

Calendar document

General description

A general description of the properties of the calendar.

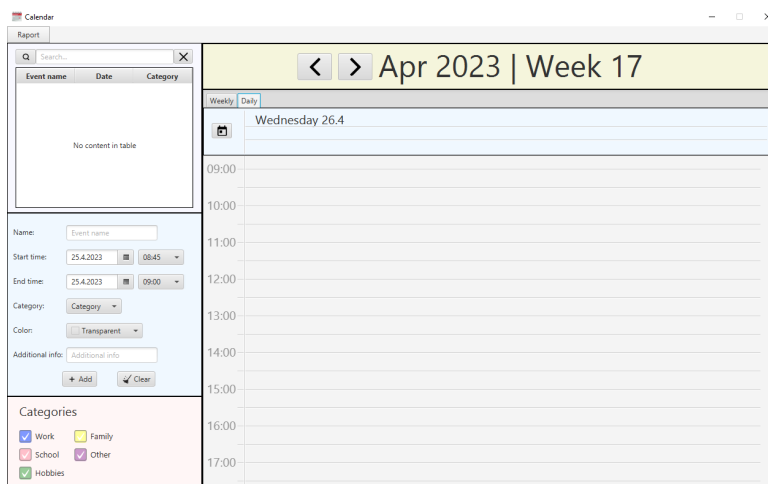
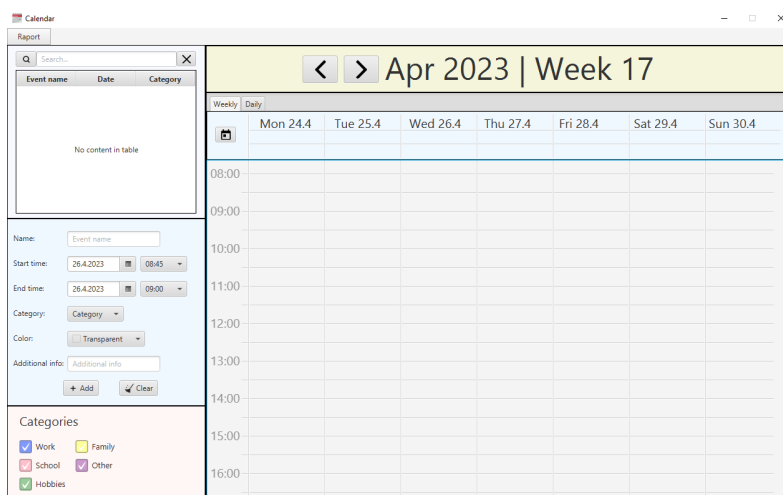
- Name, date, start and end time of the event, for example a lecture, exercise or meeting.
- Different types of things can be defined in the calendar, for example things related to studies, things related to different hobbies, working hours, meetings, parties, ..., so that each of these related things can be dealt with in its own group.
- Reminders about a task that needs to be done.
- Public holidays
- The calendar shows one week at a time, each day in its own column, with the things recorded for that day.
- A view showing all 24 hours of the day in its own line and things are shown for the hour where they are recorded.
- The reserved time slot can be long (over midnight or several days).
- A view that only shows the things recorded for one day in a row
- By default, the current week is visible.
- The user can browse forward or backward in time one week at a time.
- Things are added to the calendar by asking the user (in text) the date, start time and subject.
- You can delete things from the calendar.
- Calendar information is stored in a file and read from there.
- You can attach a color code to time reservations, which describes different types of things, such as studying, work tasks, hobbies, etc. You can use different shades of color to indicate a more precise division, e.g. lectures in dark red, exercises in pink, etc.
- From the calendar, which shows the hours of one day or one week, you can reserve a suitable time by painting with the mouse and enter the desired information.
- Additional information can be entered in the calendar entry (e.g. place, course to which the matter is related, web link, persons affected by the matter in addition to yourself, such as members of the work group, friends, thesis supervisor)
- You can choose which things from the calendar are displayed, for example only working hours or things related to studying.
- You can search for all events related to a specific course, hobby or person who participates in them in addition to yourself.

