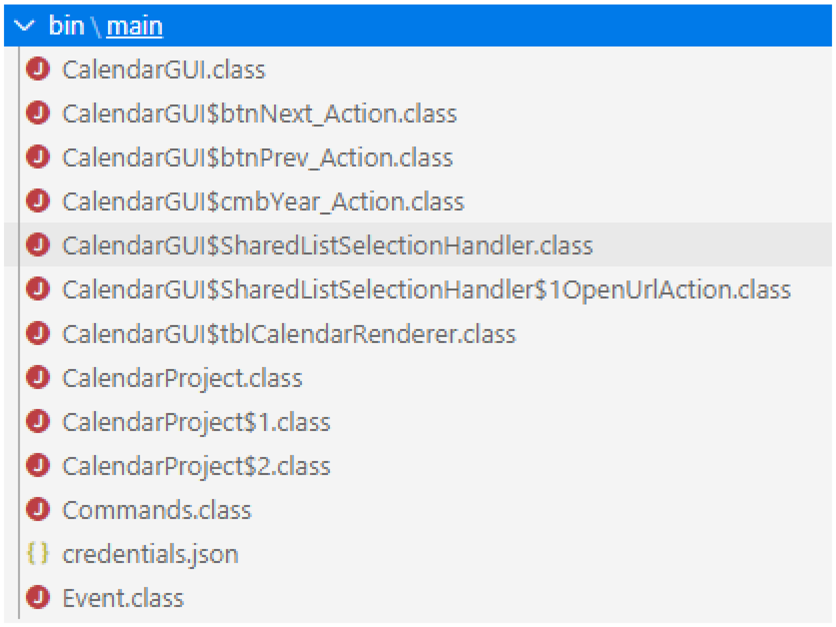
Criterion C: Development

All Classes:



bin \ main

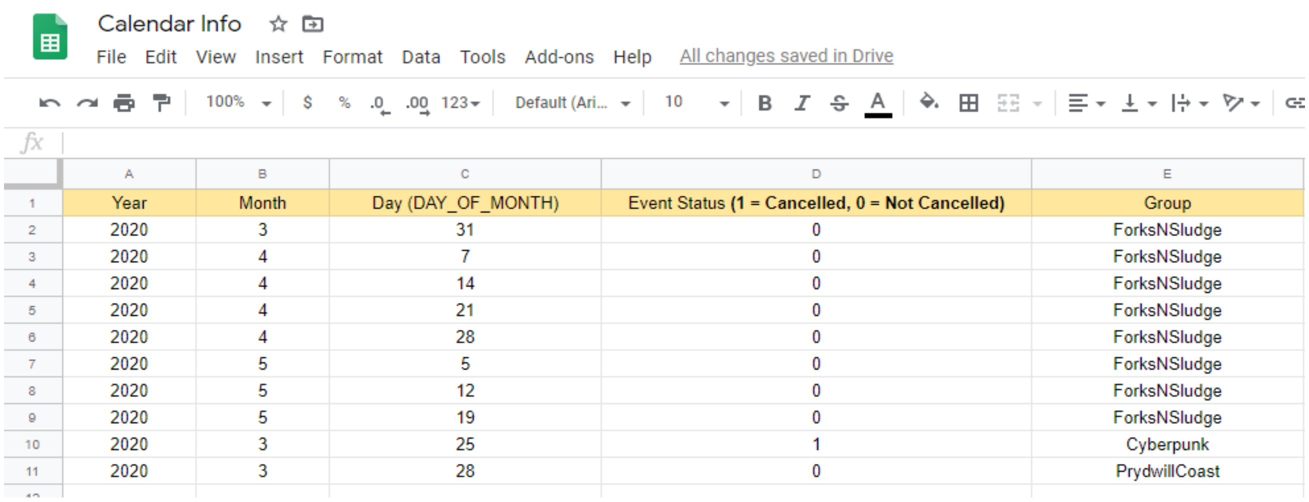
CalendarGUI.class
CalendarGUI$btnNext_Action.class
CalendarGUI$btnPrev_Action.class
CalendarGUI$cmbYear_Action.class
CalendarGUI$SharedListSelectionHandler.class
CalendarGUI$SharedListSelectionHandler$1OpenUrlAction.class
CalendarGUI$tblCalendarRenderer.class
CalendarProject.class
CalendarProject$1.class
CalendarProject$2.class
Commands.class
credentials.json
Event.class

List of Techniques

A. Collections
B. I/O and Exception Handling
C. Inheritance
D. Abstraction
E. Overriding methods
F. GUI
G. For Loops
H. Loading from external sources

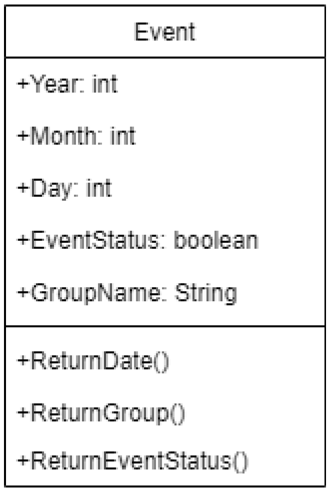
Program Setup

For the program to function properly, it needs to connect to both APIs, and set up the GUI components. The first thing the program needs to do is get the event information from the GoogleSheet.


