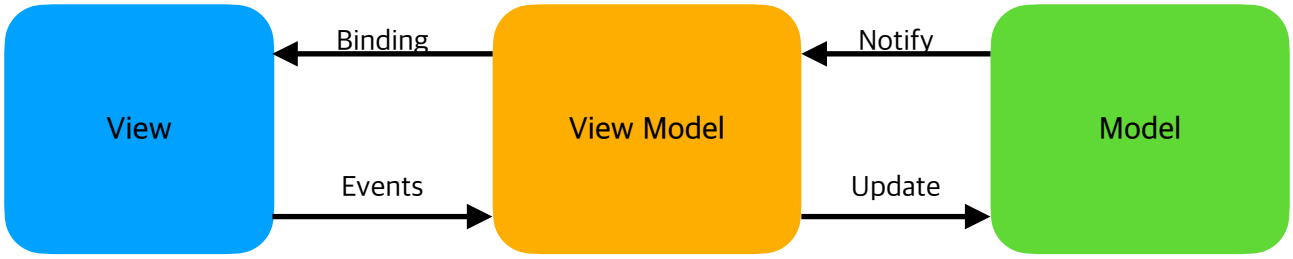
실습 보충 자료

디자인 패턴 - MVVM (Model - View - ViewModel)



* View 는 바인딩 된 데이터를 유저에게 보여줍니다.
* View 는 사용자의 입력을 ViewModel 에 전달합니다
* ViewModel 은 입력을 처리하여 Model에게 데이터를 요청합니다
* ViewModel 은 Model 로 부터 받은 데이터를 View에 전달하기 위한 처리 및 저장을 합니다

!. 디자인 패턴이란 것은 '법칙' 은 아닙니다. 객체지향 프로그래밍 분야에서 효율적이고 유지보수 하기 좋은 설계 방법으로 '검증' 된 방법들의 모음인 것입니다. 패턴을 응용해서 각각의 상황에 더 맞는 구현을 하시면 됩니다

예를 들어, 실습의 뷰모델(StudentViewModel)에서 데이터의 변경점을 뷰에 알리기 위해
INotifyPropertyChanged 인터페이스를 구현했습니다.

OnPropertyChanged 메소드를 통해 뷰에 데이터 변경을 알리면, 뷰에서 해당 데이터를 갱신하여 유저에게 보여줍니다.
이러한 객체간의 커뮤니케이션 방법들을 **행동 패턴 (Behavioral Pattern)** 이라 합니다.
그 중 INotifyPropertyChanged 와 같은 **구독(Subscribe) - 배포(Publish)** 로 구성되어 상태 변경을 알리는 패턴을 **Observer 패턴**이라 합니다.

키워드 async

```
1 reference
private async void ReadResources(string filename)
{
    // 데이터를 수용할 모델의 리스트 자료구조 초기화
    _dataSource = new List<Student>();

    // FireSystem.OpenAppPackageFileAsync 를 통해 패키지에 포함된 파일을 엽니다
    var stream = await FileSystem.OpenAppPackageFileAsync(filename);
    var reader = new StreamReader(stream);
```

* 비동기 프로그래밍을 지원하는 기능으로, 이 동안 프로그램이 멈추지 않고 다른 작업을 수행할 수 있도록 합니다.
* 비동기 메소드는 내부에서 await 키워드를 사용하여 비동기 작업을 실행합니다.
    !. Task 에 관해서는 나중 실습에서 다룰 예정입니다.

네임스페이스 namespace

```
using StudentManager.Models;
using System.ComponentModel;

namespace StudentManager.ViewModels;

8 references
public class StudentViewModel : INotifyPropertyChanged{
```

클래스나 인터페이스 등의 정의를 그룹화 하여 관리하는 개념입니다.
하나의 큰 프로그램 내에서 같은 이름을 가질 수 있는 다른 클래스나 인터페이스들과 충돌하지 않도록 해 줍니다

System.Collection

C# 에서 사용되는 자료구조 라이브러리 입니다. 그 중에서 주로 언급되는 것은 제네릭 / 비 제네릭 입니다.

System.Collection.Generic

```
3 references
public List<StudentViewModel> StudentAsList {get;private set;}
4 references
private List<Student> _dataSource;
```

컴파일 타임에 명시된 데이터 타입 자료구조를 포함한 라이브러리 입니다.
비 제네릭 콜렉션은 c# 의 기본 타입인 object 타입으로 데이터를 저장하므로 사용상 편의 (* c#의 모든 타입은 object 타입으로 캐스팅이 가능하므로) 가 있을 수 있으나 런타임에 타입 미스매치 오류나 타입 변환 오버헤드가 있을 수 있으므로, 가능한 시나리오에서는 제네릭 콜렉션을 사용하도록 합니다.

실습에서 리스트 자료구조로 사용한 List<T> 는 제네릭 콜렉션입니다.
* 리스트란 순서를 가지며, 인덱스로 값에 접근 가능하고, 동적으로 크기 조정이 가능한 자료구조 입니다.

