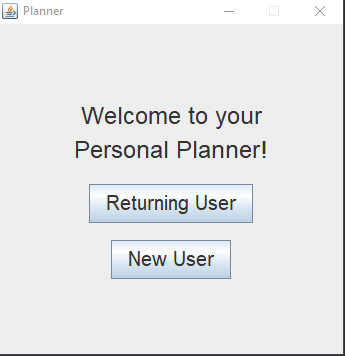
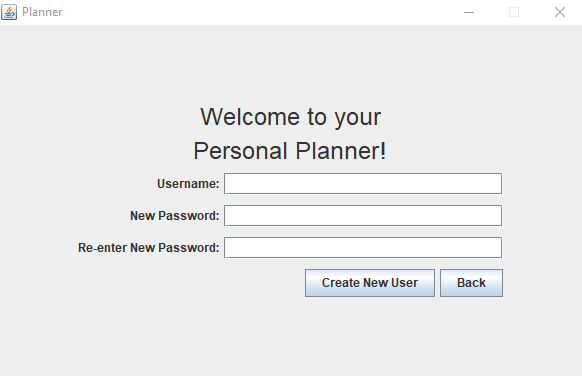
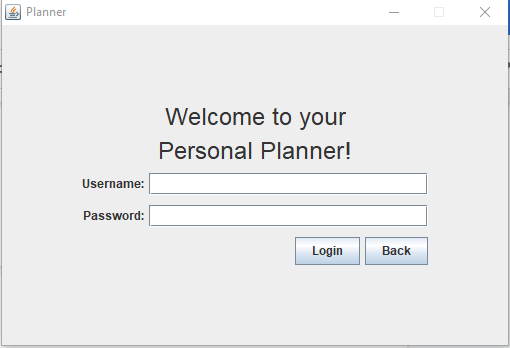
# Criterion C: Development

## Start Screen (Start.java)



For the Starting screen, the program gets 2 inputs from buttons *Returning User* and *New User*, which are held in the local variable *buttonPanel*. When either one is clicked, the *buttonPanel* is replaced with a new JPanel that is either *login1* or *login2*, which makes the frame become:



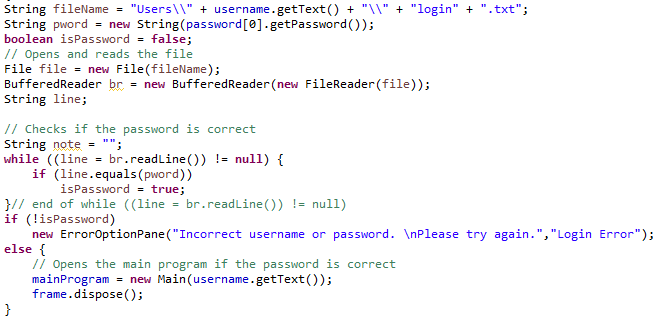
Created by login2 Created by login1

## Login.java



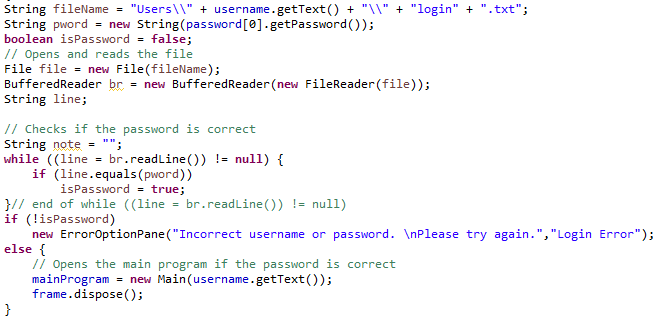
The constructor for the Login class has paramaters int logType, ActionListener a and JFrame frame. The parameter logType tells the class which if it needs to make a login panel for a returning user of a new user. The parameter *a* will be discussed later on.

Login for Returning User (variable *login1*):



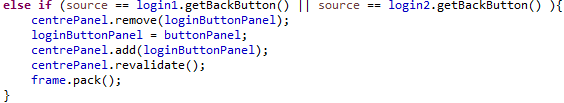
When the “Login” button is clicked, the program opens the file which contains the login information to see if the input matches the information. If it matches, the *Start* frame closes and the *Main* frame, which is the main program, opens, otherwise an error message pops up.

