The Ultimate Beginner's Guide to Android Development: Creating "Hello World" with Kotlin

Language: English

Target length: ~10 pages (print-friendly)

Table of Contents

---

1. Introduction

Welcome! This guide is written for absolute beginners who want to build a simple Android app that shows the text **"Hello World"** using **Kotlin** — the modern, officially supported language for Android development. Every line of code will be shown and explained so you understand what the project contains and why.

By the end of this guide you will know how to:

* Install Android Studio and configure the environment.
* Create a new Android project with Kotlin.
* Understand the essential files in the project.
* Write and explain the XML layout and Kotlin activity code.
* Run your app on the emulator or a real device.

This guide assumes you have a basic understanding of programming concepts (variables, functions, classes). No prior Android experience required.

---

2. What you need (tools & system requirements)

1. **Android Studio** (latest stable version). Download from the official Android developers website.

2. **Java Development Kit (JDK)** — Java 11 or the version recommended by Android Studio. Usually bundled with Android Studio.

3. A computer with enough resources: at least 8 GB RAM recommended for a smooth emulator experience (4 GB minimum but slower).

4. Optional: an Android device and a USB cable for testing on real hardware.

Installation tips:

* Choose the **Android Studio** installer for your OS (Windows/macOS/Linux).

* During installation, allow Android Studio to download the Android SDK, SDK tools, and an emulator image (Android Virtual Device, AVD).

---

3. Creating a new Android project in Android Studio

1. Open Android Studio.

2. Click **New Project**.

3. Choose the **Empty Activity** template and click **Next**.

4. Configure your project:

   * **Name:** HelloWorldKotlin

   * **Package name:** com.example.helloworldkotlin (you can change domain and project name)

   * **Save location:** choose a folder on your computer

   * **Language:** Kotlin

* **Minimum SDK:** API 21 (Android 5.0 Lollipop) is a common choice; it covers most devices. Choose according to your needs.

5. Click **Finish** and wait for Gradle to sync and the project to build.

Android Studio will create a default project. You will see a structure in the Project window. We'll explain the important files next.

---

4. Project structure explained (essential files)

Important files/folders you'll work with:

* `app/src/main/AndroidManifest.xml` — app metadata, declares the main activity and permissions.

* `app/src/main/java/.../MainActivity.kt` — Kotlin source for your main activity.

* `app/src/main/res/layout/activity_main.xml` — layout XML that defines the UI for `MainActivity`.

* `app/build.gradle` — module-level Gradle build file (dependencies, compile SDK version).

* `build.gradle` — project-level Gradle build file.

* `gradle.properties`, `settings.gradle` — gradle settings.

Gradle will build your app and package it into an APK (or AAB) for installation on devices.

---

5. The layout file: `activity_main.xml`

Location: `app/src/main/res/layout/activity_main.xml`

This XML file describes the user interface for the activity. For a Hello World app we'll use a simple `ConstraintLayout` with a centered `TextView`.

Full file (complete code)

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textHello"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:text="Hello World!"
        android:textSize="24sp"
        android:layout_margin="16dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Line-by-line explanation

* `<?xml version="1.0" encoding="utf-8"?>` — XML declaration.
* `<androidx.constraintlayout.widget.ConstraintLayout ...>` — root layout. `ConstraintLayout` lets you position child views using constraints.
* `xmlns:android` / `xmlns:app` / `xmlns:tools` — XML namespaces used for attributes.
* `android:layout_width="match_parent"` — the layout fills full width of the screen.
* `android:layout_height="match_parent"` — the layout fills full height of the screen.
* `tools:context=".MainActivity"` — hint for the layout preview in Android Studio; not used at runtime.

Inside the layout, we declare a `TextView`:

* `android:id="@+id/textHello"` — unique identifier for the view; used to find it from code.

* `android:layout_width="wrap_content"` — width adjusts to content width.

* `android:layout_height="wrap_content"` — height adjusts to content height.

* `android:text="Hello World!"` — the text shown on screen.

* `android:textSize="24sp"` — font size in scaled pixels (sp) which respects user font settings.

* `android:layout_margin="16dp"` — margin around the view in density-independent pixels (dp).

* `app:layout_constraint...` — four constraints pin the `TextView`'s start/end/top/bottom to the parent's respective sides, which effectively centers it.

Notes:

* `ConstraintLayout` allows flexible positioning. For a simple center you could also use `FrameLayout` or `LinearLayout` with gravity, but `ConstraintLayout` is modern and efficient.

---

6. The Kotlin file: `MainActivity.kt`

Location: `app/src/main/java/com/example/helloworldkotlin/MainActivity.kt` (path will reflect your package name)

This file contains the Kotlin class that represents a screen (Activity) in Android. The default `Empty Activity` template generates a `MainActivity` that extends `AppCompatActivity`.

