Team Project - OOAD

Timetable

(instructor :: 유준범 교수님)

Konkuk univ. dept of CSE

Team 4

200611517 정훈섭

200711420 권준수

200711448 오희수

200710118 유희찬

Team Project - Timetable development

Contents

Introduction

Decomposition Description

Dependency Description

Interface Description

Detailed Description

Introduction

Purpose

Software 개발 단계 중 analysis 단계를 지나 design 단계에 왔습니다. design 단계에서는 analysis 단계를 통해 만든 use case와 수집한 requirement를 이용하여 실제 구현에 앞서 시스템의 구조를 설계하는데 있습니다.

design 단계에서는 software의 재사용, 확장을 위해 OOP 기법 및 design pattern을 이용하게 됩니다. 견고하고 유연한 software는 잘 짜여진 design으로부터 시작하므로 결국 design의 목적 역시 견고하고 유연한 software를 작성하는데 있습니다. 특히나 이번 TimeTable 프로그램의 경우 범위 자체를 작게 잡은 관계로 훗날에 확장 및 수정의 가능성이 높습니다. 이를 염두에 두고 이번 design 단계에서는 **훗날 이뤄질 수 있는 변경에 대해 유연하게 작성**하도록 합니다.

Scope

SDD(Software Desgin Description)을 제작하기에 앞서 범위를 설정합니다. Class diagram을 작성할 때는 우선 GUI 부분은 제외하고 그리나, MFC을 사용하는 특성상 일부 MFC class가 들어가야 class diagram이 그려지는 경우 MFC class를 추가합니다. 대신 내부적으로는 class 간의 관계를 찾는데 필요한 attribute만 표기하였습니다.

OOD(Object Orient Design)를 따라 설계하므로, 설계 과정에서 SRP, DIP 등 OOP 법칙이 사용 됩니다.

또한 부가적인 일부 기능(종합강의시간표, 수강신청창 보기)에 대해서는 인터넷에 연결되어 있어야 하나, 주 기능인 강의 등록, 일정 등록 등의 기능은 인터넷 연결 없이도 실행이 되도록 설계합니다.

Reference

Head first OOAD - 브랫 맥리프란 외 (O'REILY)
UML 객체지향 분석 · 설계 - 조완수 (홍릉과학출판사)
Software Engineering (8th) - Ian Sommerville (Addison-Wesley)

Decomposition description

Analysis 단계에서 작성한 use case로부터 구현을 위한 class 후보들을 찾아냅니다. Use case에서 명사들은 시스템에서 작성해야할 class들입니다.

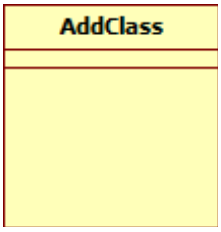
Use Case

빨간색 : class 후보

1. 사용자가 **프로그램**을 실행 한다
2. 사용자는 저장된 **시간표**를 불러오거나 새로이 작성(초기화)한다.
 2-1 사용자가 **시간표**를 새로이 작성한다.
  2-1-1 사용자는 등록하고 싶은 **시간의 칸**을 선택한다.(right click)
   2-1-1-1 사용자가 **선택한 칸**에 비어있다.
    2-1-1-1-1 '**강의** 등록' 을 선택한다.
     2-1-1-1-1-1 강의명을 **메뉴**에서 선택한다.(**팝업창**)
     2-1-1-1-1-2 입력을 다했으면 '확인 '버튼을 누른다.
     2-1-1-1-2-3 시간표에 **강의**가 추가된다.
    2-1-1-1-2 '**일정** 등록' 을 선택한다.
     2-1-1-1-2-1 활동명, 장소, 시간을 입력한다.(**팝업창**)
     2-1-1-1-2-2 입력을 다했으면 '확인 '버튼을 누른다.
     2-1-1-1-2-3 시간표에 일정이 추가된다.
   2-1-2 사용자가 **선택한 칸**에 일정이 있다.
    2-1-2-1 '변경' 을 선택한다.
    2-1-2-2 '삭제' 를 선택한다.
    2-1-2-3 '강의 정보' 를 선택한다.
     2-1-2-3-1 선택한 강의에 대한 **강의 정보 창**이 뜬다.(**인터넷 팝업창**)
 2-2 사용자가 저장된 **시간표**를 불러온다.
  2-2-1 위의 내용과 동일한 작업 반복
3. 사용자가 **시간표**를 저장한다.
4. 사용자가 '수강 신청' 버튼을 클릭한다.
 4-1 건국대학교 **수강신청 팝업창(인터넷 팝업창)**이 뜬다.
5. 사용자가 '전체 강의 검색' 버튼을 클릭한다.
 5-1 건국대학교 **종합강의시간표(인터넷 팝업창)**가 뜬다.
6. 사용자가 **시간표**를 종료한다.

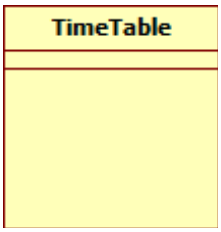
위 use case에서 찾은 명사들을 중복을 제외하고 나열해 봅니다. 이 중에서 우리가 집중해야할 명사들은 "**시간표, (선택한) 칸, 팝업창(강의 등록, 일정 등록), 강의, 일정** 이 나옵니다. 강의 정보 팝업창, 수강신청 팝업창, 종합강의시간표 팝업창은 바로 인터넷 브라우저를 띄우게 되므로 이것들은 제외하였습니다.

이제 이 6개 명사를 class 후보로 두고, class 틀만 그려봅니다.