Login for New User (variable *login2*):



When the “Create New User” button is clicked, the program checks if the password is valid. If it is a file for the login information is created, and the *Start* frame closes and the *Main* frame opens. If it isn’t, an error message pops up.

Login class *back* button



*Code for back button from Start.java*

The back button uses an ActionListener created in Start.java, which is why there is a parameter ActionListener *a* in the Login constructor. This allows the program to swap between the Start screen and the login screens. The reason I made the *back* button in the Login class rather than the start class is because it makes the user-interface look nicer.

## ActionFrame.java

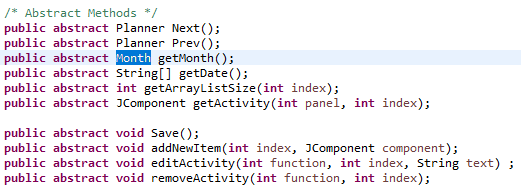
  
  
  
The ActionFrame has 3 constructors. The 1st constructor is for adding items, the second is for editing/removing items from the agenda specifically and the 3rd is for editing/removing items from the calendar, and make frames accordingly. It is also assigned a number tells the Main class what type of panel the component is from.

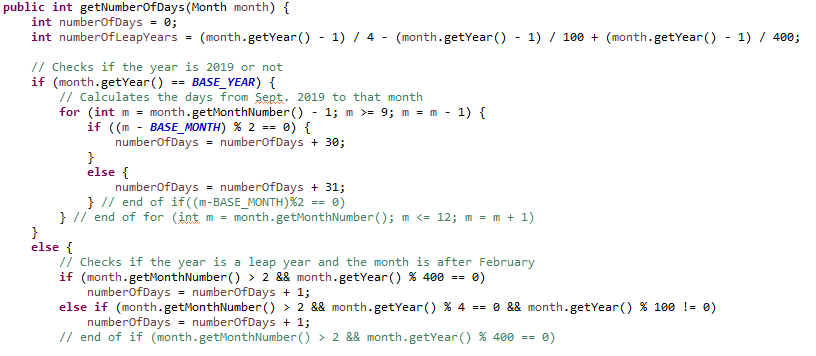
## ToDoCheckBox.java / ImpDateLabel.java / AddItemButton.java

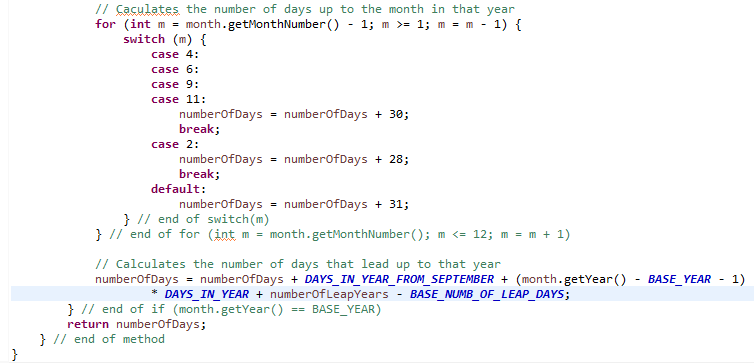
These are all custom JComponents that I made, which are assigned a number that tells the Main class what type of panel and what type of planner the component is from.

## Planner.java

Planner is an abstract class and is the parent class of the *Calendar* and *Agenda* class. I made this superclass to create a structure in my classes, as well as allowing me to write less code. Any Planner subclass have the following methods:



