Exercise 10: Smart Home

TECHNISCHE UNIVERSITÄT DARMSTADT

Software Engineering Design & Construction
WS 2016/17 - Dr. Michael Eichberg, M.Sc. Matthias Eichholz

This exercise will be graded. Please submit your solution until the 6th of February, 23:59 via email to `eichholz@st.informatik.tu-darmstadt.de`. Make sure you zip your **complete** sbt project and make sure that it works out of the box by running `sbt run` and `sbt test`. Your attachment for the submission must adhere to the following naming scheme: ex10_*LASTNAME_FIRSTNAME_MATRICULATIONNUMBER*.zip. Every student must submit an individual solution, group submissions won't be accepted.
Support can be found in the office hours of our tutors and in the forum at `https://www.fachschaft.informatik.tu-darmstadt.de/forum/viewforum.php?f=234`.

Introduction

The goal of this exercise is to implement a simple simulation of a smart home. In a smart home, multiple sensors and actuators are deployed in each room. Sensors are responsible for sensing the environment. They can provide information about the ambient temperature, air humidity, fraction of pollutants in the air, number of persons in a room and many more. Actuators translate the signals provided by sensors into different actions. For example, the detection of motion can trigger a light being turned on an increased air humidity can adapt the ventilation.
This exercise combines different topics covered in the lecture, like path-dependent types, stackable traits and reactive programming[1]. In order to keep the domain logic simple, especially for the sensors, we will simulate sensor inputs by using UI controls. We use ScalaFX[2] for the user interface, which is a Scala library sitting on top of JavaFX. In case you are working on Linux, it might be necessary to additionally install the respective JavaFX package for your distribution.
In the project template you find several base traits on which your implementation should build on. Do **not** change the signatures of traits we already provide, except when stated otherwise.

```
trait Building {

  trait TLocation

  type Location <: TLocation

  trait TRoom extends TLocation

  type Room <: TRoom with Location

  trait CompositeLocation[L <: Location] extends TLocation {
    def locations: Seq[L]
  }

  trait THouse extends CompositeLocation[Room]

  type House <: THouse with Location

  def buildHouse(clock: Clock): House
}
```

The `Building` trait serves as a foundation for implementing various kinds of buildings. It is a slightly adapted version of the code from the lecture slides, so you should already be familiar with the code. A house only consists of multiple rooms, but it does not contain multiple floors in our case.

[1] `http://www.rescala-lang.com`
[2] `http://www.scalafx.org`

```
trait Sensor[T] {
  val value: Signal[T]
}
```

In our implementation Sensors can provide measurements of different data types. For example, a temperature sensor could provide the measured temperature as a `Double`, while a motion detector uses `Boolean` to indicate whether there is any motion sensed or not. We use a signal to model sensor readings as a continuous value.

```
trait Actuator {}
```

The `Actuator` trait should be used as basis for all of your actuator implementations.

Task 1 Clock — 1P

Actions carried out by actuators might be time-dependent. For example, on weekdays we might want to turn the heating down during the day and automatically turn it up again before people return from work. Our smart home therefore needs a central clock. Complete the class `Clock` and model seconds, minutes, hours, days and the current time of the day in seconds since midnight using signals.
Hint: You can use the method `Signal.changedTo(value)` which gives you an event that fires as soon as the signal value changes to the value provided as parameter and the method `Event.count` to count the occurrences of an event.

Task 2 Buildings — 4.5P

The goal of this task is to create a reusable class hierarchy to implement different kinds of Smart Homes and to create a very simple graphical user interface for our Smart Home. Since we want to adhere to the Single Responsibility Principle, we need to separate these two aspects in our implementation.

Task 2.1 Smart Home — 1.5P

As the first step, create a new trait `SmartHome` that extends the `Building` trait and allows us to add a sequence of sensors and actuators to each room and we also want to give names to rooms, like “Living Room” or “Bed Room”. It should also be possible to get a sequence of all sensors and actuators respectively from all rooms together.