AddClass :: 강의 등록 팝업창

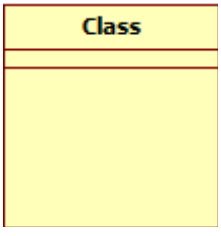
AddPrivaate :: 개인 일정 등록 팝업창



TimeTable :: 시간표

Time :: (시간) 칸

Private :: 개인 일정

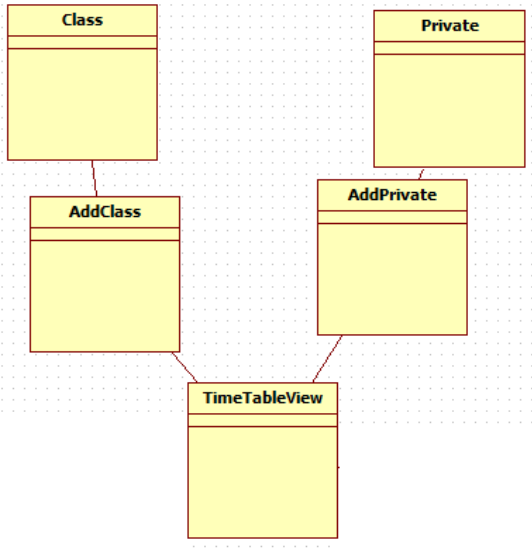


Class :: 강의명, 강의실, 교수명, 시간 정보를 가지는 클래스

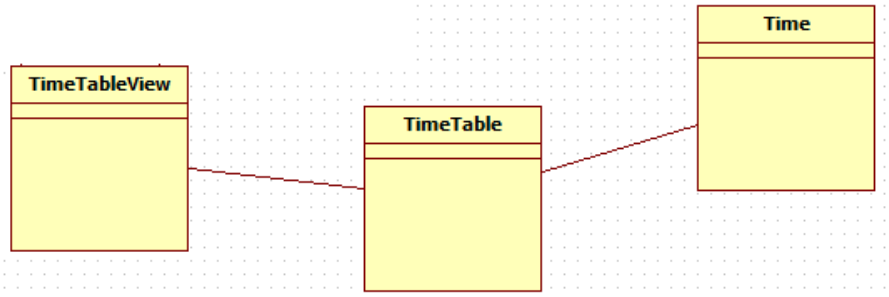
Decomposition Description에서 시스템 구현을 위한 큼직한 class들을 추려냈습니다. 이 class들을 바탕으로 필요한 class들을 새로이 추가하고, class 들의 연관 관계를 알아내 class diagram을 완성해갑니다.

Dependency Description

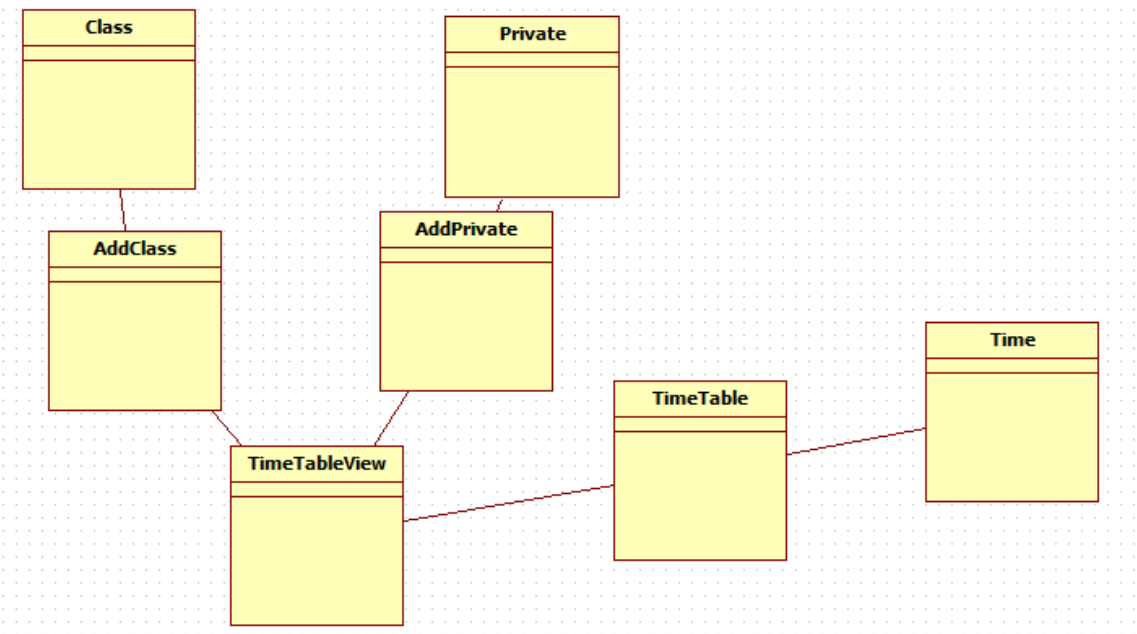
위의 Decomposition description에서 만든 class들을 찬찬히 살펴봅니다. 지금은 각 class들이 아무런 연관없이 따로 떨어져 따로따로 있지만, 전체적인 시스템은 class들 간에 상속, 이용(use) 등의 상호작용을 통하여 동작하게 됩니다. Actor가 행하는 동작(강의 등록, 개인 일정 등록 등)을 통하여 각 class 들 간의 관계를 알아내 봅니다.



