Ohjelmointistudio 2 Project - Calendar

Project document

Joanna Keränen 100674666
Information Networks, 2. year (IV22)

12.2.2024

General description

This program is a calendar. Events can be marked to the calendar, and these events contain the information of the name, start date, end date, start time, end time and category of the event. The default categories are school, work and hobbies but the user can also define new categories. The user can also define whether they want a reminder of an event. Events can also be deleted from the calendar. The calendar shows public holidays in Finland.

The program shows as a default a weekly view of the current week. This view contains the dates of the current week with the corresponding weekdays and the events are presented in columns under the correct days. There is also an additional column for reminders, in which an event is added in case the user specifies that they would like a reminder of that event. The view also shows the current month. The user can browse different weeks one week at a time.

In addition the calendar contains a daily view that contains all of the 24 hours of the day and shows the events marked in the correct time slots. The daily view also contains an all day section into which the events are added if the user has specified them to be all day events.

Events, categories and Holidays are saved into files and read from there. Events are stored in a standard format (iCalendar), which allows them to be transferred to other calendars.

This project was completed as an easy task. However, in addition to meeting the requirements of the easy task, the program has a graphical user interface, in which CSS is utilized (in the forward and backwards arrows) as well as font files from an external source. Using gridpane and percentage sizes for elements makes the gui scalable.

User interface

The program is launched by running the Main class. The program has a graphical user interface through which the user interacts with the program.

