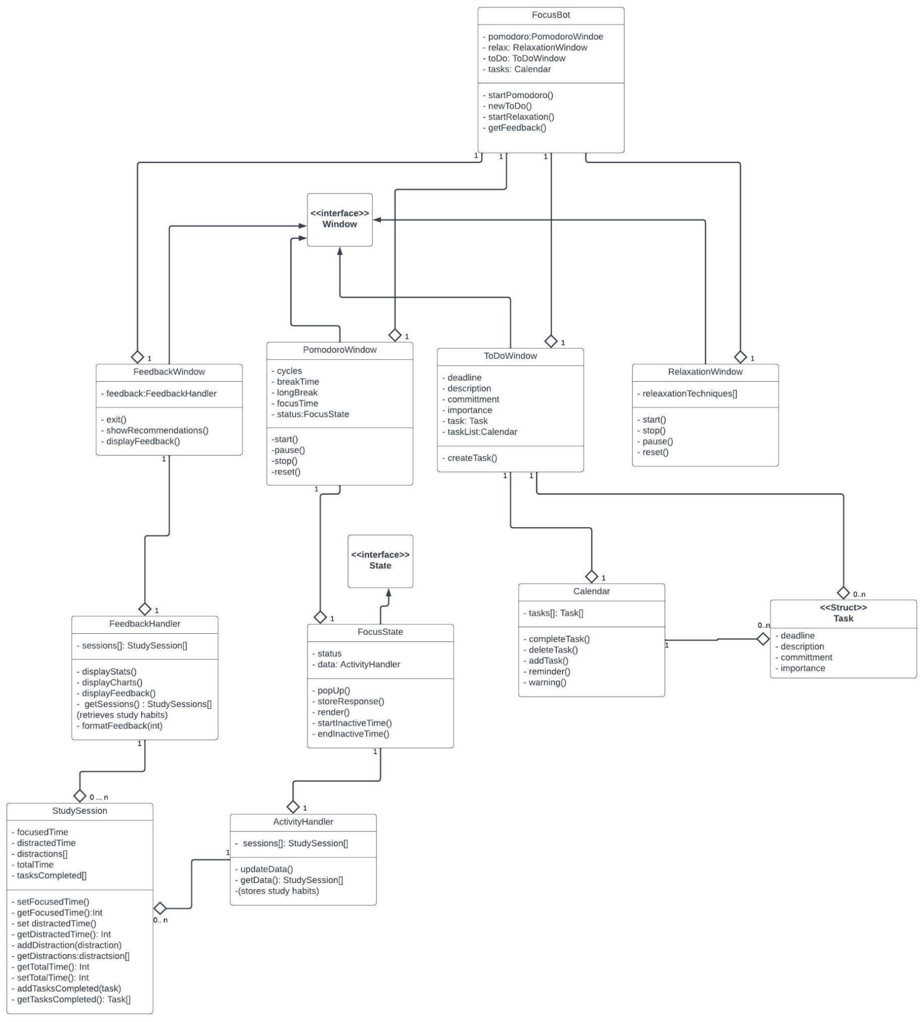
Low-level Design PM3 CS25

For this project, we believe that the best design pattern family would be the behavioral one. The reason why is that it greatly relies on status changes, inputs, data changes, and different objects having different and separate responsibilities. This much is clear when looking at the rough class diagram for the FocusBot. This is the entire diagram with all the relationships and a general and rough display of the kind of methods and functions each class or object could carry out:



There is a very evident lack of parameters in a few of the classes. The reason behind this is that the fields in most of them are set through the input given by the user when they are created

Two main design patterns this diagram displays are Chain of Responsibility and State.

Here is an example where State is displayed:

```
public void popUp()
{
    if (user leaves window) or (goes idle)
    {
        startInactiveTime();
        status = distracted;
        alert("Do you wish to continue studying") //Like the react popup
        if (user clicks yes)
        {
            popUp goes away
            timeInactive += endInactiveTime();
        }
        else
        {
            timeInactive += endInactiveTime();
            data.getData().setDistractedTime(timeInactive);//stores distracted
data to the field in the handler
            data.updateData(); // the handler then does any necessary adjustments
to the  study session object that itself and the feedback
            //handler have access to
        }
    }
}
```

All these actions are triggered automatically by the FocusState object when the pomodoro cycle is set as active in the FocusBot. It only happens when the user leaves a window or is away from keyboard for a certain amount of time. In other words when an event like onPointerLeave or a status that determines if a user is idle or not Is triggered. Otherwise, this method should not be called at all. Realistically, it should be implemented with an event handler to achieve this goal.

Here is an example where chain of responsibility is displayed:

For the FocusBot we have two different handlers that deal with the data storage and retrieval for different study sessions. These are used for feedback and are modified depending on what the user does during a study session (like a pomodoro cycle). How long the user gets distracted, what distracts the user, time spent studying and total study session time are some of the aspects stored in a StudySession object. The handlers responsible for keeping the sessions up to date and keeping these session readily available for the FocusBot are the Activity and Feedback handlers. This would be a perfect example of chain of responsibility in this project as the responsibilities both handlers have are very clearly outlined and even their uses of the same class is differentiated. The feedback handler is more closely related to processing as it uses the information and provides it to the Feedback window when necessary. The activity handler is more closely related to passing as it gets new information, and passes it to the appropriate field in the StudySession object. How the ActivityHandler deals with its responsibilities is, roughly, outlined in the previous example related to state. The way Feedback handler deals with its responsibilities could be described as follows:

```
public void displayFeedback(int days)

{

    //Get relevant fields from StudySession and display them in a simple

    //and user friendly interface

    //assume Handler takes care of displaying them in the right format

    sessions = feedback. getSessions();

    for (int i = 0; i < days; i++)

    {

        feedback.formatFeedback(sessions[i]);//formatFeedback will deal with not only retrieving all the info through other functions in the feedback handler and the StudySession object, but also putting it inot the right format for display and navigation

        //sessions stores the very last session at the head of the array (better implementation with linked list most likely) and then goes

        //through the latest sessions until it reaches as many days back as the user specified.

        //filter would allow for much more specific feedback and session selection.

    }




}
```