ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Desenvolvimento de Aplicações Móveis
Mobile Application Development
DAM
Tutorial 1 - Hello World

Ana Duarte Correia
ana.correia@isel.pt

Pedro Fazenda
pedro.fazenda@isel.pt

Abstract

This tutorial presents the Android development and execution environments and the development of a Hello World Android application.

Deadline: 21st March 2020

Hello Mobile Android World!!

24 February 2020

Contents

List of Figures

1 Introduction

This tutorial starts with an introduction to Android, then it proceeds with the download and installation of the official Android Integrated Development Environment (IDE) that is the Android Studio. The tutorial proceeds with the configuration and update of the necessary tools. Thereupon, it requires the development of a simple "Hello World" application.

Follows a link that should be consulted during the development of Android applications.

Android Developers `https://developer.android.com/guide`

Do not miss the tutorial deadline!!!

All tutorials are expected to be finished and presented to the teacher during class hours. The resulting projects must be completed, presented, evaluated and uploaded to the course page on the Moodle platform in class, until the deadline date shown on the cover page. After each deadline date the final grade, for the project, will have **a penalty value of 5/20 for each passing week/Monday** until the project is presented and evaluated by the teacher in class. This condition applies regardless if the project was submitted or not on the Moodle platform.

2 Introduction to Android

Android OS was originally created inside Android Inc. by Andy Rubin in the early 21st century as a mobile operating system. In 2005, Google acquired Android Inc. and made Andy Rubin the director of Google's mobile platforms.

Android OS is an operating system supported by the **Linux Kernel**. That's why Android devices are essentially Linux computers. The project responsible for developing the Android system is called "Android Open Source Project (AOSP)" and is led by Google.

The **Dalvik Virtual Machine (DVM)** is the executor of Android applications. The compilation of Android code generates Dalvik Executable format files with extension **.dex**. These files, plus resources and the manifest files are grouped in **.apk** files. One **.apk** file is one Android application ready to be installed.

There are four (plus one - Layouts) main components in Android applications (see `https://developer.android.com/guide/components/fundamentals`):

- **Activities** - (organizational layer) an Activity supports a screen for user interaction;
- **Layout Containers** - (presentation layer) a Layout is a container for user interface elements in a screen;
- **Services** - (background processing layer) a Service is a component that runs in background, providing services for Activities;

- **Broadcast Receivers** - (notification layer) a Broadcast Receiver is a component that allows the Activities to receive notifications from the system (and react to them);
- **Content Providers** - (storage layer) a Content Provider is a component that store and provide data for Activities.

There are also some important helper classes:

- Android **Fragments** - they are reusable parts of the user interface;
- Android **Intents** - they enable communication between components.

These elements will be introduced in more detail when they are used. For now, we will focus on Android **Activities** and **Layouts**.

An **activity** is a screen, in an application, providing the context for one user interaction. An application may have multiple activities, providing multiple contexts for the user. This is the way used to spread all the information, of the application, in the reduced size screens of mobile devices. An activity has one Layout Container to join the user interface elements. One activity can also change its Layout Container and provide a mutable user interface.

A **Layout Container** groups design elements, called **Widgets**, to build a screen. The design elements can be user interface elements for text, graphics, 3D content, digital video, menus, animation, ...

Layouts Containers are built based on **ViewGroup** class and **Widgets** based on **View** class.

For Android graphics such as images or animations the term **Drawables** is used.

The Event Handling capabilities of Android User Interface elements offers a way to applications react to user actions or other source of events.

3 Preparatory actions

To enable the development and execution of Android applications this tutorial presents the installation of IDE, the development libraries (Software Development Kit (SDK)) and emulation capability.

3.1 Download and installation of Android Studio

1. Download the Android Studio from the following link:

   `https://developer.android.com/studio`

   The Android Studio package includes all the tools necessary for the development of android applications:

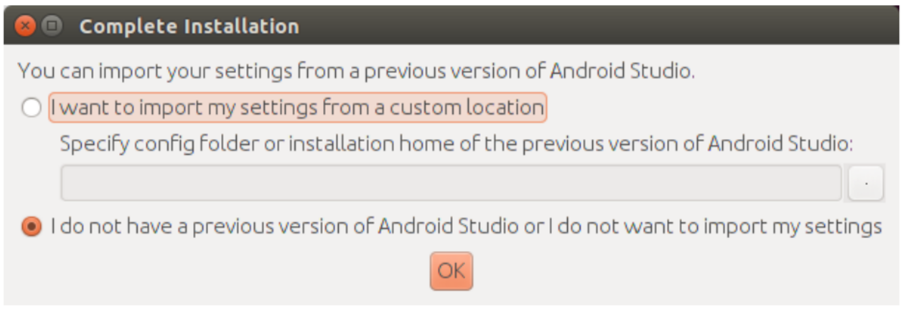
   - **Android Studio IDE** - to edit, run, debug Android code
   - **Android SDK and Tools Manager** - to install Android SDKs and tools

- **Android Emulator** - to execute Android apps in Android emulated devices
- **Android Virtual Device (AVD) manager** - to create and manage AVDs.

Because the programming language used is Java you must have installed the Java Development Kit (JDK) and the Java Runtime Environment (JRE).

2. Execute the "android-studio-ide-xxx.xxxxxxx-windows.exe" file obtained with the previous download to install the Android Studio package.

3. Run the Android Studio IDE. The first time you execute Android Studio you can choose whether you want to import applications or settings from a previous version as shown in Figure 1.



You can import your settings from a previous version of Android Studio.

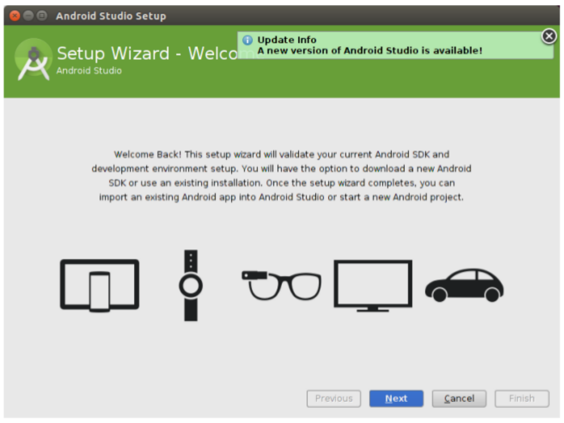
○ I want to import my settings from a custom location

Specify config folder or installation home of the previous version of Android Studio:

● I do not have a previous version of Android Studio or I do not want to import my settings

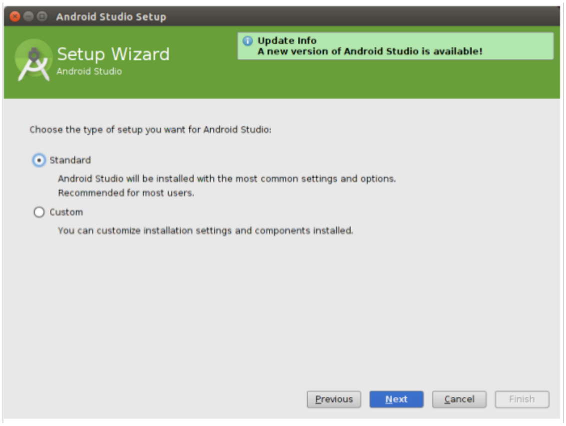
Figure 1: Android Studio first execution configuration

4. After clicking the "OK" button you will go through several configuration steps in order to install Android Studio (see Figures 2 to 5).