Task 2.2 Rendering the GUI — 3P

Next, create a trait `DrawableSmartHome` that adds a `render` method to each location. Since we are using ScalaFX, the render method should return a `scalafx.scene.Node`, which allows us to process the rendering result further.

```
def render(): scalafx.scene.Node
```

Rendering rooms

A renderable room must provide at least the following properties: a width, a height and its position. For each room we want to render a rectangle with the provided width and height. Inside of this rectangle we want to put a label displaying the name of the room and at a later point of time we want to render all sensors and actuators installed in the room.
Hint: You can use a `scalafx.scene.layout.StackPane`[3] to display UI controls on top of each other. A `scalafx.scene.layout.VBox`[4] can be used to lay out controls in a single vertical column.

Rendering the house

Figure 1 shows an example of how the UI for a house might look like. At the top, we want to display the current time in the format *“Day Hours:Minutes:Seconds”*, e.g. Tuesday, 09:50:00. Create a new signal (`Signal[String]`) that returns the formatted time and use a label to display the current time in the UI. We then want to display the rooms below the label containing the current time.
Hint: You can use a `scalafx.scene.layout.AnchorPane`[5] to draw the rooms at their respective positions. To do so, have also a look at the methods `AnchorPane.setTopAnchor` and `AnchorPane.setLeftAnchor`.

[3] https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/StackPane.html
[4] https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/VBox.html
[5] https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/AnchorPane.html

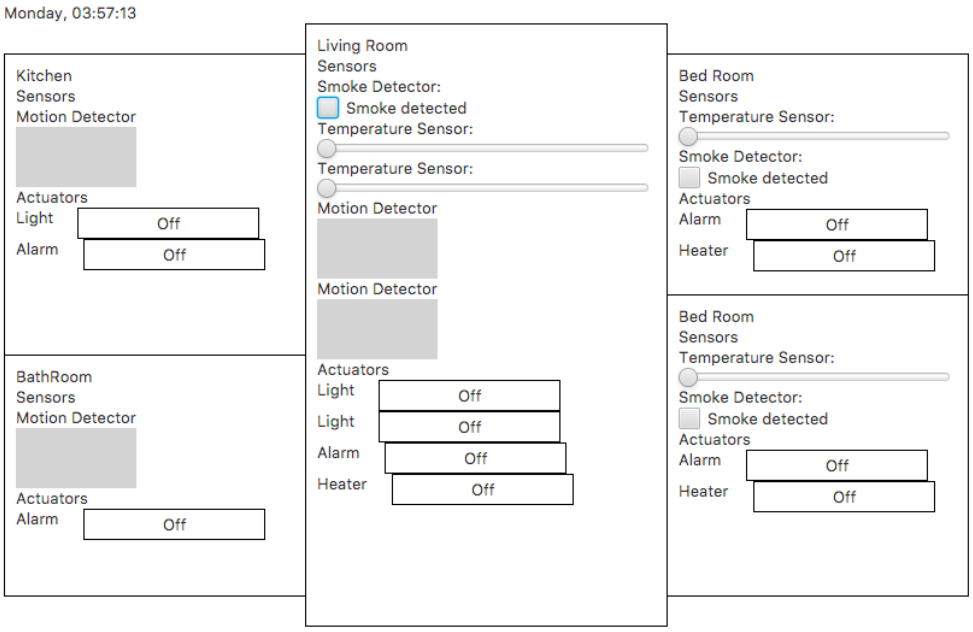

Figure 1: Example of how the final UI might look like

Task 3 Sensors — 3P

In this task we want to implement a set of different sensors for our smart home. Since we also want to display the sensors in the UI, start by creating a new trait `DrawableSensor` that extends the provided trait `Sensor` and adds a render method **`def`** `render(): scalafx.scene.Node`. For the following concrete sensor implementations, you can directly inherit from `DrawableSensor`. Make sure that only subtypes `DrawableSensor` can be added to the list of sensors of a `DrawableSmartHome`. *Hint:* This can be achieved by using abstract types.

Task 3.1 Smoke Detector — 1P

Implement a simple smoke detector that just tells us whether smoke is detected or not. Render a CheckBox[6] in the UI. When the CheckBox is selected, the sensor value should be true, otherwise false.
Hint: JavaFX/ScalaFX UI controls use so-called properties to represent their internal state. For example, if you want to check, whether a CheckBox is checked, you can use `CheckBox.selected` which returns a `BooleanProperty`. In the template you can find an implicit conversion from a ScalaFX property to a REScala signal, which can be used as follows:

```
import JFXAdapter._
val checkbox = new CheckBox()
val value: Signal[Boolean] = checkbox.selected.toSignal
```

Alternatively, you can register an `onChange` listener on a property to get notified when the value of a property changes. More information about properties in ScalaFX can be found at `http://www.scalafx.org/docs/properties`.