User interface

- Program startup
- Adding Events (Manually as well as by Mouse Painting)
- Searching events
- Filtering events
- Notification of upcoming events
- Editing/deleting events
- Current day/week
- Finnish public holidays
- Invalid input

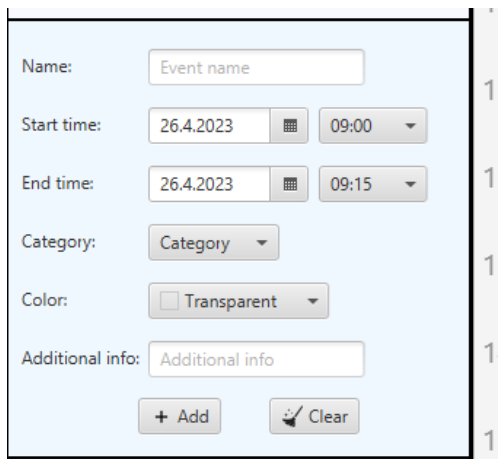
1. Program startup

An overview of the calendar appears. The view can be selected as a week or day view. The arrows at the top can be used to navigate through days/weeks depending on the selected view.



2. Adding events

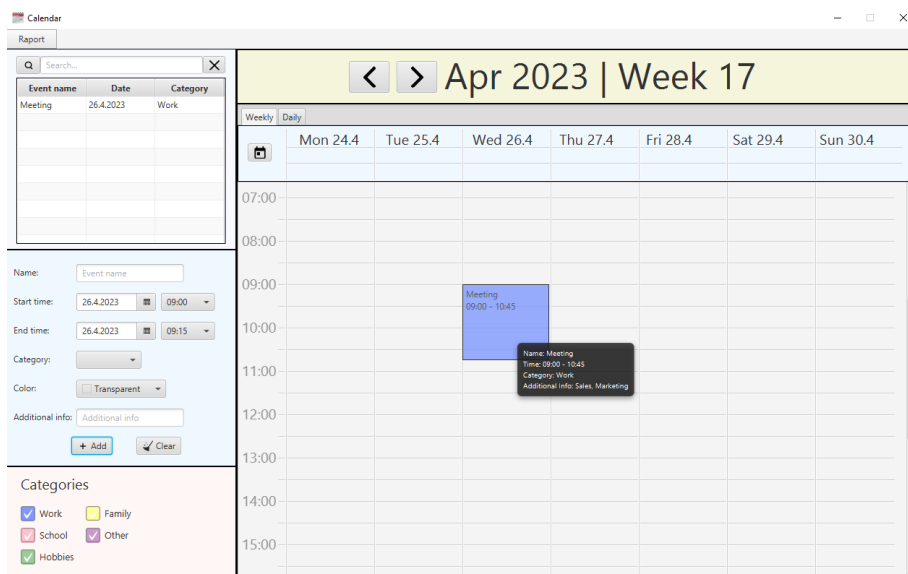
The user enters the required information into the designated fields. The mandatory fields include name, start time, end time, and category. The event is added by pressing the "Add" button. The user can clear the entered data by pressing "Clear". The system clock is used to determine the start and end times.



A screenshot of a web form for adding a new event. The form is light blue and contains the following fields and controls:

- Name:** A text input field with the placeholder text "Event name".
- Start time:** A date and time selector showing "26.4.2023" and "09:00".
- End time:** A date and time selector showing "26.4.2023" and "09:15".
- Category:** A dropdown menu with the text "Category".
- Color:** A checkbox labeled "Transparent" followed by a dropdown menu.
- Additional info:** A text input field with the placeholder text "Additional info".
- Buttons:** Two buttons at the bottom: "+ Add" and "Clear" (with a trash icon).

Event information can be quickly viewed by moving the cursor over it.



A screenshot of a calendar application interface. The interface includes a sidebar on the left and a main calendar area on the right.

