Project: **Smart Home Simulator**

Project Report - Deliverable 2

STUDENT NAMES:

Gregory Tucker - 40092432
Aymen Metallaoui - 40057294
Tyler Znjog - 40005987
Adrien Kamran - 40095393
DeepKumar Patel - 40096716

Tutorial Section HC

Github Repository: https://github.com/gregtuc/SmartHomeSimulator

Link to Demo: https://drive.google.com/file/d/1ETqH9e_gJsdXnzbog3Eecn5FDfTtQwcJ/view?usp=sharing

Table of Figures

Table of Contents

1.0. Project Scope

The primary purpose of the second deliverable was to give functionality to the core and security modules. This meant that the objects involved in the house simulation, such as windows, doors, and lights, had to be created in their entirety. Furthermore, a permissions system had to be created that would manage access to these components. This permissions system draws from a text file containing details of user profiles.While the simulator will require development with respect to the temperature module moving forward, the management of user locations and the creation of the security module should be approaching the fully developed stage as this deliverable is completed.

2.0. Design Patterns

Deliverable two made use of both creational and structural design patterns. Creationally, the Singleton design pattern was a key component that allowed communication between controllers and classes possible. In the Smart Home Simulator, there exists elements that must be shared between classes such as details about the Active User (name, type, current location). The creation of the Singleton ActiveUser class is shown below:

```
3   public class ActiveUser {
4       private static volatile ActiveUser instance = null;
5       private static String profileName = "";
6       private static String profileType = "";
7       private static String profileLocation = "";
8       private static String oldProfileLocation = "";
9       private static Boolean awayMode = false;
10      private ActiveUser() {}
11
12      //Only a singular instantiation of this class will be allowed.
13      //This method will return that instance.
14      public static ActiveUser getInstance() {
15          if (instance == null) {
16              synchronized(ActiveUser.class) {
17                  if (instance == null) {
18                      instance = new ActiveUser();
19                  }
20              }
21          }
22          return instance;
23      }
```

Figure 1: *ActiveUser Singleton class in src/models/ActiveUser.java*

At any given point in the operation of the application, there should only exist a singular instance of the ActiveUser class. This means that attributes contained in this class all point to the same value and will return the same static reference to any calling method or class. The

utilization of this design pattern was equally important in storing JavaFX elements on the home.fxml page that must be changed my classes other than the HomeController class. By creating a class UniversalElements and storing JavaFX elements on it, any change to these elements would change the original reference and in turn we would not be required to pass JavaFX elements as parameters in methods.

The structural design pattern used in this deliveral was Private Class Data. Security being a key component of the application, class attributes (not including FXML elements) were typically made private to limit access. Accessors and mutators were created to manage requests to change or access attribute values in classes. This creates a more-secure way of managing communication between classes.

Finally, the observer pattern was used for management of the security system. An interface Observer was created, with the applications Window, Light, Door, and Communication manager classes implementing it. Next, a class AlarmSystem was created that would subscribe each of these concrete classes and notify each of them to execute method alarm() in the event of a break in. This allows for a quick response among all the necessary components in the event of a break in. A snippet of the observer interface is shown below:

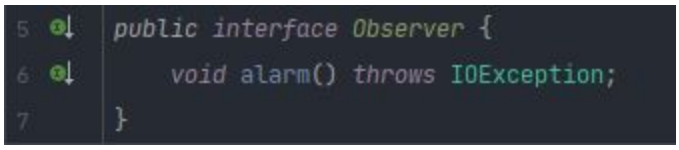


Figure 2: *Observer Interface in src/security/Observer.java*

A second screenshot is shown below of the implementation of the alarm method in the DoorManager class:

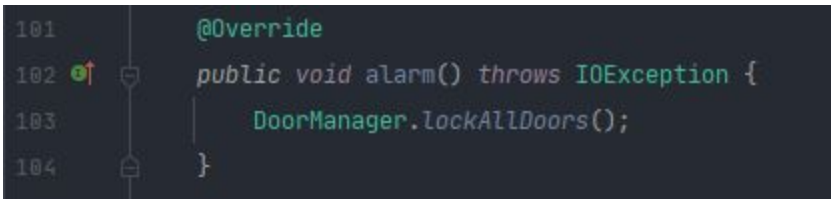


Figure 3: *Overriding alarm method in src/utilities/DoorManager.java*

3.0. Use Cases

3.1. Save a new profile.

Use case name	Save a profile
Level	User-Level
Brief description	Created accounts must be saved in a text file to avoid loss of information after closing down the simulator.
Preconditions	The user is on the simulator dashboard.
Triggering event	The user presses the “Profile” button.
Main flow	1. A window appears and the user enters the new account name in the available field along with the account type.. 2. The user submits the information and the window closes. 3. The information is saved in the text file and the user is logged into their account.
Extensions	During step 2 of the main flow, the user enters an invalid name: 1. An alert box appears informing the user of their error. 2. The user closes the alert box. 3. Continue from step 2 of hte main flow.
Postconditions	The users profile information is saved in the profiles text file. The user is signed in to their account.

3.2. Login to an existing profile

Use case name	Login to an existing profile
Level	User-Level
Brief description	Created accounts are saved in a text file to avoid loss of information after closing down the simulator. In order to access account information after closing the simulator, the user must be able to select their account from the text file.
Preconditions	The user is on the simulator dashboard.
Triggering event	The user presses the “Profile” button.
Main flow	1. A window appears and the user selects their account from the list of available profiles. 2. The user submits the information and the window closes. 3. The window closes and the user is logged into their account.
Extensions	During step 2 of the main flow, the user doesn’t select a profile:: 1. An alert box appears informing the user of their error. 2. The user closes the alert box. 3. Continue from step 2 of hte main flow.
Postconditions	The user is signed in to their account.

