5COSC023W - MOBILE APPLICATION DEVELOPMENT
Lecture 7: The Android DataStore

Dr Dimitris C. Dracopoulos

Saving Data in an Android Application

- Use `rememberSaveable` for configuration changes or system destroying and re-creating the activity.
  - *Can only be used within Composable functions.*
  - Can only be used with types that can be saved in a `Bundle`, such as primitives. Alternatively, your custom class needs to implement the `Parcelable` interface (e.g. use the `Parcelize` annotation) or use `listSaver/MapSaver` or implement a custom saver class extending the `Saver` class.
- Use `onSaveInstanceState(Bundle)` for configuration changes or system destroying and re-creating the activity.
  - *Can be used outside Composable functions (even if using Views instead of Jetpack Compose.)*
- Saving Key-Value Sets (small amounts)
  1. DataStore (new way)
  2. SharePreferences (old way)
- Saving in Files
- Saving in SQL databases (large amounts of structured data)

Preferences DataStore (Saving Key-Value Pairs)

1. Add the following dependency in the `build.gradle.kts` (Module:app) Gradle file of the module in your project:

```
dependencies {
    implementation("androidx.datastore:datastore-preferences:1.1.3")
}
```

2. Create a Preferences datastore at the top level of your Kotlin file and use it throughout the application:

```
val Context.datastore: DataStore<Preferences> by preferencesDataStore(name = "settings")
```

3. Define a key for <u>each</u> value that you would like to store in the datastore:

```
val totalKey = intPreferencesKey("total")
```

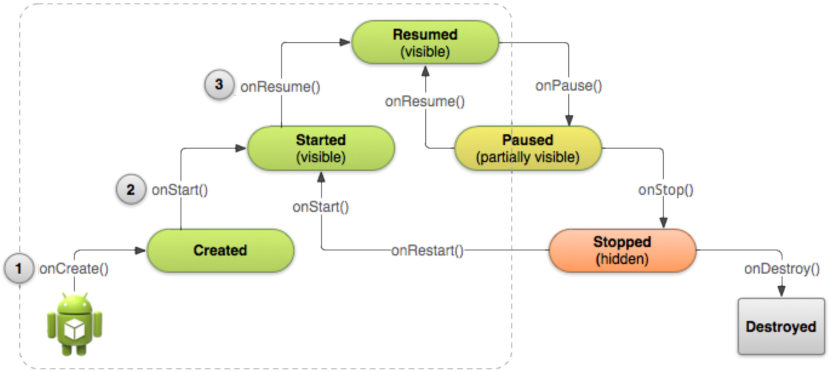
4. Read from the datastore:

```
/* read from the data store */
var preferences: Preferences
runBlocking {
    // the preferences need to be retrieved within a coroutine
    preferences = datastore.data.first()
}
// restore the value of total from the data store
total = preferences[totalKey] ?: 0  // assign 0 if the value in the datastore is null
```

5. Write to the datastore:

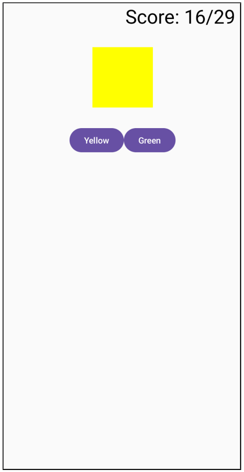
```
// save in the datastore the value for total
runBlocking {
    datastore.edit { settings ->
        settings[totalKey] = total  // total can be any value that will be associated with the key
    }
}
```

The Activity Lifecycle (cont'ed)



An Example Application for DataStore

An application which the user can guess the displayed colour. The score is persisted even the application is killed and restarted (even if the device reboots).



