5COSC023W - MOBILE APPLICATION DEVELOPMENT
Lecture 10: Network Connectivity and Background Tasks

Dr Dimitris C. Dracopoulos

Steps to connect to the Internet

1. Add permissions to Android Manifest
2. Implement background task (coroutine with a new Thread)
3. Create URI
4. Make HTTP Connection
5. Connect and GET Data
6. Process (parse) results

Android connectivity to the Internet

- Android does not allow to connect to the network in the main thread.
  - Start a new coroutine
  - Start a new thread that the coroutine will run
- Android will not allow a network connection without adding the relevant permission to the manifest file.

Adding the Network Permission to the Manifest file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.INTERNET"/>
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.BookFinderComposable"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.BookFinderComposable">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Coroutines

Used for concurrency (parallel execution of tasks within the same application/program).

- Similar to threads but more light weight.
- The coroutines are not bound to a specific thread. The may suspend their execution and resume in a different thread.
- To create a first coroutine (scope) from a regular function use `runBlocking{}`.
- Once inside a coroutine (scope) you can create additional coroutines running concurrently using `launch{}`
- `coroutineScope{}` can be used instead of `runBlocking{}` but only inside a coroutine
  - This means you need at least one `runBlocking{}` in a program using coroutines
  - Another difference is that `runBlocking` will block the main thread while `coroutineScope` suspends, releasing the underlying thread for other usages.
  - Both `runBlocking{}` and `coroutineScope{}` do not complete until all their body and children coroutines complete.

Creating the Background Task for Network Activity

1. Connect the main code with a new coroutine scope using a `runBlocking` block of code.
2. Start a new coroutine with a `launch` block of code.
3. Run the code of the coroutine in a new thread different than the main, e.g. by using `withContext(Dispatchers.IO)`.

Example:

```
runBlocking {
    launch {
        // run the code of the coroutine in a new thread
        withContext(Dispatchers.IO) {
            // code of coroutine goes here
        }
    }
}
```

What Runs Where?

```
import kotlinx.coroutines.*

fun main() {
    println("0: " + Thread.currentThread())

    runBlocking {
            println("1: " + Thread.currentThread())
            launch {
                println("2: " + Thread.currentThread())
                // run the code of the coroutine in a new thread
                withContext(Dispatchers.IO) {
                    println("3: " + Thread.currentThread())
                }
                println("4: " + Thread.currentThread())
            }
            println("5: " + Thread.currentThread())
    }
}
```

The output is:

```
0: Thread[main,5,main]
1: Thread[main @coroutine#1,5,main]
5: Thread[main @coroutine#1,5,main]
2: Thread[main @coroutine#2,5,main]
3: Thread[DefaultDispatcher-worker-1 @coroutine#2,5,main]
4: Thread[main @coroutine#2,5,main]
```

Using Dispatchers to Specify Threads for Coroutine Execution

To specify the thread that a coroutine runs, Kotlin provides 3 dispatchers:

- **Dispatchers.Main**: run a coroutine on the main Android thread. This should be used only for interacting with the UI and performing quick work.
- **Dispatchers.IO**: perform disk or network I/O outside of the main thread. Uses a shared background pool of threads.
- **Dispatchers.Default**: optimised to perform CPU-intensive work outside of the main thread. Uses a shared background pool of threads.

Add Kotlin Coroutines Dependencies in the gradle build file

This is needed if you need to use the Dispatchers.Main coroutine dispatcher which you cannot use for networking as you cannot connect to the network via the main thread.

Add the following in the module `build.gradle` file, in the dependencies section in the very end:

```
implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.10.1")
```

For the latest version number check:

https://github.com/Kotlin/kotlinx.coroutines/tree/45893cec51c63490ce294e46ae25cef3e4d625bf

The Book Finder App

Create an app which retrieves books from the Google Books Web service.

The manifest file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.INTERNET"/>
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.BookFinderComposable"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.BookFinderComposable">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

The Activity

```
package uk.ac.westminster.bookfindercomposable

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.verticalScroll
import androidx.compose.material3.Button
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.rememberCoroutineScope
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext
import org.json.JSONArray
import org.json.JSONException
import org.json.JSONObject
import java.io.BufferedReader
import java.io.InputStreamReader
import java.net.HttpURLConnection
import java.net.URL
```

The Activity (cont'd)

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

The Activity (cont'd)

```
@Composable
fun GUI() {
    var bookInfoDisplay by remember { mutableStateOf(" ") }
    // the book title keyword to search for
    var keyword by remember { mutableStateOf("") }

    // Creates a CoroutineScope bound to the GUI composable lifecycle
    val scope = rememberCoroutineScope()

    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Row {
            Button(onClick = {
                scope.launch {
                    bookInfoDisplay = fetchBooks(keyword)
                }
            }) {
                Text("Fetch Books")
            }
            TextField(value = keyword, onValueChange = { keyword = it })
        }

        Text(
            modifier = Modifier.verticalScroll(rememberScrollState()),
            text = bookInfoDisplay
        )
    }
}
```

The Activity (cont'd)

```
suspend fun fetchBooks(keyword: String): String {
    //val url_string = "https://www.googleapis.com/books/v1/volumes?q=android&maxResults=25"
    val url_string = "https://www.googleapis.com/books/v1/volumes?q=" + keyword + "&maxResults=25"
    val url = URL(url_string)
    val con: HttpURLConnection = url.openConnection() as HttpURLConnection

    // collecting all the JSON string
    var stb = StringBuilder()

    // run the code of the launched coroutine in a new thread
    withContext(Dispatchers.IO) {
        var bf = BufferedReader(InputStreamReader(con.inputStream))
        var line: String? = bf.readLine()
        while (line != null) {  // keep reading until no more lines of text
            stb.append(line + "\n")
            line = bf.readLine()
        }
    }

    val allBooks = parseJSON(stb)

    return allBooks
}
```

The Activity (cont'd)

```
fun parseJSON(stb: StringBuilder): String {
    // this contains the full JSON returned by the Web Service
    val json = JSONObject(stb.toString())

    // Information about all the books extracted by this function
    var allBooks = StringBuilder()

    var jsonArray: JSONArray = json.getJSONArray("items")
    // extract all the books from the JSON array
    for (i in 0..jsonArray.length() - 1) {
        val book: JSONObject = jsonArray[i] as JSONObject // this is a json object

        // extract the title
        val volInfo = book["volumeInfo"] as JSONObject
        val title = volInfo["title"] as String
        allBooks.append("${i + 1}) \"$title\" ")

        // extract all the authors
        try {  // in case there is no author in the info
            val authors = volInfo["authors"] as JSONArray
            allBooks.append("authors: ")
            for (i in 0..authors.length() - 1)
                allBooks.append(authors[i] as String + ", ")
        } catch (jen: JSONException) {
            // missing author in the information received
        }

        allBooks.append("\n\n")
    }

    return allBooks.toString()
}
```