Application development for Android

Android project creation

Lifecycle of an activity

Object Application

Android

- An Android project consists of:
  - Manifest file
    - XML file with general application settings, registration of needs (permissions) and others
  - Variable number of components belonging to the following 4 possible types:
    - `Activity`
    - `Service`
    - `BroadcastReceiver`
    - `ContentProvider`
  - Resources (e. g., images, screen *layouts, ...)*
  - Files with build configurations and dependencies (e. g., `build.gradle`, `build.gradle.kts`)

Project creation

- Run Android Studio
- Choose "**New Project**" to start the creation project wizard
  - Choose the type of application
    - Platform: **Phone and Tablet** , Wear OS, TV, Automative
    - **No Activity**
  - "**Next**"
  - Specify the **name of the application** (e. g., Lesson2)
  - Enter the "**Package name**"
    - The *package name* should be constituted by the company domain, in reverse order, followed by the application name (without spaces), in lowercase
    - The *package name* must be unique
      - It will be used to identify the application at *Google Play Store*
    - **Suggestion: pt.isec.a<student_number>.<appname>**
  - Choose language : **Kotlin**
  - Select the Android version – Minimum API Level
    - Suggestion: **API 24** (try "*Help me choose*")

Project

- Constituted by…
  - *Manifests*
    - `AndroidManifest.xml`
    - General application settings
  - *Java/Kotlin*
    - *Java* (.java) or *Kotlin* (.kt) files organized into *packages*
  - *Resources* (res)
    - *drawables, layouts, menus, values, mipmap, …*
  - *Gradle Scripts*
    - Build Settings
      - Including additional libraries/dependencies
- During project development, other components may be added

Test the created project

- Create an emulator
    - If it has not already been created in the previous class

- Try running on the emulator
    - *Build* options (*build, clean, rebuild*, ...)
    - Execution options (*run, debug*, ...)

    - Verify that it does not display any application (depending on the Android version, an error may appear stating that there is no activity)
    - Open *Settings* on the emulator and find the created application among the installed applications

Compilation and execution

- After a project is compiled, a "*package*" is generated with the application
  - The package name will match the one defined for the application
  - Extension `.apk`
    - This file is a *jar* Java file, but digitally signed
- *apk* file must be downloaded and installed on the device
- In *Android Studio*, all that is needed is to click on the run option
  - … the project will be compiled, the *apk* will be generated and sent to the device connected to the computer or an emulator is launched to test the application (chosen through an additional window)

Manual activity creation

- Create a new folder named `layout` in the resources folder
- Create a layout *xml* file (`my_activity.xml`) inside that folder:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:background="#ffb040"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</FrameLayout>
```

- The two previous actions can be performed at once with the option to create a *Resource File* of type *Layout*, indicating `FrameLayout` as *Root element*

Manual activity creation

- Create a Kotlin class, `MyActivity` , derived from the `android.app.Activity` class, in the already available package
  - Process the `onCreate` event
    - In the context of the class start writing " `onCr` " and accept the suggestion of the `onCreate` function that has just one parameter
  - Inside this method, after the superclass method has been called, add a line with:

```
setContentView(R.layout.my_activity)
```

```
class MyActivity : Activity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.my_activity)
    }
}
```

Manual activity creation

- Register the activity in the manifest file, in the context of the `application` structure

```
<activity android:exported="true" android:name=".MyActivity">
</activity>
```

- Add an `intent-filter` with the `MAIN` *action* and `LAUNCHER` *category*

```
<intent-filter>
  <action android:name="android.intent.action.MAIN" />
  <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

Debug

- It is possible to *debug* as in regular *Java* applications
  - Insert *breakpoints*, execute instructions step-by-step, add *watches*,…

- The *Android* 's existing *log system* can be used to help debugging
  - Use methods of the `android.util.Log` class (`Log.d`, `Log.i`, `Log.w`, …)
    - Insert a log line into the `onCreate` method

      ```
      Log.i("AMovApp", "onCreate:")
      ```

  - the logs can be read using *Logcat*
    - Available on *Android Studio*

Debug

- There are several tools available on *Android Studio* that can assist us in the tasks of building and verifying applications
  - Integrated into the environment itself
  - Executed from the command line
    - adb
      - Available in the `platform-tools` folder
      - It allows to consult logs, upload and download files, run a shell on the device, etc.
        - Examples:
          - `adb logcat`
          - `adb shell`

Lifecycle of an activity: Practical activity

- Process events that occur in the lifecycle of an activity
  - Process the following methods and generate an appropriate *log message* in each of them
    - `onCreate`
    - `onStart`
    - `onRestart`
    - `onResume`
    - `onPause`
    - `onStop`
    - `onDestroy`
    - `onSaveInstanceState`
    - `onRestoreInstanceState`

Enter in each method:

```
   Log.i("AMovApp", "<method name>");
```

Lifecycle of an activity

- With the help of *logcat*, analyze the message order when...
  - Starting the application
  - Finalizing the application
  - Restarting the application
  - Pressing the home button
  - Rotating the screen with the application active (Ctrl+F11/F12)
  - Other situations (e.g., making a call and answering or declining it, using *Google Assistant*, ...)

Application

- Create an object of type `android.app.Application`
  - Name `MyApp`

- Configure the object in the manifest file
  - Add the `name` attribute to the `<application>` *tag*, setting its value to the name of the created `Application` class

- Insert *a log* line in the `onCreate`
  - Check for other "*`onXXX`*" methods

Application

- Suggestion:
  - Add an integer counter to the `Application` object
    - Implement with the help of a *Kotlin property* that automatically increments the value

```
private var _my_value = 0
val my_value : Int
      get() = ++_my_value
```

  - Display the counter value in all the defined log lines
    - Use the `application` property to access the `Application` object
    - Use a *lazy* variable in the `MyActivity` class to access and *cast* the `Application` object to `MyApp`

```
val app : MyApp by lazy { application as MyApp }
```

Exercise with an object (singleton)

- Create a counter similar to the one placed in the `MyApp` class, but implemented through an `object` (singleton)
  - Name it: `MyObject`

- Check messages generated in the context of the application lifecycle