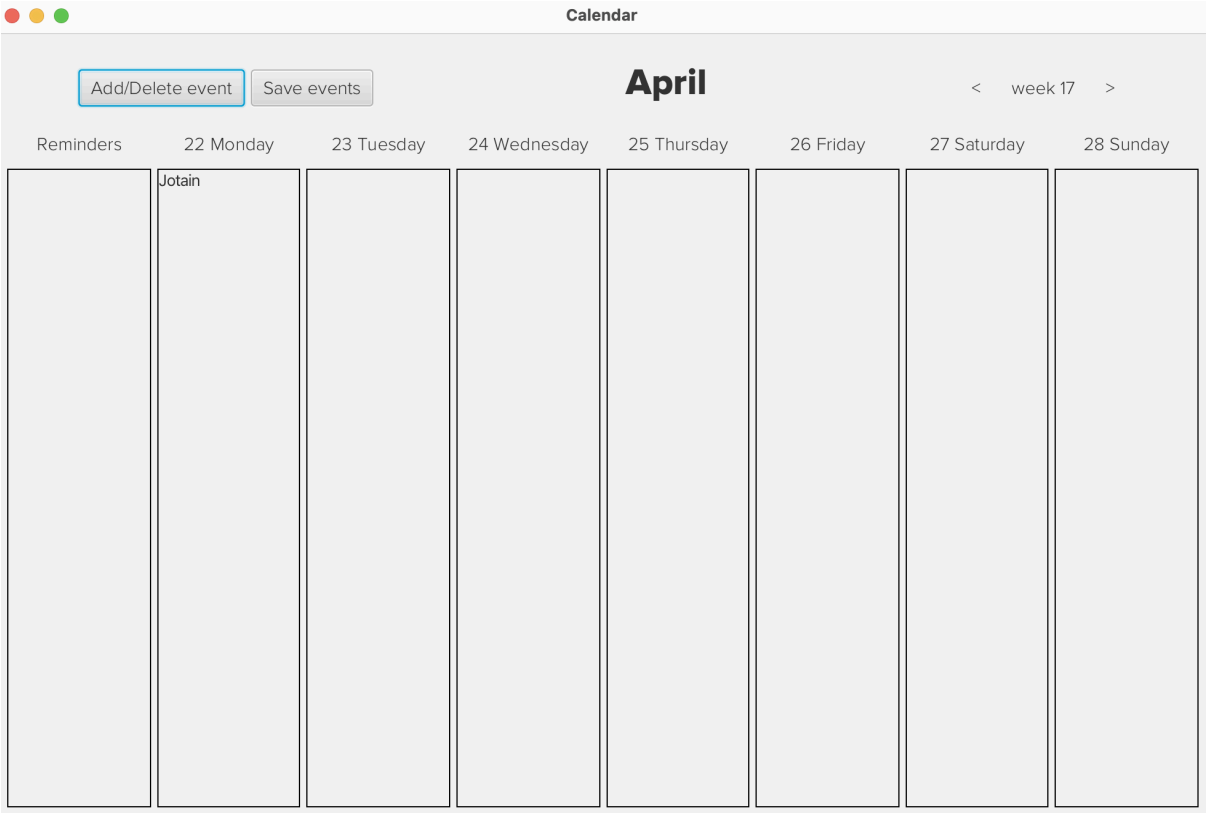


Figure 1: Weekly view of the calendar

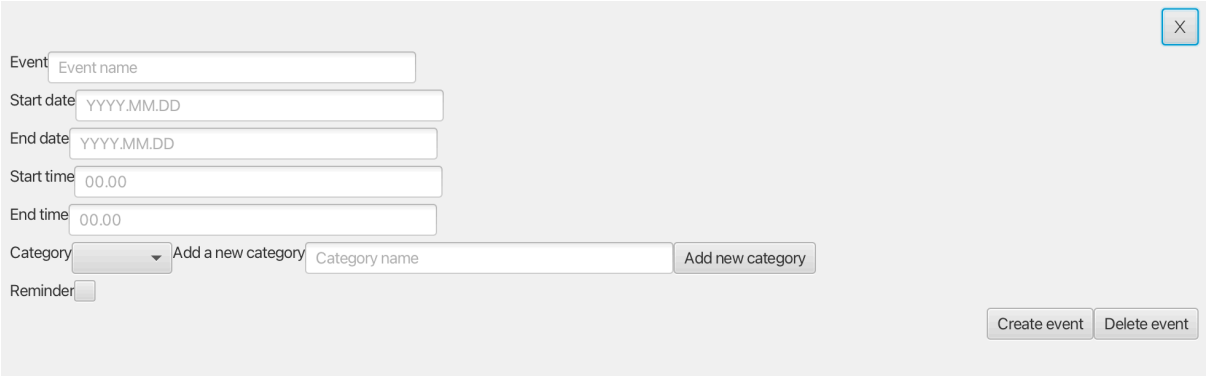


Figure 2: View for adding an event

The user can add events to the calendar by clicking the add/delete events button in the top left corner of the gui. Clicking this button changes the view to one which asks the user for input in text fields, a choice box and a check box, as presented in figure 2. The user inputs the name, start date, end date, start time and end time as string. The required formats for the dates (yyyy.MM.dd) and times (HH.mm) are presented as a prompt text in the corresponding text fields. If the user gives the input in an incorrect format, the program communicates this through an alert that prompts the

user to make sure that their input is in correct format. If the user wishes to add an all day event they need to write “ALLDAY” to the start and end time sections. One flaw in the program is that it doesn’t communicate this to the user in a clear way. The user selects the category from a choice box. The user can create a new category by writing the name of the new category to the add new category text field as string and then click on the add new category button. This creates the new category, also saving it to the categories file, and when opening the options of the choice box again, the new category can be chosen along with already existing categories. The user can click on the reminder checkbox, and by doing this, the event is added to the reminders column of the weekly view. The user can leave the checkbox unselected, and in this case, a reminder is not created from the event.

After filling out all of the input, the user clicks on the create event button in the bottom right corner. This switches the view back to the weekly view, and the name of the event is visible in the column for the correct day. For example in figure 1 “Jotain” is an event that is on monday 22.4.2024. The user can also delete events. This is done by clicking on the add/delete events button in the weekly view. The user then needs to input all of the information of the event, as written when adding the event, and then click on the delete event button. This deletes the event from the calendar and takes the user back to the weekly view.

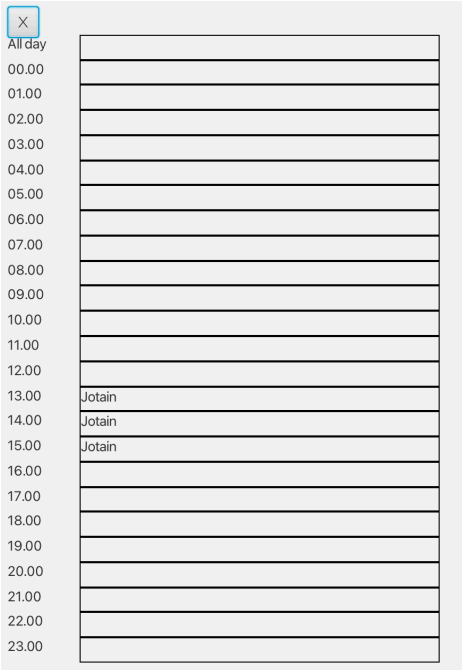


Figure 3: Daily view of the calendar

Clicking on the weekday labels in the weekly view, the daily view of the calendar is shown. This view shows all 24 hours of the day and the events are shown in their correct hourly slots. For example, in figure 3 the event “Jotain” on monday 22.4.2024 starts at 13.00 and ends at 15.00. The user can exit the daily view by clicking on the X button in the top left corner. This takes the user back to the weekly view. In the weekly view, the user can browse weeks forwards and backwards by clicking on the arrows around the week label. Clicking on these arrows, the dates change to the dates of the following or previous week, the week label changes to represent the correct week and the month changes when 4 or more days of the week belong to a different month.

