

소프트웨어 프로젝트 레포트 6

SchedulePlanner Program

20191975 임혜민

소프트웨어학과 소프트웨어학부

이 과제는 SchedulePlanner 의 통합 버전을 만드는 것이다. 월별 일정 및 일별 일정을 표시하고 외부 DB 파일과의 연동을 통해 일정을 관리하는 일정 관리 프로그램이다. 이 레포트는 아래와 같은 파트로 나누어진다.

1. 설계 노트

2. 프로그램 소스 코드

2-1. ScheduleGUI 클래스: main()

2-2. Monthly 클래스: 월별 일정

2-3. Daily 클래스: 일별 일정

2-4. Schedule 클래스: 개별 일정

2-5. ScheduleList 클래스: 일정 리스트

3. 최종 테스트 과정 화면

4. 자체 평가표

1. 설계 노트

일정 관리 프로그램이 실제로 동작하기 위해서, 외부 파일로 존재하는 데이터베이스에서 클래스로, 하나의 클래스에서 또 다른 클래스로 정보가 공유되는 동작을 구현하려고 노력하였다.

```
public class ScheduleGUI{
    public static void main(String[] args) {
        ScheduleList list = new ScheduleList("C:\\Users\\임해민\\eclipse-workspace\\Schedule_management\\src\\schedule-file.data");
        Monthly frame = new Monthly(list);
        frame.MonthlyFrame();
    }
}
```

프로그램 구성(main함수 -> 월별 일정): 원래 코드에서는 일별 일정 클래스인 Daily에서 ScheduleList 객체를 생성하였지만 프로그래밍 구조를 개선하기 위하여 구조를 바꾸었다. 프로그램을 직접 실행하는 main 함수에서 ScheduleList 객체를 생성하고 그 객체를 월별 일정 클래스인 Monthly의 객체를 생성할 때 인자로 넘겨준다. 받은 ScheduleList는 생성자를 통해 데이터 필드에 추가된다. 그러면 GUI 안에서도 ScheduleList에 접근할 수 있지만, 직접 객체를 생성하지 않으므로 ScheduleList 중복 생성과 같은 구조상의 불안정성을 해결할 수 있다.

Member visibility: 또한 모든 클래스의 데이터 필드는 private으로, 다른 클래스에서 직접 접근할 수 없다. 프로그램 기능상 접근이 필요할 경우, public 메소드를 이용해 가져올 수 있도록 설계하였다.

프로젝트 6의 구현 내용 중 하나인 월별 이동은 프로젝트 5 레포트에서 이미 설명하였으므로 레포트 내용의 캡처로 대신한다.

달력의 월별이동을 구현할 필요는 없지만 추후의 편한 활용을 위하여 LocalDate 클래스를 이용하여 구현하였다. 열거형 상수인 DayOfWeek 가 내 프로그램에 맞게 구성되어 있지 않아서 switch를 이용하여 맞춰주었다. 달마다 날짜 배치가 달라지는 것은 첫날의 요일을 구하고 그 달의 일수를 구하여 버튼을 날짜만큼 채운 다음 빈칸에 빈 라벨(JLabel)을 채워 넣는 것으로 구현하였다.

프로그램 구성(월별 일정 Monthly -> 일별 일정 Daily): 이미 프로젝트 5의 GUI에서, 월별 일정의 날짜 버튼을 누르는 순간 이벤트 핸들링으로 새로운 일별 일정 객체가 생성되고 그 창이 뜨는 구조로 프로그램을 구현하였다. Daily 객체를 생성할 때 생성할 일별일정에 해당하는 연도, 월, 일, 데이터베이스로 활용할 ScheduleList 객체를 인수로 전달한다.

```
class Listener implements ActionListener{//버튼을 누르면 발생하는 이벤트를 위한 클래스
    public void actionPerformed(ActionEvent act) {
        //날짜 버튼을 누르면 일별 일정 프레임이 나온다.
        for(int i = 0; i < day; i++) {
            if(act.getSource() == dayButton[i]) {
                Daily daySchedule = new Daily(year, (month), (i+1), list);
                daySchedule.setTitle("Daily Schedule- "+year+"-"+(month)+"-"+(i+1));
                daySchedule.DailyFrame();
            }
        }
    }
}
```

Monthly 클래스의 데이터 필드 중 dayButton이라는 배열은 버튼의 배열이다.

```
private JButton[] dayButton = new JButton[day];
```

//day는 이 달의 날짜 수를 나타낸다. 날짜수만큼 그 날짜에 해당하는 숫자 텍스트를 가진 버튼을 추가한다.

```
for(int i = 0; i < day; i++) {
    dayButton[i] = new JButton(""+(i+1));
    dayButton[i].addActionListener(new Listener());
    p_under.add(dayButton[i]);
}
```

dayButton의 인덱스 번호에 +1을 한 것이 날짜의 정보이다. 년도/월 정보는 Monthly 클래스의 데이터 필드에 있으며, 월별 이동을 할 때마다 해당 일자에 맞게 바뀐다. 이렇게 눌린 버튼에 맞는 날짜 정보를 Daily 객체를 생성할 때 인수로 넣으면 된다.

즉, 월별 일정 클래스는 모든 날을 나타내는 각각의 객체를 생성하고 저장하고 있지 않다는 이야기이다. 그냥 날짜 버튼을 누르는 순간 해당 날짜의 정보를 담은 일별 일정 객체와 창이 생기고, 그 창을 닫으면 생긴 일별 일정 객체가 사라진다. 버튼을 다시 누르면 또 새로운 객체가 생긴다. Daily 객체가 액션 리스너 안에서만 생기는 로컬변수이기 때문이다.

나는 따라서 이러한 구조를 계속 가지고 가서, 일별 일정 클래스도 '표시'에만 중점을 두고, 일정들을 저장하는 기능을 온전히 ScheduleList라는 별도의 클래스에 할당하였다.

일별 일정 클래스인 Daily 객체가 생성될 때 마다(날짜 버튼이 눌릴 때 마다) 생성자는 GUI를 생성한 후 ScheduleList를 탐색한다.

```
//파일에 있는 일정 추가
for(int i = 0; i < list.numSchedules(); i++) {
    Schedule def = list.getSchedule(i);
    if(def.getStartYear() == year && def.getStartMonth() == month && def.getStartDay() == day) { //일정이 있는 날일때 초기화면
        JPanel newSchedule = new JPanel(new GridLayout(1,4));

        JTextField title = new JTextField();
        title.setText(def.getName());
        title.setPreferredSize(new Dimension(100,100));
        newSchedule.add(title);

        JTextField s_time = new JTextField();
        s_time.setText(def.getStartHour());
        newSchedule.add(s_time);

        JTextField e_time = new JTextField();
        e_time.setText(def.getEndHour());
        newSchedule.add(e_time);

        JTextField memo = new JTextField();
        memo.setText(def.getMemo());
        newSchedule.add(memo);

        p_middle.add(newSchedule);
    }
    else { //일정이 없는 날일때 초기화면
    }
}
```

