Mobile Dev Week 2 Lab

Daniel Coady (102084174)

16/08/2019

Orientating Ourselves

Task 1

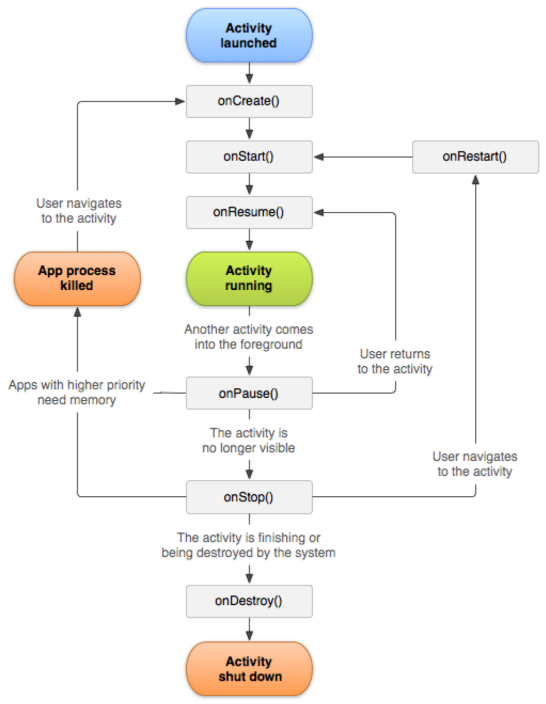

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure 1: The activity lifecycle of an app on Android

Android applications have a variety of functions that can be called when a certain event happens on the device such as when the app is resumed or closed. This is so that the application can handle these events appropriately without losing things like user data or progress. It's important to consider this when developing, as seen in the application

"Static Clock App" where the time display is changed each time the orientation of the phone is changed.

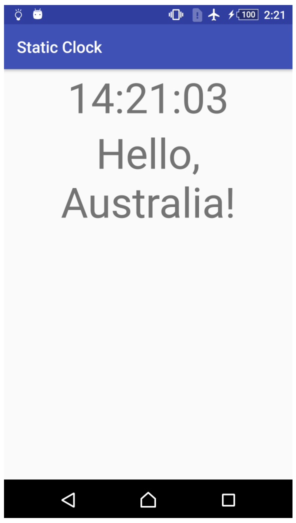


Figure 2: The application in portrait

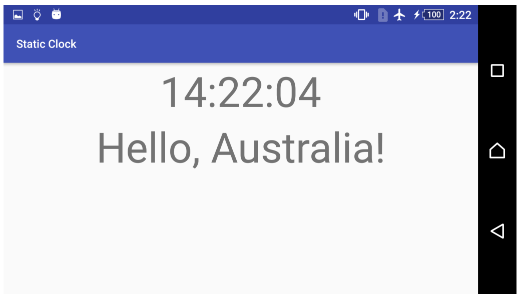
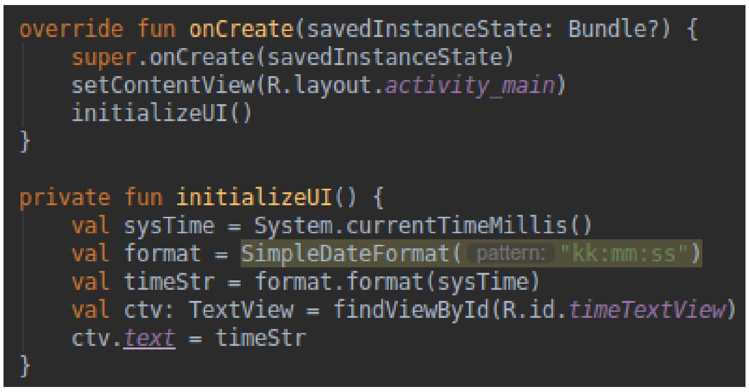


Figure 3: The application in landscape

This is due to what we call the "activity lifecycle" which contains some of the aforementioned functions that are called when a certain event happens. In this case, the events are cases where a significant state of the app changes such as when it is first launched or when it is put into the background. In this case, we can actually see in the code itself when the time display is handled: within the function onCreate! It's obvious then that each time the orientation changes in the application, this function is called again which is what causes the time display to update.