Sidebar:

- Report:** A section with a search bar and a table showing event details.
- Event details form:** A form for adding or editing events, identical to the one in the previous image.
- Categories:** A list of categories with checkboxes: Work (checked), Family (checked), School (checked), Other (checked), and Hobbies (checked).

Main Calendar Area:

- Header:** Shows the current month and week: "Apr 2023 | Week 17".
- Grid:** A weekly grid showing days from Monday to Sunday. The time slots range from 07:00 to 15:00.
- Event:** A blue event block is visible on Wednesday, April 26th, from 09:00 to 10:45. A tooltip is displayed over the event, showing the following details:
 - Name: Meeting
 - Time: 09:00 - 10:45
 - Category: Work
 - Additional info: Sales, Marketing

An alternative way to select a time range for an event is by using mouse selection. The information updates automatically when the user has selected the desired time period by dragging the mouse.

Calendar

Report

Q

Search...

X

Event name	Date	Category
Meeting	26.4.2023	Work

Weekly

Daily

Calendar icon

Mon 24.4

T

07:00		
08:00		
09:00		
10:00		
11:00		
12:00		
13:00		

Name:

Lecture

Start time:

24.4.2023

Calendar icon

08:00

Dropdown arrow

End time:

24.4.2023

Calendar icon

10:00

Dropdown arrow

Category:

School

Dropdown arrow

Color:

Pink

Dropdown arrow

Additional info:

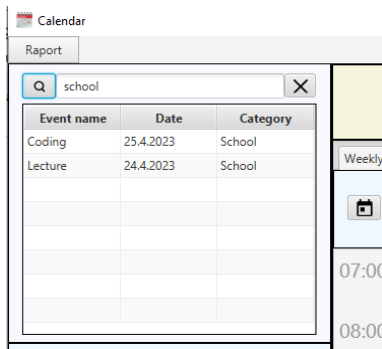
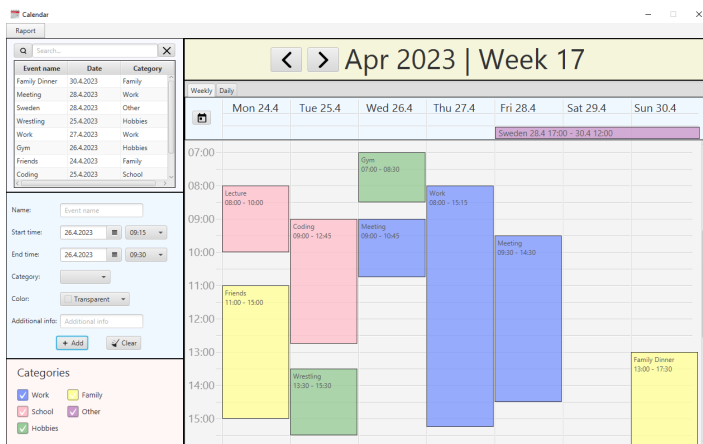
Additional info

