Refactors

Code Smell addressed	Description of code prior to refactoring	Description of code after refactoring
Feature envy	The login scene had many features/methods that could've been combined into a class. These features included: string parsing/ checking if the string is valid (characters, and not empty), and sending error/success messages.	The authenticator class has all the aforementioned features and also handles finding the correct user based on the user and password. The login scene can now call an instance of the authenticator class to handle error and success cases.
Duplicated Code	Prior to refactoring, the User class had many methods which served the same function: ● public ArrayList<Attribute> getCopyOfAgePreferences(ArrayList<Attribute> agePreferences) ● public ArrayList<Attribute> getCopyOfSexPreferences(ArrayList<Attribute> sexPreferences) ● public ArrayList<Attribute> getCopyOfEnergyLevelPreferences(ArrayList<Attribute> EnergyLevelPreferences) ● public ArrayList<Attribute> getCopyOfSizePreferences(ArrayList<Attribute> sizePreferences) Despite the fact that the return types were the same, the main difference between these functions is that a different type of object had to be created. This is why these methods were duplicated in the first place.	After refactoring, these four methods were generalised into one method called getCopyOfPreferences(). The general functionality is the same, but to emulate the distinct different preference attribute objects being created, a new abstract method was created in the Attribute (parent) class called: cloneAttribute(); In each child class (Sex, EnergyLevel, Age, Size), this method was overridden to return the correct type. Thus, the same functionality was reproduced with 4x less code.
Data Clumps	Before refactoring, in each GUI scene file, there were clumps of code that repeated, like the navigation bar, the main container, style, instances of various scenes for navigation purposes and local data/objects that needed to be passed between scenes.	In order to remedy this, and prevent repeating the same clumps of code in all UI files, a new scene file was created, called “PrimaryScene”. This new UI file contains all the information that needs to be passed between scenes. To utilise it, all other scenes extend it in order to gain access to the information. A method called to initialise the PrimaryScene is called which sets all the variables and

		effectively passes the needed information.
Long Parameter List	Before refactoring, DogProfileScene had a DogProfileSceneController class connected to it, and running the background logic. The issue was that virtually every single attribute which belonged to DogProfileScene was being passed through to the controller. Only for the functionality of three methods!	To refactor, this extra controller class was removed, and the methods pertaining to it were transferred over to the DogProfileScene class. This way, the need to pass EVERY attribute to another class was removed, and the functionality stayed the same. The code was cleaned up, and now it is easier to read and more maintainable!
Duplicated Code	In the CalendarScene, there was a repeated code block for creating a calendar for the first time, and updating that calendar. The only difference between initialising for the first time and updating was that a few variables had to be set.	These code blocks were made into two new functions: createCalendar() and updateCalendar(). In updateCalendar(), we just call createCalendar() and some logic to refresh the page. In createCalendar(), the calendar is created (this is the block of code that was being repeated). In the start method, we were just able to createCalendar() and initialise the variables we needed at that time. Thus, we removed the duplicated code and made our code much more readable and easy to maintain.
Speculative Generality	Earlier on in the design/planning phase, our team had created a tagPreferenceWeight attribute, which was a ‘possible’ implementation for later. This feature was actually never used, and the naming was getting confused and messing up some of the logic we had!	This attribute and its getters/setters/constructors were removed since this feature was never going to be implemented in the first place. Some adjustments had to be done to some tagList copy functions, in order to account for this new change as well (change in constructor)