

COURSE

Course : Android™ Basics with Kotlin

Level : Beginner

Duration: 5 Days; Instructor-led

Android Basics is designed primarily for Software Developers, Software Engineers, IT graduates and professionals who are interested to learn to get started on building Android apps incorporating most of the key aspects of the platform using the official integrated development environment (IDE) with the latest Android software development kit (SDK).

Begin with Hello World, participants will work all the way up to create apps that display user interfaces, provide alternative resources to different device configurations such as localize user interface text into one other language, handle events, play music, connect and communicate with network, monitor the connection, parse and consume JSON data, save data on device and display a Google map.

Along with these practical exercises, participants will also learn to work with various top level application components and animations, the Android single-threaded model and support multiple screen sizes.

Combined with extensive real world experiences and examples provided by our instructor, participants will be on a good starting point to build his/her first app on Android platform.

PREREQUISITES

- Java programming language* or one object-oriented programming (OOP) language is required to get the most benefit from this training.
- Experienced in working with IntelliJ IDEA or other similar integrated development environment (IDE) and extensible markup language (XML) would be an additional advantage.

METHODOLOGY

This training is conducted in a combination of lecture and workshop. With practical / hand-ons exercises and step-by-step tutorials, participants will be guided through to create apps that utilise most fundamentals and top-level components of the platform, which can be served as a basis for their future Android projects.

Other times, participants learn the architecture, frameworks and paradigms for the platform through interactive lectures and code demonstration. Blending theory and practice, participants will learn how to build Android apps the right way in this course.

OBJECTIVES

Learn to build Android apps incorporating most of the key aspects of the platform using Android Studio with the latest Android SDK.

By completing this training, participants will be able to:

- Understand how an Android app is built.
- Develop an Android app with most of the key aspects of the platform which include application components, user interface components, app resources, animations, multimedia, networking, data storage, and location.
- Learn to design and create user interfaces with the latest Android stock design and backward compatibility with the AppCompat and Design Support library.
- Make effective and asynchronous network requests with framework API and HTTP client library.
- Build location-aware and Google Map integrated apps.
- Learn how to run the apps on the Emulator, and how to package and install onto Android devices.
- Debug, test and publish Android™ app on Google Play Store.

COURSE MATERIALS

- A Learning app which is downloadable at Google Play store.
- A handout which contains the general information, course outline, detailed chapters for further reading on topics learned including a chapter for Java Fundamental for Android Development, and practical exercises.
- Source code in GitHub for demo apps.
- A set of graphical resources for tutorial apps.

DEVICES AND TOOLS

Participants do not need to have access to an Android device for this training. However, it is good to have one as the test device to use in the training. In the latter case, a compatible USB cable is required to connect the device to the development machine.

Participants who bring his/her own laptop to attend the training, is advised to set up the Android Studio by following the instructions on the link below:

<https://developer.android.com/studio/index.html>

See Android Studio's system requirements at <https://developer.android.com/studio/index.html#Requirements>.

WHAT THIS COURSE COVERS

Module 1 Get Started Begin with an overview of the Android system and its version history, participants will learn how to create a “Hello World” project with Android Studio, explore how an Android app module is structured, and run it on a real device or an emulator.

Module 2 User Interface (UI) This module is largely devoted to the fundamental building blocks of the user interface in Android. We will discuss each of the pre-built UI components, its characteristics and attributes. Participants will learn to use the Android Studio Layout Editor to create layouts by drag-and-dropping the components and by writing the XML vocabulary. Besides that, we will also talk about the special interfaces such as app bar, menus, pop-up messages, dialogs and notifications.

Module 3 Animations & Multimedia Animations give users feedback on their actions and help keep them oriented to the user interface. In this module, we will take a closer look at the various animation frameworks and learn how each of these frameworks implements. Also, we will explore the core APIs from the Android Multimedia framework and learn to play audio or video from media files stored in an app’s resources, in the filesystem, or from a data stream arriving over a network connection.

Module 4 Application Components & Intents App components are the essential building blocks of an Android app. In this module, we will discuss all four different types of app components, learn the distinct purpose each type serves and the lifecycle that defines how these components are created and destroyed. We will further look into the messaging object that facilitates communication between the components and learn to request an action from another app component.

Module 5 Working with Internet & Threads We will first look into the Android single-threaded model in this module, discuss several rules when working with the main thread, also known as the UI thread, as well as the ways to create a separate thread which is known as the worker thread. Participants will be writing the basic code that is involved in connecting to the network, monitoring the network connection, and to parse and consume JSON data. We will also discuss the various HTTP libraries that come with the lengthy list of capabilities that makes networking for Android apps more complete, and learn to present web-based content to users with the WebView framework.

Module 6 Working with Data and Persistent Storage Android provides several options to save app and user data persistently, each of these options is designed to specific needs, such as how much space the data requires, kind of the data to be stored, and whether the data is private to the app or accessible to other apps and the user. In this module, participants will learn how to preserve data either as in key-value pairs, files on the device, or in a database.

Module 7 Location-Aware & Google Map Integration One of the unique features of mobile apps is location awareness. By adding this feature to an app, it could offer users a more contextual experience. In this module, participants will learn how to integrate and show a Google map, and get the user’s location to display a marker on the map.

Module 8 App Publishing In this module, participants will learn the steps to generate a signed Android Package Kit (APK), the package file that is used by the Android system for distribution and installation of mobile apps, and have a walk through to the process to upload an app to the Google Play store.

Module 9 Handling Multiple Screen Sizes Android devices come in all shapes and sizes, by supporting as many screens as possible, an app can be made available to the greatest number of users with different devices, using a single APK or separate APKs. In this module, we will learn the various techniques to support different screen sizes, and also, in some cases, declare restricted screen support.

Remark

Java programming language is a large subject, participants need not know everything but rather having basics knowledge on the following topics for this training:

- Language fundamentals, e.g. statement and block, data types, expressions and operators, flow control.
- Classes and Objects
- Attributes and Behaviour (Data members and Methods)
- Inheritance, Interfaces and Packages
- Access Modifiers and Scope
- Exceptions
- Input and Output Streams
- Data Structure: Arrays and List

MODULES

Module 1 – Get Started

- Overview of Android System
- Android Studio – The Official Integrated Development Environment
 - SDK Tool Updates with SDK Manager
- Android Project Structure
 - Module
 - Source (src)
 - Resource (res) types and Configuration qualifier
 - Libraries (libs) and Assets (assets)
 - The Manifest file
 - Gradle build configuration
 - Dependencies and Support Library
- Running the app on
 - Device
 - Emulator and the Emulator configuration (Android Virtual Device (AVD))

Module 2 – User Interface (UI) & Interaction

- UI Overview
 - ID, Size, Position, Padding and Margins
- Dynamic Instantiation vs XML-Based Layouts
- View Components (Widgets)
 - TextView
 - TextClock (API 17)
 - ImageView
 - EditText
 - Spinner
 - Button
 - ImageButton
 - ToggleButton
 - Checkbox
 - Radio Buttons
 - Switch (API 14)
 - Rating Bar
 - Seek Bar
 - Progress Bar
 - CalendarView
 - DateTimePicker
 - Chronometer
- Working with ViewGroups (Containers)
 - ConstraintLayout (Constraint Layout library)
 - FrameLayout
 - CardView (API 21 - Support library v25.1.0)
 - RelativeLayout
 - LinearLayout
 - TableLayout
 - ScrollView
 - GridLayout (API 14)
 - SwipeRefreshLayout (Support library v22.1.0)
- Special ViewGroups
 - TabLayout
 - ViewPager
 - Custom ViewPager with Gesture motion
 - Adapter ViewGroup
 - GridView
 - ListView
 - StackView (API 11)

- RecyclerView (API 21 – Support library v22.1.0)
 - Getting fancy with Lists
- Special UI Components
 - AppBar
 - Notifications
 - Toasts
 - Snackbar (API 21)
 - Showing Popup Messages (Dialogs)
 - Status bar notifications
- UI Interaction - Events and Event Listeners
- Styles and Themes

Module 3 – Animations & Multimedia

- Animation
 - View Animation
 - Property Animation (API 11)
 - Interpolation
 - Transition framework (API 19)
- Android Multimedia framework
 - MediaPlayer
 - VideoView

Module 4 – Application Components & Intents

- Activity and its lifecycle
 - Starting an Activity
 - Save / Restore Activity State
 - Activity in The Manifest
 - Specify app's launcher activity
- Service (Part 1)
 - Unbounded Service
 - IntentService
- Broadcast Receiver
- Introduction to Content Provider
 - Built-in Providers
 - Accessing a Provider
- Intents and Intent Filters
 - Interacting within an App
 - Passing data in and out of Activities
 - Interacting with other Apps
- Application Permissions
 - Runtime Permissions (API 23)

Module 5 – Working with Internet & Threads

- Android Single-threaded Model
- Worker / Background Threads
 - Threads and Runnables
 - Message Handler
 - AsyncTask
- HTTP Operation with HttpURLConnection
- HTTP Client Libraries - Volley
 - RequestQueue
 - Setting up a RequestQueue with own implementation of Cache and Network Stack
 - Requests
 - Request Customization
 - Send a Request
 - The Singleton Pattern
 - ImageRequest vs ImageLoader
 - NetworkImageView
- The WebKit / Chrome (AOI 19) Embedding Browser

Module 6 – Working with Data & Persistence Storage

- Persistence Storage
 - o Shared Preferences
 - o Internal Storage
 - o External Storage
 - o SQLite Database
 - SQL Statements
 - Data Definition
 - Queries and Data Manipulation
 - Query and Cursor
- Parsing JSON / XML data

Module 7 – Location-Aware and Google Map Integration

- Getting Started
- Annotating The Map
- Location Listeners
 - o Android's Network Location Provider
 - o GPS

Module 8 – App Publishing

- Building and signing an app
 - o Debug keystore & Release keystore
- Releasing app on Google Play
 - o Account Registration
 - o App's Listing
 - o Configuring options
 - o Create Release
 - Alpha, Beta and Production
 - o Reports
 - Crashes & ANRs
 - Reviews
 - Statistics
 - Financial Reports

Module 9 – Handling Multiple Screen Sizes

- The Default Rules
- Screen size and Density Differs
- Various Dimension Units
- Scalable Drawables
 - o 9-Patch Drawable
 - o Vector Drawable (API 21)
- Tailor Made
 - o Screen Sizes Support <supports-screens>

OPTIONAL MODULES

Module - Java Fundamentals for Android

- Introduction to Java
- Statements and Expressions
- Variables and Data Types
- Expressions and Operators
 - o Assignment Operator
 - o Arithmetic Operators
 - o Increment and Decrement Operators
 - o Comparison Operators
 - o Logical Operators
 - o Operator Precedence
- Logic and Loops
 - o If Conditionals
 - o Switch Conditionals
 - o For Loops
 - o While Loops
 - o Do-while Loops
 - o Breaking out of loops
 - o Labeled Loops
- Working with Exceptions
- Object-Oriented Programming
 - o Objects and Classes
 - o Attributes and Behavior
 - o Class Constructors
 - o Inheritance, Polymorphism, Interfaces, and Packages
 - o Access Modifiers and Encapsulation
- Data Structure
 - o Arrays
 - o List
 - o Hash Tables
- Coding Convention

Module – Action on Phone

- Working with Sensors
 - o Introduction to Sensors
 - o Type of Sensor
 - o The Android Sensor Framework
- Motion Sensors
 - o The Accelerometer
 - o The Gravity Sensor
 - o The Gyroscope
 - o The Rotation Vector Sensor
- Position Sensors
 - o The Orientation Sensor
 - o The Geomagnetic Field Sensor
 - o The Proximity Sensor
- Environment Sensors
 - o The Light, Pressure, and Temperature Sensors
 - o The Humidity Sensor

ANDROID™ API VERSION TABLE

API 3 – Android 1.5 Cupcake	/
API 4 – Android 1.6 Donut	/
API 5 – Android 2.0 Éclair	/
API 8 – Android 2.2 Froyo	/
API 9 – Android 2.3 Gingerbread	/
API 11 – Android 3.0 Honeycomb	/
API 14 – Android 4.0 Ice Cream Sandwich	/

Android Studio Setup

Android Studio is the official IDE for Android application development, based on IntelliJ IDEA. Some key features including:

- Flexible Gradle-based build system
- Build variants and multiple apk file generation
- Code templates to help you build common app features
- Rich layout editor with support for drag and drop theme editing
- Lint tools to catch performance, usability, version compatibility, and other problems
- ProGuard and app-signing capabilities
- Build-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine.

⚠ Before setting up Android Studio, be sure you have installed JDK 6 or higher (JDK 7 is required when developing for Android 5.0 and higher).

To check if your computer has JDK installed (and version), open Command Prompt in Windows or Terminal in Mac OS X and type `javac -version`. If the JDK is not available or the version is lower than 6, download and install the JDK at

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

Windows steps:

1. Download Android Studio at <https://developer.android.com/sdk/index.html>.
2. Launch the downloaded .exe file.
3. Follow the setup wizard to install Android Studio and any necessary SDK tools.

① If the launcher script does not find where Java is installed, you need to set an environment variable indicating the correct location. Select Start menu > Computer > System Properties > Advanced System Properties. Then open Advance tab > Environment Variables and add a new system variable `JAVA_HOME` that points to your JDK folder.

Mac OS X steps:

1. Download Android Studio at <https://developer.android.com/sdk/index.html>.
2. Launch the .dmg file you just downloaded.
3. Drag and drop Android Studio into the Applications folder.
4. Open Android Studio and follow the setup wizard to install any necessary SDK tools.

Getting the Demo Project

The source code for demo in this training is available as a public repository on GitHub. You can find the repository at <https://github.com/CodePlay-Studio/android-demo>.

The following steps install Git, create a copy of the demo project repository on your own GitHub account and import the project from the repository to your Android Studio. It requires a GitHub account to be signed up at www.github.com.

Install and configure Git

1. Download the latest stand-alone installer for your machine's operating system at <https://git-scm.com/downloads>.
2. Launch the installer and follow the prompts to install Git.
3. Open Command Prompt on Windows (Terminal on Mac or Linux) and verify the installation was successful by typing `git --version`:

```
$ git --version
git version 2.21.0
```

4. Configure your Git username and email using the following commands, replacing the values within the double quote with your own from the GitHub account:

```
$ git config --global user.name "Your name"
$ git config --global user.email "your@email.com"
```

These details will be associated with any commits¹ that you create later in Git.

Setup Git and GitHub account on Android Studio

1. In the bottom-right of the **Welcome to Android Studio** window, select **Configure > Settings** (**Preferences** on Mac) from the dropdown menu.
2. In the left pane of the **Settings (Preferences)** window, expand **Version Control** section and click **Git**.

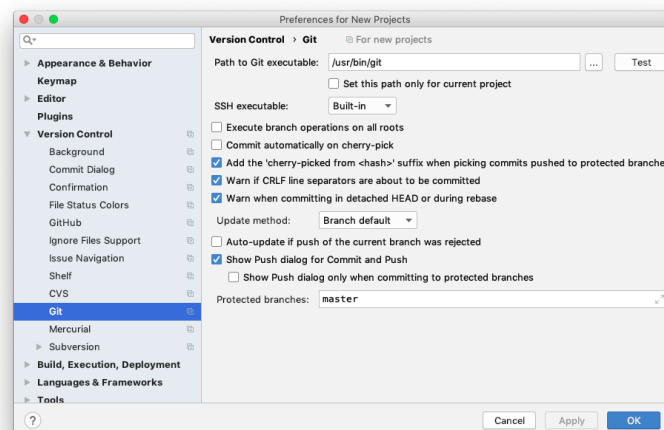



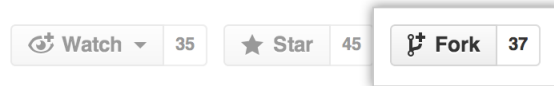
Figure 1 Version Control > Git in Settings window


¹A commit is like a snapshot of all the files in your project at a particular point in time.

3. Click on the **test** button to test Path to Git executable. Proceed to next step if “Git Executed Successfully” message dialog shows, otherwise change the path to the git.exe from where you installed and test again.
4. Click **GitHub** in the left pane, click **Add**  and enter the email and password of your GitHub account in the **Log In to GitHub** popup dialog and **Log In**.
5. Click **OK** to close the window after your GitHub account has been added successfully.

Create a copy of the repository

1. Open the demo project repository on a browser and sign in with your GitHub account.
2. Click the **Fork** button in the header of the repository.



 You will notice that the repository name is now followed by your account username after forking is completed. A fork is a copy of a repository and forking is the process of producing a personal copy of one's project as the starting point for your own, which allows you to freely experiment with changes without affecting the original project.

3. On your fork of the demo project repository, click **Clone or Download** and copy the web URL for your repository in the **Clone with HTTPS** section.

Import project to Android Studio

To be able to work on the project, you will need to clone it to your local machine. Cloning get a copy of an existing repository. It creates a directory, initializes a .git directory inside it, pulls down all the data for that repository, and checks out a working copy of the latest version.