Calendar Info
File Edit View Insert Format Data Tools Add-ons Help All changes saved in Drive
100% $ % .0 .00 123 Default (Ari... 10 B I S A

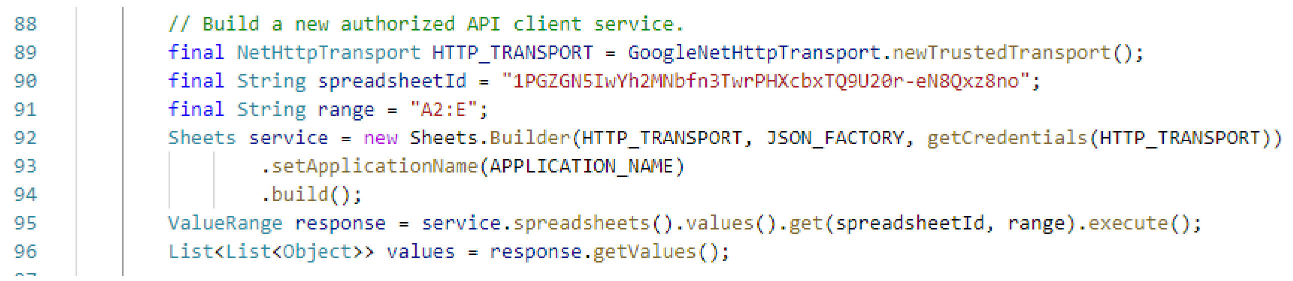
Year	Month	Day (DAY_OF_MONTH)	Event Status (1 = Cancelled, 0 = Not Cancelled)	Group
2020	3	31	0	ForksNSludge
2020	4	7	0	ForksNSludge
2020	4	14	0	ForksNSludge
2020	4	21	0	ForksNSludge
2020	4	28	0	ForksNSludge
2020	5	5	0	ForksNSludge
2020	5	12	0	ForksNSludge
2020	5	19	0	ForksNSludge
2020	3	25	1	Cyberpunk
2020	3	28	0	PrydwillCoast

The spreadsheet is set up so each row corresponds with one event. By getting each cell in the row the program instantiates an event object.



The event object conveniently stores the needed values and can easily be expanded to include more details such as an event description.

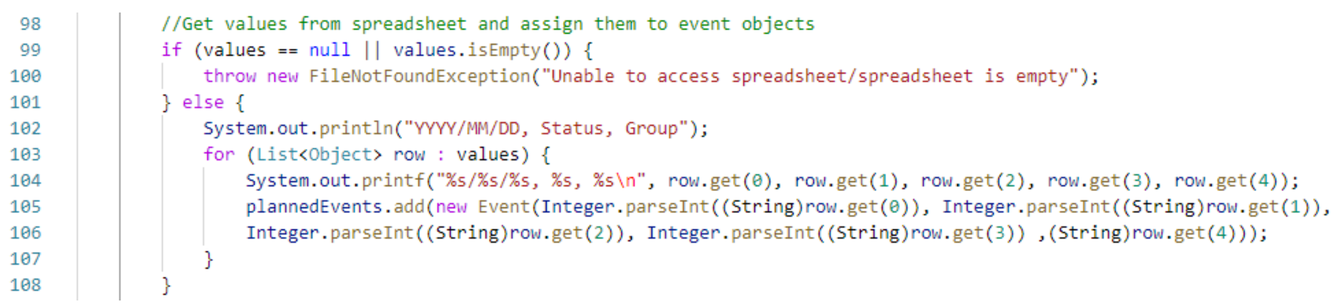
In order to connect to the spreadsheet, the program creates a NetHttpTransport object, using it to build a Sheets object so the application can read from the spreadsheet specified by spreadsheetId. It then gets the values from the desired range, and saves them to a list of lists named values. The outer list represents all of the data, and the inner lists each represent a row, and each item in the inner list represents one value. All these methods and classes come from the Sheets API library[1].



```
        // Build a new authorized API client service.
        final NetHttpTransport HTTP_TRANSPORT = GoogleNetHttpTransport.newTrustedTransport();
        final String spreadsheetId = "1PGZGN5IwYh2MNbfn3TwrPHXcbxTQ9U20r-eN8Qxz8no";
        final String range = "A2:E";
        Sheets service = new Sheets.Builder(HTTP_TRANSPORT, JSON_FACTORY, getCredentials(HTTP_TRANSPORT))
                .setApplicationName(APPLICATION_NAME)
                .build();
        ValueRange response = service.spreadsheets().values().get(spreadsheetId, range).execute();
        List<List<Object>> values = response.getValues();
```

1 Google Developers. 2020. Java Quickstart | Sheets API | Google Developers. [online] Available at: <https://developers.google.com/sheets/api/quickstart/java> [Accessed 25 March 2020].

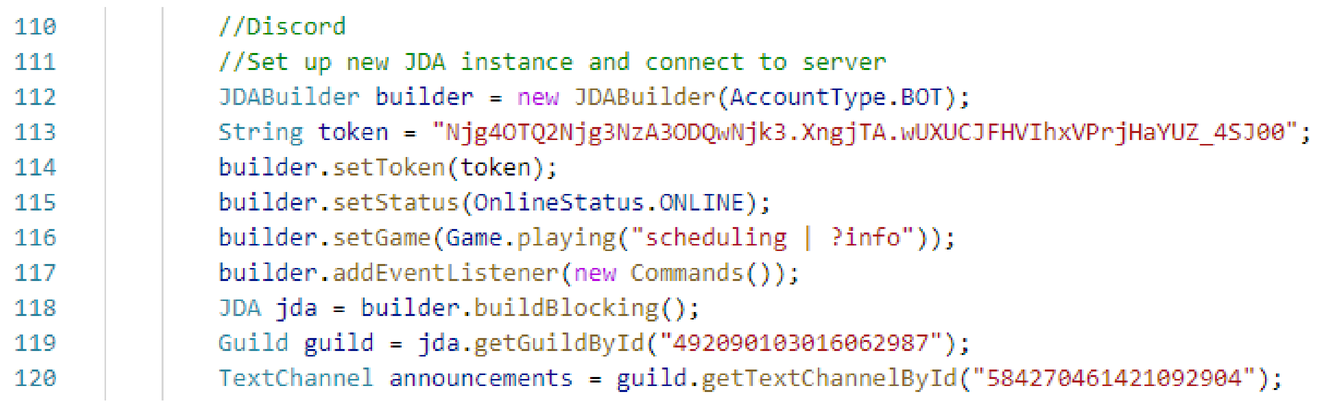
The app then checks to make sure that values is not empty, throwing a FileNotFoundException if it is. Then it iterates through values, assigning each row to an event object and adding it to an ArrayList of events.



```java
//Get values from spreadsheet and assign them to event objects
if (values == null || values.isEmpty()) {
    throw new FileNotFoundException("Unable to access spreadsheet/spreadsheet is empty");
} else {
    System.out.println("YYYY/MM/DD, Status, Group");
    for (List<Object> row : values) {
        System.out.printf("%s/%s/%s, %s, %s\n", row.get(0), row.get(1), row.get(2), row.get(3), row.get(4));
        plannedEvents.add(new Event(Integer.parseInt((String)row.get(0)), Integer.parseInt((String)row.get(1)),
        Integer.parseInt((String)row.get(2)), Integer.parseInt((String)row.get(3)) ,(String)row.get(4)));
    }
}
```

