList RegSubjects = new RegSubjectList();
120 }
121
122 class Program
123 {
124     static void Main(string[] args)
125     {
126         Student student = new Student();
127         student.Name = "김인하";
128         double avg = 0.0;
129         if(student.RegSubjects.Avg(out avg))
130         {
131             Console.WriteLine("현재 성적의 평균:" + avg);
132         }
133         else
134         {
135             Console.WriteLine("성적을 등록한 과목이 없습니다.");
136         }
137
138         student.RegSubjects.RegSubject("영어");
139         student.RegSubjects.RegSubject("국어");
140         student.RegSubjects.RegSubject("수학");
141
142         if (student.RegSubjects.Avg(out avg))
143         {
144             Console.WriteLine("현재 성적의 평균:" + avg);
145         }
146         else
147         {
148             Console.WriteLine("성적을 등록한 과목이 없습니다.");
149         }
150     }

```

```

151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171

```

```

student.RegSubjects.SetScore("영어", 100);
student.RegSubjects.SetScore("국어", 50);

if (student.RegSubjects.Avg(out avg))
{
    Console.WriteLine("현재 성적의 평균:" + avg);
}
else
{
    Console.WriteLine("성적을 등록한 과목이 없습니다.");
}

Console.WriteLine("등록 과목 현황");
for (int i=0; i < student.RegSubjects.Count; i++)
{
    Subject sub = student.RegSubjects[i];
    Console.WriteLine($"{sub.Title}-{sub.Score ?? 0}");
}

```

null 병합연산자
인덱서 구현한 것을 사용하는 예

성적을 등록한 과목이 없습니다.
 성적을 등록한 과목이 없습니다.
 현재 성적의 평균: 75
 등록 과목 현황
 영어-100
 국어-50
 수학-0

abstract / interface

```
class AClass
{
    public string Name;
    public void Print1()
    {
    }
    //public abstract void Print2();
    //public void Print3();
    //void Print3();
}
```

```
abstract class BClass
{
    public string Name;
    public void Print1()
    {
        Console.WriteLine("반드시 구현해야 한다0");
    }
    public abstract void Print2();
    //public void Print3();
    //void Print3();
}
```

```
interface CInterface
{
    //public string Name;
    //public void Print1()
    //{
    //}
    //}
    //public abstract void Print2();
    //public void Print3();
    void Print3();
}
```

```
class Program
{
    static void Main(string[] args)
    {
        AClass a = new AClass();
        BClass b = new BClass();
        CInterface c = new CInterface();
    }
}
```

```
class BClassDerived_1 : BClass
{
    public override void Print2()
    {
        Console.WriteLine("반드시 구현해야 한다1");
    }
}
```

```
class BClassDerived_2 : BClass
{
    public override void Print2()
    {
        Console.WriteLine("반드시 구현해야 한다2");
    }
}
```

```
class CInterfaceDerived1 : CInterface
{
    public void Print3()
    {
        Console.WriteLine("반드시 구현해야 한다3");
    }
}
```

```
class CInterfaceDerived2 : CInterface
{
    public void Print3()
    {
        Console.WriteLine("반드시 구현해야 한다4");
    }
}
```

```
class CInterfaceBClassDerived : BClass, CInterface
{
    public override void Print2()
    {
        Console.WriteLine("반드시 구현해야 한다5");
    }

    public void Print3()
    {
        Console.WriteLine("반드시 구현해야 한다6");
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        BClass b1 = new BClassDerived_1();
        BClass b2 = new BClassDerived_2();

        CInterface c1 = new CInterfaceDerived1();
        CInterface c2 = new CInterfaceDerived2();

        BClass m1 = new CInterfaceBClassDerived();
        CInterface m2 = new CInterfaceBClassDerived();
        CInterfaceBClassDerived m3 = new CInterfaceBClassDerived();

        b1.Print1();    b1.Print2();
        b2.Print1();    b2.Print2();

        c1.Print3();
        c2.Print3();

        m1.Print1();    m1.Print2();
        m2.Print3();
        m3.Print1();    m3.Print2();    m3.Print3();
    }
}
```

반드시	구현해야	한다0
반드시	구현해야	한다1
반드시	구현해야	한다0
반드시	구현해야	한다2
반드시	구현해야	한다3
반드시	구현해야	한다4
반드시	구현해야	한다0
반드시	구현해야	한다5
반드시	구현해야	한다6
반드시	구현해야	한다0
반드시	구현해야	한다5
반드시	구현해야	한다6

구조체

- 단순한 자료 저장이나 기능만 존재할 때 만듦
- 상속을 받을 수 없음, interfac를 구현할 수는 있음
- 매개변수가 없는 생성자를 명시적으로 만들 수 없음.
 - 인스턴스 변수등 필드 생성 시, 초기화 할 수 없음
 - 생성자 생성시, 모든 필드를 해당 생성자에서 초기화 해야함
- 지역변수가 구조체 타입인 경우 해당 데이터는 Stack에 생성
 - `int a = 0; string b = "0";`

```
struct Student
{
    public string Name;
    public int Grade;

    //public Student()
    //{
    //    //
    //}

    //public Student(string name)
    //{
    //    this.Name = Name;
    //}

    public Student(string name)
    {
        this.Name = name;
        this.Grade = 1;
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Student stu1; //new를 통해 생성할 필요없음
        stu1.Name = "김테스";
        stu1.Grade = 3;
        Console.WriteLine(stu1.Name + "/" + stu1.Grade);

        Student stu2 = new Student("김인하");
        Console.WriteLine(stu2.Name + "/" + stu2.Grade);
    }
}
```

```
김테스/3
김인하/1
```