5COSC023W - MOBILE APPLICATION DEVELOPMENT
Lecture 5: Some Loose Ends, Dialogs and Creating Lists with Selectable Items

Dr Dimitris C. Dracopoulos

Nullable References - An Attempt to fix Tony Hoare's "Billion Dollar Mistake"

- By default, references cannot receive the value of `null`.

```
var s: String = null  // Compiler error!
```

- A question mark ? needs to be appended to make a variable nullable:

```
var s: String? = null // OK
```

- A nullable type cannot be dereferenced:

```
var s2: String? = "abc"
s2.length  // Compiler error!
```

- Use the safe call `?.` to attempt to dereference a nullable value:

```
var s2: String? = "abc"
s2?.length  // Will give back a value of null if s2 is null
```

- Alternatively, use the non-null assertion operator `!!`

```
var s3: String? = "abc"
s3!!  // if null throws a NullPointerException
```

Comparing Variables

- Use == (or `equals`) for structural comparison
- Use === to check if 2 references point to the same object

For primitive types such as `Int`, === is the same as ==.

The When Expression

Similar to the switch in Java and other programming languages in the C family.

```
fun translate(word: String): String =
    when (word) {
        "Bonjour" -> "Good Morning"
        "Bonne Nuit" -> "Good Night"
        "Dobré Ráno" -> "Good Morning"
        "Dobrý Večer" -> "Good Evening"
        else -> "Unknown word"
    }

fun main() {
    var meaning = translate("Bonjour")
    println(meaning)
}
```

Access Specifiers

Similar usage to other programming languages supporting object oriented programming but with different meaning.
When used for members (properties, functions) of a class:

- `public`: available to everyone
- `private`: available to the class only
- `protected`: subclasses can access and override these.
- `internal`: access only within the module where it is defined.

Default access is `public`.

- `public` and `private` can be used before the definition of a class, function or variable (property).

In such cases the meaning of `private` is access only within the same file.

Modules vs Packages

- Modules divide code at a higher level than packages.
- A library is often a single module consisting of multiple packages.
- The way a project is divided into modules, depends on the build system (e.g. gradle or maven).

The Manifest file

Do not forget to modify the `AndroidManifest.xml` file for your project by adding to it any new activity. If you create the activity using the Android Studio menu (create new activity→Gallery→Empty Activity) it will be automatically added to the manifest.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <application
        ...tools:targetApi="33">
        <activity
            android:name=".ContactDetails"
            android:exported="false"
            android:label="@string/title_activity_contact_details"
            android:theme="@style/Theme.TheLostDogComposableApp" />
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.TheLostDogComposableApp">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Modifiers

Use to decorate or change the appearance or behaviour of Composables (UI elements).

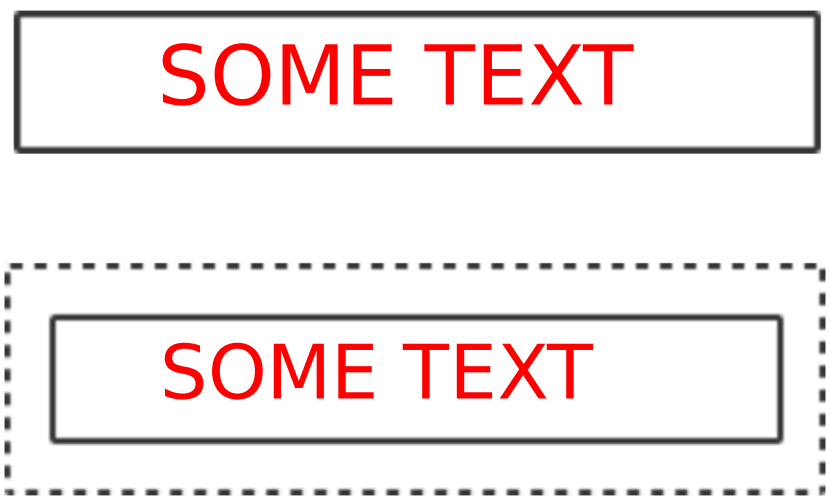
Can make UI elements:

- Clickable
- Scrollable
- Animated

Padding in Jetpack Compose

Unlike `Views` where padding and margins are different (margins is the extra space outside the border of the view, while padding is the extra space within the view, e.g. the space between the text displayed as part of the view and the border of the view).

- In Jetpack Compose padding defines extra space which is added to the composable container.
- The padding makes the Composable increasing its size.
- The content of the Composable (e.g. text or image) cannot use the extra space.



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);
```

A Full Example of 2 Communicating Activities

Activity 1:

```
package uk.ac.westminster.activitycommunicationcomposable

import android.content.Intent
import android.net.Uri
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.style.TextAlign

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

        setContent {
            GUI()
        }
    }
}
```

A Full Example of 2 Communicating Activities (cont'd)

```
@Composable
fun GUI() {
    val context = LocalContext.current

    Button(onClick = {
        val i = Intent(context, Activity2::class.java)
        i.setData(Uri.parse("tel:02079115000"))
        i.putExtra("Score", 5200)
        context.startActivity(i)
    }) {
        Text("Send data", textAlign = TextAlign.Center,
             modifier = Modifier.fillMaxWidth())
    }
}
```

A Full Example of 2 Communicating Activities (cont'd)

Activity 2:

```
package uk.ac.westminster.activitycommunicationcomposable
import android.net.Uri
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.sp

class Activity2 : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setContent {
            GUI2()
        }
    }

    // Note that GUI2 is defined inside the activity class here
    @Composable
    fun GUI2() {
        // extract the data sent by Activity1 from the intent
        var uri: Uri? = intent.data
        var score = intent.getIntExtra("Score", 0)

        // display the extracted data
        Text("Received: \n $uri \n Score: $score", fontSize = 24.sp,
              modifier = Modifier.fillMaxSize(), textAlign = TextAlign.Center)
    }
}
```

How to Create a Dialog (Popup Window)

A different ways but mostly the following can be used:

- `Dialog`
- `AlertDialog`

An Example of Creating a Dialog

```
package uk.ac.westminster.alertdialogcomposableexample

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Info
import androidx.compose.material3.AlertDialog
import androidx.compose.material3.Button
import androidx.compose.material3.Icon
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
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

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

An Example of Creating a Dialog (cont'd)

```
// An example of an alert dialog, pop up window
@Composable
fun GUI() {
    // is the pop up window open?
    var openDialog by remember { mutableStateOf(false) }


    Column {
        Text("My app is great")
        Button(onClick = { openDialog = true }) { Text("Open dialog") }
    }
    if (openDialog) {
        AlertDialog(
            icon = {
               Icon(Icons.Default.Info, contentDescription = "Example Icon")
            },
            title = {
                Text(text = "Title")
            },
            text = {
                Text(text = "dialogText")
            },
            // what happens when you click outside the dialog
            onDismissRequest = {
                openDialog = false
            },
            confirmButton = {
                TextButton(
                    onClick = {
                        openDialog = false
                    }
                ) {
                    Text("Confirm")
                }
            },
```

An Example of Creating a Dialog (cont'd)

```
            dismissButton = {
                TextButton(
                    onClick = {
                        openDialog = false
                    }
                ) {
                    Text("Dismiss")
                }
            }
        )
    }
}
```

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

```
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) {
        Text("Menu", fontSize = 36.sp, fontWeight = FontWeight.Bold)
        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))
    }
}
```