Jake Felzien
A02019985
OO
HW Assignment 2

For this homework assignment I made the decision to use the OO capable language python. The main reason for this decision was due to the nature of the GUI's with this assignment. Between Java and C#, I am much more familiar with C#, however I do not have access to a windows machine to generate window forms, and I have little to zero experience with Java in general. Thus, having done a python GUI for a data science project before, I decided to give it another try. I believe it worked quite well.

To begin with, I needed to generate my own data simulation. While at first appearing tedious this allowed me to make a unique approach to my design implementation (**Refer to my UML diagram for full implementation details). The main difference in approach was that I decided the need for an athlete class was too granular and I didn't think the class was necessary to produce a much more flexible model. Instead, I generate a Subject class that was responsible for being the data source as well as the simulation. The Subject class read in input from the simulation file, and spun up the background simulation thread. This allowed me to create one main observer class that consumed the data generated. To see my full interaction diagram see the attached interaction diagram based on Dr. Clyde's figure 03**.

What I enjoyed about the design implementation was that I created a single Subscriber class. This subscriber class was able to be registered from by the Subject data source, and thus any time the data source updated, the accompanying Subscriber classes received these updates. With this I was also able to use the Subscriber class as a strategy pattern and every other observer pattern class inherited the strategies associated with a Subscriber. These included my emailObserver, guiTicker, etc. These classes all were the Subscriber strategy and were able to consume the data with updates from the Subject registration, but then with that were able to have their own unique properties as well and were able to link to different portions of the GUI to all for flexible updates and flexible modularity within the GUI.

Finally, the decorator pattern was used on my streamer observer class. With decorations, checkboxes are able to be selected with this observer that allows for dynamic changes to the GUI and data displayed. This particular observer views a constant stream of all updates and through the use of the decorator pattern I was able to generate additional columns within this window of the GUI.

Here is a list of all of the observers created and their functionality:

- emailObserver - sends text messages to my phone number for updates on a specific athlete/jersey number. Email and password is required.
- guiTicker - the graphical representation of the athlete in respect to the overall distance selected (thus if a larger distance is selected then the athlete ends up travelling, the ticker will not travel the entirety of the window). The GUI follows the specific individual that was selected.
- listObserver - displays athlete data updates in a list like fashion

- topRacersObserver - displays the top 10 racers jersey numbers in respect to who has travelled the furthest distance. This is updated anytime that the ranking of the top ten distance is changed in real time with the updates recieved.
- streamer - displays real time updates for everyone in the race
    - decorator - adds a column to the streamer
    - decoratorTwo - adds another column to the streamer

As far as GUI window choices and implementation is concerned the following design decisions were made. The opening home screen allows for the user to enter the distance they wish to observe to as well as name the race and enter other aesthetic data. Once, ready, they can select the Start Race button. From here, another window will pop up with all of the selection choices for the various class and observer selections. The user can select one or several observers depending on their needs. Recommended use is to select one at a time, as each new observer generates a new window for the user to interact with and experience the data flow. For an email observer the email and password is required as is a jersey number for the other cases for a window to be generated.

Once selected, the “Generate Observers” button can be selected. This will generate the various observer patterns and will display their functionality. Various options exist for each, or viewing options are the only ones once completed. Every page has an “Unsubscribe” button that can be selected to close the window and also unsubscribe the given observer from the subject. This button works two-fold, and thus I disable the displayed x-button to allow for the correct unsubscription to take place. To kill the program, return to the main page and select the red “Quit” button and the program will end.

Also, see the testCases.py file to run and generate all of the test cases written for this project. To run the test cases, run the test case file, and immediately kill the gui with the red quit button, and the test cases will run and display their results.