Name: *CSC-490, Winter 2020*

Observer Pattern

In order to complete this worksheet you will need the files located in the folder `Observer` in CoCalc.

1. (a) (2pts) Copy down the intent of the Observer pattern according to the GoF:

   (b) (8pts) Draw the class diagram for the Observer pattern given by the GoF (including pseudocode) using our conventions for class diagrams.

   (c) (10pts) In Cocalc you'll find a partial implementation of the class diagram above in the folder `Observer/Subject`. Your job is to complete the implementation by completing all of the commented TODOs. When you're done, run the program and see if the watcher's state changes when you change the tower's state.

last updated: January 18, 2020

2. (15pts) In this exercise we will use the Observer pattern to design a program simulating a group chat. Here's the setup:

   - There is a class `GroupChat` that has a data member `conversation` that holds the the conversation of a group chat as a string. This class also has methods called `getConversation()` and `addToConversation(message)`.
   - There is another class called `GroupChatMember` that has its own `conversation` and it also has a `GroupChat`. The class `GroupChatMember` has methods `postMessage()`, `viewGroupConversation()`, `leaveGroupChat()`, and maybe others.

   Draw a class diagram for this program using the Observer pattern. Include pseudocode for all of the methods in the concrete subject and concrete observer classes. If you are unclear as to what any of the methods are supposed to do, ask!

3. Note: you need to have part 1 completed before you complete this part.

   In the CoCalc folder `Observer/H2O` you'll find a program that partially implements the State pattern with the `H2O` class that we discussed in the previous worksheet. There are a couple new classes called `Temperature` and `WeatherStation`.

   (a) (5pts) Draw the class diagram for the current state of the program, but leave room since we will be adding more to it.

   (b) (5pts) Modify your diagram using the Observer pattern so that the `H2O` class is a concrete observer of the concrete subject `WeatherStation`.

   (c) (10pts) Implement the `handleTemperatureChange()` methods in the Liquid and Gas states. The implementation in the Solid state is already done. You can modify the `Main.cpp` file to check that your implementation is correct.

   (d) (10pts) Implement the Observer pattern described above so that the `H2O` class updates its temperature (by calling `setTemperature()`) whenever its weather station sends a notification. You can reuse the implementations of the classes `Subject` and `Observer` from part 1 here.

   (e) (10pts) Rewrite `Main.cpp` so that the user can repeatedly update the temperature of a weather station, and after doing so the weather station notifies an `H2O` object. If everything is hooked up correctly, after each update you should see the water appropriately freezing, melting, evaporating, etc. according to the user's inputs.