Habit Tracker

Software Entwicklung 2 (113217), Sommersemester 2022

Bestvater, Tom (tb173)
Beutel, Julius (jb266)
Riester, Samuel (sr185)
Singer, Steffen (ss546)

Link to the repository
(https://gitlab.mi.hdm-stuttgart.de/ss546/habit-tracker)

1. Description

Our project is a combination of a habit tracker and a journal app. The goal of our app is to help people to live better lives. The idea is that by doing simple positive habits every day, life becomes much better in the long run.

When starting the program, you'll see all the main functions. We wanted to create a simple utility app which is easy to use and where the user can see all useful information at a glimpse. On the left you have the "Habit" section, on the right your daily journal. You can add a habit by clicking on the "Add a new Habit" button. A new window will pop up, where you can name your new habit and set the days where you want to accomplish the habit. As soon as you click on "OK", the habit will be added to your main screen. The days where you haven't accomplished the habit (yet) will be colored red. Accomplished ones will be marked green.
When you want to remove a habit, just click on the "Remove Habit" button and delete the habit from the table below. To exit the "delete mode", click on the "Remove Habit" button again. To see your progress over the last weeks, click on the arrows above the habit - table and skip through the weeks.
On the right side of the screen is the "Daily Journal ". Here you can write a short paragraph (no more than 200 Words) to keep track of your thoughts or things you've done over the day.

2. Starting class

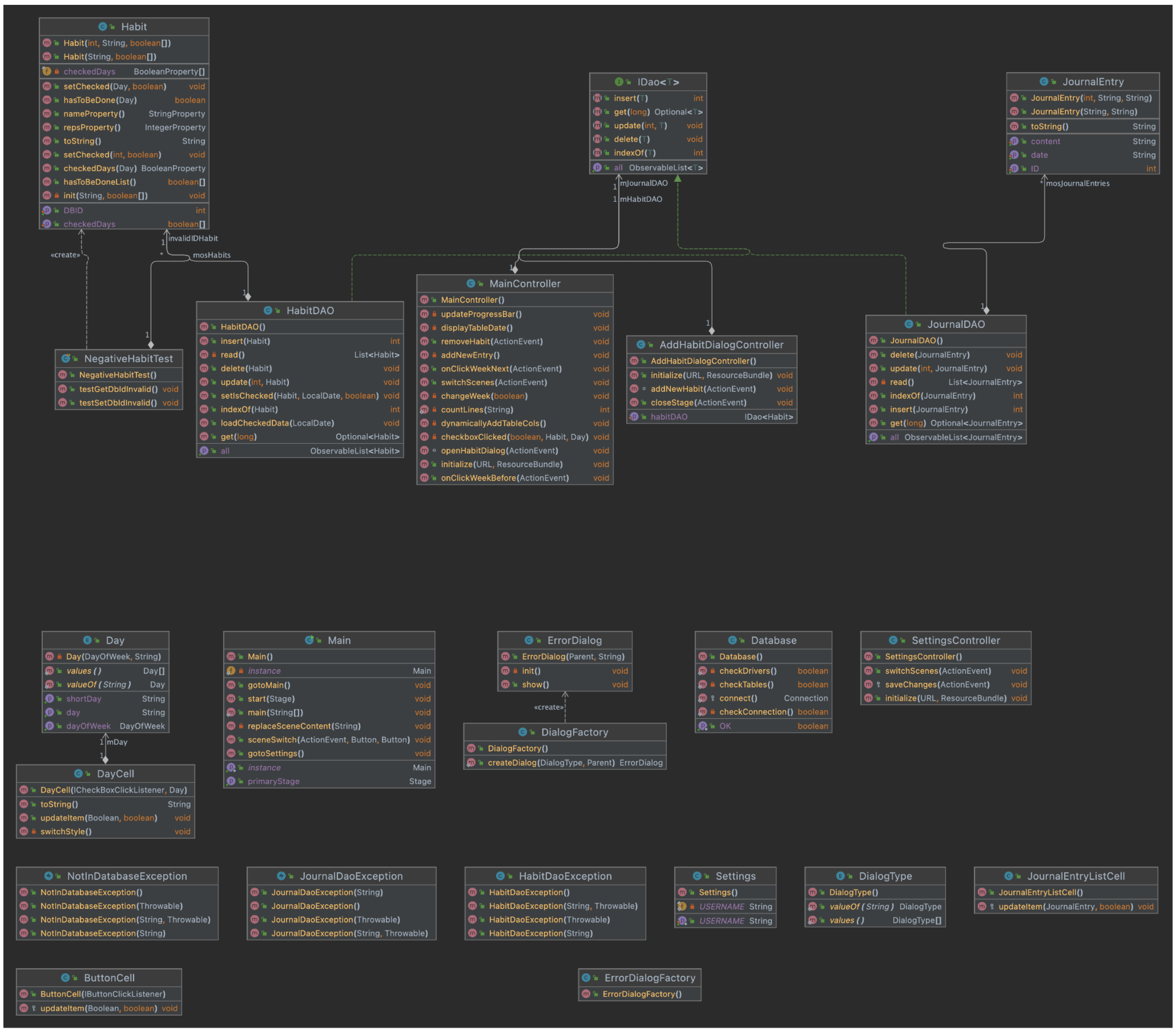
The main method can be found in *Main.java*.

