Comp 401 - Assignment 8: Observer Pattern

Date Assigned: Thu Oct 08, 2015

Completion Date: Fri Oct 23, 2015

Early Submission Date: Wed Oct 21, 2015

In this assignment you will learn how to write observable Bean objects that are observed by both ObjectEditor and an observer object you will write. As ObjectEditor now observes them, there will be no need for you to explicitly call the ObjectEditor refresh() method. In fact, you are no longer allowed to call this method, and should *no longer get a warning of the form*:

W***Refreshing complete object: ….

As mentioned in class, you should aim for efficiency by not sending coarser-grained notifications than necessary and not creating new shapes in getters. You are free to use any object to store the list of observers including the ones presented in class lectures.

The following new material is relevant to this assignment. *The assignment should be trivial if you read and understand this material and may be impossible if you do not.*

MVC	PowerPoint PDF YouTube Mix	Docx PDF Drive		- MVC	lectures.mvc.monolithic Package lectures.mvc.interactor Package lectures.mvc Package	MVC
Component Notifications	PowerPoint PDF YouTube Mix	Docx PDF Drive		-	lectures.mvc.properties Package lectures.mvc.collections Package	Component Notifications

Part 1: Observable Bean Announcing PropertyChangeEvent

Transform each of your atomic shape classes into an observable bean that announces property change events. Study the class material to understand what this exactly means. Some of the steps you have to take include:

1. Making sure that each atomic class implements the interface util.models.PropertyListenerRegisterer.
2. Storing each registering PropertyChangeListener in a list.
3. Making each setter of a visible property announces an appropriate java.beans.PropertyChangeEvent instance to every observer in the list.

A property is visible if ObjectEditor sees it, that is, it has not been hidden by the @Visible annotation.

You don't have to create new subclasses to add this functionality – you can directly change existing classes. Try to make sure that the code you write to announce property change events is shared by as many classes as possible – ideally only the locatable and bounded shape classes should change.

As mentioned above, you are free to use any collection to store the list of observers including the ones presented in class lectures

Part 2: Observing Console Scene "View"

Create a class, tagged "ConsoleSceneView," that prints on the console each property event announced by each atomic shape in a scene object. Specifically, this class:

1. Implements the java.beans.PropertyChangeListener interface.
2. Provides a constructor that takes as a single argument an instance of the scene object and registers `this`, the current instance, as a listener of each atomic shape in the scene.
3. Uses println() to display on the Console each PropertyChangeEvent it receives from the atomic objects being observed.

This class is not really a view because it does not really display the scene in a meaningful way, but it does have the characteristics of such a view – hence the name. This class is much like the view class we saw in the praxis for this material.

Part 3: Animating Demoing Main Class

To demonstrate your observable and observables, write a main class that creates a scene object and displays an animation of it using both the console scene view and ObjectEditor. Specifically, the main class:

1. Instantiates a scene object.
2. Displays it using ObjectEditor.
3. "Displays" it using your console scene view.
4. Creates an animation that moves an avatar, sets its text, and rotates each of its rotatable parts (if you did extra credit). You should not call the OEFrame refresh method in this assignment and thus should not get any warning about invoking this method.

If you have followed all of the constraints of this and previous assignments, the avatar should animate without the refresh method, and every change to a visible shape property should be printed by the console view.

In particular, the animation will not work if you have moved an avatar or rotated an avatar limb by creating new parts of the avatar, instead of directly moving or rotating existing avatar components. As mentioned before, this solution is inefficient and is prohibited. If you have used this solution, use instead the approach of directly moving or rotating existing avatar components; and notify changes to the smallest property changed, as mentioned in class.

Debugging Refresh Problems

If you feel ObjectEditor is not automatically refreshing some changed observable on the screen, please contact us after going through the following check list:

1. Is your console view printing out the event from the object you think ObjectEditor missed?
2. Does the class of the object implement the PropertyListenerRegistarar interface?
3. If it does, is some ObjectEditor object and your console view calling the registration method defined by the interface? You can use print statements or break points to answer this question.
4. Is the method adding observers in a list?
5. Is the object sending the change information to ObjectEditor and other observers in its setters?

ObjectEditor frame may occasionally flicker – because of the amount of work ObjectEditor does, it cannot keep up with a high number of property changes.

Additional Constraints

1. Now that you understand multiple inheritance in interfaces, make sure that each class implements a single interface, which could extend multiple interfaces. This may mean that you will create empty interfaces that do nothing else beyond extending existing interfaces. This requirement reduces the need for casting.
2. A class tagged T should implement an interface tagged T. This requirement was optional in some of the previous assignments. You can ignore this requirement for the command/token classes.
3. As mentioned above and in class, you should not be sending notifications about composite shapes. (For example, when an avatar moves, do not send an observer such as ObjectEditor the whole avatar – instead send it the new location (x, y coordinate or Point object) of each atomic shape that moved.) An atomic shape can send notifications for all of its properties and a composite shape can send notifications about all of its atomic properties. For the graphics to update, we require you to send notifications only about properties of atomic properties.

   You should not be sending notifications about invisible properties - you might have to override inherited code from your locatable superclass to do turn these notifications off, which is ok, it implies invisible properties require special handling. If you do not follow this rule, you will get a message of the form:***Received notification(s) for unknown (possibly invisible) property: x of object: avatars.AnAvatar@44a8253a. Updating complete object.

This message implies that you do not know if the update to the screen occurred because of some correct or incorrect notification. To convince yourself and us that you meet the requirement of announcing changes to each atomic shape, get rid of these warnings. Often there is no need really to have invisible properties as you can simply not show the main panel. You should announce changes only to those properties that ObjectEditor knows about. It does not, of course, know about invisible properties. In addition, it does not know about non-standard properties of atomic shapes such as angle and radius. Finally, it regards Point as immutable. That means changes to X and Y properties of a point will result in the above warnings. If the Point is assigned to the standard location property of an atomic shape, you must announce a change to this property and send Point objects in the notification.

4. As also mentioned above, do not call the OEFrame refresh method.

Be sure to follow the constraints of the previous assignments.

Submission Instructions

- These are the same is in the previous assignment. The TAs will run the main method to see the test cases animate.
- Be sure to follow the conventions for the name and package of the main class.

Good luck!