Welcome Back! This setup wizard will validate your current Android SDK and development environment setup. You will have the option to download a new Android SDK or use an existing installation. Once the setup wizard completes, you can import an existing Android app into Android Studio or start a new Android project.

Figure 2: SETUP: Start Setup Wizard.



Choose the type of setup you want for Android Studio:

● Standard

Android Studio will be installed with the most common settings and options. Recommended for most users.

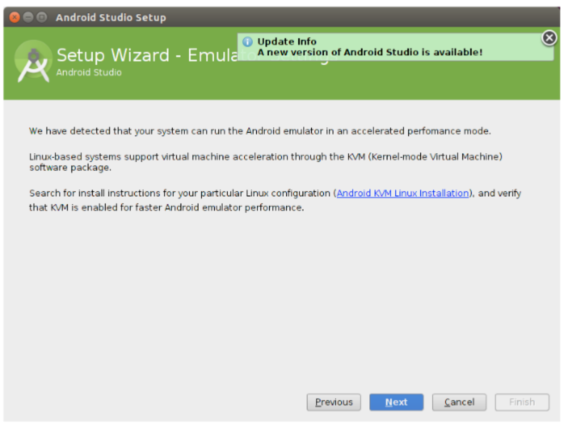
○ Custom

You can customize installation settings and components installed.

Figure 3: SETUP: Option "Standard".

3.2 Installation of Android development libraries (SDKs)

There are several versions of the Android platform. They have a version number and name and an API number, like Android 5.0 Lollipop (API 21), Android 8.0 Oreo (API 26) and Android 10 Q (API 29). To develop code for one platform it is needed its libraries, called Software Development kit (SDK). So, to develop code for Android 8.0 it is needed its Android SDK platform and tools. For the management of the SDKs installed, or to install new ones, there is the Android SDK Manager. It is recommended to have installed

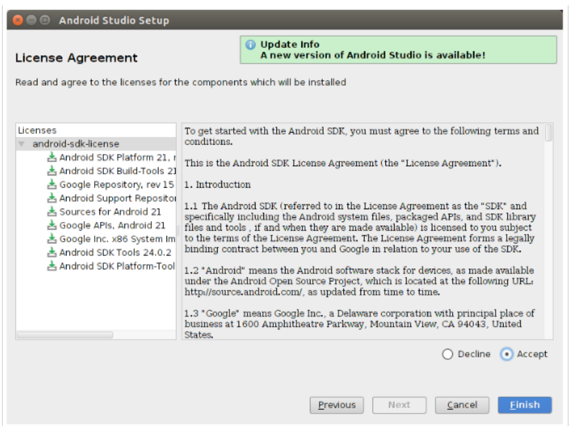


We have detected that your system can run the Android emulator in an accelerated perfomance mode.

Linux-based systems support virtual machine acceleration through the KVM (Kernel-mode Virtual Machine) software package.

Search for install instructions for your particular Linux configuration (Android KVM Linux Installation), and verify that KVM is enabled for faster Android emulator performance.

Figure 4: SETUP: Some PC's may have this option to speed up the emulator.


Android Studio Setup
Update Info
A new version of Android Studio is available!
Licenses
android-sdk-license
Android SDK Platform 21, r
Android SDK Build-Tools 21
Google Repository, rev 15
Android Support Repositor
Sources for Android 21
Google APIs, Android 21
Google Inc. x86 System Im
Android SDK Tools 24.0.2
Android SDK Platform-Tool
Decline
Accept
Previous
Next
Cancel
Finish

License Agreement

Read and agree to the licenses for the components which will be installed

To get started with the Android SDK, you must agree to the following terms and conditions.

This is the Android SDK License Agreement (the "License Agreement").

1. Introduction

1.1 The Android SDK (referred to in the License Agreement as the "SDK" and specifically including the Android system files, packaged APIs, and SDK library files and tools , if and when they are made available) is licensed to you subject to the terms of the License Agreement. The License Agreement forms a legally binding contract between you and Google in relation to your use of the SDK.

1.2 "Android" means the Android software stack for devices, as made available under the Android Open Source Project, which is located at the following URL: http://source.android.com/, as updated from time to time.

1.3 "Google" means Google Inc., a Delaware corporation with principal place of business at 1600 Amphitheatre Parkway, Mountain View, CA 94043, United States.

Decline [✓ Accept]

Figure 5: SETUP: Finishing the installation.

the Android 10 Q (API 29) and another one like Android 6.0 Marshmallow (API 23) or the highest one that is supported by the student phone.

1. Open the Android SDK Manager: it can be done in the "Welcome to Android Studio" window (when there is no Project opened), by selecting the **"Configure"** drop down list and choosing **SDK Manager** (see Figure 6) or when we are working on a project by opening the menu **Tools** > **SDK Manager** or by selecting the **SDK Manager icon** in the toolbar (see Figure 7).

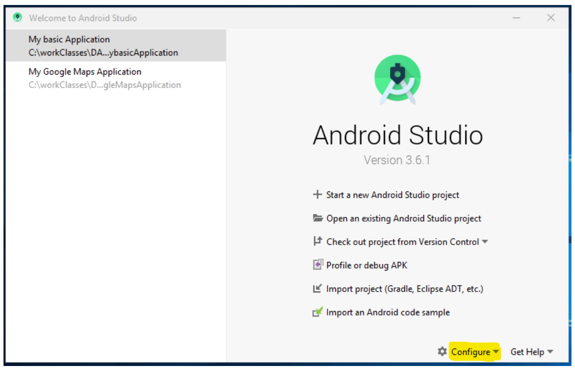

Welcome to Android Studio
My basic Application
C:\workClasses\DA...ybasicApplication
My Google Maps Application
C:\workClasses\D...gleMapsApplication
Android Studio
Version 3.6.1
Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Profile or debug APK
Import project (Gradle, Eclipse ADT, etc.)
Import an Android code sample
Configure
Get Help

Figure 6: Configure in Welcome window.