4. In the **Welcome to Android Studio** window (or if you have a project opened, select **VCS** from the menu), click the **Check out project from Version Control** and select **Git** from the dropdown menu.
5. In the **Clone Repository** dialog, enter the newly forked repository URL copied in step 3 and change the directory you may want to save the project and click **Clone**.

Upon completing the above steps, the demo project will be downloaded, and Android Studio will open and build the project.

Tutorial 1

The following tutorial teaches you how to build a simple app: Loan Calculator. You'll learn how to create a "Hello World" project with Android Studio and run it. Then, you'll create the user interfaces for the calculator, take some user input and calculate the result on a button tap. By completing this tutorial, you'll learn the fundamental aspects of Android apps development.

Lesson 1: Build your first app

This lesson shows you how to create a new Android project with Android Studio and describes some of the files in the project.

1. In the **Welcome to Android Studio** window, click **Start a new Android Studio project**.

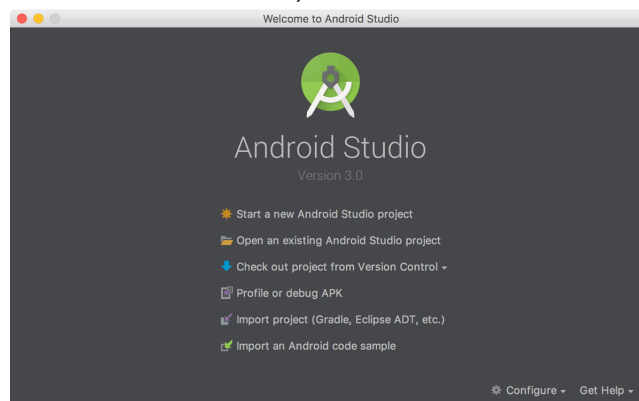


Figure 2 Android Studio Welcome window

Or if you have a project opened, select **File > New Project** from the menu bar.

2. In the **Choose your project** screen of the **Create New Project** window, leave the **Target Devices** as default (**Phone and Tablets**) and select **Empty Activity** and click **Next**.

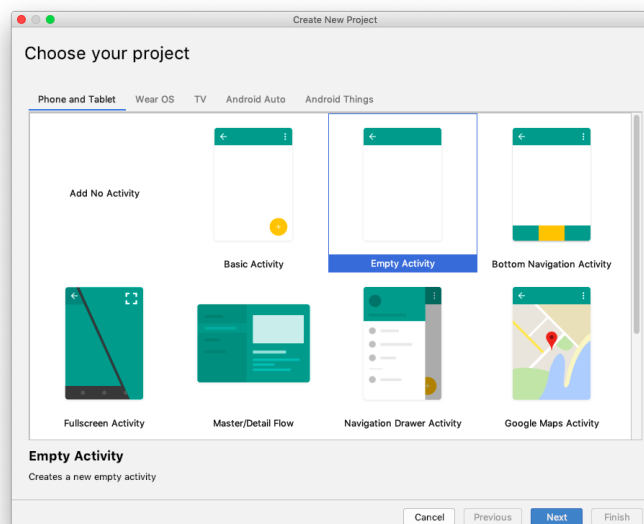


Figure 3 Create New Project window

3. In the **Configure your project** screen, enter the following values:

Name: the app name that appears to users, enter 'My First App'.

Package Name: the package namespace that uniquely identifies your app (following the same rules as packages in the Java programming language). For this reason, it is generally recommended if you could use a domain name that begins with the reverse order.

Save Location: Replaced the directory name ".../MyFirstApp" to ".../AndroidBasicsTutorials". You can place this project to a different location or leave it as default.

Language: Leave the Java option as it is, or select Kotlin if you want to include Kotlin language support in this project.

Minimum API level: set the lowest Android version that your app supports. Select **API 21: Android 5.0 (Lollipop)** for this project.

Leave the other options as they are, and click **Finish**.

The Android app module structure

If you follow the previous lesson, it will create a "Hello World" Android project with a default set of source files that allow you to immediately run the app.

An Android Studio project can have one or many modules in which a module is a collection of source files and build settings that allow you to divide your project into discrete units of functionality. Each module can be independently built, tested, and debugged. Now take a moment to review the most relevant directories and files you should be aware of in an Android app module.

Be sure the **Project** window is open (select **View > Tool Windows > Project**) and the **Android** view is selected from the drop-down list at the top of that window.

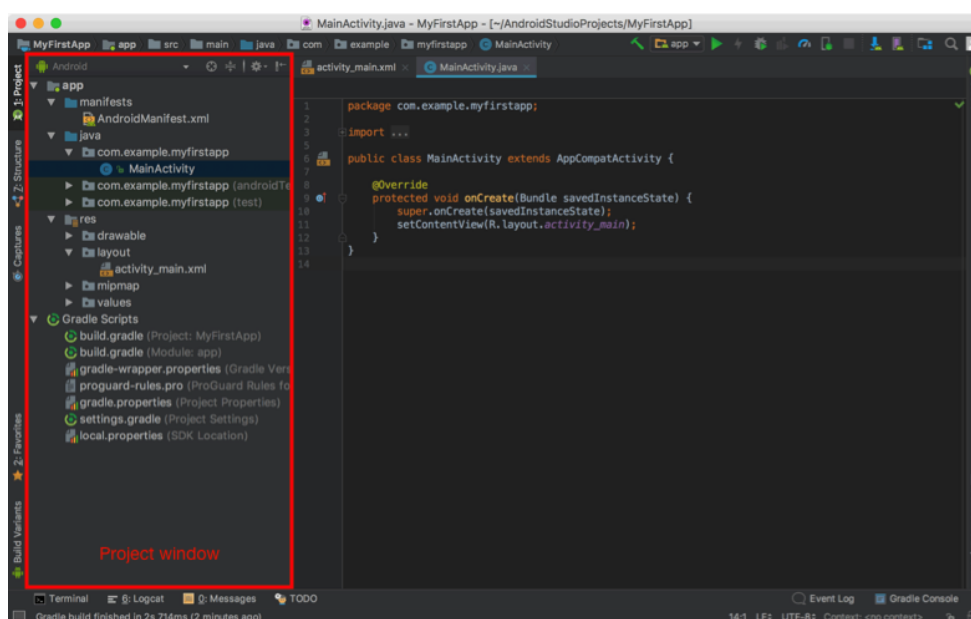


Figure 4 Project window (Left panel) in Android Studio

app > java

Contains the Java or Kotlin source code files for your app's main and test sourceset.

app > res

Resource directory that contains several predefined sub-directories for storing app resources. App resources are the additional files and static content that your app uses, such as images, layouts, user interface strings, and more. Android compiler automatically generates an ID for any files located inside these directories. These IDs are stored in the R class to let other classes and methods, and XML files to access it.

① Naming rules for Android resources are similar to Java identifiers¹. The only difference is that there is no uppercase in Android resource files.

app > manifests > AndroidManifest.xml

The manifest file describes the fundamental characteristic of an app to the Android system. It:

- names the Java package that serves as a unique identifier for the app.²
- declares the minimum level of the Android API the app requires and its target level.³
- defines each of its components and their capabilities.
- declares permissions the app must have in order to access protected parts of the API and interact between it and the other apps.
- lists the libraries (a.k.a dependencies in Gradle) that the app links against.⁴

Gradle Scripts > build.gradle

There are more than one build configuration files in a project: one for the project and one for each module. You'll mostly work with the module's build.gradle file to configure how the Gradle tools compile and build your app.

① Gradle is an open-source build automation tool that is run on the Java Virtual Machine (JVM).

¹The only allowed characters in Java identifier rules are all alphanumeric characters ([A-Z],[a-z],[0-9],'\$' and '_'). It is case-sensitive and the first character of each identifier cannot be a digit.

^{2, 3, 4}The values of these properties generally be configured in module's build.gradle of an Android Studio project, these values override the existing values of the same properties set (if any) in the manifest file.

Lesson 2: Run your app

This lesson shows you how to install and run your app on a real device or an emulator.

Run on a real device

❗ Proceed with step 2 if you are developing on Mac or Linux.

1. Install the appropriate USB driver for your device. You might find the appropriate driver for your device from the OEM drivers table in this link: <https://developer.android.com/studio/run/oem-usb.html#Drivers>.

❗ Or download and install the Universal ADB driver by ClockworkMod at <https://adb.clockworkmod.com>.

2. Enable **USB debugging** in the **Developer options** by following the instruction below:
 - a. Go to **About Phone*** in the **Settings**.
 - b. Scroll to **Build number** and tap 7 times.
 - c. Return to **Settings** to find **Developer options***.
 - d. Open **Developer options**, scroll to **USB debugging** and enable it.

*About Phone and Developer options may be named and located differently on different devices.

3. Connect your device to your development machine with a USB cable.
4. In Android Studio, right-click the **app** module in the **Project** window and then select **Run > Run** (or click **Run** ► with your app is selected from the run/debug configurations drop-down menu in the toolbar).
5. In the **Select Deployment Target** window, select your device and click **OK**.

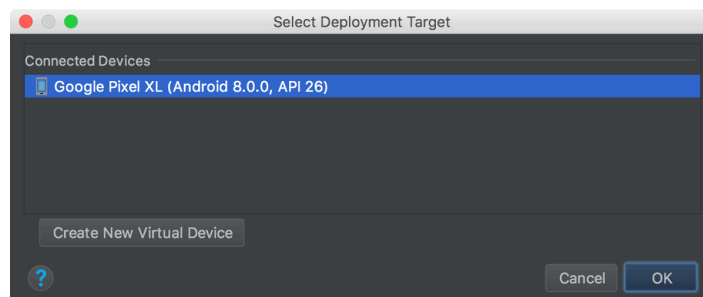


Figure 5 A connected device in Select Deployment Target window

Run on an emulator

To run your app on the emulator, you need to first create an Android Virtual Device (AVD). An AVD is a device configuration for the Android emulator that allows you to model different devices.



1. In Android Studio, select **Tools > AVD Manager** (or click **AVD Manager**  in the toolbar).
2. In the **Android Virtual Device Manager** window, click **Create Virtual Device....**



Figure 6 Android Virtual Device window

3. In the **Select Hardware** screen, select a phone device, such as Pixel, and click **Next**.
4. In the **System Image** screen, select an Android version to run on the device. If you don't have that version installed, click the Download link to complete the download. Click **Next**.
5. In the **Android Virtual Device (AVD)** screen, give it a name and click **Finish**.
6. In Android Studio, right-click the app module in the Project window and then select **Run > Run** (or click **Run**  in the toolbar).
7. In the **Select Deployment Target** window, select your device and click **OK**.

By completing the steps in this lesson, Android Studio installs the app on your connected device or the emulator and starts it. You should now see “Hello World” displayed in the app running on your device.

The result may be nothing exciting, but it is important that you understand how to run your app before you start developing. When you build and run the app, an Activity class starts and loads a layout file that shows the “Hello World”.

Lesson 3: App's user interface

The user interface for an Android app is built using a hierarchy of containers (ViewGroup objects) and widgets (View objects). Containers control how their child elements are positioned on the screen. Widgets are UI components that draw something on the screen, such as text boxes, and users may interact with, such as buttons.

Figure 7 on the right shows a root ViewGroup (1) that contains the layout of the entire screen, where it is also the parent of the first group of child elements (2) in the hierarchy.

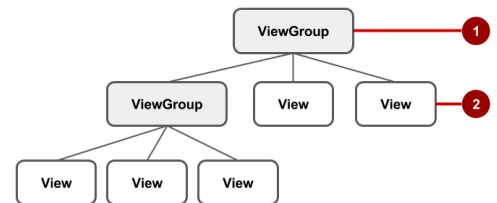


Figure 7 View Hierarchy

In addition to ViewGroup and View classes, Android provides an XML vocabulary for these classes to allow an interface to be defined in XML files. The advantages of doing so are:

- Separating the presentation of the app from code that controls its behaviour, makes it easy to maintain the UI.
- Able to visualise the structure of UI in the Android Studio Layout Editor without the app running on a device or an emulator, so it is easier to work with and to debug.

In this lesson, you will use the Layout Editor to create two layouts: one for the splash screen¹, another to take some user input and respond to a button tap to calculate and display the result of compound interest loan.

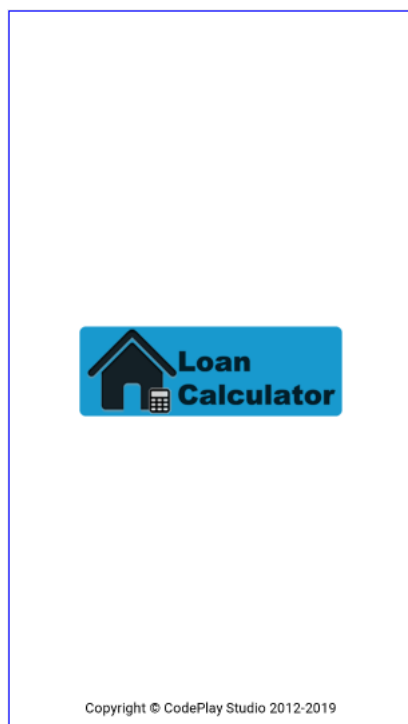


Figure 8 Splash screen

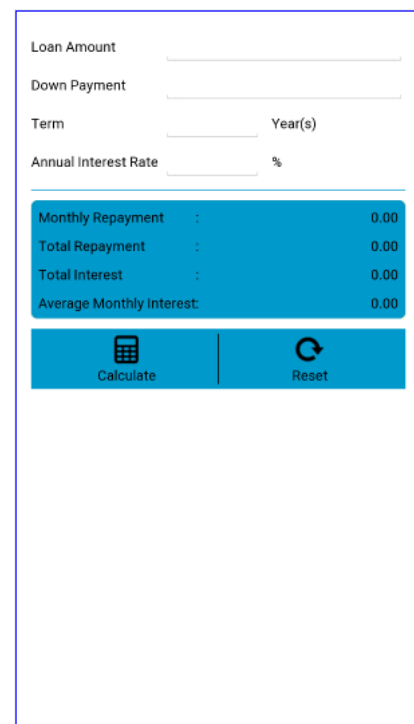


Figure 9 Calculator screen

¹Splash screen, also referred to as launch screen or startup screen, is a screen that appears upon app starts, usually with minimalistic design by presenting a name, logo, or slogan of a product.

App Resources

1. In Android Studio's **Project** window with the **Android** view selected from the drop-down list at the top, open **app > res**.

Now take a moment to review the type of resource places at each of the following sub-directories:

- **drawable**² Bitmap files (.png, .jpg, .gif) or resizable bitmaps - Nine-Patches (.9.png)³ or XML files that are compiled into the subtypes of drawable resources.
- **layout** XML files that define a user interface layout.
- **mipmap**² Drawable files for different launcher icon densities.
- **values** XML files that contain simple values, such as colors, strings, styles, and more.

① Unlike resources in other directories where the file-based resources are referenced by filename without the extension, resources in **values** are formed by key-value pairs where the key⁴ provided in the **name** attribute will be compiled into a resource ID to refer to its value. Learn more resource types at

<https://developer.android.com/guide/topics/resources/available-resources>.

2. To add drawable resources, copy the Loan Calculator image set and paste to the **app > res > drawable**, and click **OK** in the popup window.
3. The **colors.xml** in **values** is a color resources file where you can manage colors specified in hexadecimal value⁵. Open **app > res > values > colors.xml** from the **Project** window and change the respective values of the following color resources:
 - colorPrimary to #0099CC
 - colorPrimaryDark to #006B9B
 - colorAccent to #5DCAFF

① The Android project created with Android Studio applies a material theme to the app by default, as defined in the AppTheme style in **app > res > values > styles.xml**. It extends a theme from the support library and includes overrides for color attributes that reference the color resources above. These color attributes allow you to apply theme's color palette easily to your app:

- colorPrimary is the primary branding color for the app. This is the color applied to the action bar background by default.
- colorPrimaryDark is a dark variant of the primary color. This is the color applied to the status bar (via statusBarColor attribute) and navigation bar (via navigationBarColor attribute), it gives users a clear separation between system-controlled elements and your app.
- colorAccent is a bright complement to the primary color. It provides the framework controls a contrast color (via colorControlActivated attribute) to draw attention such as EditText's focused underline and cursor, current tab indicator.

²Android system will strip out other resources from the **drawable** directories that are not relevant to the device density but keep all resources in the **mipmap** directories. This is why using a **mipmap** directory for storing launcher icon for different device densities is a recommended practice by Android team.


³NinePatch drawable is an image that is resized in a way that does not distort its appearance. A one pixel border that is not displayed is defined on the top-left to specify the part of the image which can be stretched and bottom-right to specify the content area using Android ninepatch tool.

⁴A key should not contain uppercase, spaces, and symbols.

⁵Hexadecimal color value begins with a pound (#) symbol, followed by the Alpha-Red-Green-Blue information in one of the following formats: #RGB, #ARGB, #RRGGBB, and #AARRGGBB.