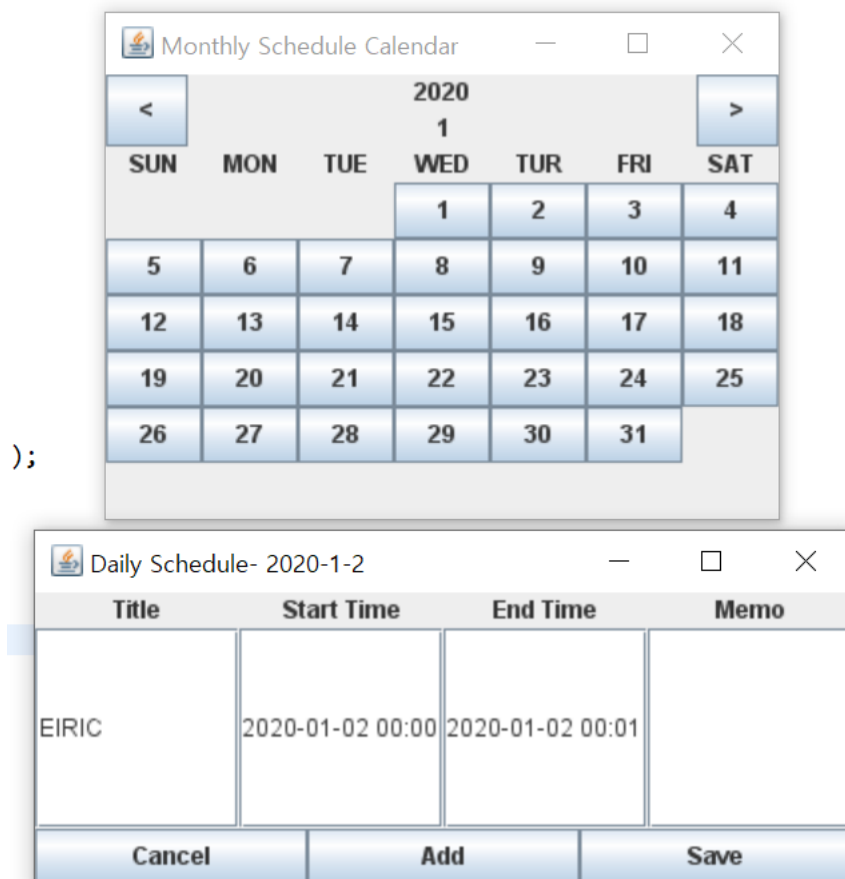
이 객체의 데이터 필드에 저장되어 있는 날짜 정보(이 Daily 객체가 나타내는 년/월/일 정보)와 ScheduleList에 저장되어있는 Schedule의 년/월/일 정보를 비교하여 만약 같으면 해당하는 일정을 표시한다. 아래의 예시로 설명한다.

```

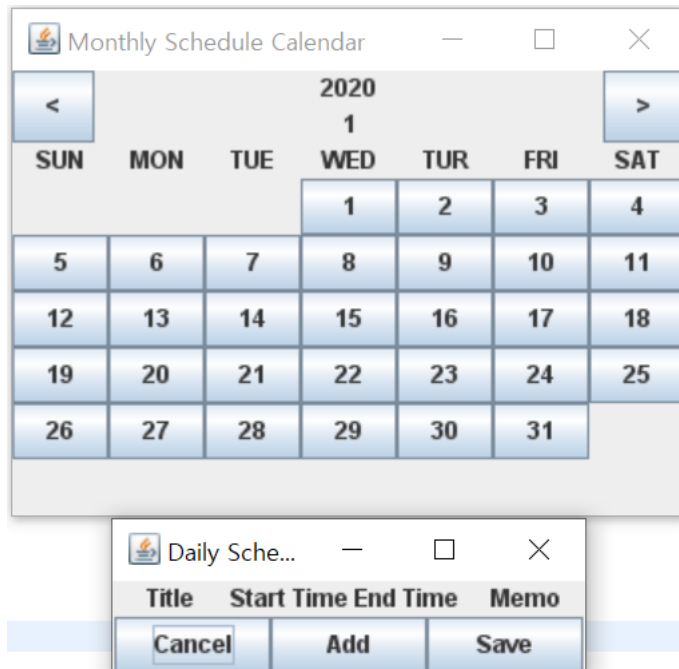
schedule-file.data - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
;
data//2020-03-25 13:00//2020-03-25 13:50//Test
;
Java Meeting 2//03-25-2020 15:01//2020-03-25 15:50//Test - 2
;
New Year//2021-01-01 00:01//2020-01-01 01:00//Happy
EIRIC//2020-01-02 00:00//2020-01-02 00:01//

```

만약 일정 DB 파일인 schedule-file.data가 이런 일정들로 구성되어 있다고 하자.



그렇다면 이렇게 월별 일정창을 띄우고, 일별 일정 창을 띄울 때 정상일정이라 ScheduleList에 저장된 EIRIC 일정이 있는 2020년 1월 2일 버튼을 누르면 해당하는 일정이 일별 일정창에 표시된다. 2020년 1월 2일 버튼을 누르는 순간 새로운 Daily 객체가 생성되고, 그 생성자가 ScheduleList에 저장되어 있는 일정을 탐색하여 2020년 1월 2일 일정을 찾아 일별 일정창에 표시하기 때문이다.



따라서 일정이 없는 다른 날짜의 버튼- 예를 들어 2020년 1월 1일- 을 누르면 아무것도 뜨지 않는다.

이러한 방식은 Daily 객체에 어떤 일정 정보가 포함되어서 날짜마다 다른 결과가 나오는 것이 아니다. Daily는 그저 일정을 찾고, 그 일정이 해당하는 날짜가 맞으면 화면에 띄우는 역할을 하는 것이고, 실제 일정 정보는 모두 ScheduleList에 저장되어 있다.

이제 기본적으로 DB 파일에 적혀 있는 일정을 화면상에 나타낼 수 있게 되었다. 그렇다면 화면(사용자)의 클릭과 같은 이벤트를 통하여 이러한 일정을 추가하고, 저장할 수 있게 하는 기능을 구현하는 것이 남았다.

이것은 당연히 Daily의 Cancel, Add, Save 버튼과 그 버튼을 클릭함으로써 발생하는 이벤트 핸들링에 의해 진행된다. 각 버튼의 구현과 그 과정에서 발생했던 고민사항을 설명하겠다.

프로그램 구성(일별 일정 Daily 에서 ScheduleList 액세스):

1. cancel

이 버튼을 누르면 수정사항을 저장하지 않은 상태로 일별 일정창이 꺼져야 한다. 이것은 그냥 창을 닫기만 하면 해결되는 문제이다. 아래에서 설명하겠지만, 프로그램 구조상 그냥 텍스트 필드를 수정하는 것만으로는 내용이 수정되어 저장되지 않기 때문이다. 따라서 Cancel을 누르면 그냥 Save를 누르지 않고 창 종료 버튼(x버튼)을 누르는 것과 같은 동작을 한다.

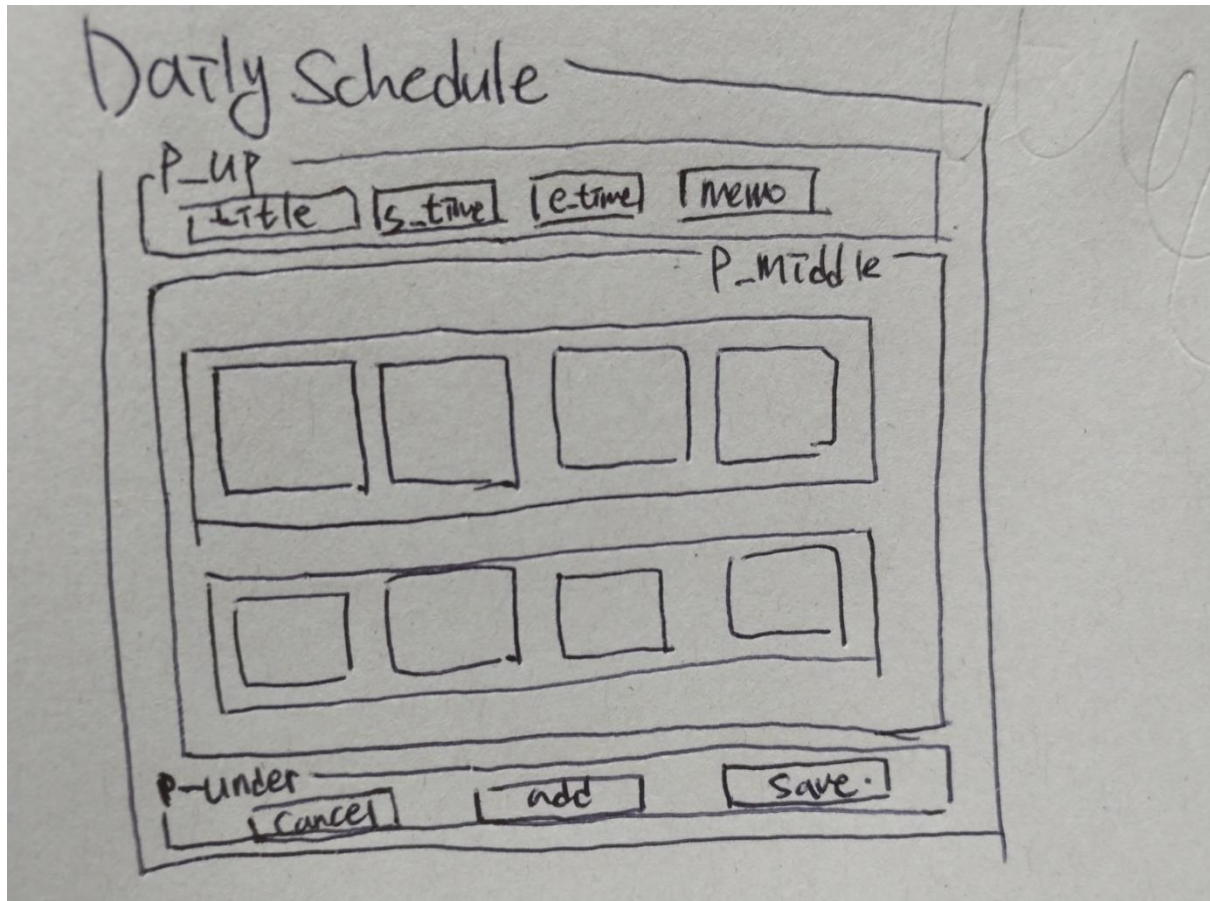
```
if(act.getSource() == cancel) {
    dispose();
}
```

ActionListener를 implement한 Listner 클래스가 Daily 안에 들어있는 inner class이고, Daily 클래스

는 JFrame의 subclass이기 때문에, Listener는 JFrame의 메소드인 dispose()를 별도의 객체 생성 없이 쓸 수 있다.

