CSCI 448 – Lab 13B
Friday, April 28, 2023
LAB IS DUE BY **Thursday, May 04, 2023 11:59 PM**!!

Now you want to test the UI following the same testing process.

Step 0 – Add a Dependency

There are two dependencies needed for UI testing, the first should already be included. Place these in your `app/build.gradle` file.

```
androidTestImplementation "androidx.compose.ui:ui-test-junit4:$compose_version"
debugImplementation "androidx.compose.ui:ui-test-manifest:$compose_version"
```

Step 1 – Create Some UI Tests

Create a test class for the `BeatBoxScreen` composable using the same Ctrl + Shift + T / Cmd + Shift + T method. In the popup, be sure JUnit5 is selected, check the box for `@Before`, and place into the `androidTest` folder.

Remove the `jupiter` import with `org.junit.Before`.

Part 1.I – Create the Rule

Create the ComposeTestRule. In the `setUp/@Before` method, set the content to be the `BeatBoxScreen()`. This will mirror our preview method somewhat.

Create a list of sounds and provide an empty lambda.

```
val sound = mutableListOf<Sound>()
for(i in 0..36)
  sounds.add( Sound( "${i}_test" )
BeatBoxScreen(
  sounds = sounds,
  onPlaySound = { }
)
```

Part 1.II – Create the Failing Tests

We're going to create six tests. The first should initially pass, the rest will fail.

Part 1.II.A – `buttonHasSoundNameAsLabel()`

Use the following finder/matcher/assertion to perform the test:

Finder: `onNode`
Matcher: has text "`1_test`"
Assertion: is displayed

This test should pass.

Part 1.II.B – `playbackSpeedLabelIsDisplayed()`

Use the following finder/matcher/assertion to perform the test:

Finder: `onNode`
Matcher: has test tag "`playbackSpeedLabel`"
Assertion: is displayed

This test will fail.

Part 1.II.C – `playbackSpeedSliderIsDisplayed()`

Use the following finder/matcher/assertion to perform the test:

Finder: `onNode`
Matcher: has test tag "`playbackSpeedSlider`"
Assertion: is displayed

This test will fail.

Part 1.II.D – `defaultPlaybackSpeedIs100()`

Use the following finder/matcher/assertion to perform the test:

Finder: `onNode`
Matcher: has text "`Playback Speed 100%`"
Assertion: is displayed

This test will fail.

Part 1.II.E – `playbackSliderMaxUpdatesText()`

Use the following finder/matcher/action/assertion to perform the test:

Finder: `onNode`
Matcher: has test tag `"playbackSpeedSlider"`
Action: perform touch input → `swipeRight(left, right + width)`

Finder: `onNode`
Matcher: has text `"Playback Speed 200%"`
Assertion: is displayed

This test will fail.

Part 1.II.F – `playbackSliderMinUpdatesText()`

Use the following finder/matcher/action/assertion to perform the test:

Finder: `onNode`
Matcher: has test tag `"playbackSpeedSlider"`
Action: perform touch input → `swipeLeft(right, left - width)`

Finder: `onNode`
Matcher: has text `"Playback Speed 5%"`
Assertion: is displayed

This test will fail.

Part 1.II – Add the UI Components

Let's get some of the tests to pass. Our `BeatBoxScreen()` compose tree currently looks like:

- `LazyColumn`

We need to wrap that in a Column to be able to add some components below the scrollable list. Our final tree will look like the following with attributes:

- **`Column – fillMaxSize()`**
    - **`Box – fillMaxWidth() and weight( 0.9f )`**
        - `LazyColumn`
    - **`Box – fillMaxWidth() and weight( 0.1f )`**
        - **`Column`**
            - **`Text – text = "Playback Speed 100%", test tag = playbackSpeedLabel`**
            - **`Slider – value = 1.0f, valueRange = 0.05f..2.0f, onValueChange = { }, test tag = playbackSpeedSlider`**

Once those are in place, Tests 1.II.B, C, and D should be passing.

Part 1.III – Connect the `Slider` and `Text`

We'll need to create a `playbackSpeedState State` object that is set initially to `1.0f`.

The `Text` we'll replace the `"100%"` with the <u>integer</u> value of the `playbackSpeedState value`.

The `Slider value` will then be the `playbackSpeedState value`.

The `Slider onValueChange` will then set the `playbackSpeedState value`.

Once the state is in place, Tests 1.II.E and F should be passing.

Deploy your app, move the slider, and verify the text label is changing to reflect the slider position.

Step XC – Complete the Challenge

For the last piece of lab extra credit, have the `Slider` actually change the playback speed of the sounds.

Looking at `BeatBox::play(Sound)`, the last argument to the `soundPool.play()` method is the playback rate. Have this argument be a variable connected to the playback speed `Slider`.

Step 2 – Submission

Submit a video with the following features for the final sign off:

- Scroll through list of sounds
- Slide the Slider to have Playback Speed text update

If completing the extra credit, then play a sound at 100% speed, then at 175% speed, and then at 25% speed (or as close as you can get to each of those percentages).

Please name this video file `<your_user_name>_L13.` For instance, my submission would be `jpaone_L13.webm`.

LAB IS DUE BY **Thursday, May 04, 2023 11:59 PM**!!