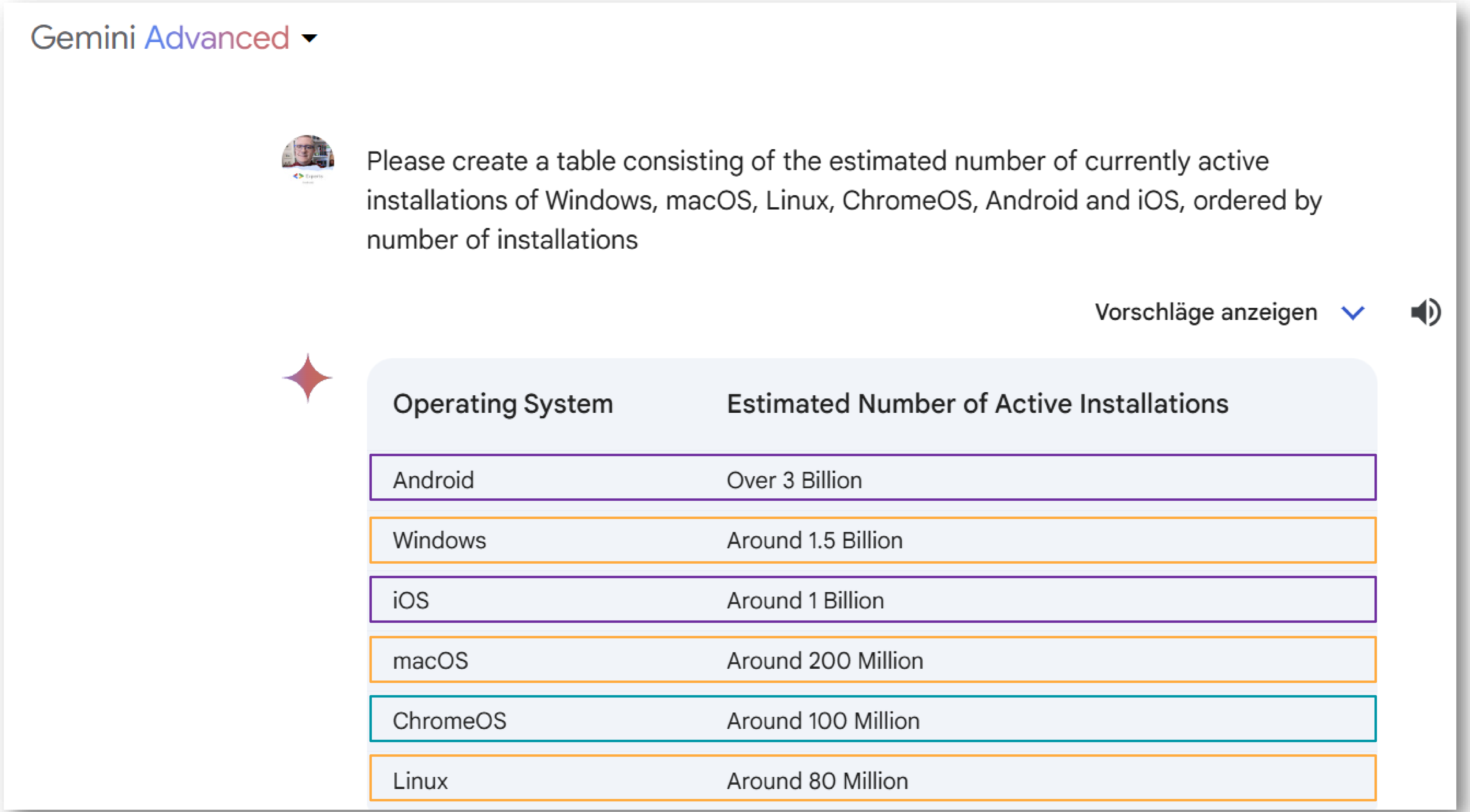
An introduction to Compose Multiplatform

Thomas Künneth

Experts

SnappMobile



Gemini Advanced

Please create a table consisting of the estimated number of currently active installations of Windows, macOS, Linux, ChromeOS, Android and iOS, ordered by number of installations

Operating System	Estimated Number of Active Installations
Android	Over 3 Billion
Windows	Around 1.5 Billion
iOS	Around 1 Billion
macOS	Around 200 Million
ChromeOS	Around 100 Million
Linux	Around 80 Million

That’s quite a user base. Why don’t we write apps for the Desktop anymore?

https://de.m.wikipedia.org/wiki/Datei:Netscape_icon.svg

Because the Browser and Web technology have won

But why do we put so much effort in wrapping Web Apps in native containers?

- Browsers (and Web Apps) achieved a lot in the last 15 years, but a few things remain cumbersome
    - Local storage
    - Background activities
- Leaving the confinements of the Browser allows for an better platform integration
    - File associations
    - Local search
    - Multiple windows

🥳 So, writing native apps for the Desktop may be a good idea

🤔 But how do we write apps for the Desktop these days?

Writing apps for the Desktop

- Native frameworks and tools
  - Make use of all platform features
  - Each platform has its own programming languages, tools, best practices
- Cross platform frameworks
  - Significant code sharing
  - Less platform-specific knowledge needed
  - Satisfactory platform integration possible (depending on the framework)

- *Cross platform vs. native* debate started many years ago
- Both approaches have distinct advantages and shortcomings
- On the Desktop, Cross platform frameworks are a good idea because writing truly native apps for all ecosystems likely isn’t feasible today

Noteworthy Cross Platform frameworks

- Electron
- Qt
- Flutter
- Compose Multiplatform

What about Java?

- Is Java a Cross Platform Framework or a platform?
- General consensus: Java is a platform, like the Browser
- In the end, it doesn’t matter, because either way
    - Apps run on multiple operating systems
    - One programming language is used
    - One tool chain is used

- Java feels like having been around forever
- Used to be a pain to get the JRE onto a machine
- Most users hated it (sorry!)

A lot has changed since then

- No Java plugin for the Browser
- No local JRE installation
- No Java Web Start

OK, so let’s use ~~Java~~ the JVM and just don’t tell our users

Compose Desktop

- Framework for building Desktop apps using Kotlin and Jetpack Compose
- Targets macOS, Linux and Windows
- Utilizies significant parts of the Java toolchain (`jpackage`)
- Apps run inside the JVM
- Part of a bigger picture: Compose Multiplatform

Compose Multiplatform

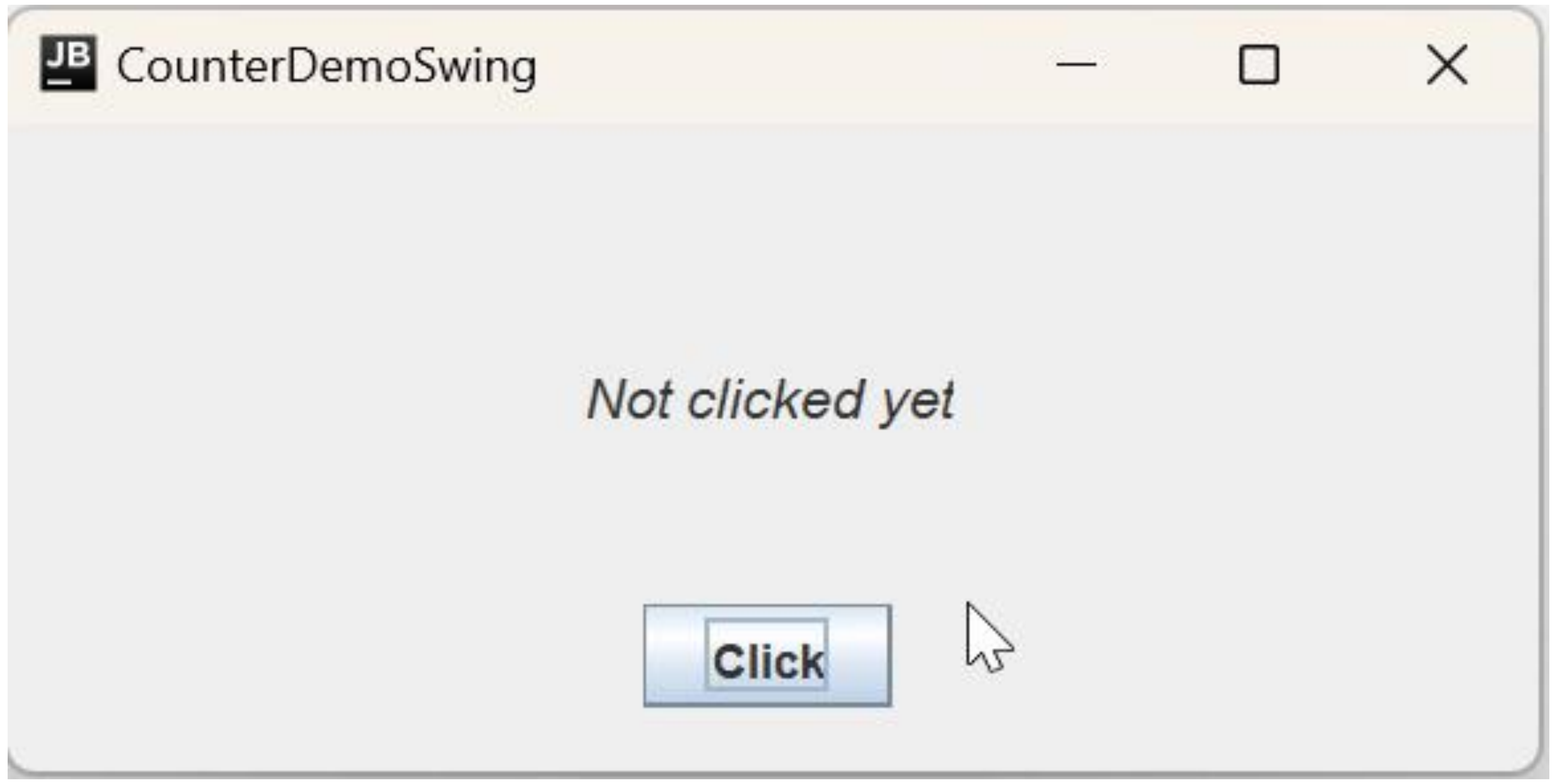
- UIs are built using Jetpack Compose and shared across supported platforms
  - Android
  - iOS
  - Desktop (Linux, macOS, Windows)
  - Web
- Based on Kotlin Multiplatform