3.3. Change the Simulator's time speed

Use case name	Change the Simulator's time speed
Level	User-Level
Brief description	The simulator runs at a set speed which can be changed by the user. It can be sped up to see results faster or slowed down so that the user can observe all the changes.
Preconditions	The user is on the simulator dashboard.
Triggering event	The user presses the "Time" button.
Main flow	1. A window appears and the user inputs their preferred simulation speed 2. The user submits the information and the window closes
Extensions	During step 1 of the main flow, the user does not enter a speed or zero:: 1. The user submits the information and the window closes 2. The system will display the correct speed but when the simulation starts, it will set the speed to 1.0
Postconditions	The simulation speed has been changed

3.4. Open or Close a Door

Use case name	Open or Close a Door
Level	User-Level
Brief description	Door must be modifiable, as the security and climate modules must change them to restrict or allow changes in air temperature and the movement of people objects.
Preconditions	The user is on the simulator dashboard.
Triggering event	The user selects the SHC tab.
Main flow	1. The user clicks on the “Door” option in the item-selection box. 2. The user selects the which room will have the door altered. 3. The user presses either the open button or the close button. 4. The status of the door changes based on the users selection.
Extensions	During step 3 of the main flow, a door was selected that is already at the desired state: 1. An alert box appears informing the user. 2. The user closes the alert box.
Postconditions	The door is at the desired state.

3.5. Open or Close Windows

Use case name	Open or Close Windows
Level	User-Level
Brief description	Windows must be modifiable, as the security and climate modules must change them to restrict or allow changes in air temperature and the movement of people objects. Windows should also not be able to be opened if an object is interfering with their movement
Preconditions	The user is on the simulator dashboard.
Triggering event	The user selects the SHC tab.
Main flow	1.The user clicks on the “Window” option in the item-selection box. 2.The user selects the which room will have the window altered. 3.The user presses either the open button or the close button. 4.The status of the window changes based on the users selection.
Extensions	During step 3 of the main flow, a window was selected that is already at the desired state or is blocked by an object: 1.An alert box appears informing the user. 2.The user closes the alert box.
Postconditions	The window is at the desired state.

3.6. Turn On/Off Lights

Use case name	Turn On/Off Lights
Level	User-Level
Brief description	Lights must be modifiable, they could affect energy consumption and saving if they are left open for an unwanted amount of time, during daytime or if no one is present in the room.
Preconditions	The user is on the simulator dashboard.
Triggering event	The user selects the SHC tab.
Main flow	1.The user clicks on the “Lights” option in the item-selection box. 2.The user selects the which room will have the lights turned on/off. 3.The user presses either the open button or the close button. 4.The status of the lights changes based on the users selection.
Extensions	During step 3 of the main flow, a window was selected that is already at the desired state or is blocked by an object: 1.An alert box appears informing the user. 2.The user closes the alert box.
Postconditions	The light is at the desired state.

3.7. Set an Auto mode

Use case name	Set an Auto-mode
Level	User-Level
Brief description	The user should be able to set a mode that opens / closes the lights automatically if a user enters , leaves a certain room.
Preconditions	The user is logged in, has the required permissions and is on the dashboard.
Triggering event	The user selects the SHP tab.
Main flow	1. The user presses the Away Mode button “Activate”. 2. The command prompt signals success of the operation 3. The lights will open and close automatically
Extensions	The user does not have the permission to activate the away mode : 1. A prompt window appears indicating that the user does not have the permission to activate the away mode 2. The user closes the window by pressing Confirm
Postconditions	If a user enters or leaves a room that has lights, they will open/close accordingly. All automatic doors are locked, all lights and windows are closed.

3.8. Set an Away mode

Use case name	The user can set an Away mode (different from auto-mode)
Level	User-Level
Brief description	A registered user is able to set an Away mode
Preconditions	The user is on Away Mode
Triggering event	The User is in the SHP tab.
Main flow	1. The user Clicks on “set away mode” 2. The user sets a wanted timer
Extensions	The user does not possess the permission to set away mode 1. A new alert windows pops up informing the user that they do not possess the required permissions 2. The user closes the alert box 3. Return to the step 1 of the main flow.
Postconditions	The Away mode is set.