2. Add

이 버튼을 누를 때마다 JTextField 4개로 이루어진 JPanel 한 개가 추가되도록 만들어야 했다.



그러나 이렇게 p_middle에 컴포넌트들이 배치되도록 하려면, 4개의 텍스트 필드를 하나의 패널로 감싸고, 그 패널들을 한 줄에 하나씩 넣을 수 있는 레이아웃을 p_middle에 설정해야 했다.

내가 알고 있었던 레이아웃은 grid, border, flow 뿐이었고 그 3개가 모두 조건에 맞지 않았다. Flow는 컴포넌트들이 가로로 줄지어 배치되었고, border와 grid는 add를 누르면 계속해서 추가되어야 하는데 생성자에서부터 구획을 정해 놓아야 해서 부적합했다.

```
//일정 추가된게 아무것도 없을 때 초기화면  
p_middle = new JPanel();  
p_middle.setLayout(new BorderLayout(p_middle, BorderLayout.Y_AXIS));
```

그래서 검색을 한 결과 BorderLayout을 찾았고 그것을 p_middle에 적용하였다. 이 레이아웃은 axis 방향을 정해서 컴포넌트들을 그 축방향에 맞게 정렬할 수 있고 추후에 추가되는 컴포넌트들도 따로 레이아웃을 바꾸거나 설정을 해 줄 필요 없이 자동으로 정렬되기 때문에 적합한 레이아웃이라

고 판단하였다.

```
else if(act.getSource() == add) {
    JPanel newSchedule = new JPanel(new GridLayout(1,4));

    JTextField title = new JTextField();
    JTextField s_time = new JTextField();
    JTextField e_time = new JTextField();
    JTextField memo = new JTextField();
    title.setPreferredSize(new Dimension(100,100));

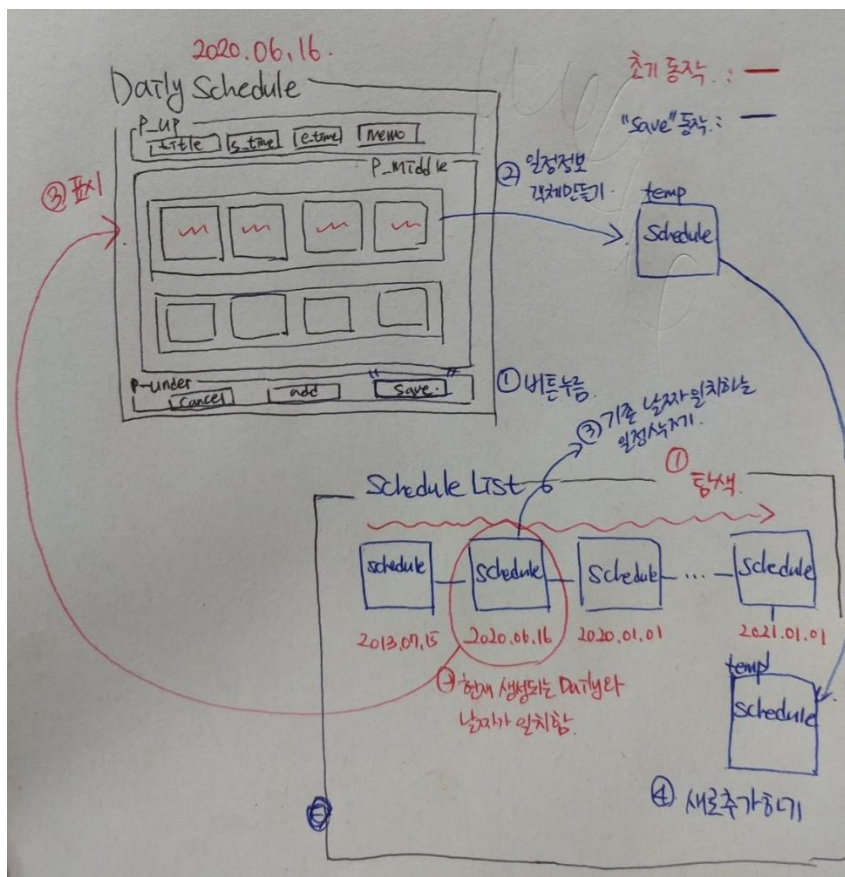
    newSchedule.add(title);
    newSchedule.add(s_time);
    newSchedule.add(e_time);
    newSchedule.add(memo);

    p_middle.add(newSchedule);

    p.revalidate();
    p.repaint();
    pack();
}
```

또한 한 줄 씩 추가될 때마다 창의 크기가 맞춰져서 바뀌도록 구현하였다. Cancel의 dispose와 같이 inner class이기 때문에 별도의 객체 생성 없이 사용 가능한 JFrame의 pack()을 사용하였다.

3. save



Save를 구현하기 위해 그렸던 설계그림이다. 붉은 선으로 그린 것이 날짜 버튼을 눌렀을 때 일어나는 초기 동작이고, 파란 선으로 그린 것이 Save 버튼을 눌렀을 때 일어나는 동작이다.

```
for(int j = 0; j < p_middle.getComponentCount(); j++) {
    JPanel temp = (JPanel)p_middle.getComponent(j);
    String temp_title = ((JTextField)temp.getComponent(0)).getText();
    String temp_start_time = ((JTextField)temp.getComponent(1)).getText();
    String temp_end_time = ((JTextField)temp.getComponent(2)).getText();
    String temp_memo = ((JTextField)temp.getComponent(3)).getText();
    Schedule sch = new Schedule(temp_title, temp_start_time, temp_end_time, temp_memo);
    list.addSchedule(sch);
}
```

Save 버튼을 누르면, (수정 여부와 상관없이) 그 순간에 p_middle의 텍스트 필드에 적힌 일정 정보가 차례로(한 줄 씩) Schedule 객체로 만들어져 ScheduleList에 추가되는 형태로 구현하였다.

```
else if(act.getSource() == save) {
    for(int i = 0; i < list.numSchedules(); i++) { //그 날의 기존일정 제거
        Schedule def = list.getSchedule(i);
        if(def.getStartYear() == year && def.getStartMonth() == month && def.getStartDay() == day) { //일정이 있는 날일때 초기화면
            list.removeSchedule(i);
            i--;
        }
    }
}
```

그런데 수정되기 전의 Schedule 객체도 ScheduleList에 남아있을 수는 없으니 추가하기 전에 먼저 기존에 해당하는 날짜에 있었던 기존 일정을 모두 지웠다. ScheduleList에 있는 Schedule과의 날짜 비교, 제거는 모두 ScheduleList와 Schedule에 설정한 메소드들을 통해 이루어졌다. 지울 때마다 i--하는 이유는 removeSchedule이라는 메소드가 활용하는 ArrayList의 remove()가 요소를 지우면서 인덱스도 같이 변화시키기 때문에 그것을 대비해서 조정해준것이다.

(기존 일정 지우는 것이 먼저임!! 추가는 나중에 함)

여기서 많이 해맸던 점이 처음에 p_middle안에 들어있는 저 컴포넌트들이 다 따로 변수로 지정되어 있지 않았다는 점이었다. Add로 인해 얼마나 더 추가될지 모르니 저 모든 컴포넌트들을 다 따로 이름이 있는 변수에 할당해서 데이터 필드에 저장할 수도 없었다.

따라서 JPanel과 JTextField의 상위 클래스인 Container 에 있는 getComponent() 함수를 이용하여 Component 형태로 각각의 요소를 꺼내고, 명시적 캐스팅을 해줘서 그 안의 텍스트를 꺼내 올 수 있게 하였다.

```
list.removeError();
list.writeScheduleList();
```

그렇게 ScheduleList를 수정된 정보들로 바꾸고 나면 removeError() 라는 ScheduleList에 설정한 메소드를 통해 비정상적으로 수정된 일정들을 지운다.


```

public void removeError() {
    for(int i = 0; i < numSchedules(); i++) {
        String error = getSchedule(i).errorcheck();
        if(error.equals("none")) {

        }
        else if(error.equals("noName")) {
            removeSchedule(i);
        }
        else if(error.equals("wrongTimeFormat")) {
            removeSchedule(i);
        }
        else if(error.equals("timeParadox")) {
            removeSchedule(i);
        }
        else;
    }
}

```

ScheduleList 객체가 Daily 객체의 생성에 따라 갱신되지 않기 때문에 생성자에 있었던 에러 체크 만으로는 추후 비정상 일정 입력에 대응할 수 없다. 따라서 메소드를 따로 만들어서 에러 체크를 했다.

그렇게 비정상 입력을 모두 관리한 ScheduleList를 데이터 베이스 파일에 덮어씌운다. 이것은 ScheduleList에 설정한 메소드, writeScheduleList()를 통해 한다.