Program structure

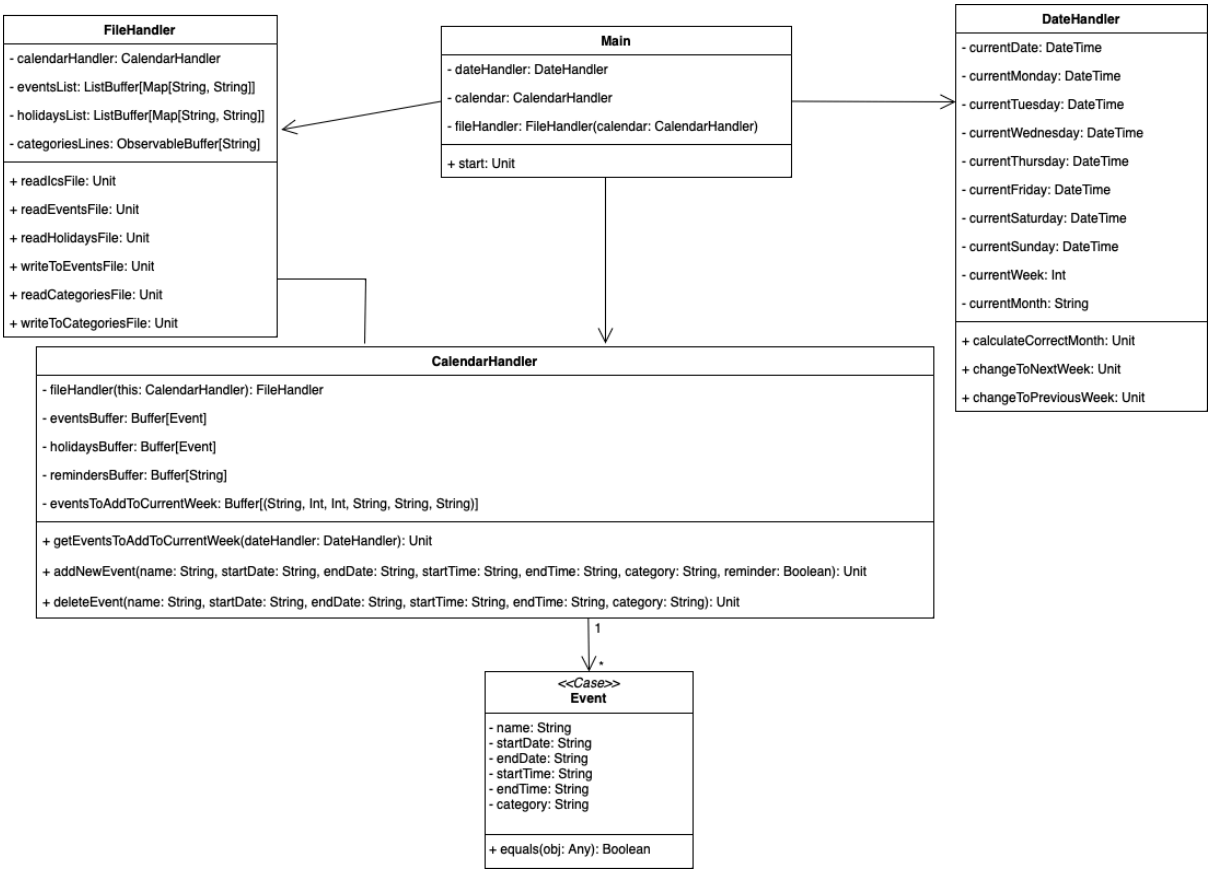


Figure 4: A UML class diagram from the final class structure of the program

The program has the classes Main, FileHandler, CalendarHandler, DateHandler and a case class Event. Main class describes the graphical user interface, FileHandler is responsible for reading and writing to all different files (events.ics, holidays.ics and categories.txt), DateHandler is responsible for retrieving the correct dates for the current week and has methods for updating the dates of the weekdays when weeks are browsed in the graphical user interface, calendarHandler is responsible for retrieving the events to add to the current week and adding and has methods for adding and deleting events. The program has separate files for events, holidays and categories.

Main only contains one function start(), and start contains the description of the user interface and has methods to update the user interface. Start contains methods to update the daily view, clear the daily view, update events to the weekly view, update the reminders to the weekly view, update the labels of the weekly view and clear the events boxes of the weekly view. These methods are called inside start. Whenever a view switches the view is cleared and updated each time. Methods from the classes FileHandler, DateHandler and CalendarHandler are called inside start.

The FileHandler class is responsible for reading and writing to files and it has separate methods for reading and writing to different files as represented in the UML class diagram. The methods are called in the Main class, for example writeToFile is called when the save events button is clicked.

The DateHandler class is responsible for retrieving the correct dates for the calendar using external libraries. DateHandler keeps track of the dates of the weekdays of the current week. The current week in this context means the week that the user is looking at in the graphical user interface. DateHandler also keeps track of the current week and current month that are also shown in the weekly view of the gui. DateHandler has methods to switch to the previous or the following week as well as to calculate the current month. These methods are called in Main when the arrows back and forward are clicked.

CalendarHandler contains the events, holidays and reminders in buffers as well as keeps track of the events that need to be added to the current week. CalendarHandler has methods to add and delete events. These add or remove the

events from eventsBuffer. CalendarHandler also has a method to get the events that need to be added to the current week. This method adds these events to a buffer. All of these methods are called in Main. The addNewEvent method takes as parameters the information of the event, and in Main the function is called with the users input given to the function as parameters.

The methods in the CalendarHandler class create instances of the case class Event. It is useful that events are represented as a case class because there is a method apply that automatically creates case-class entities without the need for a new operator, all constructor parameters are automatically val type variables meaning they can be read without additional methods, for example event.name, and the class automatically has an equals method. The equals method however was not useful in the implementation because it had to be overridden in order to compare the event's properties instead of the actual events. Two different instances of Event can have all the same values for parameters, in this case meaning that it is actually the same event, but the default implementation of equals will still compare them as different events because they are different instances of the case class. This is why the Event case class contains the method override equals(), which creates an equals method that compares the values of the parameters instead of the instances.

To make the communication between classes possible, Main has instances of DateHandler, FileHandler and CalendarHandler, and FileHandler takes as parameter the defined instance of CalendarHandler. CalendarHandler has an instance of FileHandler that takes as parameter the CalendarHandler itself and FileHandler has an instance of CalendarHandler. FileHandler takes calendarHandler as a parameter so that in the class Main the same instance of CalendarHandler is called everywhere instead of calling different instances of the class. This would cause the dates not to be updated correctly everywhere.

The UML diagram has changed quite drastically from the original plan. In the original UML diagram there were separate classes for daily and weekly views. In the final implementation the whole gui was created inside Main, however it might have been good to have these separate classes or even separate methods inside Main to create the different views.

Some classes from the original plan have been removed, such as Holiday, Category and recurring events. The recurring events were not added at all to the program, Holidays are represented as instances of the case class Event and Categories are stored in a file and directly read from there and updated when clicking the add new category button in the gui. The DateHandler class was added to the original plan because it was easier to store all the dates in a separate class instead of adding them to the CalendarHandler class.