3.9. Motion Detectors notifications

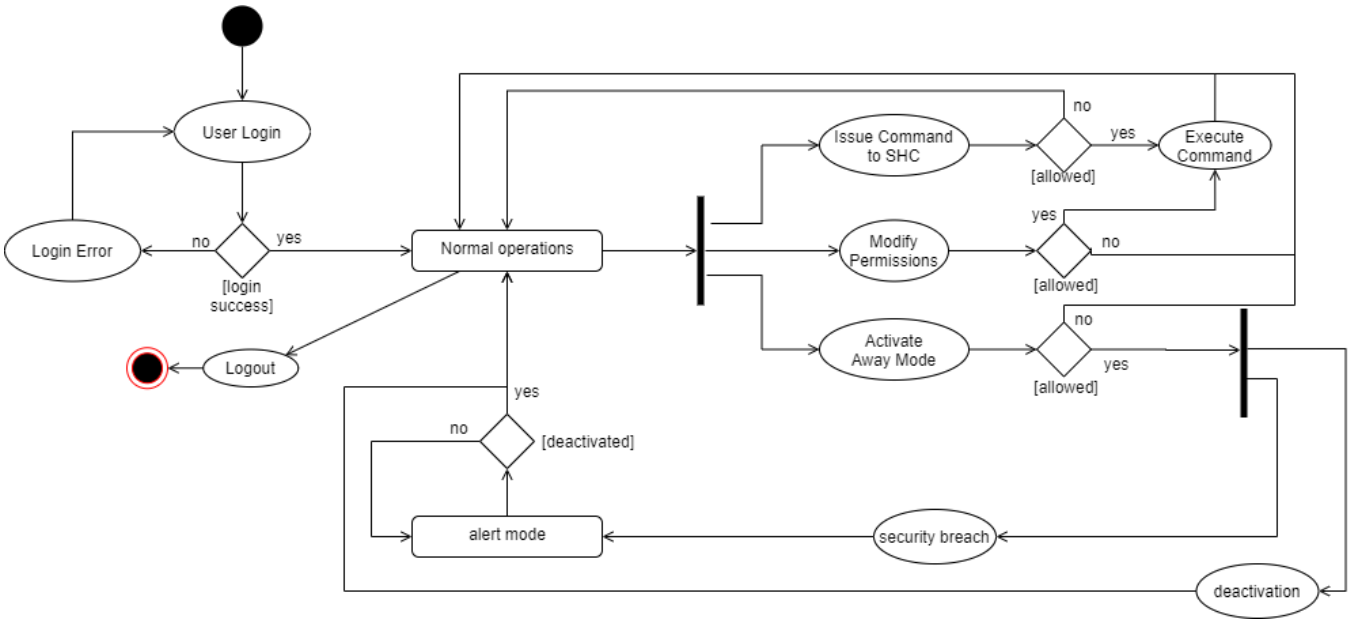
Use case name	Motion Detector Notifications
Level	User-Level
Brief description	A registered user receives notifications if motion detectors are triggered during Away mode.
Preconditions	The user is on Away Mode
Triggering event	A motion detector is triggered in the house by a non registered profile
Main flow	3. The user is in Away mode 4. The motion detector is activated 5. A non-registered profile steps in its path 6. The motion detector is triggered by the movement of the non-registered user 7. A notification is sent to the registered user
Extensions	The motion detector is not activated because the user is not in away mode The user starts from step 1 of the main flow
Postconditions	The User receives a notification on the dashboard.

3.10. Set Time before alerting authorities

Use case name	Set time before alerting authorities
Level	User-Level
Brief description	Upon triggering of the motion sensors, and after the users of the system receives a notifications, the system will alert authorities of an intrusion at the required time set by the user.
Preconditions	The user is logged in and has the permissions to change such data.
Triggering event	The user is in the SHP module tab.
Main flow	1. The user chooses the “Set Alert Timer” option 2. The user chooses the wanted timer until alerting the authorities 3. The user presses Confirm
Extensions	The user does not possess the permission to set a timer 4. A new alert windows pops up informing the user that they do not possess the required permissions 5. The user closes the alert box 6. Return to the step 1 of the main flow.
Postconditions	The wanted timer to alert authorities is set.