`dispose();`

Save 버튼을 누르면 일별 일정창이 자동으로 닫히게 하였다. 새 창을 열면 수정되거나 새로 저장된 정보까지 포함된 새 ScheduleList를 기반으로 한 일정이 표시된다.

프로그램 구성(개별 일정, 일정 리스트): 스케줄과 관련된 Schedule, ScheduleList 두 클래스는 프로젝트 3 에서 작성한 코드에서 프로그램에 맞게 메소드를 추가(외부 클래스에서 private인 데이터 필드 값에 접근할 수 있도록 하는 public method) 한 것만 다르고 기본적인 파일 입출력 과정이나 스케줄 입력 과정은 원리가 같다.

이 구조의 한계라면 저장하고 일별 일정창을 불러올 때마다 ScheduleList를 전수조사하기 때문에 ScheduleList에 저장된 Schedule이 많아지면 프로그램이 굉장히 느려질 수 있다는 점이다. 그러한 문제를 해결하기 위해서 ScheduleList를 좀 더 효과적으로 탐색하는 방법을 적용할 수도 있을 것 같다. LocalDateTime의 비교 기능을 이용해서 이진 탐색을 한다거나, ScheduleList의 Schedule을 날짜순으로 정렬해서 탐색을 쉽게 하는 등의 방법이 존재할 수 있다.

2. 프로그램 소스 코드

2-1. ScheduleGUI 클래스: main()

```

public class ScheduleGUI{
    public static void main(String[] args) {

```

```

        ScheduleList list = new ScheduleList("C:\\Users\\임혜민\\eclipse-
workspace\\Schedule_management\\src\\schedule-file.data");
        Monthly frame = new Monthly(list);
        frame.MonthlyFrame();
    }
}

```

2-2. Monthly 클래스: 월별 일정

```

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.time.*;

import javax.swing.*;

public class Monthly extends JFrame{
    //ScheduleList
    private ScheduleList list;
    //날짜와 관련된 data field
    private int year = 2020;
    private int month = 5;
    private int day = 31;
    private int firstday = 5;

    //GUI 구현 관련 data field
    private JPanel p = new JPanel();
    private JPanel p_up = new JPanel(new BorderLayout(5,10));
    private JPanel ynm = new JPanel(new BorderLayout());
    private JLabel y = new JLabel(""+year, JLabel.CENTER);
    private JLabel m = new JLabel(""+month, JLabel.CENTER);
    private JPanel p_middle = new JPanel(new GridLayout(1, 7));
    private JPanel p_under = new JPanel(new GridLayout(6,7));

    private JButton[] dayButton = new JButton[day];
    private JButton lastmonth = new JButton("<");
    private JButton nextmonth = new JButton(">");

    Monthly(){//전달받은 인자가 없을 때의 생성자
        //외부 파일을 새로 만들어야 할까?
        //외부 파일이 없어도 ScheduleList를 만들 수도 있긴한데 ... 객체 생성일까
        이것도?
    }

    Monthly(ScheduleList list){//생성자
        //전달받은 ScheduleList를 데이터 필드에 저장.
        this.list = list;

        //상단바 - 좌우 화살표와 년, 월 표시
        //p_up panel은 lastmonth(좌측화살표), nextmonth(우측화살표), ynm이라는

```

panel로 구성되어 있다.

```
p_up.add(lastmonth, BorderLayout.WEST);
lastmonth.addActionListener(new Listener());

//ynm panel은 년도를 표시하는 JLabel y와 월을 표시하는 JLabel m을 포함.
ynm.add(y, BorderLayout.NORTH);
ynm.add(m, BorderLayout.SOUTH);
p_up.add(ynm, BorderLayout.CENTER);

p_up.add(nextmonth, BorderLayout.EAST);
nextmonth.addActionListener(new Listener());
```

//중간바 - 요일 표시

```
p_middle.add(new JLabel("SUN", JLabel.CENTER));
p_middle.add(new JLabel("MON", JLabel.CENTER));
p_middle.add(new JLabel("TUE", JLabel.CENTER));
p_middle.add(new JLabel("WED", JLabel.CENTER));
p_middle.add(new JLabel("TUR", JLabel.CENTER));
p_middle.add(new JLabel("FRI", JLabel.CENTER));
p_middle.add(new JLabel("SAT", JLabel.CENTER));
```

//하단 - 날짜 표시

//firstday는 매월 1일의 요일을 나타낸다. 즉 firstday가 0이면

일요일~6이면 토요일인 식이다. 매월 1일 전에는 아무것도 없으므로 빈 JLabel을 grid에 추가한다.

```
for(int i = 0; i < firstday; i++) {
    p_under.add(new JLabel(""));
}
```

//day는 이 달의 날짜 수를 나타낸다. 날짜수만큼 그 날짜에 해당하는 숫자 텍스트를 가진 버튼을 추가한다.

```
for(int i = 0; i < day; i++) {
    dayButton[i] = new JButton(""+(i+1));
    dayButton[i].addActionListener(new Listener());
    p_under.add(dayButton[i]);
}
```

//Grid의 나머지 부분을 빈 라벨로 채운다.

```
for(int i = 0; i < ((6*7) - day - firstday); i++) {
    p_under.add(new JLabel(""));
}
```

//전체 패널인 p에 지금까지 만든 모든 패널을 더한다.

```
p.setLayout(new BorderLayout());
p.add(p_up, BorderLayout.NORTH);
p.add(p_middle, BorderLayout.CENTER);
p.add(p_under, BorderLayout.SOUTH);
```

}

public void MonthlyFrame() { //월별 일정을 프레임으로 나타낸다.

this.setTitle("Monthly Schedule Calendar");

this.add(p);

this.pack(); //크기조정은 컴포넌트에 맞춰 자동으로 조절된다.

this.setLocationRelativeTo(**null**);

```

        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }

```

class Listener **implements** ActionListener{//버튼을 누르면 발생하는 이벤트를 위한 클래스

```

        public void actionPerformed(ActionEvent act) {
            //날짜 버튼을 누르면 일별 일정 프레임이 나온다.
            for(int i = 0; i < day; i++) {
                if(act.getSource() == dayButton[i]) {
                    Daily daySchedule = new Daily(year, (month),
(i+1), list);
                    daySchedule.setTitle("Daily Schedule-
"+year+"-"+(month)+"-"+(i+1));
                    daySchedule.DailyFrame();
                }
            }
            //좌측 화살표를 누르면 전월 달력이 나온다.
            if(act.getSource() == lastmonth) {
                LocalDate cal = LocalDate.of(year, month, 1);
                cal = cal.minusMonths(1);
                year = cal.getYear();
                month = cal.getMonthValue();
                day = cal.lengthOfMonth();
                switch (cal.getDayOfWeek()) {
                    case SUNDAY:
                        firstday = 0;
                        break;
                    case MONDAY:
                        firstday = 1;
                        break;
                    case TUESDAY:
                        firstday = 2;
                        break;
                    case WEDNESDAY:
                        firstday = 3;
                        break;
                    case THURSDAY:
                        firstday = 4;
                        break;
                    case FRIDAY:
                        firstday = 5;
                        break;
                    case SATURDAY:
                        firstday = 6;
                        break;
                }

                y.setText(""+year);
                m.setText(""+(month));

                p_under.removeAll();

                for(int i = 0; i < firstday; i++) {

```

Listener());

```
        p_under.add(new JLabel(""));
    }

    for(int i = 0; i < day; i++) {
        dayButton[i] = new JButton(""+(i+1));
        dayButton[i].addActionListener(new
            p_under.add(dayButton[i]);
    }

    for(int i = 0; i < ((6*7) - day - firstday); i++) {
        p_under.add(new JLabel(""));
    }

    p.revalidate();
    p.repaint();
}

//우측 화살표를 누르면 내월 달력이 나온다.
else if(act.getSource() == nextmonth) {
    LocalDate cal = LocalDate.of(year, month, 1);
    cal = cal.plusMonths(1);
    year = cal.getYear();
    month = cal.getMonthValue();
    day = cal.lengthOfMonth();
    switch (cal.getDayOfWeek()) {
        case SUNDAY:
            firstday = 0;
            break;
        case MONDAY:
            firstday = 1;
            break;
        case TUESDAY:
            firstday = 2;
            break;
        case WEDNESDAY:
            firstday = 3;
            break;
        case THURSDAY:
            firstday = 4;
            break;
        case FRIDAY:
            firstday = 5;
            break;
        case SATURDAY:
            firstday = 6;
            break;
    }

    y.setText(""+year);
    m.setText(""+(month));

    p_under.removeAll();

    for(int i = 0; i < firstday; i++) {
        p_under.add(new JLabel(""));
    }
}
```

```

        for(int i = 0; i < day; i++) {
            dayButton[i] = new JButton(""+(i+1));
            dayButton[i].addActionListener(new
Listener());

            p_under.add(dayButton[i]);
        }

        for(int i = 0; i < ((6*7) - day - firstday); i++) {
            p_under.add(new JLabel(""));
        }

        p.revalidate();
        p.repaint();
    }

}
}
}