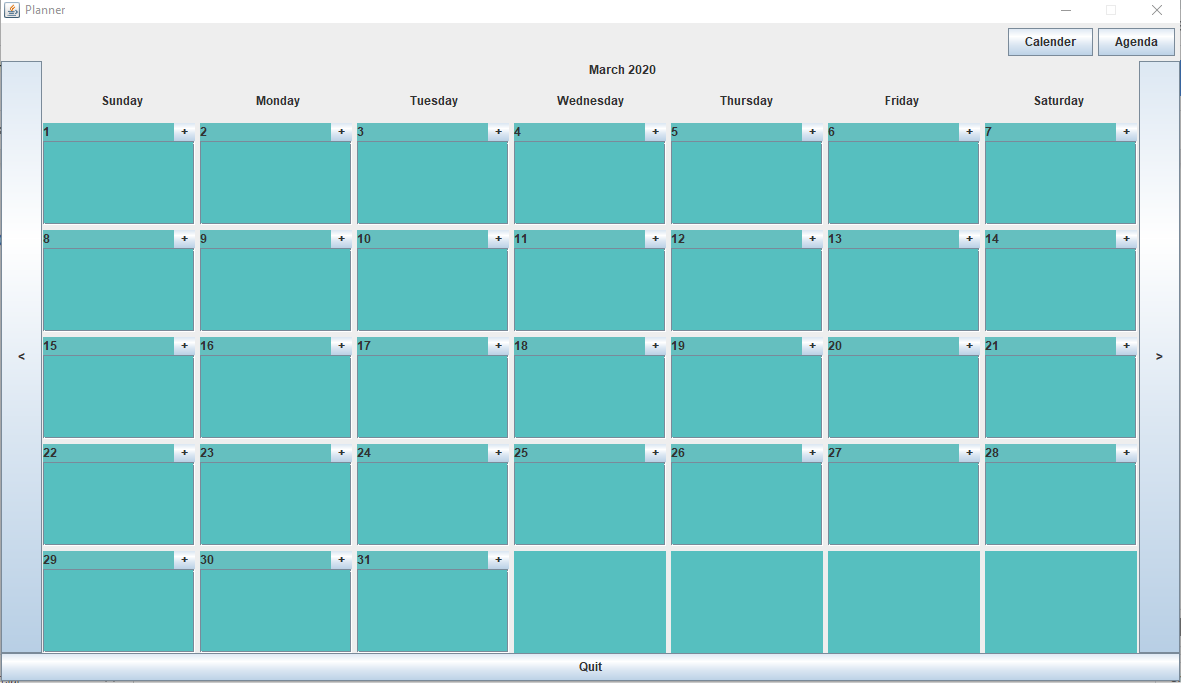
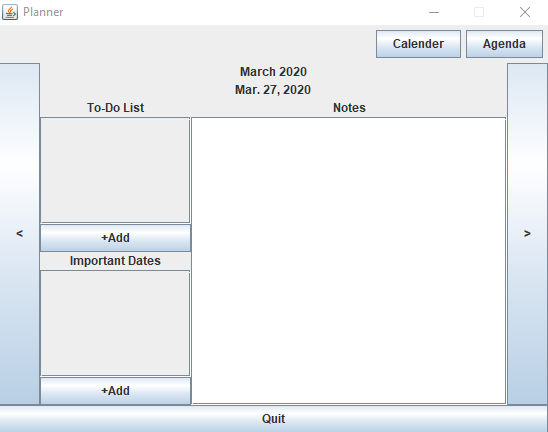




This method calculates the number of number days from September 1, 2019 to the first day of the given month. I chose that month because September 1st is on a Sunday, which is the first column of a calendar, making it easier to calculate. September 2019 has passed already so it is unlikely the user would use the planner for that month. This is why the Calendar class only works for months before September 2019. The method first checks if the month is in the year 2019.

It then calculates the number of days from the beginning of the year to the given month. It adds this number to the days from September 1st, 2019 to December 31st, 2019. By multiplying 365 to the difference between the year and 2019, I can find the number of days between those years. I find the number of leap years between those two years and add it to the sum, which will give me the number of day.

