Comp 401 - Assignment 8: MVC and Toolkits

Date Assigned: Fri Oct 12, 2012

Completion Date: Fri Oct 26, 2012

Early Submission Date: Wed Oct 24, 2012

In this assignment you will learn how to write observable Bean objects that are automatically refreshed by ObjectEditor. You will also learn how to use a toolkit (Swing/AWT) to create a widget tree and a controller. Extra credit will also let you create widget trees that change dynamically and implement views and create menu items, buttons, and progress bars. If you have time, implement a part of the regular and extra credit portion of your next assignment, and redo the scanner.

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.

Like previous assignments, the assignment may be changed in minor ways in response to student questions. So please look for tracked changes before you submit.

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.*

110 (maybe) and 401	MVC (10/8)	PowerPoint	PDF	MVC Chapter			lectures.mvc.monolithic Package lectures.mvc.interactor Package lectures.mvc Package
401	MVC: Properties and Collections (10/10)	PowerPoint	PDF	MVC Chapter			lectures.mvc.properties Package lectures.mvc.collections Package

401	MVC: Toolkits (10/12, 10/17)	PowerPoint	PDF	MVC Chapter			lectures.mvc.toolkit Package

Property Change Events

Currently, you must do explicit refreshes of the OZ simulation to see the results of the objects in the simulation. Use your new knowledge of observables, property change events, and ObjectEditor behavior to automatically update the avatars in response to execution of methods on them and their components. *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 a single class should change to move a complete avatar.

Some of 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. More important, it will be more difficult to automatically refresh such avatar operations. Therefore, use instead the approach of directly moving or rotating existing avatar components; and notify changes to the smallest property changed, as mentioned in class.

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

Toolkit and Command Interpreter User Interface

Use a Java toolkit to implement a user interface to manipulate the command interpreter. Use the model view controller design pattern to do so. You can use Swing or AWT Toolkit widgets discussed in class or other widgets to create your this user interface. The minimum requirement is that you provide a way to set the string manipulated by the interpreter. This means you must write a controller object that calls the setter. How the manually implemented user-interface look like is entirely up to you.

Looking Ahead (Optional)

Use the Swing/AWT toolkit to create a user interface that displays the Oz scene and reacts to changes to components in it. This means you must write a view object to display the objects in the scene and implement a paint() method in this view object. Your view object should ignore the OE annotations in the models it displays, but should render each of the required properties of a shape correctly. You are free to consider also the optional properties of the avatar such as color, stroke, and font. You probably have enough time to implement this part in this assignment, but since several of you are behind and it is midterm season, this part is not part of the current assignment, and will be graded in the next one. However, as mentioned above, it

may be a better use of your time to write your own OZ scene displayer than try and figure out why ObjectEditor does not seem to reach to your notifications.

Part of the extra credit for the next assignment will be to fix a problem in AWT/Swing – there is no notion of a paint listener. Define a paint listener interface, which should include a paint method – a method that takes an argument of type Graphics or Graphics2D. Now create a subclass of Component that provides a method to registers instances of this interface. Whenever the paint(Graphics) method is called in this subclass, it calls the paint method in each of the registered paint listeners. Let us call this subclass an *observable painter.*

Formatted: Font: Italic

You will no longer have a single monolithic view object that paints all of the objects in the scene. Instead, you will create multiple view objects –one for each avatar and one for the background. (You are free to create even finer-grained view objects). A view object will no longer be a subclass of Component . Instead it will be an observer or listener to the observable painter subclass you created. ~~.~~ The order in which the view objects get registered with the observable painter will determine whether a drawing is on top or bottom of another because it will determine the order in which the paint methods are called. A view object will receive property change events from the model object it paints and thus will also register itself as a listener with the model.

Extra Credit

Enhanced command interpreter user interface: If your interpreter has a read-only error property, then for extra credit, display it in your command interpreter user interface. This means you must now make the command interpreter an observable also and include a view that observsers ~~ers~~ this object.

Scanner user interface: Similarly, create a user interface to interact with an observable version of the scanner object. You can display the array in a TextArea or JTextArea , dynamically add and remove textfields to displays arrays of different sizes, or use some other mechanism to show this property. The *remove()* method of a container widget can be used to delete a component widget.

Again, you can directly modify the ~~parser and scanner classes~~existing classes to create observables.

Menu items: Add to your command interpreter user interface at least two menu items. Each of them should perform some canned (pre-programmed) action on the OZ simulation. For instance a menu item could move all of the avatars by some distance in the x direction and another could do so in the y direction.

Buttons: For additional credit create at least two buttons in the frame that perform the same or other canned actions on the scene. If your canned actions animate the avatar, you will not

Formatted: Font: Italic

actually see the intermediate steps in the animation for reasons we will study later. Feel free to create the pre-programmed animations though; later we will also see ways to overcome this problem.

Progress bar/slider: Connect a progress bar or slider to your main method that shows to what extent your animation in the main method has completed. You do not have to use MVC to display the progress bar but will get additional extra credit if you do so.

Formatted: Font: Italic

Object-oriented scanner: Now that you understand object oriented programming in more depth, you may want to revisit the extra credit feature I suggested in assignment 2: Write an object-oriented scanner along the lines suggested in the Iterators and Scanning section (links are reproduced below). This is difficult and will be worth a lot of points – do this only if you feel very comfortable with O-O programming. You can do this as part of any assignment. The points for this part will be assigned at the end of the semester based on whether we make it a regular assignment or not.

Formatted: Font: Italic

401 (maybe)	Iterators and Scanning (Nov 28, Dec 5)	PowerPoint	PDF	Iterator Chapter			lectures.iterator scanning Package

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. Does the class of the object define a registration method?
2. Does the class follow the conventions associated with the registration method – collection conventions for addVectorListener() and Bean conventions for addPropertyChangeListener()? (A class could follow both)?
3. If it does, is some ObjectEditor calling the method? You can use print statements or break points to answer this question.
4. Is it sending the change information to ObjectEditor?
5. In the next assignment, you will implement your own user interface to display the Oz scene. After the 10/17 lecture, you should know enough to implement this user interface. If you feel ObjectEditor is at fault, go ahead and implement this user interface if you have time, or simply print the notifications your observables send in response to changes to them.

Animating Demoing Main Class

To demonstrate your observables, write a main class that creates and displays an instance of the Oz scene; and creates an animation that moves the avatar, sets its text, and rotates each of its rotatable parts. You should not call the OEFrame refresh method in this assignment and thus should not get any warning about invoking this method.

To demonstrate the manual command interpreter user interface, make your main program **also** display the interpreter, and use this user interface to change the displayed Oz scene. Similarly, demonstrate the manual scanner user interface by writing additional code in the main method. Again, do not use the OEFrame refresh method. Your main code should call methods only in the models. This means that it will not cause any changes to the controller widgets such as the text field you created for entering commands. The TAs will test this text field manually. Changes made from the application program to a widget do not cause listener events to be fired. For example, calling setText() on a JTextField does not cause the listener event to be fired Thus calling widget methods in the main program does not help you demonstrate your widget listeners.

Formatted: Font: Bold

Additional Constraints

1. As mentioned above and in class, property change notifications should be sent only for atomic shapes (For example, when an avatar moves, do not send ObjectEditor the whole avatar – instead send it the new location of each atomic shape that moved.)
2. As also mentioned above, do not call the OEFrame refresh method.