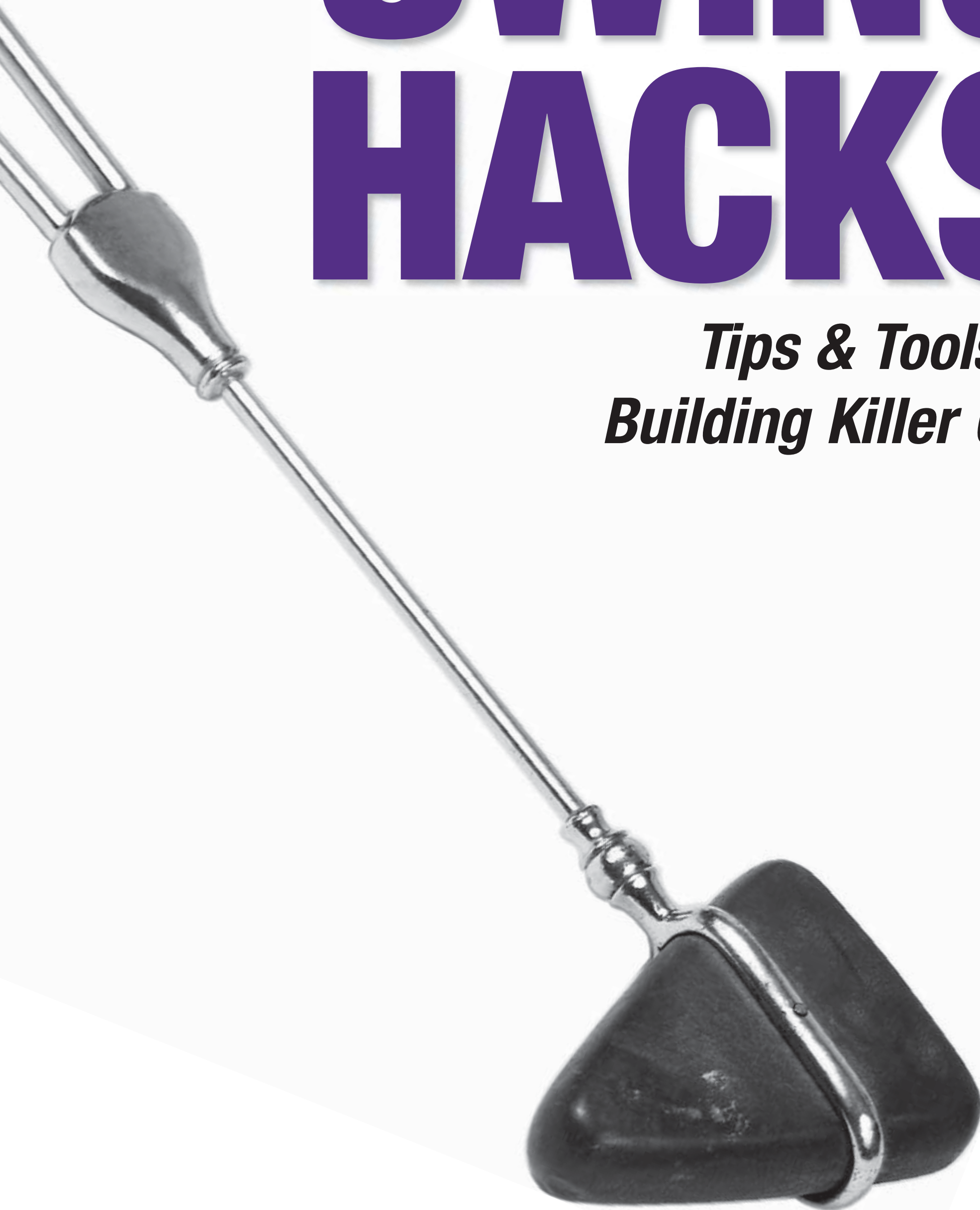
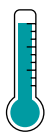
SWING HACKS™

Tips & Tools for Building Killer GUIs



O'REILLY®

Joshua Marinacci & Chris Adamson



HACK #59 Create a Color Eyedropper

Enhance your color pickers with an eyedropper tool that grabs a color from anywhere on the screen.

Most paint tools give you an eyedropper, but I've never seen a Java program do it. Getting a screen pixel requires native access, which is usually blocked off from Java programs. Java 1.3 introduced a new method to the `Robot` class, `getPixelColor( )`, which can retrieve the color anywhere on the screen. The problem is that you don't get mouse events once the cursor leaves your `JFrame`. This is fine if you only want to select colors from your own application, but a color chooser needs to select from anywhere on the screen. Java 5.0 introduces new APIs for getting complete mouse events, but that doesn't help us today.

The answer to this tricky problem, of course, is to cheat! This hack makes a screenshot and then paints it into a `JFrame` called `ColorChooserDemo`, which fills the entire screen. The screenshot is indistinguishable from the real desktop except that nothing in the background updates. However, since the screenshot is only needed while the user selects a color, this should be OK. `ColorChooserDemo` also has a `JLabel` in the center of the screen, which displays the currently selected color. Once the user has finished selecting a color by releasing the mouse, the entire frame will disappear and the component that launched the chooser—usually a `JButton`—will get the color through `setBackground( )`. While it's running, `ColorChooserDemo` looks like Figure 8-3.

Figure 8-3. Running the ColorChooserDemo

The first step is to set up the required components. The ColorChooserDemo is a subclass of JFrame with member variables to hold the screenshot (background_image), the panel to draw the image (image_panel), the JLabel to display the current color under the cursor (label), and a few support variables. The beginnings of this class are shown in Example 8-6.