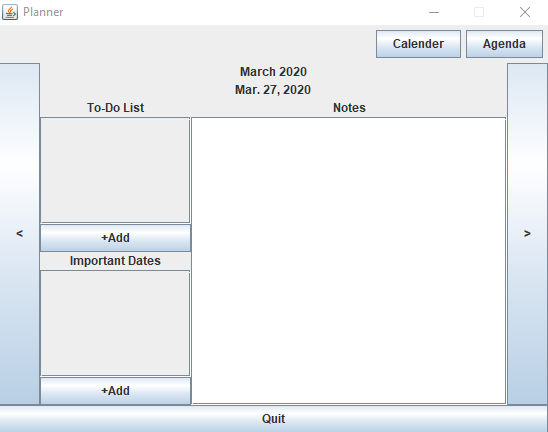
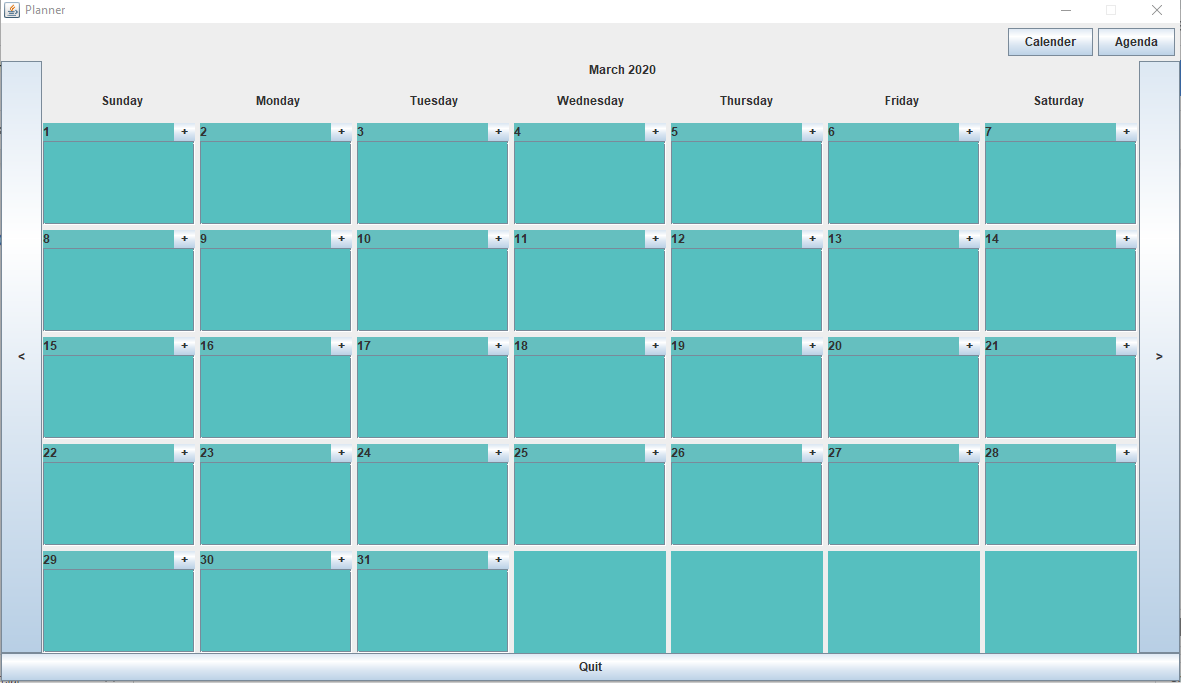
## DayPanel.java / FunctionPanel.java

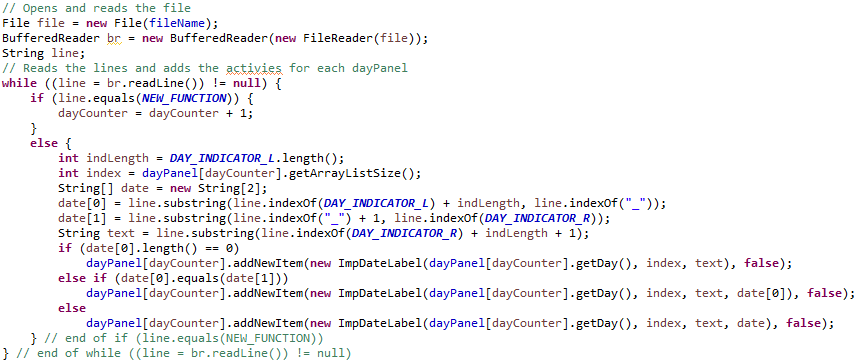
Day Panel Function Panel

The day panels are for the Calendar and they are assigned a number for the day the panel is for. While the function panels for the agenda.