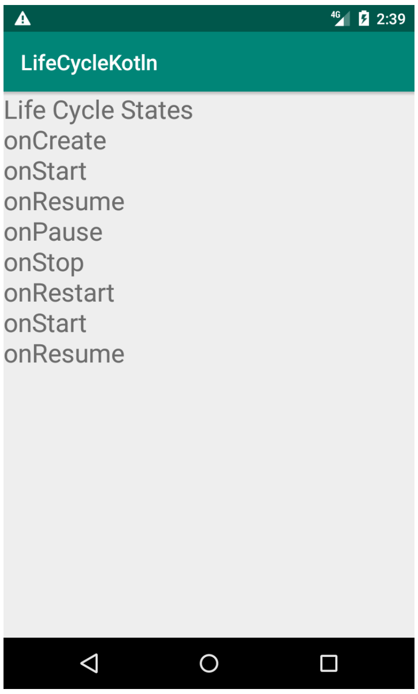
```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    initializeUI()
}

private fun initializeUI() {
    val sysTime = System.currentTimeMillis()
    val format = SimpleDateFormat( pattern: "kk:mm:ss")
    val timeStr = format.format(sysTime)
    val ctv: TextView = findViewById(R.id.timeTextView)
    ctv.text = timeStr
}
```

Figure 4: The code which handles displaying the time

onResume()	When the application enters the resumed state, it brings the application to the foreground and notifies all relevant sections of code that this has happened. this generally occurs after the application is brought back from the background (for example, a user has switched back to the application).
onPause()	Paused is the state that the application will enter if put into the background while running. Just like the resumed state, this will notify all relevant parts of the code that this has occured.
onStop()	Finally, the stopped state is what the application will enter if the user fully exits the application. This is in many ways similar to the paused state, but with the key difference of an application will only enter this state if it is well and truly about to end and it will not continue past that.