+ Add

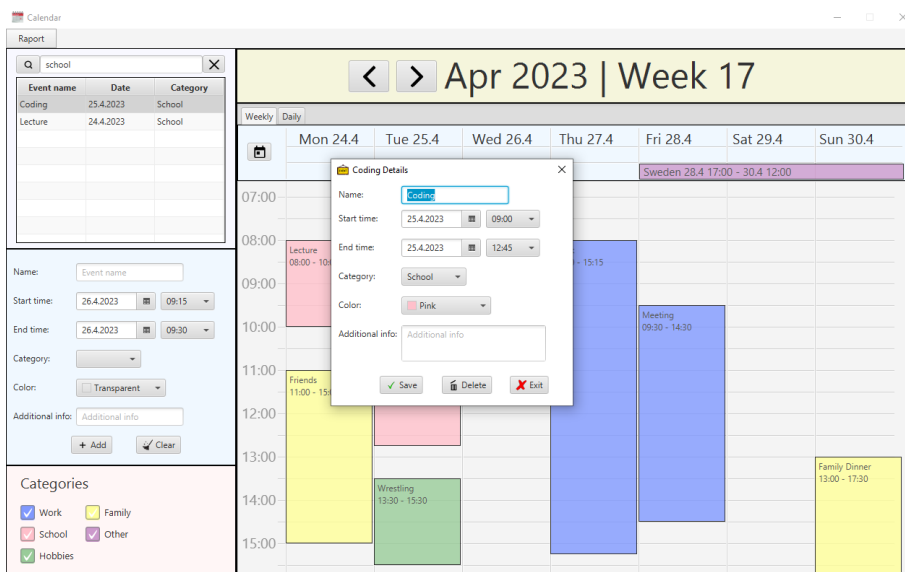
Clear

3. Searching events

Events can be searched using the search field located in the upper left corner. The search can be performed based on the event name, category, or any additional information associated with the event.



By double-clicking on an event in the search field, the event details will be displayed, and the view will automatically update to show the week containing the event.



4. Filtering events

Users have the option to choose which types of events are displayed in the calendar. They can select the specific event types they want to view.

Categories

<input checked="" type="checkbox"/> Work	<input checked="" type="checkbox"/> Family
<input checked="" type="checkbox"/> School	<input checked="" type="checkbox"/> Other
<input checked="" type="checkbox"/> Hobbies	

For example, only work and school-related events:

Calendar
< >

Report Search...

Event name	Date	Category
Emily Dinner	30.4.2023	Family
eeting	28.4.2023	Work
eden	28.4.2023	Other
restling	25.4.2023	Hobbies
rork	27.4.2023	Work
yrm	26.4.2023	Hobbies
iends	24.4.2023	Family
pding	25.4.2023	School

Name:

Start time:

26.4.2023 09:15 ▼

End time:

26.4.2023 09:30 ▼

Category:

Color:

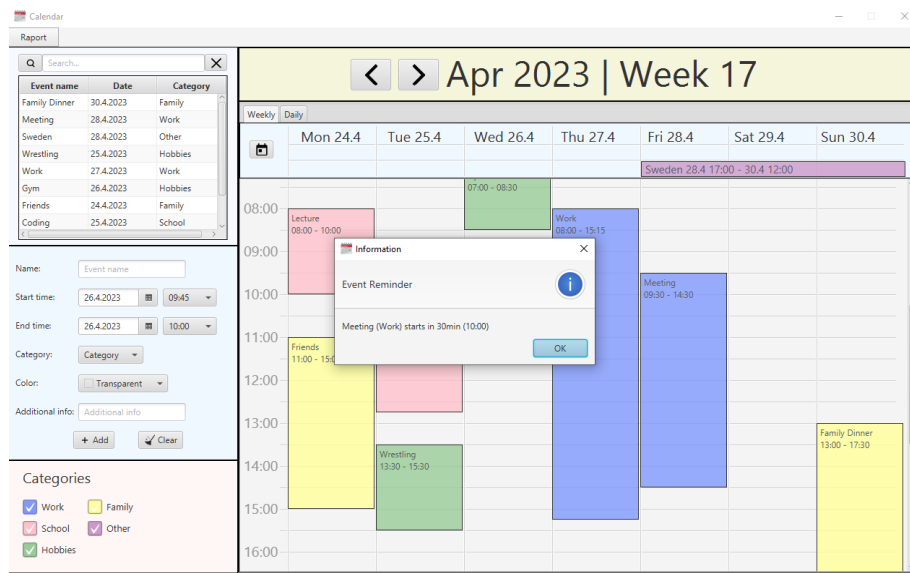
Additional info:

