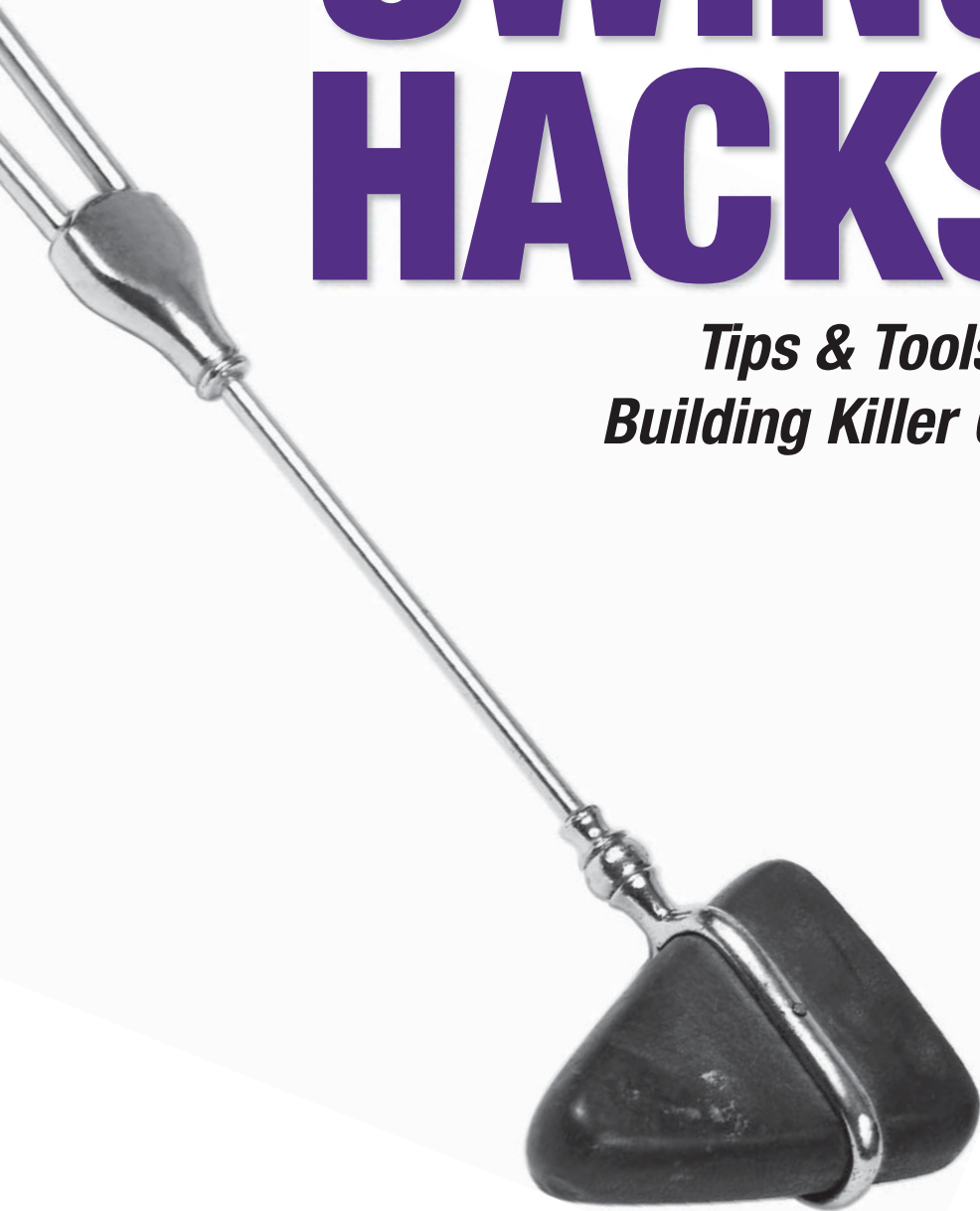


SWING HACKS™

*Tips & Tools for
Building Killer GUIs*



O'REILLY®

Joshua Marinacci & Chris Adamson

HACK
#38

Earthquake Dialog

Make sure your users really know they got their password wrong.

One of the funny ways that Mac OS X uses animation in its UI is when the user logs in. If she enters an incorrect login and/or password, the whole login dialog shakes violently for a second, like a road sign thwacked with a bat, or a cartoon character who has just run full-speed into a solid object (say, a picture of a tunnel painted over a wall).

