Java Juggernauts

HealthNCare App - Version 2.0

•••

Christian Berko
Riley Benson
Tryder Kulbacki
Nathaniel Pellegrino

Design Approach/Overview

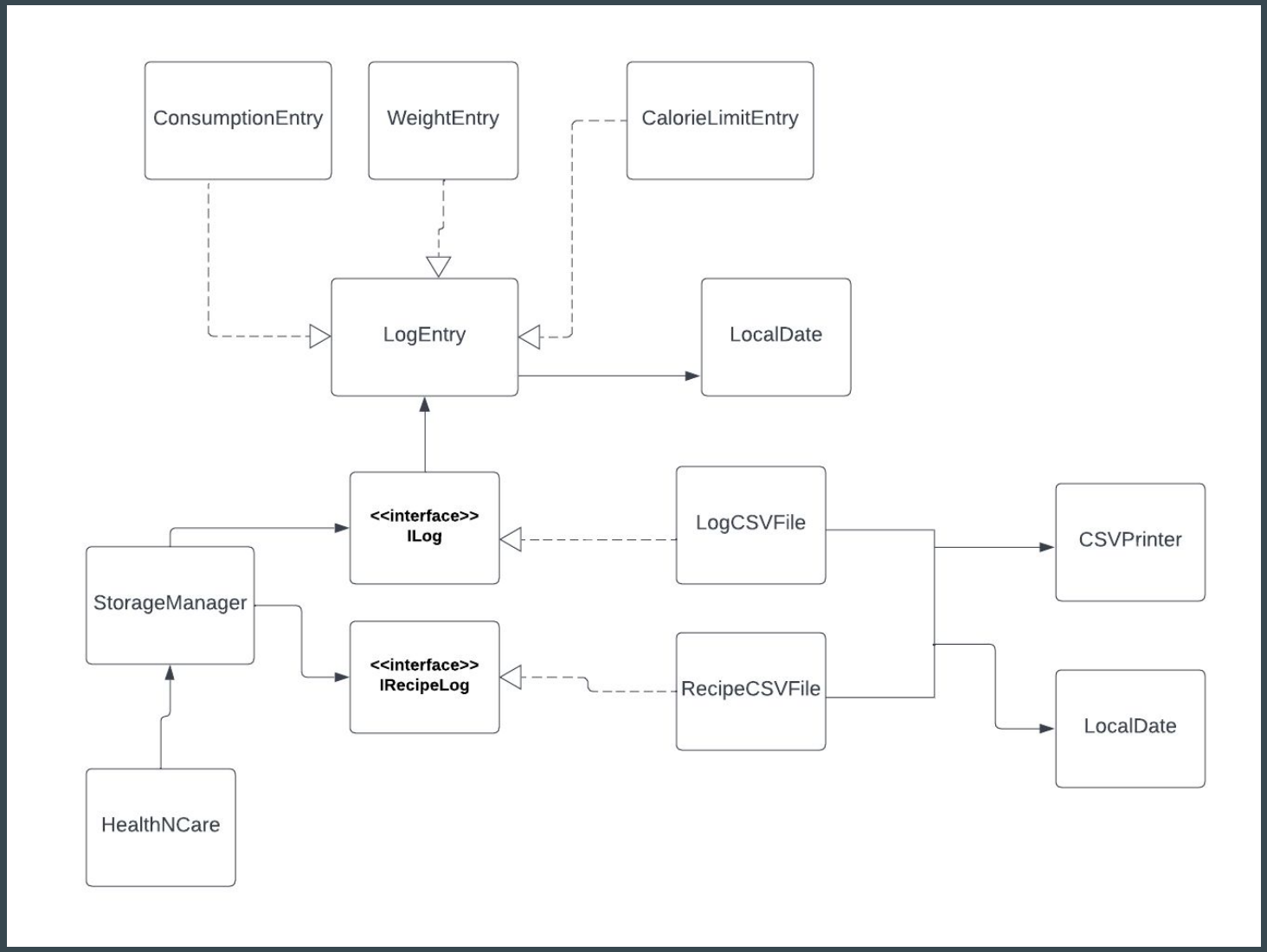
Based off feedback from the doctor we decided to...

