

1.1: Install Android Studio and Run Hello World

Contents:

- [What you should already KNOW](#)
- [What you will NEED](#)
- [What you will LEARN](#)
- [What you will DO](#)
- [App overview](#)
- [Task 1: Install Android Studio](#)
- [Task 2: Create "Hello World" app](#)
- [Task 3: Explore the project structure and layout](#)
- [Task 4: Create a virtual device](#)
- [Task 5: Run your app on an emulator](#)
- [Task 6: Add log statements to your app](#)
- [Task 7: Explore the AndroidManifest.xml file](#)
- [Task 8: Explore the build.gradle file](#)
- [Task 9: Run your app on a device](#)
- [Coding challenge](#)
- [Summary](#)
- [Related concepts](#)
- [Learn more](#)

Welcome to the practical exercises. You will learn to:

- Install Android Studio, the Android development environment.
- Learn about the Android development process.
- Create and run your first Android Hello World app on an emulator and on a physical device.
- Add logging to your app for debugging purposes.

What you should already KNOW

For this practical you should be able to:

- Understand the general software development process for object-oriented applications using an IDE (Integrated Development Environment).
- Demonstrate that you have at least 1-3 years of experience in object-oriented programming, with some of it focused on the Java programming language. (These practicals will not explain object-oriented programming or the Java language.)

What you will NEED

For these practicals, you will need:

- A Mac, Windows, or Linux computer. See the bottom of the [Android Studio download page](#) for up-to-date [system requirements](#).
- Internet access or an alternative way of loading the latest Android Studio and Java installations onto your computer.

What you will LEARN

You will learn to:

- Install and use the Android IDE.

- Understand the development process for building Android apps.
- Create an Android project from a basic app template.

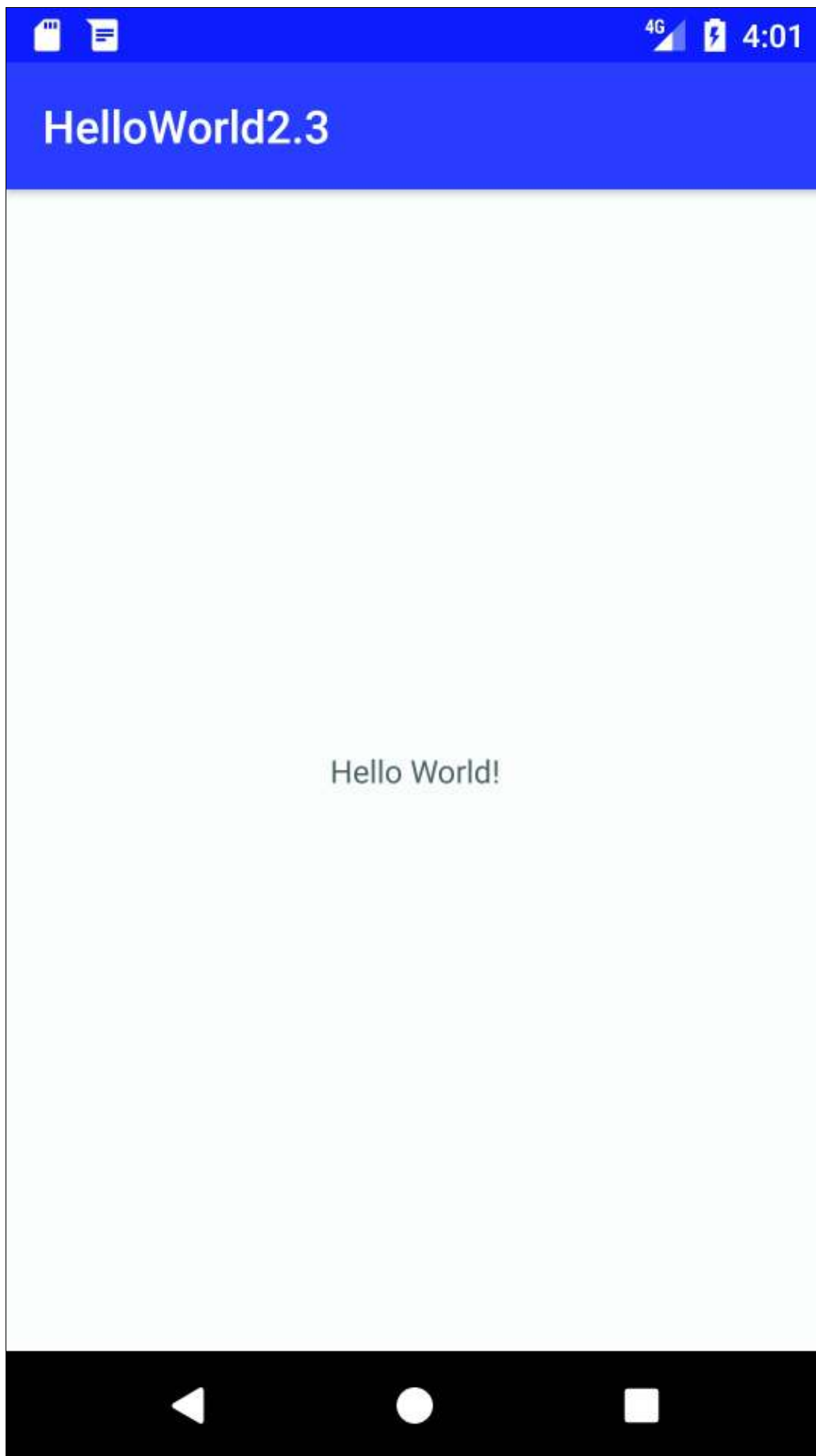
What you will DO

- Install the Android Studio development environment.
- Create a an emulator (virtual device) to run your app on your computer.
- Create and run the Hello World app on the virtual and physical devices.
- Explore the project layout.
- Generate and view log statements from your app.
- Explore the AndroidManifest.xml file.