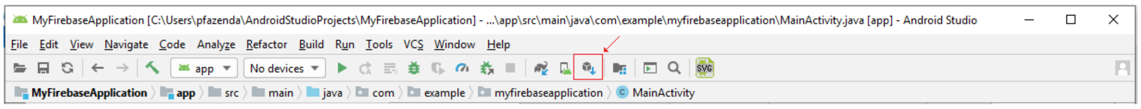
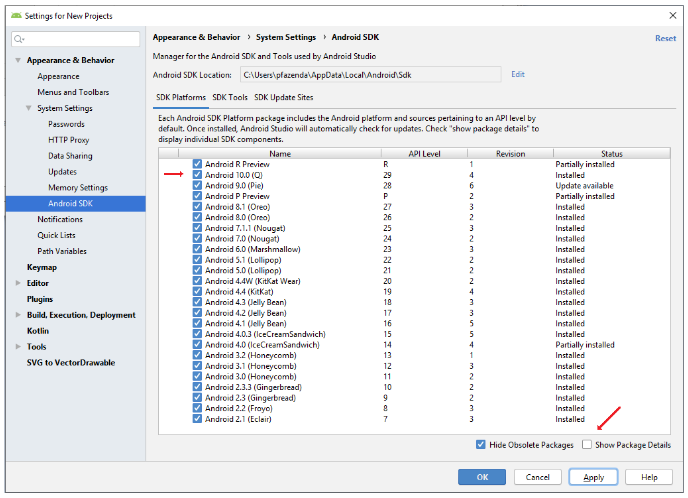

MyFirebaseApplication [C:\Users\pfazenda\AndroidStudioProjects\MyFirebaseApplication] - ...\app\src\main\java\com\example\myfirebaseapplication\MainActivity.java [app] - Android Studio
File
Edit
View
Navigate
Code
Analyze
Refactor
Build
Run
Tools
VCS
Window
Help
app
No devices
MyFirebaseApplication
app
src
main
java
com
example
myfirebaseapplication
MainActivity

Figure 7: SDK manager icon (red element in the toolbar).

2. In the SKD Manager panel (see Figure 8) select the Android version you wish to develop and are not installed by checking them and then press the "OK" button.

   Select "Show Package Details" check-box to see in detail the components that are part of each version of the Android API.

   In SDK Manager the "SDK Platforms" tab is used to install Android API versions and the "SDK Tools" tab is used to install development tools.



Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: C:\Users\pfazenda\AppData\Local\Android\Sdk

Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, Android Studio will automatically check for updates. Check "show package details" to display individual SDK components.

Name	API Level	Revision	Status
✓ Android R Preview	R	1	Partially installed
✓ Android 10.0 (Q)	29	4	Installed
✓ Android 9.0 (Pie)	28	6	Update available
✓ Android P Preview	P	2	Partially installed
✓ Android 8.1 (Oreo)	27	3	Installed
✓ Android 8.0 (Oreo)	26	2	Installed
✓ Android 7.1.1 (Nougat)	25	3	Installed
✓ Android 7.0 (Nougat)	24	2	Installed
✓ Android 6.0 (Marshmallow)	23	3	Installed
✓ Android 5.1 (Lollipop)	22	2	Installed
✓ Android 5.0 (Lollipop)	21	2	Installed
✓ Android 4.4W (KitKat Wear)	20	2	Installed
✓ Android 4.4 (KitKat)	19	4	Installed
✓ Android 4.3 (Jelly Bean)	18	3	Installed
✓ Android 4.2 (Jelly Bean)	17	3	Installed
✓ Android 4.1 (Jelly Bean)	16	5	Installed
✓ Android 4.0.3 (IceCreamSandwich)	15	5	Installed
✓ Android 4.0 (IceCreamSandwich)	14	4	Partially installed
✓ Android 3.2 (Honeycomb)	13	1	Installed
✓ Android 3.1 (Honeycomb)	12	3	Installed
✓ Android 3.0 (Honeycomb)	11	2	Installed
✓ Android 2.3.3 (Gingerbread)	10	2	Installed
✓ Android 2.3 (Gingerbread)	9	2	Installed
✓ Android 2.2 (Froyo)	8	3	Installed
✓ Android 2.1 (Eclair)	7	3	Installed

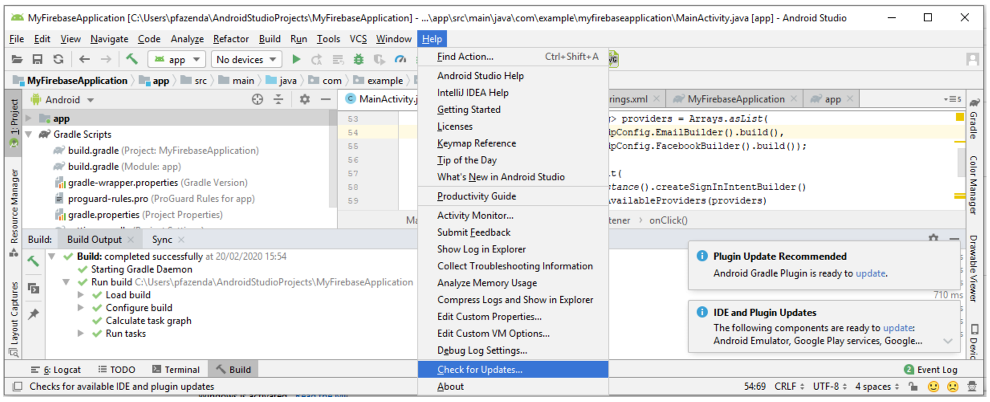
✓ Hide Obsolete Packages ☐ Show Package Details

Figure 8: SDK Manager: SDK Platforms

3.3 Checking for Updates

It is recommended that the Android Studio and all the platforms and tools to be updated. After installation is a good moment to check for the latest updates.

1. Check for updates: in the "Welcome to Android Studio" window, when there is no Project opened, by selecting the **"Configure"** drop down list and choosing **Check for updates** (see Figure 6) or when we are working on a project by opening the menu **Help** > **Check for updates...** (see Figure 9) (do not install Beta versions).


MyFirebaseApplication [C:\Users\pfazenda\AndroidStudioProjects\MyFirebaseApplication] - ...\app\src\main\java\com\example\myfirebaseapplication\MainActivity.java [app] - Android Studio
File
Edit
View
Navigate
Code
Analyze
Refactor
Build
Run
Tools
VCS
Window
Help
Find Action... Ctrl+Shift+A
Android Studio Help
IntelliJ IDEA Help
Getting Started
Licenses
Keymap Reference
Tip of the Day
What's New in Android Studio
Productivity Guide
Activity Monitor...
Submit Feedback
Show Log in Explorer
Collect Troubleshooting Information
Analyze Memory Usage
Compress Logs and Show in Explorer
Edit Custom Properties...
Edit Custom VM Options...
Debug Log Settings...
Check for Updates...
About
app
No devices
Android
app
Gradle Scripts
build.gradle (Project: MyFirebaseApplication)
build.gradle (Module: app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
Build:
Build Output
Sync
6: Logcat
TODO
Terminal
Build
Event Log
54:69 CRLF UTF-8 4 spaces

```
Build: completed successfully at 20/02/2020 15:54
  Starting Gradle Daemon
  Run build C:\Users\pfazenda\AndroidStudioProjects\MyFirebaseApplication
    Load build
    Configure build
    Calculate task graph
    Run tasks
```

Plugin Update Recommended
Android Gradle Plugin is ready to update.