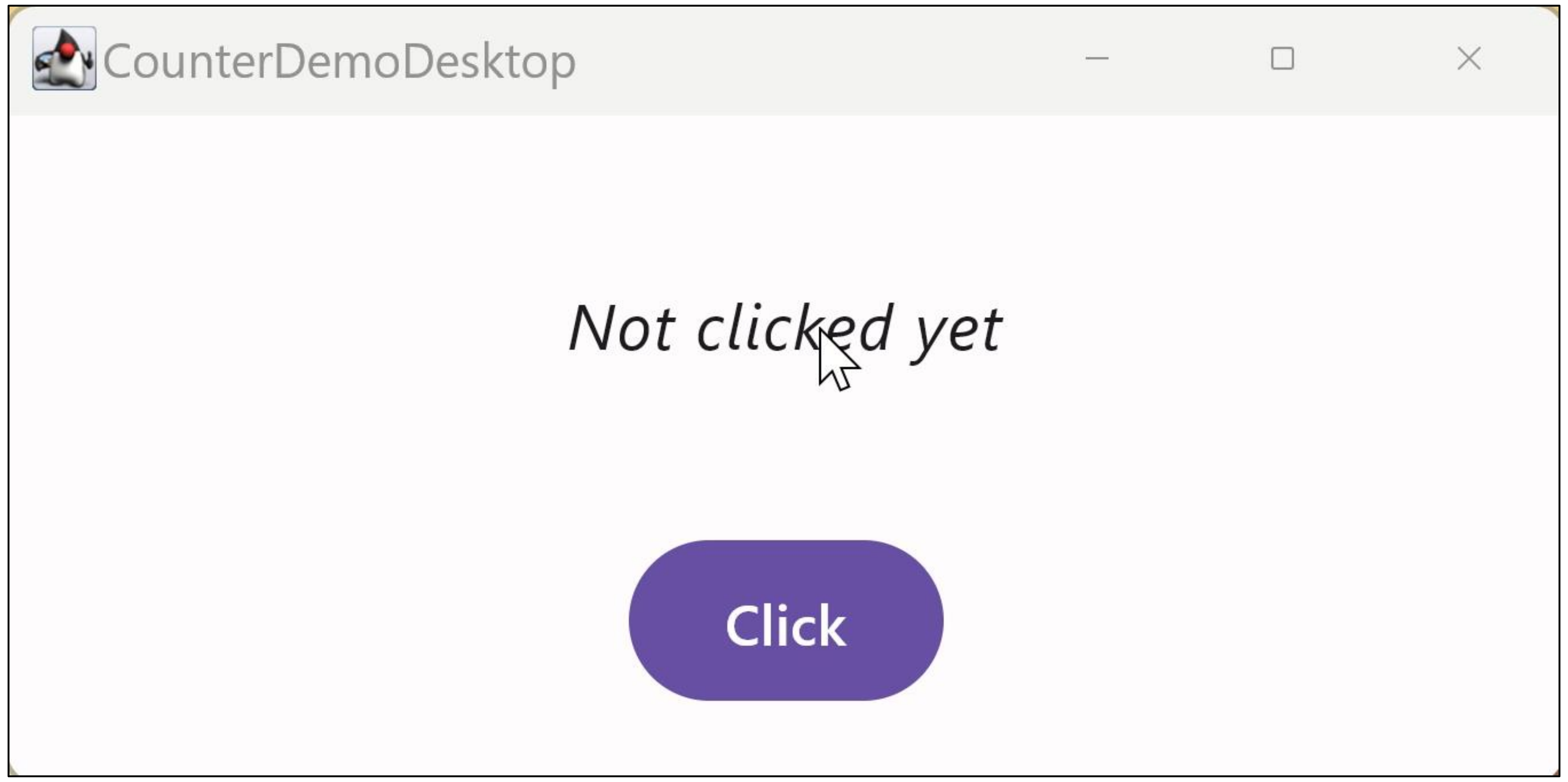
Kotlin Multiplatform

- Share Kotlin code across platforms
  - Compiler output depends on the target
  - Common libraries available on all platforms
- Part of Kotlin Multiplatform (KMP):
  - Project templates
  - Project structure
  - Gradle plugins
  - Language constructs
  - Libraries

Jetpack Compose

- (Started as) Androids new **declarative** UI framework
- UI is built by nesting **composable functions**
  - Kotlin top-level functions
  - Annotated with `@Composable`
  - Usually return `Unit`
- Compose hierarchies are hosted in native containers