4. The **strings.xml** in **values** is a string resources file where you should specify all your UI strings. To manage string resources, open the **app > res > values > strings.xml** from the **Project** window.

① Keeping values such as strings and colors in separate resource files makes it easier to manage them, especially if you use them more than once. For example, manage all UI strings in a single location, which makes it easier to find, update, and localise (compared to hard-coding strings in layout or code).

5. Click **Open editor** at the top-right of the editor window. This opens the **Translations Editor**, an editor that provides a simple interface for adding and editing default strings, and helps keep all translated strings organised.
6. To create a new string, click **Add Key**  at the top of the editor, and enter a name and value in the **Add Key** dialog and click **OK**.

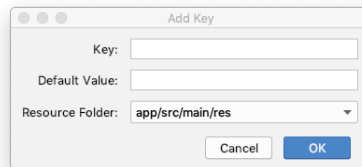


Figure 10 Enter key and value in Add Key dialog to create a new string

Repeat this step until all the strings in the following table are added:

Key	Default Value
loan_amount	Loan Amount
down_payment	Down Payment
term	Term
year	Year(s)
annual_interest_rate	Annual Interest Rate
monthly_repayment	Monthly Repayment
total_repayment	Total Repayment
total_interest	Total Interest
average_monthly_interest	Average Monthly Interest
default_result	0.00
calculate	Calculate
reset	Reset

① You can also extract hard-coded strings in an XML layout file to string resources by following these steps:

1. Click on the hard-coded string and press Alt-Enter (Option-Return on a Mac).
2. Select **Extra string resource** from the popup menu.
3. In the **Extract Resource** window, edit the **Resource name** for the string value and click **OK**.

Introduction to Layout Editor

7. In Android Studio's **Project** window with the **Android** view selected from the drop-down list at the top, open **app > res > layout > activity_main.xml**.

The Layout Editor as shown in the figures below appears when you open the XML layout file. It shows a visual representation of XML code where you can build layouts by drag-and-dropping UI elements in the design editor (left figure) or write XML in text editor (right figure). If your editor shows the XML source, click the **Design** tab at the bottom of the window to switch to design editor.

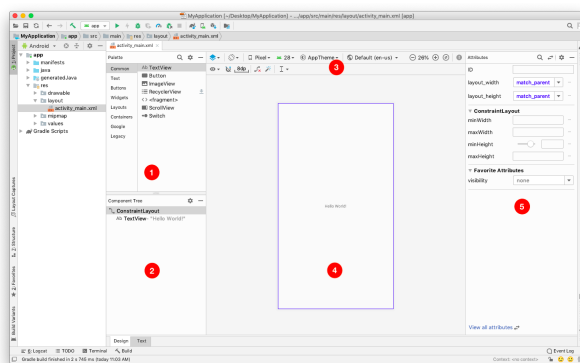


Figure 10 Layout Design Editor

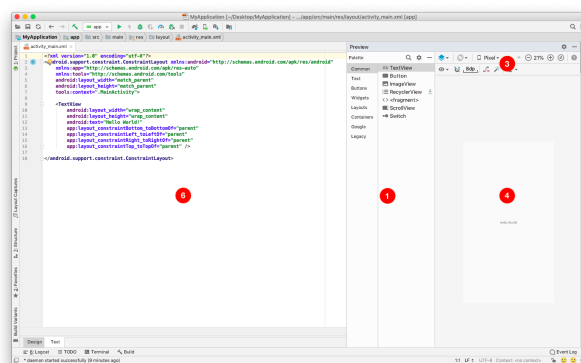


Figure 11 Layout Text Editor

Alternatively, you may use the shortcut key: **Alt (Control on Mac) + Shift + Left/Right arrow** to switch between design and text editors.

Corresponding to the numbers of figures above, the regions of the editor are as follows:

- | | |
|--------------------------|---|
| 1 Palette | List of views or viewgroups that can be dragged into the layout. |
| 2 Component Tree* | Shows the view hierarchy for the layout. |
| 3 Toolbar | The top toolbar provides you the buttons to configure the preview appearance of your layout ⁶ , while the bottom toolbar lets you configure the appearance of UI elements in a <code>ConstraintLayout</code> . |
| 4 Design editor | Shows layout in Design (a real-world preview) or Blueprint (only outlines for each view), or both ⁷ . |
| 5 Attributes* | Pane for setting attributes for a view element. |
| 6 Text editor | Shows and allows editing the layout's XML source. |

*While in the text editor, you can also view the palette, toolbar and design editor by clicking **Preview** on the right side of the window. The component tree and attributes are not available from the text editor.

In the following sections, you will build layouts using both the editors: a layout for boot up screen by drag-and-dropping views in the design editor, and a layout for calculator screen by writing XML in text editor.

⁶These configurations have no effect on your app's code or manifest. They affect only the layout preview.

⁷Press B to cycle through these views.

Working with ConstraintLayout in Design editor

In this section, you will learn the basic usage of ConstraintLayout by building a layout for the boot up screen. ConstraintLayout is a container that defines the position for each view based on constraints to sibling views and the parent with a flat view hierarchy (no nested viewgroups). It is available from a support library that is compatible with Android 2.3 (API level 9) and higher released in May 2017.

The ConstraintLayout was built from the ground up along with the Layout Editor with GUI building in mind - to build a layout entirely from the design editor by drag-and-dropping views instead of writing XML, and to reduce the number of nested views⁸.

8. Click **Select Design Surface**  in the toolbar and select **Design + Blueprint** from the drop-down list.

You will now see a blueprint view next to the design preview in the editor. The blueprint view helps you see the constraints and guidelines more clearly without getting distracted by the content or background.

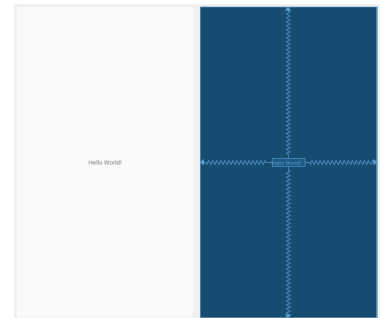



Figure 12 Design (Left) and Blueprint (right) views

9. Select the **TextView** in the **Component Tree** (or from the editor) and press Delete.
10. Drag an **ImageView** from the **Palette** pane to the design editor and drop it near the center of the layout.
11. In the **Resources** window, select a drawable named logo from the **Project** section and click **OK**.
12. Click the **ImageView** in the design editor, you can now see the resizing handles on each corner (squares), and the constraint anchors on each side (circles) in the blueprint view. 
13. Click-and-hold the anchor on the top, and drag it up until it snaps to the top of the layout and release. Similarly, create constraints from the rest of the all three sides, e.g. left side of the view to the left side of the layout. This will create constraints that act like opposite forces pulling the view apart equally, such that the view will end up being centered in the parent.

① When you drop a view into the Layout Editor, it stays where you leave it even if it has no constraints. However, this is not the actual position of the view, a view with no constraints will be drowned at position [0,0] (the top-left corner) when you run your app and showing the layout on a device.

To define a view's position in ConstraintLayout, you must add at least one horizontal and one vertical constraint for the view. Each constraint represents a connection or alignment to another view, the parent, or an invisible guideline.

To learn the various types of constraints, follow the Developer Guide in <https://developer.android.com/reference/android/support/constraint/ConstraintLayout#developer-guide>.

⁸The deeper the view hierarchy of a layout (default maximum depth is 10), the more memory it uses to initialize, layout, and draw on screen. Thus, reducing the number of nested viewgroups will improve the layout performance.

Writing XML in Text editor

It is sometimes quicker and easier to edit the XML code directly, especially when copying and pasting the code for similar views. In this section, you will be building a layout for the loan calculator by writing XML directly in the Text editor.

14. Right-click on **app > res > layout**, select **New > Layout resource file** from the popup menu.
15. In the **New Resource File** window, enter 'activity_calculator' for **File name** and 'LinearLayout' for **Root element**, click **OK**.

❗ Each layout file must contain exactly one root element. The root element can be either a ViewGroup, a View, or a <merge> element and it must have the xmlns:android attribute set to "http://schemas.android.com/apk/res/android" to define the Android namespace.

16. To view and edit the XML code, click the **Text** tab at the bottom of the newly created layout window. The following XML code shown:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

</LinearLayout>
```

Common Containers (ViewGroups)

Android provides various ViewGroup objects that are designated as containers where each of these containers organises child elements in a specific way. Some examples of container are:

- ConstraintLayout defines the position of each child element using constraints, edges, and guidelines to control how they are positioned relative to others and the parent in the layout.
- FrameLayout draws child elements in a stack, with the most recently added child on top (overlapping the other).
- The children in RelativeLayout are positioned and aligned relative to each other or to the parent.
- LinearLayout aligns children in a single direction, vertically or horizontally.
- TableLayout arranges its children into rows and columns.

XML attributes (View properties)

Views have properties that define where a view appears on the screen, e.g. its size, how the view relates to other views, and how it responds to user input. When defining views in XML, these properties are referred to as attributes. Attributes are generally in the form of:

namespace:attribute_name="value"

Example, LinearLayout is required to have the android:orientation attribute to specify the direction to align its children either in horizontal or vertical.

View Sizing

The size of a view is expressed with the `android:layout_width` and `android:layout_height` attributes. It can take one of three values:

- `match_parent` expands a view to fill its parent by width or height. When this view is the root element, it expands to the size of the device screen.
- `wrap_content` shrinks a view to the size of its content. If there is no content, the view becomes invisible.
- Measurement units typically in density-independent pixels⁹ (dp or dip).

① Android devices come in all shapes and sizes, so you should implement your layout with flexibility in mind. That is, avoid defining your layout with rigid dimensions (hard-coding position and size of your view components) whenever possible. Instead, use `match_parent` and `wrap_content` for the width and height of most views.

17. Enter the below code to add a `TableLayout`. The attributes

- `android:layout_margin` sets the space of all four edges of a `View` to its parent.
- `android:stretchColumns` specifies the column's zero-based index (Use comma separator for multiple columns and asterisk (`*`) symbol for all columns) to stretch a column or columns to fill the rest of the space (if any) in the table.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...>

    <TableLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="16dp"
        android:stretchColumns="2">
    </TableLayout>

</LinearLayout>
```

⁹Density-independent pixels (dp) are independent of screen resolution. One dp is a virtual pixel unit that is roughly equal to one pixel on a medium-density screen (160dpi; the “baseline” density). This value is translated to the appropriate number of real pixels for other density based on a scale factor.

Maintaining density independence is important because, without it, a UI element might appear larger on a low-density screen and smaller on a high-density screen. Thus, other measurement units such as pixels (px), points (pt), inches (in) are although acceptable but not used.

18. A `TableRow` defines a row in a `TableLayout` with each direct child is a column. For instance, the code below creates a row with two columns, a `TextView` for displaying 'Loan Amount', and an `EditText` as the field for user to input the amount:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...>

    <TableLayout
        ...>

        <TableRow>
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/loan_amount"/>

            <EditText
                android:id="@+id/etLoanAmount"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginStart="8dp"
                android:layout_span="2"
                android:inputType="numberDecimal"/>
        </TableRow>
    </TableLayout>
</LinearLayout>
```

① Although the typical child of a `TableLayout` is a `TableRow`, you can actually use any view where this view will be displayed as a single row that spans across all the table columns.

Referencing resources

The syntax to reference a resource in XML is as follows:

@package_name:resource_type/resource_name

- `package_name` is the name of the package in which the resource is located. The package name is not required when you reference resources that are stored in the resources directory (**res/**) of your project or libraries.
- `resource_type` is the type of resources which typically follow the name of the pre-defined sub-directories in the resources directory (**res/**), such as `drawable`, `layout`, and `mipmap`. Except that for the resources in **values/** directory, it is the children of the `<resources>` element, such as `array`, `bool`, `color`, `id`, `dimen`, `integer`, `string`, `style`, and more, to be the type.
- `resource_name` is either the resource filename without the extension, or the `android:name` attribute in the XML element.

For example, the `TextView` in the `TableRow` sets the `android:text` attribute to a string resource added in early step with `@string/loan_amount`.

Identifying and Referencing a View

The `android:id` attribute lets you specify a unique resource identifier for a view and reference it from your code. In the code above, an id called `etLoanAmount` is created for the EditText with:

```
android:id="@+id/etLoanAmount"
```

i You use the plus (+) symbol to indicate that you are creating a new id.

You can then reference to this id,

in Java: `R.id.etLoanAmount`

in XML: `@id/etLoanAmount`

19. Select and copy the `<TableRow>...</TableRow>` from the previous step, and paste it after. Change and set the TextView's `android:text` attribute to `@string/down_payment` and the EditText's `android:id` attribute to `@+id/etDownPayment`.

20. Paste the same code copied at the previous step after the second TableRow. Change and set the following:

- TextView's `android:text` attribute to `@string/term`.
- EditText's `android:id` attribute to `@+id/etTerm`.
- EditText's `android:inputType` attribute to `number`.

addition to the above, add and set the following attributes to the EditText:

- `android:gravity="center"`
- `android:minWidth="50dp"`

21. Add a TextView to display 'Year(s)' as the third child element appears after the EditText within the TableRow added in step 20.

22. Copy the `<TableRow>...</TableRow>` created in step 20 and 21, and paste it after the third TableRow. Change and set the following:

- First TextView's `android:text` attribute to `@string/annual_interest_rate`.
- EditText's `android:id` attribute to `@+id/etAnnualInterestRate`.
- EditText's `android:inputType` attribute to `numberDecimal`.
- Last TextView's `android:text` attribute to a percent (%) symbol.

By completing the above steps, you have created a simple form that lets users input the required values to calculate a compound loan. Your layout preview and component tree in design editor should now be similar to the figure below:

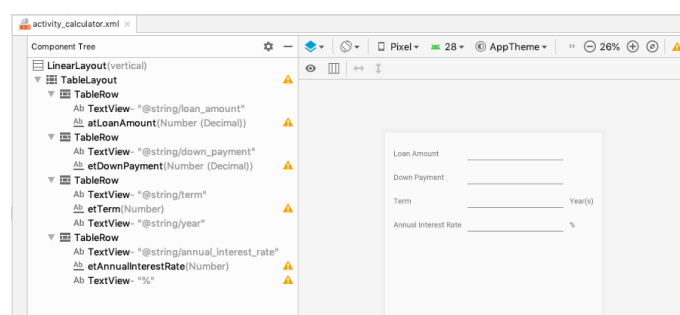


Figure 14 Preview and Component Tree in Layout Design Editor

Next, you will create a second table in the layout to display the calculation result.

23. Enter the code below to draw a line separator between the form table and the result table:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...>

    <TableLayout...>

    <View
        android:layout_width="match_parent"
        android:layout_height="1dp"
        android:layout_marginEnd="16dp"
        android:layout_marginStart="16dp"
        android:background="@color/colorPrimary"/>

</LinearLayout>
```

24. Add another TableLayout (refer to step 17) after the line separator with the following attributes set:

- android:layout_width to match_parent.
- android:layout_height to wrap_content.
- android:layout_margin to 16dp.
- android:background to a drawable resource named bg.
- android:stretchColumns to stretch the third column.

25. Enter the code below to create a row with 3 columns to display the monthly repayment from the calculation: a TextView for displaying the 'Monthly Repayment' caption, a TextView for ':', and a TextView to show the result.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...>

    <TableLayout...>

    <View...>

    <TableLayout
        ...>
        <TableRow>
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/monthly_repayment"/>

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text=":"/>

            <TextView
                android:id="@+id/tvMonthlyRepayment"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:gravity="end"
                android:text="@string/default_result"/>
        </TableRow>

    </TableLayout>

</LinearLayout>
```


The `android:gravity` attribute indicates where to position the content in its container if there is room in the container (content is smaller than the container). The content could be the child within a viewgroup, the label of a button, the text in a text view, and so on. In the code above, the text of the last `TextView` is pushed to the end of its container.

26. Select and copy the `<TableRow>...</TableRow>` added in the previous step, and paste it after thrice. Change and set the first `TextView`'s `android:text` attribute and last `TextView`'s `android:id` attribute of each `TableRow` to:
 - Row 2: `@string/total_repayment`, `@+id/tvTotalRepayment`
 - Row 3: `@string/total_interest`, `@+id/tvTotalInterest`
 - Row 4: `@string/average_monthly_interest`, `@+id/tvMonthlyInterest`
27. Give the rows within this table a margin of 8dp in between. (Hint: Set the `TableRows`' `android:marginTop` attribute)
28. The remaining steps create a calculate button and a reset button which are aligned side-by-side horizontally after the result table. Enter the code below to add a 'Calculate' Button wrapping in a `LinearLayout` with its `android:orientation` attribute set to horizontal.