Categories

☒ Work ☐ Family
☒ School ☐ Other
☐ Hobbies

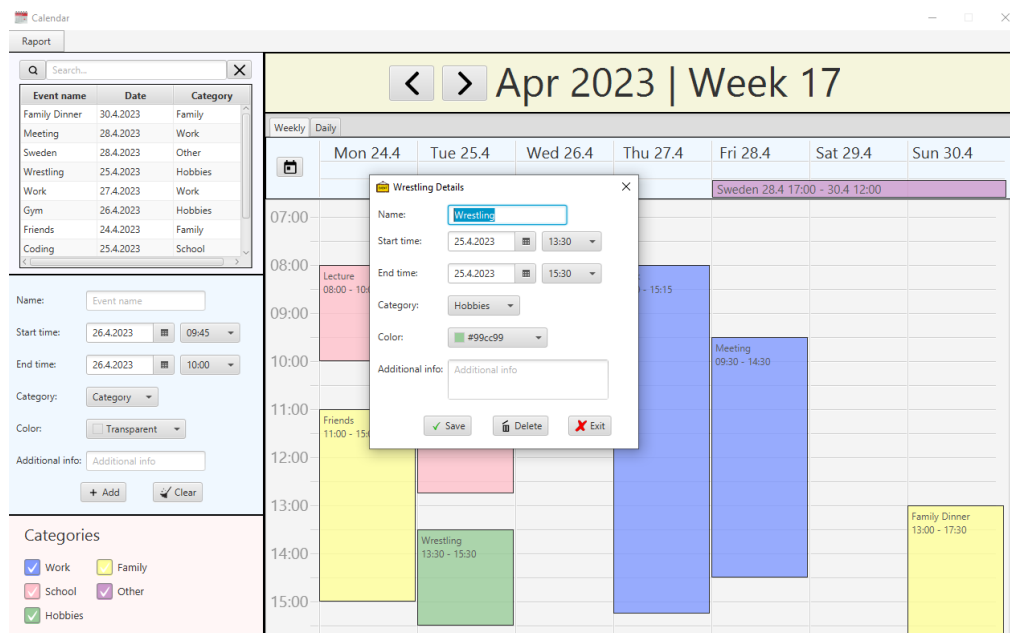
5. Notification of upcoming events

The calendar will send a reminder for upcoming events 30 minutes before their scheduled start time.



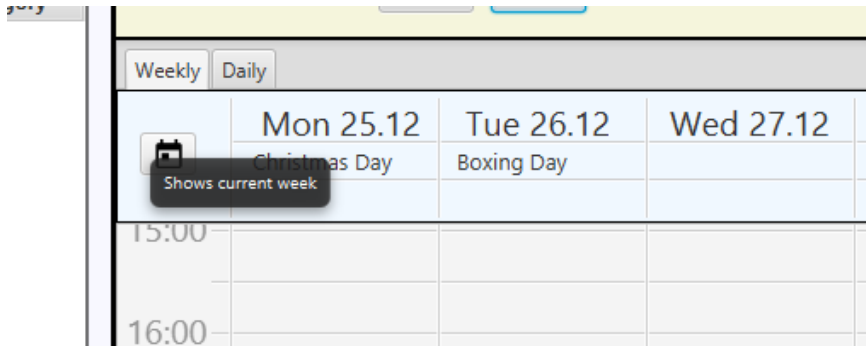
6. Editing/deleting events

By clicking on an event, a window will open where you can edit the event details or delete the event entirely.



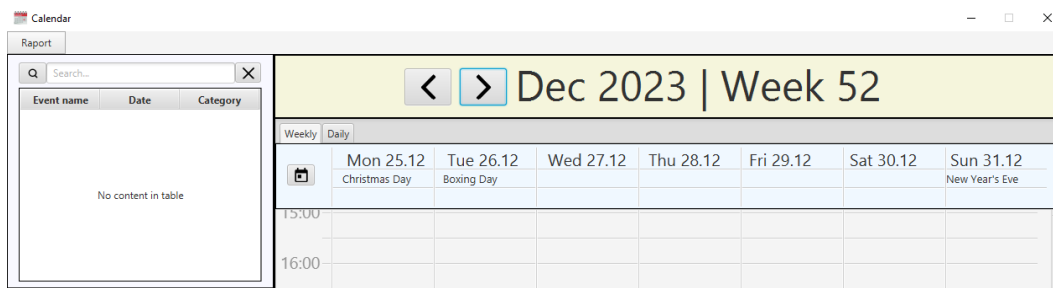
7. Current day/week

The user can quickly navigate back to the current day/week by clicking a button to refresh the view.