Algorithms

This calendar project does not use much of its own algorithms. To retrieve the correct days of the week for dates and to calculate the dates of the next and previous weeks when browsing weeks one week at a time, this project uses nscala time and joda time libraries and their methods. For example, to calculate the days of the next week based on the currently visible week, the program uses the method plusDays (currentMonday.plusDays(7)). These libraries use some kind of algorithms in their implementation, but this calendar program only uses the methods of these libraries to do the calculations necessary. Another way to have done this would have been to create a new algorithm for calculating the correct days, but using the existing libraries is more efficient.

Data structures

This project uses Scala's readily available data structures.The program needs to store information about events, categories and reminders. These are stored in buffers. Buffer is a mutable data structure and can therefore be easily modified, which is essential in this calendar program. For example events and categories need to be added.

Another option that was considered was using lists. A list is an immutable data structure meaning that it cannot be modified after creation. Elements can however be added by creating a new list. Nevertheless, buffers meet all the needs for the collections in this program, and therefore they were used.

In the class FileHandler, events are read from the file and stored into a ListBuffer that has a map inside of it. ListBuffer allows adding, updating and removing elements efficiently, from which especially addition is needed in the FileHandler class. Another possibility would have been to use an ArrayBuffer, which allows elements to be accessed by index. However, for eventsList in FileHandler this is not required. The list is created in the FileHandler class and it is iterated over in the class CalendarHandler and iterating over elements is efficient with a ListBuffer. However, an ArrayBuffer would also have worked because even though adding and removing elements to the beginning of a list is not as easy with a list buffer, adding elements to the end is possible, which is what is required in this case.

The ListBuffer eventsList has a Map inside of it, where the keys are the names of the properties of an iCal file, for example DTSTART, and the values are the actual information, for example the start time of an event. The Map allows access to the values for certain property names, because it needs to be specified which property each value is representing. The structure of having the Maps inside a listbuffer allows to separate information of different events so that it is not all in a long list.

In the class CalendarHandler, the events are stored first into a Buffer containing Events and then this buffer is iterated over and the events of the current week are added to a buffer with elements in the form: (name: String, startWeekday: Int, endWeekday: Int, startTime: String, endTime: String, category: String). The events were added to the list in this format, because this information is needed to add the events to the graphical user interface. Category is unnecessary here, because it is not shown in the graphical user interface. However, it was added to the collection, because the plan was to show different colors for different categories, and in this case the information of the category would have been necessary.

This way of first putting the events into a ListBuffer with a Map inside of it, then iterating over this and creating a buffer with Events in it and then iterating over it and selecting the events to be added to the current week to another buffer in another format seems unnecessarily complicated. A better way would have been to create the Events straight from the information that was read from the file and selecting the events of the current week from this list. However while implementing the program, I

did it one step at a time and by trial and error, this was the structure that I ended up with that worked.

Holidays are stored into the same types of collections as events.

Categories are stored in the fileHandler class as an Observable buffer. Observable buffer observes changes to its elements, which is useful when changes need to be reflected to a user interface. This is important with categories in this program, because when adding a category, the new category needs to be shown in the graphical user interface in the choicebox.

Besides storing events, categories and reminders in buffers, they were also used when creating the graphical user interface. For example, when creating the daily view with time slots for all 24 hours of the day, many HBoxes needed to be created and these were created by using a for loop and adding all of the HBoxes to a buffer. To add contents to the HBoxes, the different elements were accessed with their indices.

Files and Internet access

Events are stored as iCalendar (ICS) files (Events.ics). This is a file format that many calendars, such as Microsoft Outlook, Google Calendar and Apple Calendar use. Saving the event to this format allows to move the events to other calendars easily. The iCalendar format is a text-based format, and it is saved with the .ics extension. Each event in the iCalendar file is represented as a “VEVENT” component. Each VEVENT component contains properties describing the information about the name of the event (SUMMARY), start date and time (DTSTART), end date and time (DTEND) and category (CATEGORIES). Each line begins with the property name, which is followed by a colon and after this the property value.

Example:

BEGIN:VCALENDAR
PRODID:-//Joanna Keränen//Calendar 1.0//EN
VERSION:2.0
CALSCALE:GREGORIAN
BEGIN:VEVENT

DTSTAMP:20240419T092255Z
DTSTART:20240418T130000
DTEND:20240418T140000
SUMMARY:Lecture
CATEGORIES:School
END:VEVENT
BEGIN:VEVENT
DTSTAMP:20240419T092255Z
DTSTART:20240419T170000
DTEND:20240419T180000
SUMMARY:Dance class
CATEGORIES:Hobbies
END:VEVENT
END:VCALENDAR

Holidays are stored as iCalendar (ICS) files (Holidays.ics). The Finnish public holidays have been retrieved from google calendar, and therefore it contains more properties presented in the example below.

Example:

BEGIN:VCALENDAR
PRODID:-//Google Inc//Google Calendar 70.9054//EN
VERSION:2.0
CALSCALE:GREGORIAN
METHOD:PUBLISH
X-WR-CALNAME:Helgdagar i Finland
X-WR-TIMEZONE:UTC
X-WR-CALDESC:Helgdagar och firanden i Finland
BEGIN:VEVENT
DTSTART;VALUE=DATE:20190419
DTEND;VALUE=DATE:20190420
DTSTAMP:20240414T152121Z
UID:20190419_slnrhvhb4jbboqt2sv5cf52940@google.com
CLASS:PUBLIC
CREATED:20210826T142848Z
DESCRIPTION: Allmän helgdag
LAST-MODIFIED:20210826T142848Z
SEQUENCE:0
STATUS:CONFIRMED
SUMMARY:Långfredagen
TRANSP:TRANSPARENT
END:VEVENT

END:VCALENDAR

Categories are stored as text files (Categories.txt). This file contains the information about the names of the categories. Each category is presented on a separate line.