App Overview

After you successfully install the Android Studio IDE, you will create, from a template, a new Android project for the 'Hello World' app. This simple app displays the string "Hello World" on the screen of the Android virtual or physical device.

Here's what the finished app will look like:



Task 1. Install Android Studio

Android Studio is Google's IDE for Android apps. Android Studio gives you an advanced code editor and a set of app templates. In addition, it contains tools for development, debugging, testing, and performance that make it faster and easier to develop apps. You can test your apps with a large range of preconfigured emulators or on your own mobile device, and build production APKs for publication.

Note: Android Studio is continually being improved. For the latest information on system requirements and installation instructions, refer to the documentation at developer.android.com.

To get up and running with Android Studio:

- You may need to install the Java Development Kit - Java 7 or better.
- Install Android Studio

Android Studio is available for Windows, Mac, and Linux computers. The installation is similar for all platforms. Any differences will be noted in the sections below.

1.1. Installing the Java Development Kit

1. On your computer, open a terminal window.
2. Type `java -version`

The output includes a line:

```
Java(TM) SE Runtime Environment (build1.X.0_05-b13)
```

X is the version number to look at.

- If this is 7 or greater, you can move on to installing Android Studio.
- If you see a Java SE version is below 7 or if Java is not installed, you need to install the latest version of the Java SE development kit before installing Android Studio.

To download the Java Standard Edition (JSE) Development Kit (JDK):

1. Go to the [Oracle Java SE downloads page](https://www.oracle.com/technetwork/java/javase-downloads-1344644.html).
2. Click the Java SE Downloads icon to open the [Java SE Development Kit 8 Downloads page](https://www.oracle.com/technetwork/java/javase-downloads-1344644.html).
3. In the box for the latest Java SE Development kit, you need to accept the License Agreement in order to proceed. Then download the version appropriate for the computer you are developing on.

Important: Do not go to the demos and samples (the menus look very similar, so make sure to read the heading at the top).

4. Install the development kit. Once the installation of the JDK is completed — it should only take a few minutes — you can confirm it's correct by checking the Java version from the command line.
5. Open a terminal window and enter Type `java -version` again to verify that installation has been successful.
6. Set the `JAVA_HOME` environment variable to the installation directory of the JDK.

Windows:

1. Set `JAVA_HOME` to the installation location.
2. Start > Control Panel > System > Advanced System Settings > Environment Variables System Variables > New
 - Variable name: `JAVA_HOME`
 - Variable value: `C:\Program Files\Java\jdk1.7.0_80` (or whatever version your installation is!)
3. If the variable already exists, update it to this version of the JDK.
4. Verify your `JAVA_HOME` variable from a `cmd.exe` terminal: `echo %JAVA_HOME%`

See also: https://docs.oracle.com/cd/E19182-01/820-7851/inst_cli_jdk_javahome_t/

Mac:

1. Open Terminal.
2. Confirm you have JDK by typing "which java".
3. Check that you have the needed version of Java, by typing "java -version".
4. Set JAVA_HOME using this command in Terminal: `export JAVA_HOME=`which java``
5. enter `echo $JAVA_HOME` to confirm the path.

Linux:

See: https://docs.oracle.com/cd/E19182-01/820-7851/inst_cli_jdk_javahome_t/

Important: Don't install Android Studio until after the Java JDK is installed. Without a working copy of Java, the rest of the process will not work. If you can't get the download to work, look for error messages, and search online to find a solution.

Basic Troubleshooting:

- There is no UI, Control Panel, or Startup icon associated with the JDK.
- Verify that you have correctly installed the JDK by going to the directory where you installed it. To identify where the JDK is, look at your PATH variable and/or search your computer for the "jdk" directory or the "java" or "javac" executable.

1.2. Installing Android Studio

1. Navigate to the [Android developers site](#) and follow the instructions to download and [install Android Studio](#).
 - Accept the default configurations for all steps.
 - Make sure that all components are selected for installation.
2. After finishing the install, the Setup Wizard will download and install some additional components. Be patient, this might take some time depending on your Internet speed, and some of the steps may seem redundant.
3. When the download completes, Android Studio will start, and you are ready to create your first project.
Troubleshooting: If you run into problems with your installation, check the latest documentation, programming forums, or get help from your instructors.

Task 2: Create "Hello World" app

In this task, you will implement the "Hello World" app to verify that Android studio is correctly installed and learn the basics of developing with Android Studio.