```
...
</TableLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/colorPrimary"
    android:orientation="horizontal">

    <Button
        android:id="@+id/btnCalculate"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:background="@null"
        android:drawableTop="@drawable/ic_calculate"
        android:text="@string/calculate"/>

</LinearLayout>

</LinearLayout>
```

29. Select and copy the `<Button.../>` in the previous step, and paste it after. Change and set the following:
 - `android:id` to `@+id/btnReset`
 - `android:drawableTop` to `@drawable/ic_reset`
 - `android:text` to `@string/reset`

Note that although the two buttons have a width of 0dp, each button is actually occupied half of the width on the screen. The declared weight in the `android:layout_weight` attribute, supported only in `LinearLayout` and `GridLayout`, allocates the available space to a view in proportion. The horizontal space of the two buttons is in equal distribution, because each of their weight is set to 1. The default value is 0.

Lesson 4: Working with Activity component

Android apps are built as a combination of components that can be invoked individually. Each component is independent of the others, and can serve as an entry point through which the system or a user can enter the app. As such, an Android app may provide multiple entry points.

There are four different types of app components which an Android app can consist of. Each type serves a distinct purpose and has a distinct lifecycle that defines how it is created and destroyed.

- Activities

Activity represents a single screen with a user interface.

- Services

Service does not provide a user interface. It is a component that runs in the background to perform long-running operations or to perform work for remote processes.

- Broadcast Receivers

A broadcast receiver is a component that allows an app to listen and respond to system-wide broadcast announcements. This enables the system to deliver events to the app outside of a regular user flow.

- Content Providers

Content providers help an app manage access to data stored by itself, stored by other apps, and provide a way to share data with other apps.

These components are loosely coupled by the app manifest file – `AndroidManifest.xml` that describes them, and work together to form a cohesive user experience of the app.

Create the Calculator Activity

In this section, you will create a subclass of Activity to implement the calculator functions. The Activity class takes care of creating a window in which you can place a layout with the `Context setContentView(View)` method.

1. In Android Studio's **Project** window with the **Android** view selected from the drop-down list at the top, open **app > java** and right click on **you.package.name** from the main source set (the package name without brackets).
2. Select **New > Activity > Empty Activity** from the pop-up menu.
3. In the **Create New Activity** window, enter 'CalculatorActivity' for **Activity Name**, deselect **Generate Layout File** and select 'Java' as the **Source Language**, click **Finish**.

Android Studio automatically creates the CalculatorActivity class file and registers the required `<activity>` element in `AndroidManifest.xml`.

- In the new `CalculatorActivity`, add the following line of code to the `onCreate()` method to set the activity content from a layout resource:

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

Finding Views

Android system takes and inflates the layout XML file, and turns them into a full view hierarchy with the root of the layout at the top of the view tree. Figure 15 below illustrates partial view hierarchy of the `activity_calculator` layout:

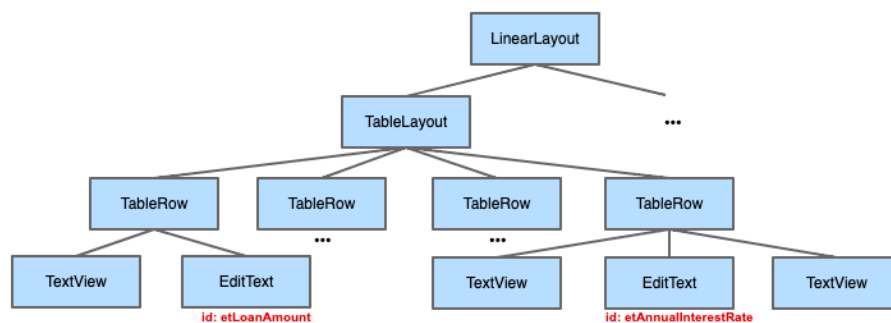


Figure 15 Partial View Hierarchy of `activity_calculator` layout

IDs (in red text) have been assigned to some of the views in the tree. These views can then be referenced from the code by calling the `findViewById(int)` method, where it traverses down the view hierarchy until it finds the first descendant view associated with the ID passed to the method and return the view, or null, if there is no matching view in the hierarchy.

- To retrieve instances of the defined views from the XML layout, declare the following member variables and initialise each in the `onCreate()` method:

```
private EditText etLoanAmount, etDownPayment, etTerm, etAnnualInterestRate;
private TextView tvMonthlyPayment, tvTotalRepayment, tvTotalInterest, tvAverageInterest;

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

    etLoanAmount = findViewById(R.id.etLoanAmount);
    etDownPayment = findViewById(R.id.etDownPayment);
    etTerm = findViewById(R.id.etTerm);
    etAnnualInterestRate = findViewById(R.id.etAnnualInterestRate);
    tvMonthlyPayment = findViewById(R.id.tvMonthlyPayment);
    tvTotalRepayment = findViewById(R.id.tvTotalRepayment);
    tvTotalInterest = findViewById(R.id.tvTotalInterest);
    tvAverageInterest = findViewById(R.id.tvAverageInterest);
}
```

① View binding eliminates the need to call `findViewById(int)` methods, allowing developers to easily write code that interacts with views and removes the risk of null pointer exceptions resulting from an invalid view ID. It is available in Android Studio 3.6 and above with Android Gradle plugin 3.6.0 or higher. Check <https://developer.android.com/topic/libraries/view-binding> for the setup and usage instructions.

Implement the Calculator functions

6. Add the following `calculate()` method to calculate and display the result of a compound interest loan from the user inputs:

```
private void calculate() {
    double loanAmount = Double.parseDouble(etLoanAmount.getText().toString())
        - Double.parseDouble(etDownPayment.getText().toString());
    double interest = Double.parseDouble(etAnnualInterestRate.getText().toString()) / 12 / 100;
    double noOfMonth = Integer.parseInt(etTerm.getText().toString()) * 12;

    double monthlyRepayment = loanAmount * (interest +
        (interest / (java.lang.Math.pow((1+interest), noOfMonth)-1)));
    double totalRepayment = monthlyRepayment * noOfMonth;
    double totalInterest = totalRepayment - loanAmount;
    double monthlyInterest = totalInterest / noOfMonth;

    tvMonthlyPayment.setText(String.format("%.2f", monthlyRepayment));
    tvTotalRepayment.setText(String.format("%.2f", totalRepayment));
    tvTotalInterest.setText(String.format("%.2f", totalInterest));
    tvAverageInterest.setText(String.format("%.2f", monthlyInterest));
}
```

and `reset()` method to clear the input fields - EditTexts and reset the result labels - TextViews to default value:

```
private void reset() {
    etLoanAmount.setText("");
    etLoanAmount.clearFocus();
    etDownPayment.setText("");
    etDownPayment.clearFocus();
    etTerm.setText("");
    etTerm.clearFocus();
    etAnnualInterestRate.setText("");
    etAnnualInterestRate.clearFocus();
    tvMonthlyPayment.setText(getString(R.string.default_result));
    tvTotalRepayment.setText(getString(R.string.default_result));
    tvTotalInterest.setText(getString(R.string.default_result));
    tvAverageMonthlyInterest.setText(getString(R.string.default_result));
}
```

Events Handling

Events occur when a user interacts with the interactive components of an app, such as button presses, screen touch etc. The Android framework maintains all events in a queue on a first-in first-out (FIFO) basis. There is more than one way to intercept these events, a collection of nested interfaces called Event listeners in the View class are more commonly used.

An event listener is an interface which contains a single callback method. These methods will be called by the Android framework when the View component to which the listener has been registered is triggered by user action. Table below lists some common listeners:

Event Listener & Description	Callback Method
OnClickListener called when a view is clicked.	onClick(View)
OnLongClickListener called when a view is clicked and held.	onLongClick(View)
OnFocusChangeListener called when the focus state of a view changed, e.g. the user navigates onto or away from the View component.	onFocusChange(View, boolean)
OnTouchListener called when an action qualified as touch event, such as a press, a release, or any movement gesture on the screen within the boundary of the View component is performed.	onTouch(View, MotionEvent)

Respond to the button clicks

- Enter the following code in the onCreate() method to create the OnClickListener in the form of anonymous class and register it to Calculate and Reset buttons to capture the click event to perform respective functions in the callback method:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...

    findViewById(R.id.btnCalculate).setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            calculate();
        }
    });
    findViewById(R.id.btnReset).setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            reset();
        }
    });
}
```

Alternatively, the OnClickListener can be implemented as a part of the Activity to avoid the extra class load and object allocation, and reduce the boilerplate when many views are listened to the same event.

Intent

An Intent is a messaging object that facilitates the communication between app components, except for content providers. The three fundamental use cases which require an Intent to be delivered are:

- Starting an activity by passing an Intent to methods: `startActivity()` or `startActivityForResult()`. The latter allows a separate Intent object to be received in the first activity's `onActivityResult()` callback when the second activity exists (is finished).
- Starting a service by calling the methods: `startService()` or `bindService()`. The latter creates a bound service, which is the server in a client-server interface where it allows components to bind to it, send requests, receive responses, and perform interprocess communication (IPC).
- Delivering a broadcast with the methods: `sendBroadcast()` or `sendOrderedBroadcast()`, which the latter delivers the broadcast to each receiver one at a time in order of priority. This provides a way to interfere the execution of the receivers with lower priority where one receiver is able to abort the broadcast by calling `abortBroadcast()`.

It contains an abstract description of an operation to be performed to the component that receives it. The primary pieces of information in an intent are:

- Component name - The fully qualified class name of the target component.
- Action - A string naming the action to be performed.
- Data - The URI of the data to be acted on and/or the MIME type of that data.
- Category - A string indicates the kind of component that should handle the intent.

An intent can also carry additional information that does not affect how it is resolved to an app component with the following:

- Extras - Key-value pairs of data to be delivered to the target component.
- Flags - Various sorts of flags which many instruct the Android system on how to launch an activity.

❗ It is good practice to define your own actions and keys for intent extras with your app's package name as prefix.

Intent Types

- Explicit intents supply either the target app's package name or a fully-qualified component class name. It is used to launch a specific app component, typically a particular activity or service within an app.
- Implicit intents do not name a specific component, but instead declare an action, data, and category¹⁰. Android system searches appropriate components by comparing to the intent filters declared in the manifest files of other apps on the device, start that component if one matches or display a dialog to let the user pick one to start if multiple are compatible. Refer to <https://developer.android.com/guide/components/intents-common> for the common intents such as capture a picture or video, compose an email, open a web page, initiate a phone call, and more.

¹⁰An explicit intent is always delivered to its target, regardless of the action, data, and category set (ignored by the system) when there is a component class name supplied in the intent.

⚠ If there are no apps on the device that can receive the implicit intent, an app that calls it will crash. Call `resolveActivity()` on the Intent object to verify if an app exists to receive the intent.

Start an Activity

In this section, you will write some code to start the `CalculatorActivity` from `MainActivity` after show up for 1 second.

8. In the file **app > java > your.package.name > MainActivity**, add the following code to the `onCreate()` method:

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

    TimerTask task = new TimerTask() {
        @Override
        public void run() {
            Intent intent = new Intent(MainActivity.this, CalculatorActivity.class);
            startActivity(intent);
        }
    };
    new Timer().schedule(task, 1000);
}
```

The above code snippet schedules a task to be executed after a delay of 1 second (in unit of millisecond). You might see errors because Android Studio cannot resolve the `Timer`, `TimerTask` and/or `Intent` class. To clear the errors, place the cursor on the error class, and press **Alt+Enter** (Option+Return on a Mac) to perform a quick fix. If a menu appears, select **Import class**.

The `Intent` constructor in the previous step takes two parameters, a `Context` which is the `MainActivity.this`, because the `Activity` class is a subclass of `Context`, and a `Class` of the app component, to which the system delivers the `Intent`, is, in this case, the activity to start.

① `Context` is an abstract class whose implementation is provided by the Android system. It allows access to app-specific resources and classes, information about the application environment, as well as up-calls for application-level operations such as launching activities and obtaining a system service. Read more about `Context` at <https://wundermanthompsonmobile.com/2013/06/context> by Dave Smith.

Styles and Themes

Styles and Themes on Android are declared in a style resource file in `res/values/`, usually named `styles.xml`. They can be used to separate the details of an app design from the UI structure and behaviour, similar to style sheets in web design.

A style is a collection of attributes, such as background color, font color, font size etc, that specify the appearance for an individual View.

A theme is a type of style that is applied to an entire app, activity, or view hierarchy¹¹ with the `android:theme` attribute. When a style is applied as a theme, every view in the app or activity, or the view and any child views of it¹² applies each style attribute that it supports and ignores the ones it does not. Themes can also apply styles to the non-view elements such as the status bar, navigation bar, or window background.

By default, the “Hello World” project created with Android Studio comes with a style applied as the theme for the entire app which looks similar to below:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
</style>
```

- A `<style>` element with a name that uniquely identifies the style and an optional parent attribute that can be used to extend an existing style from the framework or support library.

① The parent style from the framework is referenced with a prefix of `@android:style/` part, the ones without such as the above are from a library or own project. For the latter, the extending can be done with a dot notation, instead of using the parent attribute, and it is usually used for extending own project styles such as in step 9 of this lesson.

△ If a style contains both the parent attribute and dot notation, the parent style overrides any styles inherited through the dot notation.

- `<item>` defines the style attribute where the name attribute in each `<item>` specifies the XML attribute (refer to section **XML attributes (View properties)** on page 18 for detailed info) and the value in the element is the value for that attribute defined in name.

① Similar to the parent style, attribute names from own project or libraries do not use the `android:` prefix. That is used only for attributes from the Android framework.

^{11, 12}The `android:theme` attribute to a view in the layout is available from Android 5.0 (API level 21) and above, or Android AppCompat Support Library version 22.1 onwards.

In the following steps, you will create a style to be applied as the theme of the MainActivity to hide the app bar of the launcher screen.

9. In the file **app > res > styles > styles.xml**, add the following code to create a style which hides the app bar:

① If there are more than one styles.xml files, open the one without configuration qualifier.

```
<style name="AppTheme.NoActionBar">
    <item name="windowActionBar">false</item>
    <item name="windowNoTitle">true</item>
    <item name="android:statusBarColor">@android:color/transparent</item>
</style>
```

Note that the style is inherited from the AppTheme style with dot notation. With this way, you can continue inheriting styles many times by chaining on more names.

10. Apply the style to be the theme for MainActivity in **app > manifests > AndroidManifest.xml**:

```
<activity android:name=".MainActivity"
    android:theme="@style/AppTheme.NoActionBar">
```

① Previous steps show extending a style and overriding some of its attributes. This is the usual way to customise a theme to fit an app's brand. Alternatively, a theme from the framework or support library can also be applied directly if no customization is needed. For example, the Theme.AppCompat.Light.NoActionBar that is supplied by the Android Support Library could be applied to disable the app bar as well.

Lesson 5: Get Started with Testing

Testing your app is part of the app development process. By running tests against your app consistently, you can verify your app's correctness, functional behaviour, and usability before you release it publicly.

This lesson provides the basic information about the types of tests that can be constructed and run from Android Studio.

Test types and Location

The Android app module in an Android Studio project contains two source code directories (source sets) in which tests can be placed and organised according to the following types:

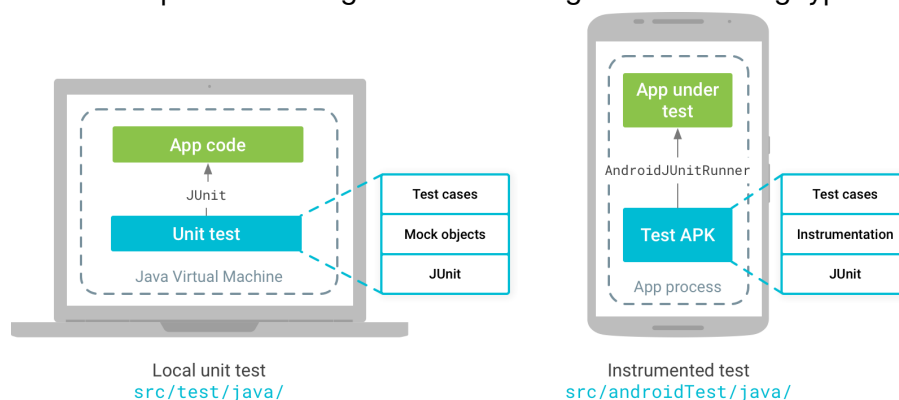


Figure 15 Local unit test vs Instrumented test

- **Local unit tests**

These tests are located in the `app / src / test` directory. It runs locally on the machine's Java Virtual Machine (JVM), allowing developers to set up a set of tests that has no Android framework dependencies or dependencies can be satisfied by using mock objects from the mocking libraries, like Mockito.

- **Instrumented tests**

These tests are located in the `app / src / androidTest` directory. They run on real or virtual devices and have access to Instrumentation information, such as the Context for the app under test and real implementation of an Android framework component: Parcelable, SharedPreferences and so on. Instrumentation is an Android framework component that provides additional monitoring and control over the application under tests.

The test code will be packaged into a separate Android Package (APK) and installed as part of the connected Android test Gradle task or when running tests in Android Studio. They will then be executed in the application's process by an instrumentation runner (AndroidJUnitRunner) as illustrated in the figure 15 above.

