

# 5COSC023W - MOBILE APPLICATION DEVELOPMENT

## Lecture 7: Network Connectivity and Background Tasks

Dr Dimitris C. Dracopoulos

# Steps to connect to the Internet

1. Add permissions to Android Manifest
2. Add Kotlin coroutines dependencies in your gradle build file
3. Implement background task (coroutine with a new Thread)
4. Create URI
5. Make HTTP Connection
6. Connect and GET Data
7. 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

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="uk.ac.westminster.bookfinderkotlin">

    <uses-permission android:name="android.permission.INTERNET"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.BookFinderKotlin">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

# Creating the Background Task

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**: optimized 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

Add the following in the app's `build.gradle` file, in the dependencies section in the very end:

```
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.6.4'
```



# 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"
    package="uk.ac.westminster.bookfinderkotlin">

    <uses-permission android:name="android.permission.INTERNET"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.BookFinderKotlin">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

# The Layout file of the activity

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
    <TextView
        android:id="@+id/tv"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:textSize="24sp" />
</ScrollView>
</LinearLayout>
```

# The Activity

```
package uk.ac.westminster.bookfinderkotlin

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.TextView
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.runBlocking
import kotlinx.coroutines.withContext
import org.json.JSONArray
import org.json.JSONObject
import java.io.BufferedReader
import java.io.InputStreamReader
import java.net.HttpURLConnection
import java.net.URL
```

# The Activity (cont'ed)

```
class MainActivity : AppCompatActivity() {
    lateinit var tv: TextView

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

        tv = findViewById(R.id.tv)

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

        val url_string = "https://www.googleapis.com/books/v1/volumes?q=android&maxResults=15"
        val url = URL(url_string)
        val con: HttpURLConnection = url.openConnection() as HttpURLConnection

        runBlocking {
            launch {
                // run the code of the coroutine in a new thread
                withContext(Dispatchers.IO) {
                    var bf = BufferedReader(InputStreamReader(con.inputStream))
                    var line: String? = bf.readLine()
                    while (line != null) {
                        stb.append(line + "\n")
                        line = bf.readLine()
                    }

                    parseJSON(stb)
                }
            }
        }
    }
}
```

# The Activity (cont'ed)

```
suspend fun parseJSON(stb: java.lang.StringBuilder) {
    // 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 = java.lang.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
        val authors = volInfo["authors"] as JSONArray
        allBooks.append("authors: ")
        for (i in 0..authors.length()-1)
            allBooks.append(authors[i] as String + ", ")

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

    tv.setText(allBooks)
}
```