Assignment 2 - Description

A2 – Smart Home Controller

In this assignment, you will continue working on the Smart Home Controller challenge playing the role of "Smart Builders", i.e., a bunch of developers who develop custom smart homes by putting together various components. You will now face three challenges:

* **Adapting an Old Sensor.**You are given an temperature sensor driver that does not satisfy the Simple Smart Device Collaboration Standard (SSDCS). You will need to use the **adapter pattern** so that you bring it up to SSDCS standards.
* **Adapting a Sensor with inverted control.**You are also given a temperature sensor that not only does not support SSDCS but it generates messages in response to temperature events, following the **observer pattern**. You will need to combine observer and adapter pattern to use this new sensor in your project.
* **A Customizable Furnace.**You are building your own family of SSDCS-compliant furnaces with various extra's and components. You will need to apply the  **decorator pattern** in order to makes sure that the furnace always initializes the appropriate components when it is turned on.

**Recall: Sensors**, **Controllers** and **Actuators** of various vendors are able to work together through the use of Interfaces. Specifically a consortium of smart home vendors, called "*ITEC3030 Smart Home Consortium”* worked together to produce the **Simple Smart Device Collaboration Standard (SSDCS)** in form of a set of **Java Interfaces**. The Interfaces can be freely downloaded from [here (jar)](https://eclass.yorku.ca/eclass/pluginfile.php/467959/mod_page/content/14/Standards.jar) and the JavaDoc-generated Documentation can be found [here (zip)](https://eclass.yorku.ca/eclass/pluginfile.php/467959/mod_page/content/14/Standards-javadoc.zip).

Meanwhile the files below represent your work on the controller:

* [**Controller.java**](https://eclass.yorku.ca/eclass/pluginfile.php/467959/mod_page/content/14/Controller-A2.java)**[you will need to rename the file or class]**, as in the exercise the controller device collects data from sensors and sends commands to actuators. Here it is simplified to exclude Rooms, Buildings etc.
* [**Main.java**](https://eclass.yorku.ca/eclass/pluginfile.php/467959/mod_page/content/14/Main-A2.java) **[you will need to rename the file or class]**, a test bed where various sensors can be created and controlled/tested.

For this assignment you will need to complete the following exercises:

**Exercise 1 (30%)**

OldTemp Inc. is a company that was developing sensors before SSDCS existed. You need to use one of their sensors OldyTempSensorOld3 which obviously does not comply with SSDCS. The company offers you[a JAR of the sensor driver](https://eclass.yorku.ca/eclass/pluginfile.php/467959/mod_page/content/14/OldyTempSensorOld3.jar) and [a JavaDoc for the Driver](https://eclass.yorku.ca/eclass/pluginfile.php/467959/mod_page/content/14/OldyTempSensorOld3-JavaDoc.zip). Use the **adapter pattern**to use OldyTempSensorOld3 in your integration project so that it complies with SSDCS. Name your Adapter **OldyTempSensorAdapter**;

**Submit:**

1. A run of scenario 1 (see Main.java).
2. A **2-3 minute uninterrupted video** presenting your solutions and demonstrating the code. The video should be staged to show: (i) how you set up your Eclipse project from scratch and import the Java files, (ii) how you import the driver in the project (including showing that it does not work without the import), and (iii) how you run it.
3. The JAVA files relating to your solution.
4. A link to a JavaDoc describing **your** classes.
5. A **class diagram** demonstrating your solution.

**Exercise 2 (40%)**

NewTemp Inc. is a company that is developing sensors just ignoring SSDCS. Instead they implement part of an observer pattern in which the sensor is a Subject generating notifications. The company offers you [a JAR of the sensor driver](https://eclass.yorku.ca/eclass/pluginfile.php/467959/mod_page/content/14/NewlyTempSensorNew4.jar) and [a JavaDoc](https://eclass.yorku.ca/eclass/pluginfile.php/467959/mod_page/content/14/NewlyTempSensorNew4-JavaDoc.zip) where the observer implementation is described. Use the **adapter pattern**to comply with the observer pattern that NewTemp imposes. . Name your Adapter **NewlyTempSensorAdapter**. [\*NEW\* print an arbitrary message in response to NewTemp' sensor *callback*. Initiate this callback yourselves through triggering the sensor's notifications when a new temperature is registered.]

**Submit:**

1. A run of scenario 2 (see Main.java)
2. **A 2-3 minute uninterrupted video** presenting your solutions and demonstrating the code. The video should be staged to should show: (i) how you set up your Eclipse project from scratch and import the Java files, (ii) how you import the driver in the project (including showing that it does not work without the import), and (iii) how you run it.
3. The JAVA files relating to your solution.
4. A link to a JavaDoc describing **your** classes.
5. A **sequence diagram**  showing what happens when newTemperature(0) is called from Main.

**Exercise 3 (30%)**

Smart Builders also decided to start building furnaces. They call their family of furnaces**OurFurnace**and they are SSDCS compliant. The basic furnace (which includes the basic components of a furnace) can be augmented with a Humidifier and a WiFi (for smartphone control). Depending on whether the furnace has these components, they must turn on when the furnace turns on. As it stands they only have one model **PlainGasF1**, and when it turns on it simply prints *"PlainGasF1: Up and Running".*The **WiFi** component prints *"Wifi: Initialized"* and the **Humidifier** prints *"Humidifier: On".*Depending on what, if any, add-ons are considered, the start sequence will print different messages. For example, PlainGasF1 with Wifi will print:

*"Wifi: Initialized".*  
*"PlainGasF1: Up and Running".*

... when turned on (i.e. method turnOn()).

Implement the product family using the **decorator pattern, and show that turnOn()**behaves accordingly.

**Submit:**

1. A**2-3 minute uninterrupted video** presenting your solutions and demonstrating the code. The video should be staged to should show: (i) how you set up your Eclipse project from scratch and import the Java files, (ii) how you import the driver in the project (including showing that it does not work without the import), and (iii) how you run it.
2. The JAVA files relating to your solution.
3. A link to a JavaDoc describing **your** classes.