Use this approach to run tests where the JVM alone cannot validate the app's functionality, such as the tests that have complex Android dependencies that require a more robust environment like integration tests and end-to-end tests.

JUnit

JUnit is an open source unit testing framework for Java programming language that was developed by Kent Beck and Erich Gamma. Android Software Development Kit (SDK) supports JUnit for both unit testing and instrumentation testing. The initial supported version was JUnit3. However, as of Android 7.0 (API 24), JUnit3 style Java tests have been deprecated for the new JUnit4 style. This session just covers the basic usages of JUnit4, you can learn more about JUnit4 at <https://junit.org/junit4> or <https://github.com/junit-team/junit4/wiki>.

In JUnit terminology, a test case is a Java class that represents a set of tests to run. Any Java class with a zero-argument public constructor can serve as a test case with the `@RunWith(AndroidJUnit4.class)` annotation, and a test method is any public methods annotated with `@Test` in a test case that test something in some production code base. While methods annotated with `@Before` and `@After` perform the setup and teardown for each test method. Similar to the static methods annotated with `@BeforeClass` and `@AfterClass`, which invoked once for the entire test case.

JUnit4 also offers a set of rules which provides addition or redefinition of the behaviour of each test method in a test case for testing certain scenarios. For example, `TemporaryFolder @Rule` allows creation of files and folders that are deleted when the test method finishes, or the `ActivityTestRule @Rule` takes over the work of creating and destroying an instance of Activity. By using these rules, multiple test methods depending on the same behaviour can share and reuse the same isolated code easily.

JUnit4 supplies an `Assert` class with static assertion methods which usually accept parameters of an expected value, an actual value, and an optional failure message for value comparison and testing objects. For example, `assertEquals()` compares two parameters of the same type to test expected value in unit tests, `assertNotNull()` test the nullity of an object

Code Challenge

You have built a simple loan calculator app by completing this tutorial. However, app development does not end here. Any software or application needs constant attention, maintenance and support after its first release as part of the software development lifecycle (SDLC).

This tutorial is designed to focus on the fundamental aspects of the Android app development, functions such that purely related to Java programming such as result rounding or are less significant, for example the input validation, are omitted for simplicity, to place the attention on the subjects.

You may complete and enhance the app with the functions and new features from the list of items below:

- Input validation for empty, zero, or negative values.
- Result decimal rounding.
- Create a layout with better interface design for landscape mode.
- Add a toggle button to let users switch to simple interest loan calculation. (Current calculation is compound interest loan)
- History - able to view all previous calculations in a list.

Animation

Animations have grown to be an important part of the app design since the first introduction of mobile apps in smartphones. The Android framework provides two animation systems: View animation and Property animation. These are tweened animations, in which it tells Android to perform a series of simple transformations (transparency, rotation, size, position, and so on) to the content of a View. Both animations are viable options, but the Property animation, in general, is the preferred method to use, because it is more flexible and offers more features.

View Animation

View animation is the original animations based upon Animation base class distributed with Android since API level 1. These animations are dedicated to animating the movement and behaviour of widgets (a.k.a views), and can be applied to container that supports animating its contents, such as TextSwitcher and ViewFlipper, defined in either application code or XML file belongs in the `res/anim/` directory of an Android project. Among those animations are:

- AlphaAnimation (<alpha>) fades a widget in or out by making it less or more transparent (1.0 indicates a fully-solid and value of 0.0 indicates fully-transparent).
- RotateAnimation (<rotate>) controls the rotation of a widget.
- ScaleAnimation (<scale>) resizes a widget.
- TranslateAnimation (<translate>) moves a widget from a position to a new position¹.

① Regardless of the animation moves or resizes, the bounds of the View that holds the animation will not automatically adjust to accommodate it. Even so, the animation will still be drawn beyond the bounds of its View. That is if a widget is clickable, it will not be clicked in its new position after the animation is transformed, but rather in its original position. Clipping will occur if the animation exceeds the bounds of the parent View.

Group of Animations

Group of Animations can be played sequentially or simultaneously. The transformation of each individual animation in the <set> is composed together into a single transform. To make several transformations happen simultaneously, give them the same start time; to make them sequential, calculate the start time plus the duration of the preceding transformation.

Listen to the Lifecycle of a View Animation

A class which implements the AnimationListener interface may set to an animation via the `setAnimationListener()` method to listen for the lifecycle of an animation. The listeners will be invoked when the animation starts, ends, or repeats.

① Refer to the [Flying Bird demo](#) in the demo app for example of animating with View animation.

¹In Android's pixel screen coordinate system, an (X, Y) coordinate of (0, 0) represents the upper-left corner of the screen.

Property Animation (API 11)

Property Animation changes the properties' value² of any (Views and non-Views) object over time³, including ones that are not rendered to the screen. These animations can be declared programmatically or with XML file belongs in the `res/anim/` or `res/animator/`⁴ directory, and are achieved via the Animator subclasses below:

- **Animating with ValueAnimator**

A property animation contains two processes:

1. calculating the animated values.
2. setting those values on the object and property that is being animated.

ValueAnimator is the main timing engine for property animation that computes the values for the property to be animated over a period of time (Process 1). It contains the timing details, an Evaluator that tells how to calculate values for a given property, information about whether an animation repeats, and listeners that receive update events. However, ValueAnimator does not actually set those values (Process 2), it merely computing the different values based on time. An AnimatorUpdateListener can be used to listen to these values, and apply it to an object and property to animate. Snippet below shows an example of translating a TextView 100 pixels over 1000 ms at its x coordinate:

```
ValueAnimator animator = ValueAnimator.ofInt(0, 100);
animator.setDuration(1000);
animator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
    @Override
    public void onAnimationUpdate(ValueAnimator updatedAnimation) {
        int animatedValue = (int) updatedAnimation.getAnimatedValue();
        textView.setTranslationX(animatedValue);
    }
});
animator.start();
```

- **Animating with ObjectAnimator**

ObjectAnimator is a subclass of ValueAnimator that automatically applies/updates the computed values to a named property of a target object to animate. Thus, it makes the process of animating values on target objects much easier. The above example of translating a TextView can be simplified using the ObjectAnimator as shown in the snippet below:

```
ObjectAnimator animator = ObjectAnimator.ofInt(textView, "translationX", 0, 100);
animator.setDuration(1000);
animator.start();
```

²A field in an object, such as x, y coordinates of a View.

³The default duration of an animation is 300 ms.

⁴Recommended. To distinguish animation files that use the new property animation APIs from those that use the legacy View animation framework.

Note that the `ObjectAnimator` updates the values to the property by calling the setter function of the property name (the translationX) in the form of `set<PropertyName>()` (in camel case) of the target object (the `TextView`). Thus, an object property to be animated must be able to access with this setter method. If this setter method does not exist, developers may:

- Add the setter method to the class.
- Use a wrapper class to receive the value with a valid setter method and forward it to the original object.
- Fall back with `ValueAnimator`.

Also, if only one value for the `values...` parameter is specified in the `ObjectAnimator` factory methods (`ofInt()`, `ofFloat()`, `ofObject()`), it is assumed that this value to be the ending value of the animation. For example:

```
ObjectAnimator.ofFloat(targetObject, "propName", 100f);
```

Therefore, the object property that is animating must have a getter function to obtain the starting value of the animation in the form of `get<PropertyName>()`, e.g. `targetObject.getPropName()`.

① Refer to [Flipping Demo](#) in demo app for a full working example of `ObjectAnimator` declaring in XML resources.

• Animating with `ViewPropertyAnimator` (API 12)

`ViewPropertyAnimator` provides a simple way to animate several properties of a `View` in parallel, using a single underlying `Animator` object. It behaves much like an `ObjectAnimator`, because it modifies the actual values of the view's properties, but is more efficient when animating many properties at once by optimising the invalidate calls to take place only once instead of each animated property independently causing its own invalidation and skipping the overhead associated with reflection or JNI for turning a `String` denoting the name of a property into a call to getter and setter functions on the target object.

Also, the syntax of using this class is much simpler to write and read. It builds up an animation via a chained series of methods calls (method chaining), each returning an instance of `ViewPropertyAnimator`, starting with the call to `animate()` on a widget. For example, the snippet showing in the section **Choreographing Multiple Animations** below can be simplified into a single line of code below:

```
view.animate().x(100).y(100);
```

① It is not required to call `start()` at the end of the chain of method calls to start the animation. Once done declaring, `ViewPropertyAnimator` will automatically handle the details of configuring the underlying `Animator` class and starting it on the next update from the UI toolkit event queue.

❗ Refer to the [Text Switching demo](#) in the demo app for a full working example.

Listen to the Lifecycle of a Property Animation

Other than listen to the calculated values (`onAnimationUpdate()` of `AnimatorUpdateListener` called) on every frame⁵ of animation stated in the section **Animating with ValueAnimator** on previous page, the events such as start, end, repeat and cancel⁶ of an animation can be listened with the `Animator.AnimatorListener` interface, or `AnimatorListenerAdapter` class if only a subset of the mentioned events callback methods is needed.

Choreographing Multiple Animations

Multiple View / Property animations can be bundled to play together via `AnimationSet`. It is important to understand that the attributes that `AnimationSet` inherits from the `Animation` interface, some are applied to `AnimationSet` itself, some are pushed down to the children, and some are ignored. Detailed is as follows:

- `duration`, `repeatMode`, `fillBefore`, `fillAfter`: These properties, when set on `AnimationSet`, will be pushed down to all child animations (override the child values if one has specified in the child).
- `repeatCount`, `fillEnabled`: These properties are ignored.
- `startOffset`, `shareInterpolator`: These properties apply to the `AnimationSet` itself.

For example to simultaneously animating the x and y property of a view with `ObjectAnimators` (Property animation):

```
ObjectAnimator animX = ObjectAnimator.ofInt(view, "x", 100);
ObjectAnimator animY = ObjectAnimator.ofInt(view, "y", 100);

AnimatorSet animSetXY = new AnimatorSet();
animSetXY.playTogether(animX, animY);
animSetXY.start();
```

⁵Default frame refresh rate is 10 ms, but this rate is ultimately dependent on how busy the system is overall and how fast the system can service the underlying timer.

⁶A cancelled animation also calls `onAnimationEnd()`.

Table of Summary

	View Animation	Property Animation
Created with application code	yes	yes
Defined in XML resource files	yes	yes
Setup time	Less	More
Code complexity	Less	More
Target Object	Only Views	Views and non-Views ⁷
Type of animation	Alpha, Rotate, Scale and Translate.	Any property other than the four supported by View animation, such as color, and is extensible for custom type.
Change of Object	Only modified where the View was drawn, but not the View itself ⁸ .	The object itself is actually modified.

⁷Except for *ViewPropertyAnimator*, it animates only View objects.

⁸The changes were handled in the container of each View, because the View itself had no properties to manipulate.

The Interpolators

A set of interpolator classes are provided to determine how a transformation is applied over time of an animation. There are several implementations of the Interpolator interface which include:

- `LinearInterpolator` (default for View animation) indicates the animation should proceed smoothly from start to finish.
- `AccelerateInterpolator` indicates the rate of change of an animation begins slowly and speeds up over the duration of the animation.
- `DecelerateInterpolator` opposes the `AccelerateInterpolator`, which starts quickly and slows down towards the end.
- `AccelerateDecelerateInterpolator` (default for Property animation) starts the animation slowly, picks up speed in the middle, and slows down again at the end.
- `AnticipateInterpolator` whose rate of change starts backward then flings forward.
- `AnticipateOvershootInterpolator` whose rate of change starts backward, flings forward and overshoots the end value, then finally back to the end value.
- `BounceInterpolator` indicates the animation bounces at the end.
- `CycleInterpolator` repeats an animation for a number of cycles.
- `OvershootInterpolator` whose rate of change flings forward and overshoots the last value then comes back.
- `PathInterpolator` (API 21) traverses a Path that extends from Point (0, 0) to (1, 1).

To apply an interpolator to an animation, specify one of the Interpolator instances above to the `setInterpolator()` method on the animation or via the `android:interpolator` attribute in the animation XML file. Implement `TimeInterpolator` (API 11) to create your own transformation if none of the provided Interpolators suit the needs.

① Refer to the [Interpolator demo](#) in the demo app to examine the various transformations for each interpolator above.

Tutorial 2

The following tutorial teaches you how to build a simple weather app with the focus on building function with network operation. In this tutorial, you'll write code to make asynchronous networking calls to a web service from OpenWeatherMap, an online service weather data provider, to retrieve and display local weather data on the app's interface. Throughout these steps, you'll learn the platform's single-threaded model, AsyncTask framework, app permissions, and working with JavaScript Object Notation (JSON) data.

Lesson 1: Create a new module

① This lesson adds a new module to the existing project created on Tutorial 1. Skip this and proceed to exercise 1 on the next page if you choose to start with a new project.

1. Have the previous project opened in Android Studio, select **File > New > New Module**.
2. In the Create New Module window that appears, select **Phone & Tablet Module**, and then click **Next**.

① Besides the Phone & Tablet module, you can create the following modules: Android Library, Dynamic Feature, Instant Dynamic Feature, ~~Glass~~, Wear OS, Android TV, Android Things, and Java Library in Android Studio. The IDE automatically creates all the necessary files for the module, including any required dependencies such as using the Leanback library for Android TV module.

3. In the **Configure your new module** form, enter the following details:

Application/Library name: 'Weather'.

Module Name: The name of the directory which contains the collection of source files and build settings of the module, enter 'weatherapp'.

Language: Leave the Java option as it is, or select Kotlin if you want to include Kotlin language support in this project.

Minimum API level: set the lowest Android version that your app supports. Select **API 21: Android 5.0 (Lollipop)** for this project.

Leave the other options as they are, and click **Next**.

4. In the following **Add an Activity to Mobile** page, select **Empty Activity** template to use as main activity, and click **Next**.
5. In the **Configure Activity** page, check the **Generate Layout File** option if it is unchecked and leave the **Activity Name** and **Layout Name** as is, and click **Finish**.

Android Studio creates all the necessary files for the new module and syncs the project with the new module gradle files. Once the project sync completes, the new module appears in the **Project** window on the left. If you don't see the new module folder, make sure the **Android** view is selected from the drop-down list at the top of that window.

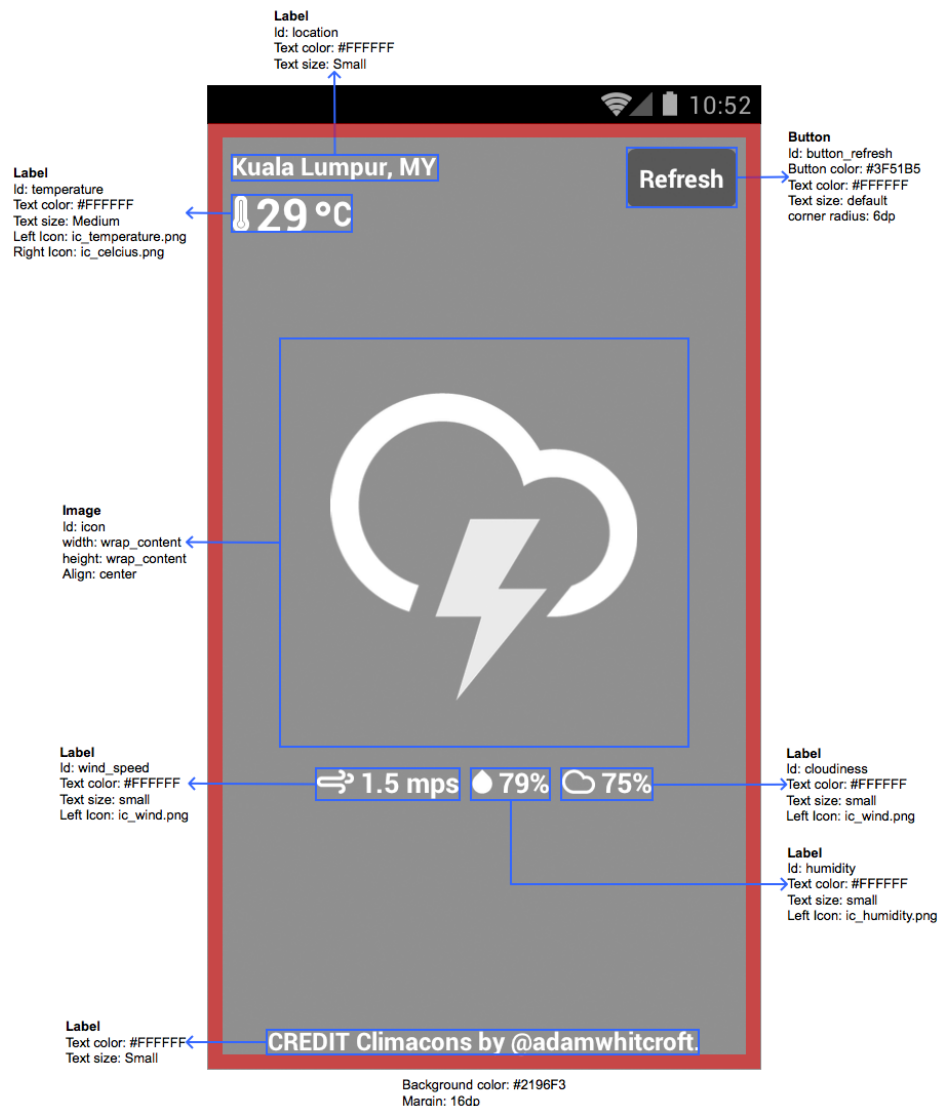
Exercise 1: Build the user interface


Figure 16 Weather app user interface

Open and modify the **weatherapp > res > layout > activity_main.xml** layout to create the user interface as shown in the figure 16 above. (Hint: refer to **Working with ConstraintLayout in Design editor** on page 17)

After the user interface is built, declare the following instance variables in the **weatherapp > java > your.package.name > MainActivity** class and initialise each in the `onCreate()` method:

```
private TextView tvLocation, tvTemperature, tvHumidity, tvWindSpeed, tvCloudiness;
private ImageView ivIcon;
private Button btnRefresh;
```

(Hint: refer to **Finding Views** on page 25)

Lesson 2: Perform network operations

By completing this lesson, you'll have the fundamental building blocks for creating Android apps that send simple requests and fetch content from a network source.