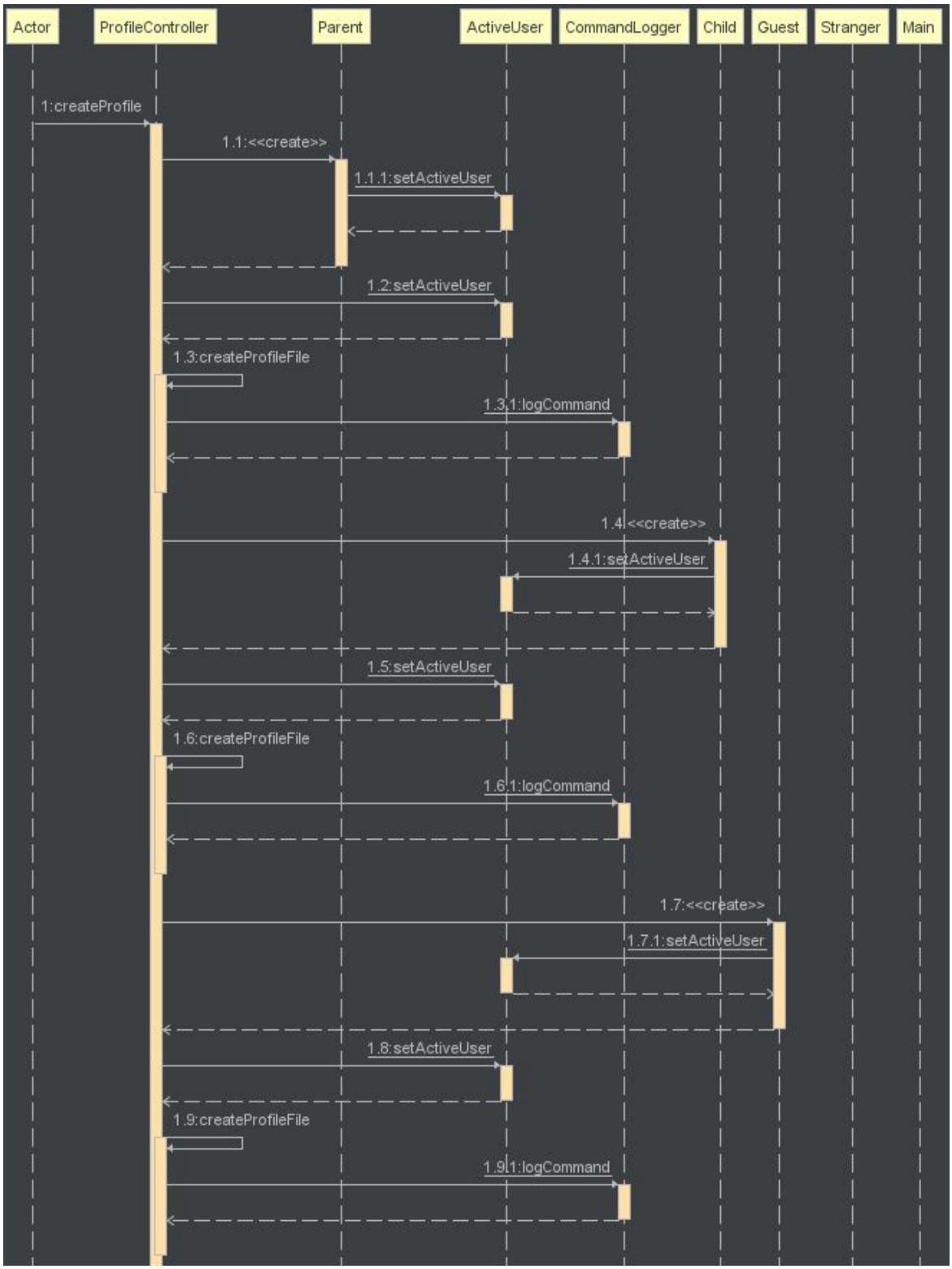
4.0. Diagrams

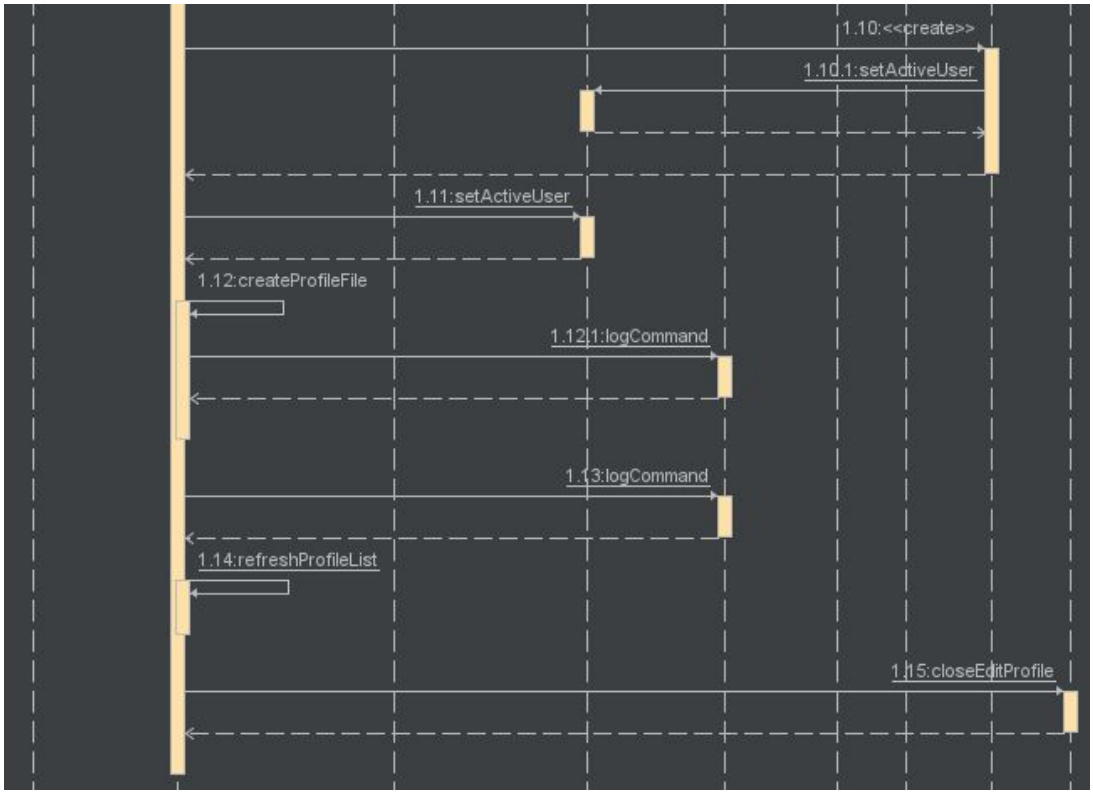
4.1. Activity Diagram



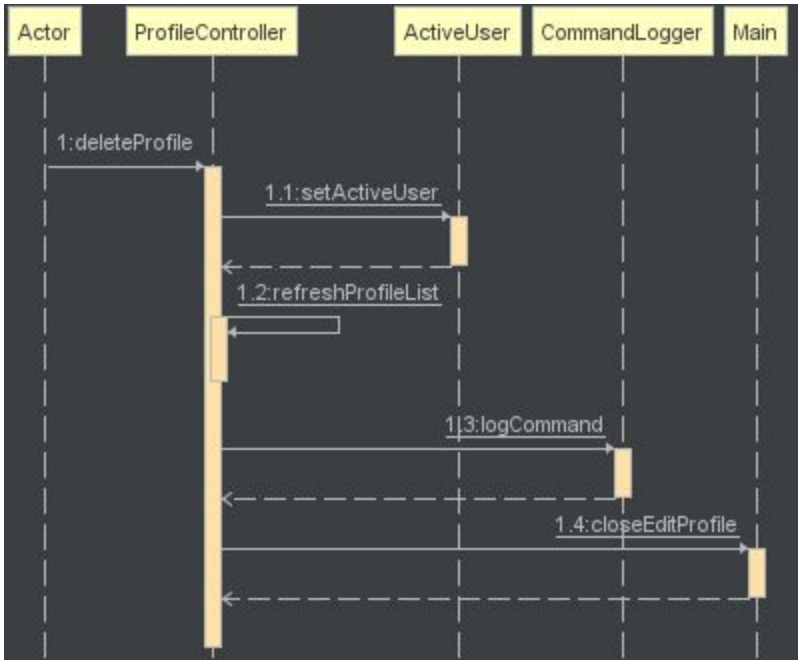
4.2. Sequence Diagrams

Save a new profile

