7주차) 사칙연산 계산기 작성

토큰화 (Tokenization)

- 텍스트나 입력값을 프로그램 내에서 의미있는 단위 (토큰) 으로 나누는 과정.
- 컴파일러나 계산기, 자연어 처리 등 파싱 시스템의 기초 단위

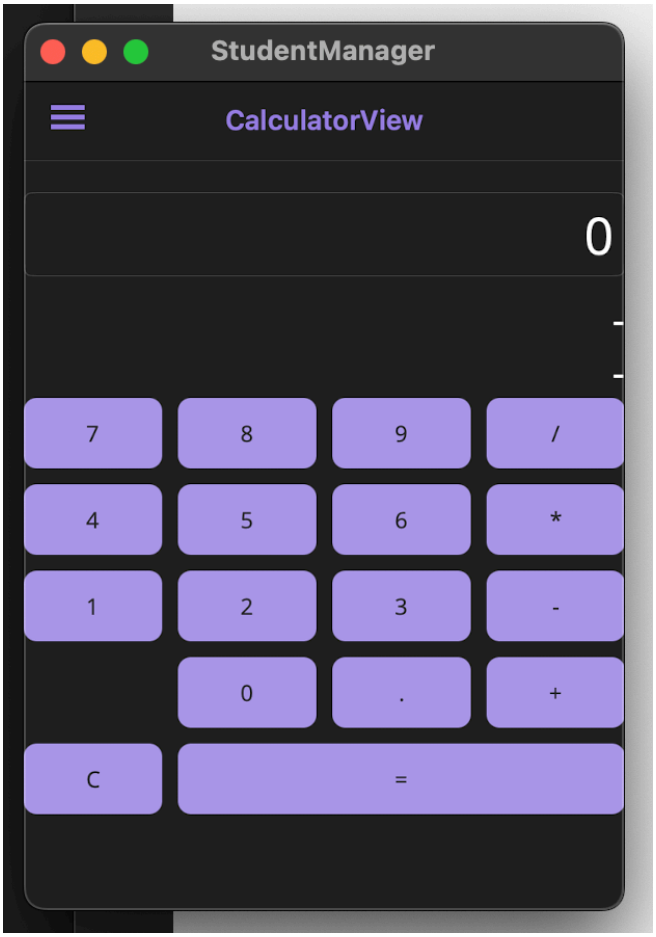
수식 계산에서의 토큰화 예)

입력 문자열 : 1 + 2 + 3 * 4 + 5

토큰 :

"1" , "2" , "3" , "4" , "5" - 숫자 토큰
"+" ,"*", - 연산자 토큰

실습 목표 - 숫자 , 연산자 버튼으로 동작하는 사칙연산 계산기 작성



Model > Calculator.cs

```
1   namespace StudentManager.Models;
2
    2 references
3   public class Calculator{
        58 references
4       public class Token{
5           [Flags]
            67 references
6           public enum eTokenType{
                5 references
7               ADD = 0b_0000_0000_0000_0001,
                5 references
8               SUB = 0b_0000_0000_0000_0010,
                5 references
9               DIV = 0b_0000_0000_0000_0100,
                5 references
10              MUL = 0b_0000_0000_0000_1000,
                13 references
11              NUM = 0b_0000_0000_0001_0000,
                3 references
12              DOT = 0b_0000_0000_0010_0000,
                2 references
13              CANCEL = 0b_0000_0000_0100_0000,
                2 references
14              SUBMIT = 0b_0000_0000_1000_0000,
                6 references
15              NONE = 0b_1000_0000_0000_0000
16          }
17
                1 reference
18          public static readonly Dictionary<string,eTokenType> STRING_TO_COMMANDTYPE =
19              new Dictionary<string, eTokenType>{
20                  {"*",eTokenType.MUL},
21                  {"+",eTokenType.ADD},
22                  {"-",eTokenType.SUB},
23                  {"/",eTokenType.DIV},
24                  {".",eTokenType.DOT},
25                  {"C",eTokenType.CANCEL},
26                  {"=",eTokenType.SUBMIT},
27              };
                2 references
28          private static eTokenType Operation =   eTokenType.ADD |
29                                                  eTokenType.SUB |
30                                                  eTokenType.DIV |
31                                                  eTokenType.MUL;
                1 reference
32          private static eTokenType OperatorOrDot = Operation | eTokenType.DOT;
```