3. Parts that do not work

If you tick a gray checkbox, it remains incorrectly ticked when you change the weeks. This can be fixed by restarting the program or by switching the weeks back and forth.

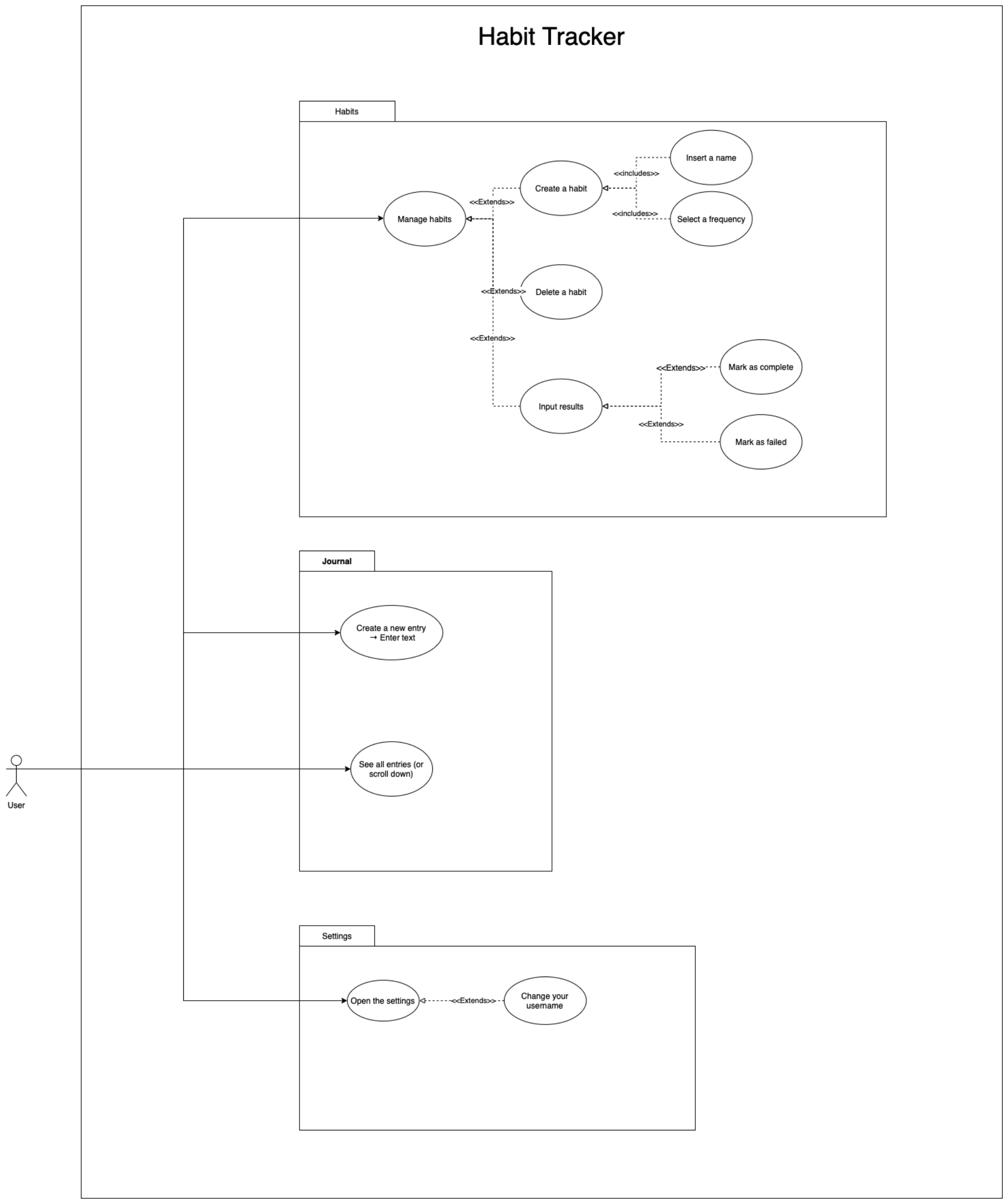
4. UML

Class Diagram


Habit
Habit(int, String, boolean[])
Habit(String, boolean[])
checkedDays BooleanProperty[]
setChecked(Day, boolean) void
hasToBeDone(Day) boolean
nameProperty() StringProperty
repsProperty() IntegerProperty
toString() String
setChecked(int, boolean) void
checkedDays(Day) BooleanProperty
hasToBeDoneList() boolean[]
init(String, boolean[]) void
DBID int
checkedDays boolean[]
«create»
1 invalidIDHabit
* mosHabits
1
1
NegativeHabitTest
NegativeHabitTest()
testGetDbIdInvalid() void
testSetDbIdInvalid() void
HabitDAO
HabitDAO()
insert(Habit) int
read() List<Habit>
delete(Habit) void
update(int, Habit) void
setIsChecked(Habit, LocalDate, boolean) void
indexOf(Habit) int
loadCheckedData(LocalDate) void
get(long) Optional<Habit>
all ObservableList<Habit>
IDao<T>
insert(T) int
get(long) Optional<T>
update(int, T) void
delete(T) void
indexOf(T) int
all ObservableList<T>
1 mJournalDAO
1 mHabitDAO
1
MainController
MainController()
updateProgressBar() void
displayTableDate() void
removeHabit(ActionEvent) void
addNewEntry() void
onClickWeekNext(ActionEvent) void
switchScenes(ActionEvent) void
changeWeek(boolean) void
countLines(String) int
dynamicallyAddTableCols() void
checkboxClicked(boolean, Habit, Day) void
openHabitDialog(ActionEvent) void
initialize(URL, ResourceBundle) void
onClickWeekBefore(ActionEvent) void
1
AddHabitDialogController
AddHabitDialogController()
initialize(URL, ResourceBundle) void
addNewHabit(ActionEvent) void
closeStage(ActionEvent) void
habitDAO IDao<Habit>
JournalEntry
JournalEntry(int, String, String)
JournalEntry(String, String)
toString() String
content String
date String
ID int
* mosJournalEntries