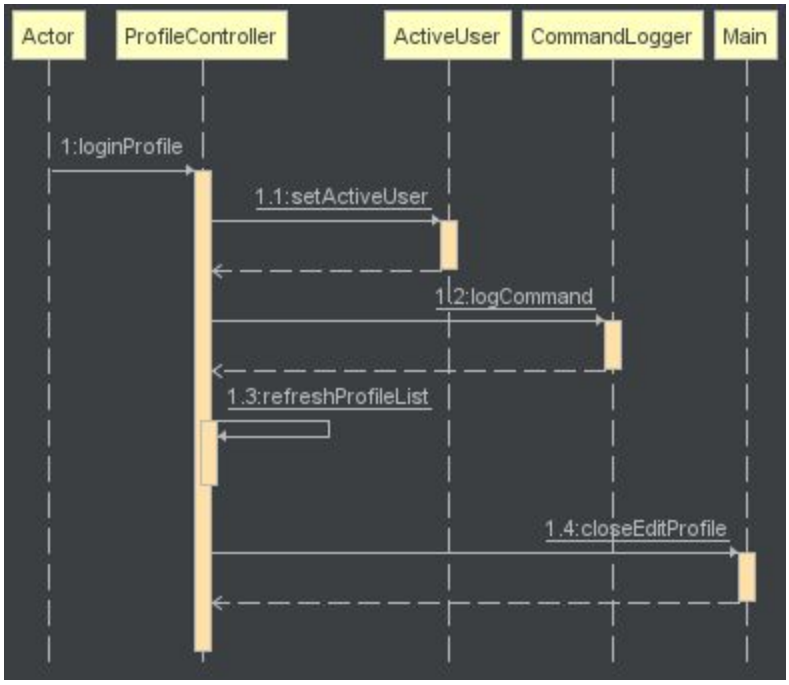




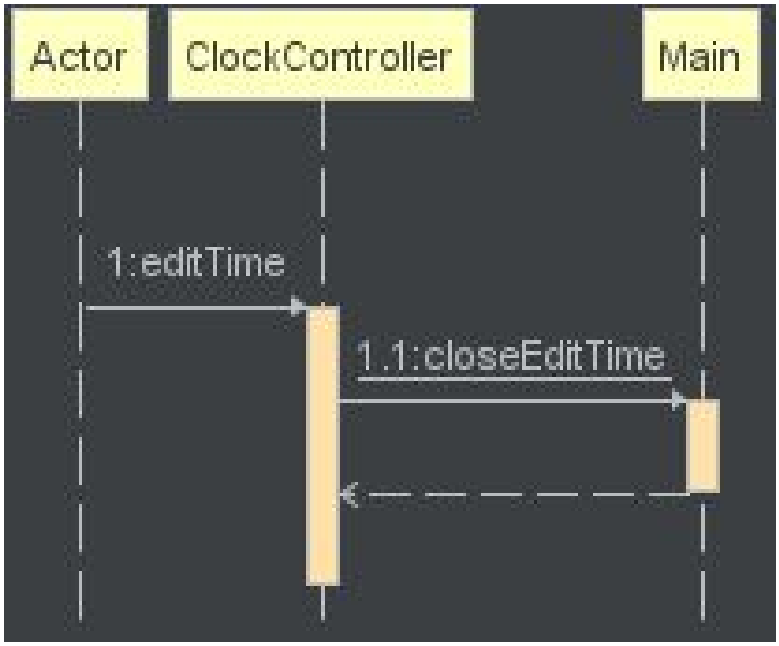
Delete an existing profile



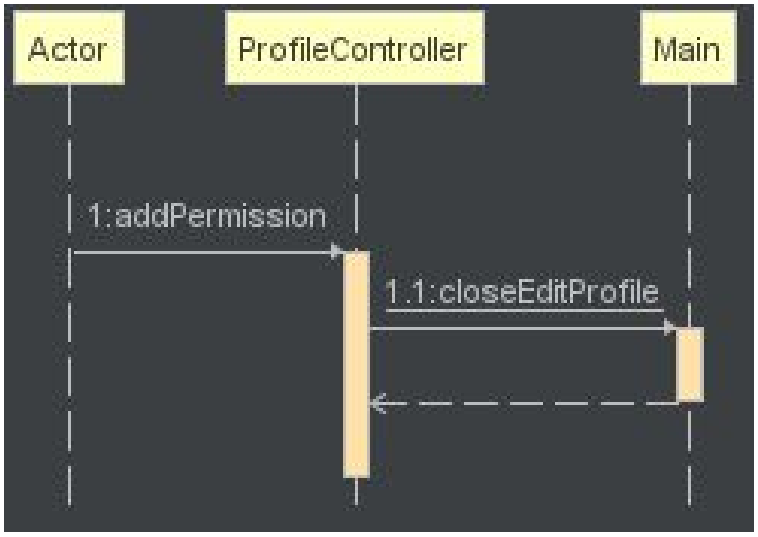
Login to an existing profile