2.1 Create the "Hello World" app

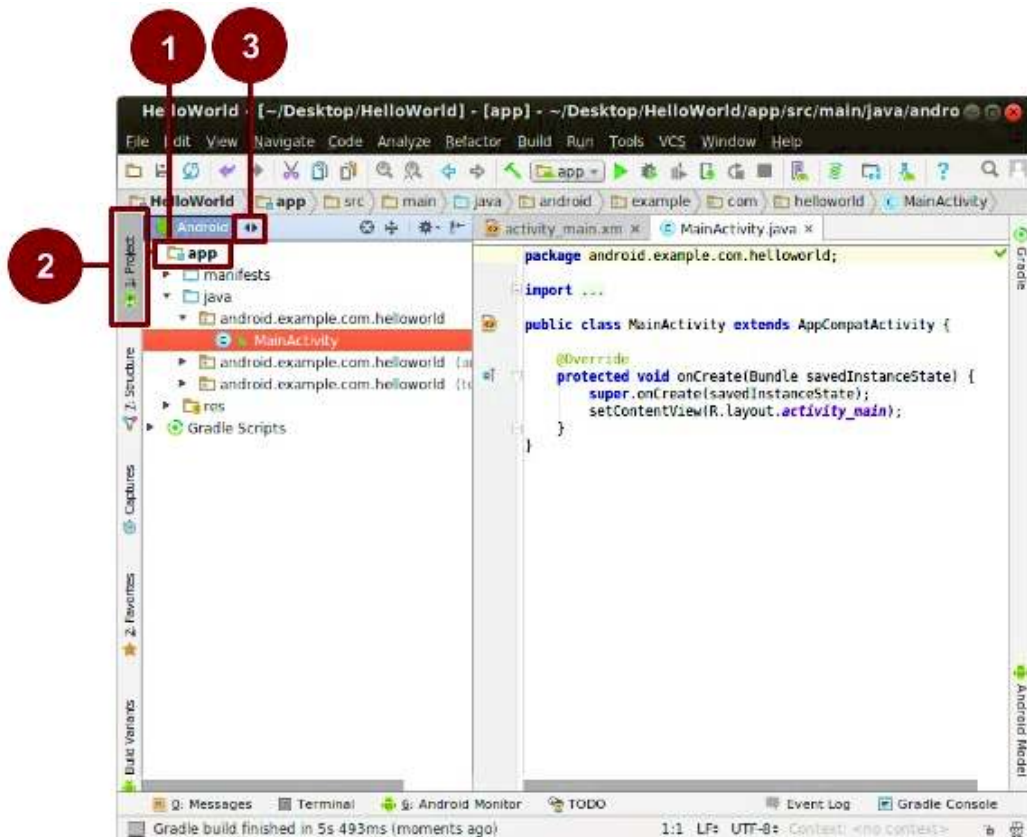
1. Launch Android Studio if it is not already opened.
2. In the main **Welcome to Android Studio** window, click "Start a new Android Studio project".
3. In the **New Project** window, give your application an **Application Name**, such as "Hello World".
4. Verify the Project location, or choose a different directory for storing your project.
5. Choose a unique **Company Domain**.
 - Apps published to the Google Play Store must have a unique package name. Since domains are unique, prepending your app's name with your or your company's domain name is going to result in a unique package name.
 - If you are not planning to publish your app, you can accept the default example domain. Be aware that changing the package name of your app later is extra work.
6. Verify that the default **Project location** is where you want to store your Hello World app and other Android Studio projects, or change it to your preferred directory. Click Next.
7. On the **Target Android Devices** screen, "Phone and Tablet" should be selected. And you should ensure that API 15: Android 4.0.3 IceCreamSandwich is set as the **Minimum SDK**. (Fix this if necessary.)
 - At the writing of this book, choosing this API level makes your "Hello World" app compatible with 97% of Android devices active on the Google Play Store.
 - These are the settings used by the examples in this book.
8. Click **Next**.

9. If your project requires additional components for your chosen target SDK, Android Studio will install them automatically. Click **Next**.
10. **Customize the Activity** window. Every app needs at least one activity. An activity represents a single screen with a user interface and Android Studio provides templates to help you get started. For the Hello World project, choose the simplest template (as of this writing, the "Empty Activity" project template is the simplest template) available.
11. It is a common practice to call your main activity MainActivity. This is not a requirement.
12. Make sure the **Generate Layout file** box is checked (if visible).
13. Make sure the **Backwards Compatibility (App Compat)** box is checked.
14. Leave the **Layout Name** as activity_main. It is customary to name layouts after the activity they belong to. Accept the defaults and click **Finish**.

After these steps, Android Studio:

- Creates a folder for your Android Studio Projects.
- Builds your project with [Gradle](#) (this may take a few moments). Android Studio uses Gradle as it's build system. See the [Configure your build](#) developer page for more information.
- Opens the code editor with your project.
- Displays a tip of the day.
 - Android Studio offers many keyboard shortcuts, and reading the tips is a great way to learn them over time.

The Android Studio window should look similar to the following diagram:



You can look at the hierarchy of the files for your app in multiple ways.

1. Click on the HelloWorld folder to expand the hierarchy of files (1),
2. Click on **Project** (2).
3. Click on the **Android** menu (3).
4. Explore the different view options for your project.

Note: This book uses the **Android** view of the project files, unless specified otherwise.

Task 3: Explore the project structure

In this practical, you will explore how the project files are organized in Android Studio.

These steps assume that your Hello World project starts out as shown in the diagram above.

3.1 Explore the project structure and layout

In the Project > Android view of your previous task, there are three top-level folders below your **app** folder: **manifests**, **java**, and **res**.