```

2-3. Daily 클래스: 일별 일정

```

import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

public class Daily extends JFrame{
    //날짜 관련 데이터필드
    private int year;
    private int month;
    private int day;
    //초기일정
    private ScheduleList list;
    //버튼
    private JButton cancel = new JButton("Cancel");
    private JButton add = new JButton("Add");
    private JButton save = new JButton("Save");
    //패널
    private JPanel p = new JPanel(new BorderLayout());
    private JPanel p_up = new JPanel(new GridLayout(1, 4));
    private JPanel p_middle;
    private JPanel p_under = new JPanel(new GridLayout(1, 3));
    //기본 생성자
    Daily(){

    }
    //생성자
    Daily(int y, int m, int d, ScheduleList list){
        this.list = list;
        //상단바
        p_up.add(new JLabel("Title", JLabel.CENTER));
        p_up.add(new JLabel("Start Time", JLabel.CENTER));
    }
}

```

```

p_up.add(new JLabel("End Time", JLabel.CENTER));
p_up.add(new JLabel("Memo", JLabel.CENTER));

//초기 일정정보 세팅 - 프로젝트 3과 연동
year = y;
month = m;
day = d;

//일정 추가된게 아무것도 없을 때 초기화면
p_middle = new JPanel();
p_middle.setLayout(new BoxLayout(p_middle, BoxLayout.Y_AXIS));

//파일에 있는 일정 추가
for(int i = 0; i < list.numSchedules(); i++) {
    Schedule def = list.getSchedule(i);
    if(def.getStartTime_year() == year &&
def.getStartTime_month() == month && def.getStartTime_day() == day) {//일정이 있는
날일때 초기화면

        JPanel newSchedule = new JPanel(new
GridLayout(1,4));

        JTextField title = new JTextField();
        title.setText(def.getName());
        title.setPreferredSize(new Dimension(100,100));
        newSchedule.add(title);

        JTextField s_time = new JTextField();
        s_time.setText(def.getStartTime());
        newSchedule.add(s_time);

        JTextField e_time = new JTextField();
        e_time.setText(def.getEndTime());
        newSchedule.add(e_time);

        JTextField memo = new JTextField();
        memo.setText(def.getMemo());
        newSchedule.add(memo);

        p_middle.add(newSchedule);
    }
    else {//일정이 없는 날일때 초기화면

    }

}

//하단바 - 버튼 세개
cancel.addActionListener(new Listener());
p_under.add(cancel);
add.addActionListener(new Listener());
p_under.add(add);
save.addActionListener(new Listener());
p_under.add(save);

```



```

        //전체 패널 p에 지금까지 만든 세 개의 패널을 추가.
        p.add(p_up, BorderLayout.NORTH);
        p.add(p_middle, BorderLayout.CENTER);
        p.add(p_under, BorderLayout.SOUTH);
    }

    class Listener implements ActionListener{
        public void actionPerformed(ActionEvent act) {//버튼 기능 콘솔출력만
간단하게 추가함.

            if(act.getSource() == cancel) {
                dispose();
            }
            else if(act.getSource() == add) {
                JPanel newSchedule = new JPanel(new
GridLayout(1,4));

                JTextField title = new JTextField();
                JTextField s_time = new JTextField();
                JTextField e_time = new JTextField();
                JTextField memo = new JTextField();
                title.setPreferredSize(new Dimension(100,100));

                newSchedule.add(title);
                newSchedule.add(s_time);
                newSchedule.add(e_time);
                newSchedule.add(memo);

                p_middle.add(newSchedule);

                p.revalidate();
                p.repaint();
                pack();
            }
            else if(act.getSource() == save) {

                for(int i = 0; i < list.numSchedules(); i++) {//그
날의 기존일정 제거

                    Schedule def = list.getSchedule(i);
                    if(def.getStartTime_year() == year &&
def.getStartTime_month() == month && def.getStartTime_day() == day) {//일정이 있는
날일때 초기화면

                        list.removeSchedule(i);
                        i--;
                    }
                }

                for(int j = 0; j < p_middle.getComponentCount();
j++) {
                    JPanel temp =
(JPanel)p_middle.getComponent(j);
                    String temp_title =
((JTextField)temp.getComponent(0)).getText();
                    String temp_start_time =
((JTextField)temp.getComponent(1)).getText();
                    String temp_end_time =

```

```

((JTextField)temp.getComponent(2)).getText();
        String temp_memo =
((JTextField)temp.getComponent(3)).getText();
        Schedule sch = new Schedule(temp_title,
temp_start_time, temp_end_time, temp_memo);
        list.addSchedule(sch);
    }
    list.removeError();
    list.writeScheduleList();
    dispose();
}
else;
}
}

public void DailyFrame() { //일별 일정을 프레임으로 나타낸다.
    this.add(p);
    this.pack();
    this.setLocationRelativeTo(null);
    this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    this.setVisible(true);
}
}

```

2-4. Schedule 클래스: 개별 일정

```

import java.time.*;
import java.time.format.*;

public class Schedule {
    private String name;
    private LocalDateTime start_time;
    private LocalDateTime end_time;
    private String memo;
    private DateTimeFormatter schedule_time =
DateTimeFormatter.ofPattern("uuuu-MM-dd HH:mm");

    //constructor
    Schedule(String infoString){
        String[] info = infoString.split("/");
        for(int i = 0; i < info.length; i++) {
            info[i] = info[i].trim();
        }

        try {
            if(info.length == 4) { // .length는 배열의 길이, .length()는
문자열의 길이(문자열은 배열이 아니므로)
                name = info[0];
                start_time = LocalDateTime.parse(info[1],
schedule_time);
                end_time = LocalDateTime.parse(info[2],
schedule_time);
                memo = info[3];
            }
            else if(info.length == 3) {

```

```

        name = info[0];
        start_time = LocalDateTime.parse(info[1],
schedule_time);
        end_time = LocalDateTime.parse(info[2],
schedule_time);
        memo="";
    }
    else;
}
catch(DateTimeParseException e){
    start_time = LocalDateTime.MAX;
    end_time = LocalDateTime.MIN;
}
}

Schedule(String name, String sTime, String eTime, String memo){
    try {
        this.name = name;
        this.start_time = LocalDateTime.parse(sTime, schedule_time);
        this.end_time = LocalDateTime.parse(eTime, schedule_time);
        this.memo = memo;
    }
    catch(DateTimeParseException e){
        start_time = LocalDateTime.MAX;
        end_time = LocalDateTime.MIN;
    }
}

Schedule(String name, String sTime, String eTime){
    try {
        this.name = name;
        this.start_time = LocalDateTime.parse(sTime, schedule_time);
        this.end_time = LocalDateTime.parse(eTime, schedule_time);
        this.memo = "";
    }
    catch(DateTimeParseException e){
        start_time = LocalDateTime.MAX;
        end_time = LocalDateTime.MIN;
    }
}

//print an entire schedule
public void print() {
    System.out.println("Schedule Name: "+name);
    System.out.println("Start: "+start_time.toString().replace("T", "
"));
    System.out.println("End: "+end_time.toString().replace("T", " "));
    if(memo != null) {
        System.out.println("Memo: "+memo);
    }
}

public String getName() {
    return name;
}

```

```

    public String getStartTime() {
        return start_time.format(schedule_time);
    }

    public int getStartTime_year() {
        return start_time.getYear();
    }

    public int getStartTime_month() {
        return start_time.getMonthValue();
    }

    public int getStartTime_day() {
        return start_time.getDayOfMonth();
    }

    public String getEndTime() {
        return end_time.format(schedule_time);
    }

    public int getEndTime_year() {
        return end_time.getYear();
    }

    public int getEndTime_month() {
        return end_time.getMonthValue();
    }

    public int getEndTime_day() {
        return end_time.getDayOfMonth();
    }

    public String getMemo() {
        return memo;
    }

    //valid check
    public String errorcheck() {
        String error = "none";
        //1. Missing Element
        if(name.equals("")) {
            error = "noName";
        }
        else
        if(start_time.equals(LocalDateTime.MAX)&&end_time.equals(LocalDateTime.MIN)) {
            error = "wrongTimeFormat";
        }
        else if(start_time.compareTo(end_time) > 0) {
            error = "timeParadox";
        }
        else;
        return error;
    }

}