Example:

School
Work
Hobbies

Testing

The program was mainly tested using the graphical user interface. A simple gui was one of the first parts to be implemented in the project, so functionalities could easily be tested using the gui from an early stage of the project. Things that were tested using the gui include getting the correct days, week and month to the calendar view, switching between weeks, adding and deleting events, reading and writing to files (text and iCalendar files), adding categories, switching between weekly and daily views, adding events to the correct time slot in the daily view, adding reminders, and adding holidays to the calendar view.

In addition to this, the program has been tested with unit tests that are at the end of each class. This was done by adding print statements to the actual methods and the methods were called in the unit testing to see if the methods were producing the correct output. However, most of these print statements have been deleted from the code so that it doesn't print everything every time the methods are called when using the program. This leads to most of the unit tests not to print anything valuable anymore at this stage. A better way to have done this would have been to create unit tests that work at all stages, for example add the print statements to the unit tests instead of inside the methods, so that the unit tests can be run again at any stage of the project. Some of the unit tests still work. For example the unit test for the class dateHandles still works because the printing of the values is done in the test function. To debug the program, print statements were added between the code, and the unit tests were run, to figure out what is causing the wrong output. Unit testing was done to the classes fileHandler, dateHandler and CalendarHandler, which contain the most important functionalities of the program, such as adding and

deleting events, reading and writing to files and getting the correct dates, week and month for the weekly view.

The format of the iCalendar file has been tested by importing the events file to google calendar. The format of the iCalendar file was also tested with an iCalendar validator on the website iCalendar.org (https://icalendar.org/validator.html).

The program passes all of the tests and the testing covers all of the significant parts of the program. Based on this there should not be any bugs in the program. The program has also been tested with incorrect inputs using the graphical user interface and this produces the correct error message in the form of an alert.

The testing followed the testing plan quite well. However, the testing plan described better unit tests that could be run at any stage of the project. These were not implemented, which was described above.

Known bugs and missing features

One bug that the program has is in the handling of incorrect inputs. Incorrect inputs are handled with the try catch structure and the program does catch incorrect inputs and show the user an alert that prompts the user to check the input. However, after inserting an incorrect input once, the program doesn't work even with correct inputs. Instead it keeps showing the alert when trying to add a new event or open the daily view.

The program also produces the following error message when launching the program:

SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.
Apr 25, 2024 7:06:40 PM com.sun.javafx.application.PlatformImpl startup
WARNING: Unsupported JavaFX configuration: classes were loaded from 'unnamed module @1dcb2879'

The program works despite this.

The program does however have some defects and incorrectly working features. When adding a multiple day event that spans over multiple weeks, for example if an events start date is sunday 28.4.2024 and end date is tuesday 30.4.2024, then the daily view for the days between the start and end days does not look exactly correct. The event name should be placed into the all day section, but instead the event name is placed in every time slot of the day. The start and end days look correct and the event start and end times are taken into account. This works correctly for multiple day events that fit within one week, so this minor issue could be fixed in a similar way but due to time resources this could not be done for this project.

Another defect is that the reminders are not saved to a file, so whenever the program is closed, the reminders are deleted because they are only stored in a buffer. Reminders could also have been implemented as alerts instead of putting them in a separate column in the weekly view. The reminders could have been saved as VTODO components to the iCal file.

One feature that is not in the requirements of the task but could be considered a missing feature is that categories cannot be deleted. At the moment, categories can only be deleted from the program by removing them directly from the categories file. To make the calendar actually usable, the user would need to be able to remove categories in the program, for example in the case of misspelling.

Another incorrectly working feature is that in the daily view, the events are placed based on full hours. For example if an event is at 13.30-15.30, the events will be placed in the time slots 13.00, 14.00 and 15.00. A better way to have done this would have been to create a box for the event that would only contain the name of the event once and it would be placed to the correct spot also taking into account minutes, or at least half hours.

This program was planned to be implemented as a medium level task. However it was completed as an easy task. Therefore some features were not implemented. These include colors attached to different categories, only showing selected categories at a certain time in the calendar, reserving the time slot for an event by painting the time with the mouse, including additional information to events and the functionality to search for events related to specific things such as course, hobby or person.

3 best sides and 3 weaknesses

One good thing is using the iCal4j library to read and write to iCalendar files. This way to do file handling is particularly good, because the code is clearly structured, easy to understand and quite concise. There is no need to separately go through all different cases, which would be the case when using strings. For example, the holidays for this calendar have been retrieved from google calendar and in this iCal file the property name for the start and end dates are DTSTART;VALUE=DATE, and in the events file the property name is DTSTART. With the methods of iCal4j, these can be read as the same thing (DTSTART). There is also a very small risk that the resulting iCalendar file would be in an incorrect format, when writing to the file.

Another good thing about the implementation of this project is that by using gridpane and percentage sizes for boxes instead of fixed sizes makes the graphical user interface scalable. This is extremely important in real websites and applications because this way the program looks nice on different size screens.

The main class of the program was extremely repetitive, but by refactoring the code to use for loops to create for example the 24 sections for all of the hours of the day made the main class approximately 200 lines shorter. This was a good fix to the program.