AddPrivate, AddClass는 actor인 사용자가 시간표를 클릭을 했을 때 나타나는 팝업창을 나타내는 class입니다. 결국 사용자의 마우스 이벤트는 TimeTableView class에서 받고, 여기서 이벤트에 맞는 method를 호출하게 되고, method는 TimeTableVew class 안의 각 AddPrivate, AddClass class의 instance를 사용(use)하여 팝업창을 띄우게 됩니다. 따라서 TimeTableView class와 이들 2개의 팝업창 class인 AddPrivate, AddClass를 사용(use)하는 관계에 있습니다. 또한 마찬가지로 AddPrivate와 AddClass class는 각각 Private, Class class를 사용(use)하는 관계에 있습니다.



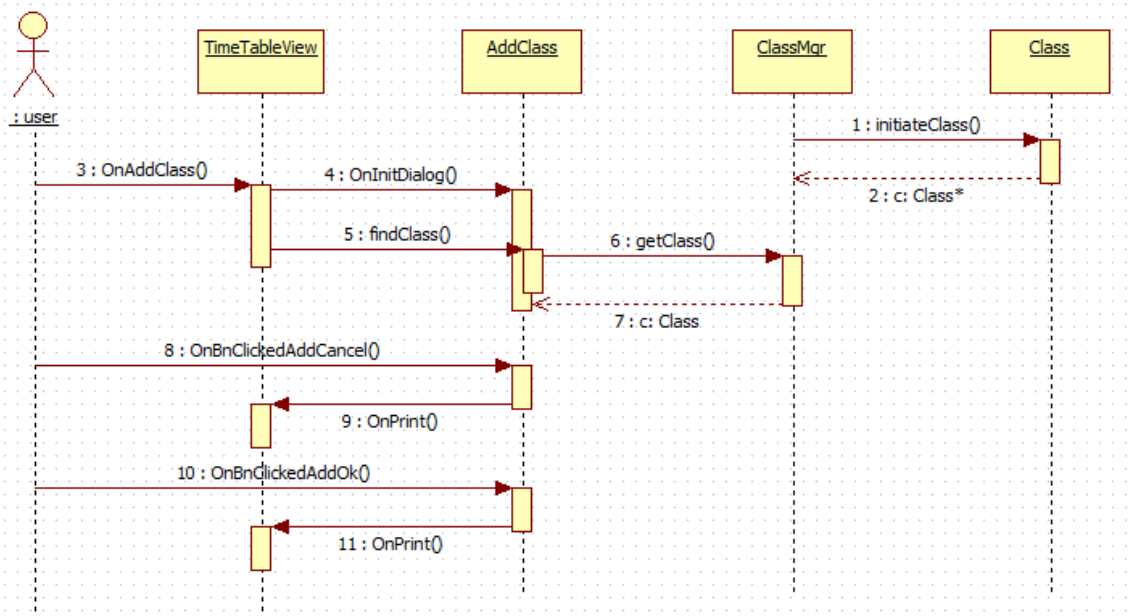
TimeTableView에서는 TimeTable instance를 초기화하고, actor가 강의 등을 팝업창을 통해 입력을 받으면 그 입력값을 TimeTable instance로 보내 actor가 입력된 값으로 변경시키는 플랫폼 역할을 하게 됩니다. 또한 프로그램을 실행했을 때 보이게 될 각 '칸' 들은 Time class들의 instance들로 이루어지게 됩니다. 이 각 칸들을 TimeTable class에서 초기화를 하게 되고 각 칸들에 강의 입력 등의 변경되는 사항도 TimeTable class 내에서 수행되게 됩니다. 따라서 sequnece diagram에서 보이는 것과 같이 TimeTableView class는 TimeTable class의 instance를 가지며, TimeTable class는 Time class instance를 배열로 갖게 됩니다. 따라서 각각 사용(use) 관계에 놓이게 됩니다.



Interface Description

Dependency Description에서 각 class 간의 관계를 그려보았습니다. 그런데 여기서 use case나 requirement description에서는 찾을 수 없는 클래스 몇 개를 더 추가하면 프로그램이 훨씬 더 유연해 질 것 같습니다. 또한 sequence diagram을 통하여 actor가 수행하는 행동과 시스템 내부 작동 내용을 보면서 새로이 class들을 추가합니다.

