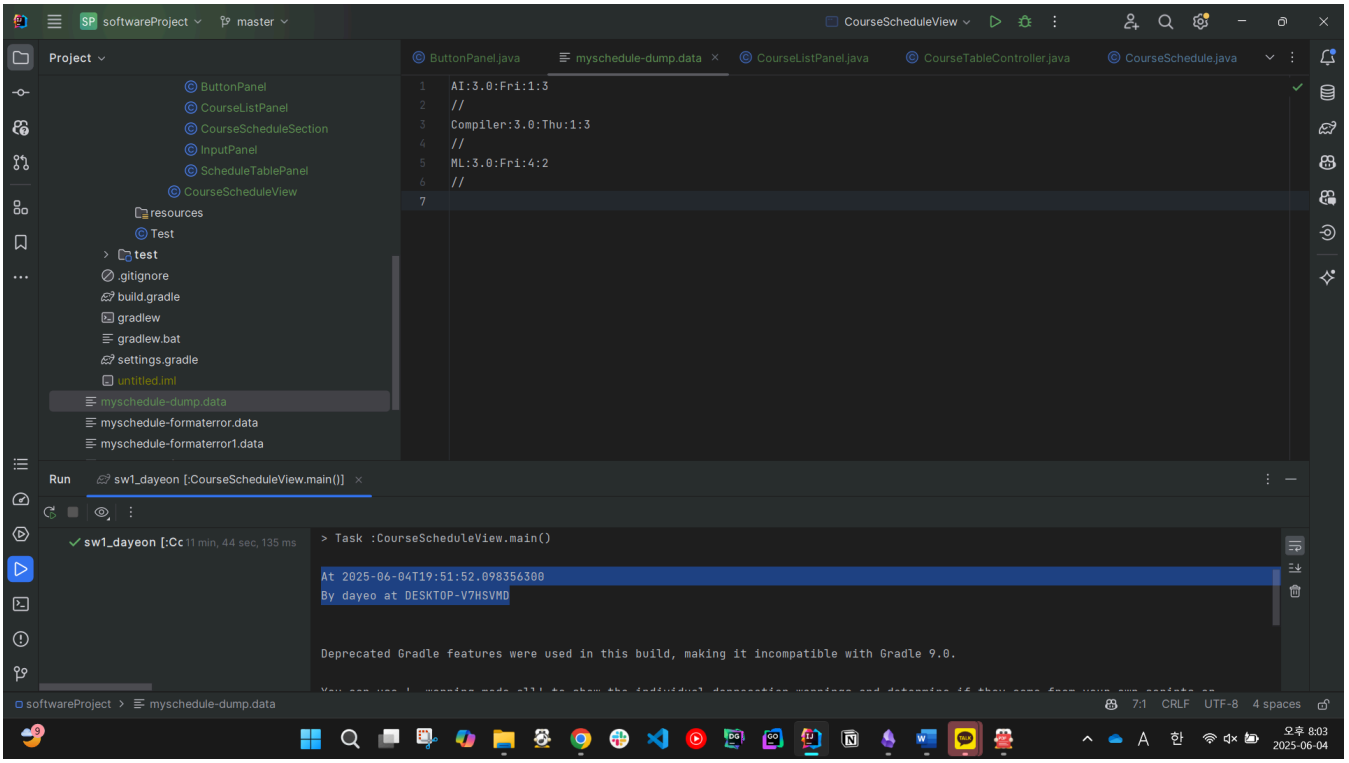
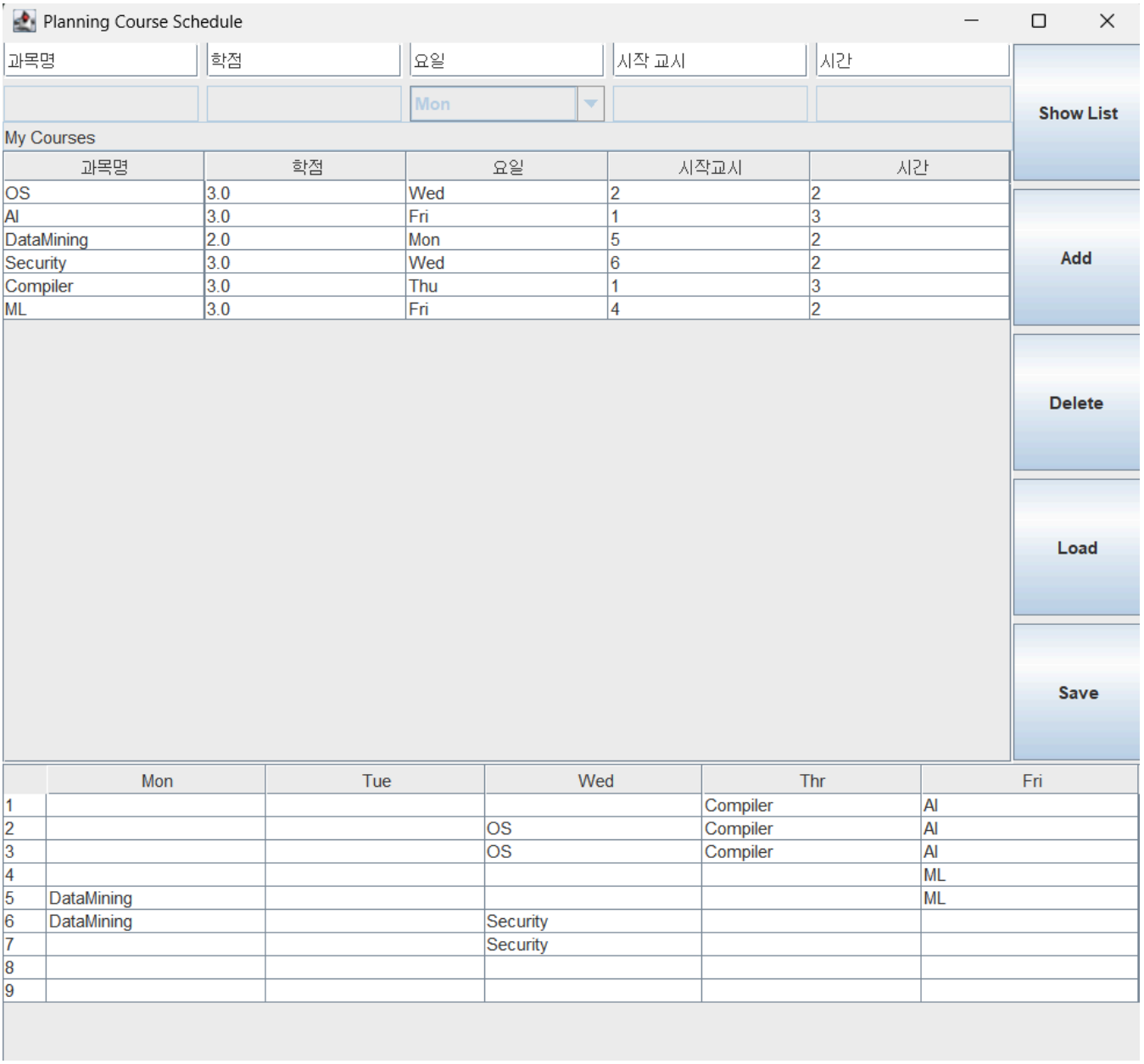
구현 동작

본인인증

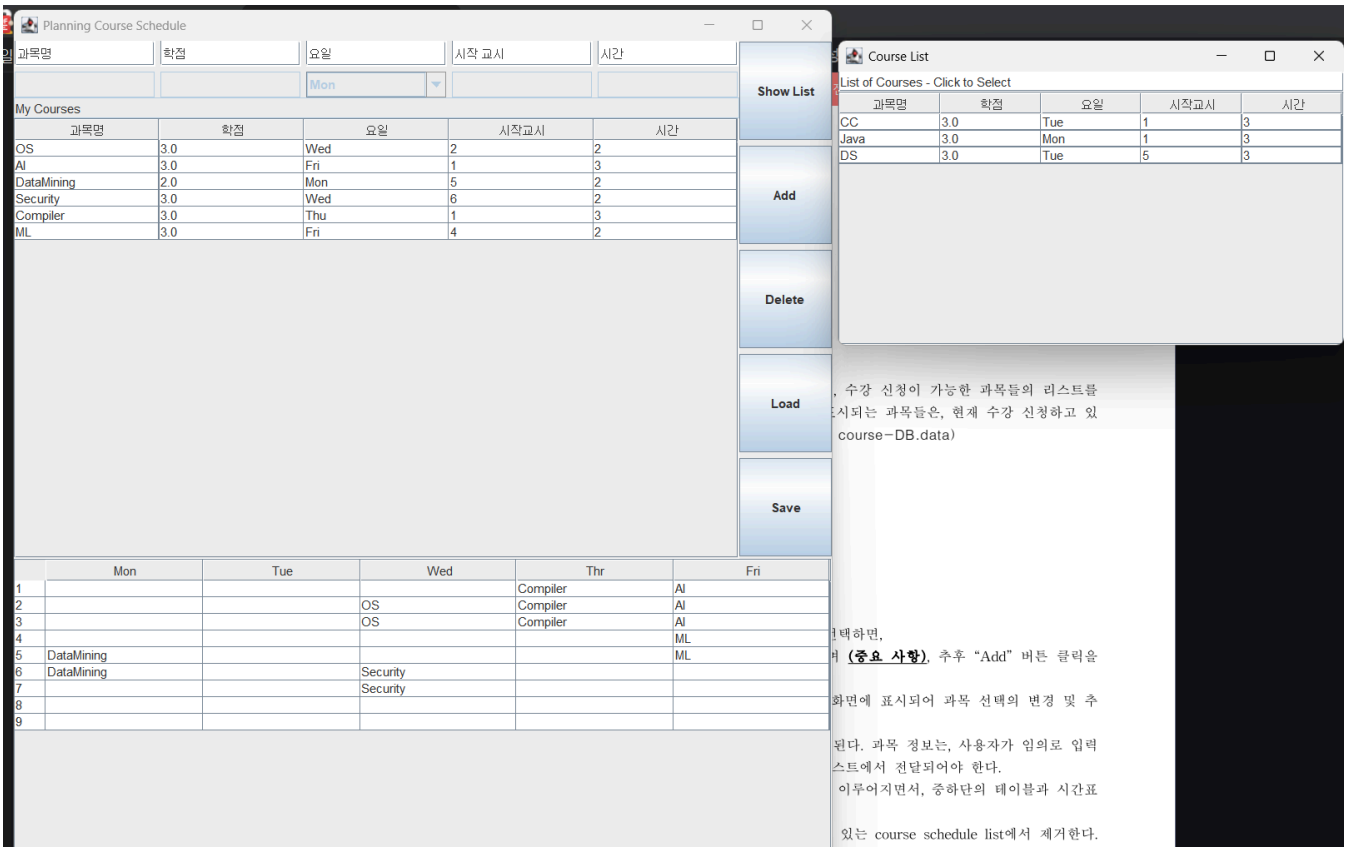


초기 화면



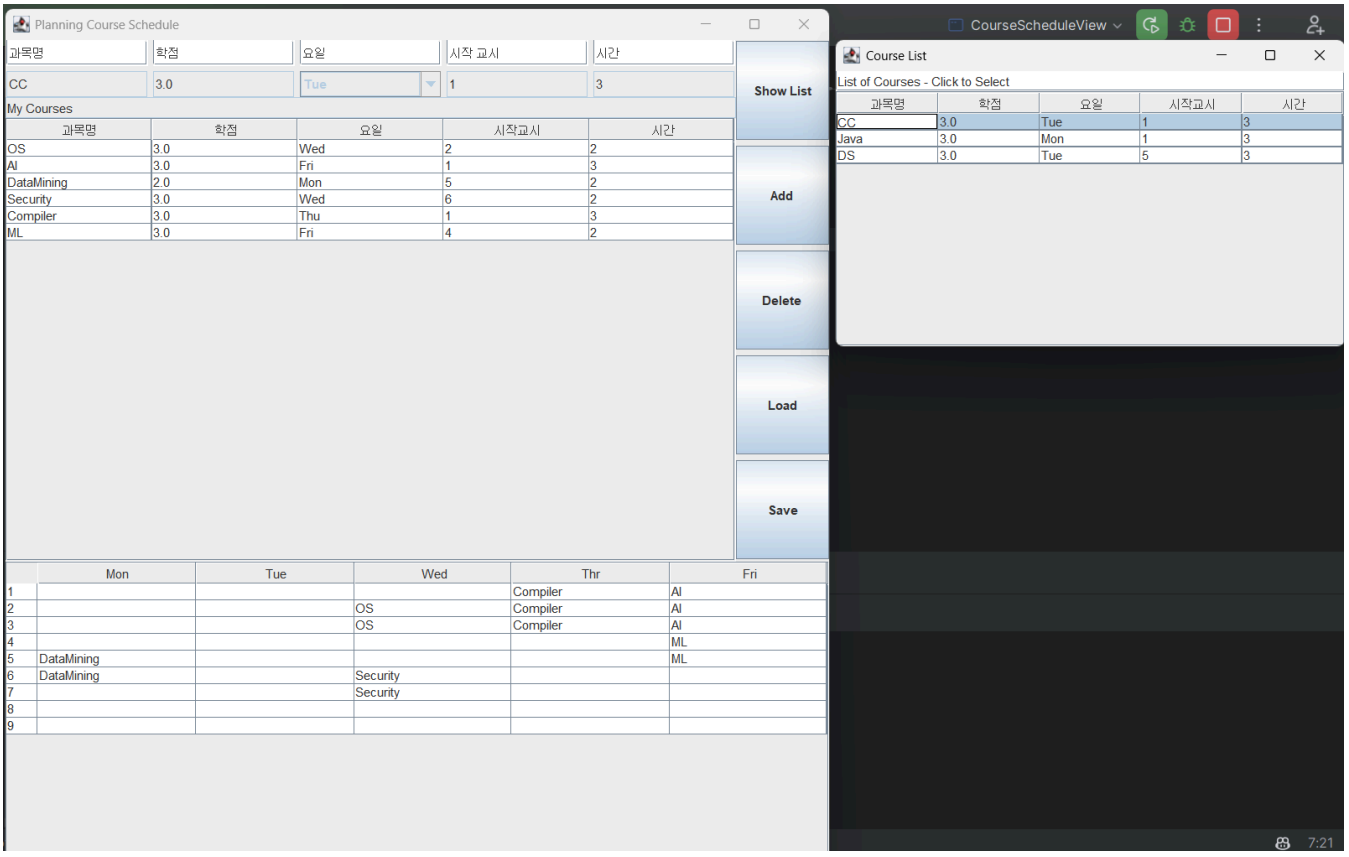
위와 같이 초기화면을 구성하였다. 과목 리스트의 선택을 통해 과목이 입력되어야 하기에 정보 창을 현재는 수정 불가능하도록 구성하였다. 현재 수강신청된 과목 정보의 경우 myschedule-dump.data의 파일을 불러오는 방식으로 위와 같이 불러온 초기 화면을 볼 수 있다.