CounterDemoSwing
Not clicked yet
Click

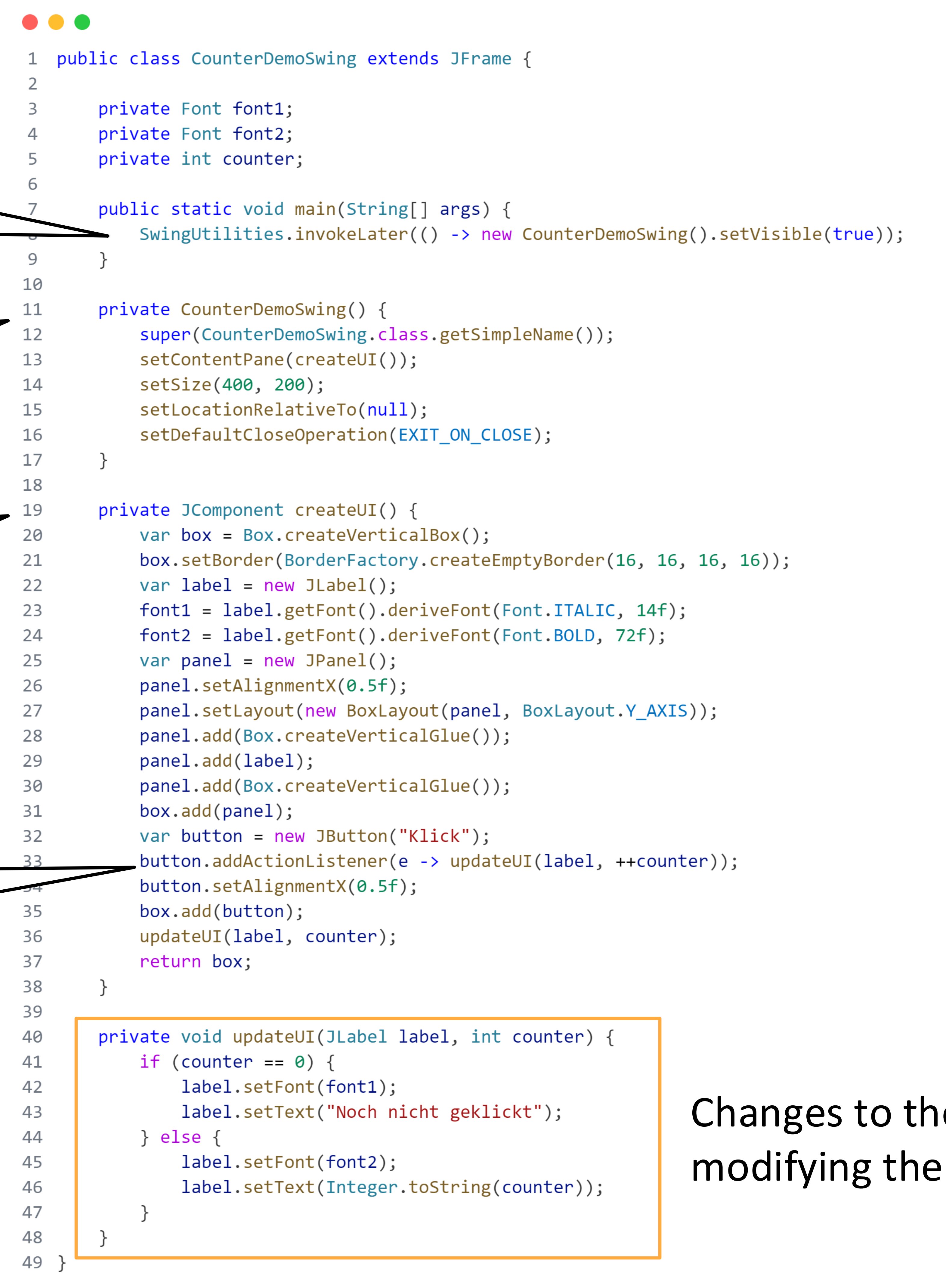

CounterDemoDesktop
Not clicked yet
Click

Make it visible

Add it to the main window

Define a component tree

Wire up behavior



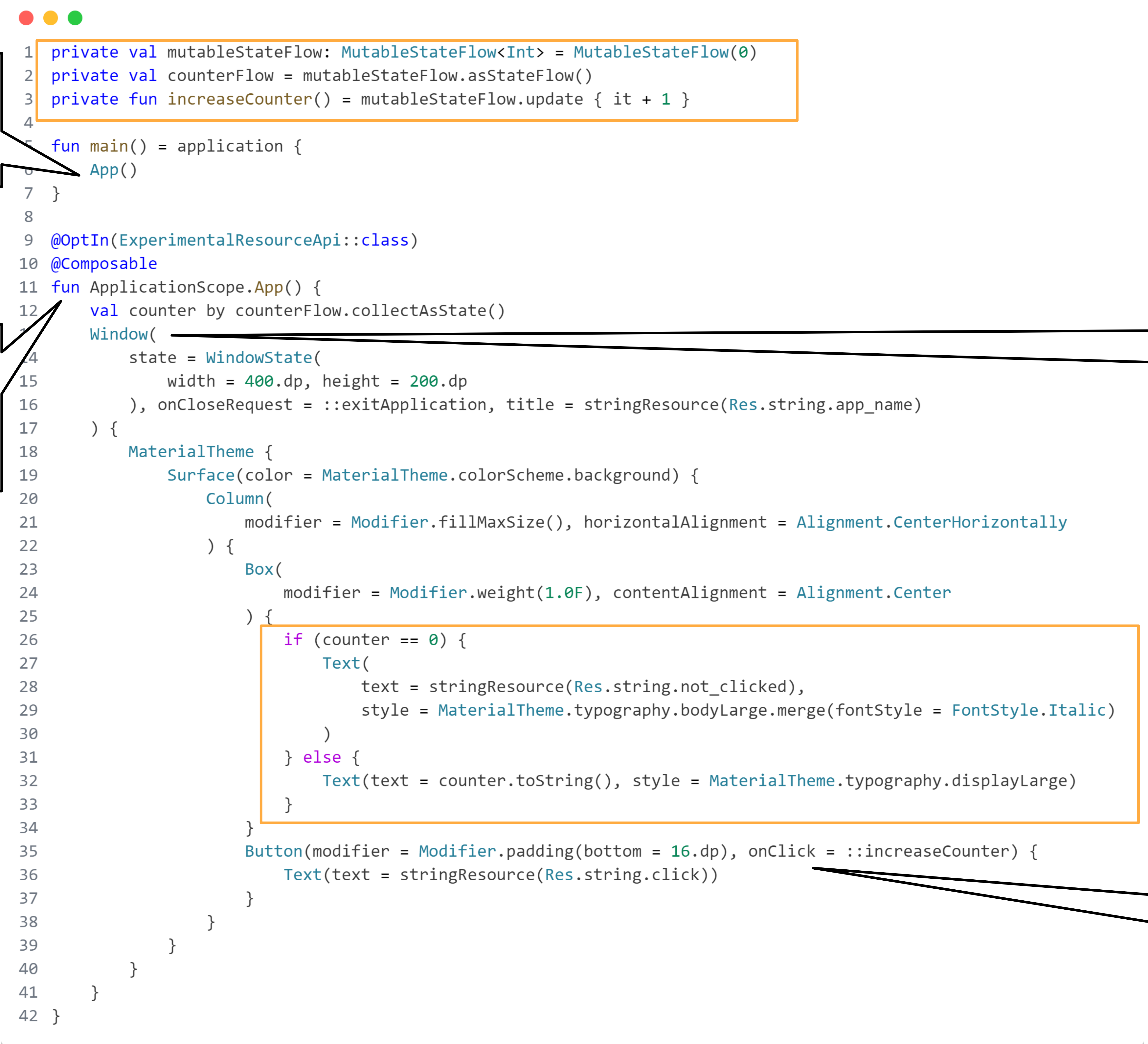
```java
public class CounterDemoSwing extends JFrame {

    private Font font1;
    private Font font2;
    private int counter;

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new CounterDemoSwing().setVisible(true));
    }

    private CounterDemoSwing() {
        super(CounterDemoSwing.class.getSimpleName());
        setContentPane(createUI());
        setSize(400, 200);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    private JComponent createUI() {
        var box = Box.createVerticalBox();
        box.setBorder(BorderFactory.createEmptyBorder(16, 16, 16, 16));
        var label = new JLabel();
        font1 = label.getFont().deriveFont(Font.ITALIC, 14f);
        font2 = label.getFont().deriveFont(Font.BOLD, 72f);
        var panel = new JPanel();
        panel.setAlignmentX(0.5f);
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
        panel.add(Box.createVerticalGlue());
        panel.add(label);
        panel.add(Box.createVerticalGlue());
        box.add(panel);
        var button = new JButton("Klick");
        button.addActionListener(e -> updateUI(label, ++counter));
        button.setAlignmentX(0.5f);
        box.add(button);
        updateUI(label, counter);
        return box;
    }

    private void updateUI(JLabel label, int counter) {
        if (counter == 0) {
            label.setFont(font1);
            label.setText("Noch nicht geklickt");
        } else {
            label.setFont(font2);
            label.setText(Integer.toString(counter));
        }
    }
}
```

Changes to the UI are made by modifying the component tree

Experts

Make it visible

Define a hierarchy of composable functions



```kotlin
private val mutableStateFlow: MutableStateFlow<Int> = MutableStateFlow(0)
private val counterFlow = mutableStateFlow.asStateFlow()
private fun increaseCounter() = mutableStateFlow.update { it + 1 }

fun main() = application {
    App()
}

@OptIn(ExperimentalResourceApi::class)
@Composable
fun ApplicationScope.App() {
    val counter by counterFlow.collectAsState()
    Window(
        state = WindowState(
            width = 400.dp, height = 200.dp
        ), onCloseRequest = ::exitApplication, title = stringResource(Res.string.app_name)
    ) {
        MaterialTheme {
            Surface(color = MaterialTheme.colorScheme.background) {
                Column(
                    modifier = Modifier.fillMaxSize(), horizontalAlignment = Alignment.CenterHorizontally
                ) {
                    Box(
                        modifier = Modifier.weight(1.0F), contentAlignment = Alignment.Center
                    ) {
                        if (counter == 0) {
                            Text(
                                text = stringResource(Res.string.not_clicked),
                                style = MaterialTheme.typography.bodyLarge.merge(fontStyle = FontStyle.Italic)
                            )
                        } else {
                            Text(text = counter.toString(), style = MaterialTheme.typography.displayLarge)
                        }
                    }
                    Button(modifier = Modifier.padding(bottom = 16.dp), onClick = ::increaseCounter) {
                        Text(text = stringResource(Res.string.click))
                    }
                }
            }
        }
    }
}
```

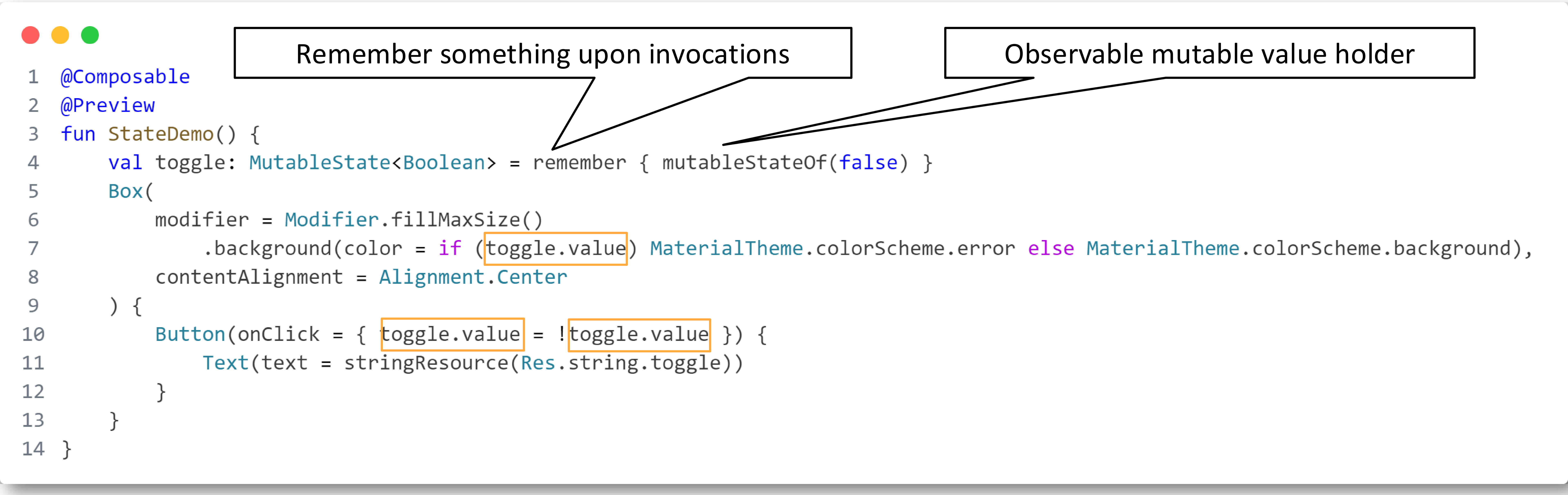
Main window is a composable, too

Changes to the UI are based on state

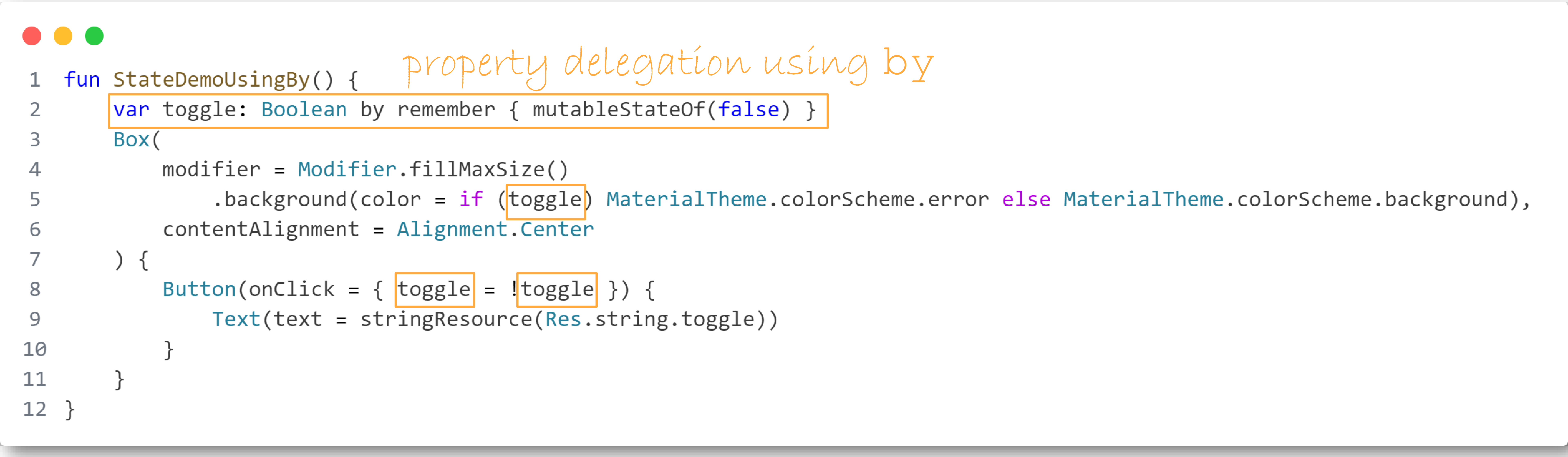
Wire up behavior

Experts

- Declarative UI frameworks don’t expose component trees
    - No references (pointers) to leaves or branches of the UI tree
    - No runtime exceptions due to outdated references
- The UI is declared based on **state**
- State changes trigger UI updates
- State relies on the Observer/Observable pattern


Remember something upon invocations
Observable mutable value holder

```kotlin
@Composable
@Preview
fun StateDemo() {
    val toggle: MutableState<Boolean> = remember { mutableStateOf(false) }
    Box(
        modifier = Modifier.fillMaxSize()
            .background(color = if (toggle.value) MaterialTheme.colorScheme.error else MaterialTheme.colorScheme.background),
        contentAlignment = Alignment.Center
    ) {
        Button(onClick = { toggle.value = !toggle.value }) {
            Text(text = stringResource(Res.string.toggle))
        }
    }
}
```



property delegation using by

```kotlin
fun StateDemoUsingBy() {
    var toggle: Boolean by remember { mutableStateOf(false) }
    Box(
        modifier = Modifier.fillMaxSize()
            .background(color = if (toggle) MaterialTheme.colorScheme.error else MaterialTheme.colorScheme.background),
        contentAlignment = Alignment.Center
    ) {
        Button(onClick = { toggle = !toggle }) {
            Text(text = stringResource(Res.string.toggle))
        }
    }
}
```