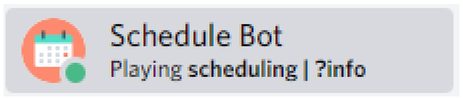
This method ensures that each Event object is initialized correctly with a year, month, day, cancelled status, and group name.

After connecting to the Google API and getting the needed data, the program must then connect to the Discord API so it send notifications. It does this by using the JDABuilder class to log into a bot account using a token provided by Discord.



```java
//Discord
//Set up new JDA instance and connect to server
JDABuilder builder = new JDABuilder(AccountType.BOT);
String token = "Njg4OTQ2Njg3NzA3ODQwNjk3.XngjTA.wUXUCJFHVIhxVPrjHaYUZ_4SJ00";
builder.setToken(token);
builder.setStatus(OnlineStatus.ONLINE);
builder.setGame(Game.playing("scheduling | ?info"));
builder.addEventListener(new Commands());
JDA jda = builder.buildBlocking();
Guild guild = jda.getGuildById("492090103016062987");
TextChannel announcements = guild.getTextChannelById("584270461421092904");
```

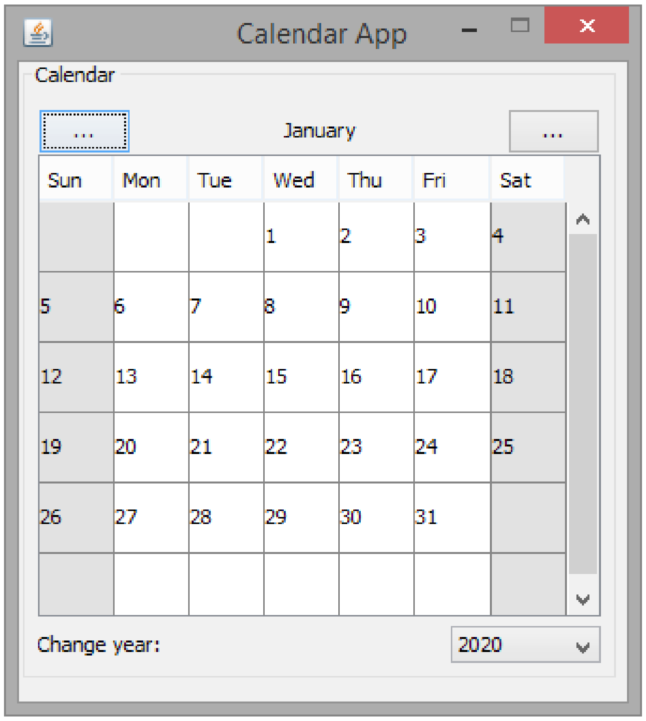
Once this is done, the application is connected to Discord and the bot account appears online.



After this the application then creates a CalenderGUI object[2], and uses it to instantiate all the Java Swing components that are needed.

```java
//Creates calendar GUI
 CalendarGUI calendar = new CalendarGUI(plannedEvents);
 try {UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());}
catch (ClassNotFoundException e) {
    throw new ClassNotFoundException();
}
catch (InstantiationException e) {
    throw new InstantiationException();
}
catch (IllegalAccessException e) {
    throw new IllegalAccessException();
}
catch (UnsupportedLookAndFeelException e) {
    throw new UnsupportedLookAndFeelException("");
}
 //Prepare frame
 calendar.frmMain = new JFrame ("Calendar App"); //Create frame
 calendar.frmMain.setSize(330, 375); //Set size to 400x400 pixels
 calendar.pane = calendar.frmMain.getContentPane(); //Get content pane
 calendar.pane.setLayout(null); //Apply null layout
```