The Single-Threaded Model

By default, an Android app runs in its own Linux process with a single thread of execution (called the 'main' thread). This main thread is the only thread in which the application interacts with components from Android UI toolkit (all classes from `android.widget` and `android.view` packages), as such, it is sometimes called the UI thread. It is also the thread in charge of dispatching and managing events to appropriate user interface widgets, including drawing events.

Thus, performing long-running or slow processing computations and operations such as decoding a bitmap, accessing the disk (writing files to memory, database queries), or performing network requests or computationally intensive algorithms should be avoided from blocking the main thread. When the thread is blocked, no events can be dispatched, the application appears to hang for more than about 5 seconds, an Application not responding (ANR) dialog is presented to the user.

Additionally, the Android UI toolkit is not thread-safe, thus, all manipulation to the user interface must come from the UI thread.

Background / Worker Thread

Because of the single threaded model described above, long-running or slow processing computations and operations should be executed in a separate background thread using one of the thread implementation below:

1. Thread and Runnable

Runnable is a Java interface which should be implemented by any class whose instances are intended to be executed by a thread. The example code snippet below starts a thread and executes the code in the `run()` method of an anonymous object that implements the Runnable interface:

```
new Thread(new Runnable() {
    public void run() {
        Bitmap bm = loadImageFromNetwork("http://example.com/image.png");
        mImageView.post(new Runnable() {
            public void run() {
                mImageView.setImageBitmap(bm);
            }
        });
    }
}).start();
```


2. Message Handler

A Handler sends and processes Message and Runnable objects associated with a thread's MessageQueue. Each Handler instance is associated with a single thread and it is bound to the thread / message queue of the thread that is creating it.

Primarily used to enqueue an action to be performed on a different thread and able to schedule messages and runnables to be executed at some point in the future.

3. AsyncTask

AsyncTask is a convenient framework that is designed to be enabled and easy work around with the platform's single-threaded model. It performs the blocking operations in a separate thread and then publishes the results back to the UI thread, without requiring handling of threads and/or handlers.

The following steps use AsyncTask framework to send a simple request to retrieve local weather data from the OpenWeatherMap's web API:

1. In the file **weatherapp > java > your.package.name > MainActivity**, declare the following constant class variable and initialise it with the OpenWeatherMap API:

```
private static final String WEATHER_API =  
"https://api.openweathermap.org/data/2.5/weather?APPID=82445b6c96b99bc3ffb78a4c0e17fca5&mode=json&units=metric&id=1735161";
```

① The API above accesses current weather data for a location based on the city ID (in id parameter, visit <https://openweathermap.org/current> for a full list of parameters). For example the city ID for Kuala Lumpur is 1735161 and George Town is 1735106. List of city IDs can be downloaded at <http://bulk.openweathermap.org/sample/city.list.json.gz>.

Alternatively, you may register for the official free service offered by Malaysian Meteorological Department at <https://api.met.gov.my> to access the general weather forecast data in Malaysia.

and add the following code to create an inner class that inherits the AsyncTask:

```
private class WeatherDataRequest extends AsyncTask<Void, Void, String> {  
  
}
```

- Click and place the text cursor anywhere within the WeatherDataRequest inner class block, select **Code > Implement Methods** from the menu bar or press Ctrl+I to open the **Select Methods to Implement** dialog.

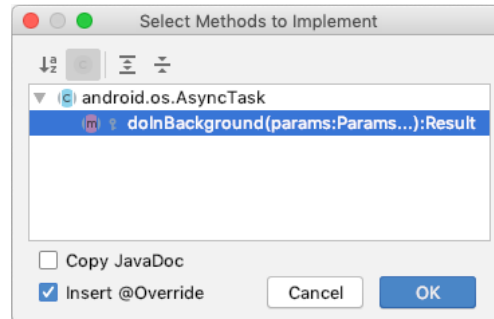


Figure 17 Select Methods to Implement dialog

Select **doInBackground(Params):Result** and click **OK**.

Android Studio creates stubs for the implemented methods, with the default return value for primitive types, and null value for objects.

① AsyncTask is an abstract class, thus it must be subclassed to be used and implement the `doInBackground(Params)` method. It is defined by 3 generic types, called `<Params, Progress, and Result>`, and goes through 4 steps, called `onPreExecute()`, `doInBackground(Params)`, `onProgressUpdate(Progress)`, and `onPostExecute(Result)`, when it is executed.

- The Google-endorsed API for HTTP operation in Android is to use the classic classes from java.net package which centred around HttpURLConnection. Enter the following code in the doInBackground(Params) method to construct a request to the OpenWeatherMap web server. It is invoked on the background thread immediately after onPreExecute() finishes executing.

```
@Override
protected void doInBackground(Void... voids) {
    try {
        URL url = new URL(WEATHER_API);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");
        conn.setConnectTimeout(15000);
        conn.setReadTimeout(15000);
        conn.connect();

        int responseCode = conn.getResponseCode();
        if (responseCode == HttpURLConnection.HTTP_OK) {
            BufferedReader bufferedReader = new BufferedReader(
                new InputStreamReader(conn.getInputStream()));
            String readline;
            StringBuilder stringBuilder = new StringBuilder();
            while ((readline=bufferedReader.readLine()) != null) {
                stringBuilder.append(readline);
            }
            return stringBuilder.toString();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

You might see errors because Android Studio cannot resolve some of the classes. To clear the errors, place the cursor on the error class, and press Alt+Enter (Option+Return on a Mac) to perform a quick fix. If a menu appears, select **Import class**.

- With the text cursor remains in the WeatherDataRequest inner class block, select **Code > Override Methods** from the menu bar or press Ctrl+O to open the **Select Methods to Override/Implement** dialog, and then select **onPostExecute(Result):void** and click **OK**.

❗ Alternatively, if you know the name of the override method, type the name and press enter (return in Mac) to create the method stub from the selection in the suggestion list.

A response contains the weather data in JSON format returns if the request in the previous step is successfully processed by the server. onPostExecute(Result) invoked on the UI thread after the background computation finishes, and receives the weather data as a result.

Write and View Logs with Logcat

Android SDK offers a log utility (`android.util.Log`) class to write messages of different verbosity into the log. The order from least to most is Error (`Log.e()`), Warn (`Log.w()`), Info (`Log.i()`), Debug (`Log.d()`), and Verbose (`Log.v()`). All these log methods take the parameters of a tag, a message, and an optional throwable, and it is a good convention to declare a constant for the tag.

The log messages will be displayed in the **Logcat** (a standalone program for viewing log messages) window in the Android Studio. As of Android Studio 2.2, the **Run** window also displays log messages for the current running app, but unlike **Logcat**, it is unable to configure the output display such as create filters, set priority levels, etc.

5. Add the following line of code to log the weather data in `onPostExecute(Result)` method:

```
@Override
protected void onPostExecute(String result) {
    super.onPostExecute(result);

    Log.d(getClass().getSimpleName(), "Weather data:" + result);
}
```

❶ To view the log in the **View > Tool Windows > Logcat**, you must build and run your app on a device or an emulator.

App Permissions

Android apps must request permission by including `<uses-permission>` tags in the app manifest to access sensitive user data (such as contacts and SMS), make use of certain system features (such as camera and internet), as well as perform any operations that would adversely impact the other apps (such as reading or writing another app's files).

Depending on the permission protection level, the system might grant the permission automatically if it is normal permissions, or require the app to prompt user to approve the dangerous permissions at runtime on device running Android 6.0 (API 23) or higher and have the app's `targetSdkVersion` is API 23 or higher. It may otherwise notifies the user of any app permissions at install time for devices running on versions lower than Android 6.0. The permission protection level can be checked at <https://developer.android.com/reference/android/Manifest.permission.html>.

6. Include the following permissions to allow app to access information on networks and make use of the network sockets in **weatherapp > manifests > AndroidManifest.xml**:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
</manifest>
```

Run the Task

7. Add the following `isNetworkConnected()` method to return the status of whether the device is connected to a network as a member method in `MainActivity` class:

```
private boolean isNetworkConnected() {
    NetworkInfo networkInfo =
        ((ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE))
            .getActiveNetworkInfo();
    return networkInfo != null && networkInfo.isConnected();
}
```

8. Execute the `WeatherDataRequest` task with the following code in the activity `onCreate()` lifecycle method:

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

    if (isNetworkConnected())
        new WeatherDataRequest().execute();
}
```

⚠ A task instance must be created and its `execute(Params)` method must be invoked on the main thread. Also note that the same instance can only be executed once.

① AsyncTask manages a thread pool from which it pulls the threads to be used by task instances. In Android 1.6 (API 4), tasks are executed in parallel with the pool of threads, and this was changed to serial execution on a single thread in Android 3.0 (API 11), however, developers may activate parallel execution by calling `executeOnExecutor()` with the `THREAD_POOL_EXECUTOR` constant.

The number of threads in the multiple-thread thread pool used to be able to climb up to as many as 128 threads. As of Android 4.4 (API 19), the max number of threads are limited to the number of CPU cores (processors) * 2 + 1, so on a dual-core device, the thread pool will cap at 5 threads.

Exercise 2: Working with JavaScript Object Notation (JSON) data

① This exercise requires **Exercise 1: Build the user interface** on page 36 to be completed to continue. It consists of steps to parse the weather data from the OpenWeatherMap web API's response and display it on the UI.

1. In the file **weatherapp > java > your.package.name > MainActivity**, add a method named getIcon which returns a weather icon resource ID by matching a weather code parameter (refer to the table below) as a member method in the WeatherDataRequest class.

Weather	Code	Drawable resource ID
Thunderstorm	200, 201, 202, 210, 211, 212, 221, 230, 231, 232	ic_thunderstorm_large
Drizzle	300, 301, 302, 310, 311, 312, 313, 314, 321	ic_drizzle_large
Rain	500, 501, 502, 503, 504, 511, 520, 521, 522, 531	ic_rain_large
Snow	600, 601, 602, 611, 612, 615, 616, 620, 621, 622	ic_snow_large
Clear Sky	800	ic_day_clear_large
Few Clouds	801	ic_day_few_clouds_large
Scattered Clouds	802	ic_scattered_clouds_large
Broken and Overcast Clouds	803, 804	ic_broken_clouds_large
Fog	701, 711, 721, 731, 741, 751, 761, 762	ic_fog_large
Tornado	781, 900	ic_tornado_large
Windy	905	ic_windy_large
Hail	906	ic_hail_large

JavaScript Object Notation (JSON) is a lightweight open-standard and language-independent format for storing and transporting data and is often used to transmit data between a server to clients such as web pages and mobile apps. The official Internet media type is application/json and JSON filename uses the extension of .json.

The JSON format is syntactically identical to the code for creating JavaScript objects. Data is written in name-value pairs: a name field in double quotes, a colon separator, and followed by a value, separated by commas for each pair. The value data types can be one of the following:

- Boolean: either of values true or false.
- Number: a signed decimal number that may contain a fractional part and may use exponential E notation. It makes no distinction between integer and floating-point.
- String: a sequence of zero or more characters delimited with double-quotation marks. It does support a backslash escaping syntax.
- Object: an unordered collection of name-value pairs delimited with curly brackets { } separated by commas.
- Array: an ordered list of zero or more values in square brackets [] with comma-separated elements, each of which may be of any type.
- Null: represent an empty value by word null.

Below is an example of the weather data in JSON format from the OpenWeatherMap web API's response:

```
{ "coord": { "lon": 101.69, "lat": 3.14 }, "weather": [ { "id": 801, "main": "Clouds", "description": "few clouds", "icon": "02n" } ], "base": "stations", "main": { "temp": 25, "pressure": 1012, "humidity": 94, "temp_min": 25, "temp_max": 25 }, "visibility": 9000, "wind": { "speed": 0.82, "deg": 353 }, "clouds": { "all": 20 }, "dt": 1573827529, "sys": { "type": 1, "id": 9446, "country": "MY", "sunrise": 1573772310, "sunset": 1573815430 }, "timezone": 28800, "id": 1735161, "name": "Kuala Lumpur", "cod": 200 }
```

- The following code parses the city name and country, wind speed, cloudiness, temperature, humidity, and weather condition id from the weather data, and set to the UI elements. Due to that only the main thread can interact with components from Android UI toolkit, this code snippet has to be called in the `onPostExecute(Result)` method which is invoked on the UI thread:

```
@Override
protected void onPostExecute(String result) {
    super.onPostExecute(result);

    Log.d(getClass().getSimpleName(), "Weather data:" + result);

    if (result != null) {
        try {
            final JSONObject weatherJSON = new JSONObject(result);
            tvLocation.setText(weatherJSON.getString("name") + ", "
                + weatherJSON.getJSONObject("sys").getString("country"));
            tvWindSpeed.setText(
                weatherJSON.getJSONObject("wind").getDouble("speed") + " mps");
            tvCloudiness.setText(weatherJSON.getJSONObject("clouds").getInt("all") + "%");

            final JSONObject mainJSON = weatherJSON.getJSONObject("main");
            tvTemperature.setText(String.valueOf(mainJSON.getDouble("temp")));
            tvHumidity.setText(mainJSON.getInt("humidity") + "%");