This program has some repetitiveness. An especially weak part of the program is the getEventsToAddToCurrentWeek. This method has repetitiveness in it because first it goes through the eventsBuffer and after this the holidaysBuffer using the same logic. Originally both the events and holidays were added to the same buffer to avoid repeating code. However this caused issues in writing to the events file because the method writeToEventsFile gets the events to write to the file from eventsBuffer. Holidays should not be added to the events file so events and holidays had to be separated to different buffers, eventsBuffer and holidaysBuffer. This error came up late in the project and due to the limited time resources, the only way to make the code work was to repeat some code. Also by now adding the events and holidays separately in CalendarHandler some repetitiveness from fileHandler could be removed if the project was continued. FileHandler has a method readIcsFile and readEventsFile calls this method. ReadHolidaysFile could also utilize this method.

Currently the wrong end date of holidays is fixed in the readHolidaysFile, but it could be taken into account in the class calendarHandler now that the holidays and events are separated to different buffers.

Another weak part of the program is the main class. The main class contains all of the graphical user interface and all of the methods to update the graphical user interface, and everything in the main class is inside the start function. A better way to have done this would have been to have separate methods inside main and not have everything inside the start function. Another possible way to have done this is to create separate classes for different parts of the graphical user interface. Both of these ways would have made the implementation clearer.

Another weakness of this program is that adding an event happens in a completely separate view. A better way to have done this would have been to use a text input dialogue described at scalafx.org (https://www.scalafx.org/docs/dialogs_and_alerts/). This would also have saved a lot of work because the dialogue automatically creates some elements to the pop up box.

Deviations from the plan, realized process and schedule

The first thing that was implemented was the graphical user interface. This allowed for easy testing of the program. Second, a very simplified version of reading files was created. After this, the correct dates, months and weeks were added using date time, so that the graphical user interface shows the correct days, and different weeks can be browsed. After this the functionality to add categories was implemented, which also required the implementation to read and write to a text file. After this, reading and writing to ics files was implemented, and then adding and deleting events, to and from the graphical user interface, was done. Then, holidays were retrieved from google calendar and added to the calendar. After this, error messages for incorrect input were created. Finally, the main class, that has all of the code for the gui, was refactored to remove some of the repetitiveness.

7.3.	- Initial idea of the class structure was created. - Reading from EventExample file and printing it (This was a file in iCal format that has been deleted from the project).

	- Started working on the graphical user interface, using GridPane.
27.3.	- Keep working on the graphical user interface. - Functionality to switch between different views was created.
14.4.	- Views for adding an event and daily view were created. - Adding the correct dates, week and month to the weekly view. - Reading and writing to categories file (text file) finished. - Reading and writing to events file (ics file). This is not complete yet. - Getting holidays as an ics file from google calendar. These have not yet been added to the project. - Creating methods to add events to the calendar. This is not complete yet.
24.4.	- Adding holidays to the calendar. - Reading and writing to an ics file finished. - Testing transferring events file to another calendar. - Adding events to the calendar finished. This includes all day events and events that last multiple days. - Error handling in the case of incorrect input.

Figure 5: Progress between sprint meetings based on the progress log.

The time estimate did not match the plan. One of the biggest differences was that I did not correctly estimate how much time creating the graphical user interface would take. I also underestimated the amount of time that it took in the beginning of the project to learn about things such as scala date time, file handling and testing, and how much time it takes to understand how to actually create a project like this, due to it being the first project like this that I have done. One thing that also affected the time estimate not matching reality was that holidays and other school projects were not taken well into account in the planned schedule.

The order of implementation was also different from the plan. The initial idea of the class structure and the overall plan for the project changed a lot and therefore the things needed to be implemented and the order that they need to be implemented in changed. Some things from the plan were completely left out, such as recurring events. One of the biggest differences between the plan and reality was that the first things implemented were the graphical user interface and file handling instead of the

classes Calendar and Event that were actually created quite far into the project. Categories were also added quite early on in the project even before adding and deleting events to the graphical user interface.

During this process I learned how an entire project is created from start to finish. By trying things and with the help of my assistant I managed to create the project in a logical order. I learned how a large project like this can be approached and in the future I believe that I would be able to create a more accurate plan of what needs to be done and in what order. I also learned not to underestimate the time that it takes to read about and learn new things.

Final evaluation

The quality of the code is good in some parts but does have room for improvement in other parts. For example FileHandler and DateHandler have good quality code. However, FileHandler does have some repetitive code, but this could be removed as described earlier in this document. CalendarHandlers methods add and delete events are good and concise in my opinion. However the method to get the events to add the current week is very long and quite unclear. It has a lot of if else statements and multiple for loops. It also goes through different collections and changes the information into another collection. This could have been made simpler as described earlier in this document. This method could possibly also have been split to multiple methods. Main only contains one method, start, and as described in this document it should have been split to either several methods or even several classes.

The program could be improved by fixing the existing bugs that were previously described and creating more features to make the program more user friendly, for example the dates could be chosen from a calendar instead of giving them as a string. The program could also accept different types of input. For example the date could be inserted as 4.4.2024 or 04.04.2024. To make the calendar better also the requirements of the medium and hard tasks could be implemented.

If I started the project again from the beginning I would have paid attention to not make the code so repetitive. This would have saved time from refactoring the code later. If I started the project again, I would also have started working faster with the

project. At the beginning of this project it was extremely difficult to create a realistic schedule but after this project I feel better equipped to understand the requirements of a project like this and be able to plan my time better. If I started this project again, I would also be better equipped to plan a more accurate class structure. when teh original class structure was created, I did not know how a gui is created in a program like this. I would also plan what the different methods are supposed to do in more detail to avoid the situation of having long methods that do multiple different things, such as the method getEventsToAddToCurrentWeek in CalendarHandler. If I did this project again, I would also commit my code more often after each functionality that is created. This would make it easier to go back to working code in case of issues. I would also create unit tests that can be run at every stage of the project to make sure that changes in one part of the code don't break other parts of previously working code. This happened a few times during this project. For example writing to the events file worked before but adding holidays to the same buffer as events broke the functionality to write to the events file. This could have immediately been fixed if unit tests were run.

