Lab 1: Primer on Android development

Design and Development of Mobile Applications
E. De Coninck, S. Leroux, P. Simoens
2015-2016

This lab session will get you to grips with the basics of Android development. You will create a 'Hello World' app to get acquainted with the debugging and compilation tools of the Android IDE. We will experiment with the different types resource files and pass `Intents` between activities.

1 IDE: Android Studio

In this course we will use Android Studio, the official Android IDE since 2014. This program is installed and pre-configured on the lab PCs, together with the drivers for the mobile devices that are at your disposal during the lab sessions. To configure your own laptop, please refer to `http://developer.android.com/sdk/index.html`.

- Open Android Studio and create a new Android Project without any Activities. In this course, we will only develop apps for phones and/or tablets that have SDK API 18 or higher. By selecting a high API we can limit the use of support libraries that make newer features available on older devices but increase complexity.
- Get to know the project structure and open the minimal manifest file. You can change the view type of your app between 'Android' and 'Project'. The project view reflects the structure on disk (APK) and the Android view type is a flattened version of your project's structure that provides quick access to key source files.

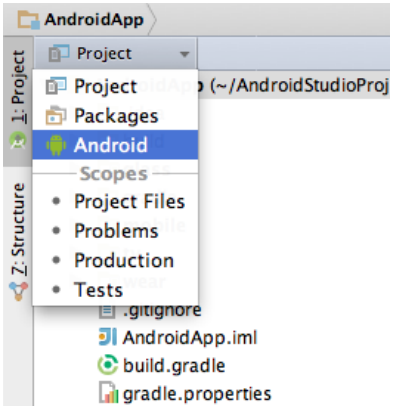


Figure 1: Toggling between project views.

The build system uses the Gradle build tool. In the Android project view you can find all Gradle related files in one section. Your Gradle application settings can be found in 'build.gradle (Module: app)', which

is the only .gradle file we will use during this course.

1.1 Android SDK Manager

The Android SDK separates tools, platforms, and other components into packages you can download using the SDK Manager. When the SDK Tools or a new version of the Android Platform is released, you can conveniently use the SDK Manager to update your environment.

- Open SDK Manager: **Tools** > **Android** > **SDK Manager** or by clicking the corresponding menu bar icon.

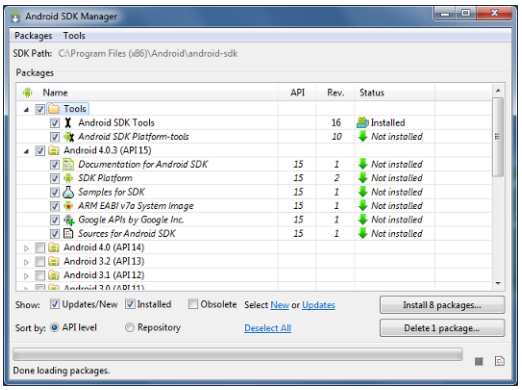


Figure 2: The Android SDK Manager lists the status of each package: available, installed, or updateable.

- Check if all required packages are installed:
  - Latest Android SDK Tools
  - Latest Android SDK Platform-tools
  - Android SDK Build-tools 18 or higher
  - Android 4.3.1 (API 18) or higher
    - SDK Platform
    - Intel x86 Atom(_64) System Image or Google APIs Intel x86 Atom(_64) System Image
  - Android Support Library
  - Google USB Driver (Requires root access)
  - Intel x86 Emulator Accelerator (HAXM installer requires root access): HAXM is an installer for hardware acceleration of the Android emulator images. Without this package the emulator is slow. This package is not supported on all machines.

1.2 Android Virtual Device (AVD) Manager

The AVD Manager is a tool to create and manage Android Virtual Devices (AVDs), device images that are executed by the Android Emulator. This allows you to develop and test Android apps without having to install them on a physical device. The emulator is slow and it can take several minutes before a virtual device image is booted.