IDE and Plugin Updates
The following components are ready to update:
Android Emulator, Google Play services, Google...

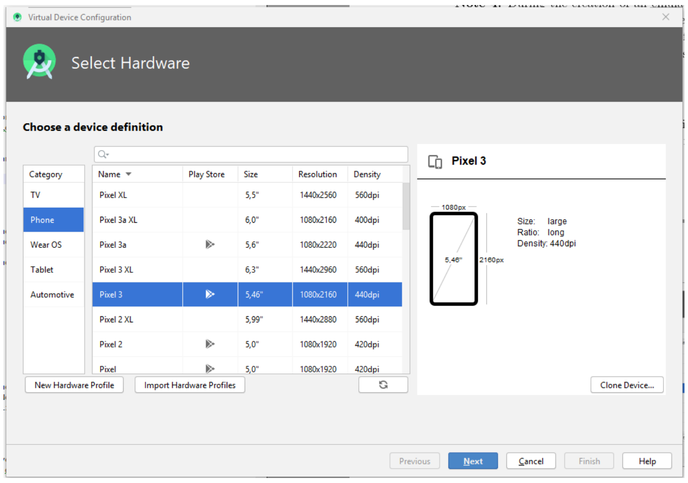
Checks for available IDE and plugin updates

Figure 9: Android Studio: Menu and "Check for Updates".

3.4 Creating Android Virtual Devices (AVDs)

An Android Virtual Device (AVD) is a piece of software that behaves similarly to one physical device. AVDs allows to test Android applications in different Android versions and hardware profiles without using physical devices. The AVD Manager is a tool that allows the creation of AVDs of the following types of devices: Phone, Tablet, TV, Automotive and Wear OS. It contains several hardware profiles but it can create or import other ones. You should have at least two AVDs: one **Pixel 3** smartphone and one **Pixel C** tablet. The AVD Manager gives the possibility to create AVDs for a specific hardware profile and for a specific Android platform version. Therefore, to have an AVD we need to choose one hardware profile and one Android platform system image. You should select the latest Android API version that is compatible with your device so that you can run applications on the virtual device and on your device without having to make changes to the code.

1. Open the AVD Manager either in "Welcome to Android Studio" window (when there is no Project opened) by selecting the **"Configure"** drop down list and choosing **AVD Manager** (see Figure 6) or when we are working on a project by opening the menu **Tools** > **AVD Manager** or by selecting the **AVD Manager icon** in the toolbar (the one at the left of SDK Manager in Figure 7).


Virtual Device Configuration

Select Hardware

Choose a device definition

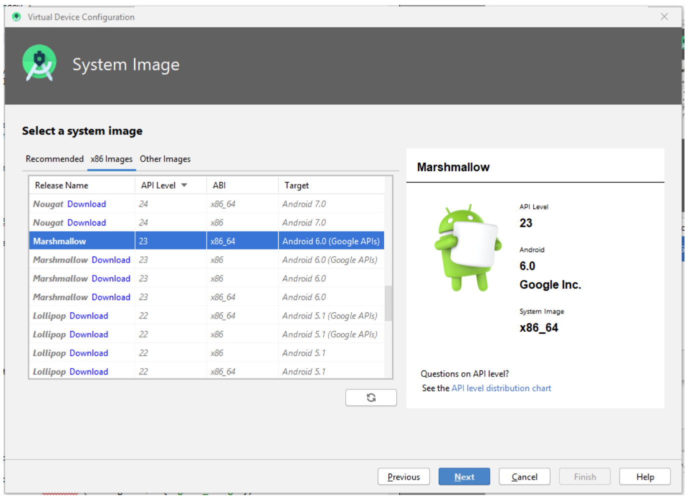
Category
TV
Phone
Wear OS
Tablet
Automotive

Name	Play Store	Size	Resolution	Density
Pixel XL		5,5"	1440x2560	560dpi
Pixel 3a XL		6,0"	1080x2160	400dpi
Pixel 3a	▶	5,6"	1080x2220	440dpi
Pixel 3 XL		6,3"	1440x2960	560dpi
Pixel 3	▶	5,46"	1080x2160	440dpi
Pixel 2 XL		5,99"	1440x2880	560dpi
Pixel 2	▶	5,0"	1080x1920	420dpi
Pixel	▶	5,0"	1080x1920	420dpi

Pixel 3

Size: large
Ratio: long
Density: 440dpi

Figure 10: Virtual Device: choosing hardware profile.


Virtual Device Configuration

System Image

Select a system image

Release Name	API Level	ABI	Target
Nougat Download	24	x86_64	Android 7.0
Nougat Download	24	x86	Android 7.0
Marshmallow	23	x86_64	Android 6.0 (Google APIs)
Marshmallow Download	23	x86	Android 6.0 (Google APIs)
Marshmallow Download	23	x86	Android 6.0
Marshmallow Download	23	x86_64	Android 6.0
Lollipop Download	22	x86_64	Android 5.1 (Google APIs)
Lollipop Download	22	x86	Android 5.1 (Google APIs)
Lollipop Download	22	x86	Android 5.1
Lollipop Download	22	x86_64	Android 5.1

Marshmallow

API Level
23

Android
6.0
Google Inc.

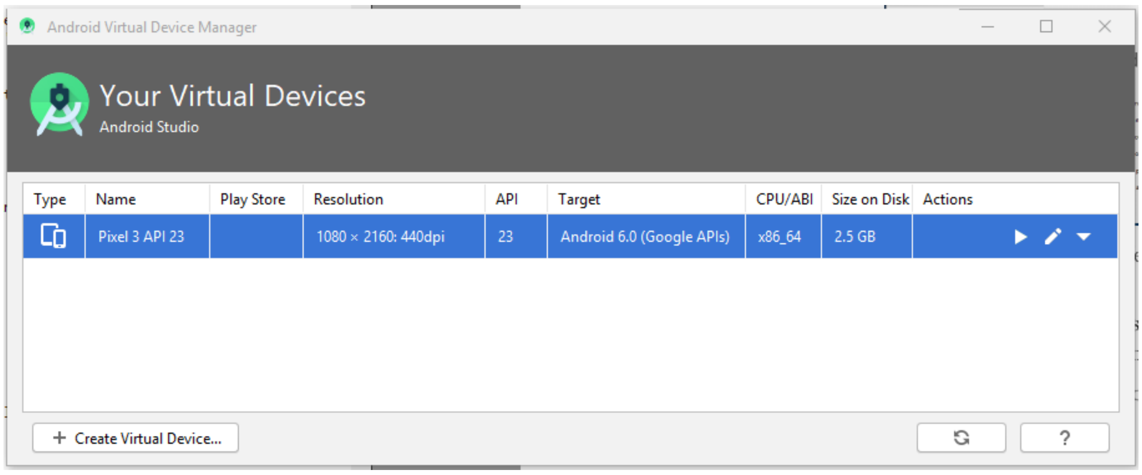
System Image
x86_64

Questions on API level?
See the API level distribution chart

Figure 11: Virtual Device: choosing system image.