We like this effect a lot, so we thought we'd bring it to Swing. It's a pretty straightforward bit of animation, so we jazzed it up...with *trigonometry!*

Exterior Animation

Here's an initial question: do you want to subclass `JDialog` and add the animation effect to that class, or create a class that animates the shaking on another `JDialog`? I thought that subclassing would be a bad choice because `JOptionPane` generates some very convenient `JDialogs`, and you wouldn't want to lose those. So, you'll have to have another class animate your dialogs. I've called it `DialogEarthquakeCenter` because it'll be a class that monitors the shaking, just like seismologists do in their earthquake centers.

Obviously, the `DialogEarthquakeCenter` needs a reference to the dialog that it will be shaking. It also needs a few other values, which I've set as constants:

`SHAKE_DISTANCE`

The maximum distance in each direction the dialog should move.

`SHAKE_CYCLE`

The time in milliseconds for a complete cycle: center, right, center, left, back to center.

`SHAKE_DURATION`

Total time in milliseconds to shake the dialog.

`SHAKE_UPDATE`

How often (in milliseconds) to update the dialog's position and repaint. You might increase this if the CPU use is excessive, but animation smoothness decreases with less-frequent updates.

Beyond that, all you'll need to keep track of is where the dialog started (so you can put it back at the end of the animation), a running clock of how far you are into the animation, and where the dialog is located. [Example 5-5](#) shows the code to put all this into action.

Example 5-5. A class to shake a JDialog back and forth

```
public class DialogEarthquakeCenter extends Object {

    public static final int SHAKE_DISTANCE = 10;
    public static final double SHAKE_CYCLE = 50;
    public static final int SHAKE_DURATION = 1000;
    public static final int SHAKE_UPDATE = 5;

    private JDialog dialog;
    private Point naturalLocation;
    private long startTime;
    private Timer shakeTimer;
    private final double HALF_PI = Math.PI / 2.0;
    private final double TWO_PI = Math.PI * 2.0;

    public DialogEarthquakeCenter (JDialog d) {
        dialog = d;
    }

    public void startShake() {
        naturalLocation = dialog.getLocation();
        startTime = System.currentTimeMillis();
        shakeTimer =
            new Timer(SHAKE_UPDATE,
                new ActionListener() {
                    public void actionPerformed (ActionEvent e) {
                        // calculate elapsed time
                        long elapsed = System.currentTimeMillis() -
                            startTime;
                        // use sin to calculate an x-offset
                        double waveOffset = (elapsed % SHAKE_CYCLE) /
                            SHAKE_CYCLE;
                        double angle = waveOffset * TWO_PI;

                        // offset the x-location by an amount
                        // proportional to the sine, up to
                        // shake_distance
                        int shakenX = (int) ((Math.sin (angle) *
                            SHAKE_DISTANCE) +
                            naturalLocation.x);
                        dialog.setLocation (shakenX, naturalLocation.y);
                        dialog.repaint();

                        // should we stop timer?
                        if (elapsed >= SHAKE_DURATION)
                            stopShake();
                    }
                }
            );
        shakeTimer.start();
    }
}
```

Example 5-5. A class to shake a `JDialog` back and forth (continued)

```
public void stopShake() {
    shakeTimer.stop();
    dialog.setLocation (naturalLocation);
    dialog.repaint();
}

public static void main (String[] args) {
    JOptionPane pane =
        new JOptionPane ("You've totally screwed up your login\n" +
            "Go back and do it again... and do you think\n" +
            "you could remember your password this time?",
            JOptionPane.ERROR_MESSAGE,
            JOptionPane.OK_OPTION);
    JDialog d = pane.createDialog (null, "Shakin'!");
    DialogEarthquakeCenter dec = new DialogEarthquakeCenter (d);
    d.pack();
    d.setModal (false);
    d.setVisible(true);
    dec.startShake();

    // wait (forever) for a non-null click and then quit
    while (pane.getValue() == JOptionPane.UNINITIALIZED_VALUE ) {
        try { Thread.sleep(100); }
        catch (InterruptedException ie) {}
    }
    System.exit(0);
}
}
```

The class includes the constants described earlier, along with:

- A reference to the `JDialog` to be animated
- The dialog's natural location (i.e., its location before the animation begins)
- The time that the animation began (for calculating offsets)
- A `javax.swing.Timer` to run the animation
- Some trigonometry constants

The advantage of the `Swing Timer` is, of course, that it keeps the repainting on the event-dispatch thread and thus keeps it thread-safe. There's a little bit of non-Swing code executed by the timer to calculate the position, but it's not so bad that you have to worry about [blocking the GUI \[Hack #92\]](#).

