CS4131
Mobile Application Development

Name: _____ ( ) Date: _____

Chapter 1: Introduction to Android Application Development

1.1 Why Android for App Development?

Portable smart devices such as smartphones, tablets and even watches, have been pervasive in our world today and will continue to grow and expand. As we can observe from those around us, the two key operating systems (OS) which consumers tend to choose for their portable smart devices will be the Android or the Apple OS.

In a global market share study, as of August 2021, Android holds the majority stake of mobile OS consumers globally (72.73%) and the trend shows that it will stay that way in the foreseeable future, hence, showing that Android will be used by consumers of majority of portable smart devices now and in the future. Furthermore, as you may have observed in Chapter 0, Android is in a way more developer friendly for developers who are familiar with Java and Python, two of the most common languages used by the tech industry in general today.

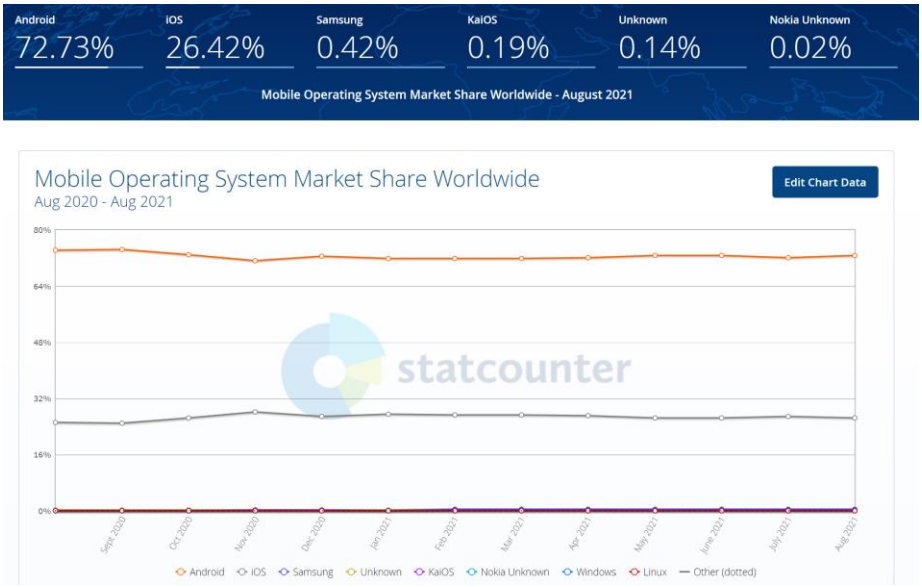


Figure 1: Statistics of Dominant Market Share of Consumers for Android Mobile

For this module, we will use Android Studio (minimum version 2020.3.1), with the Galaxy Nexus emulator run on the AVD (Android Virtual Device). The two primary languages which will be covered will be XML (Extensible Markup Language) for the user interface (UI) components and Kotlin, for the programming.

Note that:

- While it is NOT compulsory to use Kotlin and you may choose to revert to Java for the programming, the notes will cover the programming content in Kotlin as it is now and will be the preferred language to code Android apps.
- You are **expected to have completed** Chapter 0 as well as Lab 0, both of which the intent is to familiarize you with Android Studio and building a simple functional app.

1.2 User-Oriented App Design

This module will also primarily focus on user-oriented app design, and designing apps with the following considerations in mind:

- Did I consider possible constraints a mobile app may have compared to a desktop app?
- Can my app display properly in a wide range of screen sizes which is out in the market today? (phones, tablets, watches, TV)
- Can my app load its features fast?
- Does my app support its purpose in its expected scenarios and can it be easily used by a global audience or an audience with specific needs?
- Is my app simple to use, intuitive and user-friendly?

The first three questions cover the hardware and / or software limitations a mobile app may have as compared to a desktop app. Mobile devices typically will have less processing power than desktops, hence mobile applications should minimize heavy processing or have UI features like **splash or loading screens** to distract the user from slow processing times. As mobile devices are used on the go, we will need to consider aspects of **data persistence**, such that data is not lost when connectivity or battery life is lost. Also, as mobile devices are more prone to slow or erroneous clicks or typing, depending on its purpose, apps should **minimize unnecessary clicks or typing or considerations on the UI needs to be made to sufficiently space out UI components which require clicks or typing.**

The last two questions are more specific to the **user experience** of the app. Due to its portability and range of consumers portable smart devices can provide, it is important to consider the perspective of a wide range of users using your app, and if it can support its intended use for your typical consumer AND consumers **who may speak languages other than English**, and even consumers who may **require special visual or audio needs**. Also, on the topic of the user experience, it is vital to ensure that your app is **intuitive to use and user-friendly**. After all, what is the point of building an app which consumers do not like to use? Elements of the user experience, or in short, UX, will be covered in the later part of the module.

