5COSC023W - MOBILE APPLICATION DEVELOPMENT
Lecture 5: Activity Lifecyle and Configuration Changes

Dr Dimitris C. Dracopoulos

Sending Data from one Activity to Another

In the Sending Activity:

1. Create the Intent.
2. Set data or put extra data in the Intent.
3. Start the receiving (new activity) with `startActivity(intent)`.

In the Receiving Activity:

1. Get the Intent that created the Activity.
2. Retrieve the data or extras from the Intent.

Examples

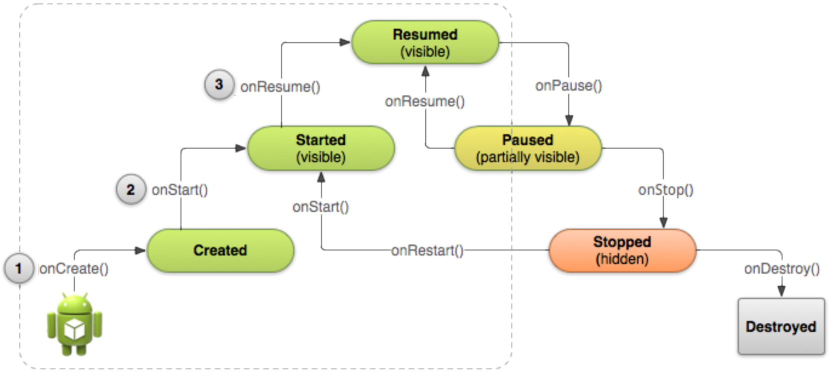
```
// Setting data and extras
intent.setData(Uri.parse("http://www.google.co.uk"));
intent.setData(Uri.parse("tel:02079115000"));
intent.putExtra("Score", 56345);

// Retrieving data and extras
Uri url = intent.getData();
int score = intent.getIntExtra("score", 0);
```

The Activity Lifecycle

- Created (not visible yet)
- Started (visible)
- Resume (visible)
- Paused(partially invisible)
- Stopped (hidden)
- Destroyed (gone from memory)

State changes are triggered by user action, configuration changes such as device rotation, or system action

The Activity Lifecycle (cont'ed)



When the Callbacks are Called?

- `onCreate(Bundle savedInstanceState)` — static initialization
  - `onStart()` — when Activity (screen) is becoming visible
  - `onRestart()` — called if Activity was stopped (calls onStart())
    - `onResume()` — start to interact with user
    - `onPause()` — about to resume PREVIOUS Activity
  - `onStop()` — no longer visible, but still exists and all state info preserved
- `onDestroy()` — final call before Android system destroys Activity

Implementing Callbacks

- Only onCreate() is required
- The other callbacks can be (optionally) overridden to change default behaviour

The `onCreate(Bundle savedInstanceState)` method

- Called when the Activity is first created
- Does all static setup: create views, bind data to lists, ...
- Only called once during an activity's lifetime
- Accepts a `Bundle` argument with Activity's previously saved state (saved with `onSaveInstanceState()`), if there was one
- Created state is always followed by `onStart()`

The `onResume` method

- Called when Activity will start interacting with user
- Activity has moved to top of the Activity stack
- The activity is both visible and interactive with the user
- This is Running state for the activity

The `onPause` method

- Called when system is about to replace the current activity with another
- The Activity is partly visible but non-interactive with the user
- Used to save data, stop animations and anything that consumes resources
- Implementations must be fast (not too much data saved) because the next Activity is not displayed until this method returns
- Followed by either `onResume()` if the Activity returns back to the front, or `onStop()` if it becomes invisible to the user

The `onStop()` method

- The activity is no more visible to the user
- Use to save data which take too long to save in `onPause`
- It is followed by either `onRestart()` if Activity is coming back to interact with user, or `onDestroy()` if Activity is going away

The `onDestroy()` method

- Final call before Activity is destroyed
- The user navigates to another activity or there is a *configuration change*
- The activity is finishing or the system destroys it to save space (you can distinguish between the 2 by calling `isFinishing`
- System may destroy Activity without calling this (by simply killing the process) , therefore use `onPause()` or `onStop()` to save data or state

Configuration Changes

Configuration changes invalidate the current layout or other resources in your activity when the user:

- Rotates the device
- Chooses different system language, so locale changes
- Enter multi-window mode
- Folding a foldable device with multiple displays
- and in other situations...

