Project document for a calendar program

Personal information

Project name: Calendar, name: Allan Kiironen, Student Number 100036042, Degree program: computer science, year of studies: 2022, Date 25.4.2024

General description:

In this calendar you can mark up upcoming events and visualize them in weekly or daily view. You can set up an alarm to remind you of upcoming task, edit, delete and filter events. Additionally, the calendar supports exporting events to different calendars, as well as importing events.

User interface:

Running the program happens from the Weekly_view object. First view is the weekly view where it will display all the events that are that week. In the right corner there are 2 buttons that help you navigate between the weeks. In the top left corner, you can navigate between the daily and weekly view tabs.

By right clicking the screen either inside the weekly view tab or daily view tab, opens a popup menu where you can choose to add, edit, delete and filter events. By clicking the add event program opens a dialog box where you can fill all the necessary fields. The only necessary fields are start date, start time, end date and end time. Start Date and end date are prefilled as current day date. You can change them by clicking the field and choosing the date from date picker. Filling the time in correct format is necessary for the program to work correctly. The format is "HH:mm:ss".Once you have filled out the necessary fields, program lets you submit the event. Other fields such as Event name, description, category, color of the event, alarm date and alarm time are optional fields. The specific event name will be shown in the weekly view, daily view, edit even dialog and delete event dialog. The program has a colour field that lets you colour the event. The colour will be shown in the weekly and daily view. These fields make it easier to differentiate events from each other. Description field is for describing the event. Description will be shown once you hover with mouse on the event that has been placed in weekly view. Category field is for filtering event. Alarm Date and alarm time are for making an alarm for the event. Alarm will go off at the specified time and date. Once it goes off an alert window will pop up reminding you of the event.

By clicking edit events, opens another dialog. On the left there is a list view box where all the events are stored. The number sequence is a combination of date and time and if the event has a name it will be displayed after the number sequence. On right of the dialog there is a option to filter the events by categories. Checkboxes represent categories which you can check and press filter. The filter button will refresh and filter the list view by checked categories. In list view you can click on the event and press edit which will open the same looking dialog as add event dialog except some of the fields are prefilled. Here you can edit the specific event details by changing the fields values and pressing the update button.

By clicking the delete button, opens a dialog where you can delete events by pressing the event and pressing the delete button. Filter events button filters the whole weekly and daily view by the chosen categories. Weekly and daily view will display only the events that have the chosen categories. To view all the events go back to filter events and uncheck all the categories.

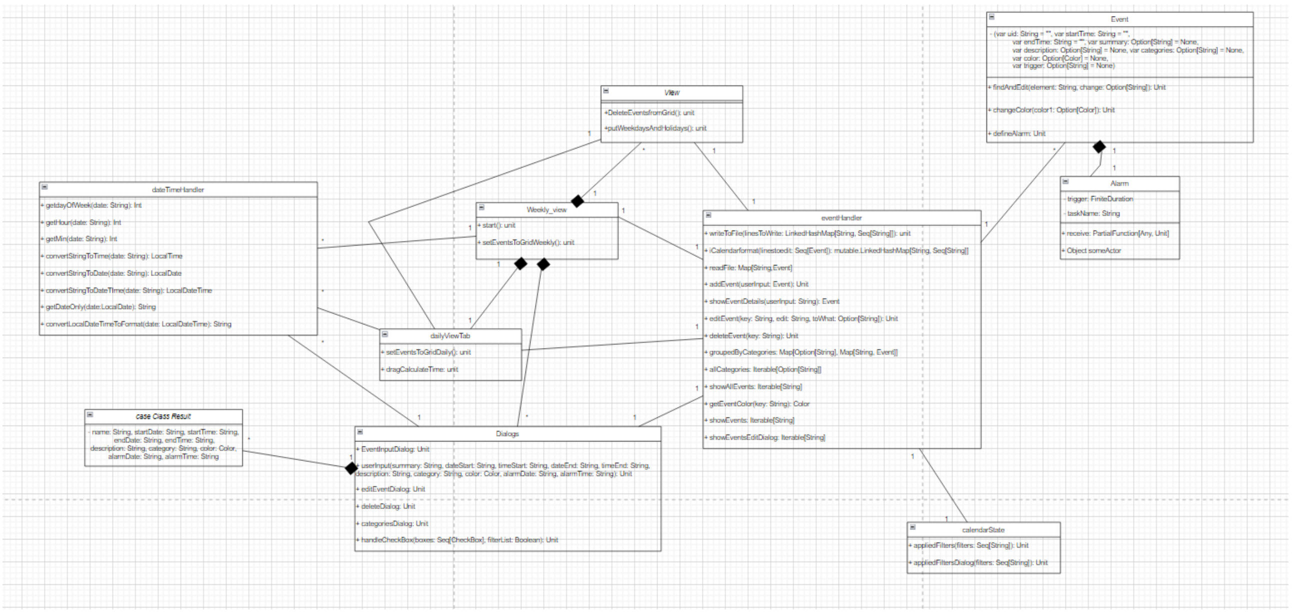
In daily view there are two buttons which help navigating between days. Here you can view same events but different view. In daily view you can reserve time slot by pressing left mouse button, dragging and releasing the button. Button press and release has to happen under the first row. If the mouse button click or release happen inside the first row, the system wont register the reserved time slot. Once system has successfully registered the time slot, it opens a add event dialog where the time and date slot are prefilled. Here you can fill out all the other fields and add the event to the calendar.