1. Expand the **manifests** folder.

This folder contains **AndroidManifest.xml**. This file describes all of the components of your Android app and is read by the Android run-time system when your program is executed.

2. Expand the **java** folder. All your Java language files are organized in this folder. The **java** folder contains three subfolders:

- **com.example.hello.helloworld (or the domain name you have specified)**: All the files for a package are in a folder named after the package. For your Hello World application, there is one package and it only contains MainActivity.java (the file extension may be omitted in the Project view).
- **com.example.hello.helloworld(androidTest)**: This folder is for your instrumented tests, and starts out with a skeleton test file.
- **com.example.hello.helloworld(test)**: This folder is for your unit tests and starts out with an automatically created skeleton unit test file.

3. Expand the **res** folder. This folder contains all the resources for your app, including images, layout files, strings, icons, and styling. It includes these subfolders:

- **drawable**. Store all your app's images in this folder.
- **layout**. Every activity has at least one layout file that describes the UI in XML. For Hello World, this folder contains activity_main.xml.
- **mipmap**. Store your launcher icons in this folder. There is a sub-folder for each supported screen density. Android uses the screen density, that is, the number of pixels per inch to determine the required image resolution. Android groups all actual screen densities into generalized densities, such as medium (mdpi), high (hdpi), or extra-extra-extra-high (xxxhdpi). The ic_launcher.png folder contains the default launcher icons for all the densities supported by your app.
- **values**. Instead of hardcoding values like strings, dimensions, and colors in your XML and Java files, it is best practice to define them in their respective values file. This makes it easier to change and be consistent across your app.

4. Expand the **values** subfolder within the res folder. It includes these subfolders:

- **colors.xml**. Shows the default colors for your chosen theme, and you can add your own colors or change them based on your app's requirements.
- **dimens.xml**. Store the sizes of views and objects for different resolutions.
- **strings.xml**. Create resources for all your strings. This makes it easy to translate them to other languages.
- **styles.xml**. All the styles for your app and theme go here. Styles help give your app a consistent look for all UI elements.

3.2 The Gradle build system

Android Studio uses [Gradle](#) as its build system. As you progress through these practicals, you will learn more about gradle and what you need to build and run your apps.

1. Expand the **Gradle Scripts** folder. This folder contains all the files needed by the build system.
2. Look for the **build.gradle(Module:app)** file. When you are adding app-specific dependencies, such as using additional libraries, they go into this file.

Task 4: Create a virtual device (emulator)

In this task, you will use the [Android Virtual Device \(AVD\) manager](#) to create a virtual device or emulator that simulates the configuration for a particular type of Android device.


Using the AVD Manager, you define the hardware characteristics of a device and its API level, and save it as a virtual device configuration.

When you start the Android emulator, it reads a specified configuration and creates an emulated device that behaves exactly like a physical version of that device, but it resides on your computer.

Why: With virtual devices, you can test your apps on different devices (tablets, phones) with different API levels to make sure it looks good and works for most users. You do not need to depend on having a physical device available for app development.

4.1 Create a virtual device

In order to run an emulator on your computer, you have to create a configuration that describes the virtual device.

1. In Android Studio, select **Tools > Android > AVD Manager**, or click the AVD Manager icon  in the toolbar.
2. Click the **+Create Virtual Device...** (If you have created a virtual device before, the window shows all of your existing devices and the button is at the bottom.)

The Select Hardware screen appears showing a list of preconfigured hardware devices. For each device, the table shows its diagonal display size (Size), screen resolution in pixels (Resolution), and pixel density (Density).

For the Nexus 5 device, the pixel density is xxhdpi, which means your app uses the launcher icons in the xxhdpi folder of the mipmap folder. Likewise, your app will use layouts and drawables from folders defined for that density as well.

3. Choose the Nexus 5 hardware device and click **Next**.
4. On the **System Image** screen, from the **Recommended** tab, choose which version of the Android system to run on the virtual device. You can select the latest system image.


There are many more versions available than shown in the **Recommended** tab. Look at the **x86 Images** and **Other Images** tabs to see them.

5. If a **Download** link is visible next to a system image version, it is not installed yet, and you need to download it. If necessary, click the link to start the download, and click **Finish** when it's done.
6. On **System Image** screen, choose a system image and click **Next**.
7. Verify your configuration, and click **Finish**. (If the **Your Android Devices** AVD Manager window stays open, you can go ahead and close it.)