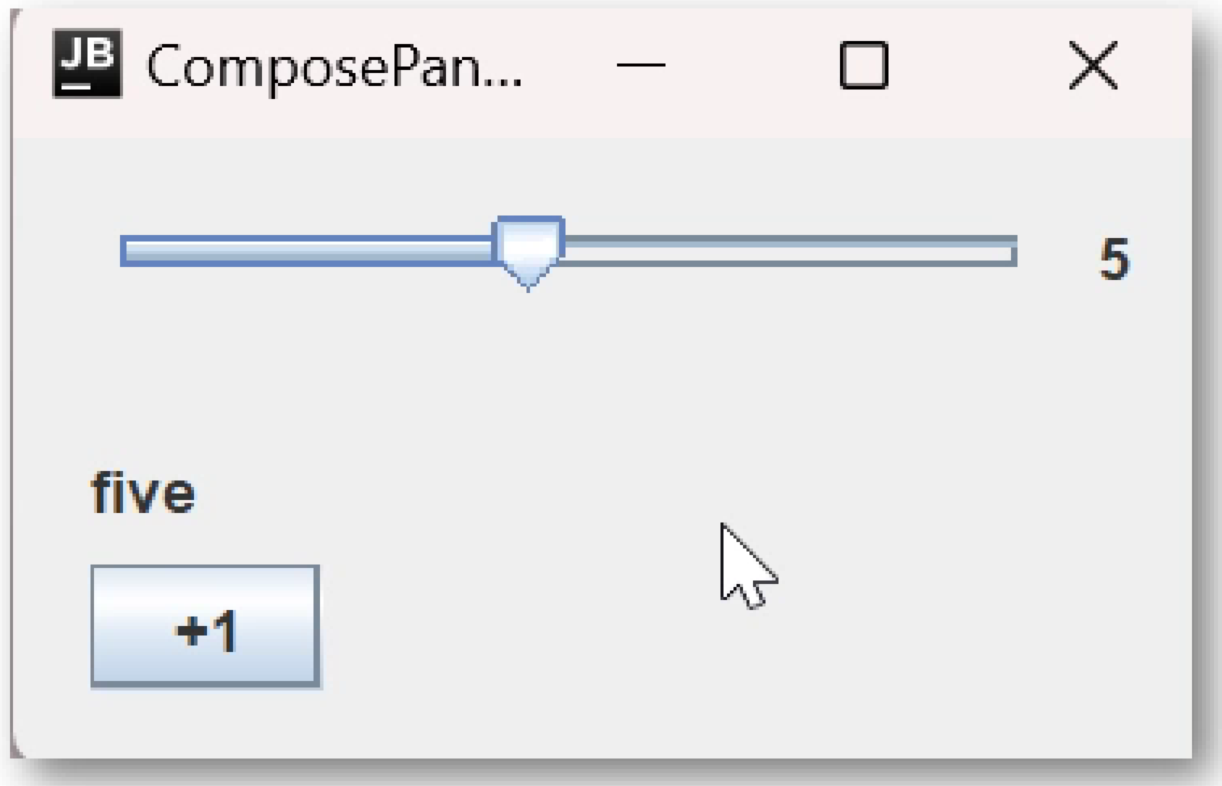
Experts

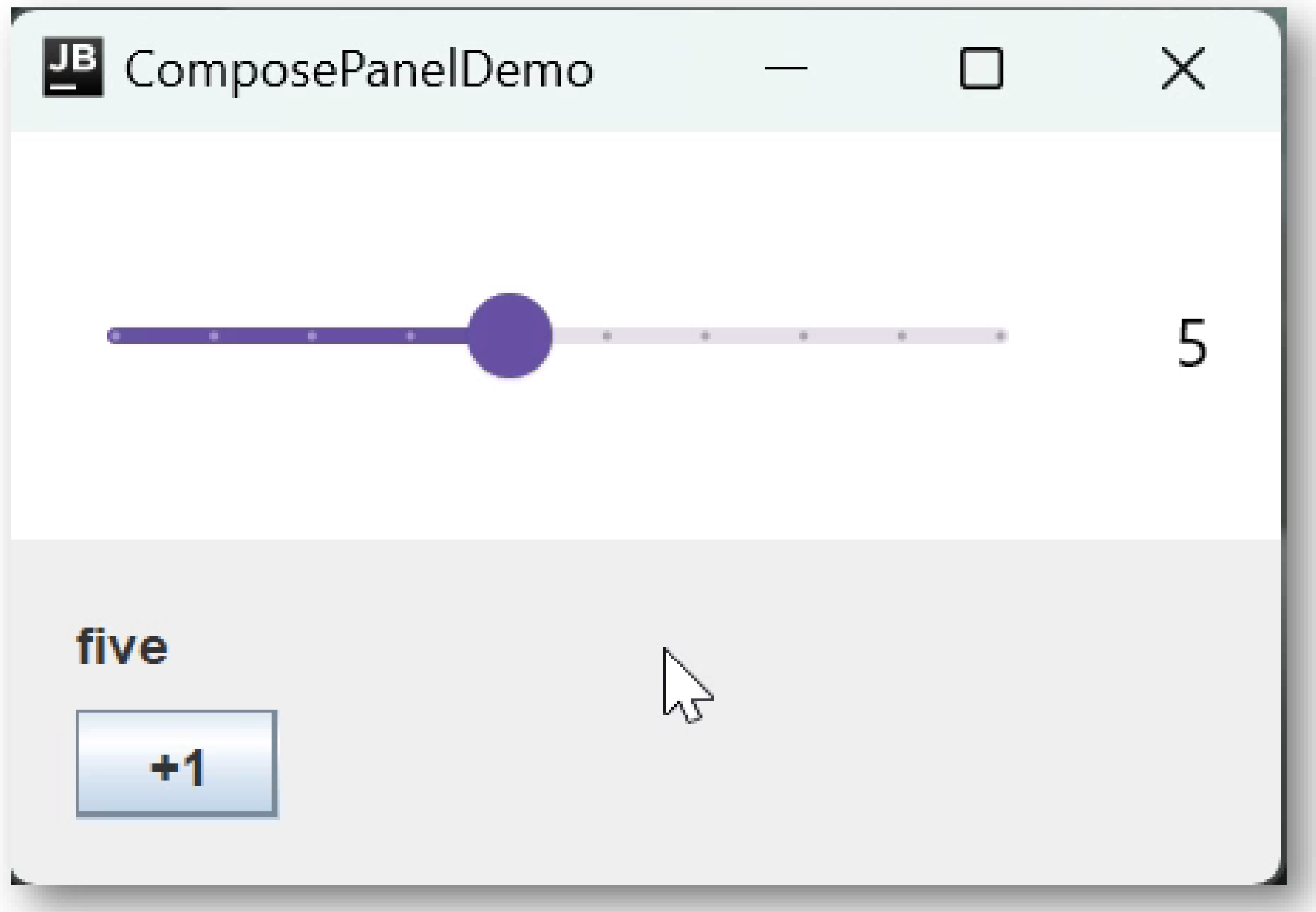
SnappMobile

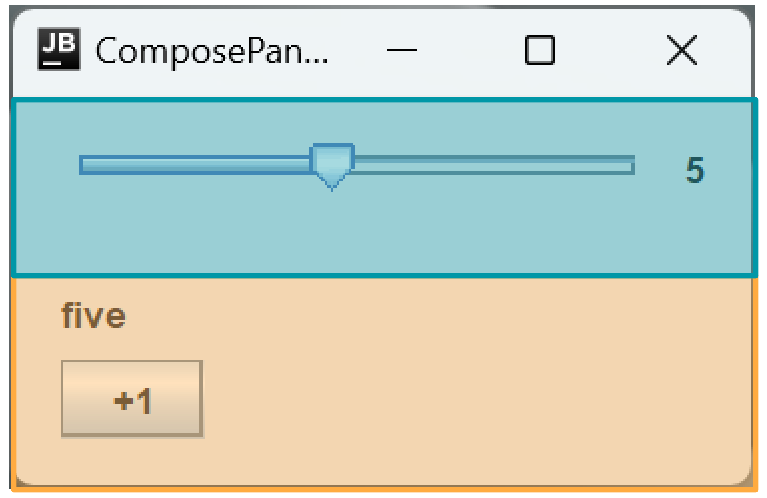
- Stateless composables
    - only depend on their input parameters (for the same set of parameters the same output is created)
    - are **composed** when they **enter the composition**
    - are **recomposed** when their input parameters change
- Stateful composables
    - hold state
    - depend on their input parameters and state associated with them
    - are composed when they enter the composition
    - are recomposed when their input parameters or the state associated with them change