ShowList



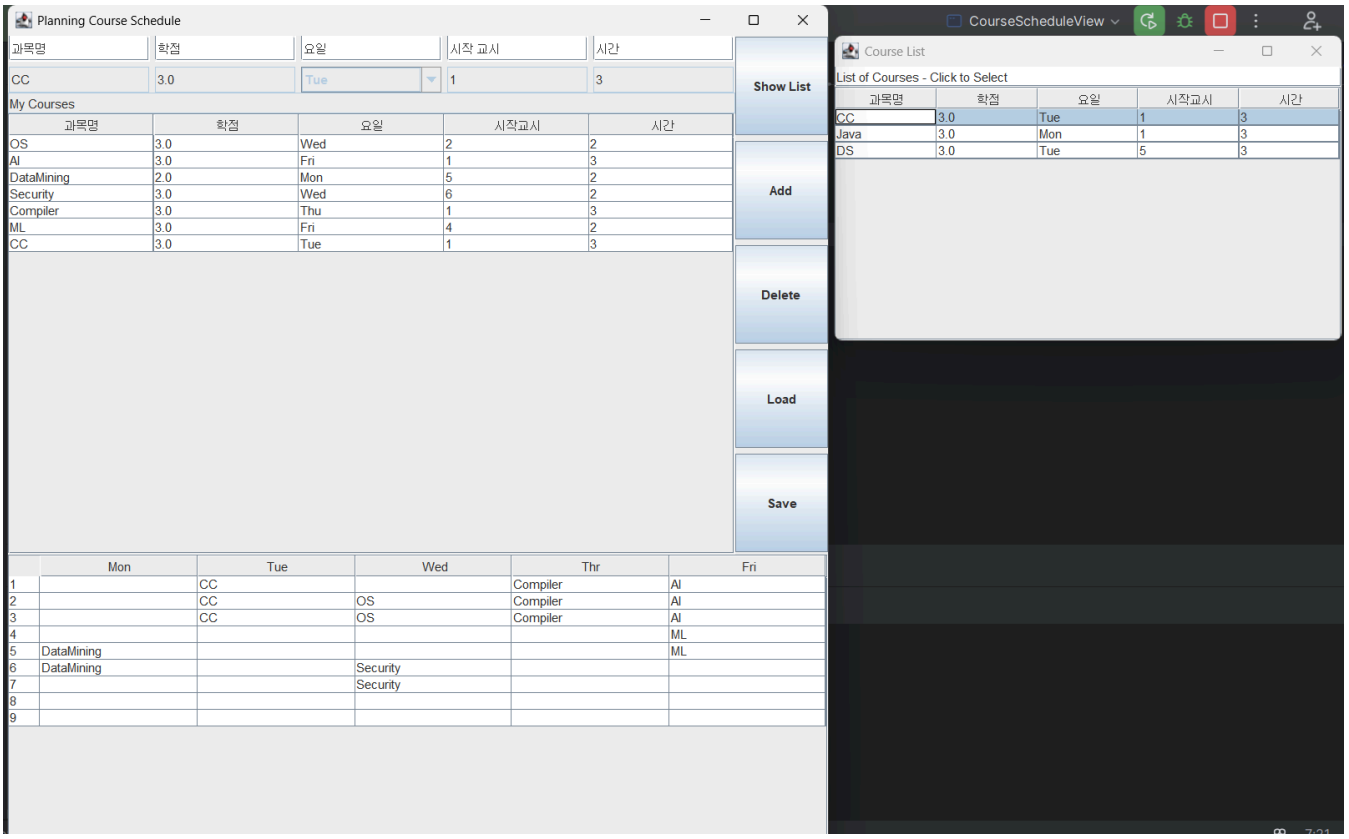
showList 버튼을 클릭하면 우측과 같은 showList의 myschedule-norm.data에서 가져오는 수강신청이 가능한 과목 리스트를 불러올 수 있다. 해당 테이블을 클릭하면 과목 선택이 가능하다.

과목 선택



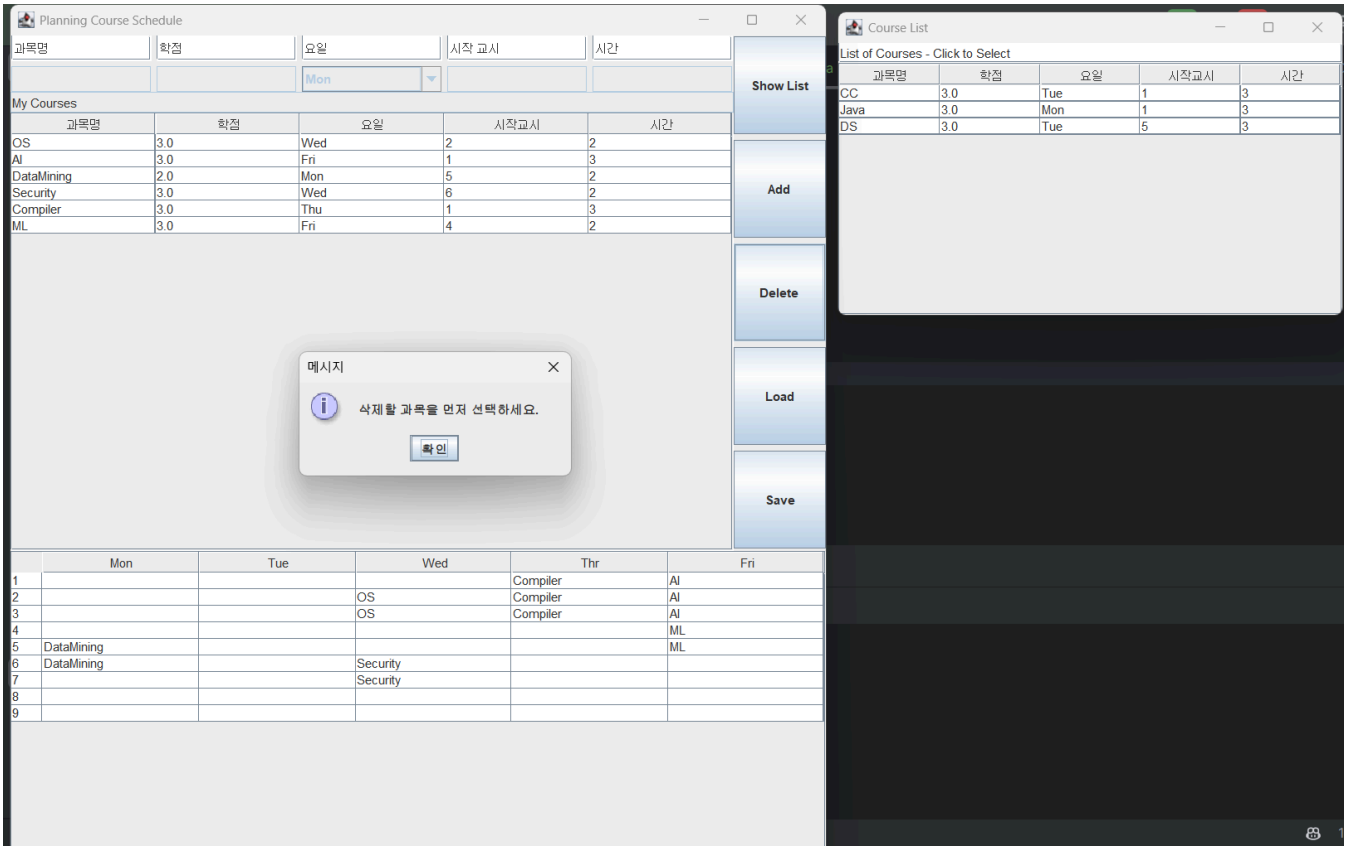
우측과 같이 CC로 되어있는 과목을 클릭한 모습이다. 과목을 클릭함에 따라 좌측에 있는 Planning Course Schedule의 상단에 선택된 내용이 입력된 것을 확인할 수 있다.

ADD

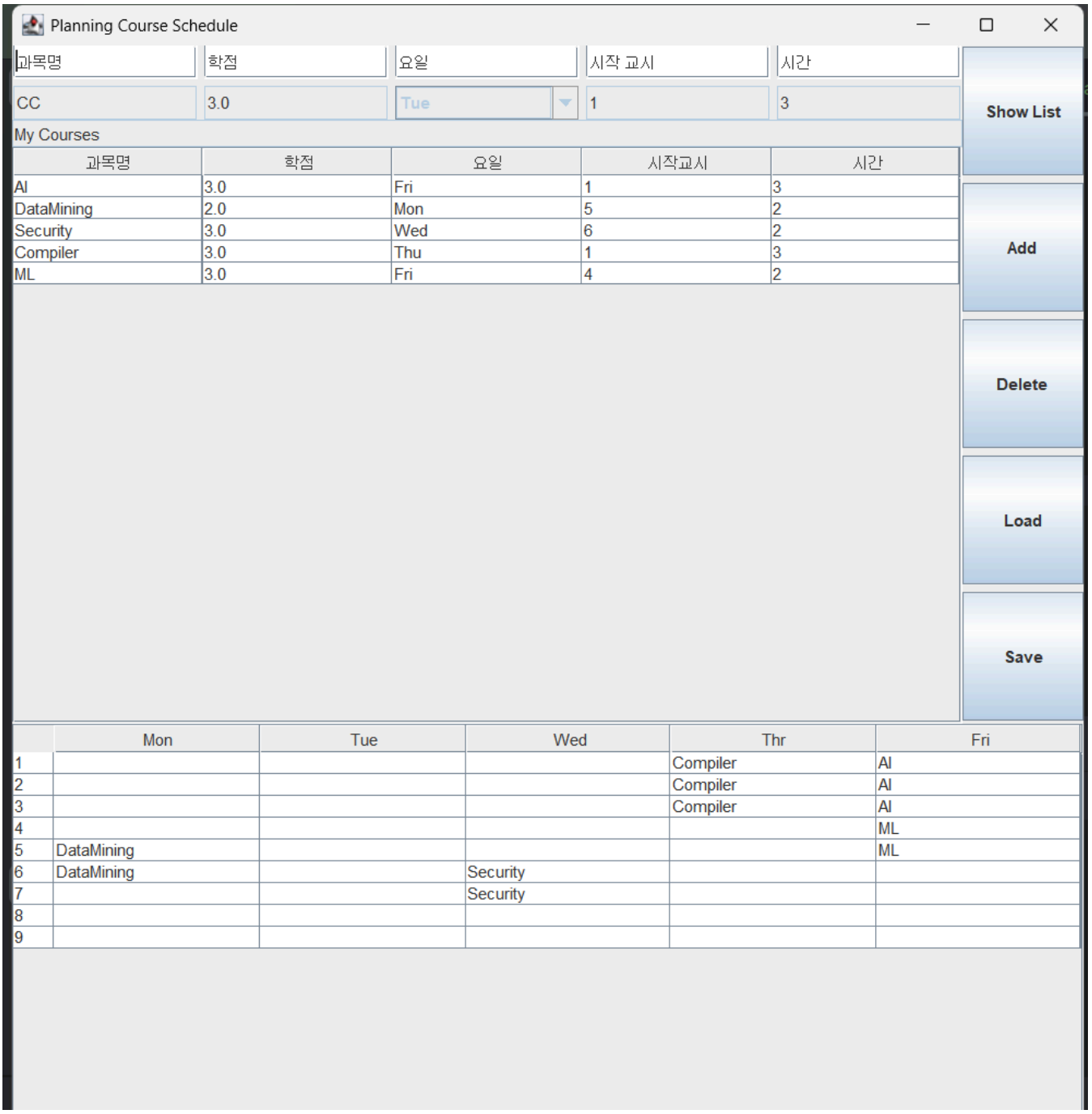


기존에 CourseList에 선택한 내용을 ADD 버튼을 누르면 이전의 화면에서 CC까 추가되어 등록된 모습을 위 화면과 같이 확인 할 수 있다.

