Schedule Class

```
String[] days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday"};
```

```
for(int row = 0; row < 4; row++){
    for(int col = 0; col < 5; col++){
        gbc.gridx = col;
        gbc.gridy =row +1;
        JButton button = new JButton( text: "Period "+(row+1));
        button.setPreferredSize(new Dimension( width: 130, height: 100));

        final String day = days[col];
        final int lessonnum = row;


        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                LessonPlan l = new LessonPlan(day, l: lessonnum+1);

            }
        });


        panel.add(button, gbc);
    }
}
```

Rather than initializing all 20 buttons of the 5x4 schedule grid separately, I decided to create them using nested For loops. The outer loop iterates through the rows, initializing from buttons from top to bottom, column by column. When defining each button, the row variable is added as a component of each button's name. I then initialize day (specified by col - e.g. 0=”Monday”, 1= “Tuesday”, etc) and lessonnum (the row number), to serve as unique identifiers for each Planner Page.

Indeed, these two variables are passed into the constructor method of LessonPlan, once a button click is detected by the action listener. These will act as unique identifiers for a LessonPlan (Planner Page) once it is constructed.

Calendar Class

Making the Calendar

```
JFrame calenderframe = new JFrame( title: "Calender");
calenderframe.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
calenderframe.setSize( width: 800, height: 600);

JCalendar calender = new JCalendar();
//add a propertychangelistener to getdaychooser, a component of JCalender
//daychooser records and updates the most recent day clicked
//the propertychangelistener will react to any changes in daychooser.
calender.getDayChooser().addPropertyChangeListener( propertyName: "day", new PropertyChangeListener() {
    @Override
    public void propertyChange(PropertyChangeEvent evt) {
        Date selectedDate = calender.getDate(); //this method retrieves the most recently selected date
        makeframe(selectedDate);


    }
});
calenderframe.add(calender);

calenderframe.setVisible(true);
```

Rather than creating a Calendar GUI from scratch, which would simply be time-consuming and impractical, I downloaded the JCalendar library class from online (Todter). JCalendar has the GUI prebuilt, as well as methods for easy use. For example, getDate. After I add a PropertyChangeListener to the Calendar's DayChooser (which logs the most recently selected day) this method retrieves the Date of the button clicked and assigns it to the variable selectedDate.

DayFrame

```
for(int row=0;row<4;row++){ //looped button creation minimises code
    gbc.gridy=row;
    JButton button=new JButton( text: "Period "+(row+1));
    button.setPreferredSize(new Dimension( width: 130, height: 100));

    int finalRow = row;
    button.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            LessonPlan l = new LessonPlan( d: "",  l: finalRow +1,date);
        }
    });
```

When making a Dayframe, (using a similar process to making a schedule, but with only one column), clicking on a Period button makes an instance of LessonPlan as usual. But in order to differentiate LessonPlans made from the Calendar and from the Schedule, the date is passed through as a parameter.

LessonPlan Class

Differentiating Calendar and Schedule Plans

```
if(date!=null){ //plan pages created from the calendar page need a specific name
    lessonFrame = new JFrame( title: "Period: "+lessonnum+ " ;"+date);
}else{
    lessonFrame = new JFrame( title: day+" - Period: "+lessonnum);
}
```

Rather than implementing method overloading - making 2 constructor methods taking different parameters - given that there would be minimal differences between a Calendar-made instance and a Schedule-made instance, I reasoned it would be more concise to simply use an if else statement. Although this requires Schedule to pass a Date as a parameter, this can simply be null, setting the Frame name to contain only the day and period number.