Program structure:

The program main parts are Weekly_view object and evenHandler object. Weekly_view takes care of the interface. Eventhandler reads the file, produces Event instances and keeps them. Important sub-parts are different views, dateTimeHandler and Event class.

Weekly_view takes care of general interface and weekly view reprentation. In the weekly_view you can access dailyViewTab and different dialogs. View takes care of deleting events from the grids and setting up week days on the grid. DailyViewTab as the name suggests takes care of daily view representation and its functionalities. Dialogs provide an interface for adding, deleting, editing, and filtering events. Dialogs have a helper case class Result, which helps to store the dialog fields values. DateTimeHandler helps to manage time related things.

EventHandler helps writing to a file, reading from a file and store Event instances. Event class stores details of an event and helps to manage and change the details.


Event
- (var uid: String = "", var startTime: String = "", var endTime: String = "", var summary: Option[String] = None, var description: Option[String] = None, var categories: Option[String] = None, var color: Option[Color] = None, var trigger: Option[String] = None)
+ findAndEdit(element: String, change: Option[String]): Unit
+ changeColor(color1: Option[Color]): Unit
+ defineAlarm: Unit
View
+DeleteEventsfromGrid(): unit
+putWeekdaysAndHolidays(): unit
Alarm
- trigger: FiniteDuration
- taskName: String
+ receive: PartialFunction[Any, Unit]
+ Object someActor
dateTimeHandler
+ getdayOfWeek(date: String): Int
+ getHour(date: String): Int
+ getMin(date: String): Int
+ convertStringToTime(date: String): LocalTime
+ convertStringToDate(date: String): LocalDate
+ convertStringToDateTIme(date: String): LocalDateTime
+ getDateOnly(date:LocalDate): String
+ convertLocalDateTimeToFormat(date: LocalDateTime): String
Weekly_view
+ start(): unit
+ setEventsToGridWeekly(): unit
eventHandler
+ writeToFile(linesToWrite: LinkedHashMap[String, Seq[String]]): unit
+ iCalendarformat(linestoedit: Seq[Event]): mutable.LinkedHashMap[String, Seq[String]]
+ readFile: Map[String,Event]
+ addEvent(userInput: Event): Unit
+ showEventDetails(userInput: String): Event
+ editEvent(key: String, edit: String, toWhat: Option[String]): Unit
+ deleteEvent(key: String): Unit
+ groupedByCategories: Map[Option[String], Map[String, Event]]
+ allCategories: Iterable[Option[String]]
+ showAllEvents: Iterable[String]
+ getEventColor(key: String): Color
+ showEvents: Iterable[String]
+ showEventsEditDialog: Iterable[String]
dailyViewTab
+ setEventsToGridDaily(): unit
+ dragCalculateTime: unit
case Class Result
- name: String, startDate: String, startTime: String, endDate: String, endTime: String, description: String, category: String, color: Color, alarmDate: String, alarmTime: String
Dialogs
+ EventInputDialog: Unit
+ userInput(summary: String, dateStart: String, timeStart: String, dateEnd: String, timeEnd: String, description: String, category: String, color: Color, alarmDate: String, alarmTime: String): Unit
+ editEventDialog: Unit
+ deleteDialog: Unit
+ categoriesDialog: Unit
+ handleCheckBox(boxes: Seq[CheckBox], filterList: Boolean): Unit
calendarState
+ appliedFilters(filters: Seq[String]): Unit
+ appliedFiltersDialog(filters: Seq[String]): Unit

During the implementation class structure changed a lot making it more complicated that it was initially planned. The cleverer solution would have been to make View handle interface instead of Weekly_view as initially planned.