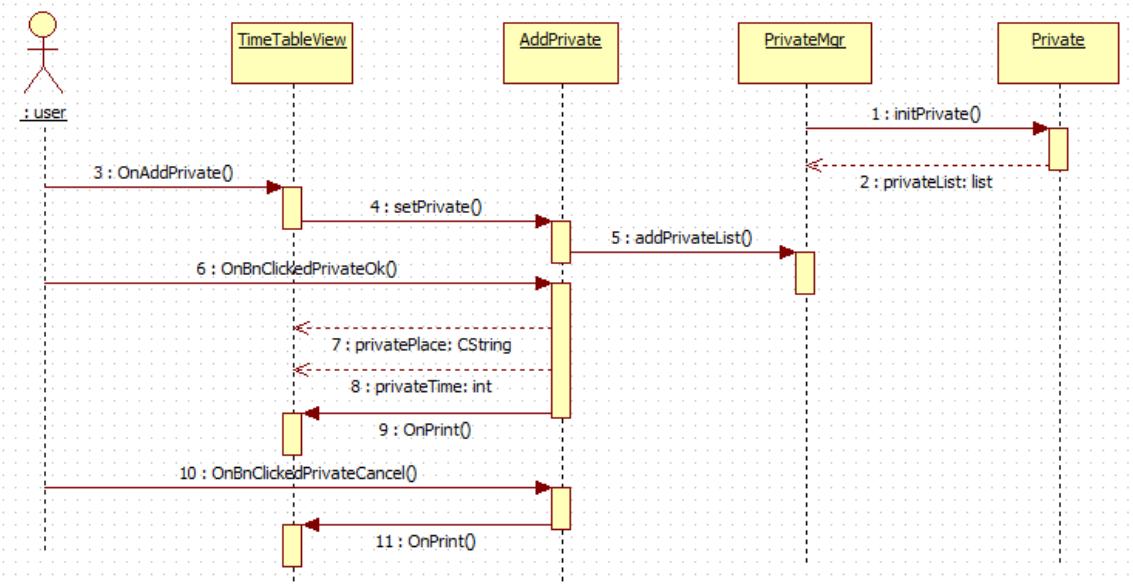
AddClass, ClassMgr, Class

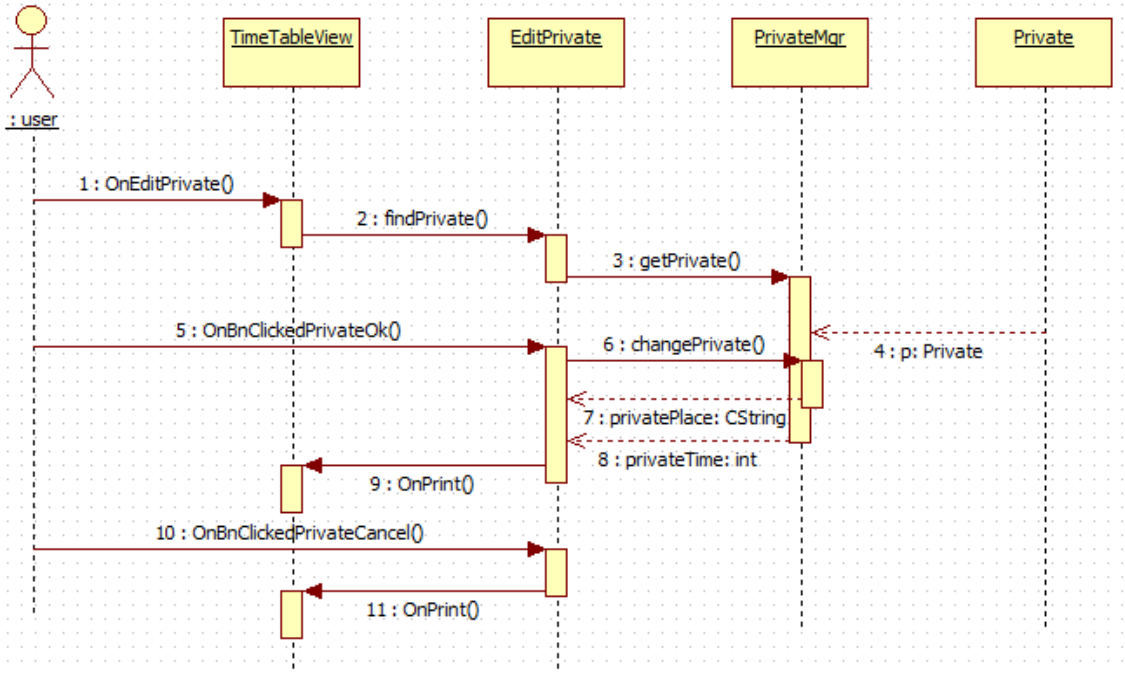


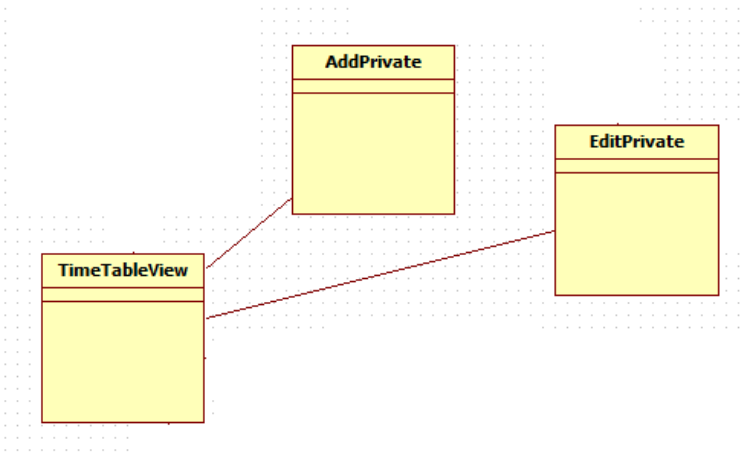
현재까지는 Class class가 강의 정보를 담는 것 외에 자체적으로 초기화를 하거나 AddClass에서 초기화를 시켜주어야 합니다. 하지만 이는 한 class는 한 가지 일만 해야하는 OOP 법칙을 위반하는 것으로 새로이 설계를 해주어야 합니다. Class class 배열을 초기화, 강의 데이터 저장 등 변경을 관리해 주는 class인 ClassMgr을 새로이 만들어 담당하게 합니다. 이로써 AddClass는 사용자로부터 값을 받는 기능, Class는 강의 내용을 제공하는 기능, ClassMgr은 Class class 배열의 (변경을) 관리하는 기능만을 담당하게 되어 OOP 원칙인 SRP(단일 책임의 원리)를 지키게 됩니다.