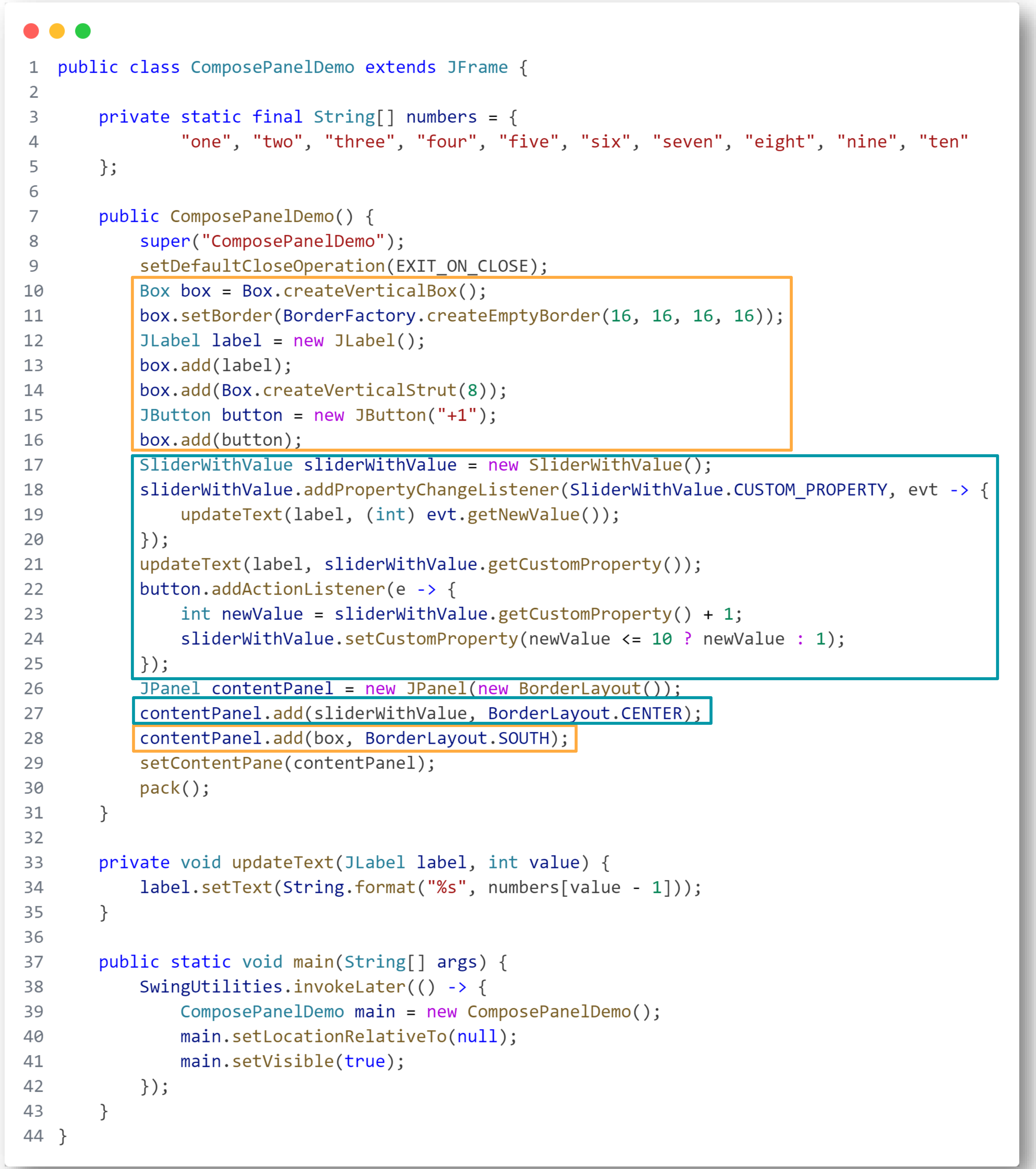
There’s more…

- Composition over inheritance
- Unidirectional data flow and state hoisting
- Animation
- Side effects
- …


ComposePan...
5
five
+1


ComposePanelDemo
5
five
+1



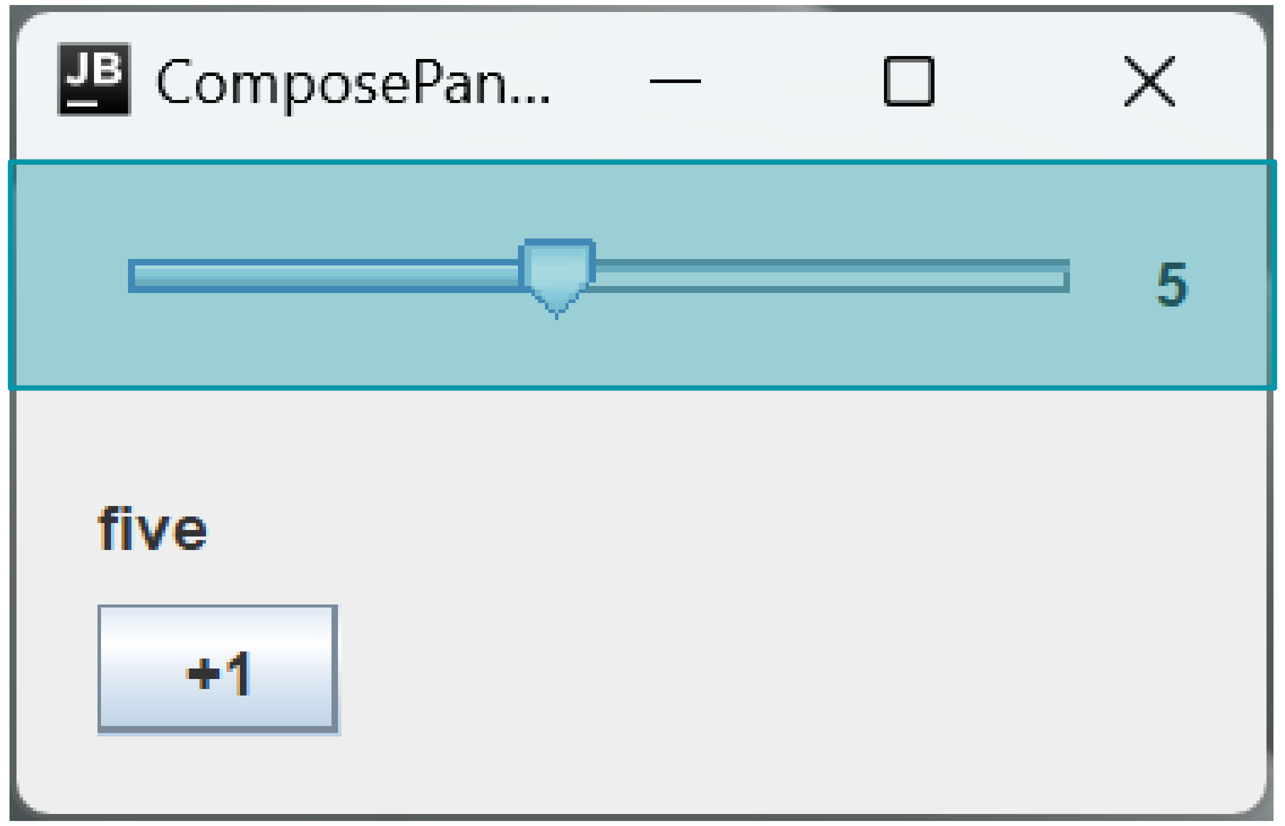


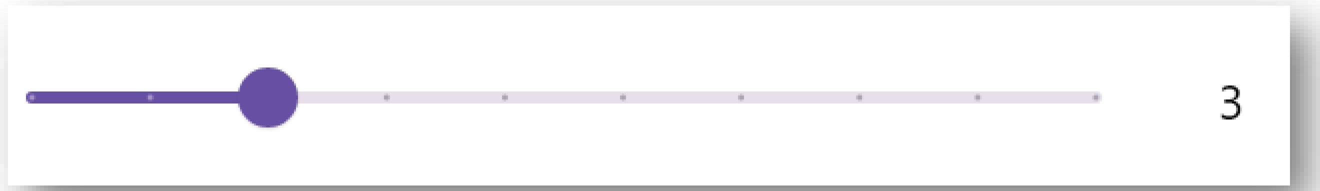
```java
public class ComposePanelDemo extends JFrame {

    private static final String[] numbers = {
            "one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "ten"
    };

    public ComposePanelDemo() {
        super("ComposePanelDemo");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        Box box = Box.createVerticalBox();
        box.setBorder(BorderFactory.createEmptyBorder(16, 16, 16, 16));
        JLabel label = new JLabel();
        box.add(label);
        box.add(Box.createVerticalStrut(8));
        JButton button = new JButton("+1");
        box.add(button);
        SliderWithValue sliderWithValue = new SliderWithValue();
        sliderWithValue.addPropertyChangeListener(SliderWithValue.CUSTOM_PROPERTY, evt -> {
            updateText(label, (int) evt.getNewValue());
        });
        updateText(label, sliderWithValue.getCustomProperty());
        button.addActionListener(e -> {
            int newValue = sliderWithValue.getCustomProperty() + 1;
            sliderWithValue.setCustomProperty(newValue <= 10 ? newValue : 1);
        });
        JPanel contentPanel = new JPanel(new BorderLayout());
        contentPanel.add(sliderWithValue, BorderLayout.CENTER);
        contentPanel.add(box, BorderLayout.SOUTH);
        setContentPane(contentPanel);
        pack();
    }

    private void updateText(JLabel label, int value) {
        label.setText(String.format("%s", numbers[value - 1]));
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            ComposePanelDemo main = new ComposePanelDemo();
            main.setLocationRelativeTo(null);
            main.setVisible(true);
        });
    }
}
```