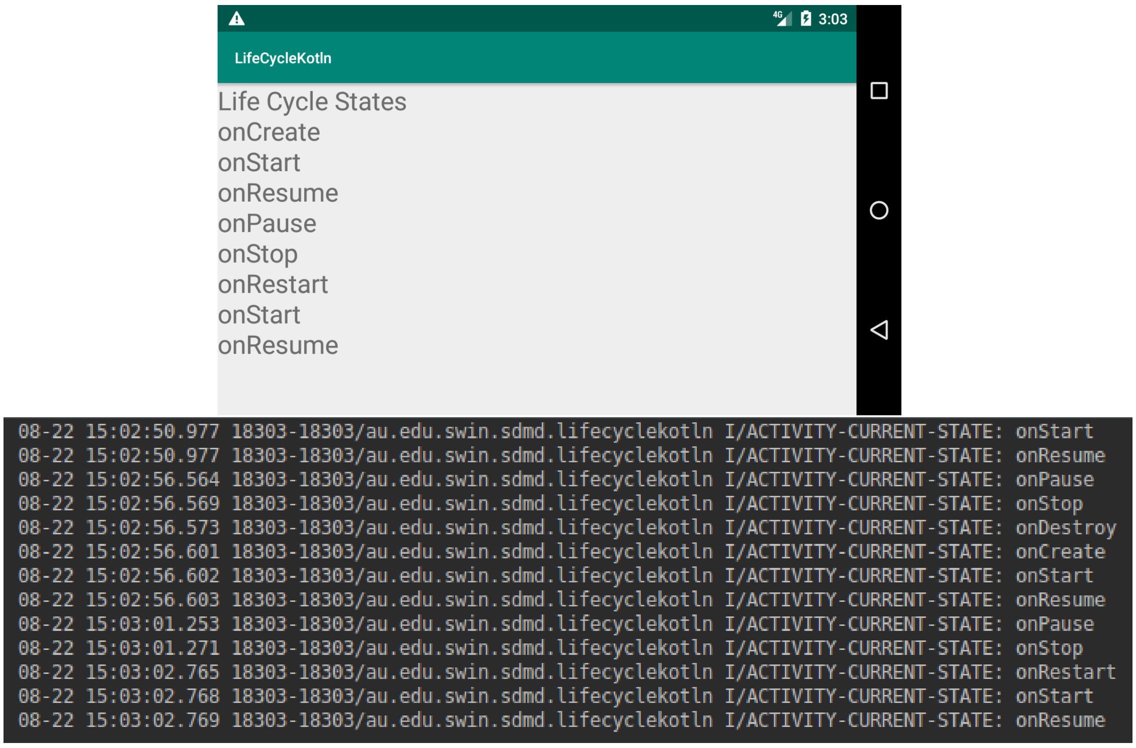
Task 2



Life Cycle States
onCreate
onStart
onResume
onPause
onStop
onRestart
onStart
onResume

Figure 5: Triggering the on restart event

To trigger the onRestart event is quite a trivial task if you understand how the activity lifecycle works. Looking at how it flows, you can see that to get to the activity initially running we have to go through onCreate, onStart, and onResume. From here, the only path we then have to get to onRestart is through onPause and then onStop. To go through these two events we can simply return to the home screen of the phone, leaving the activity in the onStop state. There's just one more thing to do from here, and that's to bring up the application again from the background either by opening it in the multitasking display or by tapping on the application again. Once we do this we will go through the onRestart event, which then leads onto the onStart and onResume events which is clearly reflected in the screenshot of this application running.



```
Life Cycle States
onCreate
onStart
onResume
onPause
onStop
onRestart
onStart
onResume
```

```
08-22 15:02:50.977 18303-18303/au.edu.swin.sdmd.lifecyclekotln I/ACTIVITY-CURRENT-STATE: onStart
08-22 15:02:50.977 18303-18303/au.edu.swin.sdmd.lifecyclekotln I/ACTIVITY-CURRENT-STATE: onResume
08-22 15:02:56.564 18303-18303/au.edu.swin.sdmd.lifecyclekotln I/ACTIVITY-CURRENT-STATE: onPause
08-22 15:02:56.569 18303-18303/au.edu.swin.sdmd.lifecyclekotln I/ACTIVITY-CURRENT-STATE: onStop
08-22 15:02:56.573 18303-18303/au.edu.swin.sdmd.lifecyclekotln I/ACTIVITY-CURRENT-STATE: onDestroy
08-22 15:02:56.601 18303-18303/au.edu.swin.sdmd.lifecyclekotln I/ACTIVITY-CURRENT-STATE: onCreate
08-22 15:02:56.602 18303-18303/au.edu.swin.sdmd.lifecyclekotln I/ACTIVITY-CURRENT-STATE: onStart
08-22 15:02:56.603 18303-18303/au.edu.swin.sdmd.lifecyclekotln I/ACTIVITY-CURRENT-STATE: onResume
08-22 15:03:01.253 18303-18303/au.edu.swin.sdmd.lifecyclekotln I/ACTIVITY-CURRENT-STATE: onPause
08-22 15:03:01.271 18303-18303/au.edu.swin.sdmd.lifecyclekotln I/ACTIVITY-CURRENT-STATE: onStop
08-22 15:03:02.765 18303-18303/au.edu.swin.sdmd.lifecyclekotln I/ACTIVITY-CURRENT-STATE: onRestart
08-22 15:03:02.768 18303-18303/au.edu.swin.sdmd.lifecyclekotln I/ACTIVITY-CURRENT-STATE: onStart
08-22 15:03:02.769 18303-18303/au.edu.swin.sdmd.lifecyclekotln I/ACTIVITY-CURRENT-STATE: onResume
```

Figure 6: Every event in the activity lifecycle being invoked

In this application we can see all of the events that form a part of the activity lifecycle being invoked. However, when we look at the log and the display in the app it looks different. If you look carefully, you can actually see that the application display doesn't show the onDestroy event when it's called, and what's more any events invoked before are no longer displayed. This is because when the onDestroy event is invoked the application is essentially "reset", in that it is taken back to the onCreate event again. This means that the display is also reset and we only see the events that have been invoked after onDestroy has happened.