- Open AVD Manager: **Tools** > **Android** > **AVD Manager** or click the corresponding icon in the toolbar.
- Create Virtual Device if Nexus 5 device with API 18 or higher is missing
  - Select Nexus 5 with API 18 or higher and system image x86(_64). The ARM images will run a lot slower since you are executing this on a CPU with x86 architecture.
  - In the Advanced Settings menu, change the RAM to 512 MB and enable keyboard input so your keyboard can be used as an input device.

- Start an instance of this new device. This may take a while to complete, so keep this instance running for the entire lab session.

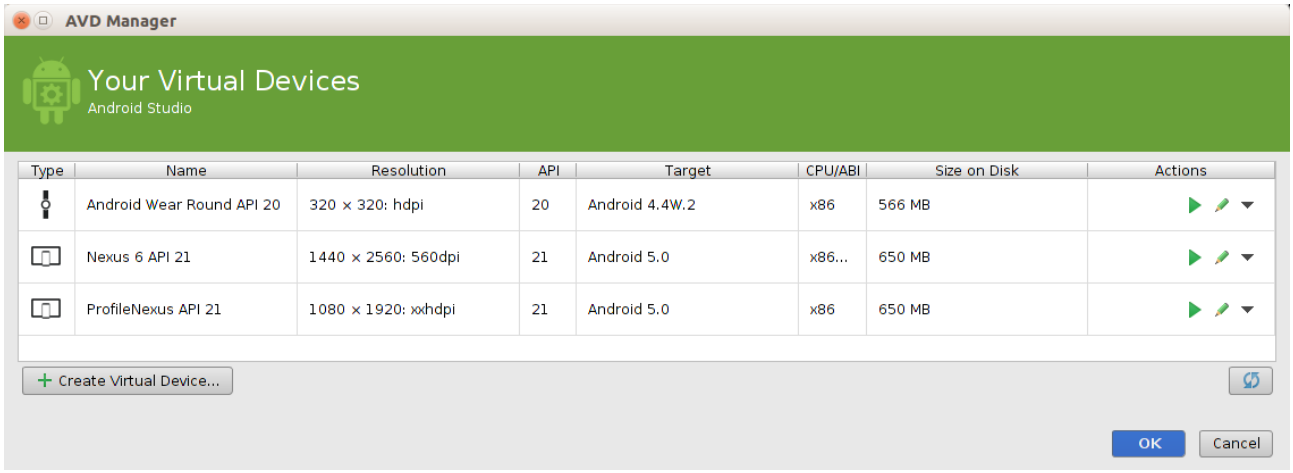


Figure 3: The AVD Manager main screen provides an overview of all your virtual devices.

2 Debugging with adb logcat

Android Debug Bridge (`adb`) is a versatile command line tool to communicate with an emulated instance or a connected physical Android device. You can find the `adb` tool in `<sdk>/platform-tools/`.

- In order to use `adb` with a physical device connected over USB, you must enable **USB debugging** in the device system settings, under **Developer options**. On Android 4.2 and higher, the Developer options screen is hidden by default. To make it visible, go to **Settings** > **About phone** and tap **Build number** seven times. Return to the previous screen to find **Developer options** at the bottom.
- Create a new blank launcher activity in your project's default package. This will create a Java class which extends `Activity` in your default package, a layout XML file in `res/layouts` and a reference to the newly created `Activity` in your application's manifest file.

Background information: The launcher or main activity is the default activity displayed when your app is started. The label of this activity is visible on the action bar and in the launcher of your device (application list and home screen). The label of the `application` tag in your manifest is visible in the application management section of the settings.