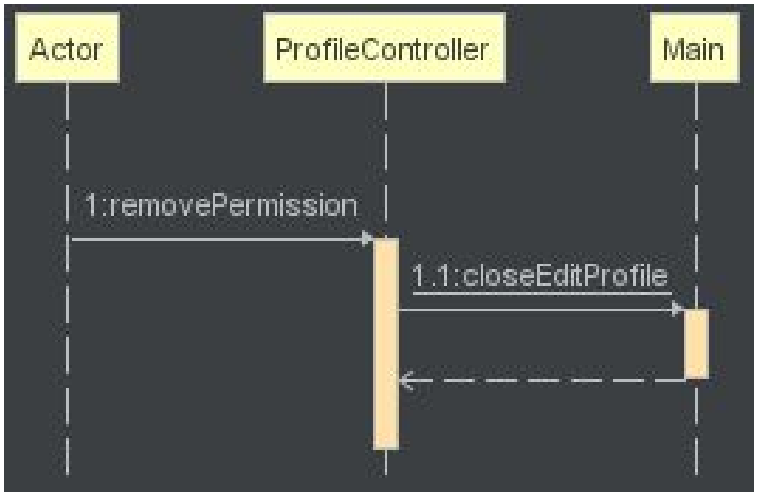


Change the Simulator's time speed

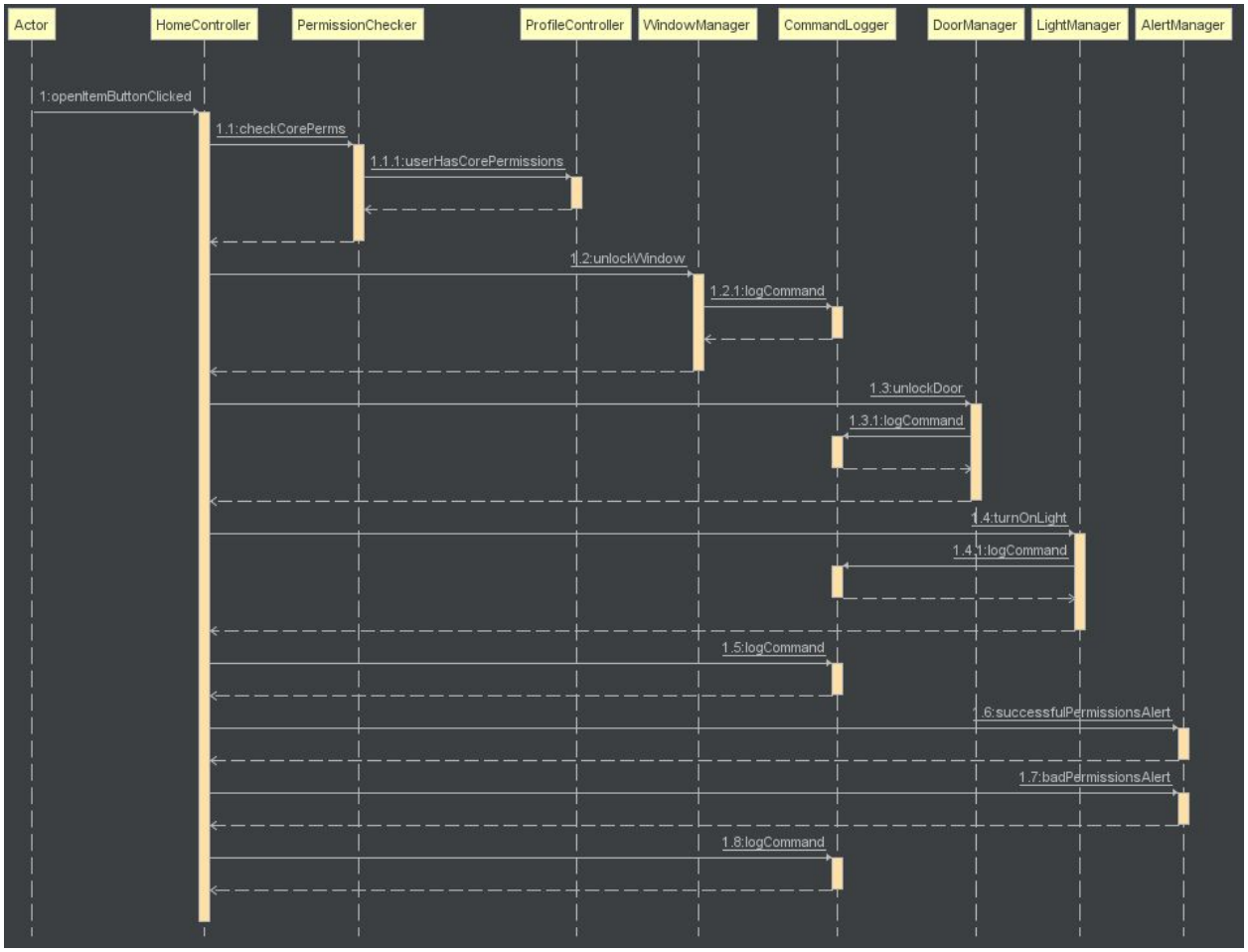


Edit permissions of an existing profile

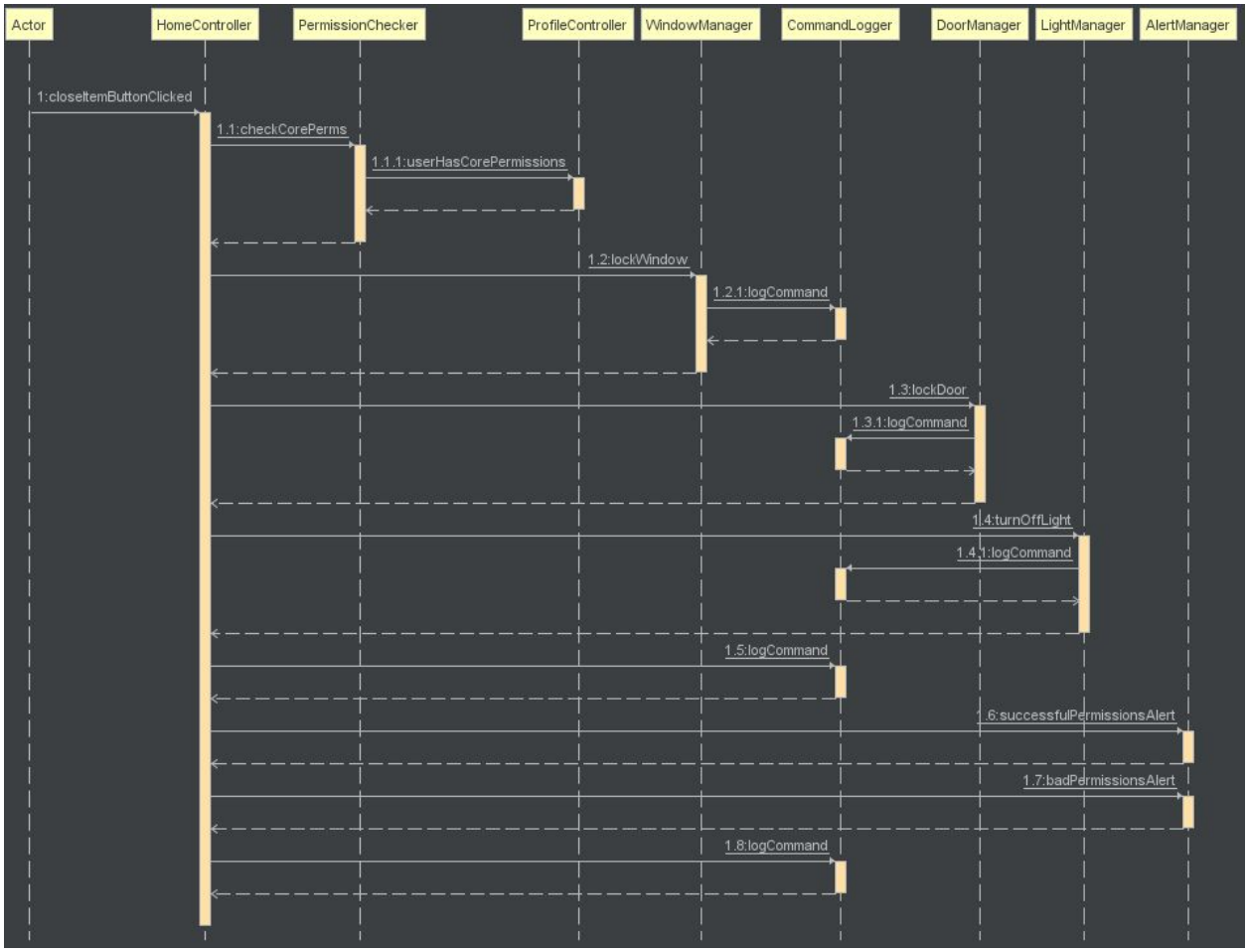