DELETE(선택 X시 경고화면)

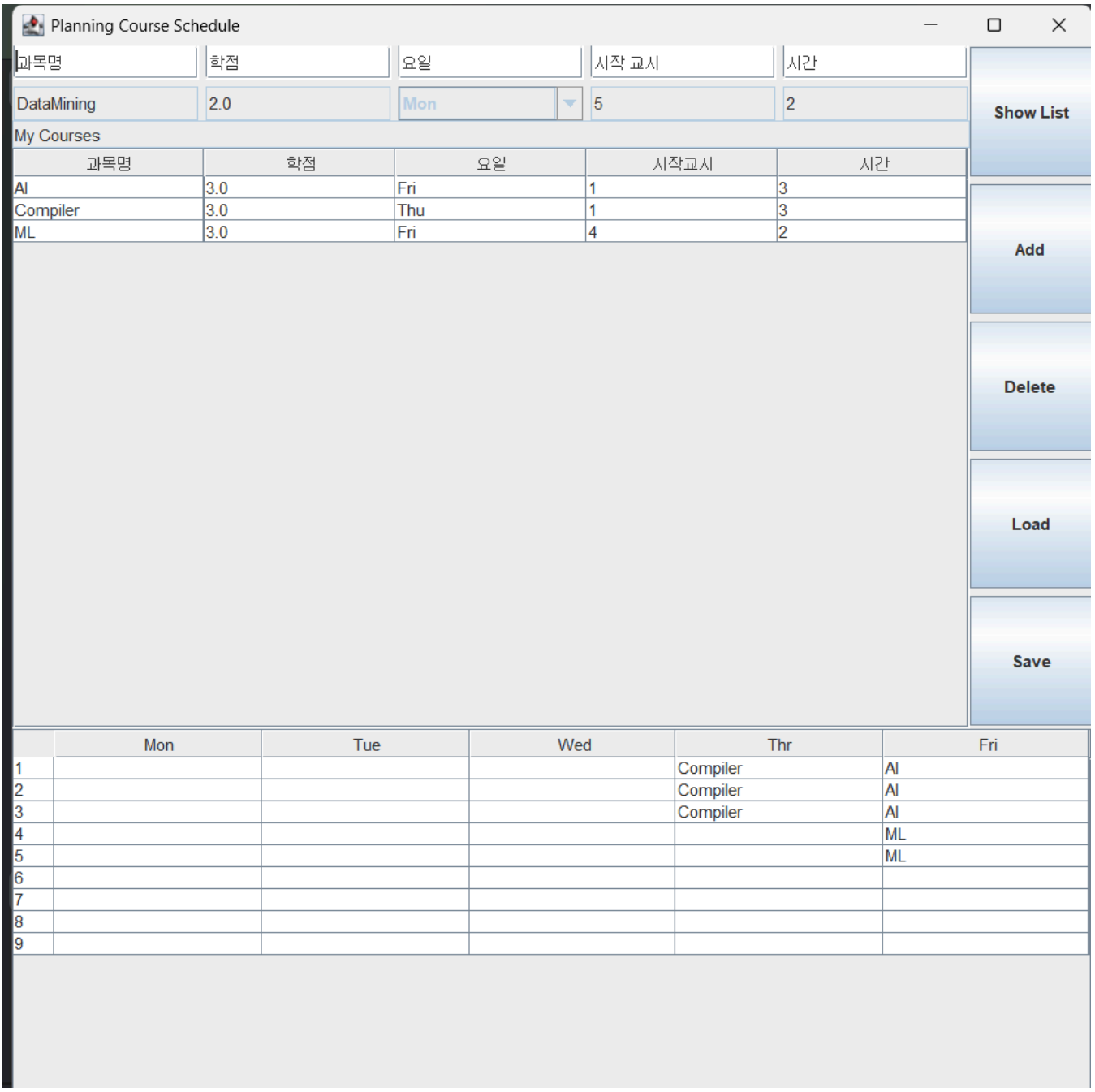


DELETE를 테스트할 때, 선택된 과목이 아무것도 없다면 삭제할 과목을 먼저 선택하세요 라는 GUI 화면이 나와서 경고를 띄우도록 화면을 구성하였다.

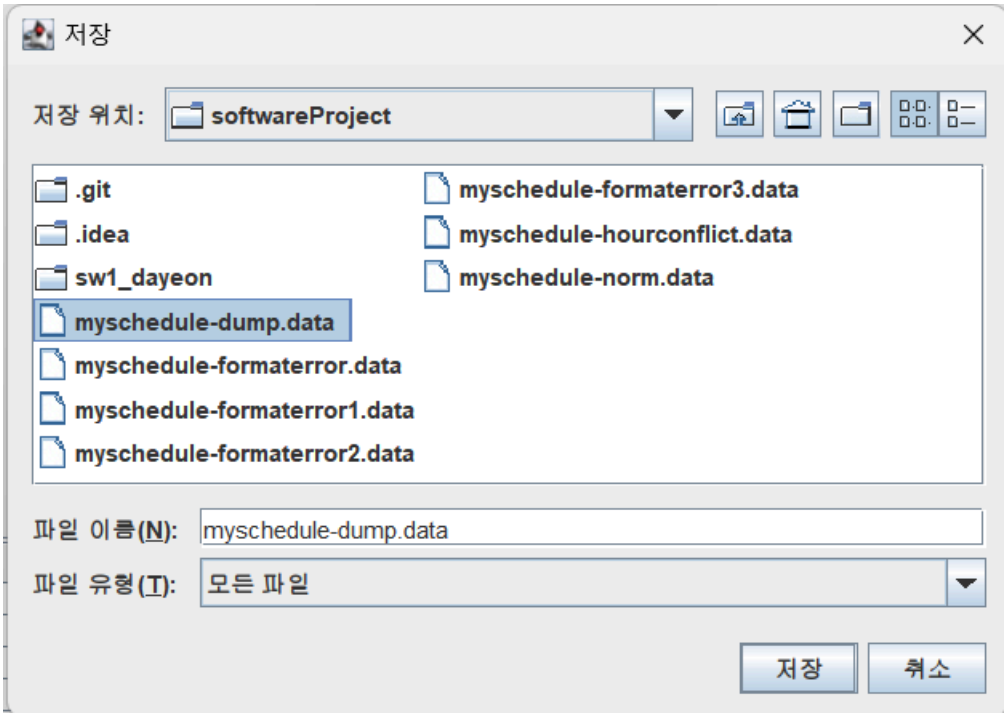


이전의 화면에서 OS를 클릭하여 DELETE 한 모습이다. 테이블과 위 목록에서 모두 삭제된 것을 확인할 수 있다.

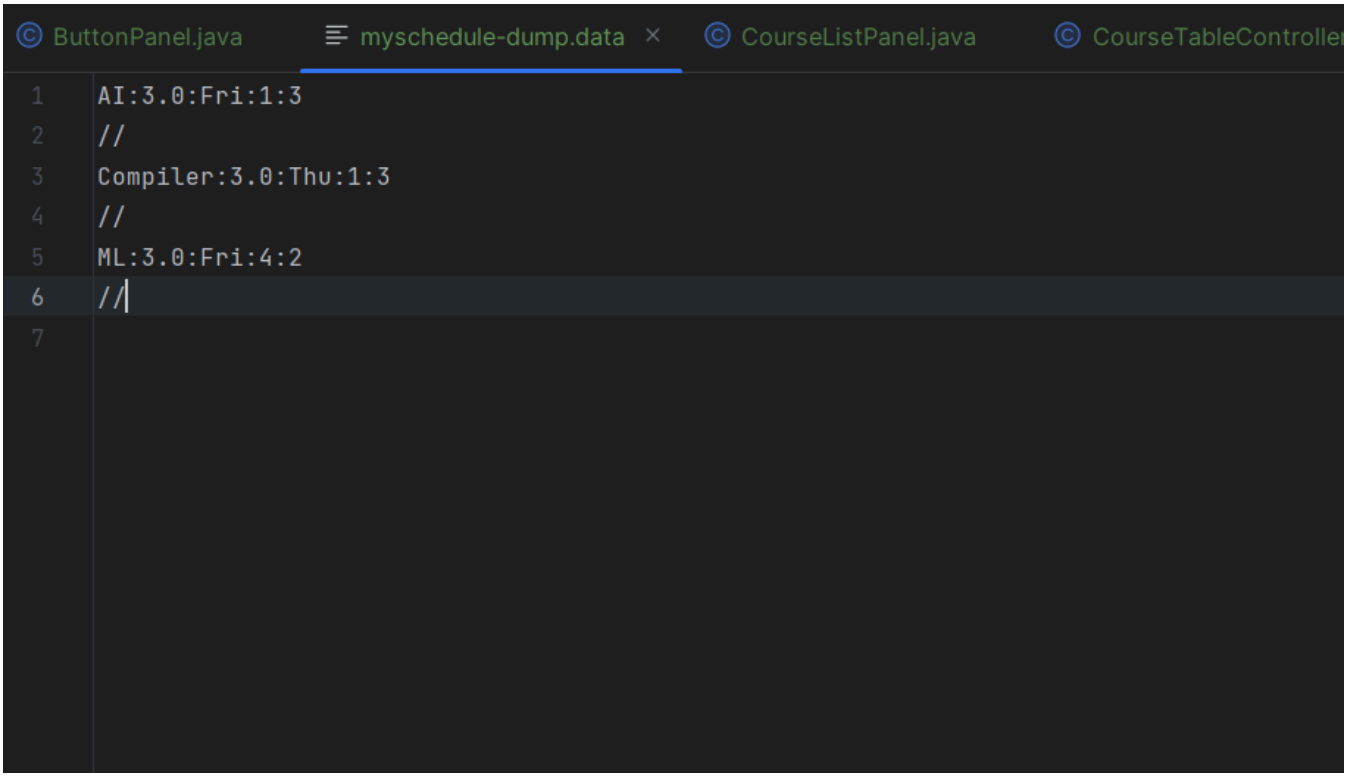
SAVE/LOAD



위와 같은 화면 구성에서 SAVE를 진행해보고자 한다. 이전과 다르게 과목을 몇개 삭제하였고, 3가지 과목만 수강신청했음을 가정하여 해당 상황에서 SAVE를 진행한다.

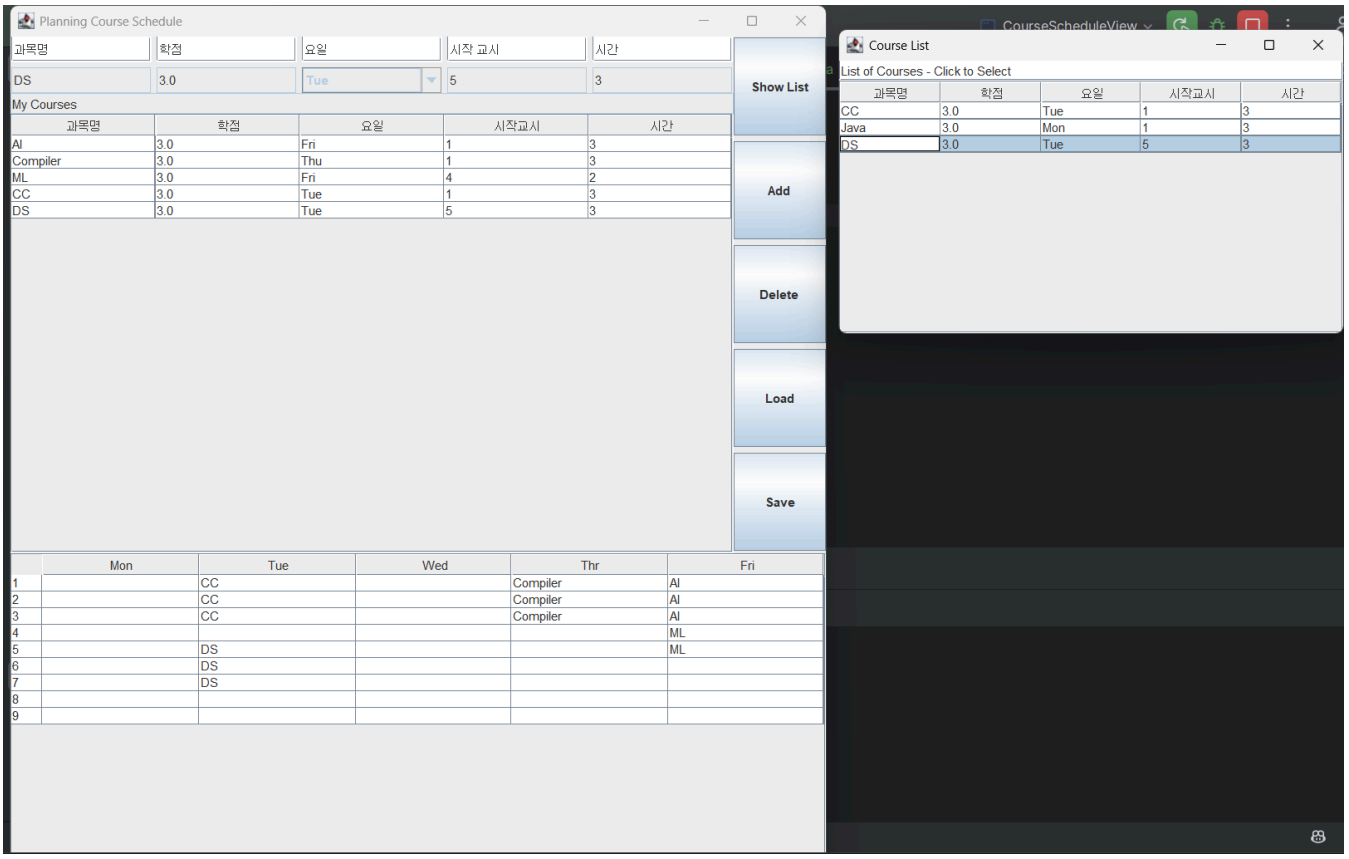


SAVE를 누르면 위와 같은 저장 위치를 택할 수 있는 GUI가 나온다. 이때, 기존처럼 수강 신청 목록은 myschedule-dump.data에 저장하도록 한다.

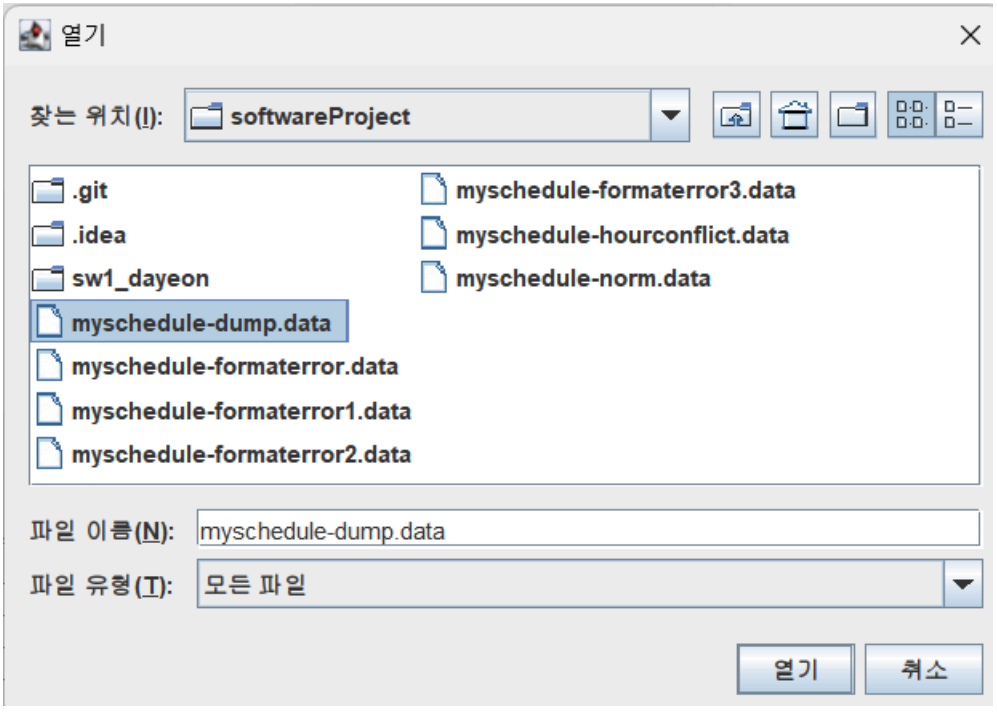


저장되어 나타나는 화면이다. 수강신청 된 3개만 나타나는 것을 볼 수 있다.