- Override all lifecycle callbacks in your `Activity` class. Use the automatic code generation via right mouse click **Generate...** > **Override Methods...**

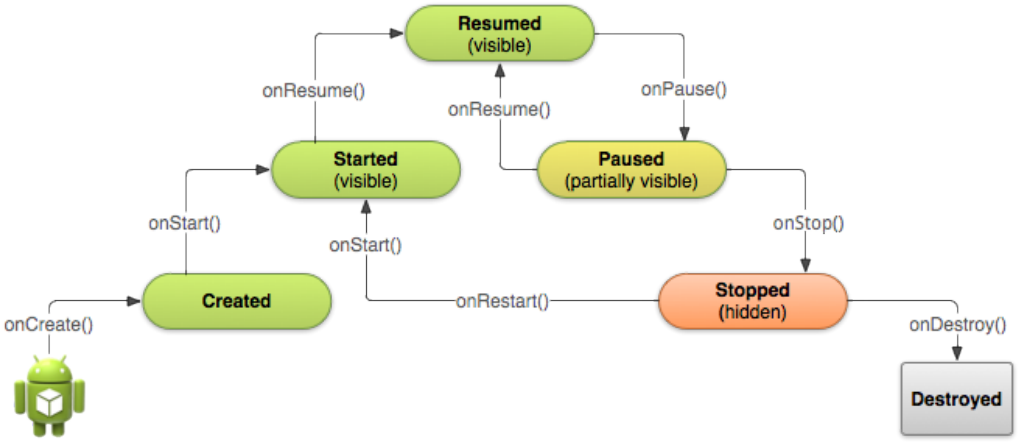


Figure 4: The `Activity` lifecycle depicted as a step pyramid. For every callback used to take the activity a step toward the *Resumed* state at the top, there is a callback method that takes the activity a step down. The activity can also return to the resumed state from the *Paused* and *Stopped* state.

- Write log messages in each lifecycle callback with different log levels.

To write log messages in your code, use the Log class. Log messages help you understand the execution flow by collecting the system debug output while you interact with your app. Log messages can tell you what part of your application failed. By default, the Android system sends `stdout` and `stderr` (`System.out` and `System.err`) output to /dev/null. On the emulator and some devices `stdout` and `stderr` gets redirected to logcat and printed using `Log.i()`. Make sure to always use the Log class for logging messages.

- View log messages in the Dalvik Debug Monitor Server (DDMS) and adjust the log level.
  - Start your application
  - Click **Android** at the bottom of the screen to open the *Android DDMS* tool window.
  - If the system log is empty in the Logcat view, click **Restart**.

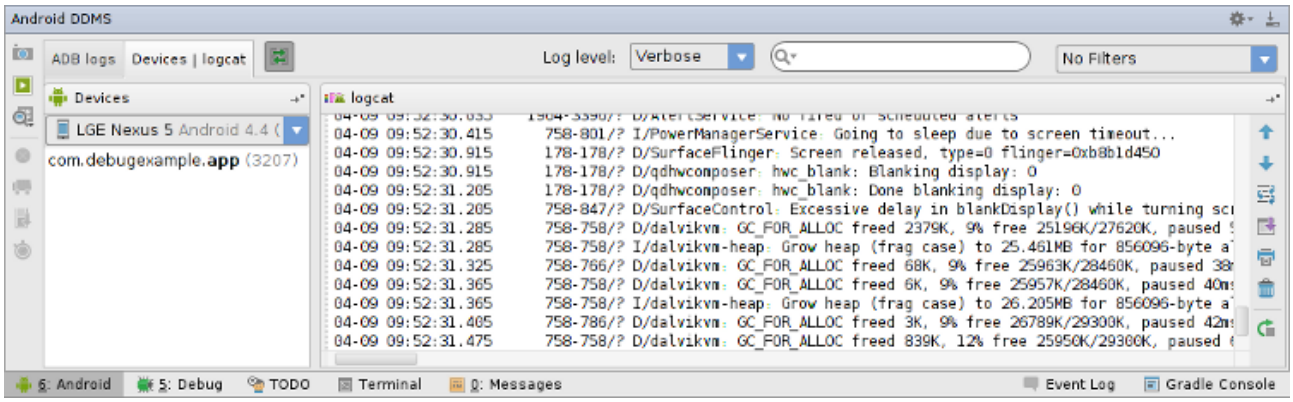


Figure 5: The system log in the Android DDMS tool window.

- Which lifecycle callbacks are called in the following scenarios: i) when pressing the home button, ii) removing a task from recent apps and iii) on device rotation?