키워드 event

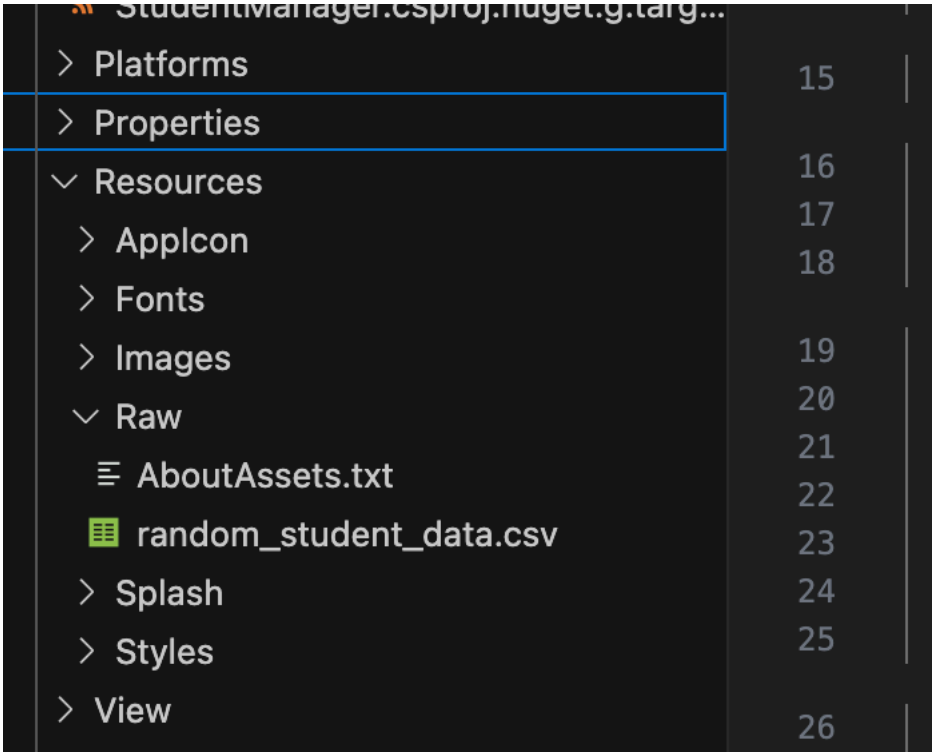
```
1 reference
public event PropertyChangedEventHandler? PropertyChanged;

1 reference
public void OnPropertyChanged(string propertyName = null)
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
```

이벤트 키워드는 += 연산자를 통해 메소드를 할당 가능하며,
해당 이벤트 함수가 호출 될 때 할당된 모든 메소드를 호출합니다.

실습 수행간 FAQ

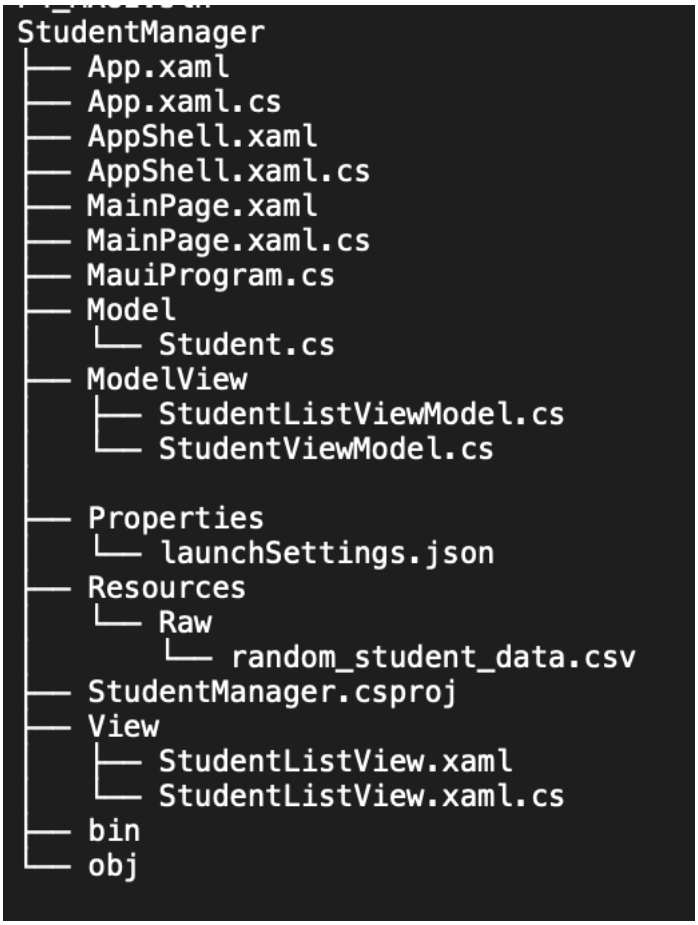
* 제공된 csv 파일은 Resources / Raw 폴더에 위치 시켜 주세요



* xaml 파일 내에서 본인이 작성한 네임스페이스와 일치하는지 확인

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:viewmodels="clr-namespace:StudentManager.ViewModels"
             x:Class="StudentManager.Views.StudentListView"
             Title="Student List View"
             x:DataType="viewmodels:StudentListViewModel">
    <VerticalStackLayout Padding="10">
        <CollectionView ItemsSource="{Binding Students}" HeightRequest="500"
            x:Name="StudentCollectionView">
            <CollectionView.ItemTemplate>
                <DataTemplate x:DataType="viewmodels:StudentViewModel">
```

* 실습 프로젝트의 파일 계층은 아래와 같습니다



* 실습 진행간 작성한 코드 내역은 아래와 같습니다 (자료실에 제공된 파일)