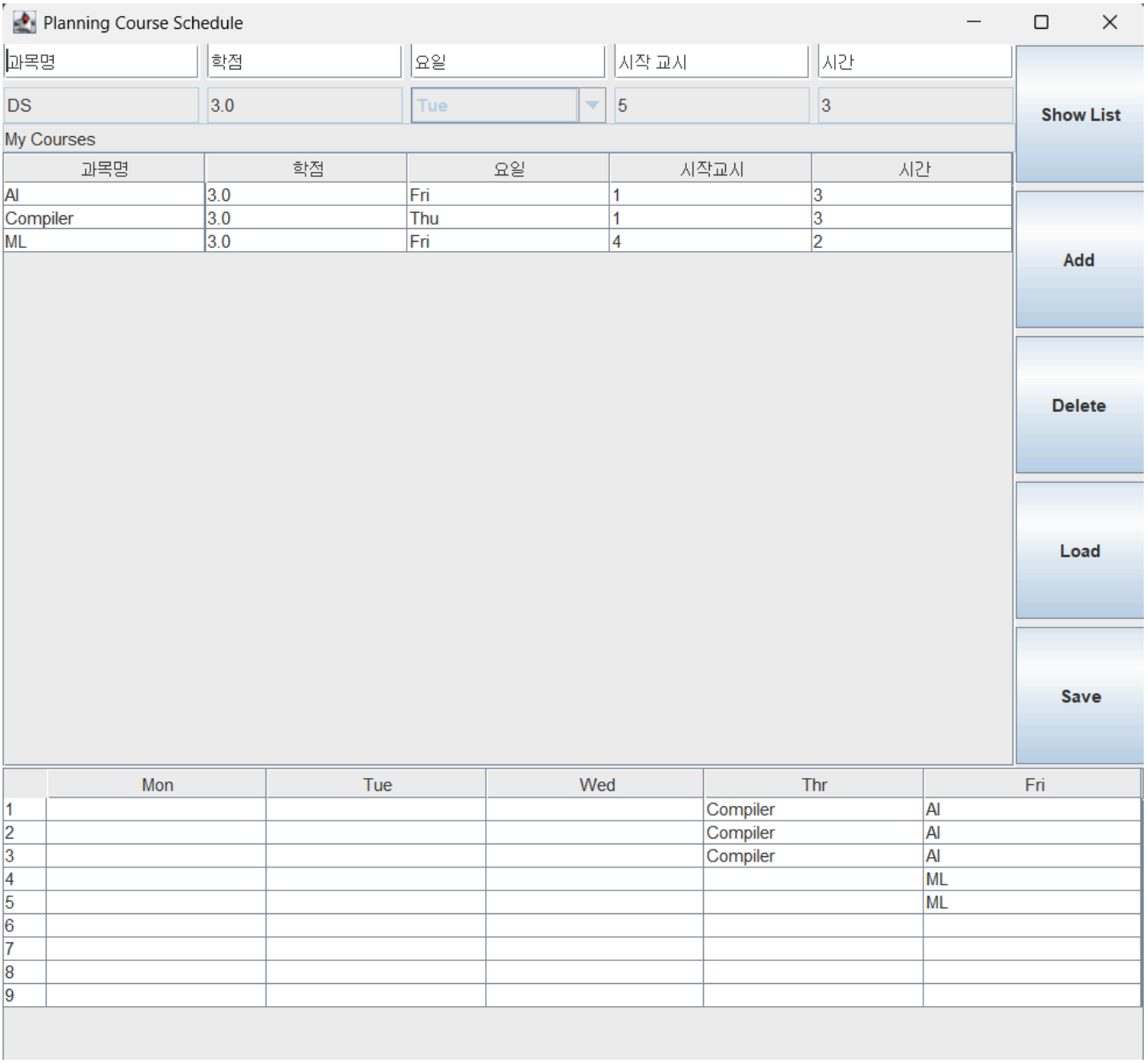
LOAD 테스트



타 과목을 위와 같이 5개로 추가한 뒤 load를 해보고자 한다.



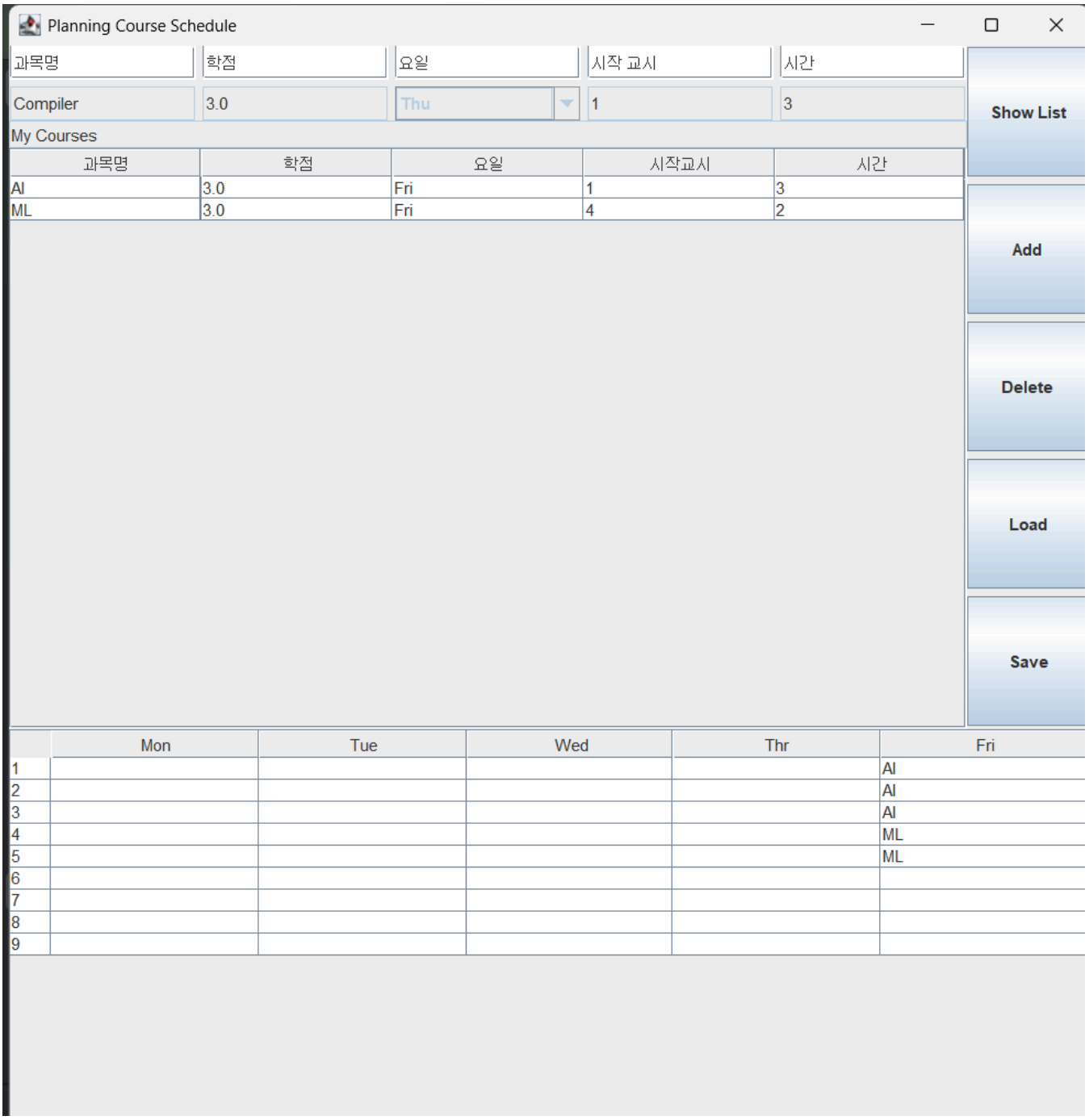
load를 누르면 파일을 선택할 수 있는 GUI가 나온다. 해당 GUI를 통해 이전에 저장한 myschedule-dump.data를 선택하여 불러온다.



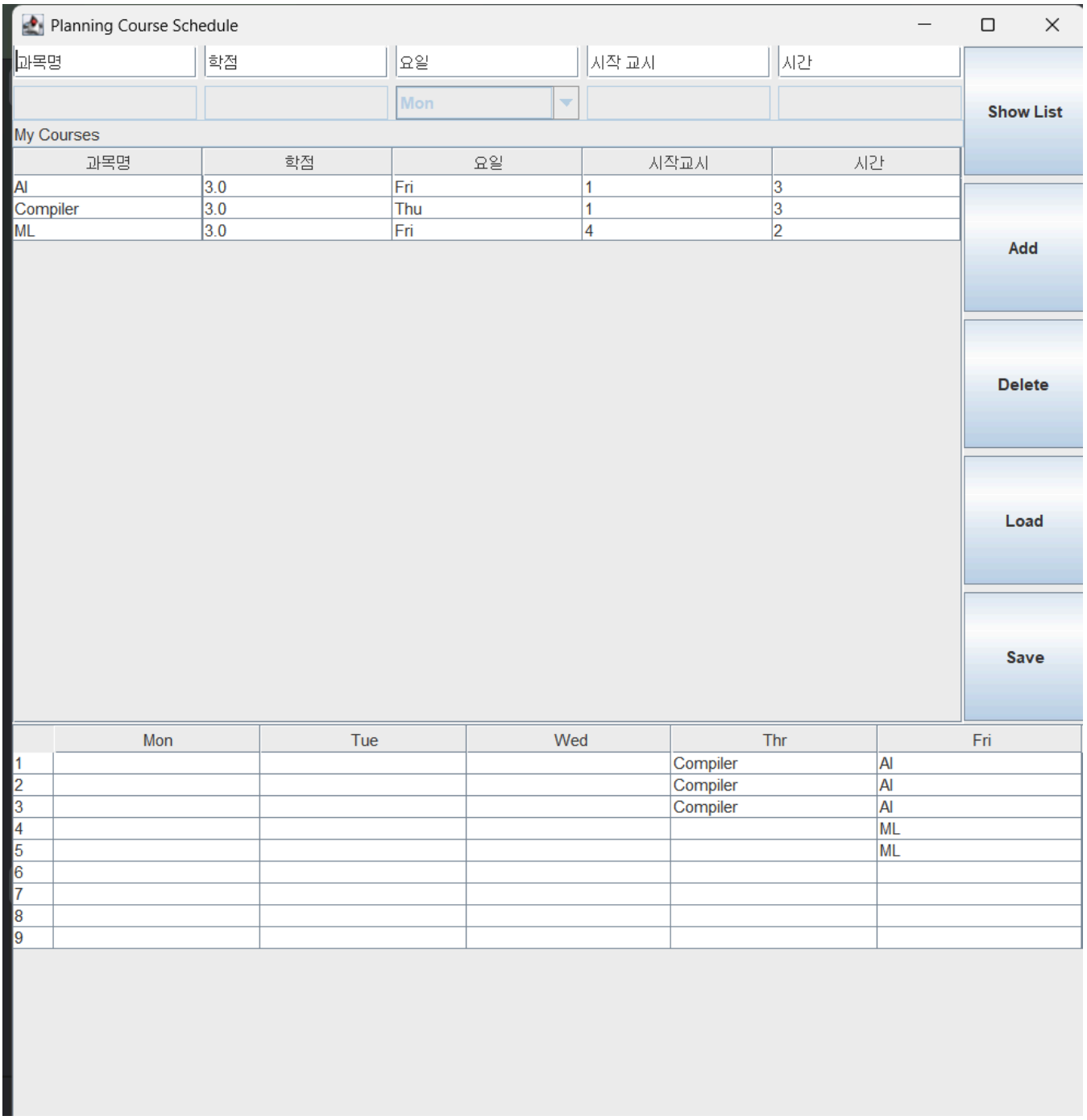
dump로 load 시 이전의 3개가 저장된 데이터 불러와지는 화면을 볼 수 있다.

Save 무효화

본 과제에서는 SAVE를 하지 않으면 정보가 갱신되지 않는다. 따라서 File을 저장하지 않은 채로 프로그램을 종료하여 SAVE가 유지되지 않는지 확인해보고자 한다.

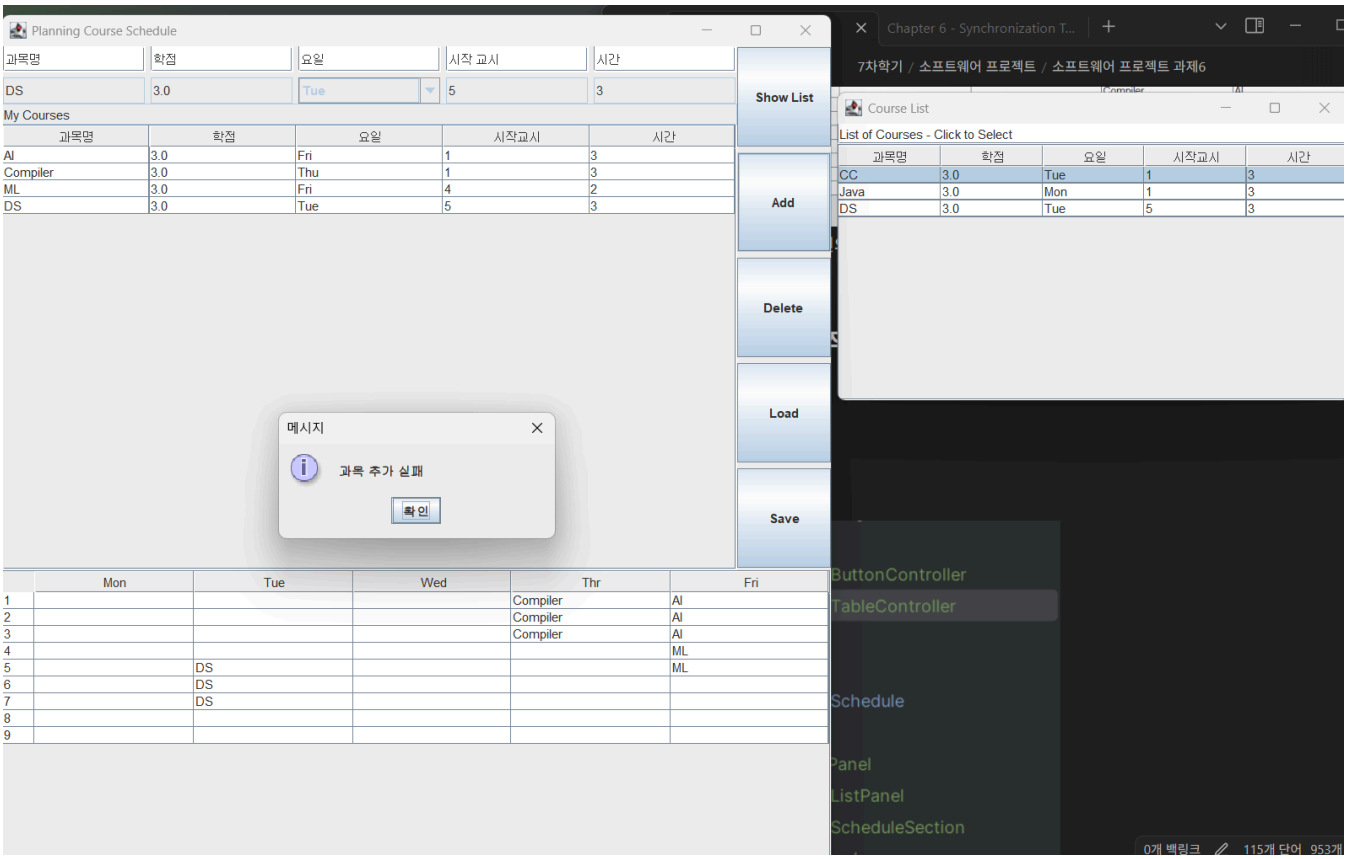


위와 같이 2과목인 상태에서 프로그램 종료를 한 뒤 열어본다.