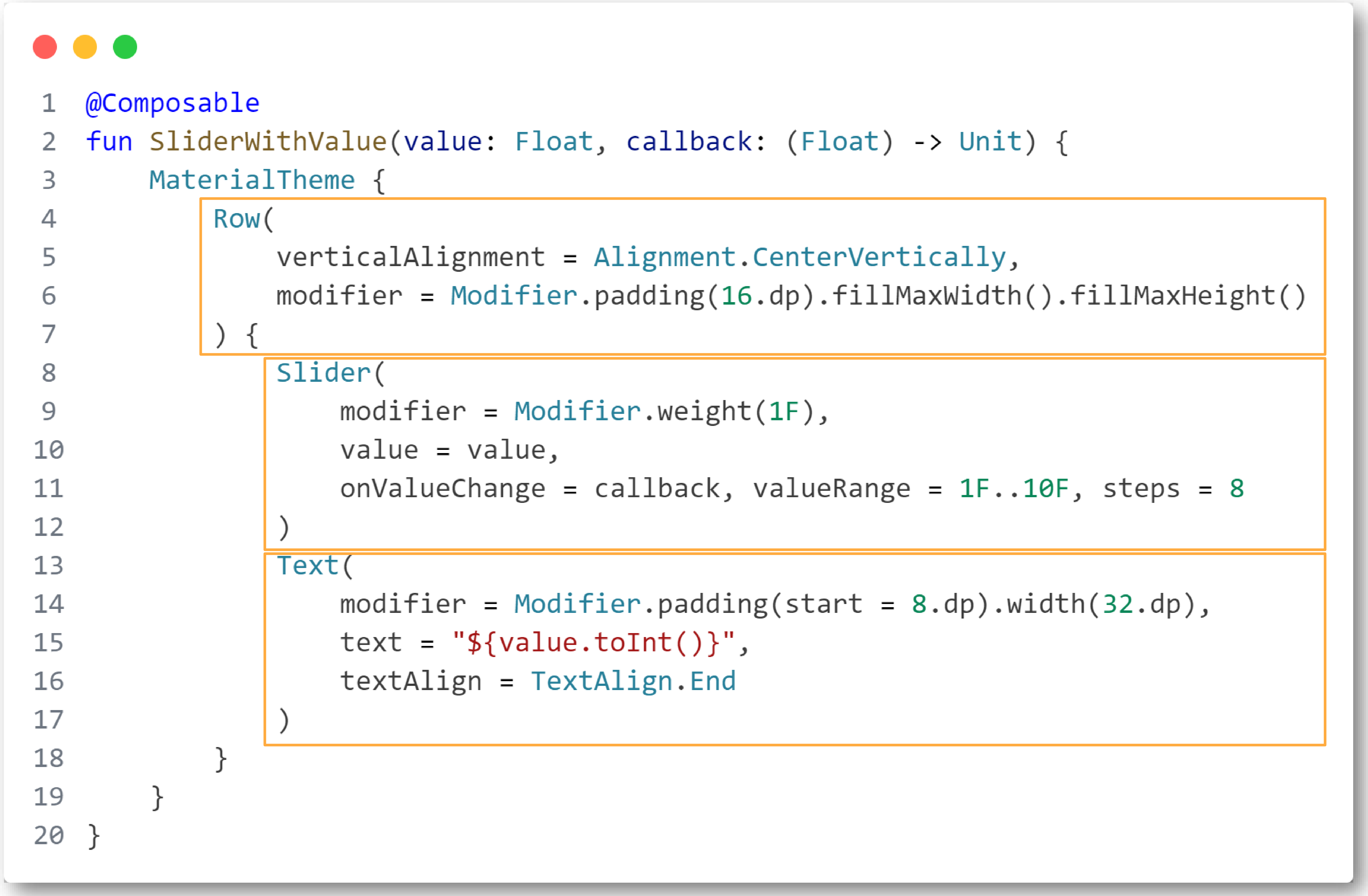
Experts

```java
public class SliderWithValue extends JPanel {

    public static final String CUSTOM_PROPERTY = "customProperty";
    private int customProperty = -1;

    public SliderWithValue() {
        super(new FlowLayout(FlowLayout.LEADING, 8, 8));
        setBorder(BorderFactory.createEmptyBorder(8, 8, 8, 8));
        setAlignmentY(TOP_ALIGNMENT);
        JSlider slider = new JSlider();
        JLabel label = new JLabel();
        slider.addChangeListener((event) -> setCustomProperty(slider.getModel().getValue()));
        slider.setMinimum(1);
        slider.setMaximum(10);
        addPropertyChangeListener(CUSTOM_PROPERTY, (event) -> {
            int newValue = (int) event.getNewValue();
            slider.setValue(newValue);
            label.setText(String.format("%d", newValue));
        });
        setCustomProperty((slider.getMaximum() - slider.getMinimum()) / 2 + slider.getMinimum());
        add(slider);
        add(label);
    }

    public int getCustomProperty() {
        return customProperty;
    }

    public void setCustomProperty(int newValue) {
        int oldValue = getCustomProperty();
        int _newValue = min(max(1, newValue), 10);
        customProperty = _newValue;
        firePropertyChange(CUSTOM_PROPERTY, oldValue, _newValue);
    }
}
```

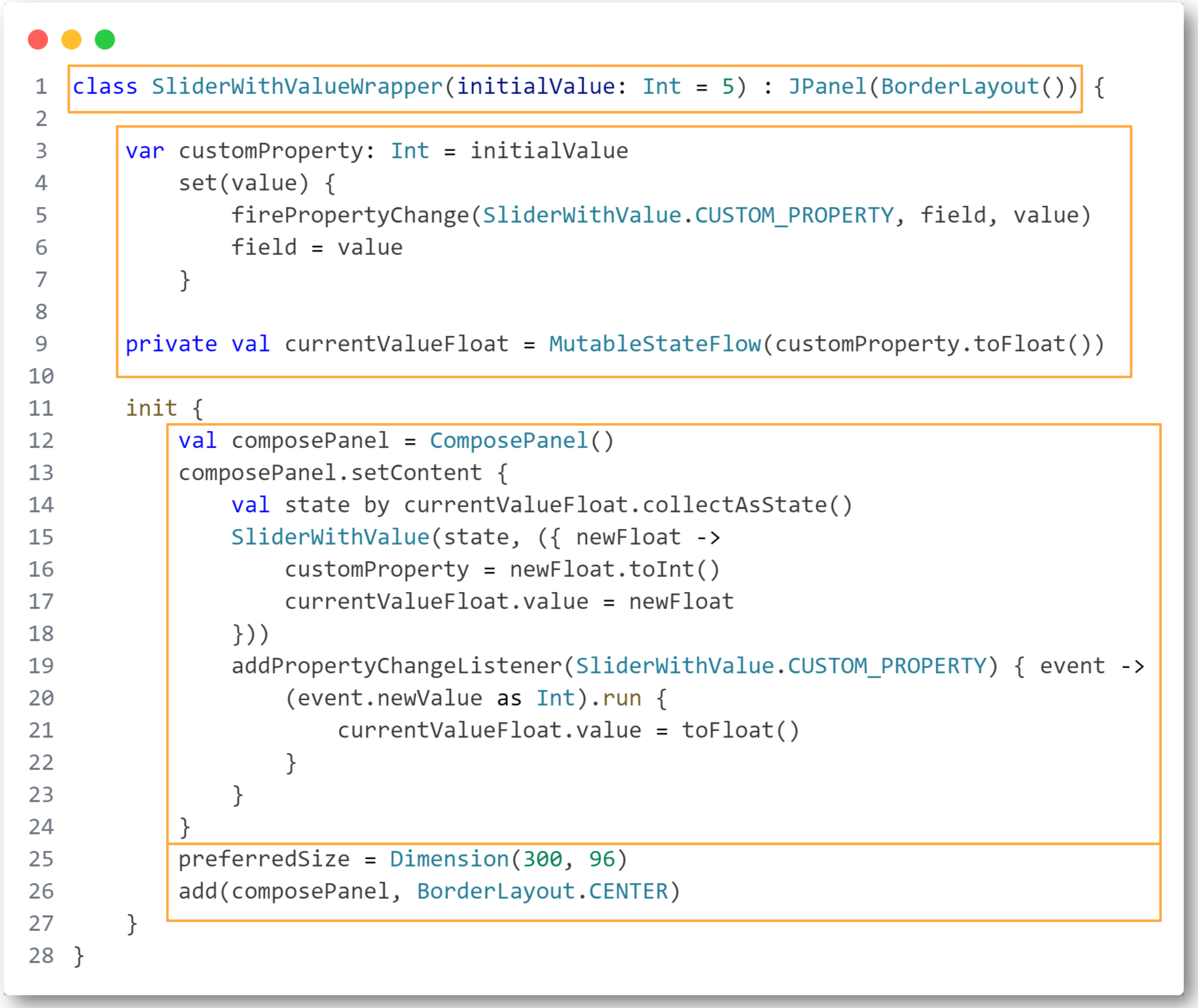
Java Beans specification


ComposePan...
5
five
+1


3

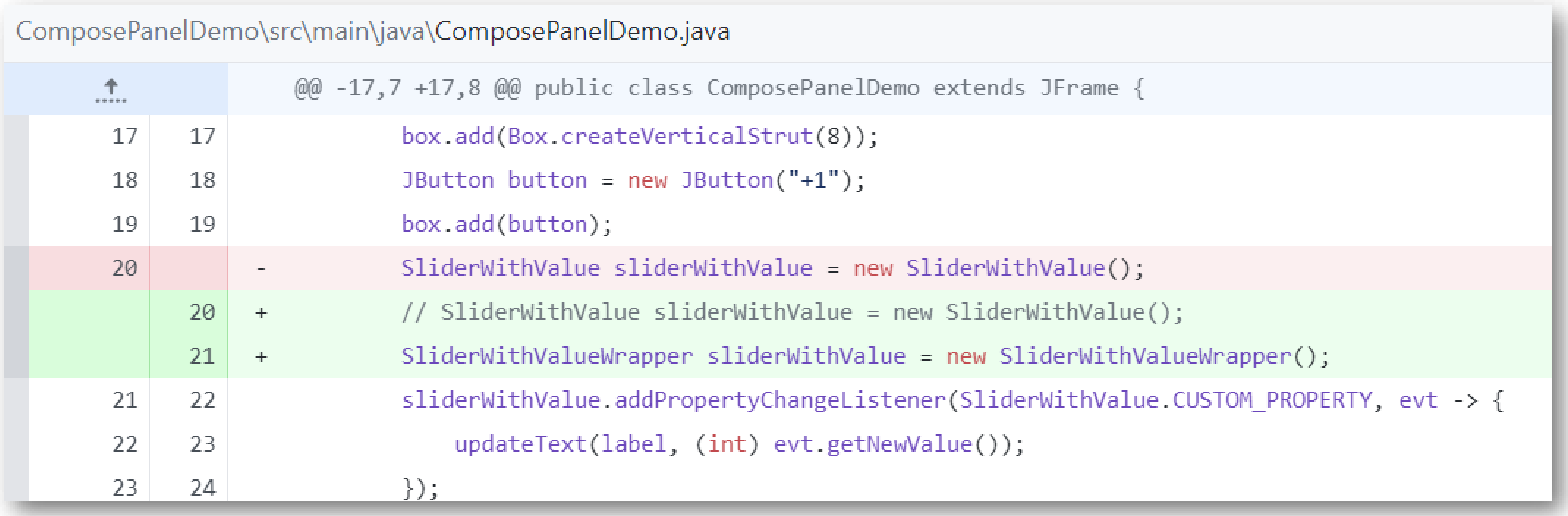


```kotlin
@Composable
fun SliderWithValue(value: Float, callback: (Float) -> Unit) {
    MaterialTheme {
        Row(
            verticalAlignment = Alignment.CenterVertically,
            modifier = Modifier.padding(16.dp).fillMaxWidth().fillMaxHeight()
        ) {
            Slider(
                modifier = Modifier.weight(1F),
                value = value,
                onValueChange = callback, valueRange = 1F..10F, steps = 8
            )
            Text(
                modifier = Modifier.padding(start = 8.dp).width(32.dp),
                text = "${value.toInt()}",
                textAlign = TextAlign.End
            )
        }
    }
}
```



