Thor Breece, Zach Wrubel
5-3-23
CSCI 4448
Project 7

Airport Simulation

[Airport Simulation Repository Link]

Final State of System Statement

A paragraph on the final state of your system: what features were implemented, what features were not and why, what changed from project 5 and 6.

The patterns we used in the end are, in no particular order; the singleton, factory, strategy, observer, & decorator patterns. We used **singletons** to make sure specific objects (factories, airlines, loggers, Date, AirTrafficControl & Simulator) only had a single instance. We used **factories** to generate specific objects and handle their storage, tracking, & accessing. We used the **strategy** pattern to assign airlines to flights and manufacturers to planes. We used the **observer** pattern to track and log individual flights and tickets to external json files. We used the **decorator** pattern to change a normal ticket to a first class ticket if it was needed.

The big feature we were unable to implement was the terminals. This used the decorator pattern to transfer ownership/oversight of a specific terminal when an airplane belonging to a certain airline docked with the terminal. This feature ended up being a little too complex for the time we had remaining to finish everything. We ran into issues with the procedures for how we paired flights, terminals, and airports together. This issue probably could have been avoided with more detailed planning because the issue was that we tried to assign terminals to flights one flight at a time which would have been easier if we did it simultaneously with multiple flights at a time. However not implementing this didn't affect how the program functions & runs in the end.

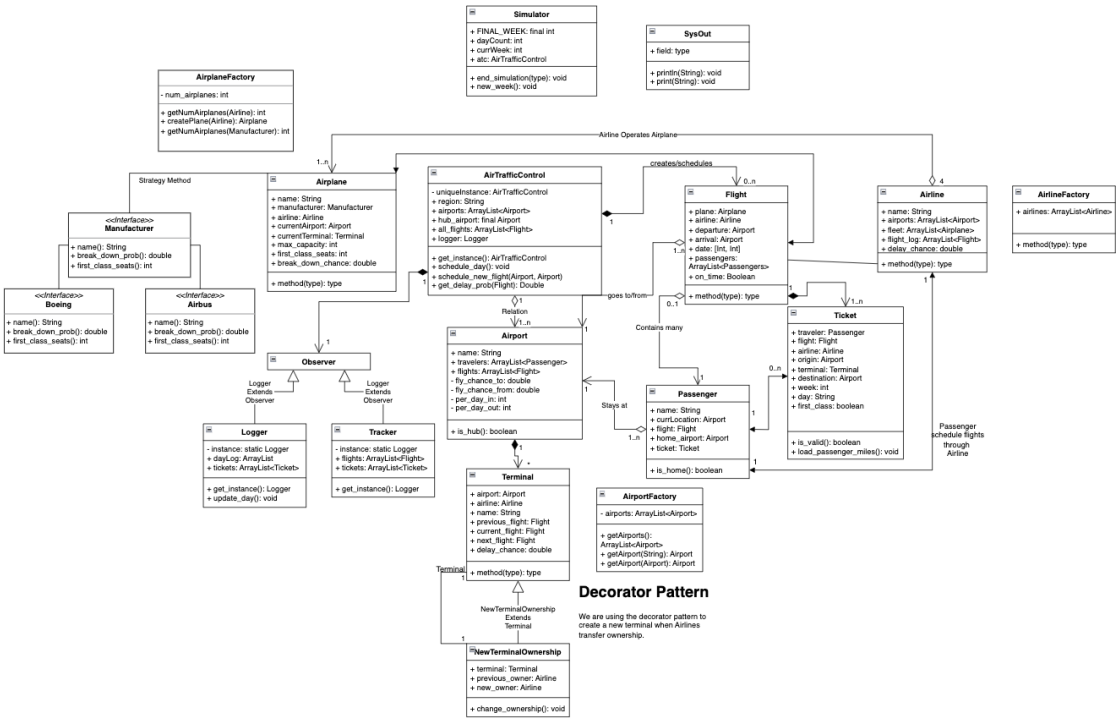
We decided to leave a lot of the code in the file since the implementation was close, doesn't cause errors when run as it is now, and still contains a lot of good work for the decorator pattern. Hopefully making it easy to understand what we were attempting to implement.

We made it easy for the user to interact with the program with a makefile and a Jupyter Notebook that can be run to easily enter input to the program. The program makes a passenger object for the user and allows the user to book flights available to them.