저장이 되지 않았으므로 SAVE를 하지 않으면 무효화가 되는 것을 볼 수 있다.

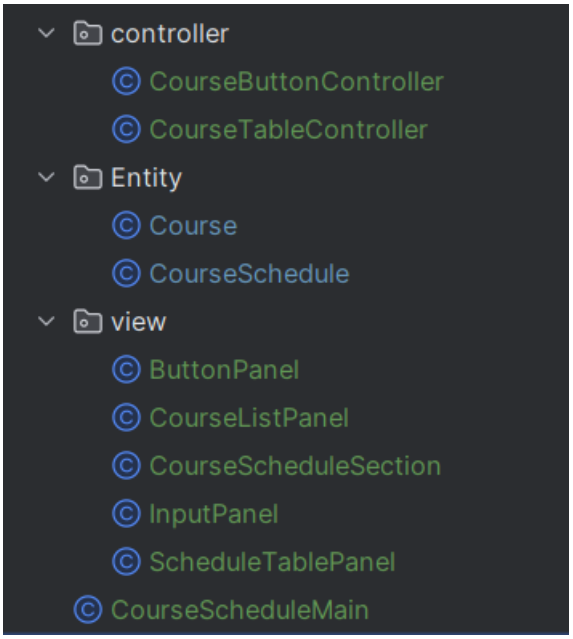
기타 비정상 동작 실험



showList에서 동일 과목 추가 혹은 추가가 불가능한 시간표 추가 시 과목 추가 실패가 나오도록 설계하였다.

설계노트

클래스 구성



다음과 같이 Event Handler는 Controller에 위치, 기존에 작성했던 Course와 CourseSchedule은 엔티티로 분류, UI는 View로 분류하였다.

- **Model(엔티티)**: `Course` 와 `CourseSchedule` 클래스는 프로그램의 핵심 데이터를 표현하며, 데이터의 로직과 상태를 관리하는 역할을 한다. 이들은 사용자 인터페이스와는 독립적인 순수 데이터 객체이므로 `Entity` 로 분류하였다.
- **View(UI)**: `ButtonPanel` , `CourseListPanel` , `CourseScheduleSection` , `InputPanel` , `ScheduleTablePanel` , `CourseScheduleView` 는 사용자와의 상호작용을 위한 화면 구성 요소들로, 시각적 표현과 레이아웃을 담당하므로 `view` 패키지로 구분하였다.

- **Controller(이벤트 처리기)**: `CourseButtonController` 와 `CourseTableController` 는 사용자 입력(Event)을 처리하여 적절한 Model 업데이트나 View 갱신을 수행한다. 이벤트 중심의 로직을 담당하므로 `controller` 로 위치시켰다.

이와 같이 역할별로 클래스를 구분함으로써 각 구성 요소 간 **의존성을 줄이고 유지보수성을 높이며**, 변경에 유연한 구조로 설계하는 것이 추후 확장에도 용이할 것으로 판단하였다. 특히 **Event Handler를 Controller에 집중시킴으로써 UI 코드와 로직의 분리를 명확히 하여**, 협업 및 기능 확장에도 용이한 구조를 만들고자 하였다.

Controller

CourseButtonController

`CourseButtonController` 는 `ButtonPanel` 에 위치한 버튼들의 클릭 이벤트를 처리하는 **Controller** 클래스로 설계하였다. 해당 클래스는 각 버튼에 대해 적절한 이벤트 리스너를 등록하고, 사용자 입력에 따라 과목 정보를 불러오거나 추가/삭제/저장하는 역할을 한다.

Member Visibility 설계

- 대부분의 **핸들러 메서드 (`addCourse()`, `deleteCourse()` 등)는 `private` 으로 선언**하여 외부 클래스에서 직접 호출되지 않도록 함.
- 이를 통해 클래스 외부에서는 오직 **생성자를 통한 초기화 및 리스너 등록만 가능**하며, 내부 구현 로직은 캡슐화(encapsulation)되어 유지보수성과 안정성이 높아짐.
- 내부 멤버인 `buttonPanel`, `courseSection`, `frame` 등도 모두 `private` 으로 선언되어 외부 접근 차단.

> **설계 목적:**
> 클래스 외부에서 이벤트 핸들러 로직에 직접 접근하거나 수정하지 못하도록 막아 **응집도는 높이고 결합도는 낮춘 구조**로 구성.

1. `addListeners()`

```
private void addListeners() {
    buttonPanel.showListButton.addActionListener(e -> showCourseList());
    buttonPanel.addButton.addActionListener(e -> addCourse());
    buttonPanel.deleteButton.addActionListener(e -> deleteCourse());
    buttonPanel.loadButton.addActionListener(e -> loadCourses());
    buttonPanel.saveButton.addActionListener(e -> saveCourses());
}
```

- **역할**: 각 버튼에 대한 이벤트 리스너를 등록한다.
- **설계 의도**: 리스너 등록을 메서드로 분리하여 **생성자 내부 코드의 가독성을 높이고**, 이벤트 바인딩의 책임을 명확히 분리하였다.

2. `showCourseList()`

```
private void showCourseList() {
    try {
        CourseSchedule schedule = new CourseSchedule("myschedule-norm.data");
        JFrame frame2 = new JFrame("Course List");
        frame2.setSize(500, 300);
        frame2.setLayout(new BorderLayout());

        InputPanel inputPanel = courseSection.getInputPanel();
        CourseListPanel listPanel = new CourseListPanel(
            CourseListPanel.getTableData(schedule)
        );

        new CourseTableController(listPanel, inputPanel);

        frame2.add(new JTextField("List of Courses - Click to Select"), BorderLayout.NORTH);
```

```
            frame2.add(listPanel, BorderLayout.CENTER);
            frame2.setVisible(true);
        } catch (Exception ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog(frame, "Error loading schedule: " + ex.getMessage());
        }
    }
```

- **역할**: 저장된 과목 정보를 테이블 형태로 새 창에 띄운다.
- **설계 의도**:
  - 별도의 창(`frame2`)을 생성하여 리스트를 분리시킴으로써 UI의 **모듈화**와 **재사용성** 확보
  - `CourseTableController`를 통해 리스트 클릭 이벤트도 컨트롤러에 위임하여 **책임 분리** 구현

3. `addCourse()`

```
private void addCourse() {
    try {
        InputPanel inputPanel = courseSection.getInputPanel();
        String name = inputPanel.subjectField.getText().trim();
        String creditText = inputPanel.creditField.getText().trim();
        String day = inputPanel.getSelectedDay();
        String startText = inputPanel.startField.getText().trim();
        String timeText = inputPanel.timeField.getText().trim();

        if (name.isEmpty() || creditText.isEmpty() || startText.isEmpty() || timeText.isEmpty()) {
            JOptionPane.showMessageDialog(frame, "모든 필드를 입력해주세요.");
            return;
        }

        float credit = Float.parseFloat(creditText);
        int start = Integer.parseInt(startText);
        int time = Integer.parseInt(timeText);

        courseSection.getSchedule().addCourse(name, credit, day, start, time);
        courseSection.updatePanels();

    } catch (NumberFormatException ex) {
        JOptionPane.showMessageDialog(frame, "학점, 시작 교시, 시간은 숫자여야 합니다.");
    } catch (IllegalArgumentException ex) {
        JOptionPane.showMessageDialog(frame, "과목 추가 실패");
    } catch (Exception ex) {
        ex.printStackTrace();
        JOptionPane.showMessageDialog(frame, "과목 추가 중 오류 발생: " + ex.getMessage());
    }
}
```

- **역할**: 사용자가 입력한 과목 정보를 검증하고, `CourseSchedule`에 추가한다.
- **설계 의도**:
  - 입력 검증과 예외 처리를 통해 **안정성 강화**
  - 직접적으로 모델(`CourseSchedule`)을 조작하여 **MVC의 Controller 역할**을 수행

4. `deleteCourse()`

```
private void deleteCourse() {
    try {
        CourseListPanel listPanel = courseSection.getListPanel();
        JTable table = listPanel.getTable();
        int selectedRow = table.getSelectedRow();
```

```
            if (selectedRow == -1) {
                JOptionPane.showMessageDialog(frame, "삭제할 과목을 먼저 선택하세요.");
                return;
            }

            String courseName = table.getValueAt(selectedRow, 0).toString();
            courseSection.getSchedule().deleteCourse(courseName);
            courseSection.updatePanels();

        } catch (Exception ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog(frame, "과목 삭제 중 오류 발생: " + ex.getMessage());
        }
    }
```

- **역할**: 테이블에서 선택된 과목을 `CourseSchedule`에서 제거한다.
- **설계 의도**:
  - 선택되지 않은 경우를 별도로 안내하여 **사용자 경험(UX) 개선**
  - `CourseSchedule`의 delete 메서드를 통해 모델 상태를 변경함

---

5. `loadCourses()`

```
private void loadCourses() {
    try {
        JFileChooser fileChooser = new JFileChooser();
        int result = fileChooser.showOpenDialog(frame);

        if (result == JFileChooser.APPROVE_OPTION) {
            File selectedFile = fileChooser.getSelectedFile();
            CourseSchedule newSchedule = new CourseSchedule(selectedFile.getAbsolutePath());
            courseSection.setSchedule(newSchedule);
        }
    } catch (Exception ex) {
        ex.printStackTrace();
        JOptionPane.showMessageDialog(frame, "파일 불러오기 중 오류 발생: " + ex.getMessage());
    }
}
```

- **역할**: 외부 파일에서 과목 정보를 불러와 현재 스케줄에 반영한다.
- **설계 의도**:
  - `JFileChooser`를 통해 **사용자 편의성 확보**
  - 새로운 객체 생성 후 `setSchedule()`을 통해 뷰 갱신을 유도함

6. `saveCourses()`

```
private void saveCourses() {
    try {
        JFileChooser fileChooser = new JFileChooser();
        int result = fileChooser.showSaveDialog(frame);

        if (result == JFileChooser.APPROVE_OPTION) {
            File selectedFile = fileChooser.getSelectedFile();
            courseSection.getSchedule().saveToFile(selectedFile);
            JOptionPane.showMessageDialog(frame, "성공적으로 저장되었습니다.");
        }
    } catch (Exception ex) {
```

```
            ex.printStackTrace();
            JOptionPane.showMessageDialog(frame, "파일 저장 중 오류 발생: " + ex.getMessage());
        }
    }
```

- **역할**: 현재 등록된 과목 정보를 파일로 저장한다.
- **설계 의도**:
  - 저장 경로를 사용자가 직접 지정하게 하여 유연성 제공
  - 예외 상황에 대비한 안정적인 파일 저장 처리 구현

CourseTableController

`CourseTableController`는 `CourseListPanel`에서 과목을 클릭할 때 발생하는 **마우스 이벤트를 처리**하는 컨트롤러로 설계하였다. 사용자가 테이블에서 과목을 선택하면, 해당 정보를 `InputPanel`에 자동으로 채워주는 역할을 수행한다.

Member Visibility 설계

- 클래스 내부 필드인 `listPanel`, `inputPanel`은 `private`**으로 선언**되어 외부에서 직접 접근할 수 없다.
- 이벤트 리스너 등록 메서드인 `attachMouseListener()` 역시 `private`으로 정의되어, **외부에서 호출 불가능하고 생성자 내부에서만 실행된다.**
- 이를 통해 **컨트롤러의 내부 동작을 외부로부터 은닉하고**, 인터페이스를 단순하게 유지함으로써 코드 안정성과 응집도를 높이는 객체지향 설계를 실현하였다.

> **설계 목적:**
> 외부 클래스는 오직 생성자를 통해 이 컨트롤러를 사용할 수 있으며, 내부 이벤트 처리 로직이나 필드에는 접근할 수 없다.
> 이는 **정보 은닉(Information Hiding)**과 **책임 분리(Single Responsibility Principle)**의 구현이다.