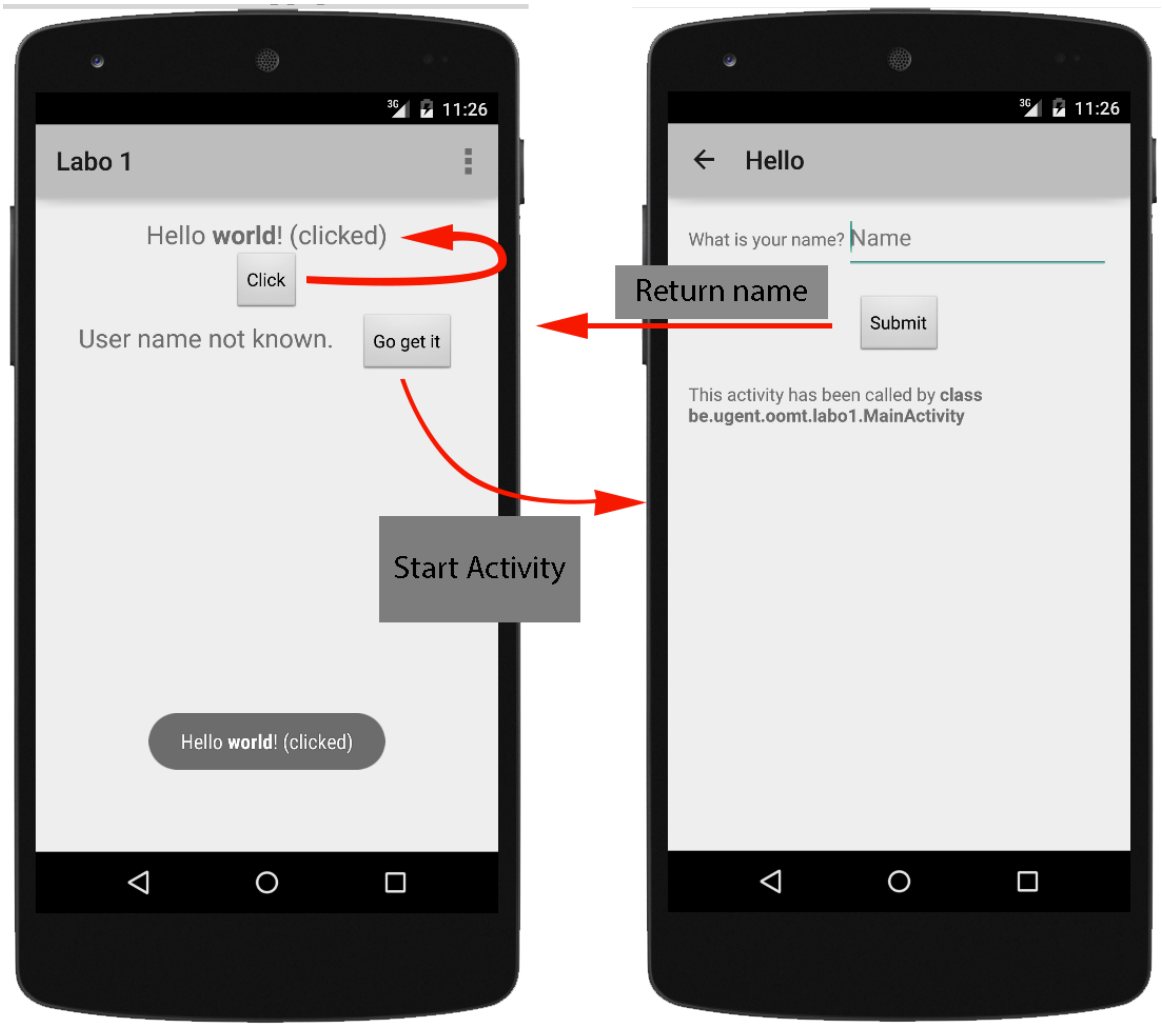
Hint: All super methods of the lifecycle methods need to be called. Methods that occur before the activity becomes visible, will have the **`super`** call as its first line. Methods called before the activity becomes hidden, will have the **`super`** call as its last line.