8. Finnish public holidays

Below the dates, there are the public holidays in Finland.

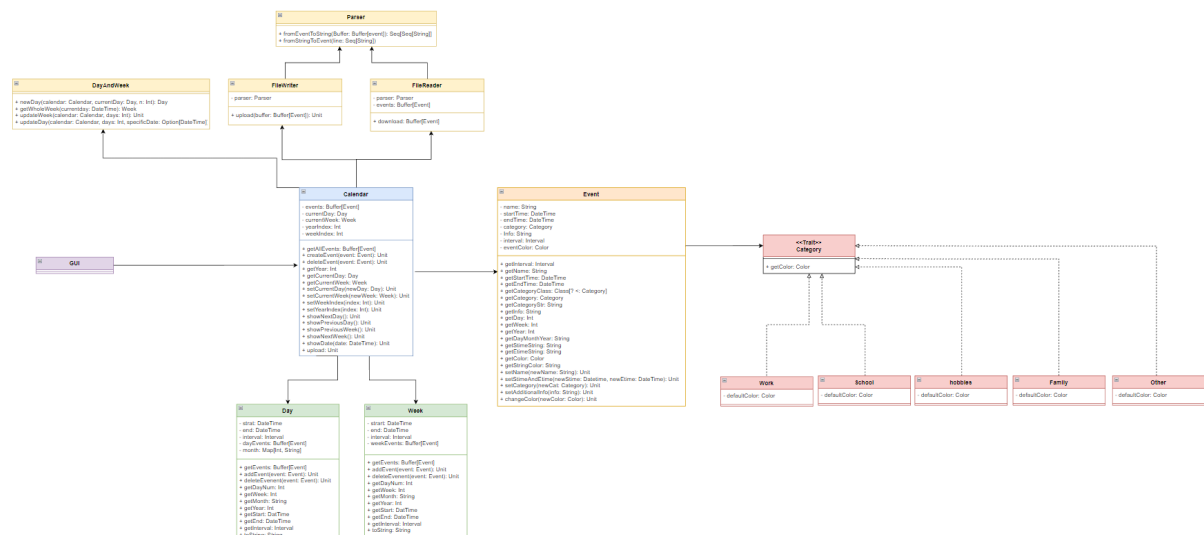


9. Invalid input

The program notifies the user about an invalid input (End date is before the start date).

The screenshot shows a form for adding a meeting. The fields are: Name (Meeting), Start time (26.4.2023, 19:15), End time (Invalid Date, 19:30), Category (Work), Color (#8099ff), and Additional info (Additional info). The 'Invalid Date' text is in red. There are '+ Add' and 'Clear' buttons at the bottom.

Program structure



The UML diagram presents the class structure of the project. The diagram excludes the classes related to the graphical user interface (GUI, shown in violet) for clarity and optimization purposes.

The diagram includes color codes to enhance the clarity of class categorization:

GUI (Graphical User Interface) = violet

Calendar = blue

Event = orange

Day, Week = green

Category and subclasses = red

Service classes = yellow

Calendar class:

- Keeps track of the events (Event) added to it.
- Knows the specific day (Day) and week (Week) associated with the events.

Key methods of the class:

- *createEvent(event: Event)*: Adds events to the calendar.
- *deleteEvent(event: Event)*: Removes events from the calendar.
- *showPreviousDay()*, *showNextDay()*, *showPreviousWeek()* and *showNextWeek()* changes the displayed day/week
- *upload()*: Uses the FileWriter class to write the events in the calendar to a CSV file for storage.

Event class:

- Represents an individual event.
- An event is characterized by its name, start time, end time, category, color, and additional information (optional).
- An interval is created for each event based on the start and end times. The interval allows for easy determination of which day or week the event belongs to
- Each event belongs to a category (*Category*), which allows for categorizing events as needed.