---

1. **생성자 및 초기 설정**

```
public CourseTableController(CourseListPanel listPanel, InputPanel inputPanel) {
    this.listPanel = listPanel;
    this.inputPanel = inputPanel;

    attachMouseListener();
}
```

- **역할**:
  - `CourseListPanel`과 `InputPanel`을 받아 내부에 저장
  - 생성과 동시에 마우스 이벤트 리스너를 등록하는 `attachMouseListener()` 호출
- **설계 의도**:
  - 컨트롤러가 뷰 컴포넌트와 분리되어 독립적으로 동작하도록 구성
  - 생성자 내에서 리스너를 설정하여 **초기화 시점에 필요한 설정을 자동으로 적용**

---

2. `attachMouseListener()`

```
private void attachMouseListener() {
    JTable table = listPanel.getTable();
    table.addMouseListener(new MouseAdapter() {
        public void mouseClicked(MouseEvent e) {
            int row = table.getSelectedRow();
            if (row != -1) {
                String subject = table.getValueAt(row, 0).toString();
```

```
                String credit = table.getValueAt(row, 1).toString();
                String day = table.getValueAt(row, 2).toString();
                String start = table.getValueAt(row, 3).toString();
                String time = table.getValueAt(row, 4).toString();

                inputPanel.setInputFields(subject, credit, day, start, time);
            }
        }
    });
}
```

- **역할**:
  - `CourseListPanel` 의 테이블에서 마우스 클릭 이벤트를 감지하여
    클릭된 행(row)의 데이터를 추출한 뒤 `InputPanel` 의 입력 필드에 자동 입력
- **설계 의도**:
  - 사용자의 직접 입력을 최소화하고, **화면 간 정보 전달의 흐름을 명확히 구현**하여 사용성 향상을 도모
  - 초기 설계에서는 과목 정보를 수동으로 입력하고 추가하는 구조였으나, 과제의 핵심 목표인 **프레임 간의 정보 연동 및 자동화**를 실현하기 위해 리스트 클릭 기반의 데이터 전달 방식으로 개선하고자 함.
  - 과목 정보 창은 향후 과목 수정 및 신규 입력 기능 확장 가능성을 고려한 핵심 인터페이스로, ADD 버튼을 통한 수강 신청은 반드시 해당 입력 창의 정보를 기반으로 동작하도록 구성

Entity

CourseSchedule

`CourseSchedule` 클래스는 수강 신청 프로그램에서 시간표 데이터를 관리하는 핵심 **Model(Entity)** 클래스이다. 이 클래스는 과목 정보를 추가, 삭제, 파일로부터 불러오고 저장하는 기능을 제공하며, 시간표 출력 기능도 포함되어 있다.

이번 리팩토링에서는 내부 자료구조를 `ArrayList` 로 통일하고, 과목 데이터를 외부 파일에 저장할 수 있는 `saveToFile()` 메서드를 새롭게 도입하여 **데이터 관리의 유연성과 확장성**을 높였다.

Member Visibility 설계

- 내부 필드 `courses` 는 `private` **으로 선언**되어 외부에서 직접 접근할 수 없고, 반드시 메서드를 통해 접근해야 함.
- 비즈니스 로직 메서드(`addCourse()` , `deleteCourse()` , `saveToFile()` 등)는 `public` 으로 선언되어 외부 컨트롤러나 뷰에서 사용할 수 있도록 공개됨.
- 내부 헬퍼 메서드인 `isConflict()` 는 `private` 으로 선언되어 외부 호출이 불가능하며, 내부 로직의 일부로만 사용됨.

> **설계 목적:**
>
> - 모델 클래스의 내부 상태(`courses`)는 외부에서 임의로 변경될 수 없도록 보호하고,
> - 외부에는 **의도된 방식으로만 접근할 수 있도록 인터페이스를 제한**하여 **데이터 무결성 보장** 및 **모듈화된 구조** 유지.
> - `private` 메서드는 캡슐화(Encapsulation)를 통해 내부 구현 세부사항을 숨기고, 필요한 로직만 외부에 노출.

---

1. **내부 자료구조 리팩토링 (`ArrayList` 사용)**

```
ArrayList<Course> courses = new ArrayList<>();
```

- **설계 이유**:
  - 과목은 동적으로 추가 및 삭제가 자주 발생하므로, 배열보다 **동적 크기 조절이 가능한 ArrayList**가 적합함
  - `ArrayList` 는 순서 보장이 가능하고 반복 접근이 용이하여, 출력과 파일 저장 기능 구현에도 유리함
- **기대 효과**:
  - 코드의 간결성 향상
  - 과목 추가/삭제 기능의 성능 및 확장성 개선

2. 과목 추가 기능 (`addCourse()`)

```
public void addCourse(String name, float credit, String day, int start, int time)
```

- **기능**:
  - 사용자 입력을 바탕으로 새로운 `Course` 객체를 생성한 뒤,
    - ① 유효성 검사(`course.isValid()`),
    - ② 시간 충돌 검사(`isConflict(course)`)
      를 거쳐 `courses` 리스트에 추가한다.
- **예외 처리 (리팩토링 반영)**:
  - 기존에는 실패 시 단순히 반환하거나 메시지만 출력하는 방식이었다면,
    리팩토링 이후에는 **예외를 명확하게 발생시켜 호출자에게 실패 원인을 전달**하는 구조로 변경하였다.
  - 아래와 같은 경우 각각 `IllegalArgumentException` 예외를 명시적으로 발생시킨다:
    1. 입력된 과목 정보가 유효하지 않을 경우:

```
if (!course.isValid()) {
    throw new IllegalArgumentException("Invalid course data: " + course);
}
```

    2. 기존 과목들과 시간 충돌이 발생하는 경우:

```
if (isConflict(course)) {
    throw new IllegalArgumentException("Conflict detected for course: " + course);
}
```

- **설계 의도**:
  - 예외 발생을 통해 문제 상황을 **명확하게 호출자(GUI 등)에 전달**하고,
    GUI 레벨에서는 이를 감지하여 사용자에게 적절한 메시지 박스로 안내할 수 있게 한다.
  - 예외 처리를 함수 내부에서 숨기기보다는 호출 측에 위임함으로써, **로직과 UI의 분리를 강화**하고,
    재사용성과 테스트 가능성을 높이는 방향으로 설계하였다.
  - 이를 통해 Controller나 View는 try-catch 문을 통해 적절히 예외를 핸들링하고, **에러 메시지 UI 표시**, **입력 필드 클리어**, **로그 출력 등 다양한 후속 조치를 자유롭게 설계**할 수 있다.

---

3. **과목 삭제 기능 (`deleteCourse()`)**

```
public void deleteCourse(String name)
```

- **기능**:
  - 지정한 이름의 과목을 리스트에서 제거
  - `removeIf()` 를 사용하여 **가독성과 효율성**을 동시에 만족하는 구현

---

4. **시간표 출력 (`print()`)**

```
public void print()
```

- **기능**:
  - 내부적으로 `getTable()` 메서드를 통해 2차원 배열을 구성한 후 시간표 형식으로 출력

- 수업 요일과 시작 교시 정보를 활용하여 각 위치에 과목명을 채워 넣는 방식
- **설계 의도**:
  - **시각적으로 확인 가능한 텍스트 기반 시간표** 제공
  - 출력 방식은 `콘솔 테스트` 나 `디버깅` 시 유용함

---

5. **파일 저장 기능 (`saveToFile()`)**

```
public void saveToFile(File file) throws IOException
```

- **기능**:
  - 현재 수강 목록을 지정된 파일에 저장
  - 각 과목은 `과목명:학점:요일:시작교시:시간` 포맷으로 저장되며, 각 줄마다 구분 주석(`//`)을 추가
- **설계 의도**:
  - `CourseSchedule` 객체의 **직렬화 역할**을 수행
  - 사용자가 GUI를 통해 저장 버튼을 누를 경우 시간표 상태를 파일로 보존 가능
  - 이후 `CourseSchedule(String filename)` 생성자를 통해 파일로부터 다시 불러오는 순환 구조 완성
- **포맷 예시**:

```
DataStructures:3.0:Mon:1:2
//
Algorithms:3.0:Wed:3:2
//
```

---

6. 충돌 검사 (`isConflict()`)

```
private boolean isConflict(Course course)
```

- **기능**:
  - 새로 추가하려는 과목이 기존 과목들과 **동일한 요일**에 있고,
    시간대가 겹치는 경우가 있는지 검사한다.
  - 검사 조건:

```
if (c.getStart() < course.getStart() + course.getTime() &&
    c.getStart() + c.getTime() > course.getStart())
```

    즉, 시간 구간이 **부분적으로라도 겹치면 true를 반환**한다.

- **설계 의도**:
  - 중복되거나 겹치는 수업의 등록을 사전에 방지하여 **논리적으로 유효한 시간표 유지**
  - **논리 분리를 위한 헬퍼 메서드**로서 `addCourse()` 내부의 코드 가독성을 높이고, 중복 검사 기능을 모듈화함
- **예외 처리 방식이 `boolean` 반환인 이유**:
  - `isConflict()` 는 내부 로직의 한 조건으로 사용되며,
    그 자체가 **에러 상황을 발생시키는 행위가 아니라 충돌 여부를 판단하는 판단 도우미 역할**을 수행한다.
  - 실제로 예외를 발생시킬지 말지는 `addCourse()` 등 **상위 호출자에서 결정**하며,
    호출자는 다음과 같이 처리한다:

```
if (isConflict(course)) {
    throw new IllegalArgumentException("Conflict detected for course: " + course);
}
```

- 이처럼 `isConflict()`를 boolean으로 설계함으로써,
  - 다양한 상황(예: 단순 미리보기 vs 실제 등록 시)에 **재사용 가능**
  - 충돌을 반드시 예외로 처리하지 않고도 **조건 분기 처리 가능**
    → 결과적으로 **유연하고 모듈화된 설계**가 가능해질 것으로 판단하여 위와 같은 방식의 예외처리를 사용하였다.

Course

본 구현에서는 `Course` 클래스에 대한 구조 변경 없이, **이전 과제에서 구현한 내용을 그대로 활용**하였다. 해당 클래스는 하나의 과목 정보를 표현하는 **기본 단위 모델(Entity)** 로, 과목명, 학점, 요일, 시작 시간, 강의 시간 등의 정보를 필드로 포함하고 있다.

설계 목적

- 시간표를 구성하는 **기본 단위인 과목 객체**를 표현하기 위한 모델 클래스
- 유효성 검사 기능(`isValid()` , `isDayError()` , `isStartError()`)을 통해 **잘못된 입력을 사전에 걸러내는 역할** 수행
- 문자열 배열을 파싱하여 객체로 변환하거나, 외부에서 전달된 파라미터로 객체를 구성하는 **생성자 오버로딩** 제공
- 출력, 수정(setter), 조회(getter) 기능을 제공하여 **외부에서의 조작과 연동이 용이하도록 설계**

Member Visibility 설계

- 모든 필드(`name` , `credit` , `day` , `start` , `time` , `valid`)는 `private` 으로 선언되어 외부에서 직접 접근할 수 없고, 반드시 getter/setter를 통해 접근해야 함
- 입력 파싱 및 유효성 검사 관련 메서드(`getName()` , `isDayError()` 등)는 모두 `private` 으로 설정되어 **내부 로직 캡슐화**
- 외부 접근이 필요한 메서드 (`getName()` , `getCredit()` 등)는 `public` 으로 선언하여 컨트롤러 및 뷰에서 자유롭게 사용 가능
- 생성자 중 하나는 `String[]` 을 인자로 받아 외부 파일로부터 데이터를 불러올 때 활용되며, **입력 검증 로직을 포함**

> 이러한 visibility 구성은 객체지향 설계 원칙 중 **캡슐화(Encapsulation)**와 **정보 은닉(Information Hiding)**을 충실히 따르며, 데이터의 일관성과 안정성을 보장하는 구조이다.

Member Visibility 설계 (공통)

- 모든 UI 구성요소 클래스들은 **필드 대부분을** `private` **으로 선언**하여 외부에서 직접 UI 구성 요소에 접근하지 못하도록 함.
- 외부에서 필요한 필드(버튼, 텍스트 필드 등)에 대해서는 **선택적으로** `public` **또는** `getter` **메서드**를 통해 제한적으로 접근을 허용.
- 내부 레이아웃 구성, 이벤트 핸들링 관련 메서드는 `private` **으로 은닉**하여 외부에서 호출할 수 없고, 객체 내부에서만 사용되도록 설계.
- 일부 유틸리티 성격의 메서드는 `static` + `public` 으로 제공하여 전역적으로 동일한 결과를 생성하고 인스턴스 생성 없이 사용 가능.

> **설계 목적:**
>
> - UI 요소에 대한 **직접적인 접근을 방지하고**, 외부에서는 명시된 인터페이스(getter 등)를 통해서만 필요한 정보에 접근 가능하게 하여 **UI 안정성과 일관성** 확보
> - 공통 데이터 변환 메서드 등은 static으로 제공하여 **재사용성과 호출 편의성 향상**

View

`org.example.view` 패키지는 사용자가 직접 상호작용하는 화면(UI)을 구성하는 요소들로 이루어져 있으며, 과목 입력, 버튼 조작, 리스트 출력, 시간표 표시 등의 기능을 시각적으로 제공한다. 이는 과제 5에 사용한 내용들을 View package로 옮겨서 사용하였고, 아래와 같은 역할을 하는 클래스가 존재한다.

1. `InputPanel`

- **역할**: 과목 정보 입력창
- **구성**:
  - JTextField: 과목명, 학점, 시작 교시, 수업 시간
  - JComboBox: 요일 선택
- **기능**:
  - 과목 정보를 수동 입력하거나, 리스트에서 과목을 선택했을 때 자동으로 채워짐
  - 필요 시 `setReadOnly(true)` 로 편집을 비활성화할 수 있음

---

2. `ButtonPanel`

- **역할**: 사용자 액션(등록, 삭제, 불러오기, 저장 등)을 위한 버튼 집합
- **구성**:
  - JButton 5개: Show List, Add, Delete, Load, Save
- **기능**:
  - 각 버튼은 Controller와 연결되어 이벤트를 트리거함
  - 수직 정렬(GridLayout)을 통해 UI 구성

---

3. `CourseListPanel`

- **역할**: 현재 등록된 과목 리스트를 테이블로 표시
- **구성**:
  - JTable + JScrollPane
- **기능**:
  - 리스트 데이터를 정적 메서드 `getTableData()` 를 통해 제공받아 렌더링
  - 클릭 시 해당 과목 정보가 `InputPanel` 에 자동 전송됨 (Controller 연결)

해당 클래스에서는 Static 메서드가 사용되었다. 사용 이유는 다음과 같다
`CourseListPanel.getTableData(CourseSchedule courseSchedule)`

- **역할**: `CourseSchedule` 객체로부터 `Object[][]` 형태의 테이블 데이터를 추출하여 `JTable` 에 제공할 수 있도록 변환
- **정당성**:
  - **객체 상태에 의존하지 않으며, 외부에서 호출 시 전역적으로 동일한 동작을 보장**함
  - 뷰의 재사용성과 유틸리티 기능을 분리하기 위해 정적(static) 메서드로 제공
  - 예: `updatePanels()` 등에서 별도의 인스턴스 생성 없이 호출 가능

```java
Object[][] data = CourseListPanel.getTableData(schedule);
```

`ScheduleTablePanel.toScheduleTable(Object[][] courseData)`

- **역할**: 일반적인 테이블 데이터(`Object[][]`)를 기반으로 시간표 형태의 배열(9x5)을 생성
- **정당성**:
  - 인스턴스 변수나 내부 상태에 의존하지 않고, **입력값만으로 결과를 생성하는 순수 함수**
  - 시간표 형식 변환이 특정 인스턴스의 책임이 아닌, 공통 유틸리티로 사용되므로 static이 적절

```java
Object[][] schedule = toScheduleTable(courseData);
```

- 설계 의도:
  - View 컴포넌트들은 모두 시각 표현에 집중하며, 사용자 입력, 버튼 이벤트, 테이블 선택 등 **UI 기능의 시각적 표현을 분리된 모듈로 구성**
  - `static` 메서드는 내부 상태에 의존하지 않는 유틸리티 성격의 기능을 제공하여 **불필요한 인스턴스 생성을 방지**하고, 호출 편의성을 높였다.
  - 전체적으로 `View` 는 **역할별 UI 컴포넌트의 분리**, **이벤트 리스너와의 명확한 연동 지점 확보**, **재사용 가능한 로직 구현**에 초점을 두었다.