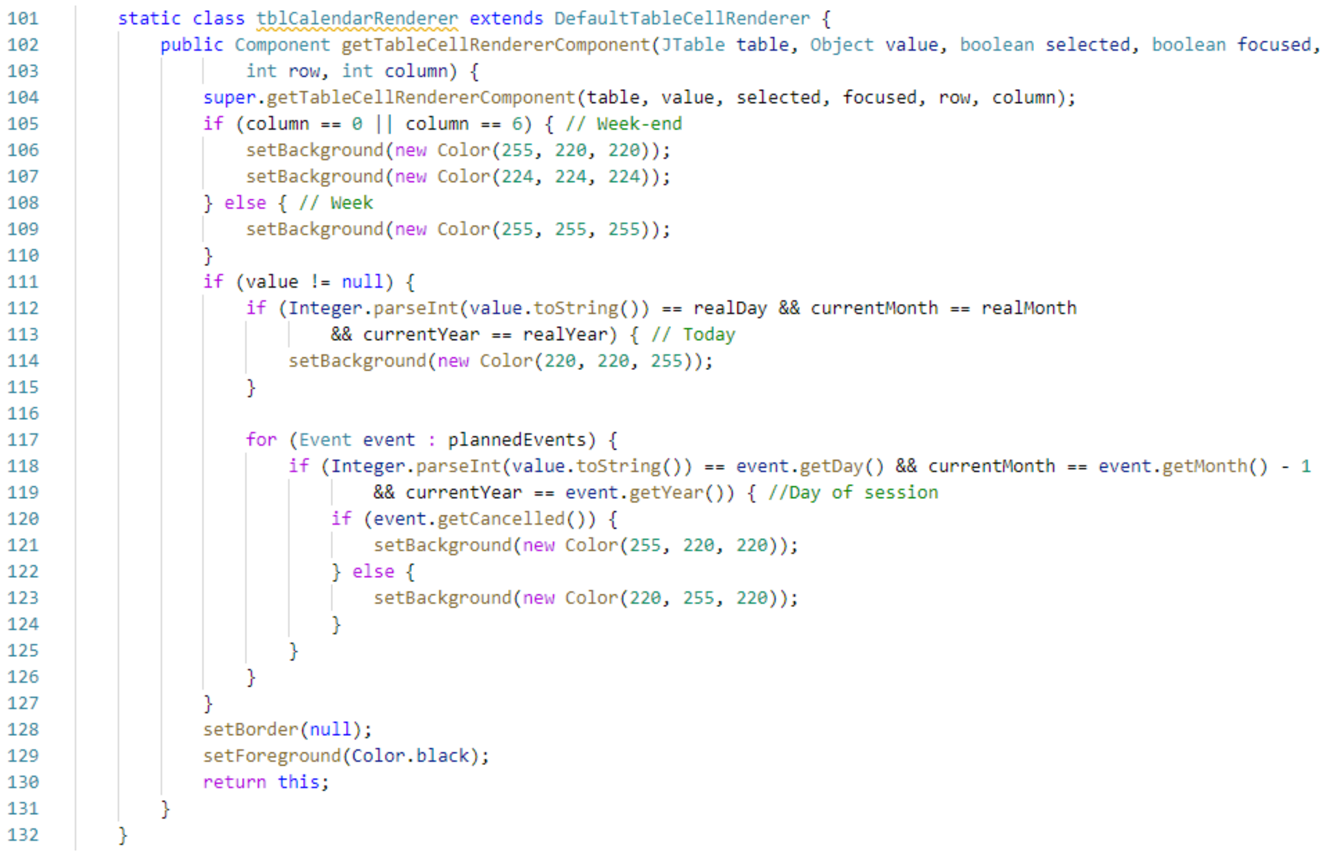
After adding all the buttons, getting the current date from the GregorianCalendar class, and populating the table, the frame is made visible.



Sun	Mon	Tue	Wed	Thu	Fri	Sat
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

2 Javahungry.blogspot.com. 2020. Calendar Implementation :Swing GUI Based Java Program | Java Hungry. [online] Available at: <https://javahungry.blogspot.com/2013/06/calendar-implementation-gui-based.html> [Accessed 25 March 2020].

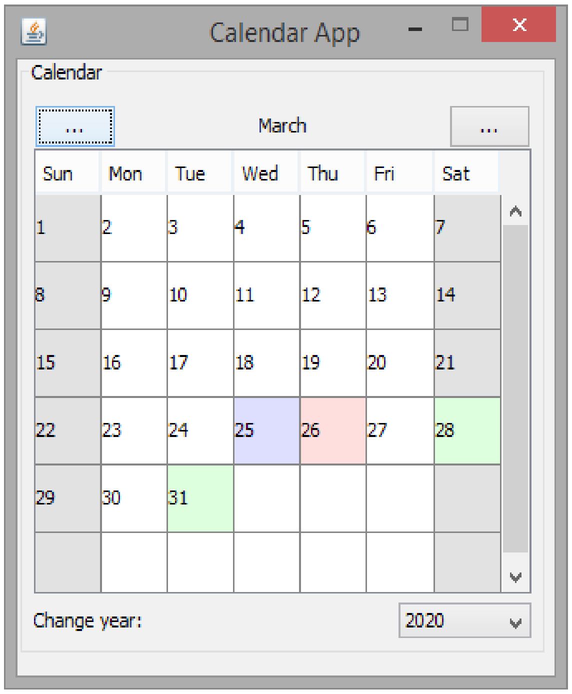
To display which days have sessions planned on them, I used the tblCalendarRenderer class which extends the DefaultTableCellRenderer class to check all the dates being displayed and compare them to every element in plannedEvents.



```java
static class tblCalendarRenderer extends DefaultTableCellRenderer {
    public Component getTableCellRendererComponent(JTable table, Object value, boolean selected, boolean focused,
            int row, int column) {
        super.getTableCellRendererComponent(table, value, selected, focused, row, column);
        if (column == 0 || column == 6) { // Week-end
            setBackground(new Color(255, 220, 220));
            setBackground(new Color(224, 224, 224));
        } else { // Week
            setBackground(new Color(255, 255, 255));
        }
        if (value != null) {
            if (Integer.parseInt(value.toString()) == realDay && currentMonth == realMonth
                    && currentYear == realYear) { // Today
                setBackground(new Color(220, 220, 255));
            }

            for (Event event : plannedEvents) {
                if (Integer.parseInt(value.toString()) == event.getDay() && currentMonth == event.getMonth() - 1
                        && currentYear == event.getYear()) { //Day of session
                    if (event.getCancelled()) {
                        setBackground(new Color(255, 220, 220));
                    } else {
                        setBackground(new Color(220, 255, 220));
                    }
                }
            }
        }
        setBorder(null);
        setForeground(Color.black);
        return this;
    }
}
```

