Theory: Window listeners

13 minutes 0 / 5 problems solved

Skip this topic Start practicing

171 users solved this topic. Latest completion was about 21 hours ago.

Current topic:

Window listeners ···

Topic depends on:

× Swing components ···

§1. What are Window Listeners

Window listeners are a special type of event listener used to trigger code on events related to the user form displayed on the screen. Some examples of these types of events include closing a window, minimizing a window, or moving a window across the screen.

When we first introduced windows in Swing, we used a function to set the default close operation of the form.

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

This method is a simple example of a window listener. It is waiting for an event, the window closing, and takes action when that event occurs. Throughout this lesson, we will look at other window listeners that exist in Java, and see how they can be used in different applications.

§2. Types of Window Listeners

There are a number of different types of window listeners available to us in Swing. In this section, we will take a look at a few common listeners, and how they are used.

We can take action when the window is first opened using the `windowOpen` method. An example of this would be asking a user if they want to open an existing document or create a new document when they first open a text editor.

We can act on the window being closed by the user using the `windowClosing` and `windowClosed` methods. The `windowClosing` method is used to take action when a window is closed by a user. An example of this would be asking a user to confirm if they wish to close a document, or asking a user if they want to save before closing.

The `windowClosed` method is used once the window has completed closing, and is typically executed after the `windowClosing` method. An example of this would be loading a new window once the user has closed the current window.

If we want to trigger an event when a window is minimized, we can use the `windowIconified` and `windowDeiconified` methods. The `windowIconified` method is used to take action when a user minimizes a window. An example of this would be to silence notifications in a chat app when the app is minimized

The `windowDeiconified` method is the opposite of the `windowInconified` method. It takes action when a user opens a minimized window. From our previous example, a use of this could be to re-enable notifications when the chat app is opened again.

We can take action based on the active status of a window using the `windowActivated` and `windowDeactivated` methods. The `windowActivated` method is used to take action when a window becomes active. In Swing, the active window is the one that was last clicked on by the user. An example of this would be to clear an unread message notification once the window has been clicked.

The `windowDeactivated` method is used to take action when a window is not active. An example of this would be activated missed message notifications for a window.

§3. The WindowListener Interface

Table of contents:

One way that we can implement window listeners in our application is through the `WindowListener` interface. This interface will allow us to implement actions for each of the window listener methods we discussed in the last section.

Our first step is going to be creating a new class that implements the `WindowListener` interface. Once this is done, we will need to implement each of the window listener methods.

```
class WindowListenerExample implements WindowListener {
    ...

    public void windowOpened(WindowEvent e) {
        // a logic on the window open event
    }

    ... other methods of WindowListener interface
}
```

We will need to repeat the method implementation for each of the methods we discussed in the previous section. When doing this, the arguments remain the same, the only thing that changes is the method name itself. Since this is an interface, it requires for all methods to be implemented, which makes this a good option if we wish to implement all of the window listener methods.

If we want to now add these listeners into our form, we simply use the `addWindowListener` method.

```
JFrame demoFrame = new JFrame();
demoFrame.addWindowListener(new WindowListenerExample());
```

At this point, we now have a window, called `DemoFrame`, which contains all of the window listeners implemented in the `WindowListenerExample` class. From here, we can add any code to the `WindowListenerExample` class to get behavior on each of the window listener events we have discussed.

§4. Window Adapters

In the last example, we saw how we could implement the `WindowListener` interface in order to add window listeners to our application. You will notice that using this method, we need to implement every single window listener, due to the use of an interface. This is great if we want to write code for each window listener, however, if we only want one or two, this create unnecessary overhead.

To solve this problem, we can turn to the `WindowAdapter` class. Using this class, we can define window listeners as they are needed, rather than all at once. To do this, we simply change our implements for the window listener class to extend the `WindowAdapter` class instead of the `WindowListener` interface.

Once you have done this, you can add methods as they are required. For example, if we want to add a `windowClosing` method, we can add that single function.

```
class WindowAdapterExample extends WindowAdapter {
    public void windowClosing(WindowEvent e){

    }
}
```

Once the `WindowAdapter` is declared, it can be added to a frame in the exact same way as the `WindowListener` interface.

```
JFrame demoFrame = new JFrame();
demoFrame.addWindowListener(new WindowAdapterExample());
```

In general, it is better to use the `WindowAdapter` class when you don't plan to implement all of the window listeners. The `WindowListener` interface exists due to the fact that many applications will extend all of the listeners, so if this is the case for your application, it is a good option.

