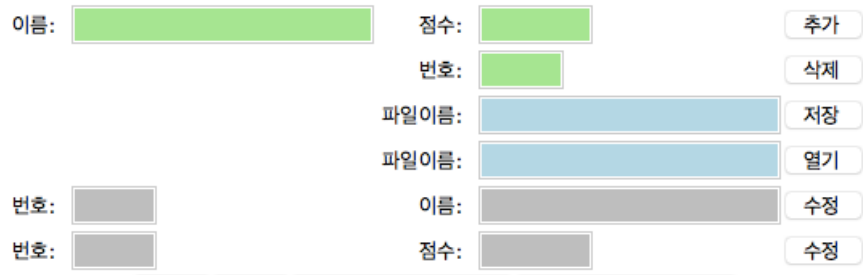
설계 보고서

컴퓨터공학부 20153240 한채연

1. UI.py : 학생관리 프로그램의 전체적인 프레임 관리(class : UI)

Frame : window Frame 으로부터 2 가지로 분류

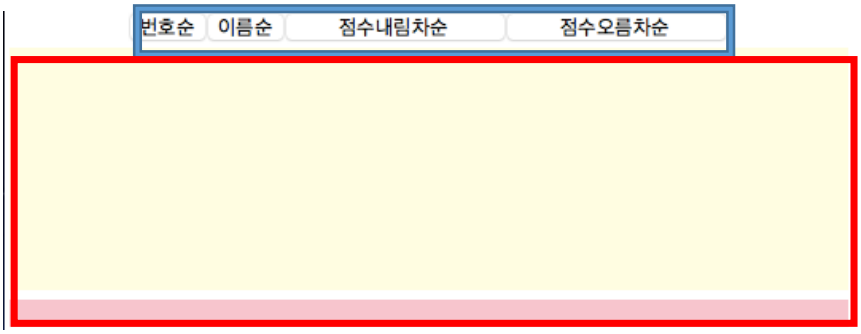
1) top_row : 데이터에 접근할 수 있고 수정 가능한 프레임. 6 개의 행과 5 개의 열로 구분되어 있다.



프레임의 label 과 Entry, Button 은 단순히 나열하여 출력할 수 있지만, 아래 코드와 같이 list 와 for 문을 이용해 간편하게 Entry 를 생성할 수 있는 코드를 구현하였다. 아래의 코드는 top_row 의 3 번째 열 label 들을 간편하게 출력하는 방법이다.

```
r=0
label_list = ["점수: ","번호: ","파일이름: ","파일이름: ","이름: ", "점수: "]
for label in label_list:
    Label(self.top_row, text=label).grid(row=r, column=2,sticky=E)
    r += 1
```

2) bottom_row : 주로 출력에 관여하는 프레임. bottom_row 는 두 가지 프레임으로 다시 나뉜다.



1. bottom_row_first : 딕셔너리를 정렬할 수 있는 버튼들이 존재하는 프레임.
2. bottom_row_second : 데이터들을 출력하는 Text 위젯이 존재하는 프레임

버튼을 작동시키기 위해 데이터나 파일 관리(파일 저장, 파일 열기)의 모듈들을 가져와야 하는 경우들이 있다. 이를 위해 데이터 모듈을 가져올 수 있는 get_dataModule, get_fileModule 메소드를 각각 만들어 버튼 관리를 보다 더 편리하게 할 수 있게 하였다.

1)get_dataModule(self, dataAdmin)

```
def get_dataModule(self,dataAdmin):
    self.data_admin = dataAdmin
    add_button = Button(self.top_row,text="추가",width=5,command=self.data_admin.add_click)
    add_button.grid(row=0,column=4,sticky=W)
    del_button = Button(self.top_row, text="삭제", width=5, command=self.data_admin.del_click)
    del_button.grid(row=1, column=4, sticky=W)

    editname_button = Button(self.top_row,text="수정",width=5, command=self.data_admin.change_name)
    editname_button.grid(row=4, column=4,sticky=W)
    editscore_button = Button(self.top_row,text="수정",width=5,command=self.data_admin.change_score)
    editscore_button.grid(row=5,column=4,sticky=W)

    numseq_button = Button(self.bottom_row_first, text="번호순", width=5, command=self.data_admin.sort_numseq)
    numseq_button.grid(row=0, column=0,sticky=N)
    nameseq_button = Button(self.bottom_row_first, text="이름순", width=5, command=self.data_admin.sort_nameseq)
    nameseq_button.grid(row=0, column=1,sticky=N)
    scoreseq1_button = Button(self.bottom_row_first, text="점수내림차순", width=15, command=self.data_admin.sort_scoreseqrev)
    scoreseq1_button.grid(row=0,column=2, sticky=N)
    scoreseq2_button = Button(self.bottom_row_first, text="점수오름차순", width=15, command=self.data_admin.sort_scoreseq)
    scoreseq2_button.grid(row=0,column=3,sticky=N)
```