```

2-5. ScheduleList 클래스: 일정 리스트

```
import java.util.ArrayList;
import java.util.Scanner;
import java.io.PrintWriter;
import java.io.File;
import java.io.FileNotFoundException;

public class ScheduleList {
    //data field
    private ArrayList<Schedule> list_of_schedule = new ArrayList<Schedule>();
    private String fileName;

    ScheduleList(){

    }

    //reads file and make a list
    ScheduleList(String fileName){
        try {
            this.fileName = fileName;
            File file = new File(fileName);
            Scanner scan = new Scanner(file);
            int i = 0;
            while(scan.hasNext()) {
                String temp = scan.nextLine();
                if(temp.equals("")) { //빈줄 스킵
                }
                else if(temp.substring(0,1).equals(";")) { //;뒤의
코멘트는 모두 주석처리되어 스킵됨.
                }
                else {
                    Schedule s = new Schedule(temp);
                    String error = s.errorcheck();
                    if(error.equals("none")) {
                        list_of_schedule.add(s);
                    }
                    else if(error.equals("noName")) {
                        //System.out.println("No schedule
name: invalid: skip input line: "+temp);
                    }
                    else if(error.equals("wrongTimeFormat")) {
                        //System.out.println("Wrong time
format: invalid: skip input line: "+temp);
                    }
                    else if(error.equals("timeParadox")) {
                        //System.out.println("Start time is
set after End time: invalid: skip input line: "+temp);
                    }
                    else;

                }
            }
            scan.close();
        }catch(FileNotFoundException e){
            System.out.println("Unknown File");
        }
    }
}
```

```

    }
}

//show the number of schedules in this list
public int numSchedules() {
    return list_of_schedule.size();
}
//show the ith schedule in this list
public Schedule getSchedule(int i) {
    return list_of_schedule.get(i);
}

public void addSchedule(Schedule s) {
    list_of_schedule.add(s);
}

public void removeSchedule(int i) {
    list_of_schedule.remove(i);
}

public void removeSchedule(Object o) { //overloading
    list_of_schedule.remove(o);
}

public void removeError() { //비정상 스케줄 지우기
    for(int i = 0; i < numSchedules(); i++) {
        String error = getSchedule(i).errorcheck();
        if(error.equals("none")) {

        }
        else if(error.equals("noName")) {
            removeSchedule(i);
        }
        else if(error.equals("wrongTimeFormat")) {
            removeSchedule(i);
        }
        else if(error.equals("timeParadox")) {
            removeSchedule(i);
        }
        else;
    }
}

public void writeScheduleList() {
    try {
        File file = new File(fileName);
        PrintWriter p = new PrintWriter(file);
        for(int i = 0; i < numSchedules(); i++)
{ //list_of_schedule.length로 하지 말고... numSchedules()로 했어야 했다...
            //print나 write는 자동으로 개행이 안되고 println만 된다.
            p.println("");
            p.print(getSchedule(i).getName()+"//");
            p.print(getSchedule(i).getStartTime()+"//");
            p.print(getSchedule(i).getEndTime()+"//");
            p.println(getSchedule(i).getMemo());

```

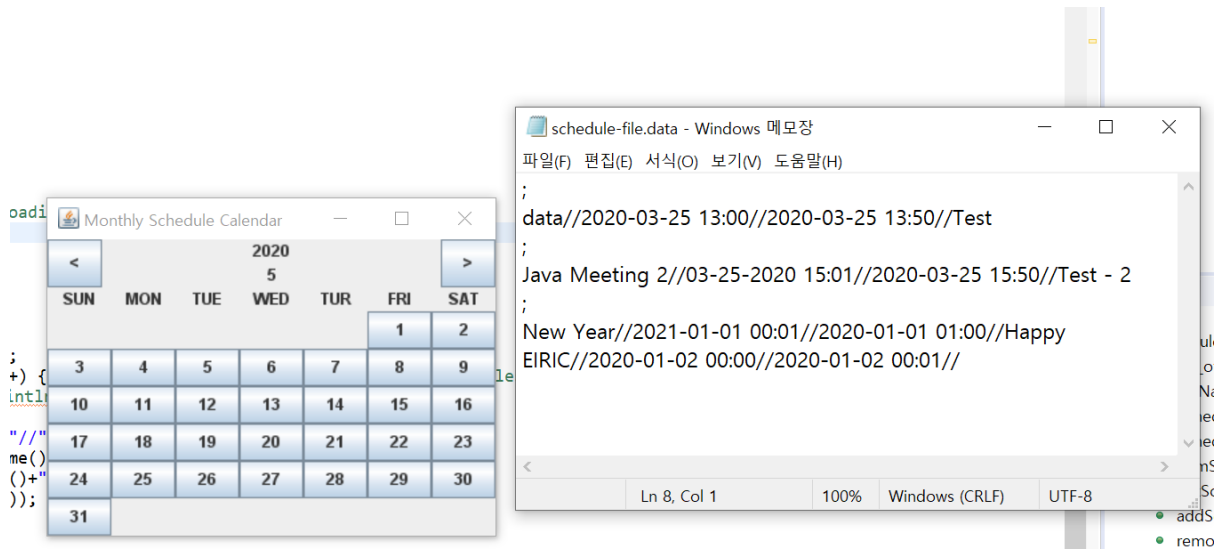
```

    }
    //memory leak 방지
    p.close();
}
catch(FileNotFoundException e){
    System.out.println("Unknown File");
}
}
}

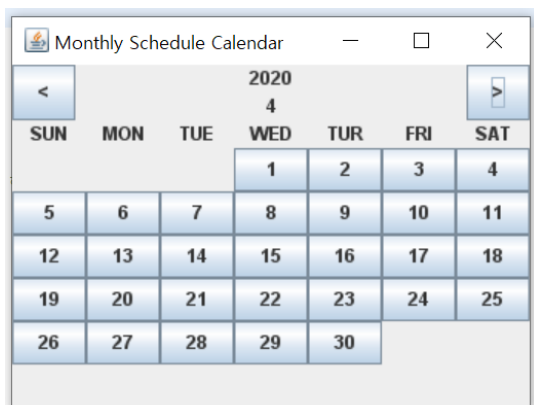
```

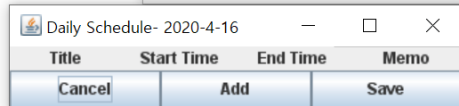
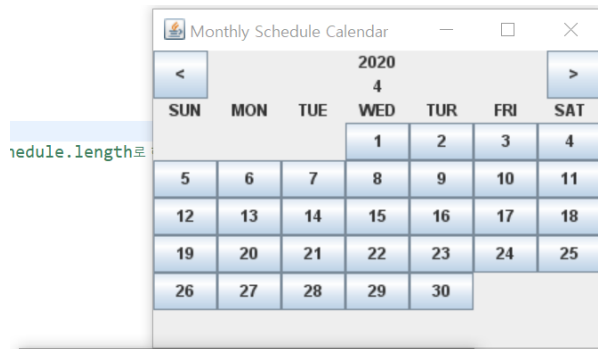
3. 최종 테스트 과정 화면

1. 초기 동작

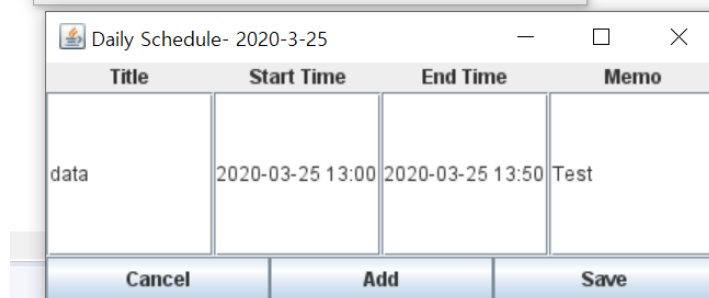
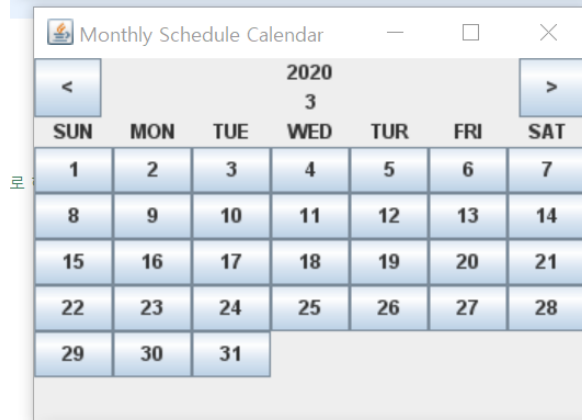
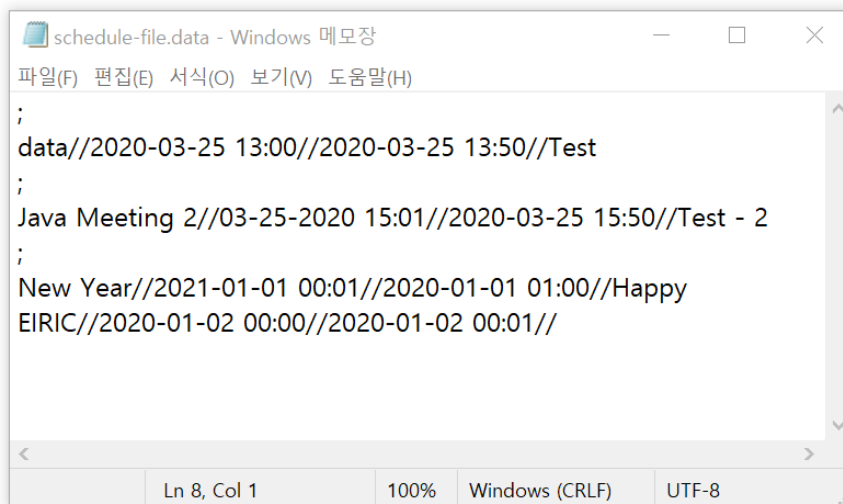


처음 실행했을 때 뜨는 GUI 화면과 데이터베이스 파일의 상태.