Algorithms:

The calendar app doesn't have algorithms that are overly complicated. The Gregorian calendar algorithm that got introduced in the original plan was not used because of the already existing library that took care of that problem. The program got a lot simple because of that.

Programs most important and complex algorithm is setting events to the grid. The problem was calculating the height of the event based on start and end times and placing the event in the grid correctly. Algorithm starts by calculating the height by converting the start time and end time to a ratio. Then program calculates the difference between start and end time ratio which gives proportion of the total grid that events should occupy. Afterwards height is calculated multiplying it by total height of the window. Once height is calculated the event offset is calculated by converting minutes into hour fraction and then multiplying it by height of one grid cell. This will ensure that the rectangle that represents the event will have correct height and will be placed correctly into the grid.

There are two variations of this algorithm. One is used to se the events in weekly view grid and another in daily view grid. The only difference in these variations is that weekly view one check if the start and end date fall within the current week. Daily view checks if the start and end date are equal to the current day.

Data structures:

The program used mostly mutable Maps to track Events. The event starting time served as a key and a value was an Event class. Event class had all the event variables such as starting time, end time, description and so on. Some of these values are in the Option wrap so that it is easier to handle optional event values.

The maps are first formed by the readFile function which reads the file and converts the data into Maps. Other functions such as add, edit and delete can modify this data. There are other helper functions such as for example getEventName or getEventEndtTime that can help access the event data in the Map.

Additional separate data structure is string sequences. This data structure is meant for grouping all the events together and converting these events into correct format. Afterwards this sequence is converted to Linkedhasmap and passed to the writeTofile function. This change in data structure is a bit clunky and it could have been implemented more simply. Possibly more clear solution would have been just to keep the string sequence data structure and pass it as such to the writeToFile function.

Firstly I implemented Map data structure that had start date as key and string sequence as their value. The string sequence had event details as its elements. These elements had to be in specific order which did not make the program very scalable. Managing the event details went harder as I started implementing more features. Finally, the Map changed so that the values were Event classes instead of string sequences. This made accessing the Event details easier which turned out to be a better solution.

Files and internet access:

Program has its own file for storing events, allowing users to save and load their schedules. The file that everything gets written to is eventInfo.ics. Even though the file is in ICS format it acts as a txt file. The format is made so that the exporting and importing events from different calendars are easier. The program doesn’t have any reliance on internet connectivity.

The file format looks as follows:

```
BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//My Calendar Program//Example Corp//EN
CALSCALE:GREGORIAN

BEGIN:VEVENT
UID:d4cd80b7-042b-4395-be5d-4c7724a1e170
DTSTART:20240425T100000
DTEND:20240425T120000
SUMMARY:testName
DESCRIPTION:testDescription
CATEGORIES:testCategory
Color:[SFX]0x008080ff
BEGIN:VALARM
ACTION:AUDIO
TRIGGER:20240425T190000
END:VALARM
END:VEVENT
END:VCALENDAR
```

All the red fields are necessary for the app to register the event. White text is optional. The format starts with the BEGIN:VCALENDAR, VERSION:2.0, PRODID:-//My Calendar Program//Example Corp//EN and CALSCALE:GREGORIAN these fields are for exporting purposes only, they don’t have any impact on my calendar app. Each event starts with BEGIN:VEVENT and ENDS with END:VEVENT. Afterwards comes UID which is unique for every event. DTSTART and DTEND describes event start and end time. Summary is for giving the event name, description is a field to add more information for this event. Categories field is for organizing and filtering the events. Color can be used for coloring the event. For alarm setting up an alarm, trigger field is more important. Trigger field takes start time as its parameter. Begin:VALARM, ACTION:AUDIO and END:VALARM is for exporting purposes. After all the event have been recorded the last line of the file is END:VCALENDAR which is there for exporting purpose.

Testing:

Program testing mainly happened in the interface and in the Test class. Test class tested all the eventHandler and dateTimeHanlder object methods. Through eventHandler the Event class got tested also. All the interface related problems were tested using interface itself. The code used set of main functions to test rather than traditional unit testing. Although unit tests could be clearer way to test the backend with assertions and using more automation, most of the tests happened using interface itself. For that I did not find that much use in unit testing and opted out for using print statements and main functions to test the backend. In terms of scalability traditional unit testing could be more efficient way to test the program.

