Internet of Things Automated Home

Assignment 3 Seng 330, Fall 2018

Kai Kovacik kkovacik@gmail.com
Seth Tolkamp stolkamp@uvic.ca

Progress

Of the 35 required Acceptance Tests, we have implemented 20:

A 4
B: 2, 3, 4, 6
C: 1 - 5
D: 1 - 5
E: 1, 2
F: 1-3

We have a UI that allows the addition of new devices and control of those devices.

A login page, removal of devices, persistent system information/logging, and managing of device notifications to users is still required.

How to Test

There are two ways for you to test the system.

1) Type ‘grade run’. This will run the UI Application Client.java and you can easily verify the functionality manually.
2) Type ‘gradle test’. This will run both the UI acceptance tests and the acceptance tests that only interact with the system’s controllers and models.

   The non-UI Acceptance Tests cover as much of the system’s specified functionality without user input.

   The UI Acceptance Tests add cameras and thermostats to the system and verifies their functionality in an organic use case manner. The Std Error that prints during testing of the Thermostat is not a test failure, it is the Thermostat notifying that the entered temperature was too large.

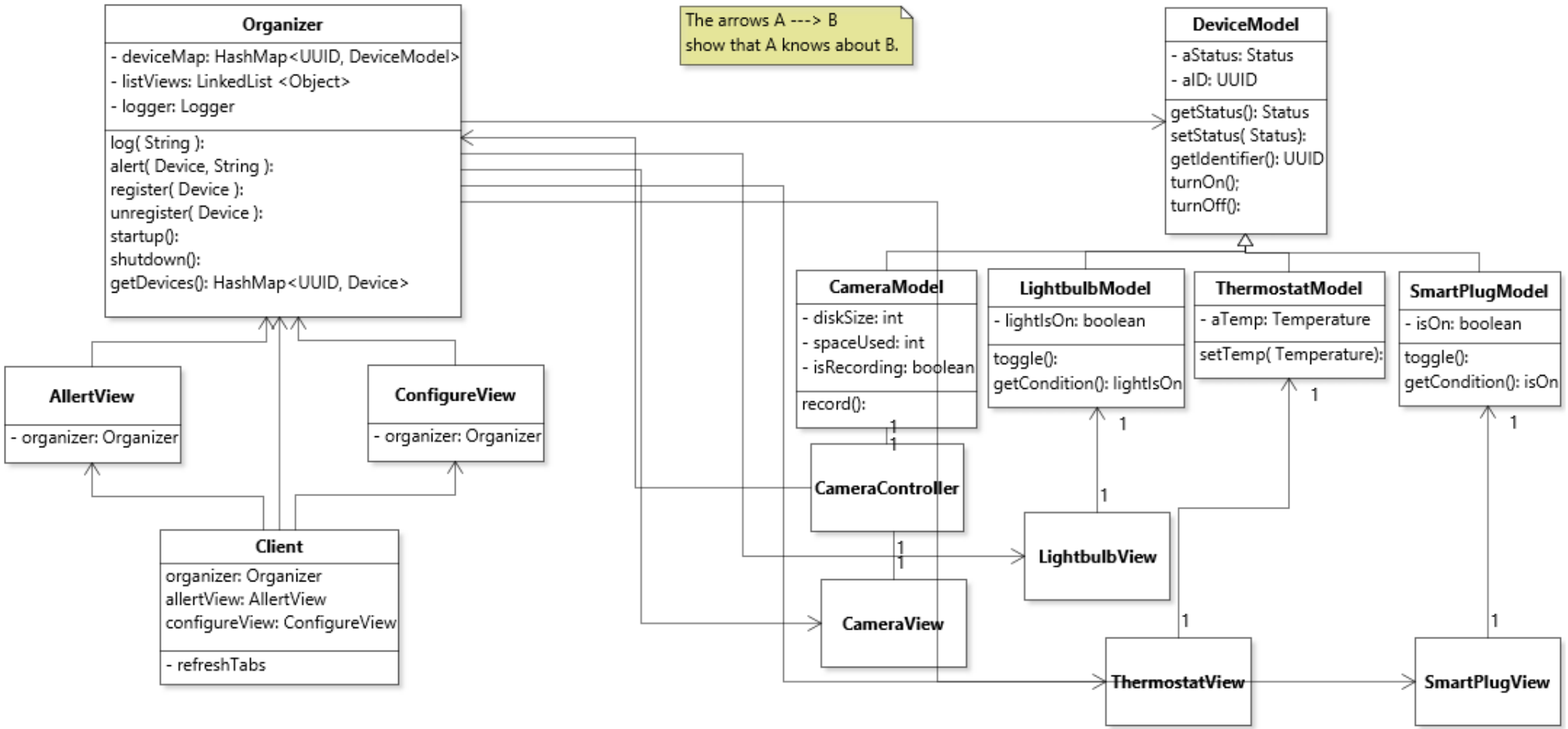
System Structure

This system was constructed with the ultimate goal of having a functioning system. Due to this, the system does not follow the best principles both regarding how OO systems should run and how Javafx applications should be built and implemented. It has been a huge learning experience and still needs to be refactored to improve the encapsulation and decoupling.

Most of the devices consist of a model and a view, with the Camera also having a controller. Further work will move the cameras control code into the model similar to the other devices. This choice was made because, as seen in the camera controller, the controller ends up duplicating code and implementing getters for getters in the camera model. This system does not require a controller as the control code is quite simple in the first place and separation of data storage and control just leads to more work and more code.

The organizer holds a collection of the views and the models of each device. The list of views was required to display views in the Client, this will be improved upon in the future, probably by having the organizer hold a list of views only, where each view is aware of its controller and model. This will greatly reduce coupling in the system.

The Client is a Javafx driven tabular application with a tab for configuring the system and for each device. The system alerts are displayed at the bottom of the Client always. The Client instantiates the organizer and populates the tabs with the correct views. Each device tab refreshes based on the list of views in the Organizer and objects are added to the Organizer in the Configuration tab. In the future, when a user logs in as a basic user the Configuration tab will be hidden.


Organizer
- deviceMap: HashMap<UUID, DeviceModel>
- listViews: LinkedList <Object>
- logger: Logger
log(String):
alert(Device, String):
register(Device):
unregister(Device):
startup():
shutdown():
getDevices(): HashMap<UUID, Device>
The arrows A ---> B
show that A knows about B.
DeviceModel
- aStatus: Status
- aID: UUID
getStatus(): Status
setStatus(Status):
getIdentifier(): UUID
turnOn();
turnOff():
CameraModel
- diskSize: int
- spaceUsed: int
- isRecording: boolean
record():
LightbulbModel
- lightIsOn: boolean
toggle():
getCondition(): lightIsOn
ThermostatModel
- aTemp: Temperature
setTemp(Temperature):
SmartPlugModel
- isOn: boolean
toggle():
getCondition(): isOn
AllertView
- organizer: Organizer
ConfigureView
- organizer: Organizer
Client
organizer: Organizer
allertView: AllertView
configureView: ConfigureView
- refreshTabs
CameraController
CameraView
LightbulbView
ThermostatView
SmartPlugView
1