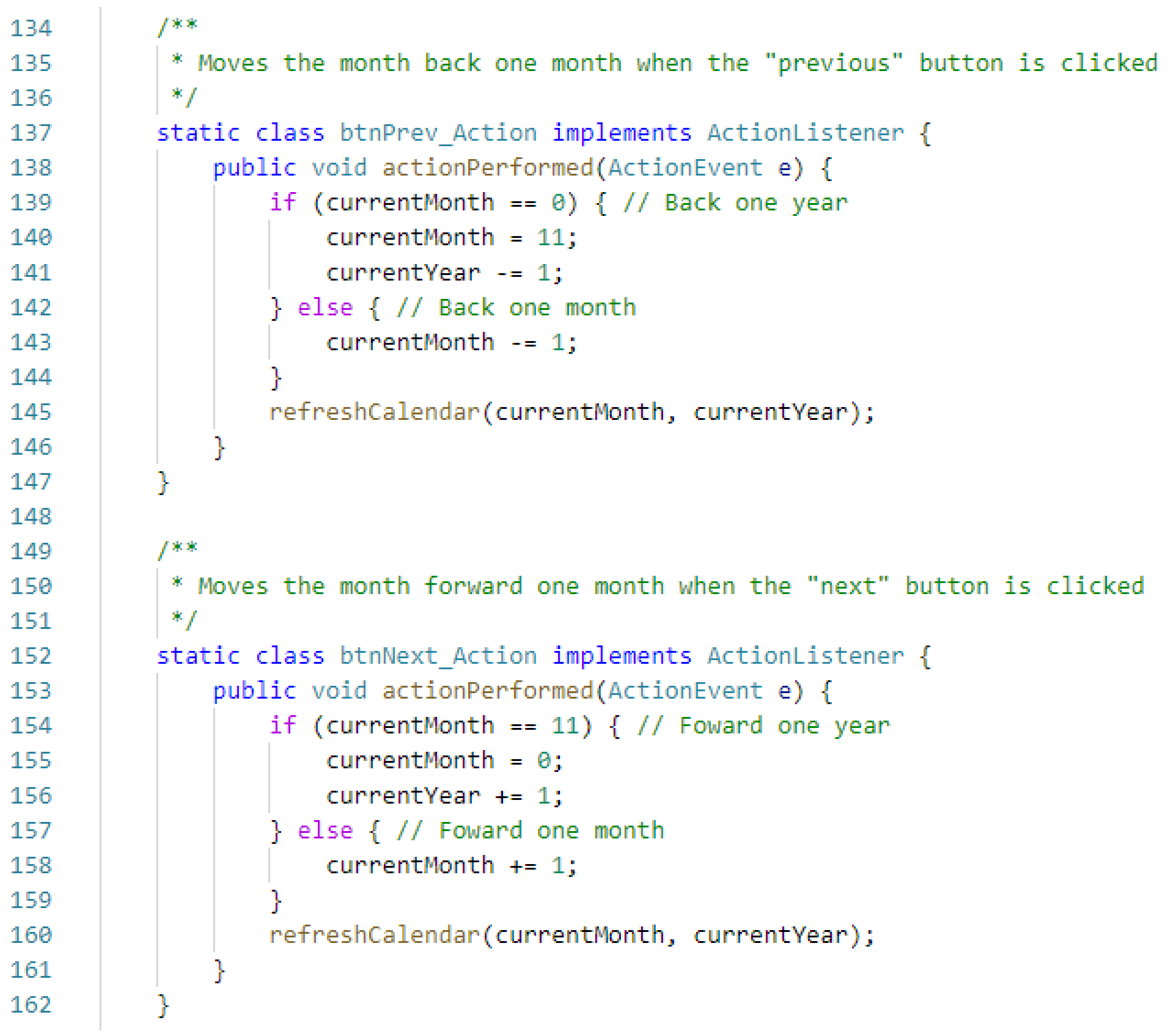
If the program finds a match, it sets the color of that date to green if getCancelled() returns false, and red if it returns true. Otherwise it sets it to blue to indicate the current date.



Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

The Running Program

Once everything has been initialized, it is ready to recieve user input. On the calendar, there are two buttons on either side of the month label. Both implement the ActionListener class, and when clicked, either add one to currentMonth or subract one from currentMonth. The method is also able to handle changing the year as needed. It then calls the refreshCalendar method with the updated date information.

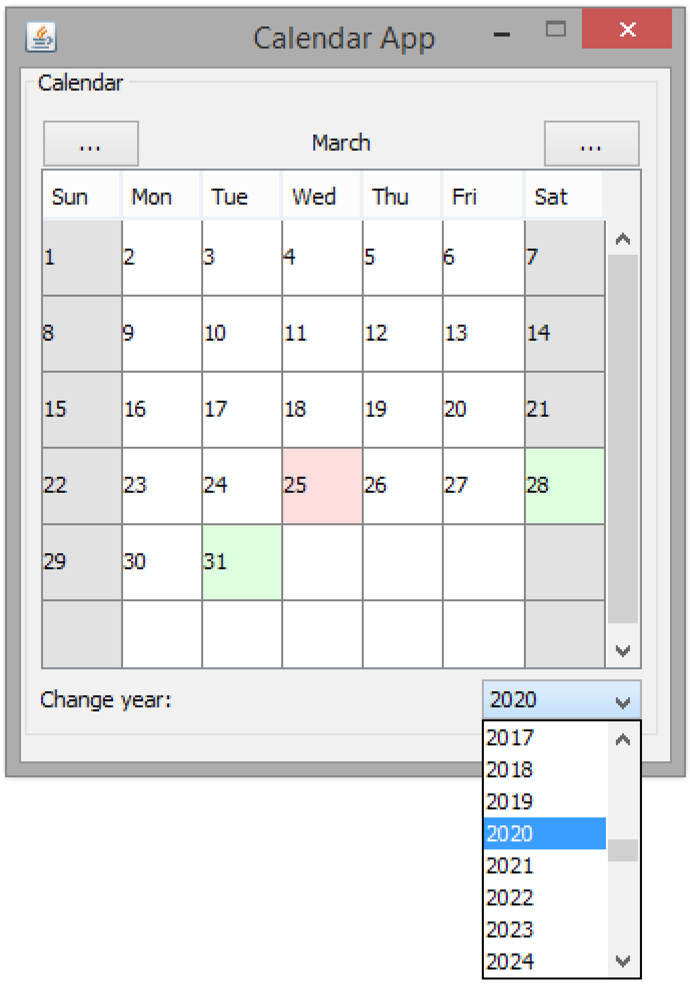


```java
    /**
     * Moves the month back one month when the "previous" button is clicked
     */
    static class btnPrev_Action implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            if (currentMonth == 0) { // Back one year
                currentMonth = 11;
                currentYear -= 1;
            } else { // Back one month
                currentMonth -= 1;
            }
            refreshCalendar(currentMonth, currentYear);
        }
    }

    /**
     * Moves the month forward one month when the "next" button is clicked
     */
    static class btnNext_Action implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            if (currentMonth == 11) { // Foward one year
                currentMonth = 0;
                currentYear += 1;
            } else { // Foward one month
                currentMonth += 1;
            }
            refreshCalendar(currentMonth, currentYear);
        }
    }
```