가장 작은 단위인 Token 을 먼저 정의해봅니다

간단하게 토큰의 종류를 구분 할 수 있도록 플래그화 합니다.

연산 등의 특수입력들을 문자열 입력수준에서 구분 할 수 있도록 맵을 만들어 둡니다

토큰이 연산자인지 , 소수점인지 판단 할 수 있도록 플래그를 만듭니다.
이것은 맵으로도 구현 가능합니다

```
33      public float Value;
        9 references
34      public eTokenType TokenType = eTokenType.NONE;
        1 reference
35      public static bool IsOperation(eTokenType tType)
36      {
37          return (Operation & tType) == tType;
38      }
39
        2 references
40      public static bool IsOperatorOrDot(eTokenType tType)
41      {
42          return (OperatorOrDot & tType) == tType;
43      }
44
        2 references
45      public int OperatorPrecendence()
46      {
47          if(TokenType == eTokenType.DIV || TokenType == eTokenType.MUL)
48              return 2;
49          else if(TokenType == eTokenType.ADD || TokenType == eTokenType.SUB)
50              return 1;
51          return 0;
52      }
53  }
54
        1 reference
55  public delegate void OnCalculationComplete(string result);
        1 reference
56  public delegate void OnCalculationStatus(string status);
57
        3 references
58  public OnCalculationComplete OnCalculationCompleteAction;
        3 references
59  public OnCalculationStatus OnCalculationStatusAction;
60
        6 references
61  private string _numberBuffer = "";
        12 references
62  private List<Token> _inputToken = new List<Token>();
        5 references
63  private Token.eTokenType _lastProcessedToken = Token.eTokenType.NONE;
        5 references
64  private string _validStrings = "";
```