AddPrivate, EditPrivate

위에서처럼 Private class 경우도 마찬가지로 Private을 관리하는 PrivateMgr class를 만들어 SRP 원칙을 지키게 합니다. Dependency Description에서는 AddPrivate class에서 개인일정을 수정하는 method까지 포함되게 설계되었습니다. 이는 한 class가 일정 수정과 일정 추가라는 2가지 행동을 하는 것이므로 역할을 나누어 AddPrivate, EditPrivate class를 따로 구분하여 만듭니다. 이 역시 SRP(단일 책임의 원리)를 지키게 됩니다.







PrivateDialog

AddPrivate과 EditPrivate의 경우, 공통적으로 강의/일정의 이름, 시간, 장소가 들어가고 입력받아 data로 저장해야 합니다. 또한 기본적으로 새로운 data를 받기위해 각 class들의 변수를 초기화해야 하는 점, 더 나아가 UI에서 마우스 이벤트 처리 등을 똑같이 하므로 공통적인 부분을 묶어 PrivateDialog 라는 abstract class를 만들어 super class로 두고, EditPrivate, AddPriavate는 이를 상속받아 사용하게 합니다.

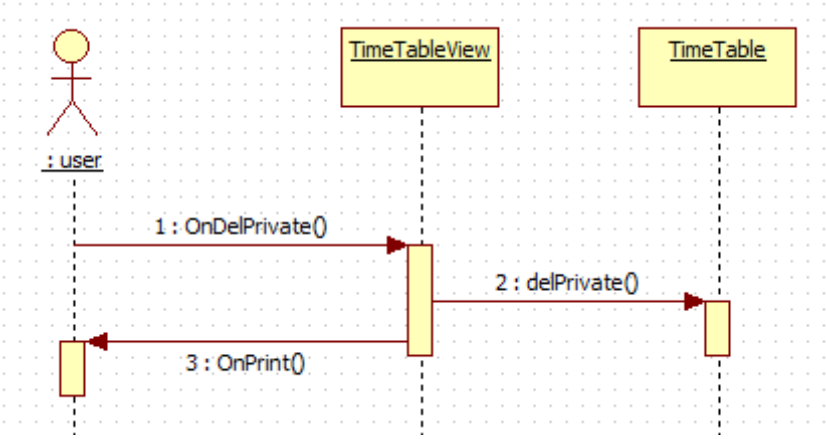
이렇게 하면, super class의 코드를 수정하지 않으면 subclass에서도 역시 변경되지 않습니다. 그리고 상속을 받은 sub class에서는 method를 오버라이드하여 각 class에 맞는 형태로 확장해서 사용할 수 있도록 해줄 수 있습니다. 이렇게 super class에서 변경을 원하지 않은 method를 지키면서도 하위 subclass에 맞게 새로이 확장하여 사용하여 사용할 수 있습니다.

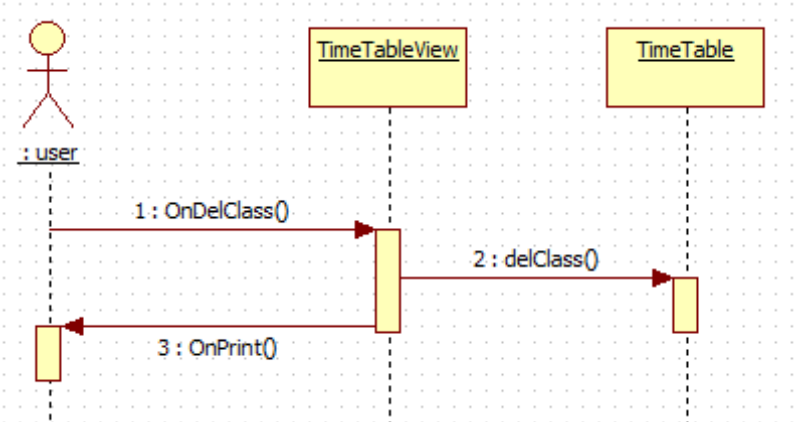
또한, superclass인 PrivateDialog class에서 하위 subclass (AddPrivate, EditPrivate)의 instance를 생성하여도 각 instance에 맞게 무리 없이 작동을 합니다.

상속을 사용하면 강의나 개인 일정 외에 아예 '학원 일정' 같이 새로운 일정을 따로 만들어 사용하기에도 편할 뿐더러 OOP 원칙인 OCP, LSP를 충족하게 되었습니다.

delePrivate, delClass

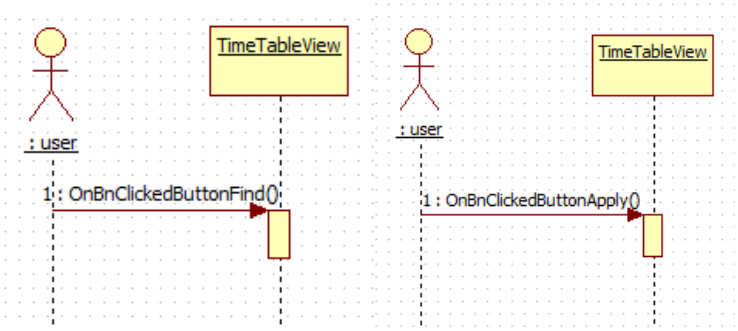
Actor가 등록된 시간표에서 일정이나 강의를 지웁니다. TimeTableView class에서 TimeTable class의 instance의 method를 호출하여 강의 및 일정 data를 삭제하게 됩니다.