            final JSONArray weatherJSONArray = weatherJSON.getJSONArray("weather");
            if (weatherJSONArray.length() > 0) {
                int code = weatherJSONArray.getJSONObject(0).getInt("id");
                ivIcon.setImageResource(getIcon(code));
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
}
```

- Re-fetch the weather data on the Refresh button clicked. (Hint: refer to **Respond to the button clicks** on page 27 and **Run the Task** on page 41)

Code Challenge

You have built the backbone of a weather app by completing this tutorial. However, it is still a lack of the functions a minimal weather app should perform. This tutorial is designed to focus on the subjects related to performing the network operation, such as the single-threaded model, AsyncTask framework, app permissions, and how to work with JSON, by omitting other irrelevant functionalities.

You may complete and enhance the app with the functions and new features from the list of items below:

- Add a progress indicator to let users be aware of the retrieving of weather data.
- Prompt users on the scenario such as no connection, request failed, etc.
- Add a function to periodically refresh the weather data in a fixed interval e.g. 30 minutes when the Activity is active.
- Support multiple cities and allow users to select a city to view the current weather.
- Show the weather info of multiple cities on a Google map.
- Add weather forecast for the next 5 days. Refer to the 5 days weather forecast API document at <https://openweathermap.org/forecast5>.

Android Networking Libraries

Often times, writing code to perform network operation requires more than just simply download the content and display it on screen, things like threading (e.g. performing network request on a background thread and deliver the result to the main application thread a.k.a UI thread), retries or cancelation (e.g. device failed over from WiFi to mobile data mid-transaction), network requests scheduling and caching, data parsing and marshaling, debug and traceable, and so on, need to be properly handled.

Networking libraries designed to assist with this. It can be further subcategorized into HTTP client and Image loader libraries. Sections below describes some of the most used networking libraries in Android community and their features:

HTTP Client Libraries

- OkHttp (<https://square.github.io/okhttp>)

OkHttp is a third-party library developed by Square for send and receive HTTP-based network requests efficiently with the capable of:

- Synchronous and asynchronous calls.
- GZIP compression shrinks download sizes.
- HTTP/2 support allows all requests to the same host to share a TCP socket connection, or fallback to connection pooling when HTTP/2 is not available to reduce request latency.
- Response caching.
- Perseveres to recover from common connection problems silently.
- Attempt alternate addresses if multiple IP addresses are available and the first connect fails.

It is built on top of the Okio library, a library which tries to be more efficient about reading and writing data than the standard Java I/O libraries by creating a shared memory pool.

① As of Android 4.4 (API 19), HttpURLConnection is based on OkHttp library version 1.1.2 for its internal implementation. In version Android 5.0 and higher, the OkHttp version has been upgraded to 2.x.

- Retrofit 2 (<https://square.github.io/retrofit>)

Retrofit is a type-safe REST client for Android developed by Square. The library provides a powerful framework for interacting with REST APIs by turning them into Java interfaces and uses annotations to describe the HTTP request, and works with structured data.

By default, Retrofit only accepts OkHttp's ResponseBody type for its HTTP request body @Body and deserializes HTTP bodies into the same type. It can be configured to use the provided converters below or custom converters to serialise and deserialize data of different content-format:

- Gson: com.squareup.retrofit2:converter-gson
- Jackson: com.squareup.retrofit2:converter-jackson
- Moshi: com.squareup.retrofit2:converter-moshi

- Protobuf: com.squareup.retrofit2:converter-protobuf
- Wire: com.squareup.retrofit2:converter-wire
- Simple XML: com.squareup.retrofit2:converter-simplexml
- Scalars (primitives, boxed, and String): com.squareup.retrofit2:converter-scalars

① Starting from Retrofit 2.x, the library by default leverages OkHttp as the networking layer and is built on top of it.

- Volley (<https://github.com/google/volley>)

Volley is an HTTP client library developed by Google for making networking in Android apps faster and easier. It was first used by the internal team in Google Play app and introduced to developers at the Google I/O 2013 conference. It was released as an open source library available through Android Open Source Project (AOSP) repository and moved to standalone distribution as a Maven artifact in early 2017.

Besides doing everything that has to do with networking and consuming REST APIs like Retrofit library, Volley provides powerful customization abilities and flexible caching mechanism. The library also comes with inbuilt image loading support (refer to the next section **Image Loader Libraries**) with the ImageLoader class and a custom ImageView widget called NetworkImageView, together they are ideal to use for cases such as fetching a lot of images in RecyclerView, ListView or GridView.

For more information and learn how to work with Volley library, refer to the chapter Volley or the official guide at <https://developer.android.com/training/volley/index.html>.

① HTTP client libraries are not suitable for large download or streaming operations, since the responses are held in memory during parsing. Consider using DownloadManager and ExoPlayer for respective operations. Refer to more details about ExoPlayer at <https://developer.android.com/guide/topics/media/exoplayer>.

Image Loader Libraries

Image loader libraries are designed to help with asynchronously loading images from network (http:// or https://), local files, a ContentProvider, assets or resources in app. The most popular open-source Android loading libraries are Glide developed by Bumptech and officially endorsed by Google (<https://github.com/bumptech/glide>), Fresco by Facebook (<https://frescolib.org>) and Picasso by Square (<https://square.github.io/picasso>).

Each of these libraries take slightly different approaches but generally they provide easy to integrate implementation with images caching in memory or in storage, image transformation (down-sampling, resizing or cropping), display placeholder and error images, and aware of Activity lifecycle and cancel tasks when the view the image is loading into get recycled or moves offscreen / out of window.

Volley

Volley is an HTTP client library created by Google and first introduced to developers at the Google I/O 2013 conference which aims to make networking for Android apps easier and faster. In early 2017, Volley finally started being distributed as a Maven artifact that developers can add to the Android Studio projects' dependencies via a simple implementation statement.

```
dependencies {  
    implementation 'com.android.volley:volley:1.0.0'  
}
```

RequestQueue

At a high level, Volley is using a RequestQueue object to manage worker threads (a thread pool) for running the network operations (processing Requests that send to the queue), reading from and writing to the cache. On newer versions of Android (API 9 or higher), the actual HTTP stack to be used for doing the network I/O is delegated to HttpURLConnection. However, this could be configured if the developer wishes to have his own HttpStack implementation¹ as shown in the second initialization statement below and in the session of Setting up a RequestQueue with his own implementation of Cache and Network.

A new RequestQueue can be set up via the static newRequestQueue() helper method in the Volley class.

```
RequestQueue queue = Volley.newRequestQueue(context);  
  
// provide a custom HttpStack implementation  
RequestQueue queue = Volley.newRequestQueue(context, httpStack);
```

Setting up a RequestQueue with own implementation of Cache and Network

In the previous example, a RequestQueue is created by calling the convenience method of Volley.newRequestQueue(), taking advantage of Volley's default behavior. Developers may also create a RequestQueue explicitly to supply their own custom behavior.

A RequestQueue requires a network to perform transport of the requests, and a cache to handle caching. The standard implementations of these available in the Volley toolbox:

- BasicNetwork provides a network transport based on the preferred HTTP client. It must be initialized with the HTTP client an app is using to connect to the network.
- DiskBasedCache provides a one-file-per-response cache with an in-memory index.

A particular use case to set up a RequestQueue is when it involves loading a lot of images. When looking at the Volley.newRequestQueue() default implementation, notice that it uses default cache directory - Context.getCacheDir(), and according to the method explanation in the API document below:

“You should always have a reasonable maximum, such as 1MB, for the amount of space you consume with cache files, and prune those files when exceeding that space.”

which is not a good choice if a lot of images have to be cached. The code snippet below shows the steps involved in setting up a `RequestQueue` to use external cache directory with an incremental maximum cache size of 25 MB²:

```
private static final String DEFAULT_CACHE_DIR = "images";
...

// define cache folder
File rootCache = context.getExternalCacheDir();
if (rootCache == null) {
    // external cache directory is not available, rollback to context's cache directory
    rootCache = context.getCacheDir();
}
File cacheDir = new File(rootCache, DEFAULT_CACHE_DIR);
cacheDir.mkdirs();

// Set up the network to use HttpURLConnection as the HTTP client.
Network network = new BasicNetwork(new HurlStack());

Cache cache = new DiskBasedCache(cacheDir, 25 * 1024 * 1024); // 25MB cap

RequestQueue queue = new RequestQueue(cache, network);

// important step to kick off the cache processing and network dispatcher thread pool.
queue.start();
```

Requests

On the other hand, a request – in the form of a `Request` instance – embodies the URL to be retrieved, any additional information (e.g. HTTP headers), and the rules for parsing of raw response received from the server. The built-in request types that supported on Volley are:

- `StringRequest`

```
StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
    new Response.Listener<String>() {
        @Override
        public void onResponse(String response) {
            // Todo: process the response, such as update the UI with info.
        }
    },
    new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            // Todo: handle the error, such as prompt user the error dialog.
        }
    });
```

¹ See Volley's `HurlStack` on how to create own `HttpStack` at <https://android.googlesource.com/platform/frameworks/volley/+1a39583f0ee06329f7918ed9a4c7d0e7cd342917/src/main/java/com/android/volley/toolbox/HurlStack.java>

² Default Volley cache size is 5 MB.

- JsonRequest (JsonObjectRequest and JsonArrayRequest classes)

```
JsonObjectRequest jsonObjRequest
    = new JsonObjectRequest(Request.Method.GET, url, jsonReq,
new Response.Listener<JSONObject>() {
    @Override
    public void onResponse(JSONObject response) {
        // Todo: process the response, such as update the UI with info.
    }
},
new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        // Todo: handle the error, such as prompt user the error dialog.
    }
});
```

- ImageRequest

```
ImageRequest imageRequest = new ImageRequest(url,
new Response.Listener<Bitmap>() {
    @Override
    public void onResponse(Bitmap response) {
        // Todo: process the response, such as update the UI with info.
    }
},
0, // max width, zero for none
0, // max height, zero for none
ImageView.ScaleType.CENTER_CROP,
Bitmap.Config.RGB_565, // image format to decode the bitmap
new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        // Todo: handle the error, such as prompt user the error dialog.
    }
});
```

Request Customization

When all the ready-to-use built-in request types are not one that fits the requirement, developers may implement a custom Request via the below steps:

1. Create a class that extends the Request<T> class, where <T> represents the type of parsed response the request expects.
2. Supply the HTTP method, URL, and a Response.ErrorListener to the superclass constructor.
3. Override getHeaders() to return a Map of HTTP headers to inject into the request, null if there are none.
4. Override parseNetworkResponse() to process the raw data from the server and turning it into a Response wrapped around the actual data to be returned to the caller. Note that Volley calls parseNetworkResponse() from a worker thread to ensure that expensive parsing operations, such as decoding an image into a Bitmap, does not block the UI thread.
5. Override deliverResponse(), which most cases invoke a callback interface here to deliver the response back to a component. Volley calls this method on the main thread with the object returned in parseNetworkResponse().

The `GsonRequest` in network demo app (available at the Github repository, <https://github.com/CodePlay-Studio/android-network-demo>) is an example of custom request written by the author of Volley, Ficus Kirkpatrick, complement to the built-in `where the uses the legacy org.json classes from the Android SDK, which work, but are slow and clunky.`

Send a Request

To send a request, construct one and add it to the `RequestQueue` by calling the method `add()`. The request will then moves through the pipeline, get serviced by either one cache processing thread or a network dispatch thread, and has its raw response parsed and delivered as illustrated in the flow diagram below:

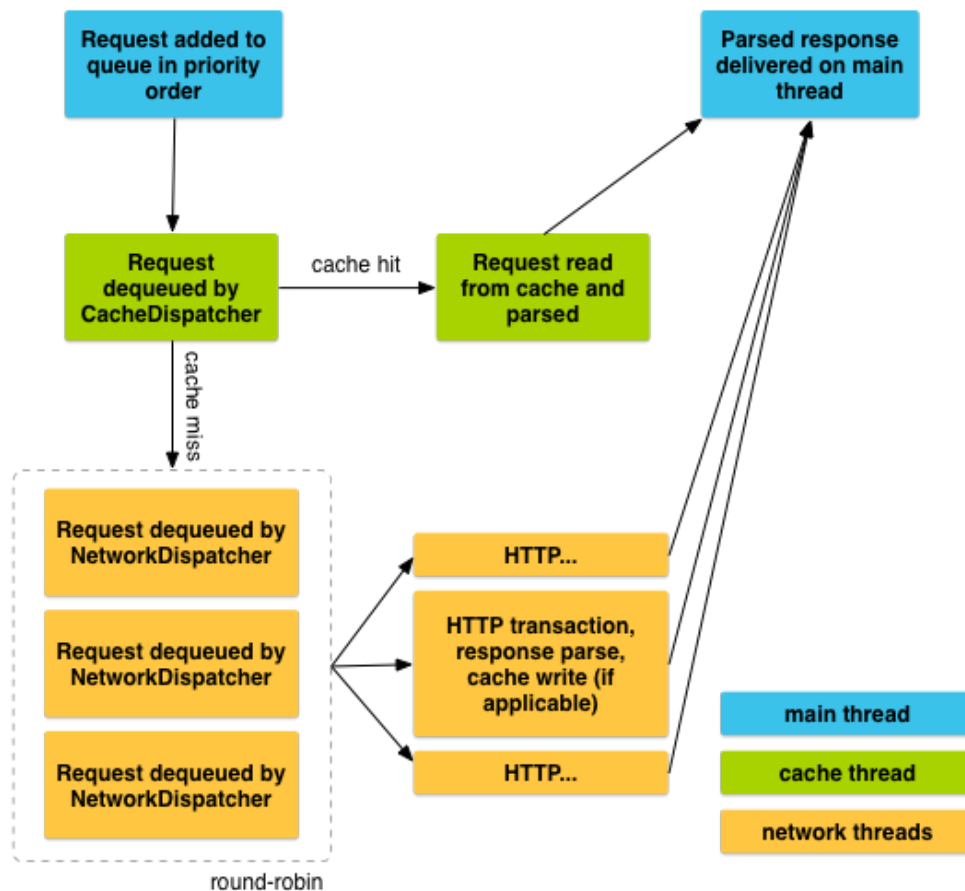


Figure 18 Volley flow diagram

Singleton Pattern

For one-time requests, developers can create the `RequestQueue` wherever it is needed and call `stop()` once the response or error has come back to stop the cache and network dispatchers. However, for applications that make constant use of the network, the recommended approach is to implement a singleton class that encapsulates `RequestQueue` and other Volley functionality that will last the lifetime of the apps.

```
public class VolleyManager {
    private static volatile VolleyManager instance;
    private final RequestQueue requestQueue;
    private final ImageLoader imageLoader;

    public static synchronized VolleyManager getInstance(Context ctx) {
        if (instance == null) {
            instance = new VolleyManager(ctx.getApplicationContext());
        }
        return instance;
    }

    private VolleyManager(Context ctx) {
        requestQueue = Volley.newRequestQueue(ctx);
        imageLoader = new ImageLoader(requestQueue, new LruBitmapCache(ctx));
    }

    public <T> void enqueue(Request<T> request, String tag) {
        if (!TextUtils.isEmpty(tag))
            request.setTag(tag);
        enqueue(request);
    }

    public <T> void enqueue(Request<T> request) {
        requestQueue.add(request);
    }

    public void cancelRequest(String tag) {
        if (!TextUtils.isEmpty(tag))
            requestQueue.cancelAll(tag);
    }

    public void loadImage(String url, ImageView iv, int placeholderDrawable,
        int errorDrawable) {
        imageLoader.get(url, ImageLoader.getImageListener(
            iv, placeholderDrawable, errorDrawable));
    }
}
```

Note that in this singleton implementation, a key concept is that the `RequestQueue` must be instantiated with the Application context, and not an Activity context. This ensures that the `RequestQueue` instance will last for the lifetime of the app.

ImageRequest vs ImageLoader

You have probably noted that the Volley singleton implementation in the previous session includes an `ImageLoader` instance, where early on, there is an `ImageRequest` in Volley toolbox to retrieve a `Bitmap` from a given URL.

`ImageRequest` handles the basic image request by giving back a `Bitmap` without any additional functions such as image caching³ and cancelling requests when views get recycled. This makes it good to use particularly only for occasional one-off requests for images.

`ImageLoader`, on the other hand, coordinates loading many images smartly with:

- Cancelling requests when views get recycled
- Having a memory cache for images, supplement the disk cache that Volley uses for responses
- Trying to minimise redraws of the UI when multiple images are decoded roughly simultaneously
- And so on

This makes `ImageLoader` ideal to use for cases such as fetching a lot of images in something like `RecyclerView`, `ListView` or `GridView`.

NetworkImageView

Volley introduced a custom `ImageView` widget called `NetworkImageView` to handle fetching an image from a URL as well as the lifecycle of the associated request. Snippet below shows the usage of a `NetworkImageView`:

```
<com.android.volley.toolbox.NetworkImageView
    android:id="@+id/networkimageview"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

```
mNetworkImageView = (NetworkImageView) findViewById(R.id.networkimageview);
mNetworkImageView.setDefaultImageResId(R.drawable.placeholder);
mNetworkImageView.setErrorImageResId(R.drawable.error);
mNetworkImageView.setImageUrl(url, imageLoader);
```

³In the case of `ImageRequest`, Volley decides whether to cache the response image or not, based on headers "Cache-Control" and "Expires".

Persistent Storage

Android uses a file system that is similar to disk-based file systems on other platforms. The system provides several options to save app and user data:

SharedPreferences

An interface that provides a general framework with convenient mechanism to save and retrieve primitive and String values in the form of key-value pairs. A SharedPreferences object points to a file stored in the private folder, specific to the app, containing the key-value pairs and provides simple methods to read and write them. This object can be retrieved by calling one of these methods:

- `getPreferences(int mode)` for accessing preference data that are private to an Activity. This calls the underlying `getSharedPreferences(String, int)` method by passing in the current Activity's class name as the preferences file name.
- `getSharedPreferences(String name, int mode)` to access application-level preference data in a preferences file with the given 'name'. If the preferences file by this 'name' does not exist, it will be created when an Editor object is retrieved (`SharedPreferences.edit()`) and write to it (`Editor.commit()` or `Editor.apply()`, refer to **Write to SharedPreferences** section.).

❗ For the mode parameter in both methods, use `MODE_PRIVATE`. The `MODE_WORLD_READABLE` and `MODE_WORLD_WRITEABLE` modes have been deprecated since API 17 and starting from Android 7.0 (API 24), a `SecurityException` is thrown if these modes are used.

Read from SharedPreferences

To retrieve values from a preferences file, call the following methods from the SharedPreferences object by providing the key, and a default value to return if the key is not present.

- `getBoolean(String key, boolean defValue)`
- `getFloat(String key, float defValue)`
- `getInt(String key, int defValue)`
- `getLong(String key, long defValue)`
- `getString(String key, String defValue)`
- `getStringSet(String key, Set<String> defValues)`

For example:

```
// get a SharedPreferences object from within an Activity
SharedPreferences sharedPref = getSharedPreferences("com.packagename.prefs_user",
    Context.MODE_PRIVATE);
String username = sharedPref.getString("KEY_USERNAME", "unset");
String email = sharedPref.getString("KEY_EMAIL", "unset");
boolean allowBackup = sharedPref.getBoolean("KEY_BACKUP", false);
```

Write to SharedPreferences

To write to a preferences file, call `edit()` on the `SharedPreferences` object, it will return a new instance of the `SharedPreferences.Editor` interface, which contains a set of setters prefix with 'put' that mirror the getters on the parent `SharedPreferences` interface and methods to remove and commit the preference data. For example:

```
// get a SharedPreferences object from within an Activity
SharedPreferences sharedPref = getSharedPreferences("com.packagename.prefs_user",
    Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putString("KEY_USERNAME", "user");
editor.putString("KEY_EMAIL", "user@example.com");
editor.putBoolean("KEY_BACKUP", true);
editor.commit(); // or apply();
```

Note that the `Editor` instance must call `apply()` or `commit()` to have the data to be persisted on the preferences file. `apply()` added in Android 2.3 (API 9), changes the in-memory `SharedPreferences` object immediately but writes the data to disk asynchronously, while `commit()` writes the data to disk synchronously (calling it from main thread could result in blocking the thread from UI rendering).