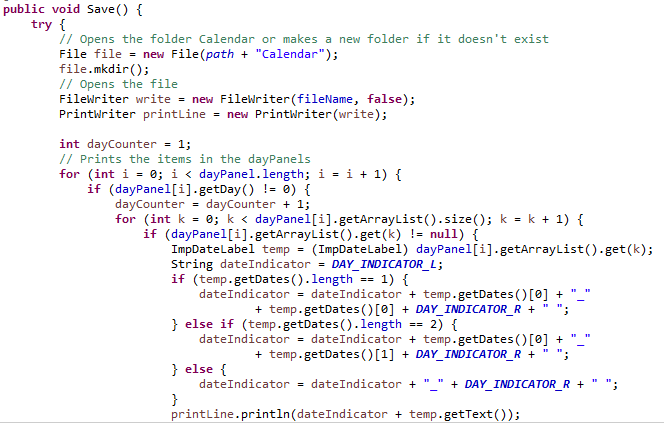
## Calendar.java / Agenda.java

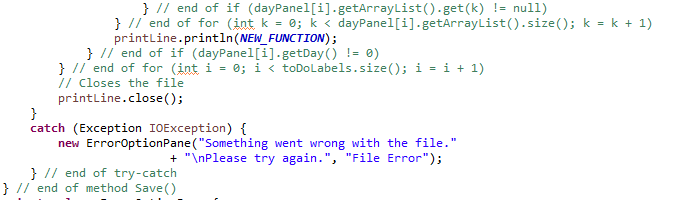
They both extend abstract class Planner.

Calendar Panel Agenda Panel



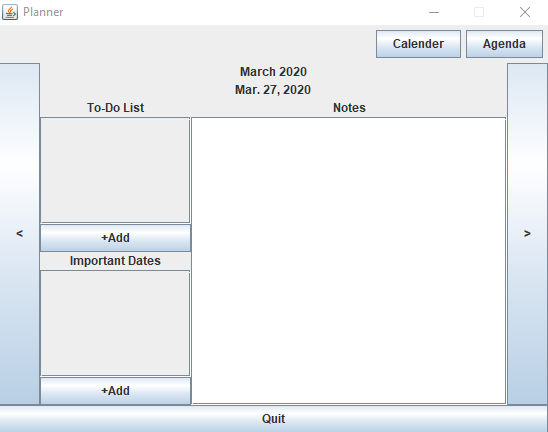
This algorithm is used to initialize the labels inside each day panel. It opens the respective text file of the Calendar. It reads through the file and add the activities to the proper dayPanel. When the line is a NEW\_FUNCTION it tells the program that it’s a new day panel. The text in between the DAY\_INDICATORs are the dates for the ImpDatePanels, which tell what dates that activity is in for agendas. The agenda uses a similar algorithm to create the labels in its function panels.





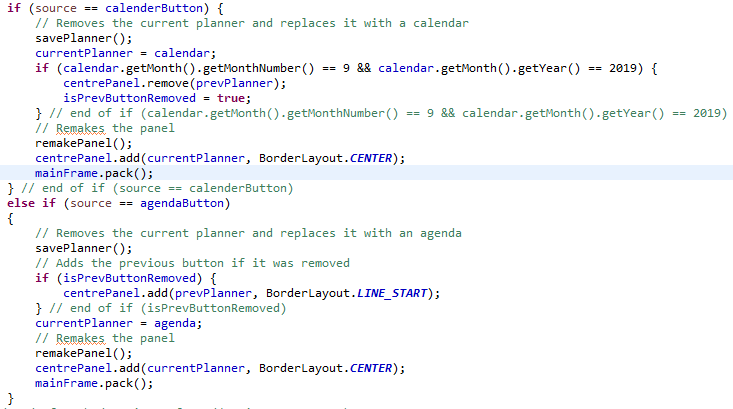
This function saves all the text on the ImpDateLabels of each day panel into a text file for that Calendar’s month. It opens the respective text file, or creates one if it doesn’t exist. It then goes through each day panel to print the data of each ImpDateLabel. It then prints NEW\_FUNCTION when it moves on to the next day panel. The agenda uses a similar algorithm for its Save() method.

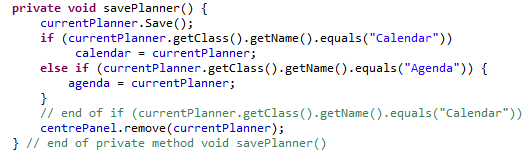
## Main screen (Main.java)



After logging in, the program opens the Main screen. The Main class planner that is being showed currently is a local *Planner* variable called *currentPlanner*.