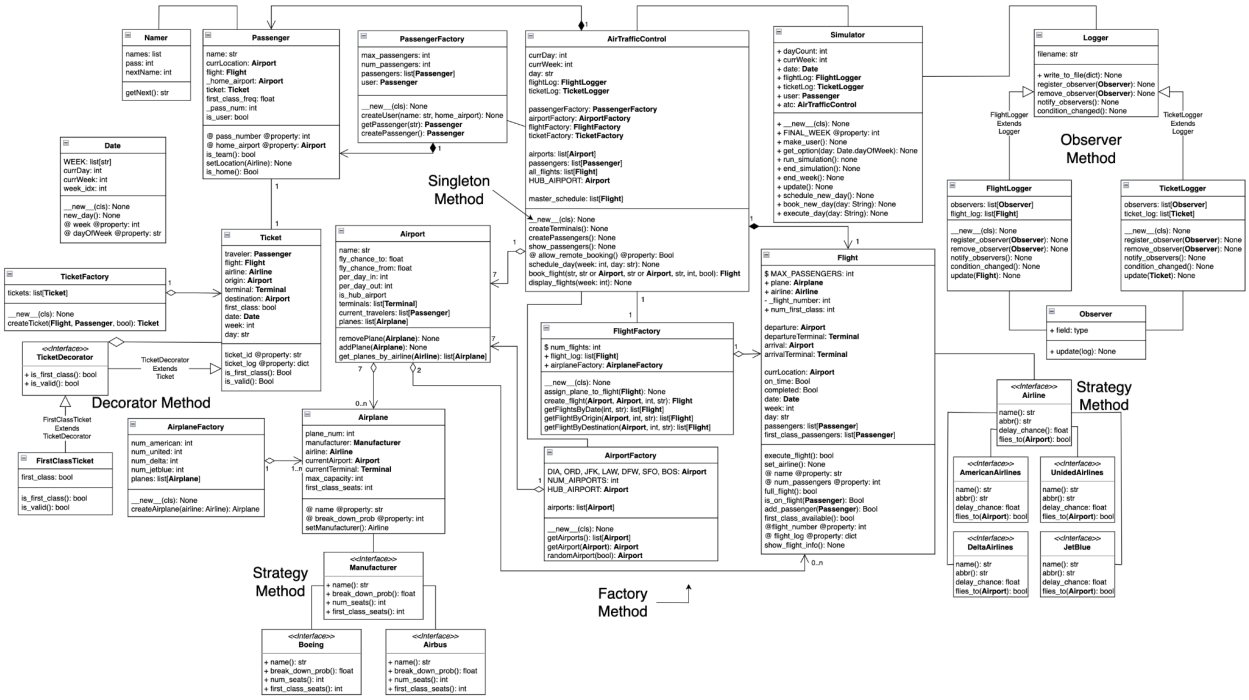
Final Class Diagram and Comparison Statement

- A thorough UML class diagram representing your final set of classes and key relationships of the system
- Highlight and document in that diagram any patterns that were included (in whole or part) in your design
- Include the class diagram submitted in Project 5, and use it to show what changed in your system from that point into the final submission
- Support the diagrams with a written paragraph identifying key changes in your system since your design/work was submitted in Projects 5 and 6

Project 5 Diagram



Final Project 7 Diagram



Patterns are indicated in bold pointing to or next to their implementation(s).

Changes between Projects 6 & 7

- Airlines are now singletons
- Reorganization of some variables/trackers between classes. (Ex. Moved airplaneFactory from AirTrafficControl to flightFactory)
- Decorator pattern implemented for converting Tickets to FirstClassTickets.
- Strategy Pattern
    - Used for airlines within flight object
    - Used for setting airplane manufacturer within Airplane object
- While it doesn't show in the diagram, we removed the terminals which used the decorator pattern. This is described more in the beginning of this document.
- Implementation of user interface. Not really shown in diagram except for make_user() method and user object within Simulator, and is_user boolean within Passenger.
- Implementation of Airplane and Airplane Factory
- Other minor edits/additions

Third-Party code vs. Original code Statement

- A clear statement of what code in the project is original vs. what code you used from other sources – whether tools, frameworks, tutorials, or examples – this section must be present even if you used NO third-party code - include the sources (URLs) for your third-party elements

Most of the code in the project is our original code. The project was originally roughly based on the FNCD from earlier in the semester but is almost entirely different now. The only code in the project that is third party is the singleton implementation. I sourced this from the lecture slides (lecture 16) and made a slight modification by removing one line of code.

Our implementation of the observer pattern is heavily based upon the implementation of the pattern in lecture 12 (references github repo here), but not copied from the repo. We've adapted this implementation for our use and wanted to note it here.

Statement on OOAD process for overall Semester Project

- List three key design process elements or issues (positive or negative) that your team experienced in your analysis and design of the OO semester project.

1. Multiple times during development we encountered challenges when determining when to use the strategy or the decorator method. We usually ended up going with the strategy pattern. We thought it would be a good idea to have the airlines be their own class that also tracked multiple elements like the flight and airport classes do, but we eventually found it would unnecessarily clutter the code and that a simple strategy pattern implementation would work best for this use case.
2. The biggest issue we faced was implementing the terminals to link them together with flights. We used the decorator pattern to do this and couldn't find a way to make it work in time to submit so we cut this portion from the project.
3. One of the big issues we had was deciding if we should have the AirTrafficControl focus on any flight between any airport or just focus on a single 'hub' airport. We eventually choose the latter as it would have been significantly more complicated to track all flights between all the airports. This didn't affect our use of design patterns since we felt we already had enough patterns we were using that we were happy with.