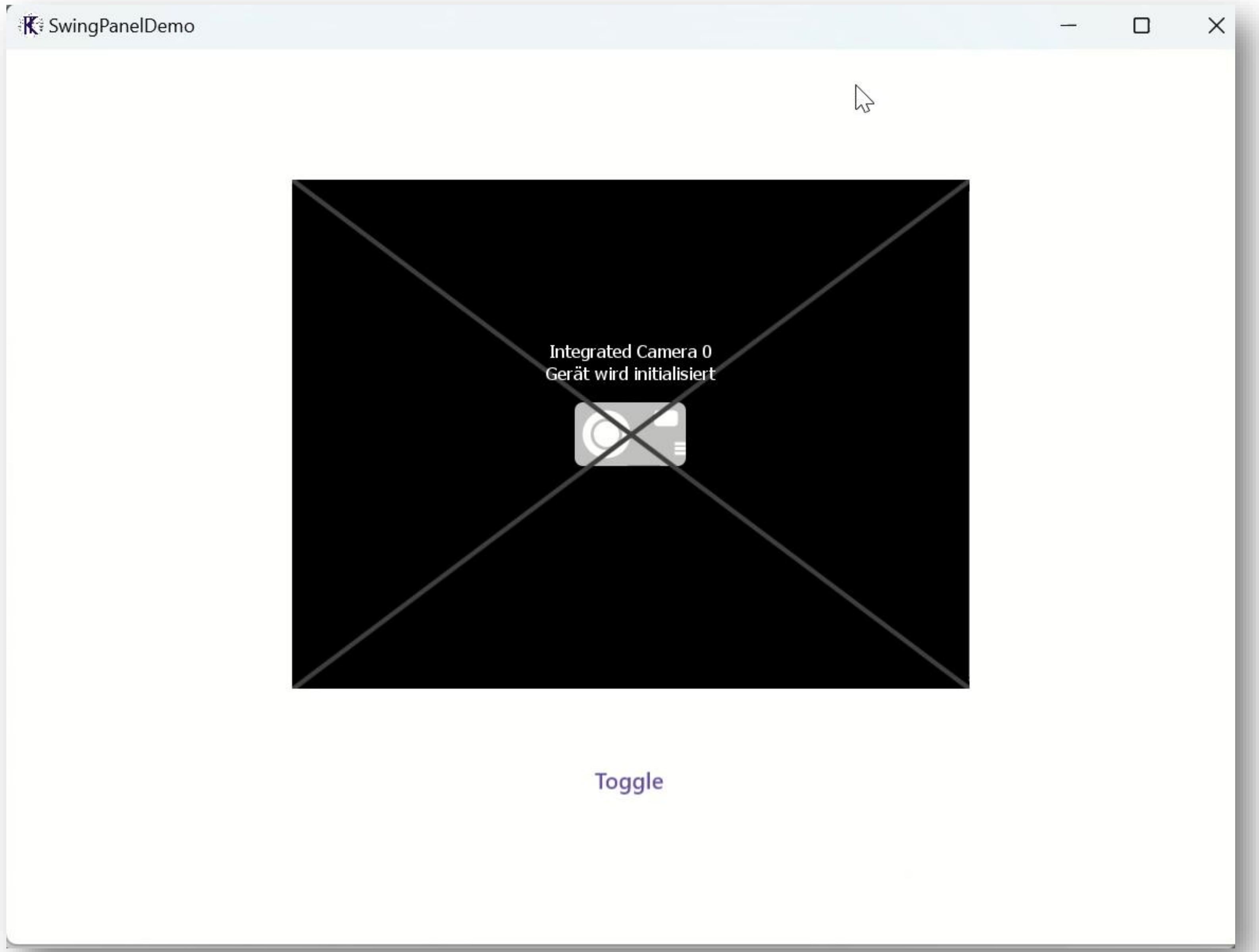
```
class SliderWithValueWrapper(initialValue: Int = 5) : JPanel(BorderLayout()) {

    var customProperty: Int = initialValue
        set(value) {
            firePropertyChange(SliderWithValue.CUSTOM_PROPERTY, field, value)
            field = value
        }

    private val currentValueFloat = MutableStateFlow(customProperty.toFloat())

    init {
        val composePanel = ComposePanel()
        composePanel.setContent {
            val state by currentValueFloat.collectAsState()
            SliderWithValue(state, ({ newFloat ->
                customProperty = newFloat.toInt()
                currentValueFloat.value = newFloat
            }))
            addPropertyChangeListener(SliderWithValue.CUSTOM_PROPERTY) { event ->
                (event.newValue as Int).run {
                    currentValueFloat.value = toFloat()
                }
            }
        }
        preferredSize = Dimension(300, 96)
        add(composePanel, BorderLayout.CENTER)
    }
}
```

Experts

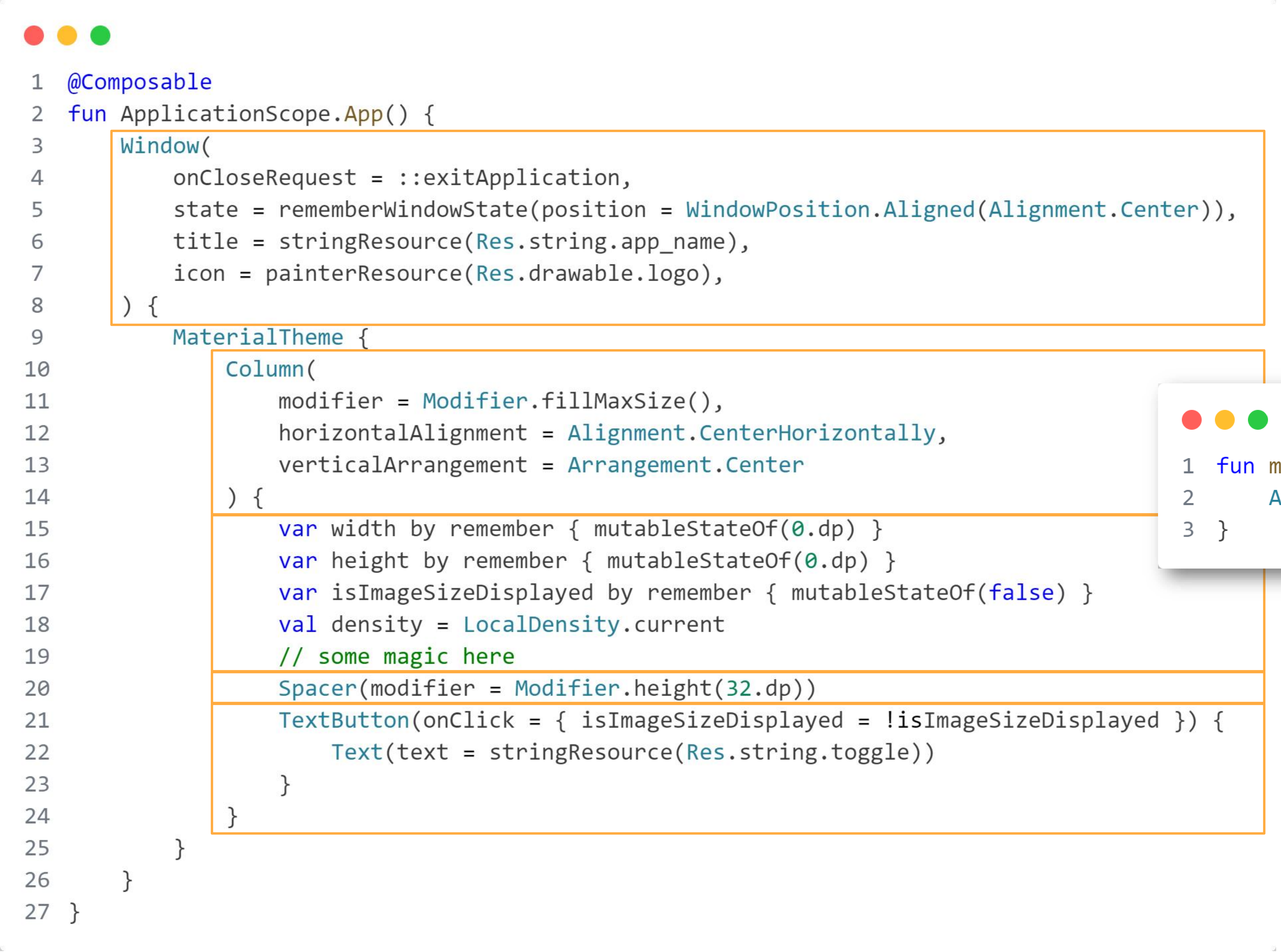

ComposePanelDemo\src\main\java\ComposePanelDemo.java

```diff
@@ -17,7 +17,8 @@ public class ComposePanelDemo extends JFrame {
             box.add(Box.createVerticalStrut(8));
             JButton button = new JButton("+1");
             box.add(button);
-            SliderWithValue sliderWithValue = new SliderWithValue();
+            // SliderWithValue sliderWithValue = new SliderWithValue();
+            SliderWithValueWrapper sliderWithValue = new SliderWithValueWrapper();
             sliderWithValue.addPropertyChangeListener(SliderWithValue.CUSTOM_PROPERTY, evt -> {
                 updateText(label, (int) evt.getNewValue());
             });
```

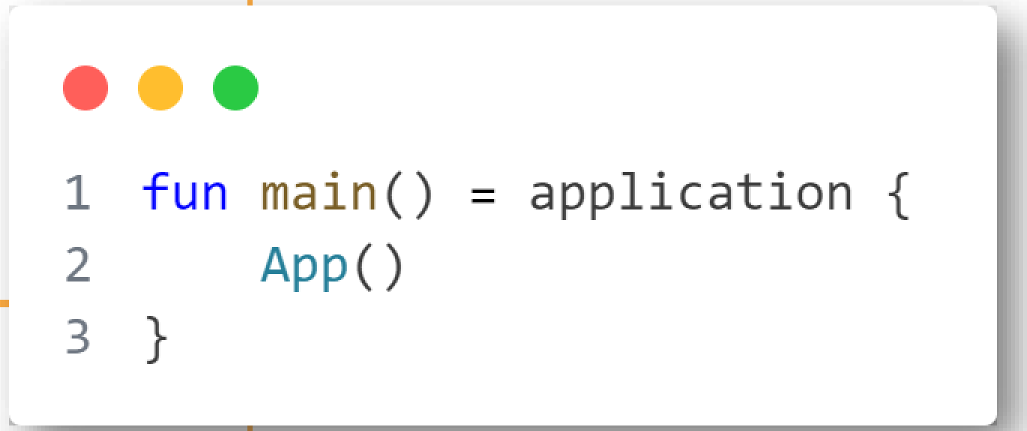
- `ComposePanel` allows replacing parts of a Swing UI with a Compose hierarchy
- Should be used for branches of the component tree, not on a component level