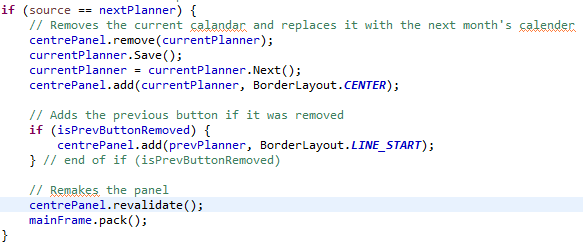
**“Calendar” Button and “Agenda” Button**

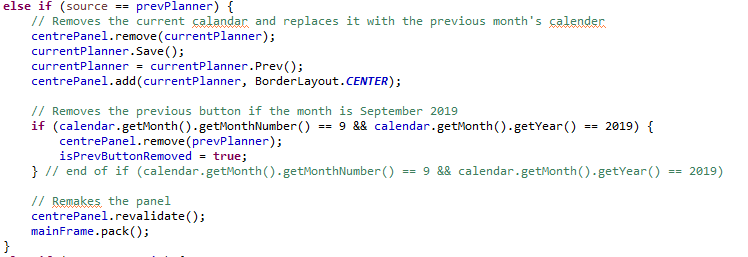




When the “Calendar” button or “Agenda” button is clicked, it switches the currentPlanner from a calendar to an agenda or vice versa. The savePlanner() method calls method Save(), will be explained later, and then saves the currentPlanner into either the local variable calendar or agenda, which allows the currentPlanner to know the date of the calendar or agenda before it was swapped when it swaps back.

**“<” Button and “>” Button**

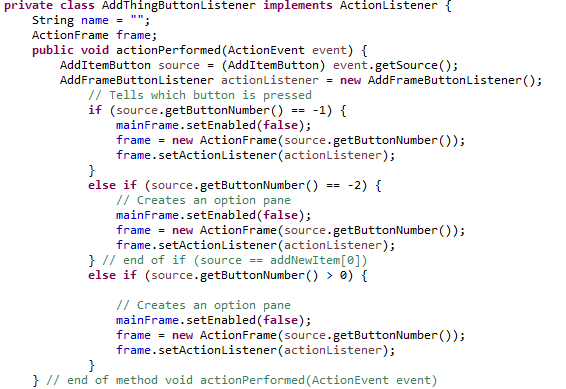




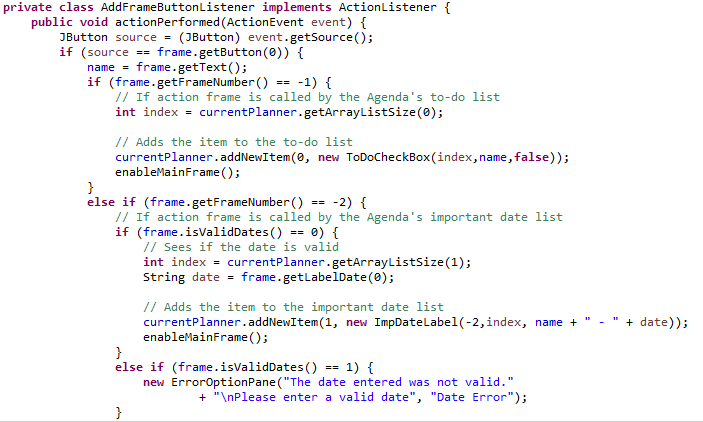
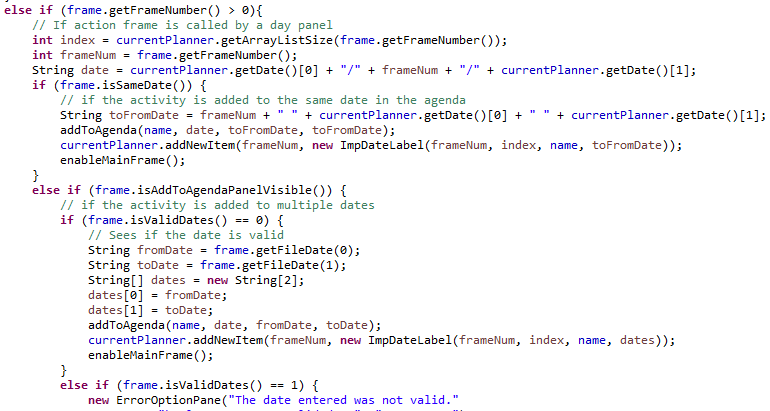
These buttons give the next and previous planners if the current planner. When clicked, the currentPlanner is first saved and then the currentPlanner becomes the previous or next planner from the Prev() and Next() methods. If currentPlanner is a calendar of September 2019, the frame removes the “<” button. This is because the *Calendar* class only works for any month after September 2019, which will be explained later.