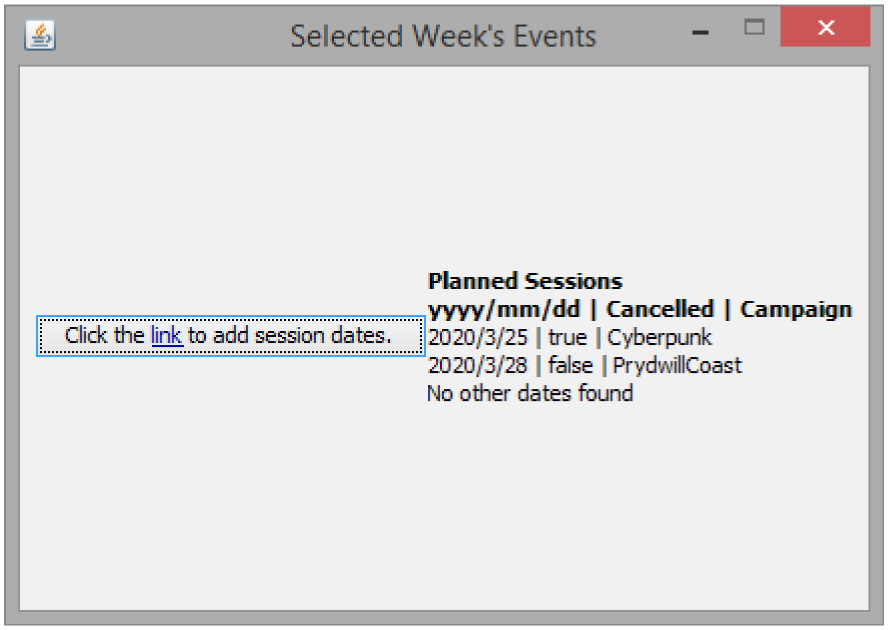
If the user ever wants to view a specific year, they can use the JComboBox object which has an ActionListener added to it so that when it is clicked, a drop down menu appears.

```java
    /**
     * Changes the year to be displayed to the one selected by the user
     */
    static class cmbYear_Action implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            if (cmbYear.getSelectedItem() != null) {
                String b = cmbYear.getSelectedItem().toString();
                currentYear = Integer.parseInt(b);
                refreshCalendar(currentMonth, currentYear);
            }
        }
    }
```



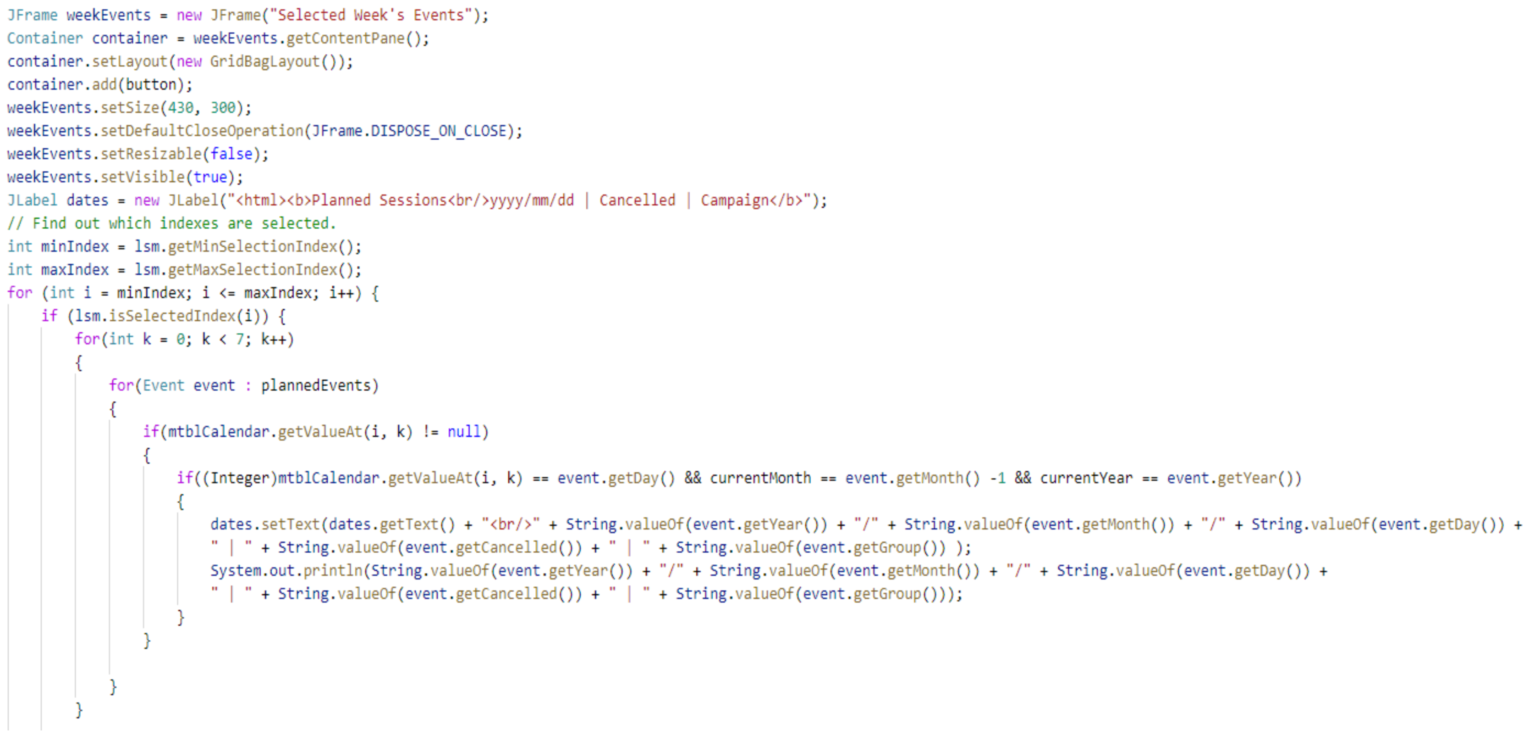
Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

