디자인 패턴과 콜백

팩토리 패턴 (Factory Pattern)

- 객체를 코드 사용측에서 직접 생성하지 않음으로써
결합도 (사용 코드-클래스 에서의 의존성) 를 낮추는 방법
- 객체가 생성되는 시점은 일반적으로 코드의 수정이 많은 부분입니다.
이 부분을 중간 객체가 대신해준다면 사용측 코드의 수정을 줄일 수 있습니다
- 팩토리 패턴을 활용하면, 인출되는 객체가 바뀌더라도 대응이 유연해집니다

싱글톤 패턴 (Singleton Pattern)

- 특정한 클래스의 인스턴스가 해당 프로그램 내에서 한개만 존재해야할 때
- 해당 클래스의 인스턴스가 프로그램 전역에서 사용되어야 할 때
- 동작하는 내용이 여러 인스턴스에 걸쳐 중복 동작하면 안되는 경우
> (네트워크, 파일 관리, 렌더링 등) 에 유용합니다.

```
8   public class StudentFactory{
        3 references
9       private static StudentFactory _instance;
10
        1 reference
11      public static StudentFactory Instance{
12          get
13          {
14              if(_instance == null)
15              {
16                  _instance = new StudentFactory();
17              }
18
19              return _instance;
20          }
21      }
22
23
        1 reference
24      public Student GetStudent(string name,string id,float score)
25      {
26          return new StudentBuilder().SetName(name).SetID(id).SetScore(score).Build();
27      }
28
        1 reference
29      public Student GetStudentFromData(string rawString)
30      {
31          string[] tokens = rawString.Split(",");
32          float score = 0;
33          if(float.TryParse(tokens[2],out float parsed))
34              score = parsed;
35
36          return GetStudent(tokens[0],tokens[1],score);
37      }
38  }
```

< 실습 프로젝트의 Student 객체를 대신 생성해주는 팩토리 싱글톤 >

빌더 패턴 (Builder Pattern)

- 객체의 생성 시점에 코드 변경이 많을것으로 예상되는 경우 ,
초기화 할 멤버 변수가 개발 과정에 따라 추가-제거가 빈번하게 일어나는 경우,
초기화 해야 할 내용이 선택적인 경우 불필요한 코드의 작성이나 수정을 줄일 수 있습니다.

```
7   public class StudentBuilder{
        5 references
8       private Student _currentBuildContext;
9
        1 reference
10      public StudentBuilder()
11      {
12          _currentBuildContext = new Student();
13      }
        1 reference
14      public StudentBuilder SetName(string name)
15      {
16          _currentBuildContext.Name = name;
17          return this;
18      }
19
        1 reference
20      public StudentBuilder SetID(string id)
21      {
22          _currentBuildContext.ID = id;
23          return this;
24      }
25
        1 reference
26      public StudentBuilder SetScore(float score)
27      {
28          _currentBuildContext.Score = score;
29          return this;
30      }
31
32      // build 함수를 통해 Student 인스턴스에 대한 변수 설정을 마무리하고
33      // 생성된 Student 인스턴스를 반환합니다
        1 reference
34      public Student Build()
35      {
36          return _currentBuildContext;
37      }
38  }
```

< 실습 프로젝트에서 Student 객체의 초기화를 담당하는 빌더>

* 단, 일반적으로 싱글톤 패턴으로 설계하는 경우 쉽게 해당 클래스의 의존도가 높아지는 상황이 발생합니다.
그러므로 필요할 때만 사용하고, 너무 많은 기능을 담지 않도록 유의하며 설계하도록 합니다.

* 빌더 패턴은 초기화 단계의 복잡도가 높을것이라 예상되거나 , 생성 후 추가적인 프로세스에 의해 특정 시점까지 객체를 보호해야 할 필요가 있을 때만 선택적으로 활용하도록 합니다.

콜백

- 메소드의 호출 시점을 해당 객체가 결정 할 수 없을 때 (UI 입력에 의한 호출 , 다른 메소드의 작업 완료 등), 다른 함수의 매개변수로써 전달해 필요한 시점 (이벤트의 발생) 에 호출되는 함수
- C# 에서는 대리자(delegate 키워드) 로 함수의 시그니처 (매개변수의 타입과 갯수) 를 정의하여 같은 시그니처를 가진 함수의 참조를 정의 할 수 있습니다.
- C# 에서는 event 키워드를 통해 대리자의 구독을 정의 할 수 있습니다.

#region - #endregion

- 가독성 측면에서 의미단위로 구획을 만들기 위함입니다.
- 기술적인 의미는 없습니다

기존 StudentListViewModel 에서 하던 데이터 초기화 기능을 StudentFactory 로 가져오면서, 대리자를 선언하고 데이터 초기화 완료시 콜백 하도록 변경 해 봅니다

```
12  #region Callback
        1 reference
13      public delegate void OnDataInitialized(List<Student> container);
14  #endregion
15
46      public async Task InitData(OnDataInitialized onInit,string filename)
47      {
48          // 데이터를 수용할 모델의 리스트 자료구조 초기화
49          List<Student> container = new List<Student>();
50          ReadResources(container,filename);
51          onInit.Invoke(container);
52      }
53
        1 reference
54      private async void ReadResources(List<Student> container,string filename)
55      {
56          // FireSystem.OpenAppPackageFileAsync 를 통해 패키지에 포함된 파일을 엽니다
57          var stream = await FileSystem.OpenAppPackageFileAsync(filename);
58          var reader = new StreamReader(stream);
59
60          // 텍스트 스트림은 한줄씩 읽거나 한번에 모두 읽을 수 있습니다.
61          // 우리가 읽는 csv 포맷은 데이터가 line by line 으로 이루어져 있기 때문에
62          // 편의상 루프에서 한 문자열 라인씩 읽도록 합니다.
63          string rawDataLine = reader.ReadLine();
64
65          while(!string.IsNullOrEmpty(rawDataLine))
66          {
67              container.Add( StudentFactory.Instance.GetStudentFromData(rawDataLine) );
68              rawDataLine = reader.ReadLine();
69          }
70      }
```

< StudentFactory 에 추가된 대리자 선언과 데이터 초기화 코드 >