Lab 5: Behavioural Patterns

Due: October 25, 8 p.m.

1 Introduction

This week kicks off our introduction to design patterns. The Model-View-Contoller pattern plays a role in your second assignment, and the Observer pattern in covered in a course video. In this lab we'll be diving a little more deeply into the Observer pattern while we revisit our work with sewers.

To begin, copy the files from the course Quercus (in the 'lab05.zip' archive) into your local copy of your GitLab repository. After you have done this, you should have one folder named 'lab05' in your repository. You should find a folder called week5 in the lab05 folder and five java files therein: *Main.java, Maintainer.java, MississaugaSewers.java, Sewer.java* and *ObserverTest.java.* There is also a data file and a package called 'observer'. In the 'observer' package are the following files: *ObservableSewer.java* and *SewerObserver.java.*

2 Programming Task

In the past, we visualized the city's sewers; this week we'll consider maintaining those sewers. Imagine that there are several sewer maintainers around the city, yet each has a specific specialty and each has a location. Some are specialized to work on catchbasins, while others can work on manholes. Maintainers like to work close to home, and will therefore prioritize maintenance of sewer infrastructure in their region.

Your task is to use the Observer pattern to allow sewer maintenance staff to **register** for status updates that are related to Mississauga's sewers. Registered maintainers will receive updates if and when the maintenance status of sewers happen to change. Registered maintainers, when they receive updates, will add sewer infrastructure that is in need of maintenance to their to-do lists if, and only if, they are both specialized to work on the infrastructure and the infrastructure is located within their operating radius. Registered maintainers can also un-register if and when they would like to stop receiving such maintenance updates.

Your starter code contains classes that will help you get started. You have code for an **SewerObserver** interface and an **ObservableSewer** base class. You also have code for a **Maintainter** class; this will "observe" instances of the **MississaugaSewers** class and receive "updates" when the status of any sewer infrastructure changes. You will be altering the file called *MississaugaSewers.java* to implement:

- The method *register*. This is inherited from the ObservableSewer class, and should be used to register a new observer to the list of sewer observers.
- The method *unregister*. This is also inherited from the ObservableSewer class, and should be used to unregister a new observer from the list of sewer observers.
- The method *notifyObservers*. This should broadcast a message to all observers so that they can"update" their to do lists to include sewers needing maintenance and/or remove sewers that do not.

You will also implement the following in *ObeservableSewer.java*:

- The method *setObservableState*. This accepts a sewer and should add or update the sewer the HashMap that is called 'sewers'. This HashMap tracks the current maintenance status of sewers in the area. Use the sewer's assetID as the key in the HashMap and make the sewer itself be the value. Once you have updated the HashMap, notify all the sewer observers that there has been a status change.

And finally, you will implement the following in *Maintainer.java*:

- The method *update*. This accepts a sewer and should add sewer to the to-do list of a Maintainer if, and only if, the sewer need maintenance, the Maintainer is specialized to work on the infrastructure and the infrastructure is located within the operating radius of the Maintainer. If the sewer does not need maintenance, it should be removed from the to do list of the Maintainer, if it happens to be there.

3 Testing your Classes

A small number of test cases have been provided to help you test your implementations. Make sure you pass these tests! And, as ever, make sure you write your own tests, too. **As before, we ask that you add one additional test to *ObserverTest.java* for each method you implement**.

4 What to Submit

1. ObserverTest.java
2. ObeservableSewer.java
3. Maintainer.java
4. MississaugaSewers.java

HAVE FUN AND GOOD LUCK!