The constructor is trivial—it just remembers the dialog as an instance variable. The real fun begins in the `startShake()` method. It starts by storing the current time and location; the time is for use in the animation, and the location is for cleaning up later. Next, it creates the `javax.swing.Timer` and sets it to fire every `SHAKE_UPDATE` milliseconds.

Now for some math and the methodology behind moving the dialog. Ignoring friction, air resistance, and other real-world factors—this is a dialog box in the fantasy world of the desktop after all—I opted for *simple harmonic motion*, which is motion that can be expressed by a sine function and is not driven or dampened externally. Values of the sine function range from -1 to 1, so the dialog's horizontal offset can be expressed at the sine of some value from 0 to 2π , multiplied by the maximum offset (namely `SHAKE_DISTANCE`).

Trig? Really?

Let me take a moment to explain why I'm forcing you to think about trigonometry in a Swing book. When you knock something back and forth, like a tuning fork or a pendulum, it doesn't move at a constant speed to one extreme, then stop and immediately move at a constant speed in the other direction. It slows down as it reaches the extreme, stops momentarily, and then accelerates in the other direction, moving faster until it crosses the center, at which point it starts decelerating. You need more than simple addition to model this—and trig is the ticket.

So, that's what happens in the `actionPerformed()` callback. The method takes the elapsed time and figures out how far into a cycle it is (given a cycle time of `SHAKE_CYCLE`), expressed as a `double` between 0 and 1. Multiply this by 2π , and you've got an angle you can pass to `Math.sin()`. Multiply that by `SHAKE_DISTANCE`, and you'll have an x-offset in the range $-\text{SHAKE_DISTANCE} \leq n \leq \text{SHAKE_DISTANCE}$. Add that to the `naturalLocation`'s x-value, keep the natural y-value, and you have the new location for the dialog. Call `setLocation()` with this point and `repaint()`.

`actionPerformed()`'s only other responsibility is to check to see if the animation time has expired and, if so, to call the `stopShake()` method, which is public and could thus be called by an outsider to end the animation prematurely. `stopShake()` stops the `Timer`, returns the dialog to its natural location, and `repaint()`s.

Shake, Rattle, and Roll

I've provided a `main()` method to demonstrate the `DialogEarthquakeCenter`. To show its flexibility, I made it shake a `JOptionPane` dialog, to prove you can still use option dialogs as well as normal dialogs, although you do need to work with option dialogs slightly differently:

```
public static void main (String[] args) {
```

Earthquake Dialog

```
JOptionPane pane =
    new JOptionPane ("You've totally screwed up your login\n" +
        "Go back and do it again... and do you think\n" +
        "you could remember your password this time?",
        JOptionPane.ERROR_MESSAGE,
        JOptionPane.OK_OPTION);
JDialog d = pane.createDialog (null, "Shakin'!");
DialogEarthquakeCenter dec = new DialogEarthquakeCenter (d);
d.pack();
d.setModal (false);
d.setVisible(true);
dec.startShake();

// wait (forever) for a non-null click and then quit
while (pane.getValue() == JOptionPane.UNINITIALIZED_VALUE ) {
    try { Thread.sleep(100); }
    catch (InterruptedException ie) {}
}
System.exit(0);
}
```

The `main()` method builds a `JOptionPane` dialog through what can only be called “the other way.” Most developers will call `JOptionPane.show...Dialog()` because it’s convenient to get the dialog on screen immediately and provide the user’s selection as a return value, and because it’s not necessary to ever have a reference to the dialog. With `DialogEarthquakeCenter`, however, you need that reference. So, instead, you provide the usual `JOptionPane` values (message, message type, user options, etc.) to the `JOptionPane` constructor, and then derive a dialog with `createDialog()`. Don’t worry—you still end up with the same option dialog.

Next, you create a `DialogEarthquakeCenter` from that dialog. Then you can return to the dialog, `pack()` it, and make it visible. One hazard of working with the `JOptionPane` dialog is that it is modal, meaning it will block the AWT event-dispatch thread when shown; thus, it won’t shake because `DialogEarthquakeCenter` won’t get any animation callbacks from `Timer`. To get around this, I just made the dialog non-modal—after all, the user probably isn’t going to be able to click it when it’s moving. Another option might be for the `DialogEarthquakeCenter` constructor to remember if its dialog is modal, and to reset this state in `stopShake()`.

Unfortunately, there’s not a great way to show the effect of this animation in book form. The best I can suggest is that you compile and run the hack for yourself. While you’re at it, change some of the constant values to see the effect of longer or shorter shake cycles, distances, and durations.