Task 5. Run your app on an emulator

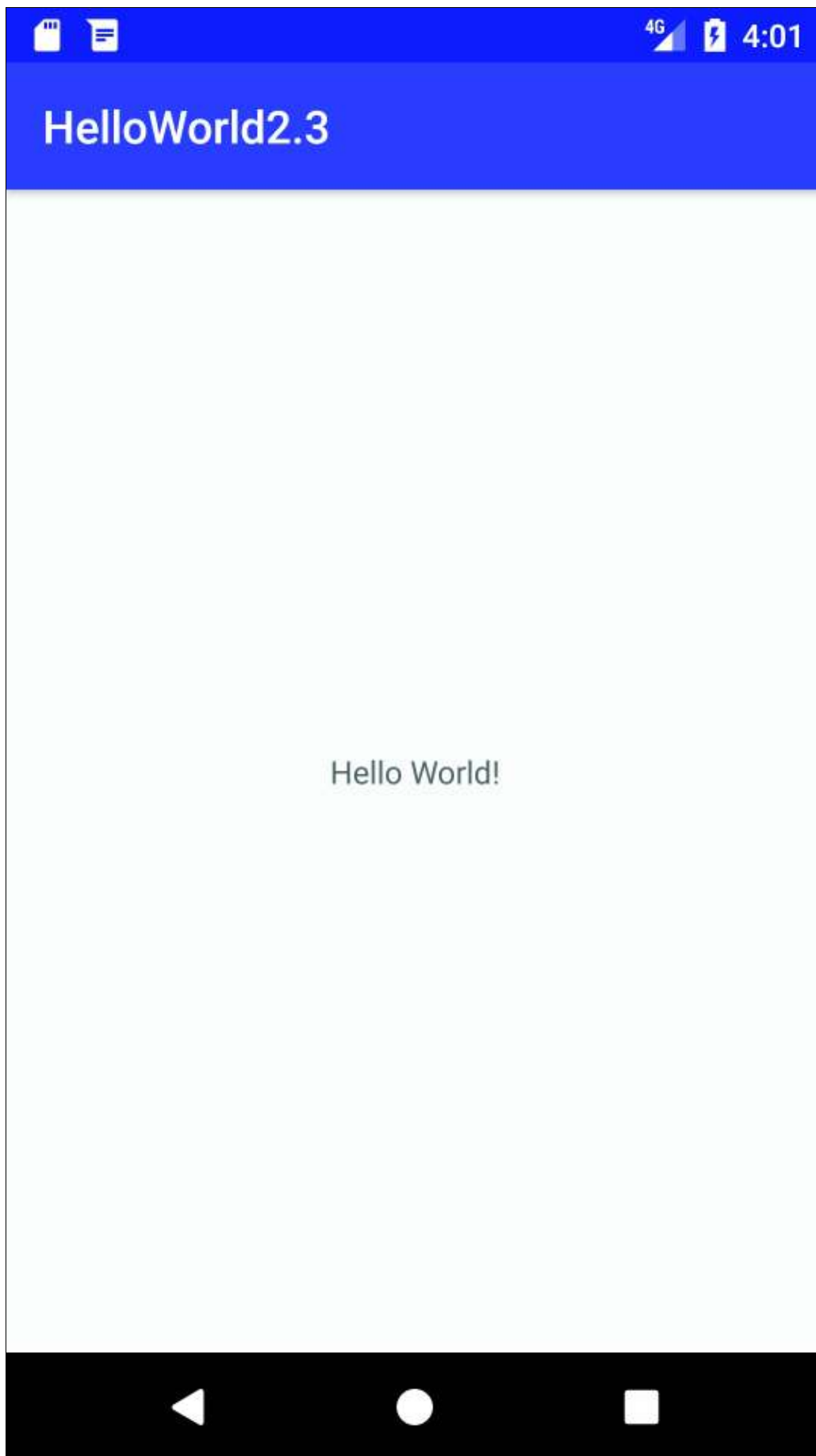
In this task, you will finally run your Hello World app.

5.1 Run your app on an emulator

1. In Android Studio, select **Run > Run app** or click the **Run icon**  in the toolbar.
2. In the **Select Deployment Target** window, under **Available Emulators**, select **Nexus 5 API 23** and click **OK**.

The emulator starts and boots just like a physical device. Depending on the speed of your computer, this may take a while. Your app builds, and once the emulator is ready, Android Studio will upload the app to the emulator and run it.

You should see the Hello World app as shown in the following screenshot.



Note: When testing on an emulator, it is a good practice to start it up once, at the very beginning of your session. You should not close the emulator until you are done testing your app, so that your app doesn't have to go through the boot process again.

Coding challenge

Note: All coding challenges are optional, and are not requirements for subsequent practicals.

Challenge: You can fully customize your virtual devices.

- Study the [AVD Manager documentation](#).
- Create one or several custom virtual devices.

You may notice that not all combinations of devices and system versions work when you run your app. This is because not all system images can run on all hardware devices.

Task 6. Add log statements to your app

In this practical, you will add log statements to your app, which are displayed in the logging window of the Android Monitor.

Why: Log messages are a powerful debugging tool that you can use to check on values, execution paths, and report exceptions.