§5. Implementing a Confirmation Dialog

To give a practical example for window listeners, we will take a look at how to create a dialog that confirms if a user really wants to exit an application. To do this, we will need three components:

1. A confirmation dialog, that allows a user to select yes or no, and take action based on the input
2. A `windowClosing()` listener to activate the confirmation dialog
3. A main window to attach the window listener to.

We will start by implementing the confirmation dialog. This component will have two pieces, the window, and an action listener to determine which button was pressed. We will start by creating the window layout. This code will create a label, as well as two buttons. The label will ask the user if they wish to exit the application, and if they select "Yes" the application will close. If they select "No", then the application will stay open. Along with this, we will implement an action listener, which will cause the application to exit if the Yes button is pressed. If the No button is pressed, it will dispose of the confirmation dialog, and keep the main window open. This is all of the logic required for the confirmation dialog.

To add the window listener, we will use a simple `WindowAdapter`, that creates the confirmation dialog when the window is going to close.

For our main window, we can create a very basic window. The only thing we need to make sure to do is add the window listener and set the window to do nothing when the exit button is pressed. Doing so will allow the window listener to take over the event. To do this, we will start with setting up our `WindowAdapter`function for handling the `windowClosing`event.

```
public class CheckExitExample extends JFrame {
    private class CheckOnExit extends WindowAdapter {
        public void windowClosing(WindowEvent e) {
            ConfirmWindow checker = new ConfirmWindow();
            checker.setVisible(true);
        }
    }
```

After this, we can declare our `JFrame` and setup our GUI and listeners.

```java
private class ConfirmWindow extends JFrame implements ActionListener {
        public ConfirmWindow() {
            setSize(250,100);
            setLayout(new BorderLayout());

            JLabel confirmLabel = new JLabel(
"Are you sure you want to exit?", SwingConstants.CENTER);
            add(confirmLabel, BorderLayout.CENTER);

            JPanel buttonPanel = new JPanel();
            buttonPanel.setLayout(new FlowLayout());

            JButton exitButton = new JButton("Yes");
            exitButton.addActionListener(this);
            buttonPanel.add(exitButton);

            JButton cancelButton = new JButton("No");
            cancelButton.addActionListener(this);
            buttonPanel.add(cancelButton);

            add(buttonPanel,BorderLayout.SOUTH);
        }
```

We will attach an actionPerformed function to the JFrame to check if the user has exited the form or not.

```java
        public void actionPerformed(ActionEvent e) {
            String action = e.getActionCommand();

            if (action.equals("Yes")) {
                System.exit(0);
            } else if (action.equals("No")) {
                dispose();
            }
        }
    }
```

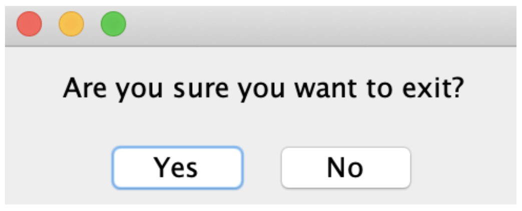
From here, we can simply declare our main function, as well as the class to handle our GUI functionality, and we are done.

```java
    public static void main(String[] args) {
        CheckExitExample demoWindow = new CheckExitExample();
        demoWindow.setVisible(true);
    }

    public CheckExitExample() {
        setSize(300,300);
        setTitle("Window Listener");
        setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        addWindowListener(new CheckOnExit());

    }
}
```

Once this is completed, attempting to exit the form will produce a dialogue asking to confirm exit.


Are you sure you want to exit?

§6. Conclusions

Here is a recap:

- Window listeners can be used to take action on the event that the user interacts with the window
- The `WindowListener` interface can be used to implement window listeners, but must implement all of the methods in the interface
- To implement only a few window listeners, the `WindowAdapter` class can be used

Report a typo

22 users liked this theory. 2 didn't like it. **What about you?**

Start practicing

Comments (2) Hints (0) Useful links (0) Show discussion