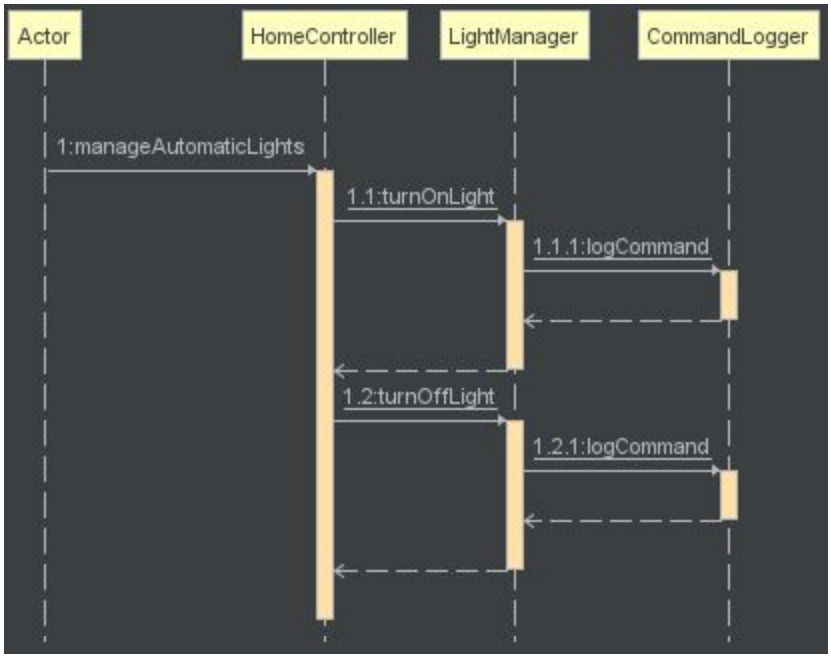
Open Windows/Doors/Lights and check for permissions



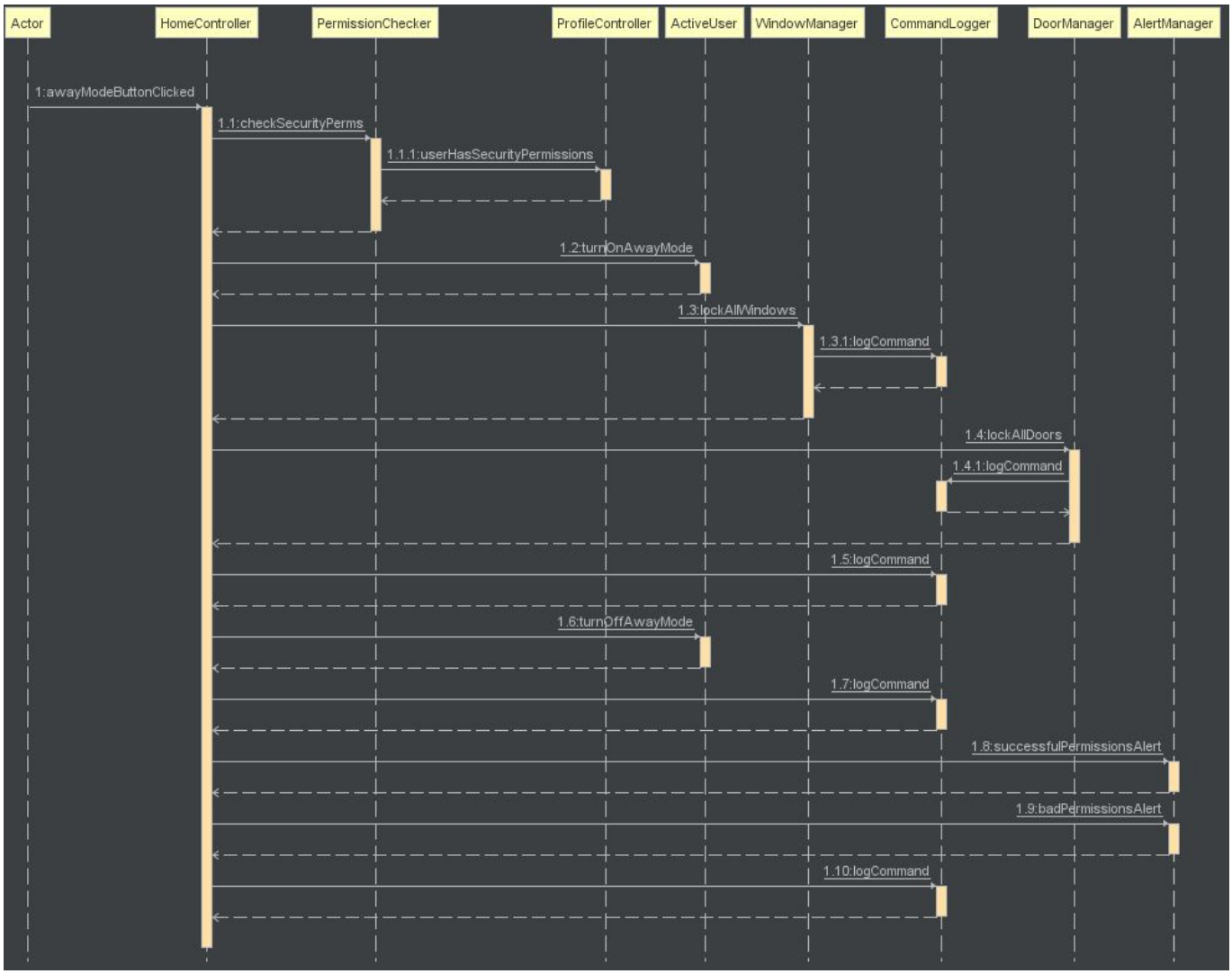
Close Windows/Doors/Lights and check for permissions