---

4. `ScheduleTablePanel`

- **역할**: 시간표 형태로 과목 정보를 시각적으로 표현
- **구성**:
  - 2차원 JTable (요일 열 × 교시 행) + rowHeader (1~9교시)
- **기능**:
  - 리스트 데이터를 기반으로 시간표 구조(`toScheduleTable()`)를 생성
  - 고정된 시간/요일 기반의 표 형태로 과목 배치 시각화

---

5. `CourseScheduleSection`

- **역할**: 위의 뷰 요소들을 통합해 상단(입력+리스트+버튼)과 하단(시간표)으로 전체 UI 구성
- **구성**:
  - 상단: `InputPanel` , `CourseListPanel` , `ButtonPanel` , "My Courses" 텍스트
  - 하단: `ScheduleTablePanel`
- **기능**:
  - `updatePanels()` 메서드를 통해 `CourseSchedule` 상태 변화에 따라 각 UI 요소를 동적으로 갱신
  - Controller(`CourseTableController`)와 연결되어 리스트 선택 → 입력창 반영 구조 형성

Main

`CourseScheduleMain` 클래스는 본 수강 신청 프로그램의 **메인 실행 진입점**이며, 전체 GUI 화면을 구성하고 필요한 객체들을 초기화하는 역할을 수행한다.

주요 구성 및 역할

```
public class CourseScheduleMain {
    public static void main(String[] args) throws Exception {
        ...
    }
}
```

- **역할**:
  - 사용자 환경 정보 출력 (시스템 시간, 사용자명, 호스트명)
  - 메인 프레임(`JFrame`) 생성 및 UI 레이아웃 구성
  - 모델(`CourseSchedule`)과 뷰(`CourseScheduleSection` , `ButtonPanel`) 초기화
  - 컨트롤러(`CourseButtonController`) 연결을 통해 이벤트 바인딩 설정
  - GUI를 실행 가능하게 화면을 표시 (`setVisible(true)`)

설계 의도 및 특징

- **프로젝트 구조의 초기 배치**를 설정하는 진입점으로, 모든 객체가 명확히 분리되어 인스턴스화됨

- View와 Controller는 각각 역할에 맞는 객체를 생성하여 전달받고,
  모든 의존성이 main에서 명확하게 설정되므로 프로그램 흐름이 직관적
- GUI 기반 프로그램의 실행 구조를 따라 설계되어, **초기 실행 흐름과 UI 설정의 가독성**을 확보

설계 및 구현에 대한 자체 평가

강점 및 성과

1. **MVC 패턴 기반의 구조적 설계**
   - Model, View, Controller의 책임을 명확히 분리하여 **유지보수성과 확장성**이 뛰어남
   - 사용자 인터페이스와 데이터 처리 로직 간의 의존도를 줄임
2. **기본적인 수강 신청 기능 구현 완료**
   - 과목 추가, 삭제, 리스트 확인, 시간표 출력 등 필수 기능 정상 동작
   - 파일 저장/불러오기 기능을 통해 **영속성**을 지원
3. **입력 검증 및 예외 처리 강화**
   - 과목 정보의 유효성 검사, 시간 충돌 방지, 포맷 검증 등을 통해 프로그램 안정성 확보
   - 예외를 던지고 호출자에서 처리하도록 하여 **로직과 UI 분리**

현재 구현의 한계점

1. **시간표 중복 과목 시 사용자 피드백 부족**

- **문제점**:
  - `CourseSchedule.addCourse()` 내에서 충돌(`isConflict() == true`)이 감지되면 예외가 발생하거나 단순히 추가가 건너뛰어짐
  - 하지만 사용자에게 **어떤 과목이 어떤 이유로 추가되지 않았는지**에 대한 시각적/구체적 피드백이 제공되지 않음
- **영향**:
  - 사용자는 "과목 추가가 실패했다"는 결과만 알 수 있고, 충돌한 기존 과목 정보나 겹치는 시간대를 확인할 수 없음
  - 사용자 경험(UX)이 저하되고, 디버깅이나 사용자 조작 오류 수정이 어려워짐
- **개선 방안**:
  - `addCourse()` 의 예외 메시지를 보다 구체화하여 충돌 요일, 시간대, 기존 과목명을 포함
    예: `Conflict detected: [DataStructs] on Mon 2-3교시와 겹칩니다.`
  - `InputPanel` 또는 별도 팝업으로 충돌 내용을 **시각적으로 강조하여 사용자에게 직관적으로 안내**

---

2. **Static 메서드 구조의 개선 필요성**

- **현재 구조**:
  - `CourseListPanel.getTableData()` , `ScheduleTablePanel.toScheduleTable()` 등 일부 기능은 static 메서드로 구현됨
  - 내부 상태에 의존하지 않고, 외부에서 호출 가능한 유틸리티 함수로 작성되어 **코드 재사용성**은 확보됨
- **한계점**:
  - static 메서드는 객체 지향 설계의 **책임 주체(Owner)**가 명확하지 않음
    → 예: `ScheduleTablePanel` 클래스에 속한 메서드지만, 실제로는 데이터 가공만 수행하고 객체 자체와 무관함
  - 테스트나 기능 확장 시, static 메서드는 **오버라이딩 불가**, **DI(Dependency Injection) 불가** 등 유연성이 떨어짐
- **개선 방안**:
  - `TableDataConverter` 또는 `ScheduleFormatter` 같은 **별도 유틸리티 클래스**로 분리
    예:

```
public class TableDataConverter {
    public Object[][] convertToListData(CourseSchedule schedule) {...}
    public Object[][] convertToScheduleTable(Object[][] rawData) {...}
}
```

- 이로써 책임 분리 및 클래스 간 응집도를 높이고, **기능 단위 테스트**도 용이해짐

소스 프로그램

CourseButtonController.java

```
package org.example.controller;

import org.example.view.*;

import javax.swing.*;
import java.awt.*;
import java.io.File;

import org.example.Entity.CourseSchedule;

public class CourseButtonController {
    private final CourseScheduleSection courseSection;
    private final ButtonPanel buttonPanel;
    private final JFrame frame;

    public CourseButtonController(JFrame frame, CourseScheduleSection courseSection, ButtonPanel
buttonPanel) {
        this.frame = frame;
        this.courseSection = courseSection;
        this.buttonPanel = buttonPanel;

        addListeners();
    }

    private void addListeners() {
        buttonPanel.showListButton.addActionListener(e -> showCourseList());
        buttonPanel.addButton.addActionListener(e -> addCourse());
        buttonPanel.deleteButton.addActionListener(e -> deleteCourse());
        buttonPanel.loadButton.addActionListener(e -> loadCourses());
        buttonPanel.saveButton.addActionListener(e -> saveCourses());
    }

    private void showCourseList() {
        try {
            CourseSchedule schedule = new CourseSchedule("myschedule-norm.data");
            JFrame frame2 = new JFrame("Course List");
            frame2.setSize(500, 300);
            frame2.setLayout(new BorderLayout());

            InputPanel inputPanel = courseSection.getInputPanel();
            CourseListPanel listPanel = new CourseListPanel(
                    CourseListPanel.getTableData(schedule)
            );

            new CourseTableController(listPanel, inputPanel);

            frame2.add(new JTextField("List of Courses - Click to Select"), BorderLayout.NORTH);
            frame2.add(listPanel, BorderLayout.CENTER);
            frame2.setVisible(true);
```

```
        } catch (Exception ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog(frame, "Error loading schedule: " + ex.getMessage());
        }
    }

    private void addCourse() {
        try {
            InputPanel inputPanel = courseSection.getInputPanel();
            String name = inputPanel.subjectField.getText().trim();
            String creditText = inputPanel.creditField.getText().trim();
            String day = inputPanel.getSelectedDay();
            String startText = inputPanel.startField.getText().trim();
            String timeText = inputPanel.timeField.getText().trim();

            if (name.isEmpty() || creditText.isEmpty() || startText.isEmpty() ||
timeText.isEmpty()) {
                JOptionPane.showMessageDialog(frame, "모든 필드를 입력해주세요.");
                return;
            }

            float credit = Float.parseFloat(creditText);
            int start = Integer.parseInt(startText);
            int time = Integer.parseInt(timeText);

            courseSection.getSchedule().addCourse(name, credit, day, start, time);
            courseSection.updatePanels();

        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frame, "학점, 시작 교시, 시간은 숫자여야 합니다.");
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frame, "과목 추가 실패");
        } catch (Exception ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog(frame, "과목 추가 중 오류 발생: " + ex.getMessage());
        }
    }

    private void deleteCourse() {
        try {
            CourseListPanel listPanel = courseSection.getListPanel();
            JTable table = listPanel.getTable();
            int selectedRow = table.getSelectedRow();

            if (selectedRow == -1) {
                JOptionPane.showMessageDialog(frame, "삭제할 과목을 먼저 선택하세요.");
                return;
            }

            String courseName = table.getValueAt(selectedRow, 0).toString();
            courseSection.getSchedule().deleteCourse(courseName);
            courseSection.updatePanels();

        } catch (Exception ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog(frame, "과목 삭제 중 오류 발생: " + ex.getMessage());
        }
    }

    private void loadCourses() {
        try {
            JFileChooser fileChooser = new JFileChooser();
```

```
            int result = fileChooser.showOpenDialog(frame);

            if (result == JFileChooser.APPROVE_OPTION) {
                File selectedFile = fileChooser.getSelectedFile();
                CourseSchedule newSchedule = new CourseSchedule(selectedFile.getAbsolutePath());
                courseSection.setSchedule(newSchedule);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog(frame, "파일 불러오기 중 오류 발생: " + ex.getMessage());
        }
    }

    private void saveCourses() {
        try {
            JFileChooser fileChooser = new JFileChooser();
            int result = fileChooser.showSaveDialog(frame);

            if (result == JFileChooser.APPROVE_OPTION) {
                File selectedFile = fileChooser.getSelectedFile();
                courseSection.getSchedule().saveToFile(selectedFile);
                JOptionPane.showMessageDialog(frame, "성공적으로 저장되었습니다.");
            }
        } catch (Exception ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog(frame, "파일 저장 중 오류 발생: " + ex.getMessage());
        }
    }
}
```

CourseTableController.java

```
package org.example.controller;

import org.example.view.CourseListPanel;
import org.example.view.InputPanel;

import javax.swing.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

public class CourseTableController {

    private final CourseListPanel listPanel;
    private final InputPanel inputPanel;

    public CourseTableController(CourseListPanel listPanel, InputPanel inputPanel) {
        this.listPanel = listPanel;
        this.inputPanel = inputPanel;

        attachMouseListener();
    }

    private void attachMouseListener() {
        JTable table = listPanel.getTable();
        table.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                int row = table.getSelectedRow();
                if (row != -1) {
                    String subject = table.getValueAt(row, 0).toString();
                    String credit = table.getValueAt(row, 1).toString();
```

```
                    String day = table.getValueAt(row, 2).toString();
                    String start = table.getValueAt(row, 3).toString();
                    String time = table.getValueAt(row, 4).toString();

                    inputPanel.setInputFields(subject, credit, day, start, time);
                }
            }
        });
    }
}
```

Course.java

```
package org.example.Entity;

public class Course {

    private String name; // 과목명
    private float credit; // 학점
    private String day ; // 요일
    private int start; // 시작시간
    private int time ; // 강의시간
    private boolean valid = true; // 유효한 과목인지 저장


    Course(String[] tokens) {
        this.name = getName(tokens);
        this.credit = getCredit(tokens);

        this.day = getDay(tokens);
        if (isDayError(day)) {
            valid = false;
            return;
        }

        this.start = getStart(tokens);
        if (isStartError(start)) {
            valid = false;
            return;
        }

        this.time = getTime(tokens);
    }

    Course(String name, float credit, String day, int start, int time) {
        this.name = name;
        this.credit = credit;
        this.day = day;
        this.start = start;
        this.time = time;
    }

    public boolean isValid() {
        return valid;
    }

    private boolean isStartError(int start) {
        if (start < 1 || start > 9) {
            System.out.println("Unresolved hour -- " + start);
            return true;
        }
```

```
        return false;
    }

    private boolean isDayError(String day) {
        switch (day) {
            case "Mon":
            case "Tue":
            case "Wed":
            case "Thu":
            case "Fri":
                return false;
            default:
                System.out.println("Unknown date -- " + day);
                return true;
        }
    }

    private String getName(String[] tokens){
        return tokens[0].trim();
    }

    private Float getCredit(String[] tokens){
        return Float.parseFloat(tokens[1].trim());
    }

    private String getDay(String[] tokens){
        return tokens[2].trim();
    }

    private Integer getTime(String[] tokens){
        return Integer.parseInt(tokens[4].trim());
    }

    private Integer getStart(String[] tokens){
        return Integer.parseInt(tokens[3].trim());
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setCredit(float credit) {
        this.credit = credit;
    }

    public void setDay(String day) {
        this.day = day;
    }

    public void setStart(int start) {
        this.start = start;
    }

    public void setTime(int time) {
        this.time = time;
    }

    public String getName() {
        return name;
    }

    public float getCredit() {
```

```
        return credit;
    }

    public String getDay() {
        return day;
    }

    public int getStart() {
        return start;
    }

    public int getTime() {
        return time;
    }

    public void print(){
        System.out.println("Course Name: " + name);
        System.out.println("Credit: " + credit);
        System.out.println("Day: " + day);
        System.out.println("Start: " + start);
        System.out.println("Time: " + time);
    }
}
```