2. Create two AVDs, one for a **Pixel 3** smartphone and another for a **Pixel C** tablet. For each one, start by clicking in the "Create Virtual Device" button. In the AVD hardware profile panel, see Figure 10, select the hardware profile for the desired AVD and then click the "Next" button. In the AVD system image panel, see Figure 11, select the Android platform version, hardware architecture (x86, ...) and the supported Google services (Google Play, ...), download it if necessary and then click the "Next" button. Leave the remaining parameters, for now, set to "default" and finish.

3. Start the Pixel 3 AVD. After you create the emulators, you can see them in the AVD Manager panel (see Figure 12). On that list click the “play” button on th **Pixel 3** smartphone emulator to launch it.



Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Pixel 3 API 23		1080 × 2160: 440dpi	23	Android 6.0 (Google APIs)	x86_64	2.5 GB	

Figure 12: AVD Manager: list of existing AVDs.

Note 1: The first launch of an AVD can take up to 10 minutes, depending on the computing power of the PC. If you stop an AVD during the launch process, the AVD may become corrupted.

Note 2: The AVD after being started should not be stopped/closed during development. If you make changes to the code, just do the “re-deploy” of the application in the AVD.

Note 3: During the creation of an emulator you can choose between the options: “Snapshot” or “Host GPU”. If you choose the first option, the emulator, when stopped, is like hibernated (saved a copy of its memory state), and a subsequent start will be fast. If you choose the second option, rendering is faster because the PC graphics card is used.

Note 4: When choosing the Android API version, when creating the emulator, you can select a version where the AVD image is based on an ARM CPU architecture or an Intel CPU architecture. An AVD that is based on an Intel system is much faster if it runs on Intel hardware, because in that case it is not necessary to convert the ARM CPU instructions to the Intel architecture of the PC.

3.5 Prepare your phone for your Apps

1. **Enable non-Play Store Apps instalation**. Go to **Settings** > **Security** and then select **Unknown sources**. Selecting this option will allow you to install apps outside of the Google Play store. From Android 8.0 on, we must go to **Settings** > **Apps & notifications** > **Advanced** > **Special app access** > **Install unknown apps** and then select **Allow from this source**.

2. **Enable “Developer options”**. To install and debug applications in your phone you must activate its **Developer options**. To activate it, in your phone, go to

Settings > **About phone** and tap **Build number** seven times.

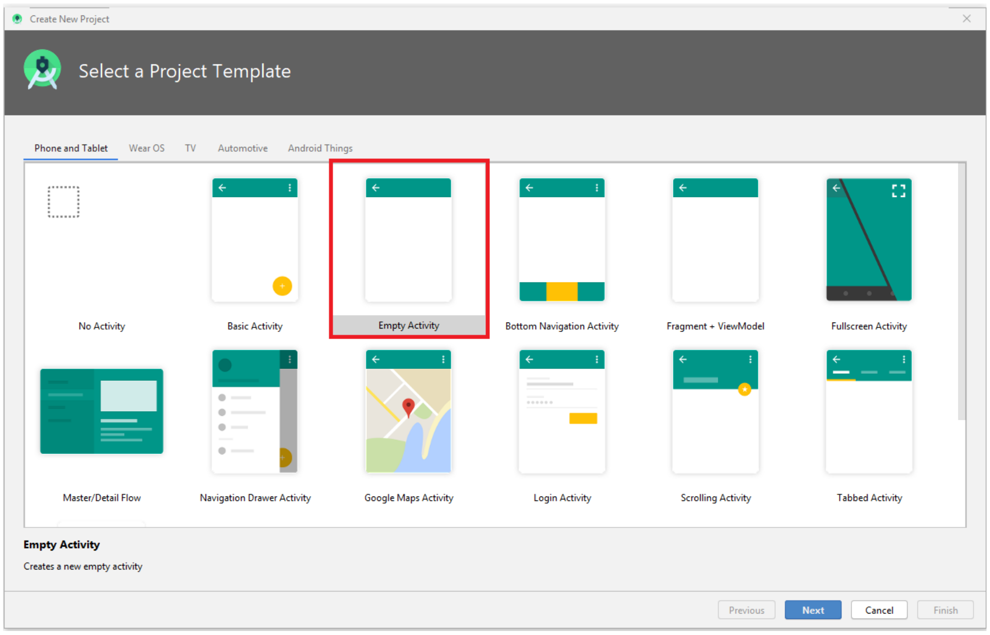
3. **Enable debugging**. App installation and debugging is done by **Android Debug Bridge (ADB)**. The ADB lets you to install and debug applications in the devices and provides you with a command line shell that you can use to send commands to the device. The ADB can connect with the device by an USB cable or by WiFi. After **Developer options** activation, you must enable debugging by Android Debug Bridge (ADB). To do it, go to **Settings** > **Developer options** and enable **USB debugging**.

See `https://developer.android.com/studio/command-line/adb`.

4 Laboratory Work

4.1 Create a "Hello Mobile World" app

1. Create an Android project selecting **File** > **New** > **New Project** to open a dialogue box with several options (see Figure 13). Choose the **Empty Activity** option and click **Next**. This option will create a basic application with one activity.