SwingPanelDemo

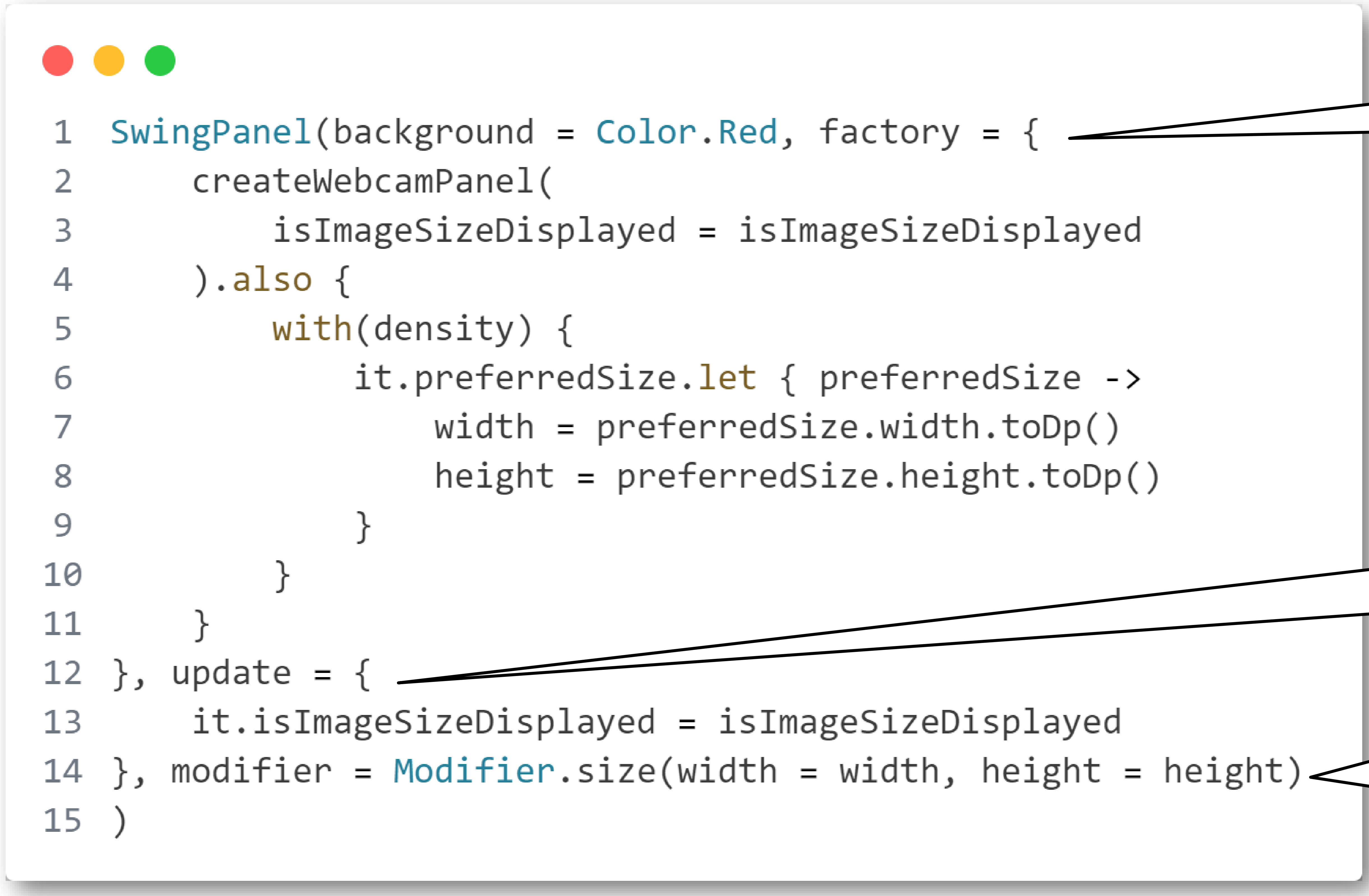
Integrated Camera 0
Gerät wird initialisiert



```kotlin
@Composable
fun ApplicationScope.App() {
    Window(
        onCloseRequest = ::exitApplication,
        state = rememberWindowState(position = WindowPosition.Aligned(Alignment.Center)),
        title = stringResource(Res.string.app_name),
        icon = painterResource(Res.drawable.logo),
    ) {
        MaterialTheme {
            Column(
                modifier = Modifier.fillMaxSize(),
                horizontalAlignment = Alignment.CenterHorizontally,
                verticalArrangement = Arrangement.Center
            ) {
                var width by remember { mutableStateOf(0.dp) }
                var height by remember { mutableStateOf(0.dp) }
                var isImageSizeDisplayed by remember { mutableStateOf(false) }
                val density = LocalDensity.current
                // some magic here
                Spacer(modifier = Modifier.height(32.dp))
                TextButton(onClick = { isImageSizeDisplayed = !isImageSizeDisplayed }) {
                    Text(text = stringResource(Res.string.toggle))
                }
            }
        }
    }
}
```



```kotlin
fun main() = application {
    App()
}
```



```
SwingPanel(background = Color.Red, factory = {
    createWebcamPanel(
        isImageSizeDisplayed = isImageSizeDisplayed
    ).also {
        with(density) {
            it.preferredSize.let { preferredSize ->
                width = preferredSize.width.toDp()
                height = preferredSize.height.toDp()
            }
        }
    }
}, update = {
    it.isImageSizeDisplayed = isImageSizeDisplayed
}, modifier = Modifier.size(width = width, height = height)
)
```

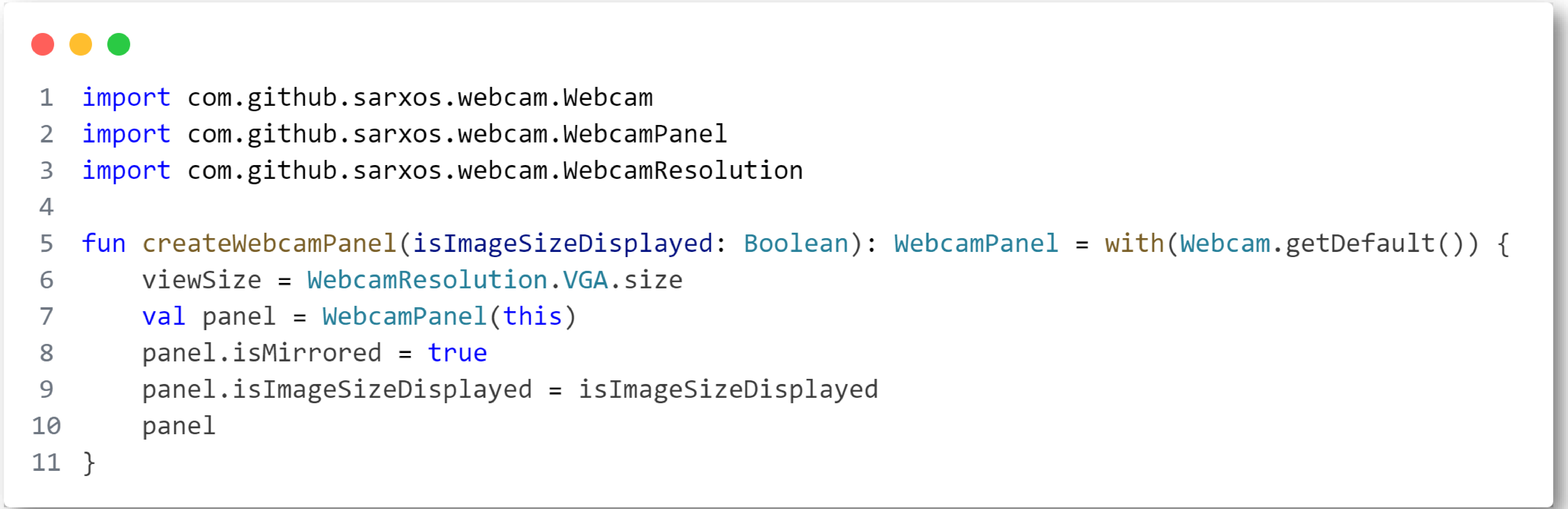
Returns `java.awt.Component` or derived classes

Called when related state changes (`isImageSizeDisplayed`)

Size of the composable is controlled by the size of the (Swing) component

Experts

SnappMobile



```kotlin
import com.github.sarxos.webcam.Webcam
import com.github.sarxos.webcam.WebcamPanel
import com.github.sarxos.webcam.WebcamResolution

fun createWebcamPanel(isImageSizeDisplayed: Boolean): WebcamPanel = with(Webcam.getDefault()) {
    viewSize = WebcamResolution.VGA.size
    val panel = WebcamPanel(this)
    panel.isMirrored = true
    panel.isImageSizeDisplayed = isImageSizeDisplayed
    panel
}
```

Webcam Capture API by Bartosz Firyn
https://github.com/sarxos/webcam-capture

- `SwingPanel` helps adding Swing components to a Compose hierarchy
- Should be used when…
  - composables providing a similar functionality are not available
  - the existing components hold considerable value (roi)

Things we saw

- Jetpack Compose isn’t deeply integrated into a platform
  - A root composable is placed inside a native container
  - It has its own (highly customizable look)
  - Two-way interop is provided on all supported platforms
- Compose Desktop benefits from a rock-solid foundation: the JVM
  - Everything possible on the JVM can be leveraged in a Compose Desktop app
  - To access truly native libraries, for now we need JNI

Things we couldn't cover

- Enjoying the multiplatform benefits
- Preconditions imposed by the framework
  - Heavy focus on Gradle and related plugins
  - Project structure
- Functionality and features of the UI elements provided by Jetpack Compose

Should I use it? (Checklist)

- Write a new app for the Desktop
- Update an existing AWT/Swing app
  - Evaluate converting the project to a structure defined by Compose Desktop or integrating the sources into a newly setup one
  - Evaluate general fitness/stability of the app
  - Evaluate the general code structure; is replacing individual parts possible or is everything intertwined?

Go

It depends

Thank You!

https://github.com/tkuenneth/compose-swing-interop

Experts

SnappMobile