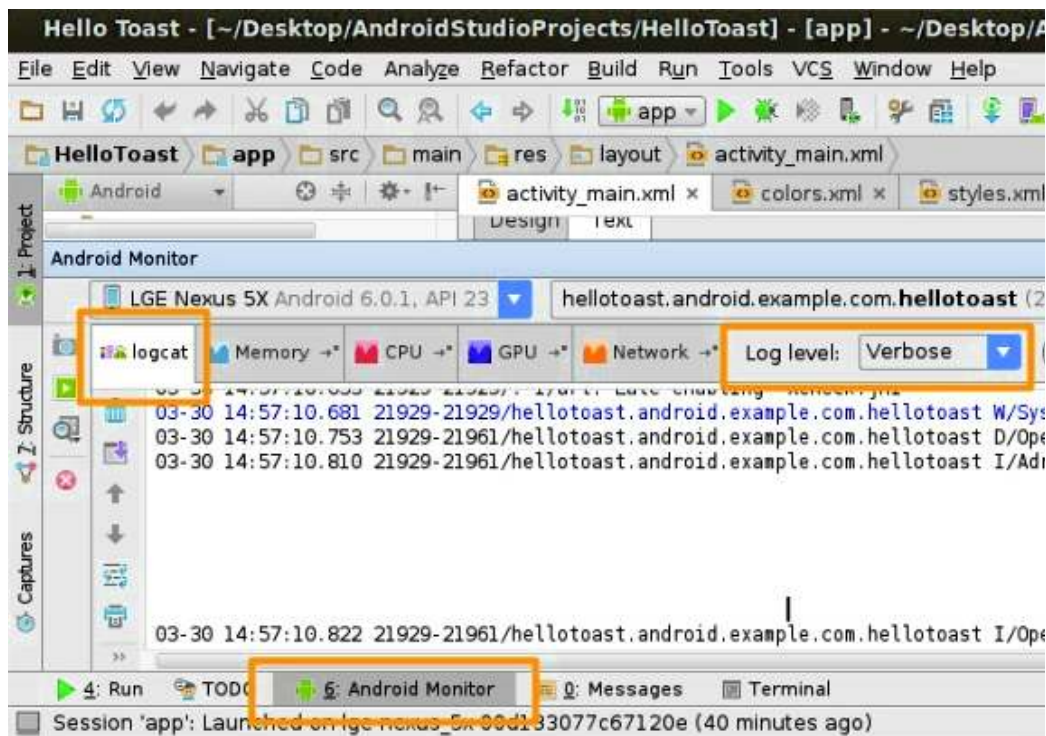
The **Android Monitor** displays information about your app.

1. Click the **Android Monitor** button at the bottom of Android Studio to open the Android Monitor.

By default, this opens to the **logcat** tab, which displays information about your app as it is running. If you add log statements to your app, they are printed here as well.

You can also monitor the Memory, CPU, GPU, and Network performance of your app from the other tabs of the Android Monitor. This can be helpful for debugging and performance tuning your code.

2. The default log level is **Verbose**. In the drop-down menu, change the log level to **Debug**.



Log statements that you add to your app code print a message specified by you in the logcat tab of the Android Monitor. For example:

```
Log.d("MainActivity", "Hello World");
```

The parts of the message are:

- Log – The [Log class](#). API for sending log messages.
- d – The Log level. Used to filter log message display in logcat. "d" is for debug. Other log levels are "e" for error, "w" for warning, and "i" for info.
- "MainActivity" – The first argument is a tag which can be used to filter messages in logcat. This is commonly the name of the activity from which the message originates. However, you can make this anything that is useful to you for debugging.

By convention, log tags are defined as constants:

```
private static final String LOG_TAG = MainActivity.class.getSimpleName();
```

- "Hello world" – The second argument is the actual message.

6.1 Add log statements to your app

1. Open your Hello World app in Android studio, and open MainActivity file.
2. **File > Settings > Editor > General > Auto Import** (Mac: **Android Studio > Preferences > Editor > General > Auto Import**). Select all check boxes and set **Insert imports on paste** to **All**. Unambiguous imports are now added automatically to your files. Note the "add unambiguous imports on the fly" option is important for some Android features such as NumberFormat. If not checked, NumberFormat shows an error. Click on 'Apply' followed by clicking on the 'Ok' button.
3. In the onCreate method, add the following log statement:

```
Log.d("MainActivity", "Hello World");
```

4. If the Android Monitor is not already open, click the Android Monitor tab at the bottom of Android Studio to open it. (See screenshot.)
5. Make sure that the Log level in the Android Monitor logcat is set to Debug or Verbose (default).
6. Run your app.

Solution Code:

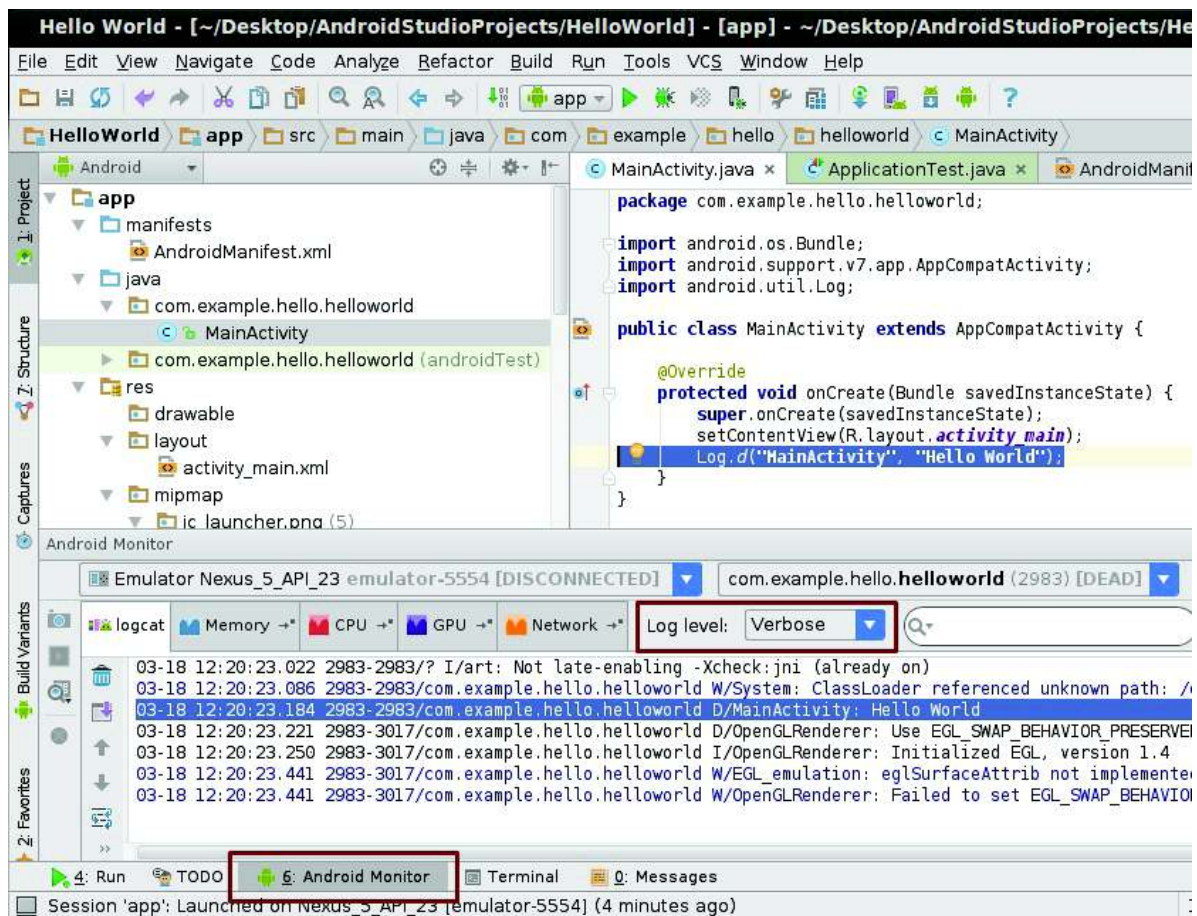
```
package com.example.hello.helloworld;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d("MainActivity", "Hello World");
    }
}
```