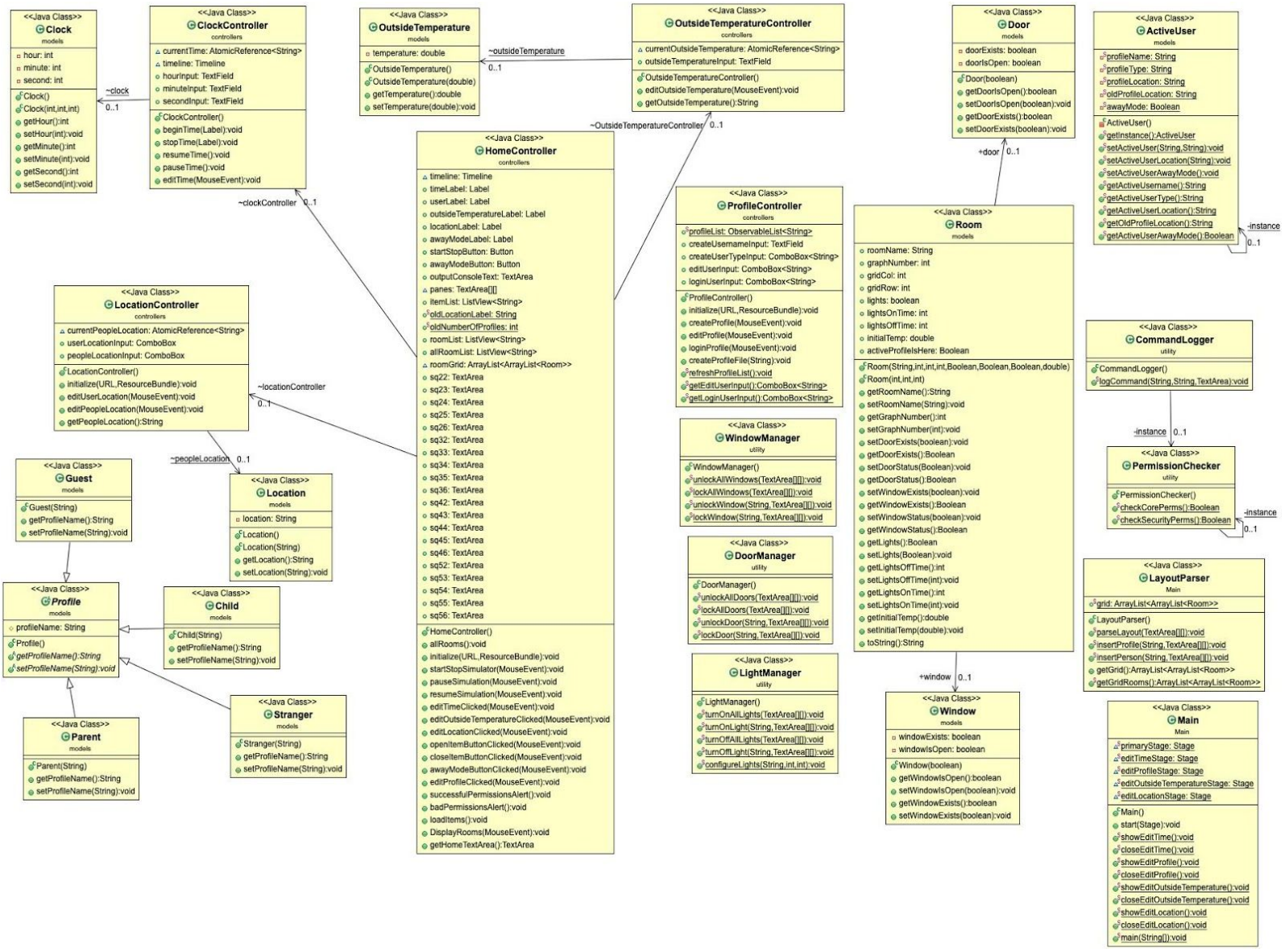
Set an “Auto” mode



Set an “Away” mode



4.3. Class Diagram



4.4. State Diagram