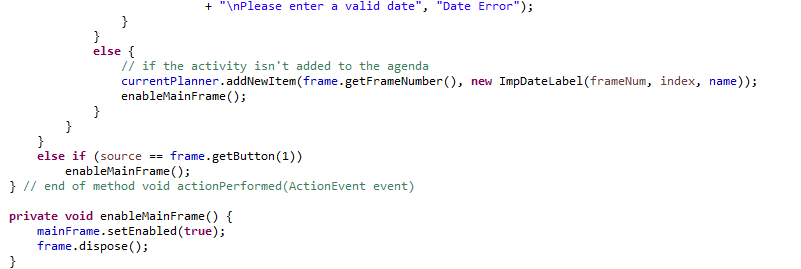
One of the most important functions of the Main class is that it manages all the functions (adding, editing or removing activities) of the planners.

**Adding Items**



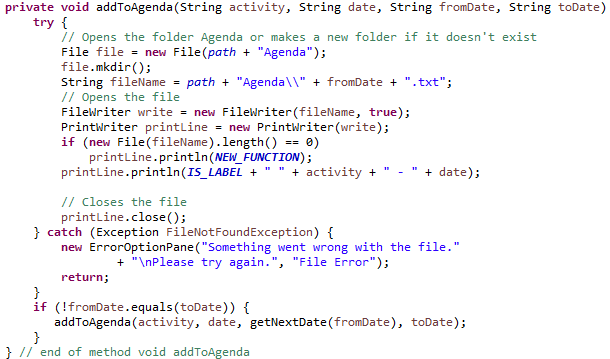
Since the *source* is an AddItemButton, it knows its assigned number, which tells the Main what panel is calling the method. It then creates an ActionFrame accordingly.





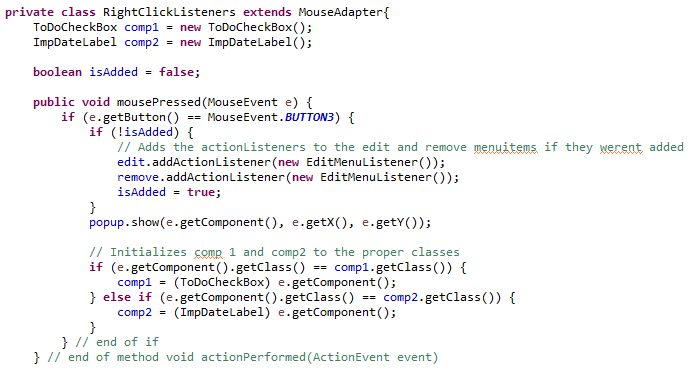
This ActionListener is for the buttons on the ActionFrame. Even though it looks complicated the function it does is simple.

tells the frame if it is the “add” button. If it is, it finds the ActionFrame’s frame number, which tells the Main which planner’s panel is calling the method. It then prompts for the user-input and adds the item accordingly. isValidDates() tells the program if the user entered a valid date or not (if it returns 0 it is a valid date).

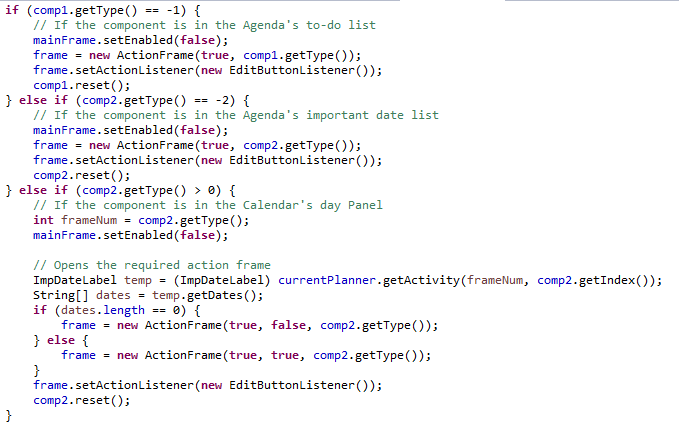


This method is only used by the calendar to add activities to multiple agendas. It recursively goes through the Agenda’s text files of the given dates and adds the String *activity + “ “ + date*. If the file doesn’t exist, it creates one and then adds it.

