Name: SmartMirror
Team Members: Michael Gilroy, Lucas Laughlin, Nicholas Volpe

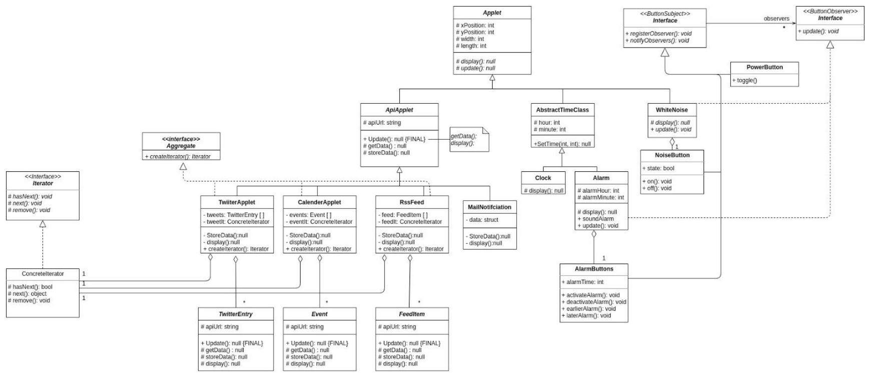
Final State of System:

- The view was implemented
- Time and weather display were implemented
- The Twitter, Google Calendar, and RSS feeds were implemented
- The Gmail use case was altered so that the mirror displays the number of unread emails
    - Easier to view the unread emails on the computer
- Power button was omitted because it was deemed unnecessary for the function of the mirror
- The alarm feature was omitted from the final version but was functional. The keyboard input was detectable but the link between keyboard input and alarm functions wasn’t working properly. Also a main loop that could execute generateHTML after key presses wasn’t functioning properly since the Keyboard Listener can’t be restarted after exit
- White noise wasn’t included since it needed keyboard functionality and the audio did not stream reliably

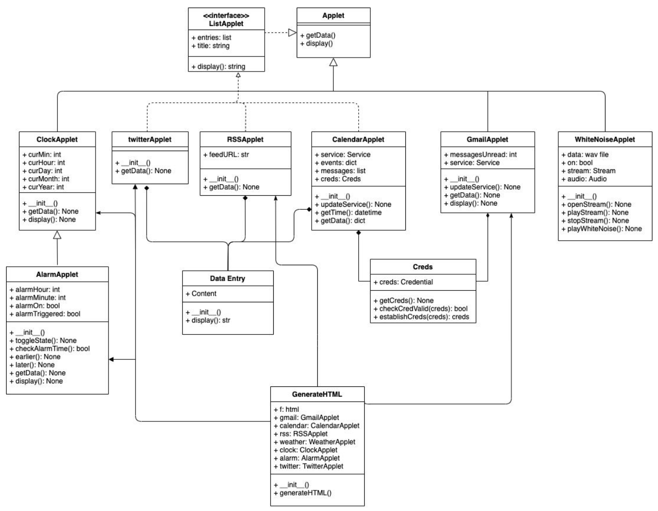
*** Report continued on next page**

Final Class Diagram:

Project 4 Class Diagram



Project 6 Class Diagram



- Personal credentials that were used for the Gmail, Google Calendar, and Twitter APIs were not included
- The applets for each API shared common attributes and methods, so a template class was made for the apps
    - The getData and display are the remaining methods in Applet
    - The ListApplet defines a common display used by the RSS, Google Calendar, and Twitter feeds
- The ListApplet child classes all contain an iterable list
- The power button feature was not implemented because it was deemed unnecessary for the project. The display could be turned off without the need for code.
- The observer pattern was implemented for use with buttons, but were not included since the keyboard functionality couldn't be implemented

Third Party Code:

Used libraries: feedparser, webbrowser, datetime, pyaudio, pygame, requests, requests-oathlib, responses, pytest, pytest-cov, pytest-runner, mccabe, six, coverage, coveralls, codecov, check-manifest, tox, tox-pyenv, pycodestyle, pynput

https://programminghistorian.org/en/lessons/creating-and-viewing-html-files-with-python

- Example for how to construct HTML view with Python

https://developers.google.com/calendar/quickstart/python

- Authentication code and examples for Google APIs

https://getbootstrap.com/docs/4.4/getting-started/introduction/

- Used to create the view

https://stackoverflow.com/questions/24352768/python-key-press-and-key-release-listener

https://developers.google.com/web/fundamentals/native-hardware/fullscreen

- Used to make the view full screen on the Raspberry Pi

Statement on the process

As the project was implemented, the team realized that some of the design patterns originally selected would actually make the features more complicated to implement. Simply using patterns for the sake of making code object oriented was not conducive to a functional project. More suitable design patterns were found after a majority of the code was written, so the project was a large learning experience in terms of design. A larger amount of time spent in the design phase would yield a more carefully thought out plan, and it would also provide a better sense of what patterns each feature could use.

A more effective approach would have been to create all of the features and ensure their functionality. Then using the remaining time to refactor the code into classes and create abstractions where they are applicable. This process would also include a systematic approach to encapsulating the data into classes that performed each feature. Refactoring was performed at the end, but with more time a thorough sweep could have been done through the code

The most difficult part of the process was finding a solution to updating the html. The first few solutions that we attempted to implement were not efficient and turned into ‘dead ends’. The final method of building the html strings with Python was much simpler. We were able to dynamically generate the view with API data.