미리 정의해둔 플래그를 활용하여 토큰의 성질을 구분해 냅니다

연산자의 우선순위를 구분합니다

연산의 과정과 결과를 반환할 콜백을 작성합니다

```
66      public void ReceiveInput(string token)
67      {
68          Token.eTokenType currentToken = Token.eTokenType.NONE;
69          if(Token.STRING_TO_COMMANDTYPE.TryGetValue(token,out currentToken))
70          {
71              if(TokenIsValid(currentToken))
72              {
73                  if(Token.IsOperation(currentToken))
74                  {
75                      if(_lastProcessedToken == Token.eTokenType.NUM)
76                      {
77                          _inputToken.Add(new Token(){ Value = float.Parse(_numberBuffer),
78                                                       TokenType = Token.eTokenType.NUM });
79                          _numberBuffer = "";
80                      }
81
82                      _inputToken.Add(new Token(){TokenType = currentToken});
83                  }
84                  else if(currentToken == Token.eTokenType.DOT)
85                  {
86                      _numberBuffer += token;
87                  }
88                  else if(currentToken == Token.eTokenType.SUBMIT)
89                  {
90                      if(_lastProcessedToken == Token.eTokenType.NUM)
91                      {
92                          _inputToken.Add(new Token(){ Value = float.Parse(_numberBuffer),
93                                                       TokenType = Token.eTokenType.NUM });
94                          _numberBuffer = "";
95                      }
96
97                      OnCalculationCompleteAction(Calculate());
98                      return;
99                  }else if(currentToken == Token.eTokenType.CANCEL)
100                 {
101                     OnCalculationCompleteAction("");
102                     return;
103                 }
104             }
105             else
106             {
107                 OnCalculationStatusAction("invalid input ["+_validStrings+"]");
108                 return;
109             }
110         }
111         else
112         {
113             _numberBuffer += token;
114             currentToken = Token.eTokenType.NUM;
115         }
116
117         _lastProcessedToken = currentToken;
118         _validStrings += token;
119         OnCalculationStatusAction(_validStrings);
120     }
```

```
192     private string Calculate()
193     {
194         float result = 0;
195         Token.eTokenType lastOperator = Token.eTokenType.NONE;
196         for(int i = 0; i < _inputToken.Count ; i++)
197         {
198             if(_inputToken[i].TokenType == Token.eTokenType.NUM)
199             {
200                 if(lastOperator == Token.eTokenType.NONE)
201                     result = _inputToken[i].Value;
202                 else if(lastOperator == Token.eTokenType.ADD)
203                     result += _inputToken[i].Value;
204                 else if(lastOperator == Token.eTokenType.SUB)
205                     result -= _inputToken[i].Value;
206                 else if(lastOperator == Token.eTokenType.MUL)
207                     result *= _inputToken[i].Value;
208                 else if(lastOperator == Token.eTokenType.DIV)
209                     result /= _inputToken[i].Value;
210 
211                 lastOperator = Token.eTokenType.NUM;
212             }
213             else
214                 lastOperator = _inputToken[i].TokenType;
215         }
216         return _validStrings+" = "+result;
217     }
218 
        1 reference
219     private bool TokenIsValid(Token.eTokenType tType)
220     {
221         // 연달아서 연산자를 입력하거나, dot-연산자 및 그 반대는 연산에서 허용하지 않음
222         // 혹은 첫 입력이 연산자인 경우를 허용하지 않음
223         return !(Token.IsOperatorOrDot(tType) &&
224         (Token.IsOperatorOrDot(_lastProcessedToken) ||
225                 _lastProcessedToken == Token.eTokenType.NONE ));
226     }
227 }
```

CalculatorView.xaml.cs

```
1   using StudentManager.Models;
2
3   namespace StudentManager.Views;
4
    4 references
5   public partial class CalculatorView : ContentPage
6   {
        4 references
7       private Calculator _currentCalculation;
8
        0 references
9       public void OnButtonClicked(object sender,EventArgs args)
10      {
11          Button button = sender as Button;
12          string text = button.Text;
13          LastInput.Text = text;
14          _currentCalculation.ReceiveInput(text);
15      }
16
        2 references
17      private void InitCalculator()
18      {
19          _currentCalculation = new Calculator();
20          _currentCalculation.OnCalculationCompleteAction = OnCalculationComplete;
21          _currentCalculation.OnCalculationStatusAction = OnCalculationStatus;
22      }
23
        1 reference
24      private void OnCalculationComplete(string result)
25      {
26          ResultEntry.Text = result;
27          InitCalculator();
28      }
29
        1 reference
30      private void OnCalculationStatus(string status)
31      {
32          StatusText.Text = status;
33      }
34
        0 references
35      public CalculatorView()
36      {
37          InitializeComponent();
38          InitCalculator();
39      }
40  }
```

계산기 클래스에 입력을 전달합니다

계산기 클래스에서 전달받을 콜백 함수를 작성하고 등록합니다

CalculatorView.xaml

StudentManager > View > CalculatorView.xaml

```xaml
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="StudentManager.Views.CalculatorView"
             Title="CalculatorView">
    <VerticalStackLayout>
        <Entry x:Name="ResultEntry"
               FontSize="32"
               HorizontalTextAlignment="End"
               IsReadOnly="True"
               Text="0"
               Margin="0,20,0,10"/>
        <Label x:Name="LastInput" FontSize="24" Text="-" HorizontalTextAlignment ="End"/>
        <Label x:Name="StatusText" FontSize="24" Text="-" HorizontalTextAlignment ="End"/>
```

< Content Page 내에서 Class의 네임스페이스와 경로를 확인합니다 >