In order to see specific details about which the sessions, the user is able to click on a row in the calendar and see the specific details of events that are planned for that week.



Planned Sessions
yyyy/mm/dd | Cancelled | Campaign
2020/3/25 | true | Cyberpunk
2020/3/28 | false | PrydwillCoast
No other dates found

The calendar renders dates by using a DefaultTableModel object, meaning rows can be accessed extremely easily. A ListSelectionListener then triggers when the mouse clicks on a row of the table. The valueChanged method can then find which indexes are selected, and compare the date information in each index to the date information stored in plannedEvents. It then prints out all the matches to a new JFrame window.



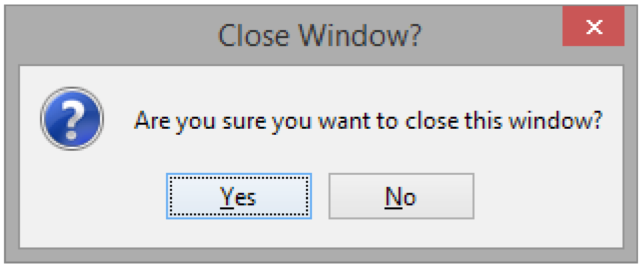
```java
JFrame weekEvents = new JFrame("Selected Week's Events");
Container container = weekEvents.getContentPane();
container.setLayout(new GridBagLayout());
container.add(button);
weekEvents.setSize(430, 300);
weekEvents.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
weekEvents.setResizable(false);
weekEvents.setVisible(true);
JLabel dates = new JLabel("<html><b>Planned Sessions<br/>yyyy/mm/dd | Cancelled | Campaign</b>");
// Find out which indexes are selected.
int minIndex = lsm.getMinSelectionIndex();
int maxIndex = lsm.getMaxSelectionIndex();
for (int i = minIndex; i <= maxIndex; i++) {
    if (lsm.isSelectedIndex(i)) {
        for(int k = 0; k < 7; k++)
        {
            for(Event event : plannedEvents)
            {
                if(mtblCalendar.getValueAt(i, k) != null)
                {
                    if((Integer)mtblCalendar.getValueAt(i, k) == event.getDay() && currentMonth == event.getMonth() -1 && currentYear == event.getYear())
                    {
                        dates.setText(dates.getText() + "<br/>" + String.valueOf(event.getYear()) + "/" + String.valueOf(event.getMonth()) + "/" + String.valueOf(event.getDay()) +
                        " | " + String.valueOf(event.getCancelled()) + " | " + String.valueOf(event.getGroup()) );
                        System.out.println(String.valueOf(event.getYear()) + "/" + String.valueOf(event.getMonth()) + "/" + String.valueOf(event.getDay()) +
                        " | " + String.valueOf(event.getCancelled()) + " | " + String.valueOf(event.getGroup()));
                    }
                }

            }
        }
```

The window for week events also has a button that allows users to add new events to the calendar. It does this by opening a window in the user's browser and sending them to the spread sheet which they can then edit directly by using a URI object.

```java
URI uri;
try {
    uri = new URI(
            "https://docs.google.com/spreadsheets/d/1PGZGN5IwYh2MNbfn3TwrPHXcbxTQ9U20r-eN8Qxz8no/edit#gid=0");
            class OpenUrlAction implements ActionListener
            {
                @Override public void actionPerformed(ActionEvent e)
                {
                    open(uri);
                }
            }
            JButton button = new JButton();
            button.setText("<HTML>Click the <FONT color=\"#000099\"><U>link</U></FONT>"
                + " to add session dates.</HTML>");
                button.setHorizontalAlignment(SwingConstants.LEFT);
                button.setVerticalAlignment(SwingConstants.TOP);
            button.setBorderPainted(false);
            button.setOpaque(false);
            button.setBackground(Color.WHITE);
            button.setToolTipText(uri.toString());
            button.addActionListener(new OpenUrlAction());
```

Lastly, the application can detect when the window is closed using the WindowAdapter object and overriding the windowClosing method. It asks the user for confirmation when closing the application.

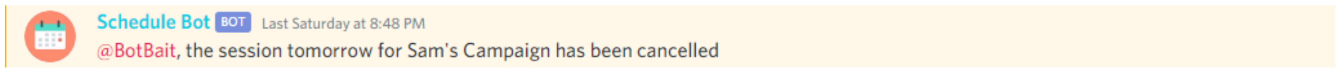

Close Window?

Are you sure you want to close this window?

If the user clicks no, the application resumes. If they click yes, the application sends out notifications, and then closes. This is because the user will have made all the changes they wanted and so notifications can be sent. The application checks all the events in plannedEvents and for each one scheduled on the next day, it sends a notification to Discord using the JDA instance. It uses a specific ID provided by Discord to notify the correct members of the group, and it sends a different message depending on if the event has been cancelled or not.