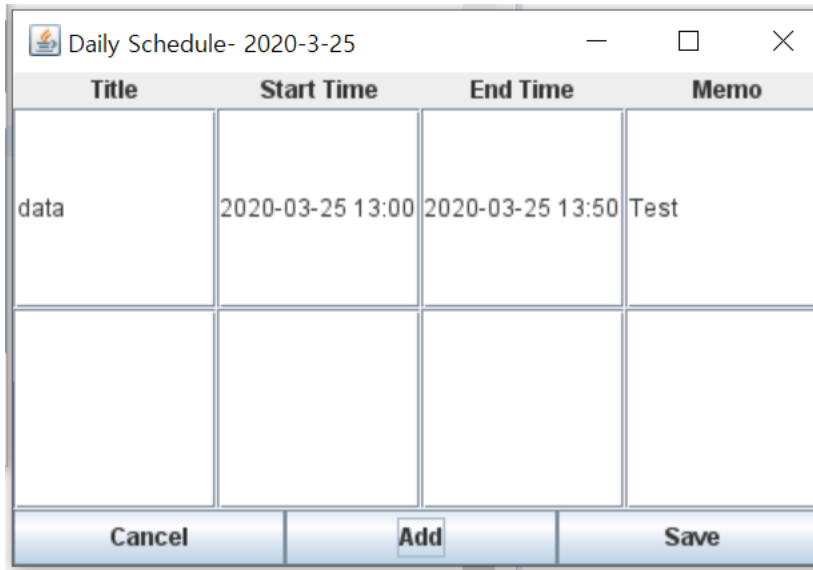


월별이동도 잘되고 일정 없는 날도 창 잘뜸.



정상일정이 제대로 추가된 모습. 비정상일정은 표시되지 않음.

2. add 동작

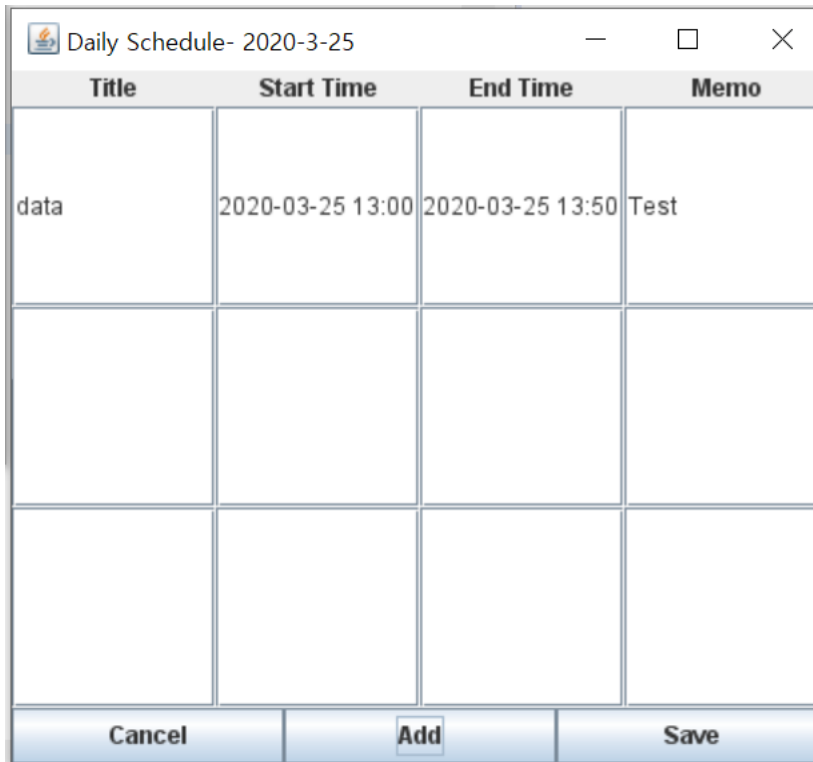


The screenshot shows a window titled "Daily Schedule- 2020-3-25". It contains a table with four columns: "Title", "Start Time", "End Time", and "Memo". The first row of the table contains the text "data", "2020-03-25 13:00", "2020-03-25 13:50", and "Test". Below the table, there are three buttons: "Cancel", "Add", and "Save". The "Add" button is highlighted with a blue border.

Title	Start Time	End Time	Memo
data	2020-03-25 13:00	2020-03-25 13:50	Test

Cancel Add Save

add버튼을 누르면 이렇게 한 줄이 추가됨.



The screenshot shows the same window as before, but now the table has two rows of data. The second row is empty. The "Add" button remains highlighted.

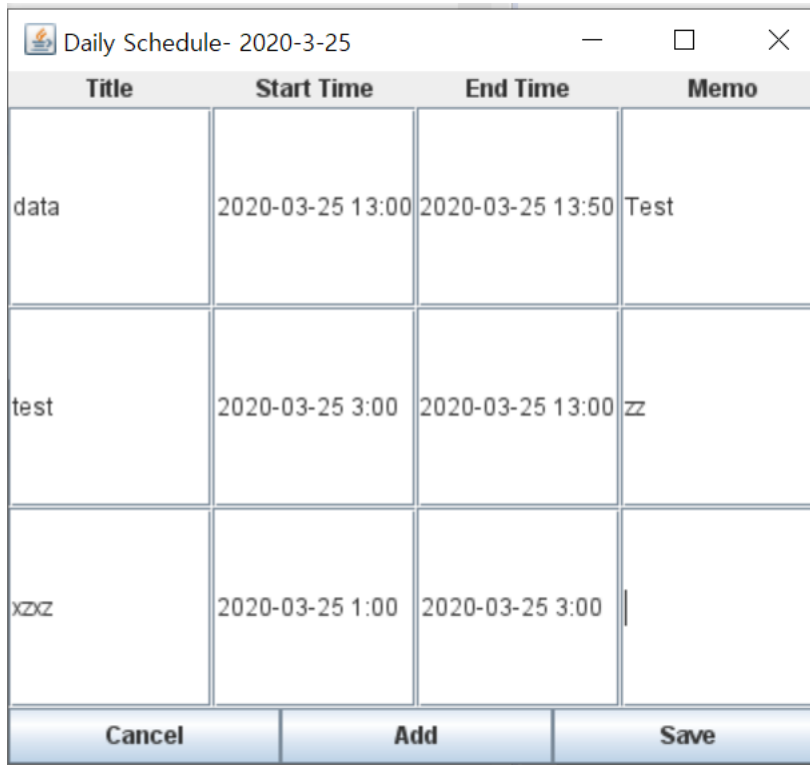
Title	Start Time	End Time	Memo
data	2020-03-25 13:00	2020-03-25 13:50	Test

Cancel Add Save

두줄도 추가할수있음.

3. cancel 동작

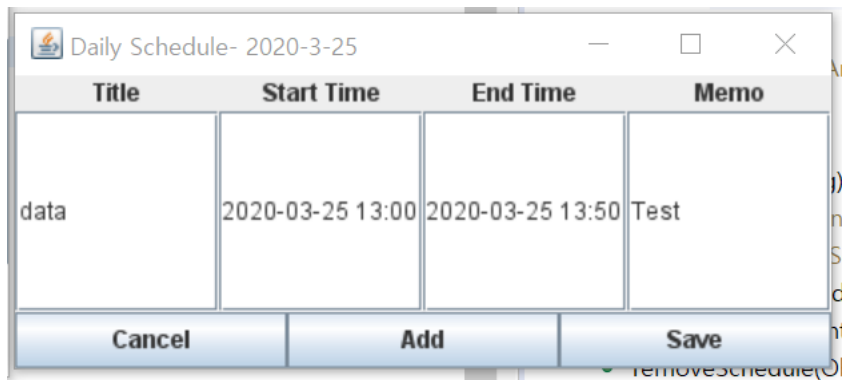
위에서 두 줄을 add한 일별 일정창에 일정을 써본다.