- Use javax.Swing as the User Interface for this application, with AWT elements.
- Add a separate bar chart for Sodium, with a different scale than the nutrient bar chart (0mg - 2,300mg, instead of a 0-100% scale)
- Use an alternate list component instead of JTables for the Daily log, since they didn't like the feel of Excel. We chose JTrees because it offers an easy way to show a list of items, while referencing them when selected.
- Use a new Swing LookAndFeel library to improve the customer engagement and ease of access. We chose to use the *FlatLaf* library from *FormDev.com*, because of its cross-platform capability and modern design language.

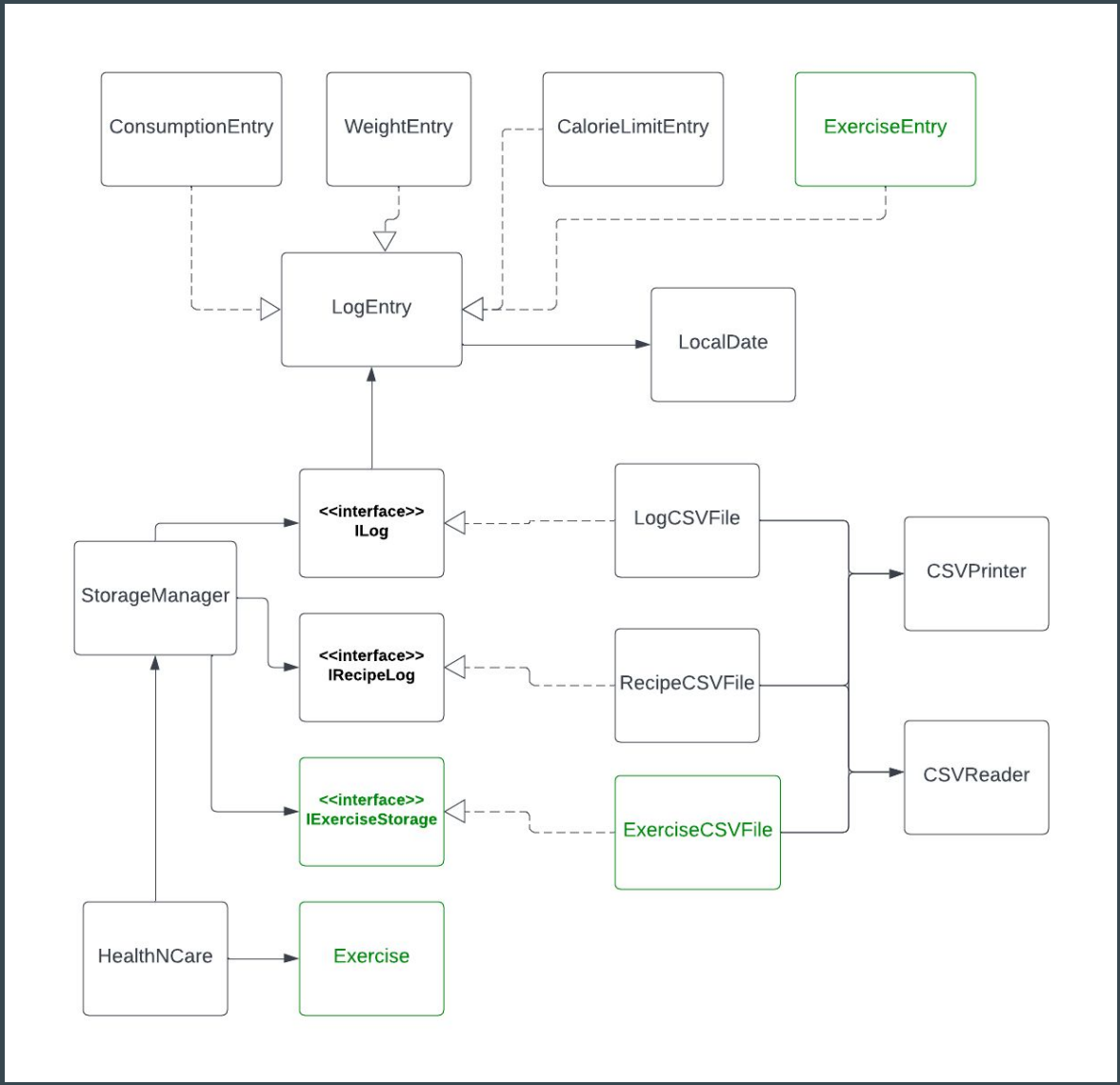
Guiding Principles

- DRY (Don’t Repeat Yourself):
  - Avoid code duplication
- Program to Interface:
  - Design code to interact with interfaces
  - Enhances flexibility and maintainability
- Dependency Inversion:
  - High-level & low-level; depend on abstractions, not each other
- Separation of Concerns:
  - Divide the system into distinct sections
  - Each section addresses a specific functionality for better maintainability and modularity