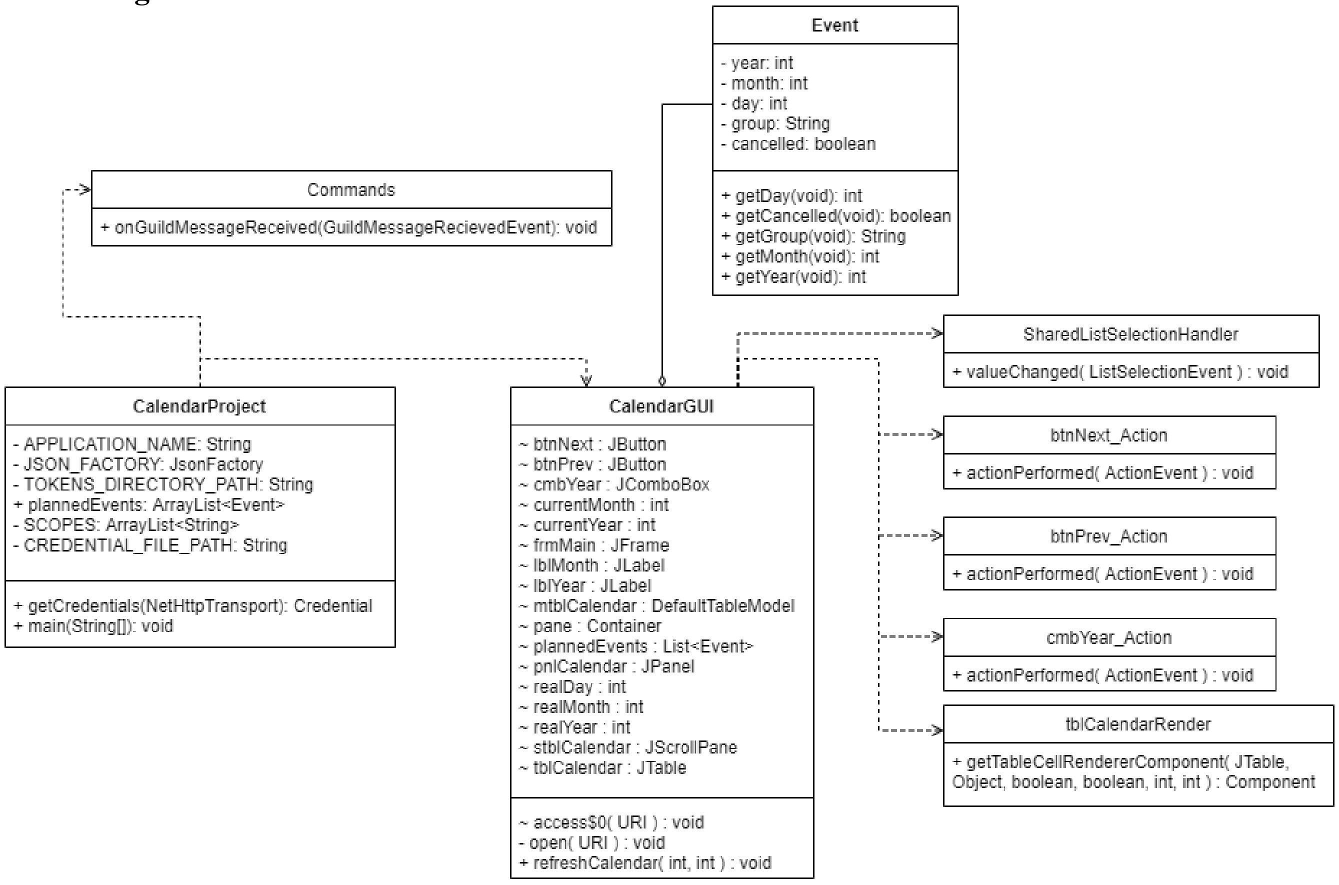
```java
case "SamCampaign":
    announcements.sendMessage("<@&627462212490887178>, the session tomorrow for Sam's Campaign has been cancelled").complete();
    break;
```

The message then appears in Discord.


Schedule Bot BOT Last Saturday at 8:48 PM

@BotBait, the session tomorrow for Sam's Campaign has been cancelled

Class Diagram


Event
- year: int
- month: int
- day: int
- group: String
- cancelled: boolean
+ getDay(void): int
+ getCancelled(void): boolean
+ getGroup(void): String
+ getMonth(void): int
+ getYear(void): int
Commands
+ onGuildMessageReceived(GuildMessageRecievedEvent): void
SharedListSelectionHandler
+ valueChanged(ListSelectionEvent) : void
CalendarProject
- APPLICATION_NAME: String
- JSON_FACTORY: JsonFactory
- TOKENS_DIRECTORY_PATH: String
+ plannedEvents: ArrayList<Event>
- SCOPES: ArrayList<String>
- CREDENTIAL_FILE_PATH: String
+ getCredentials(NetHttpTransport): Credential
+ main(String[]): void
CalendarGUI
~ btnNext : JButton
~ btnPrev : JButton
~ cmbYear : JComboBox
~ currentMonth : int
~ currentYear : int
~ frmMain : JFrame
~ lblMonth : JLabel
~ lblYear : JLabel
~ mtblCalendar : DefaultTableModel
~ pane : Container
~ plannedEvents : List<Event>
~ pnlCalendar : JPanel
~ realDay : int
~ realMonth : int
~ realYear : int
~ stblCalendar : JScrollPane
~ tblCalendar : JTable
~ access$0(URI) : void
- open(URI) : void
+ refreshCalendar(int, int) : void
btnNext_Action
+ actionPerformed(ActionEvent) : void
btnPrev_Action
+ actionPerformed(ActionEvent) : void
cmbYear_Action
+ actionPerformed(ActionEvent) : void
tblCalendarRender
+ getTableCellRendererComponent(JTable, Object, boolean, boolean, int, int) : Component

Word Count: 866