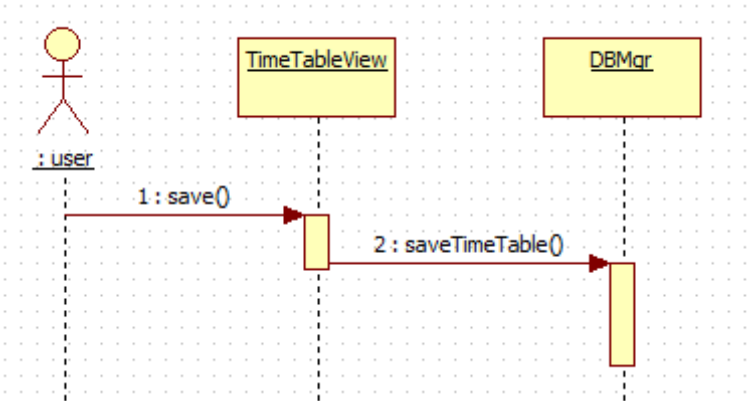
수강신청, 종합강의시간표

아까 class 후보에서 탈락되었던 (인터넷 팝업창) 수강신청, 종합강의시간표입니다. 따로 class를 구현하지 않고, TimeTableView class에서 method를 이용하여 직접 인터넷 브라우저 창을 띄우게 됩니다.

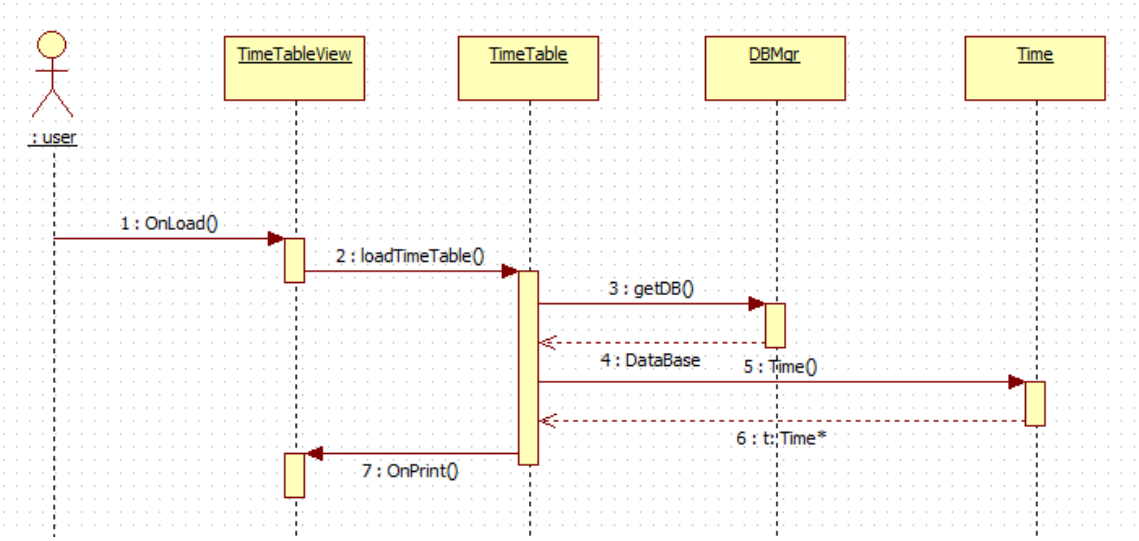


save, load

저장 및 불러오기 기능을 위해 데이터베이스를 사용하고, 이를 관리하기 위한 class인 DBMgr class를 만듭니다. DBMgr class는 Time class가 가진 data를 저장하게 되므로 DBMgr의 method를 호출할 때 현재 TimeTable class가 가진 Time class 배열을 넘겨줍니다.



또한, 불러오기를 할 때는 DBMgr class로부터 data를 불러받아서 TimeTable class 내의 Time class 배열에 넣어야 합니다.



Detailed design description

class diagram을 완성하는 마지막 단계로 각 class의 '안' 을 채워봅니다. 첫 번째 decomposition description에서 class 후보를 찾아내려고 use case에서 명사를 찾아냈듯이 이번에는 class들의 attribute들을 찾기 위하여 use case에서 '동사' 를 찾아봅니다.

Use Case

파란색 : method 후보

1. 사용자가 프로그램을 **실행 한다**
2. 사용자는 저장된 시간표를 **불러오거나** 새로이 **작성(초기화)한다**.
 2-1 사용자가 시간표를 새로이 작성한다.
  2-1-1 사용자는 등록하고 싶은 시간의 칸을 **선택한다**.(right click)
   2-1-1-1 사용자가 선택한 칸에 비어있다.
    2-1-1-1-1 **'강의 등록' 을 선택한다**.
     2-1-1-1-1-1 강의명, 교수명, 강의실, 시간을 메뉴에서 **선택한다**.(팝업창)
     2-1-1-1-1-2 입력을 다했으면 **'확인 '버튼을 누른다**.
     2-1-1-1-2-3 시간표에 강의가 **추가된다.**
    2-1-1-1-2 **'일정 등록' 을 선택한다**.
     2-1-1-1-2-1 활동명, 장소, 시간을 **입력한다**.(팝업창)
     2-1-1-1-2-2 입력을 다했으면 **'확인 '버튼을 누른다**.
     2-1-1-1-2-3 시간표에 일정이 **추가된다.**
  2-1-2 사용자가 선택한 칸에 일정이 있다.
   2-1-2-1 '변경' 을 **선택한다.**
   2-1-2-2 '삭제' 를 **선택한다.**
   2-1-2-3 '강의 정보' 를 **선택한다.**
    2-1-2-3-1 선택한 강의에 대한 강의 정보 창이 뜬다.(팝업창)
 2-2 사용자가 저장된 시간표를 **불러온다.**
  2-2-1 위의 내용과 동일한 작업 반복
3. 사용자가 시간표를 **저장한다.**
4. 사용자가 '수강 신청' **버튼을 클릭한다.**
 4-1 건국대학교 수강신청 팝업창(인터넷 팝업창)이 뜬다.
5. 사용자가 '전체 강의 검색' **버튼을 클릭한다.**
 5-1 건국대학교 종합강의시간표(인터넷 팝업창)가 뜬다.
6. 사용자가 시간표를 **종료한다.**

Use case에서 actor가 취하는 행동(동사)은 시스템에서 객체의 method가 됩니다. 각 행동들과 앞서 찾아낸 class들 중 행동의 주체가 되는 class들을 짝 맞춰 봅니다.