1
JournalDAO
JournalDAO()
delete(JournalEntry) void
update(int, JournalEntry) void
read() List<JournalEntry>
indexOf(JournalEntry) int
insert(JournalEntry) int
get(long) Optional<JournalEntry>
all ObservableList<JournalEntry>
Day
Day(DayOfWeek, String)
values () Day[]
valueOf (String) Day
shortDay String
day String
dayOfWeek DayOfWeek
1 mDay
1
DayCell
DayCell(ICheckBoxClickListener, Day)
toString() String
updateItem(Boolean, boolean) void
switchStyle() void
Main
Main()
instance Main
gotoMain() void
start(Stage) void
main(String[]) void
replaceSceneContent(String) void
sceneSwitch(ActionEvent, Button, Button) void
gotoSettings() void
instance Main
primaryStage Stage
ErrorDialog
ErrorDialog(Parent, String)
init() void
show() void
«create»
DialogFactory
DialogFactory()
createDialog(DialogType, Parent) ErrorDialog
Database
Database()
checkDrivers() boolean
checkTables() boolean
connect() Connection
checkConnection() boolean
OK boolean
SettingsController
SettingsController()
switchScenes(ActionEvent) void
saveChanges(ActionEvent) void
initialize(URL, ResourceBundle) void
NotInDatabaseException
NotInDatabaseException()
NotInDatabaseException(Throwable)
NotInDatabaseException(String, Throwable)
NotInDatabaseException(String)
JournalDaoException
JournalDaoException(String)
JournalDaoException()
JournalDaoException(Throwable)
JournalDaoException(String, Throwable)
HabitDaoException
HabitDaoException()
HabitDaoException(String, Throwable)
HabitDaoException(Throwable)
HabitDaoException(String)
Settings
Settings()
USERNAME String
USERNAME String
DialogType
DialogType()
valueOf (String) DialogType
values () DialogType[]
JournalEntryListCell
JournalEntryListCell()
updateItem(JournalEntry, boolean) void
ButtonCell
ButtonCell(IButtonClickListener)
updateItem(Boolean, boolean) void
ErrorDialogFactory
ErrorDialogFactory()

You can also find the class diagram in the GitLab repository.