Category trait:

- Categorizes individual events into different classes.
- Subclasses of *Category* include *Work*, *School*, *Hobbies*, *Family*, and *Other*.
- Determines the default color for events belonging to a specific category.

Day class:

- Represents an individual day.
- Knows which calendar it belongs to.
- Day object has an interval created from the start and end time.
- Keeps track of events on that particular day.

Week class:

- Represents an individual week.
- Knows which calendar it belongs to.
- Week object has an interval created from the start and end time.
- Keeps track of events during that particular week.

DayAndWeek class:

- Used by the Calendar class. The purpose of this class is to eliminate code duplication and divide specific tasks into helper functions.

Key methods of the class:

- `newDay(calendar: Calendar, currentDay: Day, n: Int): Day`
 - Calculates a new day in the calendar based on the current day
- `getWholeWeek(currentday: DateTime): Week`
 - Calculates the entire week in the calendar based on a specific day.
- `updateWeek(calendar: Calendar, days: Int): Unit`
 - Updates the week in the calendar by moving it forward or backward by a specified number of days.
- `updateDay(calendar: Calendar, days: Int, specificDate: Option[DateTime]): Unit`
 - Updates the day in the calendar by moving it forward or backward by a specified number of days.

Parser, FileWriter, and FileReader are responsible for ensuring that the events added to the calendar are correctly saved to a file. This ensures that the added events are not lost when the program is closed and restarted.

Parser class:

- A helper class between the Calendar and the FileWriter/FileReader.

Key methods:

- `fromEventToString(buffer: Buffer[Event]): Seq[Seq[String]]`
Converts the events in the calendar into a format that can be read by the FileWriter.
- `FromtTringToEvent(line: Seq[String]): Event`
When reading from a file (FileReader), converts a line of text into an Event object.

FileWriter class:

- Responsible for writing the events from the calendar to a CSV file.

Key methods:

- `upload(buffer: Buffer[Event]): unit`
Writes the events from the calendar to a CSV file.

FileReader class:

- Reads a CSV file line by line. It creates Event objects based on the data in the file and adds them to the calendar.

Key methods:

- `download: Buffer[Event]`

Algorithms

The most essential algorithms of the project focus on the user interface. Simple algorithms have been used for the program's logic, such as event addition/deletion, determining a new day/week, and saving events to a file. On the user interface side, more challenging algorithms have been utilized, such as calculating public holidays (e.g., Easter, Midsummer), displaying events on the screen, and calculating event durations (selecting a time range with the mouse).

Key algorithms:

- Event creation and addition:
 1. User inputs event details
 2. The input data is transformed into the appropriate format to provide the necessary parameters for creating an Event object.
 3. The event is added to the variable within the Calendar class where events are stored, and if necessary, to the events of the corresponding day (Day) and week (Week).
 4. Check if the event spans multiple days.
 5. Based on the above, calculate the position of the event on the display.
 6. Update the CSV file and the user interface accordingly
- Determining the event time by mouse painting:
 - This calculation utilizes the cursor coordinates (x, y):
 1. Determine the location where the cursor was clicked.
 2. Measure the distance the cursor has been dragged.
 3. Determine the location where the cursor is released.
- Determining holidays:

Most holidays occur at the same time every year, but algorithms have been used to determine the dates for Easter and Midsummer (Juhannus) holidays. These algorithms can be found for free on the internet:

 - Algorithm for calculating Easter:
<https://www.assa.org.au/edm>
 - Midsummer (Juhannus) determination: information obtained from <https://fi.wikipedia.org/wiki/Juhannus>

- Writing events to a file:
 - In the class descriptions, it was mentioned that the FileWriter class writes the added events to a file. The algorithm transforms the event data into a format that FileWriter can write to the file
- Reading events from a file:
 - In the class descriptions, it was mentioned that the FileReader class reads the added events from a file. The algorithm creates an Event object based on the data obtained from the file, which is then added to the calendar.

