Distributed Collaboration - Assignment 2.2: Transparent Glass Pane

Date Assigned: September 13, 2012

Completion Date: September 21, 2012

Objectives:

- Understand the nature of event processing
- Replace code patterns with abstractions.
- Finish implementation of a useful collaboration tool: A Telepointer

Dynamic Telepointer and Menus

Use JMenuBar, JMenu and JMenuItem to add a menu and two menu items that allow you to dynamically hide and display telepointers. You should be able do so by simply making the glass pane visible or invisible.

Event Processing

As mentioned in the last assignment, a glass pane blocks events from being passed to the components underneath. In particular, if a mouse or mouse motion listener is attached to the glass pane, then mouse or mouse motion events, respectively, are not passed to the components below.

There are two ways to address this problem. One is to not associate the glass pane with mouse or mouse motion listener, and associate it instead with an AWTEvent listener. This approach is illustrated in the following tutorial:

http://weblogs.java.net/blog/alexfromsun/archive/2006/09/a_wellbehaved_g.html

This is the more robust approach but requires you to translate AWTEvents into mouse and mouse motion events for the code that moves the telepointer. To do so, it must keep track of the state of the state of the mouse buttons. I have not tried to implement this approach

The other approach is to associate the glass pane with listeners, but make them redirect events to the components below. This is the approach suggested by the Java tutorial on glass panes:

http://docs.oracle.com/javase/tutorial/uiswing/components/rootpane.html

This tutorial does not address how events should be redirected to menus and menu items. The following tutorial gives an incomplete menu example.

http://www.java2s.com/Code/Java/Swing-JFC/Showhowaglasspanecanbeusedtoblockmouseandkeyevents.htm

I have implemented this approach and can make menus show and hide. Also if a menu is up and the left mouse button is down, I can make the menu items be selected. However, if a menu is being shown and no mouse button is down, I cannot make the items be selected. Your implementation can have this limitation. In this approach, I had to call the setVisible(false) method on the pop menu stored in a JMenu (getPopUpMenu() gives you the menu).

You are free to implement either approach. Having implemented the redispatching approach, I would recommend the other approach, as it is more fundamental, though I do not know its pitfalls. In either case, when the telepointer is active, it should be possible to pass events to the menu items and other components in the user interface. Use Piazza to exchange problems you encounter.

Drawing Abstraction

Make your telepointer component drawer an abstraction with an API that is actually used in the IM tool. This means it must have an API and be unaware of distribution issues. The abstraction could be used for a telepointer or for some other purpose. Do not extend the user interface of the telepointer component. For example, if it draws an oval shape as a telepointer, then when it becomes an abstraction, it could be used, for example , to create a bouncing ball that moves randomly around on the screen.

If you have implemented multiple telepointers, then a really relevant use of the drawing abstraction would be to create the status widget using this abstraction. You can create a common abstraction that combines the union of the functionality you need for displaying multiple telepointers and showing the status of multiple users. You can then use this abstraction in both applications. If you do this, you get extra credit.

Event Processing Abstraction

These tutorials suggest an organized approach to handling these events but require you to essentially use the tutorial code as a template. It is much more useful to create an application-independent abstraction. You should do so in this assignment.

In the re-dispatching case, create a redispatching class that takes in its constructor two arguments, a Java JFrame and a glass pane associated with the JFrame. It should add itself as a mouse and mouse motion listener to the glass pane and re-dispatch these events to components of the JFrame and its menu bar. It should not do any Telepointer processing; that should be done by separate TelePointer-specific listeners.

In the other, AWTEvent, case, create an event processing class that takes in its constructor a single argument, the glass pane. The class should define methods that allow other objects to register themselves as mouse and mouse motion listeners. It should track AWT events on the glass pane and call appropriate mouse and mouse motion callbacks in the registered listeners. Now a telepointer does not add listeners to the glass pane. Instead, it adds listeners to the instance of this (event processing) class. Again, I recommend this approach but have not implemented it.

Submission

By midnight September 21, submit, via a private or public Piazza message, a link to a YouTube video showing your enhanced telepointers in action and some (possibly trivial) application other than the IM tool using the drawing component as an abstraction.