On configuration change the operating system:

1. Destroys the activity calling:
   1.1 onPause()
   1.2 onStop()
   1.3 onDestroy()
2. Starts the activity again calling:
   2.1 onCreate()
   2.2 onStart()
   2.3 onResume()

Activity Instance State

- State information is created while the Activity is running, such as a counter, user text, animation progression
- State is lost when device is rotated, language changes, back-button is pressed, or the system clears memory

What the Operating System Saves

The OS saves automatically:

- State of views with unique ID (`android:id`) such as text entered into an EditText
- The Intent that started the activity and data in its extras

$\longrightarrow$ The developer is responsible for saving other activity and user progress data

Saving instance state

Implement `onSaveInstanceState()` in the activity.

- Called by Android runtime when there is a possibility the Activity may be destroyed
- Saves data only for this instance of the Activity during the current session. If the application is restarted this cannot be used

$\longrightarrow$ `onSaveInstanceState` is not called when user explicitly closes the activity (e.g. presses the Back button) or when finish() is called. Use `onPause()` or `onStop()` instead

Implementing onSaveInstanceState()

```
override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState)

    outState.putInt("counter", counter)
}
```

Restoring Instance State

Two ways to retrieve the saved `Bundle` data:

- In `onCreate(Bundle mySavedState)`
- Implement callback `onRestoreInstanceState(Bundle mySavedState)` (this is called after `onStart()`

What happens when an Application Restarts?

- When the user stops and restarts a new app session, the Activity instance states are lost and the activities will revert to their default appearance
- If you need to save user data between app sessions, use
    1. Shared preferences
    2. or a Database

A Configuration Change Example

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setContent {
            GUI()
        }
    }
```

A Configuration Change Example (cont'd)

```
@Composable
fun GUI() {
    var name by rememberSaveable{ mutableStateOf("") }
    var counter by rememberSaveable { mutableStateOf(0)}
    var button2_click_counter by rememberSaveable{ mutableStateOf(0) }
    var background_colour = change_BackGroundColour(button2_click_counter)

    Column (horizontalAlignment = Alignment.CenterHorizontally,
            modifier = Modifier
                .fillMaxSize()
                .background(background_colour)) {
        TextField(value = name, onValueChange = {
            name = it
        })

        Text(text = "" + counter, fontSize = 30.sp)

        Row {
            Button(onClick = { ++counter }) {
                Text("Increment")
            }
            Button(onClick = {
                ++button2_click_counter
                background_colour = change_BackGroundColour(button2_click_counter)
            }) {
                Text("Change Background Colour")
            }
        }
    }
}
```

A Configuration Change Example (cont'd)

```
        // returns a new colour based on an even or odd number of clicks
        // the argument passed (counter) is the number of clicks
        fun change_BackGroundColour(counter: Int): Color {
            var bg_colour = Color.Gray
            if (counter % 2 == 0)  // even clicks change to green
                 bg_colour = Color.Green
            else  // odd clicks change to red
                 bg_colour = Color.Red

            return bg_colour
        }
    }
```

How to Create a List with Selectable Items

For displayng many entries in a list visually, use `LazyColumn` instead of `Column`.

- It is more efficient
- Otherwise (depending on how many items the app attempts to display), your application might freeze or crash.

An Example of a List with Selectable Items

```kotlin
package com.example.listwithselectableitemscomposable

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.runtime.Composable
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.selection.selectable
import androidx.compose.material3.Text
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            GUI()
        }
    }
}
```

An Example of a List with Selectable Items (cont'd)

```
@Composable
fun GUI() {
    var selected by remember{ mutableStateOf("") }

    // Create an empty list
    var myList = mutableListOf<String>()

    // populate the list with some items
    for (i in 1..100)
        myList.add("Item $i")

    Column(modifier = Modifier.fillMaxSize(),
           horizontalAlignment = Alignment.CenterHorizontally) {
        LazyColumn(Modifier.height(500.dp)) {
            for (i in myList)
                item {
                    Row(
                        modifier = Modifier
                            .fillMaxWidth()
                            .clickable(onClick = {
                                selected = i
                            })
                    )
                    {
                        Text(text = i, fontSize = 24.sp)
                    }
                }
        }

        Text(text = "This is what was just selected: $selected",
             modifier = Modifier.padding(top = 20.dp))
    }
}
```