TimeTable

실행한다. 저장한다. 불러온다. 강의 등록을 선택한다.
일정 등록을 선택한다. 변경을 선택한다. 삭제를 선택한다.
강의정보를 선택한다. 화면에 출력한다 칸을 선택한다.

Time

일정이 추가된다.

AddClass

강의명을 메뉴에서 선택한다.
확인버튼을 누른다.
취소버튼을 누른다.

AddPrivate

일정 이름, 시간을 입력한다.
확인버튼을 누른다.
취소 버튼을 누른다.

Class

db로부터 data를 불러와
변수에 대입한다.

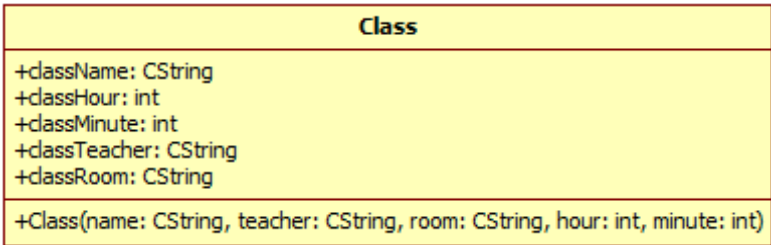
Private

db로부터 data를 불러와
변수에 대입한다.

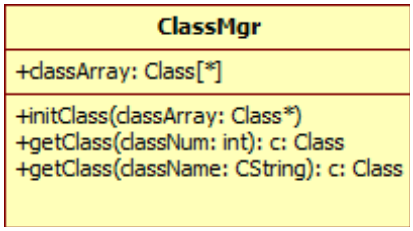
보다 구체화하고 class들이 실제 시스템에서 작동하는 모습을 앞서 그렸던 sequence diagram에서 보고 위에서 정리한 내용을 토대로 class의 method를 결정합니다.

이렇게 해서 각 class 들의 attribute, operation을 채웠습니다.