① When read values, the second parameter of the get methods is the default value when the key of the preference to retrieve does not exist. While, when write values, the second parameter of the set ('put') methods is the value to store.

⚠ Do not save sensitive data with `SharedPreferences`. The preference data persisted with `SharedPreferences` are not encrypted and can be easily manipulated on a rooted device. Refer to the Preferences Manager app at <https://play.google.com/store/apps/details?id=fr.simon.marquis.preferencesmanager>.

① Do not confuse with Preferences library¹. This library provides a consistent user experience and interface inherent in the platform to integrate user configurable settings in an app. It uses a private `SharedPreferences` instance to persist data by default. For information about building a settings screen with Preferences library, read the Settings guide at <https://developer.android.com/guide/topics/ui/settings.html>.

¹The platform `android.preference` library is deprecated in Android 10 (API 29), and replaced by the `AndroidX Preference` library from Android Jetpack.

Files on Internal Storage

For each app, the system provides app-specific directories within the internal storage on device memory. These directories have the following characteristics:

- An app does not require any system permissions to read and write files in these directories.
- The files saved in these directories are removed when the user uninstalls the app.
- The system prevents other apps from accessing these storage locations.
- On Android 10 (API 29) and higher, these locations are encrypted.

These characteristics make it a good place to store sensitive data that only the app itself can access.

Access and Store Files

Android framework provides several methods to access and store persistent files in this storage location:

- Use the File API. For example:

```
File file = new File(context.getFileDir(), filename);
```

Note that the line of code above is calling `getFileDir()` method on a `Context` object. This method returns a `File` object with the absolute path pointing to the app-specific root directory on internal storage where private files created with `Context.openFileOutput(String name, int mode)` are stored.

- Use the I/O Streams to read and write data.

The `openFileInput(String name)` and `openFileOutput(String name, int mode)` in the `Context` class open a private file with the given 'name' associated with the Context's application package for reading and writing. The former returns an instance of `FileInputStream` or throws `FileNotFoundException` if the file does not exist, while the latter returns an instance of `FileOutputStream` and creates the file if it does not already exist. The following code snippet shows how to read and write to a file as a stream:

```
String filename = "note.";
String fileContents = "My first note.";
try (FileOutputStream fos = context.openFileOutput(filename, Context.MODE_PRIVATE)) {
    fos.write(fileContents.toByteArray());
}
```

²*Sample code snippet above is using Java try-with-resources construct

**FileOutputStream is meant for writing streams of raw bytes such as image data. For writing streams of characters, consider using `FileWriter`.

⚠ On devices that run Android 7.0 (API 24) or higher, passing `MODE_WORLD_READABLE / WRITABLE` file mode into `openFileOutput()` will cause a `SecurityException`. To allow other apps to access files stored in this directory within internal storage, use `FileProvider`. The detailed lesson can be found at <https://developer.android.com/training/secure-file-sharing>.

```
try (FileInputStream fis = context.openFileInput(filename);
    BufferedReader reader = new BufferedReader(
        new InputStreamReader(fis, StandardCharsets.UTF_8)) {
    String line;
    StringBuilder stringBuilder = new StringBuilder();
    while ((line = reader.readLine()) != null) {
        stringBuilder.append(line);
    }
    fileContents = stringBuilder.toString();
} catch (IOException e) {
    // Error occurred when opening a raw file for reading...
}
```

²Sample code snippet above is using Java try-with-resources construct²

Files on External Storage

Internal storage space tends to be small and limited. Android system provides app-specific and shared directories within external storage where an app can organize its files. The physical location of external storage can be a partition of its internal (built-in / non-removable) memory, or a removable storage media such as an SD card.

Because external storage resides on a physical volume that may not always be accessible, for example, when it has been mounted by the user on a computer, has been removed from device, or some other problem has happened, the volume should always be checked if it is accessible before trying to read or write by calling `Environment.getExternalStorageState()` as shown in the sample code below:

```
String state = Environment.getExternalStorageState();
if (Environment.MEDIA_MOUNTED.equals(state)) {
    // can read and write
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
    // can only read
} else {
    // neither can read nor write
}
```

²Java try-with-resources construct was added in Java 7. It is a try statement that declares one or more resources like a Java `InputStream` or a JDBC connection or any object that implements `AutoCloseable` and ensures that each resource is closed when the execution leaves the block.

The characteristics between app-specific and shared directories within external storage are summarised in the following table:

	App-specific directories	Shared public directories
Access Method	<code>Context.getExternalFilesDir(String type)</code>	<code>Environment.getExternalStoragePublicDirectory(String type)</code>
Directory creation	Automatically created if not exist.	Require to create the directory explicitly such as with <code>File.mkdirs()</code> if it does not yet exist.
Files removed on app uninstall	Yes	No
Permissions needed and Accessible by other apps	Any app holding <code>READ/WRITE_EXTERNAL_STORAGE</code> permissions can read or write to the files. *On Android 4.4 (API 19) or higher, an app does not need to request any storage-related permissions to access <u>app-specific</u> directories.	
Available through MediaStore	No, the system media scanner does not read files in these directories. However, the files can be added to the MediaStore content provider by calling <code>MediaScannerConnection.scanFile()</code> explicitly.	Yes, the system media scanner will scan through these directories and provide the contents to other apps through the MediaStore content provider.

Note that both access methods for app-specific and shared public directories receive a String parameter which indicates the sub-directory of the given type. It could be one of the following Environment class constant fields:

- `DIRECTORY_ALARMS`
- `DIRECTORY_AUDIOBOOKS`
- `DIRECTORY_DCIM`
- `DIRECTORY_DOCUMENTS`
- `DIRECTORY_DOWNLOADS`
- `DIRECTORY_MOVIES`
- `DIRECTORY_MUSIC`
- `DIRECTORY_NOTIFICATIONS`
- `DIRECTORY_PICTURES`
- `DIRECTORY_PODCASTS`
- `DIRECTORY_RINGTONES`
- `DIRECTORY_SCREENSHOTS`

By specifying the type of the directory, it ensures that the files are treated properly by the system. For example, a ringtone audio file is identified as a ringtone, but not alarm, music or podcast. Passing null returns the root/top-level directory of the specific locations within external storage. For the latter, it is equivalent to calling the `Environment.getExternalStorageDirectory()`, apps should not place files on the root directory of shared external storage, to avoid polluting the user's root namespace.

⚠ Direct access to shared public directories within external storage is deprecated on Android 10 (API 29) and higher. When an app targets this version (Build.VERSION_CODES.Q), the path returned from `Environment.getExternalStoragePublicDirectory()` is no longer directly accessible. Apps can continue to access shareable contents stored on external storage by using the platform's MediaStore API and Storage Access Framework, detailed guides can be found at <https://developer.android.com/training/data-storage/shared>.

SQLite Database

Saving data to a database is ideal for repeating or structured data, such as contact information. Android stores the database in an app's private folder on the device's internal storage. The following sections detail the standard implementation of an app that would need to create and manage its own private database.

⚠ This topic assumes that you are familiar with Structured Query Language (SQL) databases in general.

1. Define a Schema and Contract

Schema is a formal declaration of how the database is organized. It is reflected in the SQL statements that are used to create a database.

It would be helpful to create a companion class, known as a contract class, which explicitly specifies the layout of the schema in a systematic and self-documenting way. A contract class contains the constants that define names for URIs, tables, and columns. A good practice to organize a contract class is to put definitions that are global to the whole database in the root level of the class, and then create inner classes for each table, enumerates the corresponding table's columns.

```
public final class WeatherDataContract {
    // To prevent instantiating of the contract class.
    private WeatherDataContract() {}

    // Inner class that defines the table contents.
    public static class WeatherEntry implements BaseColumns {
        public static final String TABLE_NAME = "weather_entry";
        public static final String COL_CITY = "city";
        public static final String COL_COUNTRY = "country";
        public static final String COL_TEMP = "temperature";
        ...
    }
}
```

① By implementing the `BaseColumns` interface, the inner class inherits a field called `_ID` which some Android classes such as `CursorAdapter` expect it to be the primary key field. Although it is not mandatory, this helps to create a database that works seamlessly with the Android framework.

2. Construct SQL Data Definition Language (DDL) Statements

Once you have defined the database structure, you should have the DDL statements to create and maintain the database and its tables.

Data definition language (DDL) is a set of SQL commands that can be used to create and manipulate the structures of a database. Typical statements that create and delete a table of WeatherEntry are as below:

```
private static final String SQL_CREATE_WEATHER_ENTRY =
    "CREATE TABLE " + WeatherEntry.TABLE_NAME + " (" +
    WeatherEntry._ID + " INTEGER PRIMARY KEY," +
    WeatherEntry.COL_CITY + " TEXT," +
    WeatherEntry.COL_COUNTRY + " TEXT," +
    WeatherEntry.COL_TEMP + " INTEGER)";

private static final String SQL_DELETE_WEATHER_ENTRY =
    "DROP TABLE IF EXISTS " + WeatherEntry.TABLE_NAME;
```

3. Create a database using an SQL helper

The SQLiteOpenHelper class contains a useful set of APIs for managing an app's database. When this class is used to obtain a reference to a database, Android system performs the potentially long-running operations of creating and updating the database only when needed. To use this class, create a subclass that overrides the onCreate() and onUpgrade() callback methods, and optionally the onOpen() and onDowngrade() methods. For example:

```
public class WeatherEntryDbHelper extends SQLiteOpenHelper {
    public static final int DATABASE_VERSION = 1;
    public static final String DATABASE_NAME = "WeatherEntry.db";

    public WeatherEntryDbHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    public void onCreate(SQLiteDatabase db) {
        db.execSQL(SQL_CREATE_WEATHER_ENTRY);
    }

    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL(SQL_DELETE_WEATHER_ENTRY);
        onCreate(db);
    }
}
```


4. Create, Read, Update and Delete (CRUD) operations

To access the database, instantiate the subclass of SQLiteOpenHelper:

```
WeatherEntryDbHelper dbHelper = new WeatherEntryDbHelper(getContext());
```

And then, all you need is to call `getReadableDatabase()` for reading information from a database or `getWritableDatabase()` to put or modify information into a database, from the `SQLiteOpenHelper` instance. For example, snippet below inserts data into the database by passing a `ContentValues` object to the `insert()` method:

```
SQLiteDatabase db = dbHelper.getWritableDatabase();

ContentValues values = new ContentValues();
values.put(WeatherEntry.COL_CITY, city);
values.put(WeatherEntry.COL_COUNTRY, country);
values.put(WeatherEntry.COL_TEMP, temperature);

long newRowId = db.insert(WeatherEntry.TABLE_NAME, null, values);
```

To read from a database, use the `query()` method, passing it your selection criteria and desired columns:

```
SQLiteDatabase db = dbHelper.getReadableDatabase();

Cursor cursor = db.query(
    WeatherEntry.TABLE_NAME, // The table to query
    // The array of columns to return (pass null to get all columns)
    new String[] { BaseColumns._ID, WeatherEntry.COL_CITY, WeatherEntry.COL_TEMP },
    WeatherEntry.COL_COUNTRY + " = ?", // The columns for the WHERE clause
    new String[] { "Malaysia" }, // The values for the WHERE clause
    null, // grouping of rows
    null, // filter by row groups
    WeatherEntry.COL_CITY + " DESC"); // sort order
```

Room Persistence Library

The APIs available in the `android.database.sqlite` package are fairly low-level. Although they are powerful, they require a great deal of time and effort to use. For example:

- As the database schema changes, the affected SQL queries need to be updated manually. This process can be time consuming and error prone, as there is no compile-time verification of raw SQL queries.
- An amount of boilerplate code is needed to convert between SQL queries and data objects.

For these reasons, Google has launched the Room Persistence Library, as part of the Architecture components in Android Jetpack at the Google I/O 2017 conference. The library provides an abstraction layer over SQLite to make it easier to work with local databases by decreasing the amount of boilerplate code and verifying SQL queries at compile-time.

Room library defines databases, tables, and operations with the implementation of a set of annotations. The major components in Room are:

- Entity: A class that represents a table within the database.
- Data Access Object or DAO: An interface that contains the mapping of SQL queries to functions.
- Room database: The database holder which serves as the main access point to the underlying SQLite database (SQLiteOpenHelper). It uses the DAO to issue queries to the SQLite database.
- SQLite database: The database which is created and maintained by the Room persistence library on device storage.

Steps:

1. Update Room library dependencies
2. Define Entities
3. Create DAOs
4. Implement the Room database

Codelabs:

(Java) <https://developer.android.com/codelabs/android-room-with-a-view#0>

(Kotlin) <https://developer.android.com/codelabs/android-room-with-a-view-kotlin#0>

Additional Resources

Android API Differences Report

<https://developer.android.com/sdk/api_diff/{API_level}/changes.html>

*Replace the API_level in the link with an actual value, e.g 28, to generate a report that compares the provided API (To Level) to an older API (From Level) a level below it.

A report that details the changes in the core Android framework API between two API Level specifications. The report also includes general statistics that characterise the extent and type of the differences.

Android Apps Quality Guidelines <<https://developer.android.com/docs/quality-guidelines>>

This link offers a compact set of quality criteria and associated tests for an Android app to test against before publishing to ensure the app provides an excellent user experience and functions well on many devices. There are also guidelines for other form factors including Tablet app, Wear OS app, TV app, Auto app, Daydream app, and games.

Android Distribution Dashboard <<https://developer.android.com/about/dashboards>>

This page provides information about the relative number of devices running a given version of Android, have a particular screen configuration, or support a particular version of OpenGL ES on Google Play store.

① For more robust data to help you manage your app's targeting and understand the characteristics of your users' devices, refer to the app stats in the Google Play console.

Android Developers Blog <<https://android-developers.googleblog.com>>

This site provides the latest Android and Google Play news for app and game developers.

Now In Android <<https://medium.com/androiddevelopers>>

The official Android Developers publication on Medium.

AndroidX releases <<https://developer.android.com/jetpack/androidx/versions>>

The AndroidX releases page provides the release information for Android Jetpack libraries (Artifacts within the androidx namespace). This includes:

- Versions of each library.
- The chronological history of all releases.
- A code snippet with the default Gradle dependency declarations to use the artifacts.
- Links to the Kotlin and Java reference pages for the packages in each artifact.

Android · GitHub <<https://github.com/android>>

Official Android Github for all Android samples by Google Android team.

Android Code Search (<https://cs.android.com>)

A public code search tool for the Android Open Source Project (AOSP), where the latter is made up of a collection of git repositories which are managed together using the Repo tool - a Google-built repository-management tool that runs on top of Git. Because of this, most tools (such as github, gitweb, etc) cannot see the source code the way that it is laid out when it is checked out on the system. This tool makes it easier to search and navigate cross-references across the entire code base.

Android Developers YouTube channel (<https://www.youtube.com/user/androiddevelopers>)

Material Design (<https://material.io/components?platform=android>)

Documentation for a list of Material Design components.

Bibliography

All / Multiple Topics

Last Update: 15 Jan 2020

Mark L. Murphy. *The Busy Coder's Guide to Android Development*. Version 8.8. United States of America: CommonsWare, LLC. November 2017.

"Context" Android Developers Reference. -. 13 January 2020

<<https://developer.android.com/reference/android/content/Context.html>>

Animation

Last Update: 16 Dec 2019

"Animation and Graphics Overview" Android Developers API Guides. -. 14 November 2017

<<https://developer.android.com/guide/topics/graphics/overview.html>>

"android.view.animation" Android Developers Reference. -. 14 November 2017

<<https://developer.android.com/reference/android/view/animation/package-summary.html>>

Chet Hasse. "Introducing ViewPropertyAnimator" Android Developers Blog. 30 May 2011. 17 November 2017

<<https://android-developers.googleblog.com/2011/05/introducing-viewpropertyanimator.html>>

Volley

Last Update: 15 Dec 2019

"Volley Overview" Android Developers Guides. -. 23 October 2017

<<https://developer.android.com/training/volley>>

Persistent Storage

Last Update: 15 Jan 2020

"App data and files" Android Developers Guides. -. 13 January 2020

<<https://developer.android.com/guide/topics/data>>

"Environment" Android Developers Reference. -. 13 January 2020

<<https://developer.android.com/reference/android/os/Environment.html>>