함수의 인자에는 데이터를 관리하는 클래스의 객체를 받는다. 이 함수에서는 추가, 삭제, 수정(이름, 점수), 그리고 데이터들을 sorting 하는 4 개의 모든 버튼들(번호순, 이름순, 점수내림차순, 점수오름차순), 총 8 개의 버튼들을 관리한다. command 에 있는 함수들은 모두 데이터를 관리하는 클래스 내부에 존재하는 함수들이다.

2)get_fileModule(self, FileAdmin)

```
def get_fileModule(self, FileAdmin):
    self.file_admin = FileAdmin

    save_button = Button(self.top_row, text="저장", width=5, command=self.file_admin.save_file)
    save_button.grid(row=2, column=4, sticky=W)

    open_button = Button(self.top_row, text="열기", width=5, command=self.file_admin.open_file)
    open_button.grid(row=3, column=4, sticky=W)
```

함수의 인자는 파일의 IO 를 관리하는 클래스의 객체를 받는다. 이 함수에서는 저장, 열기 총 두 개의 버튼들을 관리한다. command 에 있는 함수들은 모두 파일의 IO 를 관리하는 클래스 내부에 존재하는 함수들이다.

3. clear func(state_clear & data_clear)

```
def state_clear(self):
    self.state_output.delete('1.0',END)

def data_clear(self):
    self.data_output.delete('1.0',END)
```

또한 다른 모듈에서 Text 위젯을 clear 시켜야 하는 경우가 빈번히 있기 때문에 딕셔너리를 출력하는 data Text 부분과 맨 마지막 줄에 있는 state Text 부분을 초기화 시켜주는 함수를 따로 빼내었다.

2. dataadmin.py : 학생관리 프로그램의 전체적인 데이터 관리 (class:dataadmin)

dataadmin 모듈에는 데이터를 변경할 수 있는 함수들과 sorting 함수, 그리고 데이터를 출력해주는 출력함수가 존재한다. 출력함수는 데이터를 변경할 때마다, 즉 함수가 불러와질 때 마다 새로 update 된 정보가 담긴 dictionary 를 매번 출력해야 하기 때문에 dictionary 를 출력하는 출력함수를 따로 만들었다. 데이터들은 dictionary 에 저장된다. index 는 key 에, name 은 value[0]에, score 은 value[1]에 저장하도록 한다.

1. add_click() : 추가 함수

```
self.dic_value1 = self.ui.name.get().strip()
```

: name 을 get()을 이용하여 UI 모듈의 Entry 로부터 가져와 앞뒤 공백, 줄넘김 등을 제거해 주는 strip()을 사용한 다음 변수에 저장한다.

```
if self.dic_value1 == "":
    self.ui.state_output.insert(END, "[추가 실패] 이름을 다시 입력해 주세요.")
    return
for key,value in self.dic.items() :
     if self.dic_value1 == value[0]:
        self.ui.state_output.insert(END, "[추가 실패] 동일한 이름이 존재합니다.")
        return

try:
    self.dic_value2 = eval(self.ui.secframe_entlist[0].get())
except:
    self.ui.state_output.insert(END, "[추가 실패] 점수를 다시 입력하세요.")
    return
```

error 탐지:

1) 이름이 공백일 경우(if 문으로)
2) DB 에 이미 동일한 이름이 존재할 경우(for 문으로)

3) 점수 입력 타입이 올바르지 않을 경우(try 문으로)

2. del_click() : 삭제 함수

```
try:
    self.del_num  = eval(self.ui.secframe_entlist[1].get())
    del self.dic[self.del_num]
    self.show_info()
except:
    self.ui.state_output.insert(END,"삭제에 실패했습니다.")
    return
```

error 탐지(모두 try 문으로):