Task 3.2 Temperature Sensor — 1P

Next, implement a temperature sensor that returns the current temperature as a double value. Render a Slider[7], ranging from 0.0 to 30.0 in the UI and connect the slider value to the signal representing the sensor value.

Task 3.3 Motion Detector — 1P

Finally, implement a motion detector, which indicates the position as a tuple of doubles `(Double, Double)` at which it last detected motion. We want to simulate motion by moving the mouse over an area. Add a rectangle to the UI and use the `onMouseMoved` event handler to get the position where the mouse moved last.

[6] `https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/CheckBox.html`
[7] `https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Slider.html`

Task 4 Actuators — 1.5P

Similar to the sensors implementation, start by creating a `DrawableActuator` that provides a render method. Also, make sure that a `DrawableSmartHome` only accepts subtypes of `DrawableActuator` to be added to the list of actuators. Implement three different actuators, `Light`, `Siren` and `Heater`. For each actuator, by default a white rectangle with a label containing the text "Off" should be rendered in the UI. When the actuators are turned on, the label should contain the text "On" and the fill color of the rectangle should turn yellow, red, and blue for the three actuators respectively.

Task 5 Building the House — 2P

Create a class `SimulatedSmartHome` that extends your `DrawableSmartHome` trait and that can be used to build a house consisting of at least a kitchen, a bath room, a living room and two bed rooms. The rendered version of this house should look similar to Figure 1. The following sensors and actuators should be installed in the individual rooms:

Room	Sensors	Actuators
Kitchen	Motion Detector	Light, Siren
Bath Room	Motion Detector	Siren
Living Room	Smoke Detector, 2x Temperature Sensor, 2x Motion Detector	2x Light, Siren, Heater
Bed Room	Smoke Detector, Temperature Sensor	Siren, Heater

Task 6 Connecting Sensors and Actuators — 6P

So far, we are able to install sensors and actuators in our SmartHome, but no connection exists between these two. We therefore want to implement various policies that determine how actuators behave in response to sensor data. We want to be able to reuse these policies for different smart home implementations by using mixin-composition. The following code snippet shows how this composition might look like.

```
class MySmartHome extends SimulatedSmartHome with FireAlarm with Lights
```

Task 6.1 Fire alarm — 1P

Create a trait `FireAlarm` that connects smoke detectors and sirens. Whenever any smoke detector in the house detects smoke, all available sirens in the house should be activated.

Task 6.2 Motion-controlled Lights — 3P

Create a trait `Lights` that connects the motion detectors to the lights. Whenever any motion detector in a room detects motion, we want to turn on all available lights in the same room. Since it is usually not necessary to turn the lights on during the day, the automatic control of lights should only be enabled in the evening between 6pm and 11pm.
Hint: Our motion detectors only report the position where they have sensed any motion last. However, they do not tell us when motion has been detected. We can solve this issue by using timeouts. You can use the method `Signal.changed` to create an event that is fired when the motion detector reports a new position and indicates that motion has been detected. When the event is fired the first time, create a timer with a delay of 100ms that indicates that there is no motion anymore. Reset the timer whenever motion is detected before the timer action has been executed.

Task 6.3 Heating — 2P

Create a trait `Heating` for controlling the temperature in the house. In general a room's heater should turn on, if the average temperature per room drops below 20 degree Celsius. However, you don't want to heat the house when nobody is at home. On weekdays you want to heat the rooms in the time from 5am to 8am and from 4pm to 10:30pm. On the weekend you want to keep the temperature at 20 degree Celsius from 7am to 11pm. In the remaining time of the day you just want to ensure that the room temperature never drops below 15 degree Celsius.

Task 7 SOLID — 2P

Discuss shortly, the connection between the Single Responsibility Principle (SRP) and Signals/Events. Discuss it in relation to the smart home example.