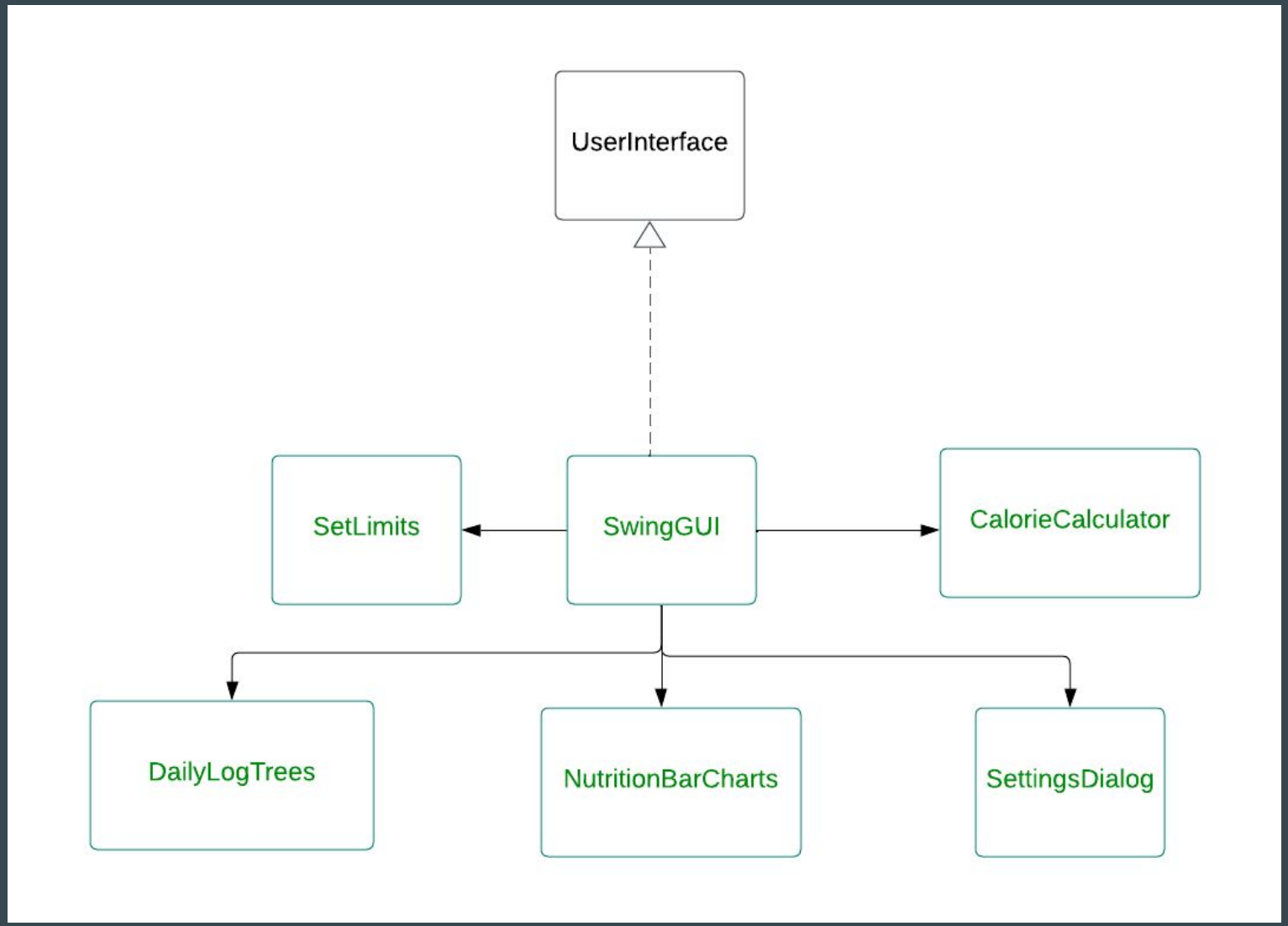
Model Before the Changes



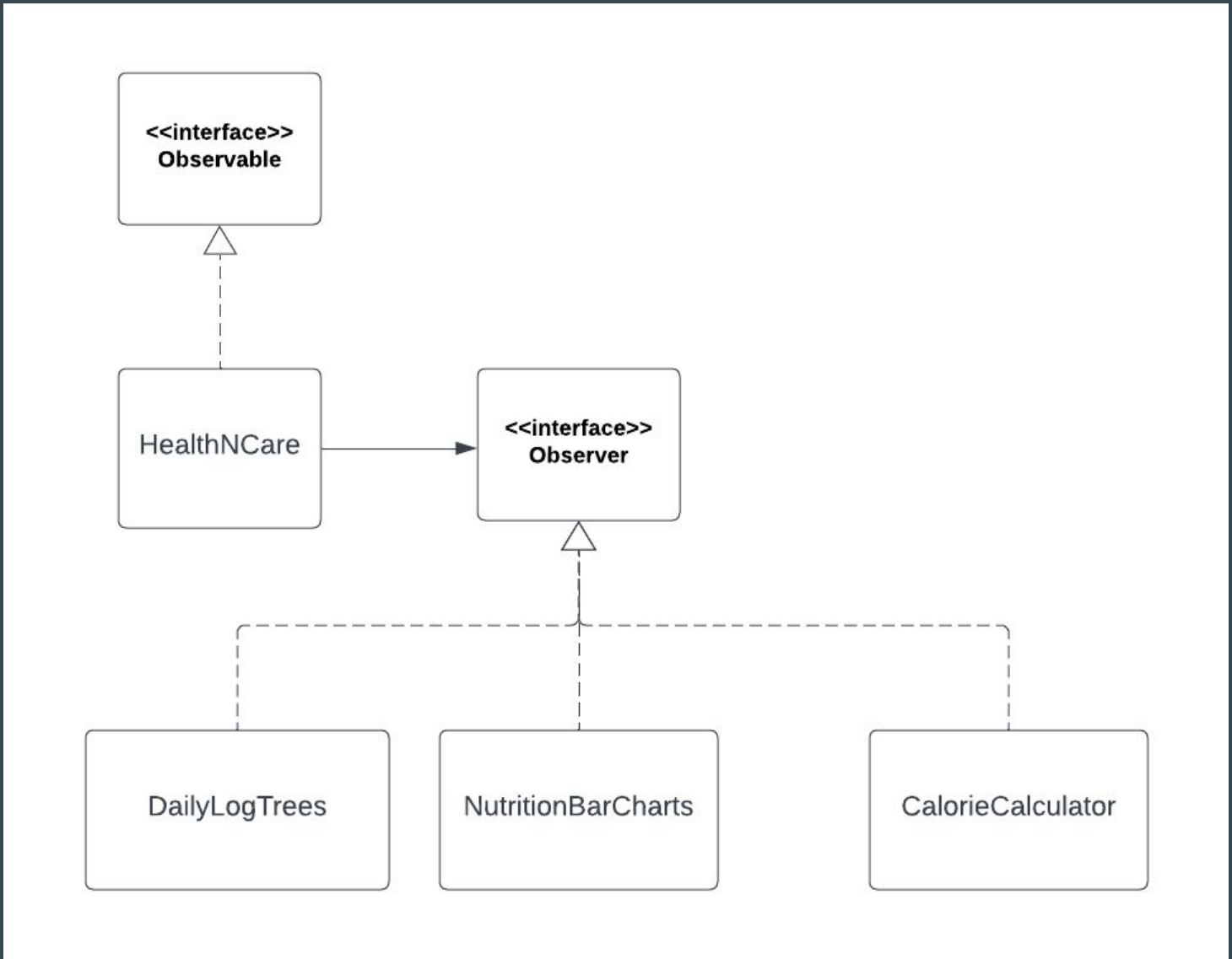
Changes to the Model



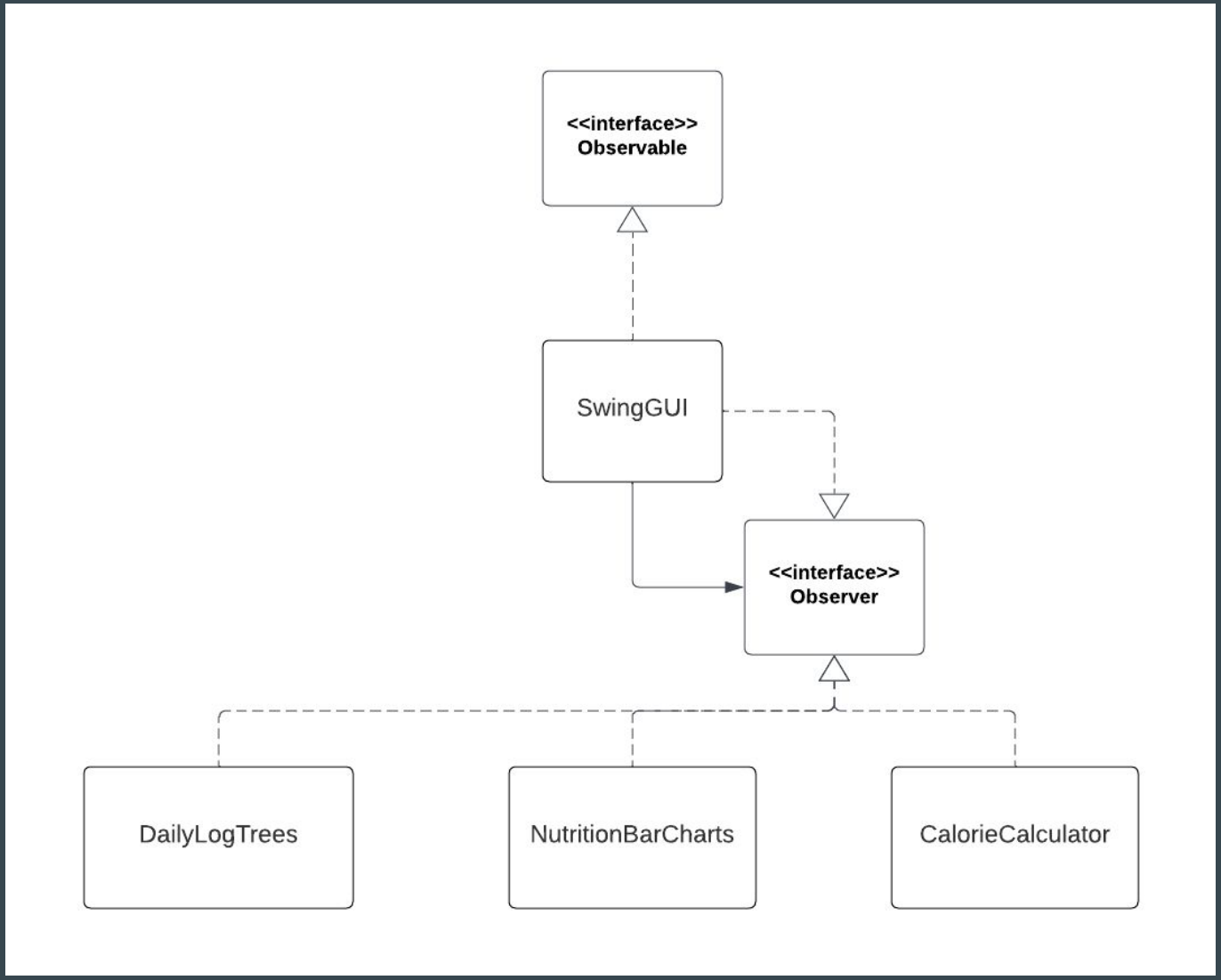
Changes to the View



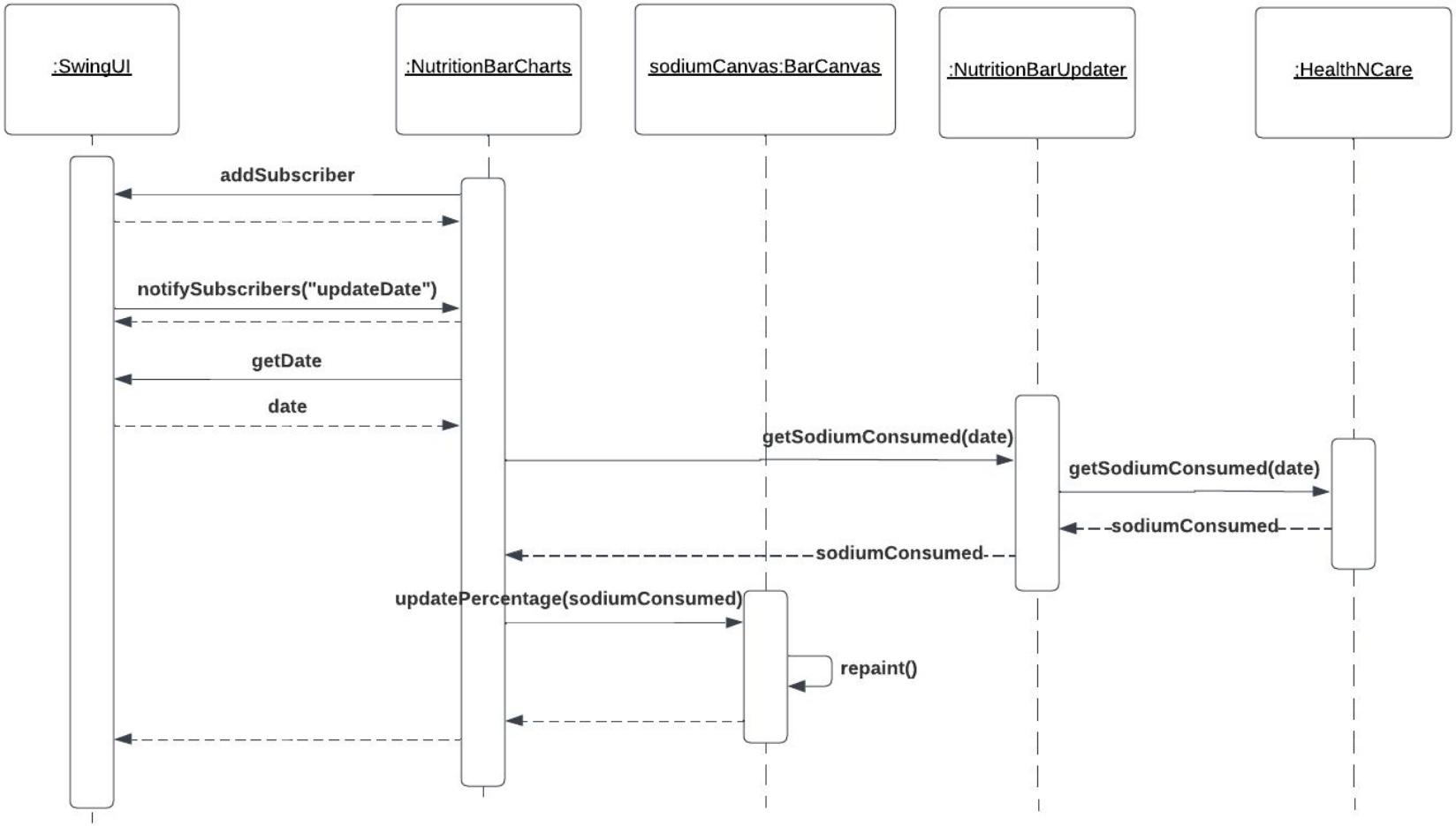
Observer Pattern - Model



Observer Pattern - View



Observer Pattern - View
SodiumCanvas



Rationale

- Observer Pattern Implementation:
  - Implemented observer pattern in the view to notify components of changes in the selected date.
- Architectural Pattern (MVC):
  - Used Model-View-Controller (MVC) for clear separation of classes, promoting extensibility, cohesion, and minimizing class coupling.
- Use of Observer Pattern for Automatic Updates:
  - Utilized Observer pattern for automatic view updates based on model changes, ensuring real-time data synchronization without constant manual checks.
- Observer Pattern for Date Selection:
  - Employed Observer pattern to notify elements of user-selected date changes, facilitating data retrieval for the chosen day.
- Composite Pattern for Part-Whole Relationships:
  - Adopted Composite pattern to represent recipes and food in part-whole relationships, promoting decoupling through a unified interface.

Areas Of Improvement

- Refactoring:
  - Removing repetitive code/classes
  - Cleaning up functions
  - Better separation of classes
  - More abstractions, to allow for better extensibility
- Incorporating more patterns:
  - Factory pattern
  - Mediator pattern
  - Composite (exercise routines)
- Better delegation of tasks:
  - Difficult due to time constraints with other projects
- Experience:
  - Little experience with SwingUI

Demo

Questions?