GridBagConstraints

To systematise my GUI creation (evident in the loops for the schedule grids), I imported the GridBagLayout layout manager rather than the standard FlowLayout used by java swing.

```
lessonFrame.setSize( width: 800, height: 600);
lessonFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
lessonFrame.setLayout(new GridBagLayout());
GridBagConstraints gbc = new GridBagConstraints();
gbc.insets = new Insets( top: 5,  left: 5,  bottom: 5,  right: 5);

gbc.fill = GridBagConstraints.HORIZONTAL;
//now adding all the components:
JLabel subjectlabel = new JLabel( text: "Subject:");
gbc.gridx= 0;
gbc.gridy=0;
gbc.gridwidth =1;
lessonFrame.add(subjectlabel, gbc);
subjectfield = new JTextField( columns: 12);
subjectfield.setPreferredSize(new Dimension( width: 150, height: 25));
gbc.gridx =1;
gbc.gridy= 0;
gbc.gridwidth=2;
lessonFrame.add(subjectfield, gbc);
```

What makes GridBagLayout unique is its use of columns and rows. Once a GridBagConstraint gridx or gridy value is added or changed, this sets the column/row to the corresponding number, arranging all components to fit. These variables streamlined my schedule creation as they provided easy values to increment in my loops.
GridBagConstraints also had many other attributes that I tweaked to arrange the layout. For example, gridwidth, which determines the number of columns a component spans. I primarily used this to adjust sizes of components.

File Writing
To specify the filename of a saved plan:

```
final String fileName;
if(date!=null){ //specific file name for plan saved from Calendar
    SimpleDateFormat simpledate = new SimpleDateFormat( pattern: "yyyy-MM-dd");
    //removes the time from Date object
    fileName = simpledate.format(date) + "; Period: " + lessonnum + ".txt";
}else{
    fileName = day+ "; Period: " +lessonnum + ".txt";
}
```

To firstly identify if this is a LessonPlan instance made from the calendar or schedule, I again checked the null status of date. Differentiating filenames is important so saved lesson plans don't overwrite each other. I also remove the time component of the Date using SimpleDateFormat. Without this, clicking on a day in the calendar would generate a unique timestamp, creating a new file each time.

```
save.addActionListener(new ActionListener()  {
    @Override
    public void actionPerformed(ActionEvent e){
        File file = new File(fileName);

        try(BufferedWriter writer = new BufferedWriter(new FileWriter(fileName))){
            writer.write(subjectfield.getText());
            writer.newLine();
            writer.write(goalfield.getText());
            writer.newLine();
            writer.write(durationfield.getText());
            writer.newLine();
```

Once the save button is clicked, a File object is constructed using fileName defined before. Next, I used a BufferedWriter with a FileWriter to efficiently write data into the file. The BufferedWriter enhances performance by writing data into the buffer first, reducing the amount of direct I/O operations, improving overall efficiency (Tomosvari).

For the first three fields, I sequentially write each textfield into the file using write(), retrieving the text through the Textfield's accessor method getText(). I increment a new line each time to separate data from different Textfields.

```
        //seperates the large text fields as these could be many lines
        writer.write( str: "===");
        writer.newLine();
        // write the materials text
        writer.write(materialsarea.getText());
        writer.newLine();

        writer.write( str: "===");
        writer.newLine();
        writer.write(planarea.getText());
        writer.newLine();

        writer.write( str: "===");
        writer.newLine();
        writer.write(notesarea.getText());

    }catch(IOException ex){
        JOptionPane.showMessageDialog(lessonFrame, message: "Error saving file: "+ex.getMessage(),
                title: "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }
    JOptionPane.showMessageDialog(lessonFrame, message: "Lesson plan saved", title: "Success",
            JOptionPane.INFORMATION_MESSAGE);
}
```

As the three Textfields near the bottom of the page were larger and could include multiple lines, in order to delineate the boundaries of each field, I added "===" as a limiter. This is important for the autoload functionality as the filereader needs to know exactly where one section of text ends and begins.

An example of the contents of a .txt file can be seen below. The text is written from the text fields in order of "Subject", "Lesson Goal", "Duration", "Materials", "Plan Details" and finally "Notes".