Example 8-6. Skeleton of the ColorChooserDemo class

```
public class ColorChooserDemo extends JFrame
    implements MouseListener, MouseMotionListener {

    JPanel image_panel;
    Dimension screen_size;
    JComponent comp = null;
    Image background_image = null;
    Robot robot;
    JLabel label;

    public ColorChooserDemo(JComponent comp) {
        // get the screen dimensions
        screen_size = Toolkit.getDefaultToolkit().getScreenSize( );

        // set up the frame (this)
        this.addMouseListener(this);
        this.addMouseMotionListener(this);
        this.comp = comp;
        this.setUndecorated(true);
        this.setSize(screen_size.width, screen_size.height);

        // set up the panel that holds the screenshot
        image_panel = new JPanel( ) {
            public void paintComponent(Graphics g) {
                super.paintComponent(g);
                g.drawImage(background_image,0,0,null);
            }
        };
        image_panel.setPreferredSize(screen_size);
        this.getContentPane( ).add(image_panel);

        // set up the display label
        label = new JLabel("Selected Color");
        label.setOpaque(true);
        label.setSize(100,100);
        image_panel.setLayout(null);
        image_panel.add(label);
        label.setLocation((int)screen_size.getWidth( )/2 - 50,
            (int)screen_size.getHeight( )/2 - 50);
    }
```

In its constructor, the ColorChooserDemo accepts a JComponent to store the selected color in. Next, the code gets the current screen size from the AWT Toolkit, and then follows the usual litany of listeners and setters. Note the call to setUndecorated(true), which turns off the window controls. This adds to the illusion that the user is clicking on the real system desktop and not a screenshot.

The image_panel is a standard JPanel with the paintComponent( ) method overridden to draw the screenshot image over its background. It is also set to fill the screen with setPreferredSize(screen_size), and then is added to the frame.

Before returning, the ColorChooserDemo constructor creates a 100 × 100 pixel JLabel to display the current selected color. By default, the label would let its parent component (the screenshot) show through instead of filling its background with the selected color, so the code calls setOpaque(true) to make sure the background is visible. Finally, the label is moved to the middle of the screen, calculated by dividing the screen dimensions in half and subtracting half of the label size. Of course a LayoutManager would mess with the explicit coordinates set here, so image_panel's layout is set to null. This gets rid of the default layout manager, BorderLayout, and allows the absolute positioning to work.

Now that the chooser frame and its components are set up, the frame needs to make the actual screenshot. ColorChooserDemo overrides the show( ) method to make the screenshot before the frame pops up on screen. The show( ) method uses the robot.createScreenCapture( ) to capture and save the screen to the background_image variable before passing control to the superclass, as shown in the following code:

```
public void show( ) {
    try {
        // make the screenshot before showing the frame
        Rectangle rect = new Rectangle(0,0,
             (int)screen_size.getWidth( ),
             (int)screen_size.getHeight( ));
        this.robot = new Robot( );
        background_image = robot.createScreenCapture(rect);
        super.show( );
    } catch (AWTException ex) {
        System.out.println("exception creating screenshot:");
        ex.printStackTrace( );
    }
}
```

Once the ColorChooserDemo frame is visible, the user can begin selecting colors by clicking and dragging anywhere on the (now fake) screen. The mousePressed( ), mouseDragged( ), and mouseReleased( ) methods of the mouse/

mouse-motion listener implementation update the selected color on each mouse event. setSelectColor( ) does the actual update by setting the background color on both the label (which the user can see) and the component that was passed into the constructor (currently hidden behind the frame):

```
// update the selected color on mouse press, dragged, and release
public void mousePressed(MouseEvent evt) {
    setSelectedColor(robot.getPixelColor(evt.getX(), evt.getY( )));
}
public void mouseDragged(MouseEvent evt) {
    setSelectedColor(robot.getPixelColor(evt.getX(), evt.getY( )));
}
// for released we want to hide the frame as well
public void mouseReleased(MouseEvent evt) {
    setSelectedColor(robot.getPixelColor(evt.getX(),evt.getY( )));
    this.setVisible(false);
}

// update both the display label and the component that was passed in
public void setSelectedColor(Color color) {
    comp.setBackground(color);
    label.setBackground(color);
}

// no-ops for the rest of the mouse-event listener
public void mouseClicked(MouseEvent evt) { }
public void mouseEntered(MouseEvent evt) { }
public void mouseExited(MouseEvent evt) { }
public void mouseMoved(MouseEvent evt) { }
```

When the user releases the mouse, the mouseReleased( ) method will do one last color update and then hide the frame. This way, when the user is done selecting a color, the final color will be visible as the background of the launching component, as seen in Figure 8-4.

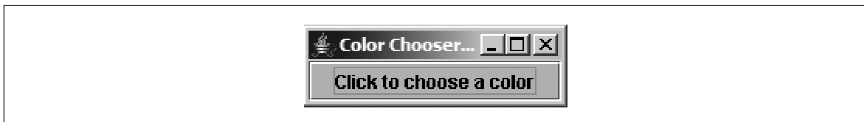


Figure 8-4. After a color is chosen

Launching the demo just requires a component to call show( ) on the ColorChooserDemo:

```
public static void main(String[] args) {
    JFrame frame = new JFrame("Color Chooser Hack");
    final JButton button = new JButton("Click to choose a color");
    button.addActionListener(new ActionListener( ) {
        public void actionPerformed(ActionEvent evt) {
            JFrame frame = new ColorChooserDemo(button);
            frame.show( );
```

```
            }
        });

        frame.getContentPane( ).add(button);
        frame.pack( );
        frame.setVisible(true);
    }
```

And that's it! Now you can add full-screen color choosing to any component without requiring native access at all. As an improvement, you could make the preview actually show a magnified view of where the cursor is instead of just the selected color.