CourseSchedule.java

```
package org.example.Entity;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Scanner;

public class CourseSchedule {

    ArrayList<Course> courses = new ArrayList<>(); // 과목 배열

    public CourseSchedule(String filename) throws Exception {
        File file = new File(filename);
        try {
            Scanner input = new Scanner(file);  // file is an instance of File
            while(input.hasNext()){
                String line = input.nextLine();
                if(line.startsWith("//") || line.trim().isEmpty()) continue; // 주석 무시
                String[] tokens = modifyFormat(line); // 형식 수정
                Course course = new Course(tokens);
                if (!course.isValid()) continue; // 유효하지 않으면 건너뜀
                if (isConflict(course)) { // 시간 충돌 체크
                    continue; // 충돌이 있으면 다음 줄로 넘어감
                }
                courses.add(course); // 과목 추가
            }
        }
        catch(Exception e) {
                System.out.println("Unknwon File " + filename);
        }
    }

    public void addCourse(String name, float credit, String day, int start, int time) {
```

```
        Course course = new Course(name, credit, day, start, time);
        if (!course.isValid()) {
            throw new IllegalArgumentException("Invalid course data: " + course);
        }
        if (isConflict(course)) {
            throw new IllegalArgumentException("Conflict detected for course: " + course);
        }
        courses.add(course); // 과목 추가
    }

    public void deleteCourse(String name) {
        courses.removeIf(course -> course.getName().equals(name)); // 이름으로 과목 삭제
    }

    private String[] modifyFormat(String string){
        String removeSpace = string.replaceAll(" ", ""); // 공백 제거
        String[] line = removeSpace.split(":");
        if (line.length != 5) {
            System.out.println("Irregular schedule line -- " + string);
            String newLine = removeSpace.replaceAll("::", ":");
            return newLine.split(":");
        }
        return line;
    }

    private boolean isConflict(Course course) {
        for (Course c : courses) {
            if (c.getDay().equals(course.getDay())) { // 요일이 같으면
                if (c.getStart() < course.getStart() + course.getTime() && c.getStart() +
c.getTime() > course.getStart()) {
                    return true; // 충돌이 있으면 true                }
            }
        }
        return false; // 충돌이 없으면 false    }

    public Course getCourse(int i){
        return courses.get(i);
    }

    public ArrayList<Course> getCourses() {
        return courses;
    }

    public void print() {
        printDays(); // 요일 출력
        printTable(); // 시간표 출력
    }

    private String[][] getTable(){
        String[][] table = new String[9][5]; // 9교시 x 5일

        // 과목들을 시간표에 채워 넣기
        for (Course c : courses) {
            int dayIndex = getDayIndex(c.getDay()); // 요일 인덱스 (Mon=0, Tue=1,...)
int start = c.getStart() - 1; // 시간대 인덱스 (1교시는 0번 인덱스)

            for (int t = 0; t < c.getTime(); t++) {
                table[start + t][dayIndex] = c.getName(); // 교시 위치에 과목명 입력
            }
        }
        return table;
    }
```

```
    private void printDays(){
        String[] days = {"Mon", "Tue", "Wed", "Thu", "Fri"};
        // 헤더 출력
        System.out.printf("%-5s", "");
        for (String d : days) {
            System.out.printf("%-7s", d);
        }
        System.out.println();
        // 구분선 출력
        System.out.println("-----------------------------------");
    }

    private void printTable(){

        String[][] table = getTable(); // 시간표 배열
        // 시간표 출력
        for (int i = 0; i < 9; i++) {
            System.out.printf("%-5d", i + 1);
            for (int j = 0; j < 5; j++) {
                String subject = table[i][j];
                if (subject != null) {
                    System.out.printf("%-7s", subject);
                } else {
                    System.out.printf("%-7s", "");
                }
            }
            System.out.println();
        }
    }

    private int getDayIndex(String day) {
        switch (day) {
            case "Mon": return 0;
            case "Tue": return 1;
            case "Wed": return 2;
            case "Thu": return 3;
            case "Fri": return 4;
            default: return -1;
        }
    }

    public void saveToFile(File file) throws IOException {
        try (PrintWriter writer = new PrintWriter(new FileWriter(file))) {
            for (Course c : courses) {
                writer.printf("%s:%.1f:%s:%d:%d%n", c.getName(), c.getCredit(), c.getDay(),
c.getStart(), c.getTime());
                writer.println("//");
            }
        }
    }
}
```

ButtonPanel.java

```
package org.example.view;

import javax.swing.*;
import java.awt.*;

public class ButtonPanel extends JPanel {
```

```
    public JButton showListButton = new JButton("Show List");
    public JButton addButton = new JButton("Add");
    public JButton deleteButton = new JButton("Delete");
    public JButton loadButton = new JButton("Load");
    public JButton saveButton = new JButton("Save");

    public ButtonPanel() {
        setLayout(new GridLayout(5, 1, 5, 5));
        add(showListButton);
        add(addButton);
        add(deleteButton);
        add(loadButton);
        add(saveButton);
    }
}
```

CourseListPanel.java

```
package org.example.view;

import java.awt.BorderLayout;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import org.example.Entity.Course;
import org.example.Entity.CourseSchedule;

public class CourseListPanel extends JPanel {

    private JTable table;
    private final String[] columns = {"과목명", "학점", "요일", "시작교시", "시간"};

    public CourseListPanel(Object[][] data) {
        setLayout(new BorderLayout());

        table = new JTable(data, columns);
        JScrollPane scrollPane = new JScrollPane(table);
        add(scrollPane, BorderLayout.CENTER);
    }

    public static Object[][] getTableData(CourseSchedule courseSchedule) {
        Object[][] data = new Object[courseSchedule.getCourses().size()][5];
        for (int i = 0; i < courseSchedule.getCourses().size(); i++) {
            Course course = courseSchedule.getCourses().get(i);
            data[i][0] = course.getName();
            data[i][1] = course.getCredit();
            data[i][2] = course.getDay();
            data[i][3] = course.getStart();
            data[i][4] = course.getTime();
        }
        return data;
    }

    public JTable getTable() {
        return table;
    }
}
```

CourseScheduleSection.java

```java
package org.example.view;

import javax.swing.*;
import java.awt.*;
import org.example.Entity.CourseSchedule;
import org.example.controller.CourseTableController;

public class CourseScheduleSection extends JPanel {
    private CourseSchedule schedule;
    private InputPanel inputPanel;
    private JTextField myCoursesField;
    private CourseListPanel listPanel;
    private ScheduleTablePanel tablePanel;
    private ButtonPanel buttonPanel;

    public CourseScheduleSection(CourseSchedule schedule, ButtonPanel buttonPanel) {
        this.schedule = schedule;
        this.buttonPanel = buttonPanel;

        setLayout(new BorderLayout());

        initializeTopPanel();
        updatePanels();
    }

    private void initializeTopPanel() {
        inputPanel = new InputPanel();
        myCoursesField = new JTextField("My Courses");
        myCoursesField.setEditable(false);
    }

    public void updatePanels() {
        removeAll();

        Object[][] data = CourseListPanel.getTableData(schedule);
        listPanel = new CourseListPanel(data);
        tablePanel = new ScheduleTablePanel(data);

        // 입력 + 리스트를 왼쪽에 묶기
        JPanel inputAndListPanel = new JPanel();
        inputAndListPanel.setLayout(new BoxLayout(inputAndListPanel, BoxLayout.Y_AXIS));
        inputAndListPanel.add(inputPanel);
        inputAndListPanel.add(myCoursesField);
        inputAndListPanel.add(listPanel);

        // 위쪽 전체 묶기: 입력+리스트 + 버튼
        JPanel topPanel = new JPanel(new BorderLayout());
        topPanel.add(inputAndListPanel, BorderLayout.CENTER); // 왼쪽 입력
        topPanel.add(buttonPanel, BorderLayout.EAST);         // 오른쪽 버튼

        // 상단 패널 (입력 + 버튼), 하단 시간표
        add(topPanel, BorderLayout.NORTH);
        add(tablePanel, BorderLayout.CENTER);

        new CourseTableController(listPanel, inputPanel);

        revalidate();
        repaint();
    }

    public void setSchedule(CourseSchedule schedule) {
```

```
        this.schedule = schedule;
        updatePanels();
    }

    public CourseSchedule getSchedule() {
        return schedule;
    }

    public InputPanel getInputPanel() {
        return inputPanel;
    }

    public CourseListPanel getListPanel() {
        return listPanel;
    }
}
```

InputPanel.java

```
package org.example.view;

import javax.swing.*;
import java.awt.*;

public class InputPanel extends JPanel {
    public JTextField subjectField = new JTextField();
    public JTextField creditField = new JTextField();
    public JComboBox<String> dayComboBox = new JComboBox<>(new String[]{"Mon", "Tue", "Wed",
"Thu", "Fri"});
    public JTextField startField = new JTextField();
    public JTextField timeField = new JTextField();

    public InputPanel() {
        setLayout(new GridLayout(2, 5, 5, 5));

        add(new JTextField("과목명"));
        add(new JTextField("학점"));
        add(new JTextField("요일"));
        add(new JTextField("시작 교시"));
        add(new JTextField("시간"));

        add(subjectField);
        add(creditField);
        add(dayComboBox);
        add(startField);
        add(timeField);

        setReadOnly(true);
    }

    // 필요하면 선택된 요일 가져오기
    public String getSelectedDay() {
        return (String) dayComboBox.getSelectedItem();
    }

    public void setInputFields(String subject, String credit, String day, String start, String
time) {
        subjectField.setText(subject);
        creditField.setText(credit);
        dayComboBox.setSelectedItem(day);
        startField.setText(start);
```

```
        timeField.setText(time);
    }

    public void setReadOnly(boolean readonly) {
        subjectField.setEditable(!readonly);
        creditField.setEditable(!readonly);
        startField.setEditable(!readonly);
        timeField.setEditable(!readonly);
        dayComboBox.setEnabled(!readonly);
    }

}
```

ScheduleTablePanel.java

```
package org.example.view;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;

public class ScheduleTablePanel extends JPanel {
    public ScheduleTablePanel(Object[][] courseData) {
        setLayout(new BorderLayout());

        String[] days = {"Mon", "Tue", "Wed", "Thr", "Fri"};
        Object[][] schedule = toScheduleTable(courseData);
        JTable table = new JTable(new DefaultTableModel(schedule, days));

        JTable rowHeader = new JTable(new DefaultTableModel(
                new Object[][]{
                        {"1"}, {"2"}, {"3"}, {"4"}, {"5"}, {"6"}, {"7"}, {"8"}, {"9"}
                }, new String[]{""}
        ));
        rowHeader.setEnabled(false);
        rowHeader.setPreferredScrollableViewportSize(new Dimension(30, 0));

        JScrollPane scrollPane = new JScrollPane(table);
        scrollPane.setRowHeaderView(rowHeader);
        add(scrollPane, BorderLayout.CENTER);
    }

    public static Object[][] toScheduleTable(Object[][] courseData) {
        Object[][] table = new Object[9][5]; // 9교시 × 5요일

        for (Object[] course : courseData) {
            String name = (String) course[0];
            String day = (String) course[2];
            int start = Integer.parseInt(course[3].toString()) - 1;
            int time = Integer.parseInt(course[4].toString());

            int dayIdx = switch (day) {
                case "Mon" -> 0;
                case "Tue" -> 1;
                case "Wed" -> 2;
                case "Thu" -> 3;
                case "Fri" -> 4;
                default -> -1;
            };

            if (dayIdx == -1 || start < 0 || start + time > 9) continue;
```

```
            for (int i = 0; i < time; i++) {
                table[start + i][dayIdx] = name;
            }
        }
        return table;
    }

}
```

CourseScheduleMain.java

```
package org.example;

import org.example.Entity.CourseSchedule;
import org.example.controller.CourseButtonController;

import javax.swing.*;
import java.net.InetAddress;
import java.time.LocalDateTime;
import org.example.view.ButtonPanel;
import org.example.view.CourseScheduleSection;

public class CourseScheduleMain {
    public static void main(String[] args) throws Exception {
        System.out.println("");
        System.out.println("At " + LocalDateTime.now());
        System.out.println("By " + System.getProperty("user.name") + " at " +
InetAddress.getLocalHost().getHostName());
        System.out.println("");

        JFrame frame = new JFrame("Planning Course Schedule");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(800, 1200);

        ButtonPanel buttonPanel = new ButtonPanel();
        CourseSchedule schedule = new CourseSchedule("myschedule-dump.data");
        CourseScheduleSection courseSection = new CourseScheduleSection(schedule, buttonPanel);

        frame.add(courseSection); // CENTER로 자동 배치
        new CourseButtonController(frame, courseSection, buttonPanel);

        frame.setVisible(true);
    }
}
```

자체 평가 표

평가표 (즉, 리포트 작성시 체크 사항)

평가 항목	학생 자체 평가 (리포트 해당 부분 표시 및 간단한 의견)	평가 (빈칸)	점수 (빈칸)
완성도 (동작 여부) - "초기" 화면/동작 - "Show List" 동작 - 과목 선택 동작 - "ADD" 동작 - "DELETE" 동작 (선택이 없을 때 경고 화면) - "SAVE" / "LOAD" 동작 파일 선택 화면 - 기타 비정상 동작 실험 (구현 한계 점검)	초기 화면/ showList/ 과목선택 / ADD/DELETE(경고포함) / SAVE LOAD 동작 + 파일선택 기타 비정상 동작실험 등 레포트 앞 순서에 작동 동작 확인 첨부 및 구현 인증 완료		
설계 노트 - 주요 결정사항 및 근거 - 한계/문제점 - 해결 방안 - 필수내용: * 프로그램 구성(Class 구조) * member visibility	각 설계 방안 별 근거 및 한계를 제시하고자 노력, class별 한계 및 해결방안을 제시 class 구조 및 memver visibility 설계 내용 포함		
리포트 - 평가자 시각으로 리포트 검토 - 위의 평가 요소들이 명확하게 기술되었는가?	위의 내용 모두 기록 완료		
총평/계			

* 학생 자체 평가는 점수에 반영되지 않음.
* 학생 스스로 자신의 보고서를 평가하면서, 체계적으로 프로젝트를 마무리하도록 유도하는 것이 목적임.