An Example Application for DataStore (cont'd)

```
package uk.ac.westminster.datastoreexample

import android.content.Context
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
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.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
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.graphics.Color
import androidx.compose.ui.graphics.RectangleShape
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
```

An Example Application for DataStore (cont'd)

```
import androidx.compose.ui.unit.sp
import androidx.datastore.core.DataStore
import androidx.datastore.preferences.core.Preferences
import androidx.datastore.preferences.core.edit
import androidx.datastore.preferences.core.intPreferencesKey
import androidx.datastore.preferences.preferencesDataStore
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.runBlocking
import kotlin.random.Random

// Create the data store
val Context.datastore: DataStore<Preferences> by preferencesDataStore(name = "settings")

var colours = listOf(Color.Black, Color.Red, Color.Green, Color.Blue, Color.Yellow, Color.White)
var colours_str = listOf("Black",  "Red",  "Green",  "Blue",  "Yellow",  "White")

var correct = 0  // number of correct answers
var total = 0   // number of colours presented to the user
```

An Example Application for DataStore (cont'd)

```
class MainActivity : ComponentActivity() {
    lateinit var preferences: Preferences
    lateinit var totalKey: Preferences.Key<Int>
    lateinit var correctKey: Preferences.Key<Int>

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // create a preferences key (totalKey) for storing an int value in a name called total
        totalKey = intPreferencesKey("total")
        // create a preferences key (correctKey) for storing an int value in a name called correct
        correctKey = intPreferencesKey("correct")

        /* read from the data store */
        runBlocking {
            // the preferences need to be retrieved within a coroutine
            preferences = datastore.data.first()
        }
        // restore the values of total and correct from the data store
        total = preferences[totalKey] ?: 0  // assign 0 if the value in the datastore is null
        correct = preferences[correctKey] ?: 0

        setContent {
            GUI()
        }
    }
```

An Example Application for DataStore (cont'd)

```
override fun onPause() {
        super.onPause()

        // save in the datastore the values for correct and total
        runBlocking {
            datastore.edit { settings ->
                settings[totalKey] = total
                settings[correctKey] = correct
            }
        }
    }
}
```

An Example Application for DataStore (cont'd)

```
@Composable
fun GUI() {
    var colour_chosen by remember{ mutableStateOf(Color.Yellow) }

    //val index = Random.nextInt(colours.size)
    //colour_chosen = colours[index]
    val index = colours.indexOf(colour_chosen)
    val colour_chosen_str = colours_str[index]

    // second colour to be displayed as one of the 2 buttons
    var second_colour_str = colours_str[Random.nextInt(colours.size)]
    while (second_colour_str == colour_chosen_str)
        second_colour_str = colours_str[Random.nextInt(colours.size)]

    // determine whether the correct colour will be displayed as the first
    // or second button - correct_button = 0 for the first button,
    // 1 for the second
    val correct_button = Random.nextInt(2)

    var first_button_label = colour_chosen_str
    var second_button_label = second_colour_str
    if (correct_button == 1) {
        first_button_label = second_colour_str
        second_button_label = colour_chosen_str
    }
```

An Example Application for DataStore (cont'd)

```
Column (
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
    ) {
        Text("Score: $correct/$total",
            fontSize = 32.sp,
            modifier = Modifier
                .padding(bottom = 30.dp, end = 10.dp)
                .fillMaxWidth(),
            textAlign = TextAlign.End)
        Button(
            modifier = Modifier.size(height = 100.dp, width = 100.dp),
            onClick = {},
            shape = RectangleShape,
            colors = ButtonDefaults.buttonColors(
                                 containerColor = colour_chosen)) {
        }
```

An Example Application for DataStore (cont'd)

```
            Row (
                modifier = Modifier.padding(top = 30.dp)
            ) {
                Button(onClick = {
                    ++total
                    if (correct_button == 0)
                        ++correct

                    colour_chosen = nextGame(colour_chosen)
                }) {
                    Text(first_button_label)
                }

                Button(onClick = {
                    ++total
                    if (correct_button == 1)
                        ++correct

                    colour_chosen = nextGame(colour_chosen)
                }) {
                    Text(second_button_label)
                }
            }
        }
    }
```

An Example Application for DataStore (cont'd)

```
// choose a new colour to display and make sure it is different
//than the previous one
fun nextGame(previous_colour_chosen: Color): Color {
    var index = Random.nextInt(colours.size)
    var colour_chosen = colours[index]

    // choose a brand new colour if the same colour was produced
    while (previous_colour_chosen == colour_chosen) {
        index = Random.nextInt(colours.size)
        colour_chosen = colours[index]
    }

    return colour_chosen
}
```