Output Log Message

```
03-18 12:20:23.184 2983-2983/com.example.hello.helloworld D/MainActivity: Hello World
```



Coding challenge

Note: All coding challenges are optional and are not a prerequisite for the next chapter.

Challenge: A common use of the Log class is to log Java exceptions when they occur in your program. There are some useful methods in the Log class that you can use for this purpose. Use the [Log class documentation](#) to find out what methods you can use to include an exception with a log message. Then, write code in the MainActivity.java file to trigger and log an exception.

Task 7: Explore the AndroidManifest.xml file

Every app includes an Android Manifest file (`AndroidManifest.xml`).The manifest file contains essential information about your app and presents this information to the Android runtime system. Android must have this information before it can run any of your app's code.

In this practical you will find and read the AndroidManifest.xml file for the Hello World app.

Why: As your apps add more functionality and the user experience becomes more engaging and interactive, the AndroidManifest.xml file contains more and more information. In later lessons, you will modify this file to add features and feature permissions.

7.1 Explore the AndroidManifest.xml file

1. Open your Hello World app in Android studio, and in the **manifests** folder, open **AndroidManifest.xml**.
2. Read the file and consider what each line of code indicates. The code below is annotated to give you some hints.

Annotated code:

```

<!-- XML version and character encoding -->
<?xml version="1.0" encoding="utf-8"?>
<!-- Required starting tag for the manifest -->
<manifest
  <!-- Defines the android namespace. Do not change. -->
  xmlns:android="http://schemas.android.com/apk/res/android"
  <!-- Unique package name of your app. Do not change once app is
    published. -->
    package="com.example.hello.helloworld">
  <!-- Required application tag -->
    <application
      <!-- Allow the application to be backed up and restored. -->
      android:allowBackup="true"
      <!-- Icon for the application as a whole,
        and default icon for application components. -->
      android:icon="@mipmap/ic_launcher"
      <!-- User-readable label for the application as a whole,
        and default icon for application components. Notice that Android
        Studio first shows the actual label "Hello World".
        Click on it, and you will see that the code actually refers to a string
        resource. Ctrl-click @string/app_name to see where the resource is
        specified. This will be covered in a later practical . -->
      android:label="@string/app_name"
      <!-- Whether the app is willing to support right-to-left layouts.-->
      android:supportsRtl="true"
      <!-- Default theme for styling all activities. -->
      android:theme="@style/AppTheme">
      <!-- Declares an activity. One is required.
        All activities must be declared,
        otherwise the system cannot see and run them. -->
      <activity
        <!-- Name of the class that implements the activity;
          subclass of Activity. -->
          android:name=".MainActivity">
        <!-- Specifies the intents that this activity can respond to.-->
        <intent-filter>
          <!-- The action and category together determine what
            happens when the activity is launched. -->
          <!-- Start activity as the main entry point.
            Does not receive data. -->
          <action android:name="android.intent.action.MAIN" />
          <!-- Start this activity as a top-level activity in
            the launcher . -->
          <category android:name="android.intent.category.LAUNCHER" />
        <!-- Closing tags -->
        </intent-filter>
      </activity>
    </application>
  </manifest>

```

Coding challenge

Note: All coding challenges are optional.

Challenge: There are many other elements that can be set in the Android Manifest. Explore the [Android Manifest documentation](#) and learn about additional elements in the Android Manifest.

Task 8. Explore the build.gradle file

Android Studio uses a build system called Gradle. Gradle does incremental builds, which allows for shorter edit-test cycles.

To learn more about Gradle, see:

- [Gradle site](#)
- [Configure your build](#) developer documentation

- Search the internet for "gradle tutorial".