Full file (complete code)

```kotlin
package com.example.helloworldkotlin

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

Line-by-line explanation

* `package com.example.helloworldkotlin` — declares the package name. This should match the folder structure.

* `import androidx.appcompat.app.AppCompatActivity` — imports the base class for activities that use the support library action bar features.

* `import android.os.Bundle` — imports the `Bundle` class used to pass data to `onCreate`.

* `class MainActivity : AppCompatActivity()` — defines `MainActivity` as a subclass of `AppCompatActivity`.

Inside the class:

* `override fun onCreate(savedInstanceState: Bundle?) { ... }` — `onCreate` is the entry lifecycle callback when the activity is first created.

  * `savedInstanceState` contains saved data from a previous instance (if available).

* `super.onCreate(savedInstanceState)` — call the superclass implementation. This is required.

* `setContentView(R.layout.activity_main)` — sets the UI layout for this activity. `R.layout.activity_main` resolves to the XML layout file `activity_main.xml`.

Notes:

* For a simple static UI, you don't need any code to change the `TextView`. The layout XML alone shows the text. If you want to change the text from Kotlin code, you can find the `TextView` by its ID and set its text programmatically.

Optional: change the text programmatically

If you want to change the text in code, update `MainActivity.kt` like this (with `import android.widget.TextView`):

```kotlin
package com.example.helloworldkotlin

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.TextView

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

        // Find the TextView by ID and update its text
        val textView: TextView = findViewById(R.id.textHello)
        textView.text = "Hello from Kotlin!"
    }
}
```

Explanation of new lines:

* `import android.widget.TextView` — imports the `TextView` UI class.

* `val textView: TextView = findViewById(R.id.textHello)` — looks up the `TextView` instance from the layout using its ID.

* `textView.text = "Hello from Kotlin!"` — sets the `text` property programmatically.

---

7. `AndroidManifest.xml` and app permissions

Location: `app/src/main/AndroidManifest.xml`

A minimal manifest for this app looks like:

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.helloworldkotlin">

    <application
        android:allowBackup="true"
        android:label="HelloWorldKotlin"
        android:icon="@mipmap/ic_launcher"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true">
```

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

    </application>

</manifest>
```

Explanation:

* The `package` attribute identifies the application package.
* `<application>` defines global properties such as label and icon.
* `<activity android:name=".MainActivity">` declares the `MainActivity` class as an activity.
* The `intent-filter` with `MAIN` and `LAUNCHER` makes this activity the app's entry point (so it appears in the launcher).

Permissions: a Hello World app does not need special permissions. If you add features like Internet or reading storage, declare them here using `<uses-permission android:name="android.permission.INTERNET"/>` above the `<application>` tag.

---

8. Gradle files and dependencies overview

Important Gradle values found in `app/build.gradle` (module-level):

```gradle
android {
    compileSdkVersion 34

    defaultConfig {
        applicationId "com.example.helloworldkotlin"
        minSdkVersion 21
        targetSdkVersion 34
        versionCode 1
        versionName "1.0"
    }

    buildTypes {
        release {
            minifyEnabled false
        }
    }
```

```
}

dependencies {
    implementation "androidx.core:core-ktx:1.10.1"
    implementation "androidx.appcompat:appcompat:1.6.1"
    implementation "com.google.android.material:material:1.9.0"
    implementation "androidx.constraintlayout:constraintlayout:2.1.4"
}
```

Notes:

* `compileSdkVersion` indicates which Android SDK you compile against.
* `minSdkVersion` sets the minimum Android version supported by your app.
* `targetSdkVersion` is the version your app targets for compatibility behavior.
* `core-ktx` adds Kotlin extensions for common Android APIs.
* `appcompat` ensures compatibility across older Android versions.
* `material` includes Material Design components.
* `constraintlayout` is the layout library used in the example.

Gradle sync will download these libraries from Maven repositories.

---

9. Building and running your app (emulator and real device)

Using the Android emulator

1. In Android Studio, open **AVD Manager** (Tools > AVD Manager).

2. Create a new virtual device (choose a phone model and a system image such as a recent Android x86 image).

3. Start the emulator.

4. Click the green **Run** button (or Shift+F10) in Android Studio and select the emulator as the deployment target.

5. Wait for Gradle to build and install the APK. The emulator will open and show your "Hello World" app.

Using a real device

1. Enable **Developer options** and **USB debugging** on the Android device.

2. Connect the device via USB (or use wireless debugging with Android 11+ and Android Studio support).

3. Approve the debugging prompt on the device.

4. In Android Studio, select the device and run the app.

Troubleshooting tips:

* If build fails, examine the **Build** window for error messages.

* If the emulator is slow, enable hardware acceleration (HAXM on Intel, WHPX or Android Hypervisor) or use a physical device.

---

10. Common errors and troubleshooting

Gradle sync failed — check your internet connection, ensure the Android SDK and plugins are properly installed, and use compatible Gradle and plugin versions.

App crashes on launch — check the **Logcat** for an exception and stack trace. Common causes:

* Wrong layout resource name in `setContentView` (typo in R.layout.activity_main).
* Null pointer exceptions when calling `findViewById` before `setContentView`.

Emulator not starting — ensure virtualization is enabled in BIOS/UEFI and install the recommended emulator image.

`Unresolved reference: R` — occurs when there are resource errors (such as malformed XML). Fix XML errors and rebuild.

---

11. Next steps and resources

After you master this "Hello World" app, try these next steps:

* Add a Button to the layout and handle clicks in `MainActivity`.

* Learn about `ViewBinding` or `DataBinding` to replace `findViewById`.

* Explore `Fragments`, `RecyclerView`, networking with Retrofit, and persistent storage with Room.

Official resources:

* [Android Developers](https://developer.android.com)

* Kotlin documentation: kotlinlang.org

Appendix: Full project file list (concise)

* `app/src/main/AndroidManifest.xml` — manifest shown earlier

* `app/src/main/java/com/example/helloworldkotlin/MainActivity.kt` — Kotlin code shown earlier

* `app/src/main/res/layout/activity_main.xml` — layout shown earlier

* `app/build.gradle` — Gradle configuration example shown earlier

Printable notes

* Use a fixed-width font for code blocks when printing.
* Keep Android Studio updated to avoid issues with deprecated APIs.

End of guide.