3 First Android application: Hello World!

The Blank Activity template includes a XML layout file in the `app/res/layout` folder with a `RelativeLayout` root view and a `TextView` child view. When you open a layout file in Android Studio, you are first presented the WYSIWYG editor with the Preview pane. In this lab session however we will edit the plain XML. At the bottom of the screen, you can switch between the *Design view* and the *Text view* of the currently opened XML layout file. Both view types show a preview of the app.

Hints:

- Use <ctrl> + <space> to open suggestions based on the location of the cursor.
- Missing attributes or methods can be automatically generated by moving the cursor onto the desired location or method. The red light bulb in the left margin of the editor can be used to generate a string resource entry from hard coded strings in code or XML.
- Keep in mind that `px` (pixel) is not used as a measure for size in Android. Instead, `dp` (density independent pixels) is used for views, and `ds` (scaled independent pixels) for fonts.



- Use a vertically oriented `LinearLayout` root container in the layout file. Within this `<LinearLayout>` element, define a `TextView` (displaying Hello World!) and a `Button`, each with their own unique `id` attribute (`@+id/name`). Register an `onClick` listener in XML and implement this callback method in the corresponding `Activity`.

  In order for the system to be able to match this listener method, its signature must have the following structure:

- `public` access modifier
- `void` return value
- a single `View` object parameter (this will be the `View` that was clicked)

When the button is pressed, change the text of the first `TextView` and show a `Toast` message. Instantiate a `Toast` message object using one of the `makeText()` methods. These methods take three parameters: the application `Context` or `Activity`, the text message, and the time that the toast message must be displayed. It returns a properly initialized `Toast` object that can be displayed by calling `show()`.

To change the text of the first `TextView` you first need to get a reference using the `findViewById()` method. Change the text every time the button is pressed by appending the current time to a fixed string resource entry (for example: 'Hello World!'). Hint: use `getString(R.string.id)`. Test and run your first interactive activity.

- Add a second `TextView` and a second `Button` in a horizontal `LinearLayout` that is a child of the root `LinearLayout`. The `TextView` is a placeholder for your name which we will get from a second `Activity` which we will create later in this lab. Instead of registering the click listener of the button in the XML file, we can also register a `View.onClickListener` in Java code. Create an object reference to this second `TextView` and the second `Button` in the `onCreate()` method, and register an anonymous inner class as the click listener for the button. When this button is pressed, a new `Activity` must be started and the name of the current `Activity` will be passed as an argument. But first we will create this new Activity.

Every activity is invoked by an `Intent`, regardless of how the user navigated there. You can get the `Intent` that started your activity by calling `getIntent()` and retrieve the data contained within the intent.

- Create a new blank activity which has an `EditText` to fill in a name, a submit `Button` to return to the calling `Activity`, and a `TextView` to show the name of the `Activity` that started the current `Activity`. You are free to configure and style this `Activity` in code or in XML. Ensure that the second `Activity` is added to your project manifest file and implement the following behaviour:
  - Open this second `Activity` when the second button is pressed in the main `Activity`.
  - Show the name of the calling `Activity` in the `TextView`.
  - When this `Activity`'s button is pressed return the content of the `EditText` to the calling `Activity`.
  - Show the returned name in the `TextView` placeholder of the main `Activity`.
- Translate your application by adding a new strings resource file with a language qualifier to the values folder. This will create a new directory and a file in the *Project* view with a name akin to `values-nl/strings.xml`. Hint: use the resource new file wizard to see all qualifiers and their values.
- As a last step style your application. Find the appropriate xml attributes to change text size, margins and padding and try to center all widgets. Note: you can change attributes for the entire application by using the `styles.xml` resource (`http://developer.android.com/guide/topics/ui/themes.html`) file, but make sure to keep the base theme as the parent theme. Try to change the background color of all buttons in your application using `styles.xml`.