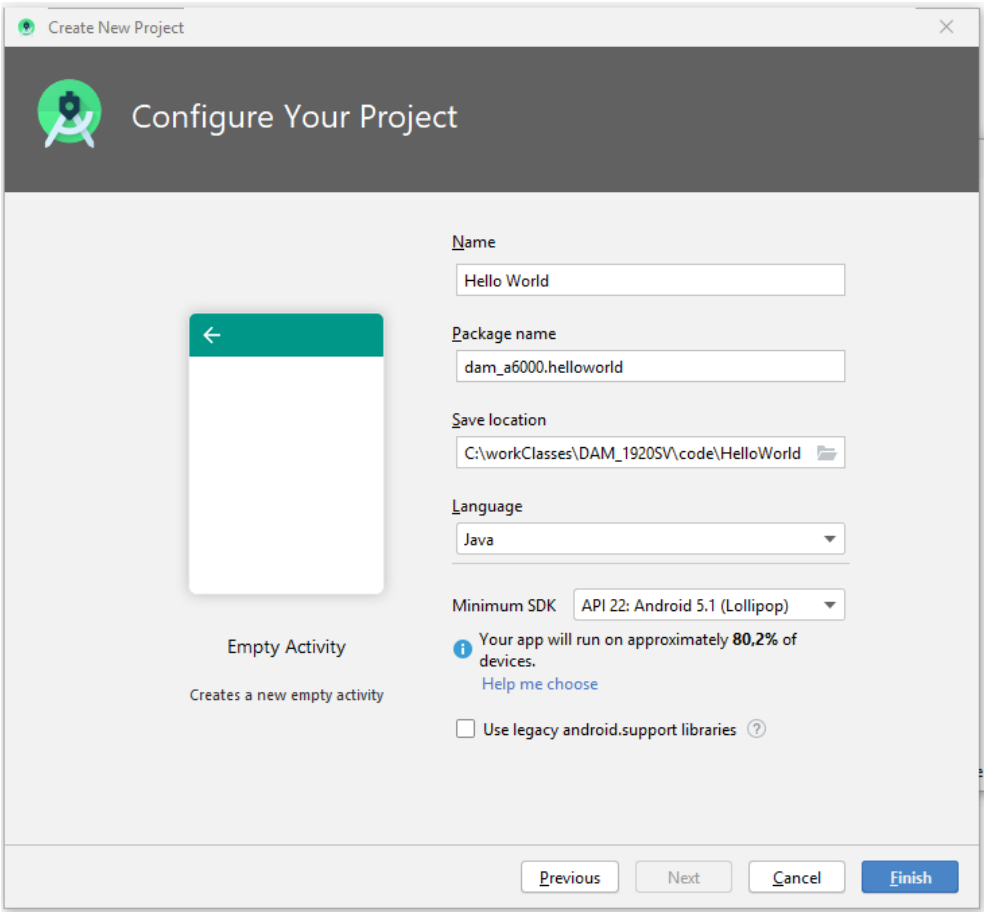


Empty Activity
Creates a new empty activity

Figure 13: Choose your project - Basic Activity

2. Type the name of the application: **Hello World**. Write the name of the package: "dam_a60000.helloworld" (see Figure 14) (a60000 should be replaced with your student number). **You should use that base package name for all your projects.** Set the path where the project should be saved, as: "dam-path\code\HelloWorld". Where dam-path should be the directory of DAM. Select Java language and the API 22: Android 5.1 (**Lolipop**) version for the Minimum API level. Click **Finish** to generate the project.

   Note 5: In the selection of Minimum API level the configuration wizard will show, for each API version, the estimated percentage of devices that supports the version



Name: Hello World
Package name: dam_a6000.helloworld
Save location: C:\workClasses\DAM_1920SV\code\HelloWorld
Language: Java
Minimum SDK: API 22: Android 5.1 (Lollipop)
Your app will run on approximately **80,2%** of devices.
Help me choose
☐ Use legacy android.support libraries

Figure 14: Configure you project.

and the new features introduced by the version. Inspect the new characteristics of each version.

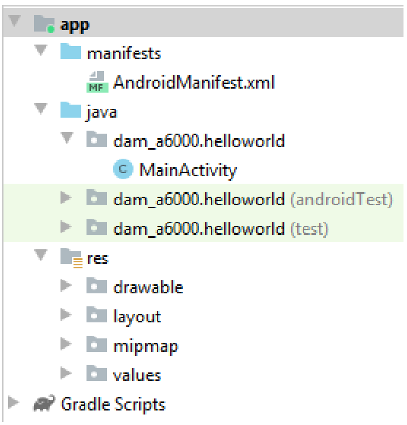


Figure 15: Main files.

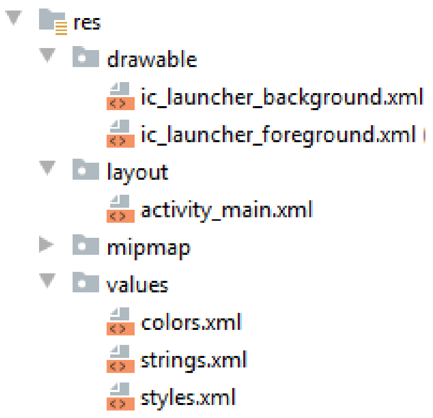


Figure 16: Resource files.

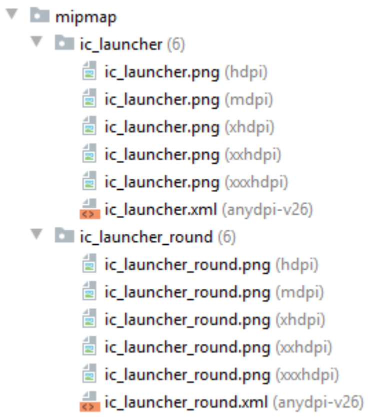

mipmap
ic_launcher (6)
ic_launcher.png (hdpi)
ic_launcher.png (mdpi)
ic_launcher.png (xhdpi)
ic_launcher.png (xxhdpi)
ic_launcher.png (xxxhdpi)
ic_launcher.xml (anydpi-v26)
ic_launcher_round (6)
ic_launcher_round.png (hdpi)
ic_launcher_round.png (mdpi)
ic_launcher_round.png (xhdpi)
ic_launcher_round.png (xxhdpi)
ic_launcher_round.png (xxxhdpi)
ic_launcher_round.xml (anydpi-v26)

Figure 17: Res. img. files.

3. Inspect generated code. As you can see in Figure 15 the Android Studio creates three sections for files: **manifest**, **java** and **res**.

   An app should contain an **manifest** file containing: app package name, components (activities, services, broadcast receivers, and content providers) and their properties, permissions needed and hardware and software features required.

   The **java** section contains Java files. It should contain Activity files and all other

Java files. The generated files are set with the defined package. We can see the app contains one Activity.

The **res** (resource) section, as we can see in Figure 16, contains the following resource directories: drawable, layout, mipmap and values. The contents of mipmap are shown in Figure 17.

Inspect the content of all files. View the .xml files in **Code**, **Split** and **Design** mode, by selecting them on the icons on top right side of the editor window.

4. Execute the application in the Pixel 3 AVD. Press **Run** > **Run "app"**, or by using Shift + F10 or by clicking the green play symbol in toolbar (on right of device drop-down box; right of "No Devices" in Figure 9). In the window choose your Pixel 3 AVD smartphone and click the "OK" button. The AVD will present the contents of the Figure 18. Check the created elements in the emulator screen.

