Lab 09: Singleton and Observer Patterns

Tripleton (15)

We call the design pattern “single”-ton because there can only be a single instance of the object. This can be extrapolated to two, three, four, etc., instances as well. In this case, we’ll build a tripleton (maximum of three instances).

Building a tripleton is exactly the same as a singleton. You need to make the constructor private, and add a public static method called **`getInstance`** to return an instance. In this case, the method will create new instances while it can, and once it has created all three of them, it will return the 3rd instance. This **Tripleton** class should also have a method called **`getInstanceN`** to retrieve the Nth instance.

Put this Tripleton in a package called **tripleton** and create some tests in a main method there. Run a test to show that when you create a 4th instance, it is equal to the 3rd instance.

University (15)

In this part of the lab, we’ll implement the observer pattern. In this case, we’ll be using a couple deprecated packages in Java (do not use these for real applications, use java.beans) as they make it simpler to understand the pattern.

Create a Course class that has two instance variables: a course name, and a Prof object. Add getters for those variables.

Next, create the Prof class which extends **java.util.Observable** and is initialized with a name. It should have a name, room number, a date for the midterm, and a collecton of Students.

The Student should implement the Observer interface and is initialized with a name. It should have a name, room number, and a course variable.

Add a Secretary class that also implements the Observer interface. It should only have a date as an instance variable that will get set in it’s update method later.

In the Prof class, create a setter for the midterm date that also sets it’s status to changed and notifies its observers with the Date like so:

```
public void setMidterm(Date date) {
   midterm = date;
   // see why it is useful to have getters and setters!
   // we can now notify observers of the change
   setChanged();
   notifyObservers( date );
}
```

Build a similar method for the room number.

Still in the Prof class, create a public **takingMidterm** method which accepts and Observer object and adds it to your collection. Also, create a method that prints all students that are taking the midterm.

Now, in your Secretary, add a method called **registerToProf** that adds itself to the objects that the Prof will notify:

```
public void registerToProf(Prof p){
   p.addObserver(this);
}
```

Add a similar method to the Student class but instead it should register to a Course instead of a Prof (you can get the Prof from the Course object). Call it **registerToCourse**.

Finally, will need to implement the update methods for the Student, and Secretary classes! These will handle a notification from an Observable object that it is registered to. Here is the update method for the Student:

```
/**
 * This method is called whenever the observed object is changed. An
 * application calls an <tt>Observable</tt> object's
 * <code>notifyObservers</code> method to have all the object's
 * observers notified of the change.
 *
 * @param   o     the observable object.
 * @param   arg   an argument passed to the <code>notifyObservers</code>
 *                method.
 */
@Override
public void update(java.util.Observable o, Object arg) {
   if(arg instanceof Integer){
      roomNumber = (Integer)arg;
      System.out.println("Student " + name + " has a midterm in room number: "
+ roomNumber);
      return;
   }
   System.out.println("Student " + name + " says  ..Doh got it dude!");
   System.out.println( ((Prof)o).getName() + " says " + arg);
   ((Prof)o).takingTheMidterm(this);
}
```

Implement a similar method in the Secretary, except it should return when the **arg** is an instance of Integer. Otherwise, assume the argument is the Date, set it to the instance variable for the Secretary, get a random integer between 1 and 10, and set that as the room number with the Prof's **setRoomNumber** method.

Let's test this now! Create a class called University, that will contain the main method. In there, create a Prof object, a Course object, and 4 Student objects (call them Kramer, Elaine, Jerry, and George). Next, create a Secretary, and register it to the Prof. Register the Students to the course. Now, have the Prof set the midterm date, and then print all the students that are taking the midterm.

Grading Criteria:

Style/submission guidelines: https://gmierzwinski.github.io/bishops/cs321/style_guidelines.html

Comments, Formatting, & Readability	**5 Marks**
Submission Guidelines	**5 Marks**
Program	**30 Marks**
Total	**40 Marks**