Title	Start Time	End Time	Memo
data	2020-03-25 13:00	2020-03-25 13:50	Test
test	2020-03-25 3:00	2020-03-25 13:00	zz
xzxz	2020-03-25 1:00	2020-03-25 3:00	

Cancel Add Save

모두 제대로 된 일정이다. 이제 이것 save하지 않고 그대로 cancel 버튼을 누른다. 다시 3월 25일 버튼을 누르면

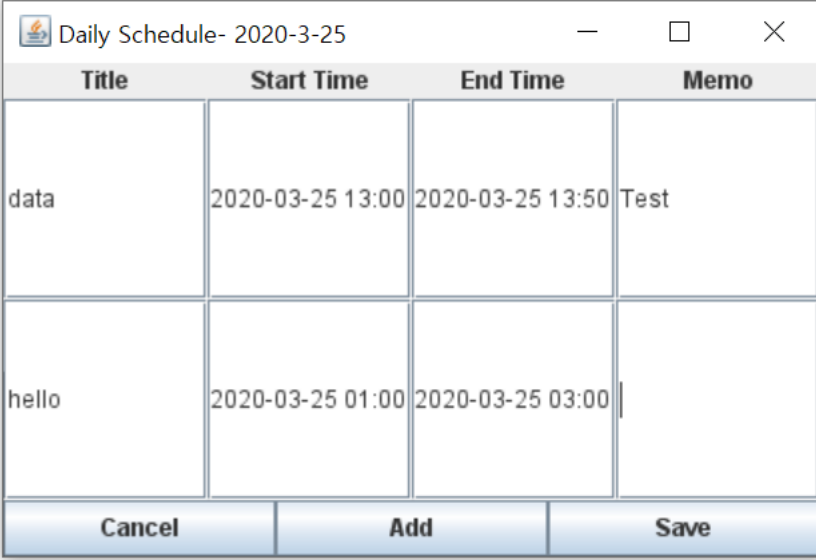


Title	Start Time	End Time	Memo
data	2020-03-25 13:00	2020-03-25 13:50	Test

Cancel Add Save

바꾸기 전 바로 그 상태로 돌아가 있다.

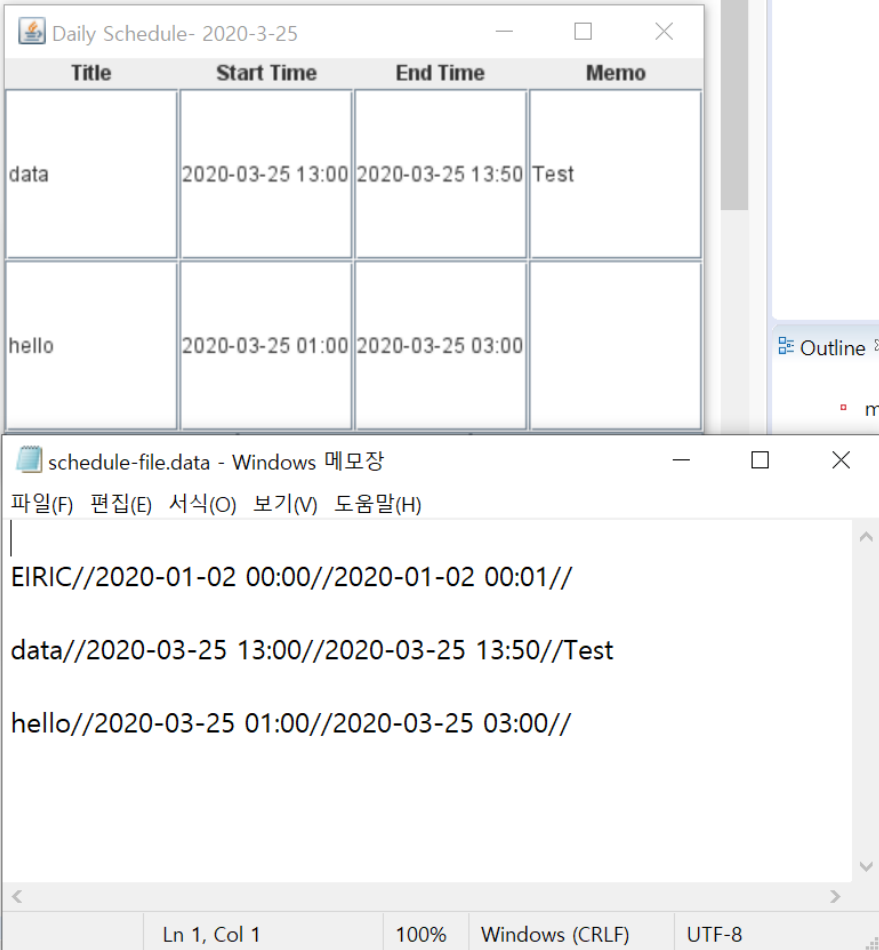
4. save 동작



Title	Start Time	End Time	Memo
data	2020-03-25 13:00	2020-03-25 13:50	Test
hello	2020-03-25 01:00	2020-03-25 03:00	

Buttons: Cancel, Add, Save

일정을 하나 추가해본다. Add를 누르고 정상일정을 기록한다.



The 'Daily Schedule- 2020-3-25' window is shown with the same data as before.

The 'schedule-file.data - Windows 메모장' window shows the following text:

```
EIRIC//2020-01-02 00:00//2020-01-02 00:01//  
  
data//2020-03-25 13:00//2020-03-25 13:50//Test  
  
hello//2020-03-25 01:00//2020-03-25 03:00//
```

Notepad status bar: Ln 1, Col 1, 100%, Windows (CRLF), UTF-8

Save를 누르면 다시 버튼을 눌러도 일정이 있고 파일에도 새 일정이 추가된 모습을 볼 수 있다.

처음 있던 비정상 일정도 모두 사라진다.

5. 기타, 비정상 동작 실험

Title	Start Time	End Time	Memo
Java Meeting 2	2020-03-25 15:01	2020-03-25 15:50	Test - 2

Cancel Add Save

Title	Start Time	End Time	Memo
	2020-03-25 15:01	2020-03-25 15:50	Test - 2

Cancel Add Save

원래 있던 일정에서 제목을 지우고 save 해 보았다.

Title	Start Time	End Time	Memo
	2020-03-25 15:01	2020-03-25 15:50	Test - 2

Cancel Add Save

비정상 일정으로 체크되어 일정이 없어진 모습이다! 이것이 Delete 기능을 간접적으로 구현한 모습이다.

Title	Start Time	End Time	Memo
j	2020-05-26 14:00	2020-05-26 15:00	?

Cancel Add Save

정상 일정이긴 한데, 날짜가 다르다면 어떻게 될까? 이 일정을 save 하면

Daily Schedule- 2020-5-21

Title	Start Time	End Time	Memo
Cancel	Add	Save	

여기에 추가되진 않고

Daily Schedule- 2020-5-26

Title	Start Time	End Time	Memo
j	2020-05-26 14:00	2020-05-26 15:00	?
Cancel	Add	Save	

맞는 날짜에 알아서 찾아간다.

Daily Schedule- 2020-5-14

Title	Start Time	End Time	Memo
fjkd		2020-05-14 15:30	
Cancel	Add	Save	

시작시간이 없는 일정을 save하면?

Daily Schedule- 2020-5-14

Title	Start Time	End Time	Memo
Cancel	Add	Save	

당연히 추가가 안된다.

schedule-file.data - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```

EIRIC//2020-01-02 00:00//2020-01-02 00:01//
j//2020-05-26 14:00//2020-05-26 15:00//?

```

(파일에도 마찬가지로)

4. 자체 평가표

평가 항목	학생 자체 평가	평가	점수
<p>완성도(동작 여부)</p> <p>-초기 동작</p> <p>-add 동작</p> <p>-cancel 동작</p> <p>-save 동작</p> <p>-기타 비정상 동작 실험</p>	<p>3. 최종 테스트 과정 화면</p> <p>에 설명</p>		
<p>설계 노트</p> <p>-주요 결정사항 및 근거</p> <p>-한계/문제점</p> <p>-해결 방안</p> <p>-필수 내용:</p> <p>*프로그램 구성(Class 구조)</p> <p>*member visibility</p>	<p>1. 설계 노트</p> <p>에 설명</p> <p>-> 프로그램 구성(클래스 구조)는 찾기 쉽게 표시하고 하이라이트 해 놓았습니다. 설계 노트 자체가 프로그램 작동 순서를 따라가면서 설명하도록 해 놓았습니다.</p> <p>프로그램 구성 이외에 주요 결정사항, 근거, 한계, 그에 대한 해결 방안도 설계 노트에 모두 서술해놓았습니다.(하이라이트 되지 않음)</p>		
<p>리포트</p> <p>-평가자 시각으로 리포트 검토</p> <p>-위의 평가 요소들이 명확하게 기술되었는가?</p>			
총평/계			