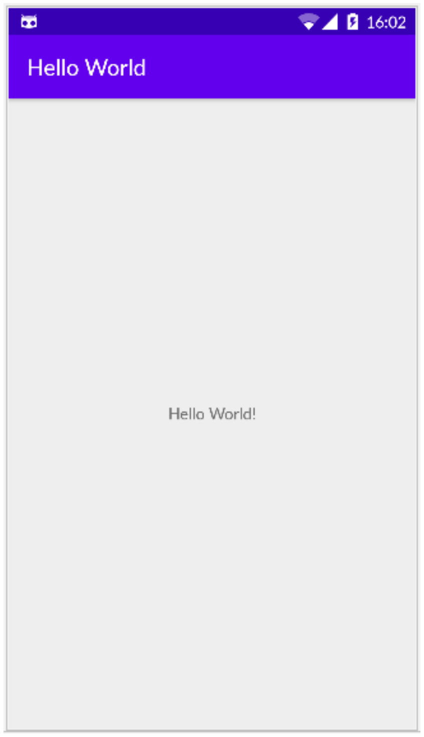


Figure 18: Hello World app.

Note 6: This procedure may take some time. If the execution blocks, try closing the AVD and restarting the execution.

5. Put **strings in strings.xml file**. String should be defined in string.xml file, to enable internationalization (we will see it latter). In strings.xml each string is identified by a name. To get the value of the string we must use "@string/strname (if named strname). Change the text in the TextView to strings.xml: in activity_main.xml select the existing string "Hello World!" and press ALT + ENTER; select "Extract string resource"; write hello_string in the name and press Ok. Check that the TextView android:text property now has: "@string/hello_string" and in strings.xml there is the new string hello_string declared. Run the app.

6. Change the string hello_string (in strings.xml) to “Hello Android World”. Run again.

7. Change the app name (in strings.xml) to “Hello World V1”. Run again.

8. Run and debug the application in your real device. Use ADB by **USB** cable and by **WiFi**.

   See `https://developer.android.com/studio/command-line/adb`.

4.2 Improve your application - Hello World v2

1. Change the app name (in strings.xml) to “Hello World V2”. Run the application (do the same for remaining points).

2. **Change the TextView** to the top and change some properties. You should have the window in Slip mode (select in top right corner), with the **Palette view** at the left, the design editor in the middle and the **Attributes view** at the right. In the design editor elect the TextView and in its bottom constraint give a left mouse click with th CTRL pressed to remove that constraint. To add a constraint of that type just stretch one of that points. Set its attributes: textColor, textSize and textStyle, to look as in 19.

3. **Add a second TextView** with “My First App” text (strings to strings.xml), by dragging one from the palette to the middle of the graphic editor. Connect the top graphical constraint to the bottom of the other TextView. Change the attributes of the TextViews to change: layout_width (set to 0dp to use all width), textSize, textAlignment and background to look as in 19.

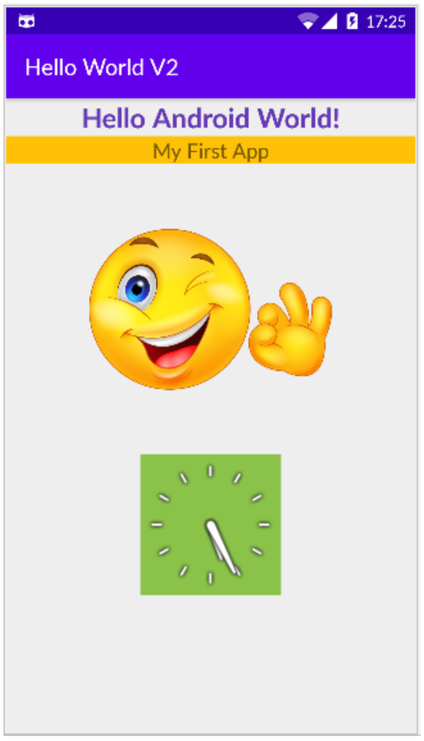


Figure 19: Hello World app v2.

4. **Add an image** to the screen, as shown in Figure 19. Download one small image from the web and add it to the "drawable" folder. The image name should only have lower-case letters. From the palette view drag an ImageView element to the layout of activity_main.xml. In "Pick a resource view" select the image. It will add to the ImageView: app:srcCompat="@drawable/smileygood" (if smile is the name of the image file). You will see an yellow warning because the image do not have an description. Add a content Description (alternative text) to the ImageView: android:contentDescription="Just smile" (strings to strings.xml).

5. **Add a watch** to the layout, as shown in Figure 19. The AnalogClock is deprecated since API 23 and for that reason it is not available in the components list of the Layouts Editor Interface. However, it can be included in the layout XML file related, using for instance in Listing 1.

```
<AnalogClock
   android:id="@+id/analogClock"
   android:layout_width="124dp"
   android:layout_height="124dp" />
```

Listing 1: AnalogClock.

The graphical editor gives and error, as we are using the deprecated AnalogClock. Nevertheless, we can ignore it and run the app.

6. **Add a Landscape layout variant**.

7. **Add an app icon**.

5 About Android Studio

5.1 Usefull shortcuts

1. **Rename**: this action enables to rename an artefact (a class, a method a variable, a package, ...). It can do it over any artefact occurrence, that it will rename all occurrences of the artefact. To activate this action go to: Menu → Refactor → Rename, or press **Shift + F6**.

2. **Format code**: this action enables to format the code automatically. To activate this action go to: Menu → Code → Reformat Code, or press **Ctrl + Alt + L**.

3. **Organize imports**: this action enables to automatically organize imports. This action will add imports needed (if artefacts are well written) or delete unused artefacts. To activate this action go to: Menu → Code → Optimize Imports, or press **Ctrl + Alt + O**.