1) 번호(index) 입력 타입이 올바르지 않는 경우
2) index 범위가 벗어난 경우

3. change_name() : 이름 수정함수

```
if index == "" or self.editname == "":
    self.ui.state_output.insert(END, "수정에 실패했습니다.")
    return
for key, value in self.dic.items():
    if self.editname == value[0]:
        self.ui.state_output.insert(END, "[수정 실패] 동일한 이름이 존재합니다.")
        return
try:
    self.dic[eval(index)][0] = self.editname
except:
    self.ui.state_output.insert(END, "수정에 실패했습니다.")
    return
```

error 탐지:

1) 번호(index)나 수정할 name 의 인풋값이 공백일 경우(if 문으로)
2) 수정할 이름이 이미 데이터(딕셔너리)에 존재할 경우(for 문으로)
3) index 타입이 올바르지 않을 경우(try 문으로)
4) index 범위가 벗어난 경우(try 문으로)

4. change_score() : 점수 수정함수

```
if index == "" or self.editscore == "":
    self.ui.state_output.insert(END, "수정에 실패했습니다.")
    return
try:
    self.dic[eval(index)][1] = eval(self.editscore)
except:
    self.ui.state_output.insert(END, "수정에 실패했습니다.")
    return
```

error 탐지:

1) index 나 수정할 score 값이 공백일 경우(if 문으로)
2) index 타입이 올바르지 않을 경우(try 문으로)
3) index 범위가 벗어난 경우(try 문으로)

5. sorting 함수 : 번호순, 이름순, 점수내림차순, 점수오름차순

- index sorting(번호순)

```
self.sorted_list = sorted(zip(self.dic.keys(),self.dic.values()))
```

zip()을 이용하면 key 와 value 값이 묶여 하나의 리스트로 만들어진다. index 는 key 이기때문에 zip 의 맨 앞 파라미터에 정렬 기준인 key 를 쓰고 sorted()를 쓰면 key 를 기준으로 정렬되고 리스트화된 데이터가 저장된다.

```
self.dic = dict(self.sorted_list)
```

데이터가 저장되어있던 딕셔너리를 clear 해준 다음 dict(리스트)를 하면 리스트가 사전화되어 다시 딕셔너리로 만들어 줄 수 있다.

-name & score sorting(이름 & 점수 정렬)

```
self.sorted_list = sorted(self.dic.items(),key=lambda x:x[1][0])
```

파이썬에만 존재하는 간결한 함수인 lambda 함수를 이용하여 sorted 를 한다. 위의 코드가 실행된 후에는 sorted 한 딕셔너리는 dic[1][0](이름) 혹은 dic[1][1](점수) 중심으로 정렬된 tuple 타입의 데이터가 변수에 저장된다. 후에 for 문을 이용하여 정렬된 데이터를 출력한다. (점수 내림차순 정렬 경우에는 sorted 메소드 파라미터의 맨 뒤에 reverse = True 를 입력하면 된다.)

3. fileadmin.py : 학생관리 프로그램 데이터의 파일 IO 관리 (class:FileAdmin)

FileAdmin 클래스에는 파일을 저장하고, 열 수 있는 메소드를 구현했다.

1. save_file() : 파일 저장 함수

if 문을 이용해 파일 이름이 입력될 Entry 를 get()한 값에 strip()메소드를 사용한 다음 파일 이름이 공백이면 return 하도록 처리했다. 쓰기모드('w')로 파일 객체를 생성한다.

2. open_file() : 파일 열기 함수

위 함수와 동일하게 error 탐지를 하고 추가로 try 문을 사용해 입력한 이름을 가지고 있는 파일이 존재하지 않으면 return 을 하도록 error 탐지를 했다. readline() 메소드를 사용해 한 줄씩 읽도록 했으며 각 key, value 값은 txt 파일에 '₩t'로 구분되어 있어서 split("₩t") 메소드를 추가로 사용하여 딕셔너리에 데이터 값을 추가할 수 있게 하였다. 맨 마지막 인덱스인 last_index 를 if 문으로 찾아내어 인덱스 관리에 조금 더 용이하게 짰다.