References and code written by someone else

I have used the material Scalaa kootusti [1] from the course Programming 1 to remind myself how basic methods work exactly, for example how the Map structure works. I have used the iCal4j documentation [2] to help use this library in the implementation of reading and writing iCalendar files. I have used the Baeldung website [3] to look for information on the date time libraries in scala, how to read and write to text files as well as information on data structures. In have also looked at the scala documentation [4] as well as GitHub documentation to look for methods of iCal4j, for example calendarOutputter [5]. I also looked at some code examples on stackoverflow, for example when overriding the equals method of the case class I utilized an example [6].

External libraries have been used in this program. The iCal4j library was used to create the methods for reading and writing to iCal files. Nscala time and joda time libraries were used to retrieve the correct dates for the

weekdays to the calendar as well as the correct month and week. Methods of these libraries were used to calculate the dates for the following and previous weeks correctly so that the weeks can be browsed.

I have used chatGPT as an aid in this project. I have not copied any code, but I have asked questions about basic functions and implementations. For example I have asked things such as “is it possible to have two elements with the same key in a Map” and “what is the difference between a List and a Buffer”.

[1] https://plus.cs.aalto.fi/o1/2023/wNN/scala/

[2] https://www.ical4j.org/examples/model/

[3] https://www.baeldung.com/

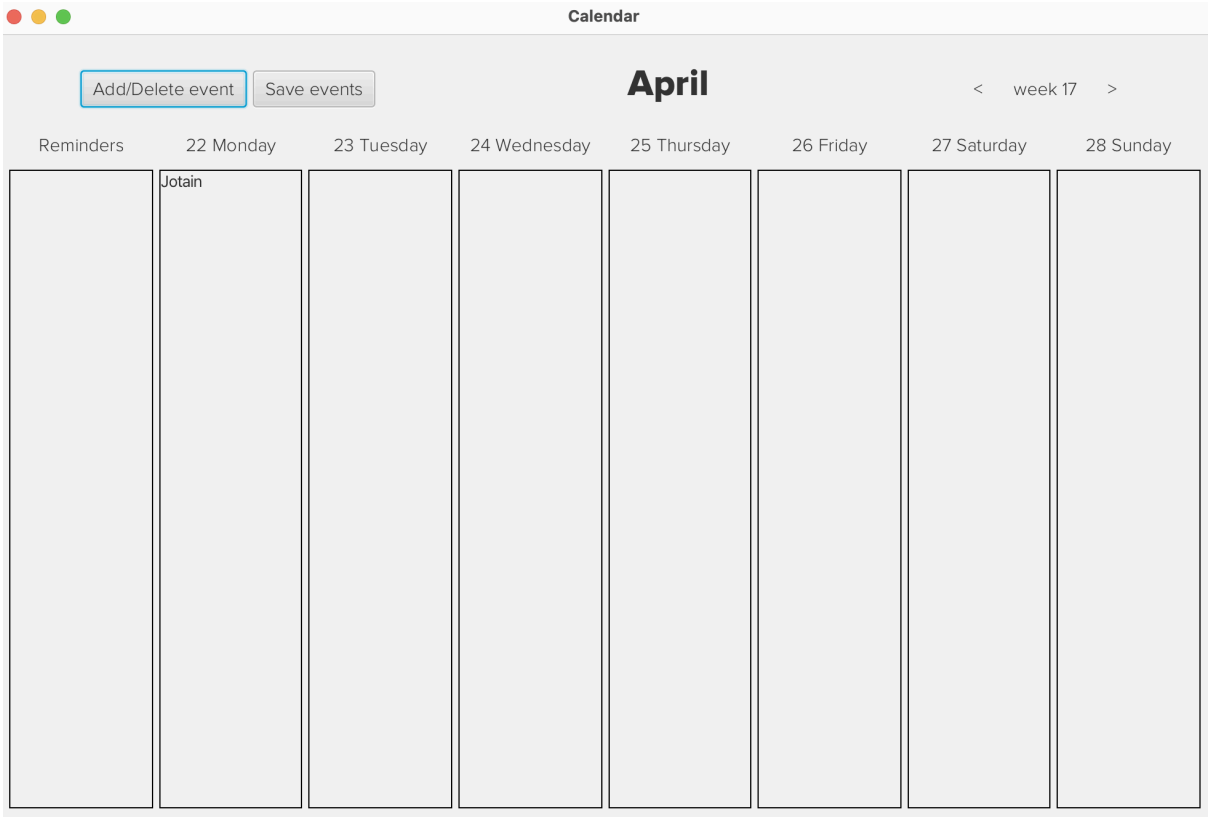
[4] https://docs.scala-lang.org/

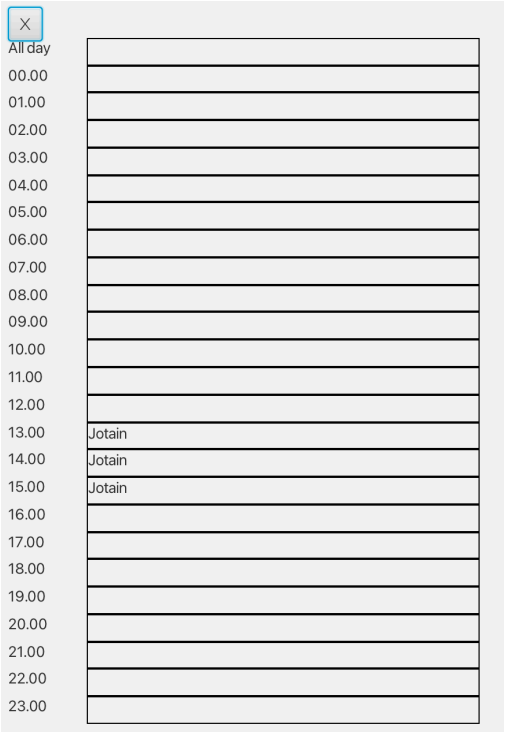
[5] http://ical4j.github.io/docs/ical4j/api/3.0.2/net/fortuna/ical4j/data/CalendarOutputter.html

