BCS 371 Lab – Kotlin Coroutines and GUI

Overview

Create an app that uses a Kotlin coroutine to perform some work and show updates in the GUI.

Create a project

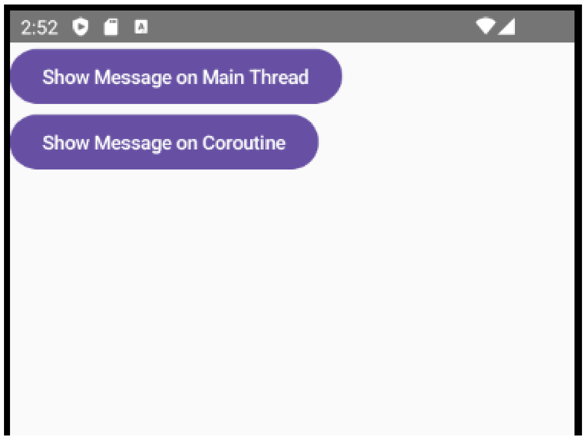
Create a new Android application in Android Studio. Choose the **Empty Activity** type to create an empty activity that uses Jetpack Compose.

Setup the Main Screen

Create a Kotlin file name MainScreen.kt. Add a composable function named MainScreen. Here is the function header:

```
@Composable
fun MainScreen(modifier: Modifier)
```

It should look like the following:



Button Event Handling

Show messages on different threads using coroutines. You can get a thread’s name using the following code: Thread.currentThread().name

- Show Message on Main Thread – This button’s event handler should show the current thread’s name in the Logcat window.
- Show Message on Coroutine – This button’s event handler should first create a coroutine. Use a CoroutineScope with Dispatcher.IO when creating the coroutine. Dispatchers.IO will ensure that the coroutine is executed on a thread other than the main thread. Inside the coroutine, print the current thread’s name in the Logcat window.

Run the app. The thread names that appear when each button is pressed should be different. For example:

```
I love Android, Thread name: main
I love Android, Thread name: DefaultDispatcher-worker-1
```

Note: If you press the Show Message on Coroutine button more than once you will likely see different worker thread numbers in the output. For example:

```
DefaultDispatcher-worker-1
DefaultDispatcher-worker-3
DefaultDispatcher-worker-3
DefaultDispatcher-worker-1
```

Default Launcher Context

Update the Show Message on Coroutine button event handler so that the coroutine uses Dispatchers.Main (instead of Dispatchers.IO).

Run the app. It should show the following output when pressing each button once:

```
I love Android, Thread name: main
I love Android, Thread name: main
```

When you are finished trying it out, put the Dispatchers.IO parameter back in for that launch statement.

UI Responded Message

Update the app to demonstrate how the UI hangs or does not hang depending on if work is being done on the main thread or not.

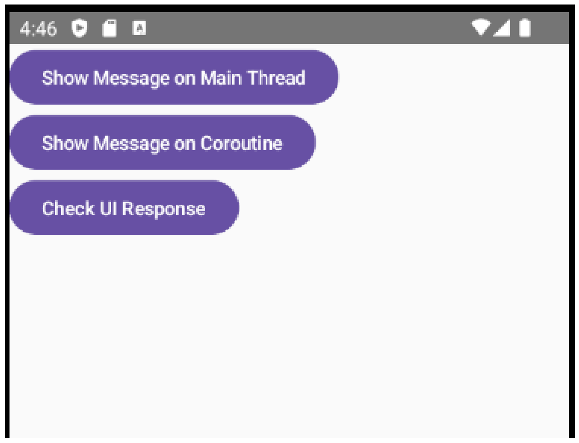
- Add a button to the layout with the text “Check UI Response”.
- When the Check UI Response button is pressed it should display a toast with the message “UI Responded”. Run this code on the main thread (no coroutine).
- Add code to the Show Message on Main Thread button event handler. Make the thread sleep for 3000 milliseconds (3 seconds). Use Thread.sleep(3000).
- Add code to the Show Message on Coroutine button event handler. Make the thread the coroutine is running on sleep for 3000 milliseconds (3 seconds). Just put Thread.sleep(3000) inside the coroutine code.

Run the app. Do the following to test it:

- Press the Show Message on Main Thread button.
- Immediately press the Check UI Response button. This should have no immediate effect because the app’s GUI will be unresponsive since it is sleeping for 3 seconds. Be sure to wait 3 seconds before moving on to the next step. It will show the toast after 3 seconds passes.

- Press the Show Message on Coroutine button.
- Immediately press the Check UI Response. The app should respond and display the toast message immediately. It displays the toast immediately because the thread the coroutine was running is sleeping instead of the main thread. The main thread is not sleeping so it is free to respond to user interactions.

Here is a screenshot of the new screen:


4:46
Show Message on Main Thread
Show Message on Coroutine
Check UI Response