In this task, you will explore the `build.gradle` file.

Why: When you add new libraries to your Android project, you may also have to update your **build.gradle** file. It's useful to know where it is and its basic structure.

8.1 Explore the build.gradle(Module app) file

1. In your project hierarchy, find **Gradle Scripts** and expand it. There several build.gradle files. One with directives for your whole project, and one for each app module. The module for your app is called "app". In the Project view, it is represented by the **app** folder at the top-level of the Project view.
2. Open **build.gradle (Module:app)**.
3. Read the file and learn what each line of code indicates.

Solution:

```
// Add Android-specific build tasks
apply plugin: 'com.android.application'
// Configure Android specific build options.
android {
    // Specify the target SDK version for the build.
    compileSdkVersion 23
    // The version of the build tools to use.
    buildToolsVersion "23.0.2"
    // Core settings and entries. Overrides manifest settings!
    defaultConfig {
        applicationId "com.example.hello.helloworld"
        minSdkVersion 15
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }
    // Controls how app is built and packaged.
    buildTypes {
        // Another common option is debug, which is not signed by default.
        release {
            // Code shrinker. Turn this on for production along with
            // shrinkResources.
            minifyEnabled false
            // Use ProGuard, a Java optimizer.
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}
// This is the part you are most likely to change as you start using
// other libraries.
dependencies {
    // Local binary dependency. Include any JAR file inside app/libs.
    compile fileTree(dir: 'libs', include: ['*.jar'])
    // Configuration for unit tests.
    testCompile 'junit:junit:4.12'
    // Remote binary dependency. Specify Maven coordinates of the Support
    // Library needed. Use the SDK Manager to download and install such
    // packages.
    compile 'com.android.support:appcompat-v7:23.2.1'
}
```

- For a deeper look into Gradle check out the [Build System Overview](#) and [Configuring Gradle Builds](#) documentation.
- There are tools to help you [shrink your code](#), remove unnecessary libraries/resource and even obfuscate your program to prevent unwanted reverse-engineering.
- Android Studio itself provides some useful features. Learn more about a valuable open-source tool called [ProGuard](#).

Task 9. [Optional] Run your app on a device

In this final task, you will run your app on a physical mobile device such as a phone or tablet.

Why: Your users will run your app on physical devices. You should always test your apps on both virtual and physical devices.

What you need:

- An Android device such as a phone or tablet.
- A data cable to connect your Android device to your computer via the USB port.
- If you are using a Linux or Windows OS, you may need to perform additional steps to run on a hardware device. Check the [Using Hardware Devices](#) documentation. On Windows, you may need to install the appropriate USB driver for your device. See [OEM USB Drivers](#).


9.1 [Optional] Run your app on a device

To let Android Studio communicate with your device, you must turn on USB Debugging on your Android device. This is enabled in the Developer options settings of your device. Note this is not the same as rooting your device.

On Android 4.2 and higher, the Developer options screen is hidden by default. To show Developer options and enable USB Debugging:

1. On your device, open **Settings > About phone** and tap **Build number** seven times.
2. Return to the previous screen (**Settings**). **Developer options** appears at the bottom of the list. Click **Developer options**.
3. Choose **USB Debugging**.

Now you can connect your device and run the app from Android Studio.

1. Connect your device to your development machine with a USB cable.
2. In Android Studio, at the bottom of the window, click the Android Monitor tab. You should see your device listed in the top-left drop-down menu.
3. Click the **Run** button  in the toolbar. The **Select Deployment Target** window opens with the list of available emulators and connected devices.
4. Select your device, and click **OK**.

Android Studio should install and runs the app on your device.

Troubleshooting

If your Android Studio does not recognize your device, try the following:

- Unplug and replug your device.
- Restart Android Studio.
- If your computer still does not find the device or declares it "unauthorized":
 1. Unplug the device.
 2. On the device, open Settings->**Developer Options**.
 3. Tap **Revoke USB Debugging authorizations**.
 4. Reconnect the device to your computer.
 5. When prompted, grant authorizations.
- You may need to install the appropriate USB driver for your device. See the [Using Hardware Devices documentation](#).
- Check the latest documentation, programming forums, or get help from your instructors.

Coding challenge

Note: All coding challenges are optional.

Challenge: Now that you are set up and familiar with the basic development workflow, do the following:

1. Create a new project in Android Studio.
2. Change the greeting to "Happy Birthday to " and someone with a recent birthday.
3. Change the background of the app using a birthday-themed image.
4. Take a screenshot of your finished app and email it to someone whose birthday you forgot.

Summary

In this chapter, you learned to:

- Install Android Studio
 - Obtain a basic understanding of the development workflow once you have launched in Android Studio.
 - Have basic comprehension of the structure of an Android app in the build environment.
 - Have a basic understanding of the Android Manifest, and what it is used for.
 - Add log statements to the code that give you a basic tool for debugging.
- Deploy the Hello World app on the Android emulator and [optionally] on a mobile device.

Related concepts

The related concept documentation is in [Android Developer Fundamentals: Concepts](#).

- [Create Your First Android App](#)

Learn more

- [Android Studio download page](#)
- [How do I install Java?](#)
- [Android Studio documentation](#)
- [Supporting Multiple Screens](#)
- [Gradle Wikipedia page](#)
- [Reading and Writing Logs](#)