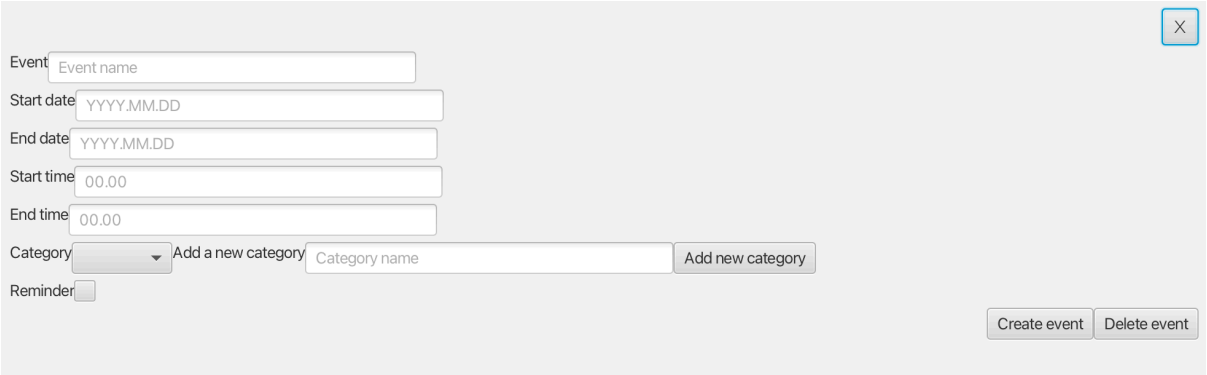
[6] https://stackoverflow.com/questions/39402494/scala-match-in-equals-method

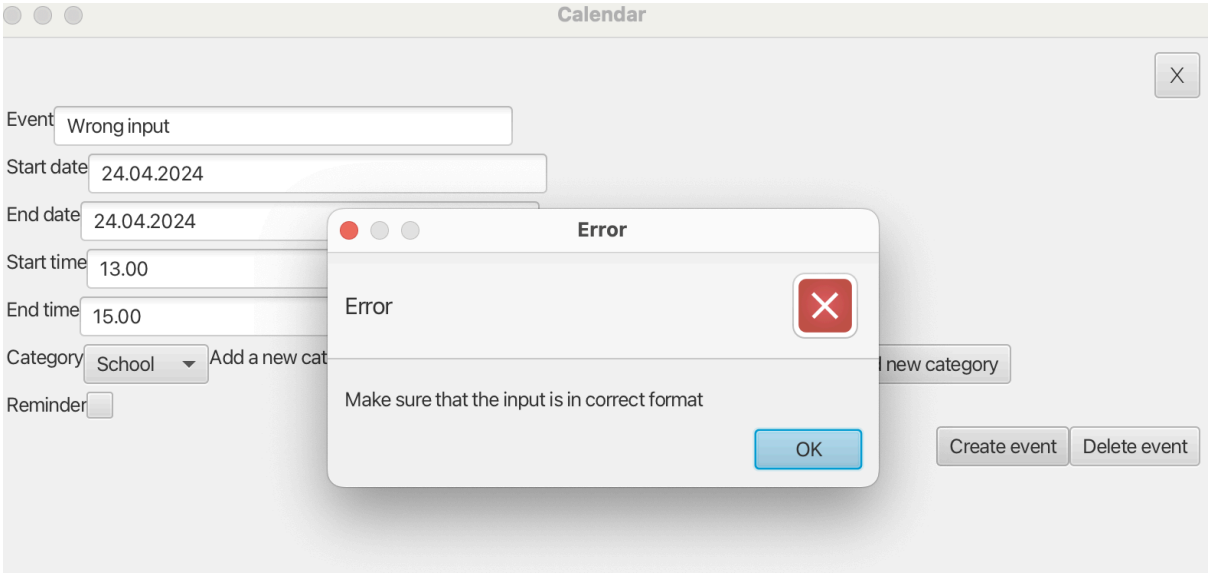
Appendices

This section has screenshots from the graphical user interface.




X
All day
00.00
01.00
02.00
03.00
04.00
05.00
06.00
07.00
08.00
09.00
10.00
11.00
12.00
13.00 Jotain
14.00 Jotain
15.00 Jotain
16.00
17.00
18.00
19.00
20.00
21.00
22.00
23.00


X
Event
Event name
Start date
YYYY.MM.DD
End date
YYYY.MM.DD
Start time
00.00
End time
00.00
Category
Add a new category
Category name
Add new category
Reminder
Create event
Delete event


Calendar
X
Event
Wrong input
Start date
24.04.2024
End date
24.04.2024
Start time
13.00
End time
15.00
Category
School
Add a new cat
new category
Reminder
Error
Error
Make sure that the input is in correct format
OK
Create event
Delete event