Use Case Diagram



You can also find the use case diagram in the GitLab repository.

5. Statements

Directory Structure

Within the main package org.teampingui you'll find the *Main.class*. There are also several subdirectories. The classes are sorted according to the Model View Controller pattern (MVC).

- **Controllers**: Contains the Controller classes → Combines the Models with the Views
- **Models**: Contains the model classes for the habits, journal, settings, the weekdays enum and error messages with the error message factory
- **DAO (Database Access Object)**: Contains every class related to the Database
- **Interfaces**: Contains our Interfaces for the Dao and listeners
- **Exceptions**: Contains our own exceptions
- **Resources**: Contains our FXML scripts (View), the SQLITE Database and the CSS

Architecture: 3/3

We ensure extendability by …

- using the MVC pattern with meaningfully named packages,
- using the DAO pattern,
- using our own interfaces,
- using enums,
- using inheritance
- and using (our own) factories.

Clean Code 3/3

We use few static methods, our getters never return writable references to members. Our core classes use few to no public members. Furthermore, we have loose coupling by using our own IDAO Interface for every instance of the database.

Documentation 3/3

We have a complete documentation according to the requirements. We even wrote a small *readme.txt* and somewhat of a javadoc for complex code.

GUI (JavaFX): 3/3

We implemented a relatively complex GUI with a nested layout, multiple pages, pop-up windows and a menu.

Logging/Exceptions 3/3

We log a whole range of useful information and use appropriately meaningful levels. Our logs get saved in a separate file called *logs.log*. We also log exceptions and a thread.

log.debug → For common information (e.g. start of methods)
log.error → When smaller errors occur (e.g. parts of the GUI couldn't be loaded)
log.fatal → Used when fatal errors happen which make the application unable to run (e.g. Database is not available)
log.info → Used at core functions or difficult functions for better error handling or general information

Exceptions: We wrote our own exceptions (e.g. “NotInDatabaseException”) to get individual error messages and manage individual error handling.

Streams and Lambdas 3/3

We implemented some lambdas and also have multiple streams. We use one stream to sort the journal entries by date so that the most recent entries appear first in the list.

With another stream, we filter out for a habit the days on which the habit must be performed and on which it has already been performed. Then we take the number of days filtered out and continue working with that number.

Tests 3/3

We have several classes with various tests that check the core functionalities (and some auxiliary functions) of our program. We also implemented some tests for minor functions and also implemented negative tests.

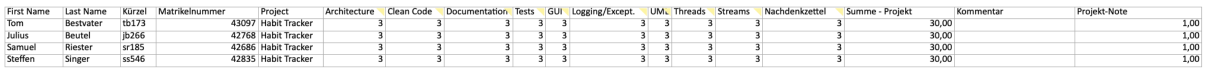
Threads 3/3

We used threads for showing multiple auto-disappearing error messages.
→ Implemented *thread.start()* in “ErrorDialog” class.

UML 3/3

We have a clear and correct class diagram with explanatory effect and a use case diagram with the core use cases.

6. Evaluation Sheet



First Name	Last Name	Kürzel	Matrikelnummer	Project	Architecture	Clean Code	Documentation	Tests	GUI	Logging/Except.	UML	Threads	Streams	Nachdenkzettel	Summe - Projekt	Kommentar	Projekt-Note
Tom	Bestvater	tb173	43097	Habit Tracker	3	3	3	3	3	3	3	3	3	3	30,00		1,00
Julius	Beutel	jb266	42768	Habit Tracker	3	3	3	3	3	3	3	3	3	3	30,00		1,00
Samuel	Riester	sr185	42686	Habit Tracker	3	3	3	3	3	3	3	3	3	3	30,00		1,00
Steffen	Singer	ss546	42835	Habit Tracker	3	3	3	3	3	3	3	3	3	3	30,00		1,00

This file is also available in our git repository (as *evaluationSheet.xlsx*).

7. Nachdenkzettel

We’ve filled out every Nachdenkzettel. Our “Nachdenkzettels” can be found in our git repository.