Data Structures

The main collection types used in the project for storing data are Buffers and Maps (both mutable).

The selection of data types is based on their suitability. Next, I will go through the collection types and data structures I have chosen:

- Buffer collection type:
 - Used for managing events (Event)
 - Can easily store/add/remove/update events
- Map data structure:
 - Highly suitable for storing information related to holidays.
 - Used for determining weekdays and months:
 - Mapping integer values (1-7) to weekdays and (1-12) to months.
 - Illustrative example:

```
val dayOfWeek: Map[Int, String] =  
  Map(1 -> "Monday",  
       2 -> "Tuesday",  
       3 -> "Wednesday",  
       4 -> "Thursday",  
       5 -> "Friday",  
       6 -> "Saturday",  
       7 -> "Sunday" )
```

Some external libraries required different collection types, such as Seq and ObservableBuffer.

- Seq collection type:
 - The Seq collection type is used for reading files in the format Seq[Seq[Any]], which allows the file to be written correctly.
 - An external library has been utilized for reading and writing files:
<https://github.com/tototoshi/scala-csv>
- ObservableBuffer collection type:
 - The ObservableBuffer collection type is used in the project's user interface, implemented with the ScalaFX library. The Tableview and Combobox components I have used require this collection type for storing data and enabling observation of changes.

Files

The program deals with CSV, CSS, and PNG files. The files have been chosen based on their suitability.

CSV file:

- The calendar saves the added events into a CSV file. The file is easily readable and editable. Each event is stored on its own line, with the event details (input parameters) as columns. When the calendar is restarted, it reads the event information from the saved file.

CSS file:

- The CSS file has been used to refine the appearance of the user interface. It helps certain components (such as combobox) achieve the desired look, improving the clarity of the user interface and creating a more meaningful user experience.

PNG:

- PNG image files have been used to enhance the user interface and improve its visual appearance.

Testing

The logic of the program, which includes the classes depicted in the UML diagram, was tested using unit testing. Before creating the user interface, the functionality of the methods was ensured. An external library, <https://www.scalatest.org/>, was used for unit testing. The tests provided key inputs to the program to verify the return values of methods and their impact on other objects in the program. Every method of the classes was tested.

Later in the project, not every added method was tested with unit tests. The reasons for this were that they were updated versions of previous methods or their testing could be done through the user interface.

During the implementation of the user interface, the program was tested through the user interface itself. As the project progressed, the methods of the tested classes were updated or removed. Manual testing of the user interface was performed to ensure proper handling of erroneous or missing data.

During the project, third parties were asked to test the program to improve the user experience and identify potential issues.

The testing plan did not deviate from the original plan.

Overall Assessment of the Final Outcome

Overall, I have been very successful in implementing the project and have learned a lot. The user interface is clear and user-friendly. The program has been thoroughly tested, and all identified runtime errors have been fixed.

The class structure is well-defined. Each class clearly represents one element of the program and performs its tasks correctly.

The program is well-suited for making changes and extensions. I have organized the code into clear files, and the code itself is readable. There is also no harmful dependency between the backend and frontend, meaning that making different changes to the user interface does not break the program's logic.

Bibliography

1. Scaladoc. (2023). [Online] Available: <https://docs.scala-lang.org/style/scaladoc.html>
2. Joda-Time 2.12.5 API. (2023). [Online] Available: <https://www.joda.org/joda-time/apidocs/index.html>
3. ScalaTest. (2023). [Online] Available: https://www.scalatest.org/user_guide/writing_your_first_test
4. StackOverFlow. (2023). [Online] Available: <https://stackoverflow.com/>
5. ScalaFX API. (2023). [Online] Available: <https://www.scalafx.org/api/8.0/#package>
6. Astronomical Society of South Australia. (n.d). [Online] Available: <https://www.assa.org.au/edm>
7. Wikipedia. (2023). Juhannus. [Online]: Available: <https://fi.wikipedia.org/wiki/Juhannus>