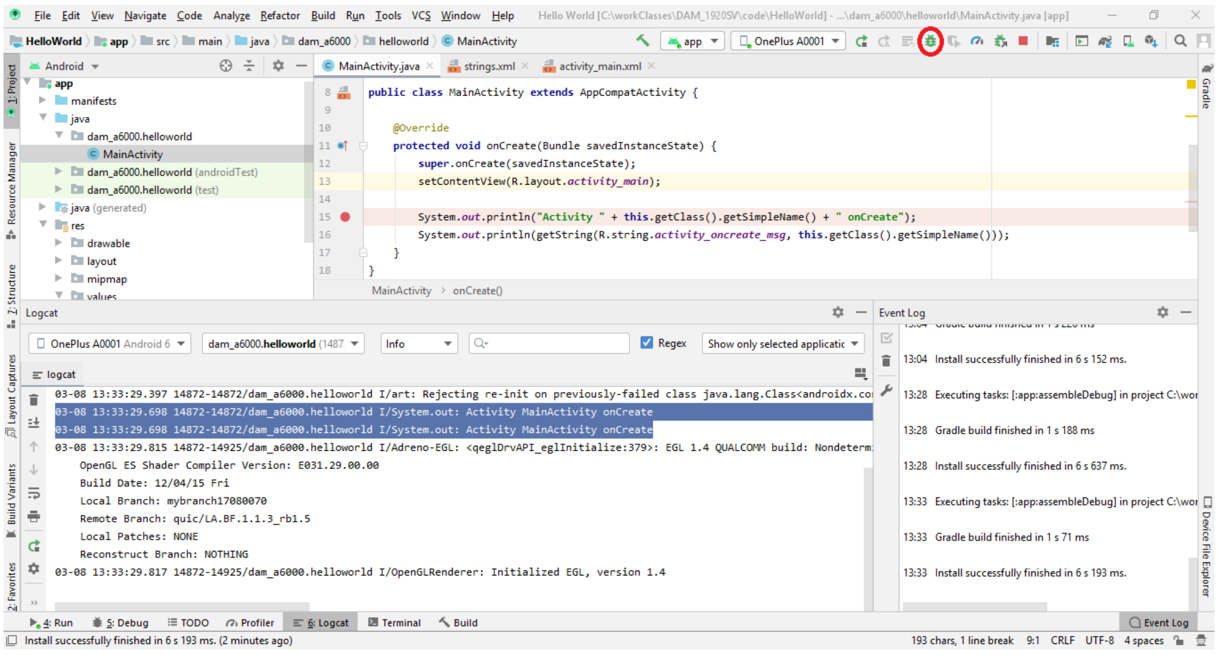
5.2 Observing app state

1. **Observing messages in LogCat**: the LogCat view enables the visualization of execution messages that the device sends, if the "No Filters" option is selected (in right drop-down box). If we select in that drop-down box the option of "Show only selected application" we filter and get only the messages related to the application

that appears in the second left drop-down box (the first one is to select the device and the third one is to select the level of visualization).

2. **Sending messages to LogCat**: to send an output to LogCat we can do it by sending contents to the standard console, that is: System.out. So, lets send a first message showing that we are inside the onCreate message of the Activity: in **MainActivity.java** file and at the end (but inside) of **onCreate** method insert: `System.out.println("Activity " + this.getClass().getSimpleName() + " onCreate");`. Execute the app and watch the LogCat contents: you will find that message. We will use the Log class, to send messages, in the next Tutorials.

3. **Setting a dynamic string**: now, we will first send that message to strings.xml file by giving a double click over the Activity string to select it and doing ALT + ENTER, select Extract string Resource, in the Extract Resource dialog box we should insert in Resource name: activity_oncreate_msg, and in Resource value: Activity %1$s onCreate. Strings with dynamic contents, like the one in onCreate, use the String.format() way, where, in our example, the %1$s says to use the first argument treated as a string in the place where %1$s is. Hence, we are now ready to complete the call to getString by completing it with the addition of a second argument with the value that we want to appear. This should result in: `getString(R.string.activity_oncreate_msg, this.getClass().getSimpleName())`. Run again the app to see the result. See the results also in Figure 20.



```java
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        System.out.println("Activity " + this.getClass().getSimpleName() + " onCreate");
        System.out.println(getString(R.string.activity_oncreate_msg, this.getClass().getSimpleName()));
    }
}
```

```
03-08 13:33:29.397 14872-14872/dam_a6000.helloworld I/art: Rejecting re-init on previously-failed class java.lang.Class<androidx.co
03-08 13:33:29.698 14872-14872/dam_a6000.helloworld I/System.out: Activity MainActivity onCreate
03-08 13:33:29.698 14872-14872/dam_a6000.helloworld I/System.out: Activity MainActivity onCreate
03-08 13:33:29.815 14872-14925/dam_a6000.helloworld I/Adreno-EGL: <qeglDrvAPI_eglInitialize:379>: EGL 1.4 QUALCOMM build: Nondeterm
    OpenGL ES Shader Compiler Version: E031.29.00.00
    Build Date: 12/04/15 Fri
    Local Branch: mybranch17080070
    Remote Branch: quic/LA.BF.1.1.3_rb1.5
    Local Patches: NONE
    Reconstruct Branch: NOTHING
03-08 13:33:29.817 14872-14925/dam_a6000.helloworld I/OpenGLRenderer: Initialized EGL, version 1.4
```

```
13:04 Install successfully finished in 6 s 152 ms.
13:28 Executing tasks: [:app:assembleDebug] in project C:\wor
13:28 Gradle build finished in 1 s 188 ms
13:28 Install successfully finished in 6 s 637 ms.
13:33 Executing tasks: [:app:assembleDebug] in project C:\wor
13:33 Gradle build finished in 1 s 71 ms
13:33 Install successfully finished in 6 s 193 ms.
```

Install successfully finished in 6 s 193 ms. (2 minutes ago)

Hello World [C:\workClasses\DAM_1920SV\code\HelloWorld] - ...\dam_a6000\helloworld\MainActivity.java [app]
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
HelloWorld › app › src › main › java › dam_a6000 › helloworld › MainActivity
app
OnePlus A0001
MainActivity.java
strings.xml
activity_main.xml
Android
manifests
java
dam_a6000.helloworld
MainActivity
dam_a6000.helloworld (androidTest)
dam_a6000.helloworld (test)
java (generated)
res
drawable
layout
mipmap
values
MainActivity › onCreate()
Logcat
OnePlus A0001 Android 6
dam_a6000.helloworld (1487
Info
Regex
Show only selected applicatic
logcat
Event Log
4: Run
5: Debug
TODO
Profiler
6: Logcat
Terminal
Build
193 chars, 1 line break 9:1 CRLF UTF-8 4 spaces
Gradle
Device File Explorer

Figure 20: AS with dynamic string and breakpoint.

4. **Capturing the screen**: the LogCat view has two icons that enables to do a **screen capture** (icon with a photo camera) and **screen record** (green icon with a play symbol) of the device screen.

5. **Executing in Debug mode**: first we have to place a breakpoint in the code to stop the debug execution. So, place a breakpoint in the System.out.println line

from previous point, by clicking in the left area/column of the code – this should insert a red circle (see red circle in line 15 of Figure 20). After that, **run the app in Debug mode** with: Menu → Run → Debug: 'app' or with **Shift + F9** or by clicking the **Bug icon** in the toolbar (surrounded by a red circle in the Figure 20). Check that the LogCat do not have the System.ou.println string. Execute the System.ou.println instruction with: Menu → Run → Step Over, or by pressing F8. The LogCat, now, contains the System.ou.println string. To stop a debug session click on the red square in the toolbar. Check the **Variables view** inside the Debug tab to see the value of existing variables.