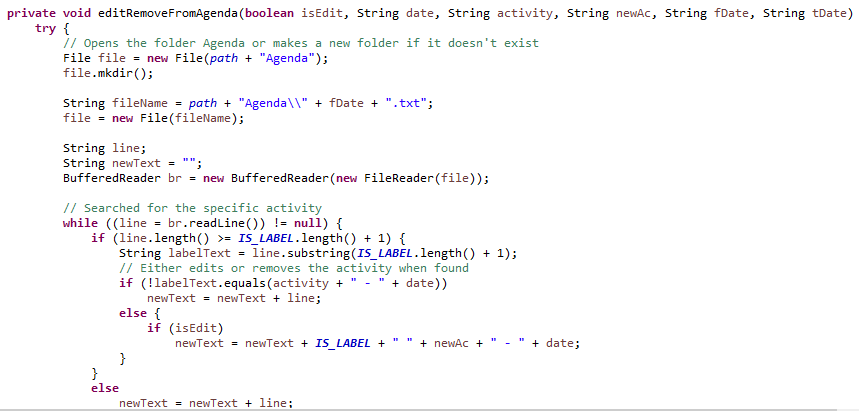
**Editing/ Removing Items**

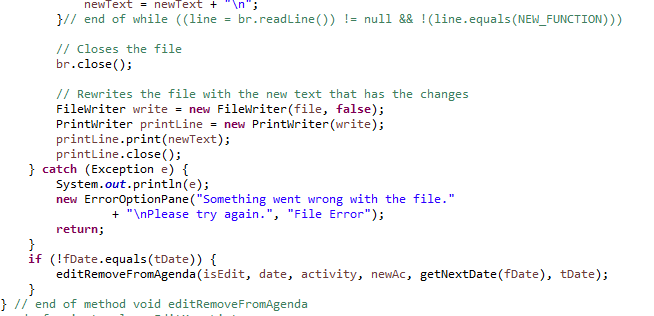


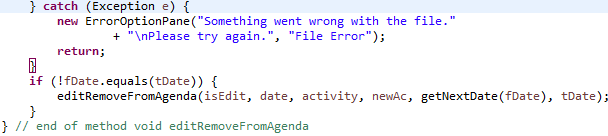
This method listens for the user right-clicking on a component to edit or remove it.



Since comp1 and comp2 are either a ToDoCheckBox or ImpDateLabel, they know their assigned number, which tells the Main what panel is calling the method. It then creates an ActionFrame accordingly. The ActionListeners for the ActionFrame are very similar as the ActionLister for adding.







This method is only used by the calendar to add activities to multiple agendas. It also recursively goes through the Agenda’s text files of the given dates. It copies the file into the String *newText* while searching the String *activity.* When found, it either adds the String *newAc* (to edit) or skips the line (to remove it). After the entire file is read, it replaces the text in the file with *newText*. I based this code off of a code found off of StackOverflow.

## Word Count: 1212

Bibliography

NarekNarek 32.7k6363 gold badges184184 silver badges337337 bronze badges, SingleShotSingleShot 16.8k1212 gold badges6161 silver badges9595 bronze badges, Adam BatkinAdam Batkin 43.5k77 gold badges115115 silver badges107107 bronze badges, ZezeZeze 17111 silver badge22 bronze badges, Tim M.Tim M. 44833 silver badges1515 bronze badges, seyed ali ziaeiseyed ali ziaei 1371010 bronze badges, … EnemyfishEnemyfish 1. (1959, July 1). Find a line in a file and remove it. Retrieved from https://stackoverflow.com/questions/1377279/find-a-line-in-a-file-and-remove-it