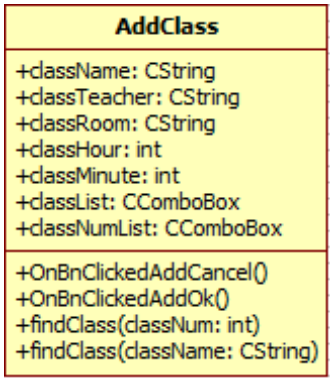
Class



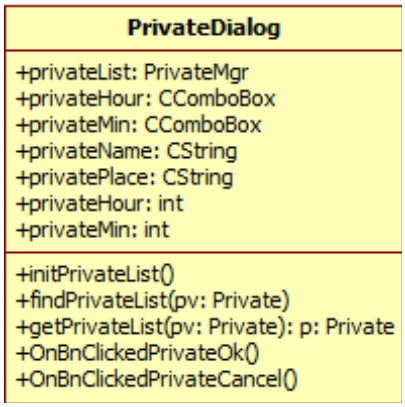
ClassMgr



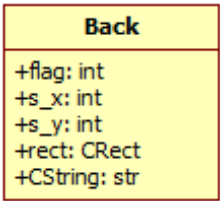
AddClass



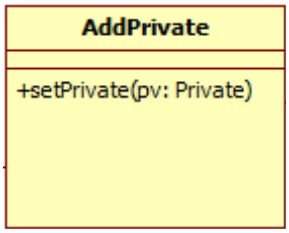
PrivateDialog



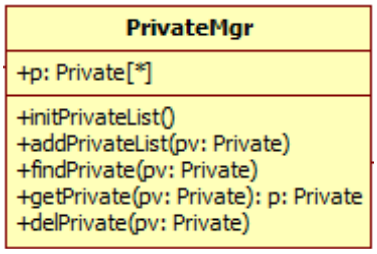
Back



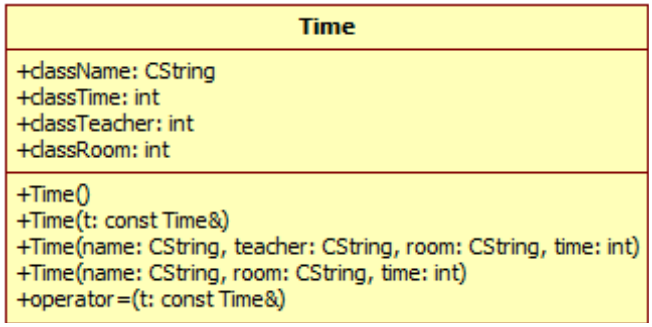
AddPrivate



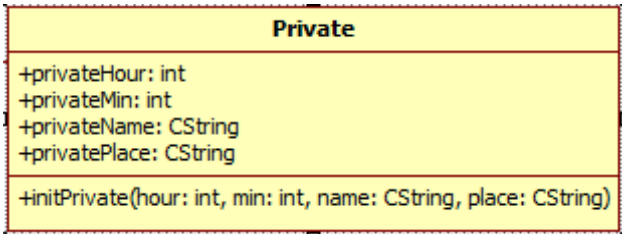
PrivateMgr



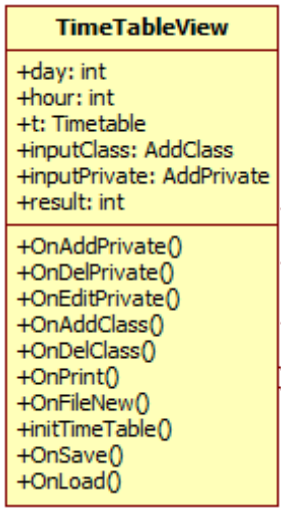
Time



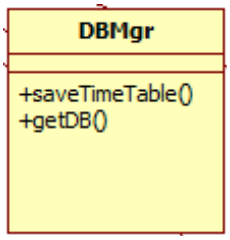
Private



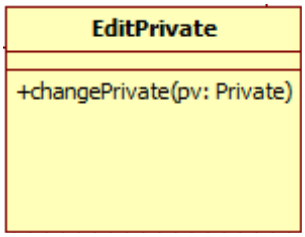
TimeTableView



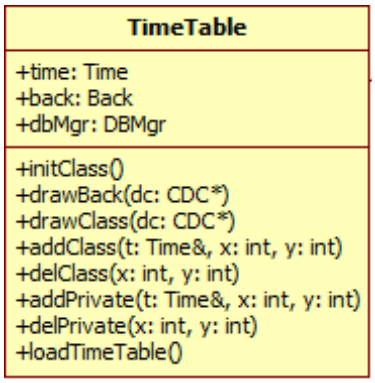
DBMgr

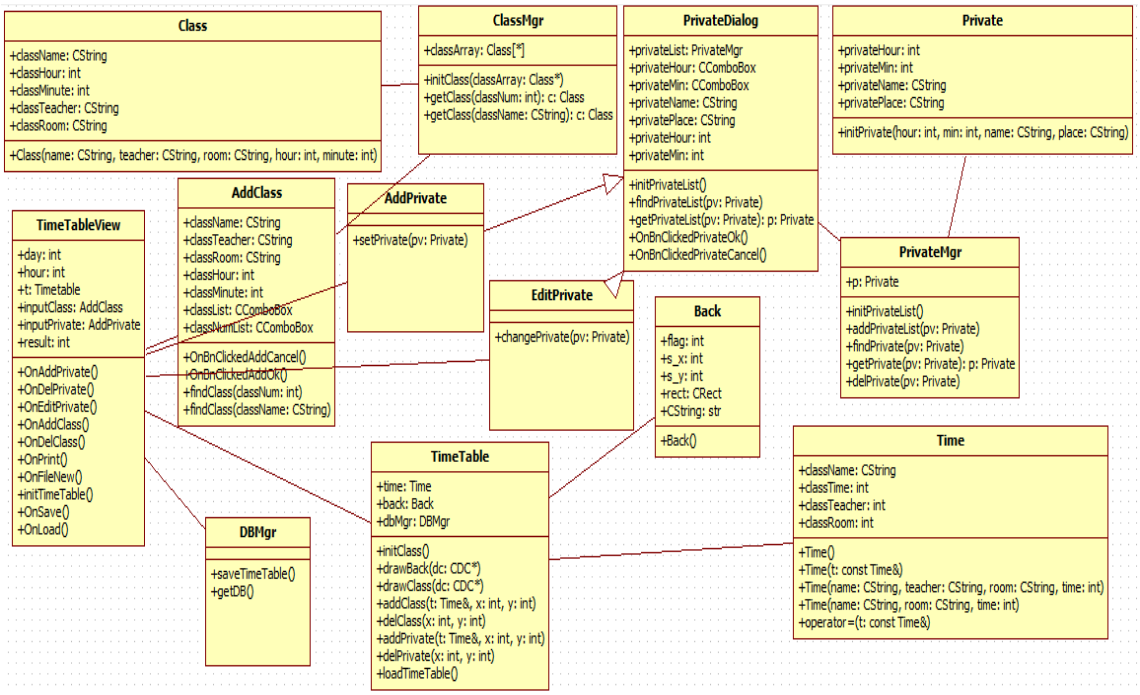


EditPrivate



TimeTable





완성된 class diagram입니다. class diagram은 앞서 살펴본 description들을 만들면서 순차적으로 완성되었습니다.

Conclusion

IEEE SDD를 따라 class diagram을 완성했습니다. 처음 decomposition description에서는 use case로부터 class 후보를 추려내어 class를 결정하였고, dependency description에서는 이 class 들의 관계를 그려냈습니다. 다음으로 interface description에서는 SRS을 통해서는 얻을 수 없는 class 들을 추가하고 새로이 관계를 찾아내고, 또 이 때 OOP 원칙을 적용함으로써 수정 및 확장에 용이한 프로그램을 설계했습니다. 마지막으로 detail description에서는 각 class의 attribute와 operation을 채움으로써 design 단계를 마무리 했습니다.

앞서 purpose에서도 언급했듯, 이번 프로젝트는 한정된 기간으로 범위를 축소하였으므로 훗날 수정 및 확장의 가능성이 높습니다. 따라서 이를 위하여 재사용이 가능한 OOD 기법을 따라 설계했습니다.