Japanese
Teach greetings and how to use them in conversation.
50 mins
===
Flashcards with greetings

Audio clips of native pronunciation

Whiteboard and markers

===
Introduction (10 min): Explain the importance of greetings in Japanese culture.

Listening & Repeat (15 min): Play audio clips of greetings like "こんにちは" (Hello) and "おはようございます" (Good morning), then have students repeat.

Pair Practice (15 min): Students practice greeting each other using flashcards.

Mini Role-Play (10 min): Each student greets the teacher using a greeting of their choice.

===
Keep pronunciation practice slow and clear to help beginners.

If time allows, introduce casual vs. formal greetings.

Autoload

```
if (file.exists()) {
    try (BufferedReader reader = new BufferedReader(new FileReader(file))) {

        subjectfield.setText(reader.readLine());
        goalfield.setText(reader.readLine());
        durationfield.setText(reader.readLine());

        reader.readLine(); //this is the "====" line

        StringBuilder materialscontent = new StringBuilder();
        String line = reader.readLine();
        while (line != null && !line.equals("===")) { //the line must not be null and must not be "==="
            materialscontent.append(line).append("\n");
            //adds the string as a new line to the stringbuilder object
            line = reader.readLine();//move to next line
        }
        line =reader.readLine(); //skip the "===" line
```

If a file exists for the instance of LessonPlan, a BufferedReader reads the first line of the .txt file using readLine(). This is inserted into subjectfield using setText(). This process repeats for goalfield and durationfield. Note that simply calling readLine() increments the reader's position in the file; This is why an additional method call is needed to skip the "===" line.

As the last 3 Textfields could span multiple lines, the file reading process is more complicated. To streamline this process, I used a StringBuilder, as they are mutable String objects that I can easily add to using the append() method (Ravikiran). To firstly identify if a line is not a boundary, two conditions must be met: the line must not be null, and it must not be the limiter “===”. Null means the file does not exist, different from an empty string (“”) which represents a blank line. This allows indentations and empty spaces to remain part of the materials String. These 2 conditions are placed inside a while loop, each iteration appending the text from readLine() as a new line using “\n”.
As each method call increments the readers line in the file, using readline() directly in the while condition would cause it to move forward twice, potentially skipping lines. Assigning the line to a variable minimises method calls, only incrementing lines once at the end of each iteration.

PlanTemplate Class

```
private void savetemplate(){
    String fileName = filename.getText();
    if (fileName.isEmpty()){ //do not allow empty filenames
        JOptionPane.showMessageDialog(lessonFrame,
                message: "Please enter a template name.",
                title: "Error",JOptionPane.ERROR_MESSAGE);
        return;
    }
    File dir=new File( pathname: "templates"); //creates a directory
    if(dir.exists()==false){ //if there is a directory folder on the computer
        dir.mkdir(); //method to create templates folder
    }
    File file = new File(dir, child: fileName+".txt");
```

Pressing the “Save” button on the Template page calls the savetemplate() method. It firstly checks if the file name Textfield is empty, displaying an error message if it is. Using File object dir, I call the mkdir() to create a folder called “templates” in the project folder - only if it doesn’t already exist. Finally, a new .txt file is created inside this directory using the provided filename. Writing this file uses the same process as before.

Word count: 1000

Work Cited

Ravikiran, A. S. “Stringbuilder in Java: Constructors, Methods, and Examples.” *Simplilearn.com*, 16 July 2024, www.simplilearn.com/tutorials/java-tutorial/stringbuilder-in-java. Accessed 9 Mar. 2025.

Todter, Kai. “JCalendar.” *Toedter.com*, toedter.com/jcalendar. Accessed 5 Mar. 2025.

Tomosvari, Imre. “Java File Writing I/O Performance.” *TMSVR - Dev Blog*, 24 Feb. 2025, tmsvr.com/java-file-writing-i-o-performance. Accessed 9 Mar. 2025.