Most of the testing happened during the interface implementation. The bugs were visual, so it was easy to notice bugs. The test scenarios came up as the implementation of the program progressed. Ui test examples were that what happens if the end time is before start time or if the program lasts over a week or a day. Edit dialog was not tested properly and some of the bugs could occur through that.

The testing phase was way less structured compared to what was planned. Many test scenarios came up as the project progressed.

Known bugs and missing features:

Bugs:

Event color, category and alarm don't get exported to google calendar. This bug could be fixed by analysing the exact format Google calendar is using. Although this fix doesn't guarantee that it will work on other calendars such as Microsoft calendars.

Events get on top of each other if they are occupying the same time slot. Fixing this issue would require some sort of algorithm that spots event overlaps. One way to implement this algorithm could be that if start time, end time or both are inside the already occupied time slot, the grid cell makes space for second event.

If you close tabs there is no way to get them back. You must restart the application. One way to fix this is to prevent tabs from closing. There is a probably method in Scalafx that does that.

Missing features:

Small amount of public holidays are displayed.

Weekly view has been implemented differently (in columns there are 24 hours of the day and rows represent each week day). Fixing this requires to changing the whole weekly view layout.

In weekly view you cannot reserve suitable time by painting with the mouse. This could be implemented by tweaking the algorithm used in daily view by taking into consideration the x coordinate.

Cannot see visually what you are painting in daily view. This could be implemented by placing for example rectangle on the point where you started painting the time slots and then the rectangle changes its height by following the mouse.

Grid cell size doesn't scale with the window size. To make the grid cells scalable with window size, I could make a program to detect when the window size is changed and then update the grid cell constraints accordingly.

3 best sides and 3 weaknesses:

Weaknesses:

Program interface is not visually pleasing and not so user-friendly. The interface could make time slots interactive, for example time slots and events could be clickable.

Deviations from the plan, realized process and schedule:

Since I was retaking this course some of the work has been already done last year such as implementing data structured for storing calendar events, writing and reading file functionalities and some of the graphical interface. The reason why I could not finish the project was that I did not dedicated enough time. Here I also learned that planning the data structure from beginning I very important. Bad data structures made implementing new features a lot harder.

I started working from week three. There I mostly spent time mostly refactoring the whole code base to make the code more scalable. Then I started to implement basic functionalities in the graphical interface. Refactoring part took unexpectedly a lot of time which made me at the end fall out of phase as planned. I had to rewrite a lot of code which was time consuming.

On week five I started to implement all the basic features so that all the easy level functionalities would been done. Roughly at the week 6 these functionalities were implemented on the interface. Here I found some problems calculating the event height and placing them correctly in the grid. It took me some time to figure out why grid cells where stretching making the events show incorrectly. This deviated me from the plane further. Once this feature was implemented correctly I decided to make project on medium difficulty instead of demanding difficulty.

Last three weeks I used for debugging and implementing the medium functionalities. Medium functionalities were quite trivial once the interface was made with the easy functionalities.

The time planning was mostly overly optimistic. I did not make into account other schoolwork which made gave me less time to spent on the project.

The order in which I chose to implement the project I think was generally good. One thing that could have been done differently was to first implement easy functionalities without going straight to implementing the interface. This approach would have made the project submittable even if I did not have time to implement the medium functionalities. However, this approach would have been overall more time consuming since I would have had to implement a text view separately.

During the process I learned to read documentations. During the implementation it got a lot easier to understand documentations and faster to implement methods. Also planning is very important with this project. That way you can get away from a lot of work.

Final evaluation:

Some of the programs strong aspects are: The program manages events, provides various views, allows for event manipulation, and supports event filtering and alarm functionalities. The program successfully reads from and writes to files. Additionally program successfully sets events in the grid.

Some of the programs weak aspects are: Interface lacks visual appeal and could benefit from improvements. Known bugs such as events overlapping in the grid or tabs being closed without ability to reopen them and class structure could be improved for better organization and clarity.

If I would start the project over I would allocate more time for planning and refactoring to avoid getting behind the phase. Also I would try to implement a more clear class hierarchy to make the code more maintainable and scalable.

References:

https://javadoc.io/doc/org.scalafx/scalafx_2.13/latest/index.html

https://www.scalafx.org/

https://www.youtube.com/playlist?list=PLLMXbkbDbVt9MIJ9DV4ps-_trOzWtphYO

https://www.geeksforgeeks.org/zellers-congruence-find-day-date/

https://devguide.calconnect.org/Data-Model/Simple-Event/

https://doc.akka.io/docs/akka/current/stream/stream-flows-and-basics.html

https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html

References and code written by someone else:

Appendices.