1.3 Android Fragmentation Problem

Other than considering user-oriented app design, there is also the issue of the Android consumers holding different versions of the Android OS simultaneously. The consequence is that if the app is built with an API tied to a later OS, the app may not work for a sizable population of Android device owners. Note that each OS is tagged to an API level, as seen in the following:

ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.0 Ice Cream Sandwich	15	
4.1 Jelly Bean	16	99.8%
4.2 Jelly Bean	17	99.2%
4.3 Jelly Bean	18	98.4%
4.4 KitKat	19	98.1%
5.0 Lollipop	21	94.1%
5.1 Lollipop	22	92.3%
6.0 Marshmallow	23	84.9%
7.0 Nougat	24	73.7%
7.1 Nougat	25	66.2%
8.0 Oreo	26	60.8%
8.1 Oreo	27	53.5%
9.0 Pie	28	39.5%
10 Android 10	29	8.2%

Figure 2: List of Android Versions and Corresponding API Levels

As of 2021, there are a total of twelve Android versions and the market share globally is such that a sizable 41.95% of users are using Android 10, while older versions are distributed amongst the rest of the users.

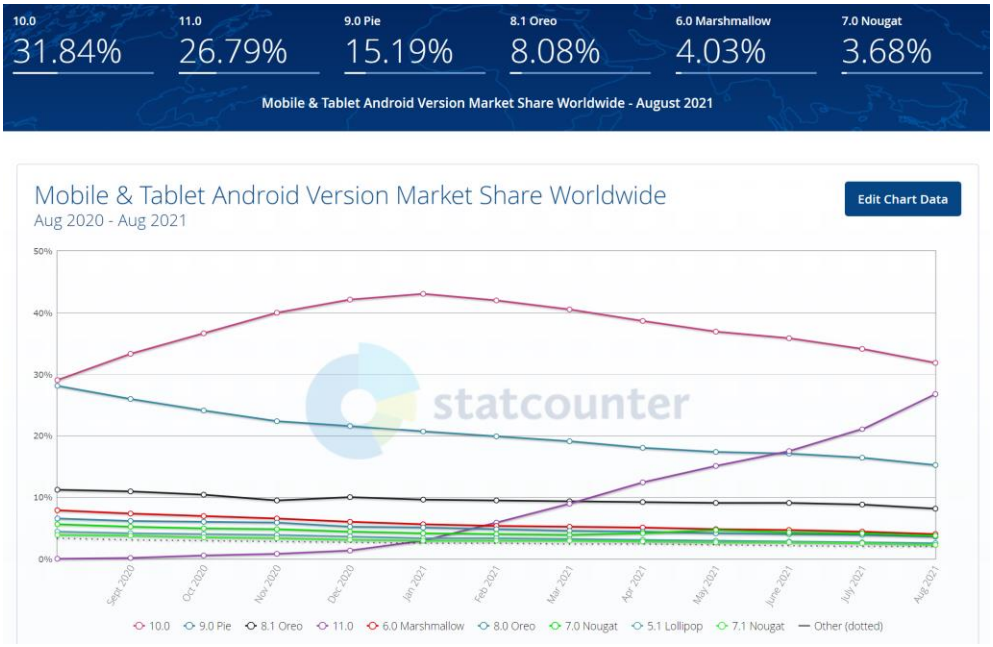


Figure 3: Android OS Worldwide Market Share (as of Aug 2021)

Hence, as we cannot control the versions of the OS users are holding, we can only cater to the majority. As later versions of the API can support apps which are made for earlier versions, we aim to build for at least 90% of your intended audience. Thankfully, when new projects are created on Android Studio, you are allowed to define which is the minimum OS is the app to be built for, and it goes to the extent of also showing how much of the market will your app run for:

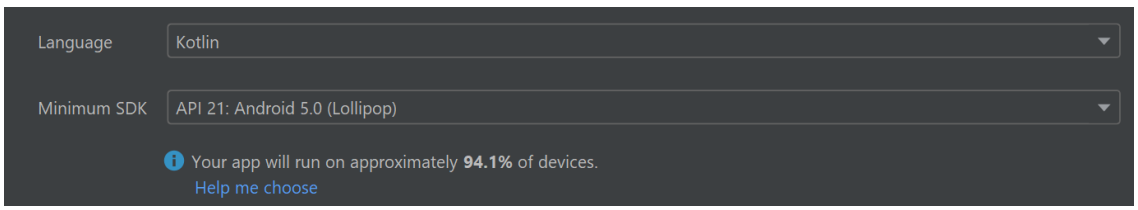
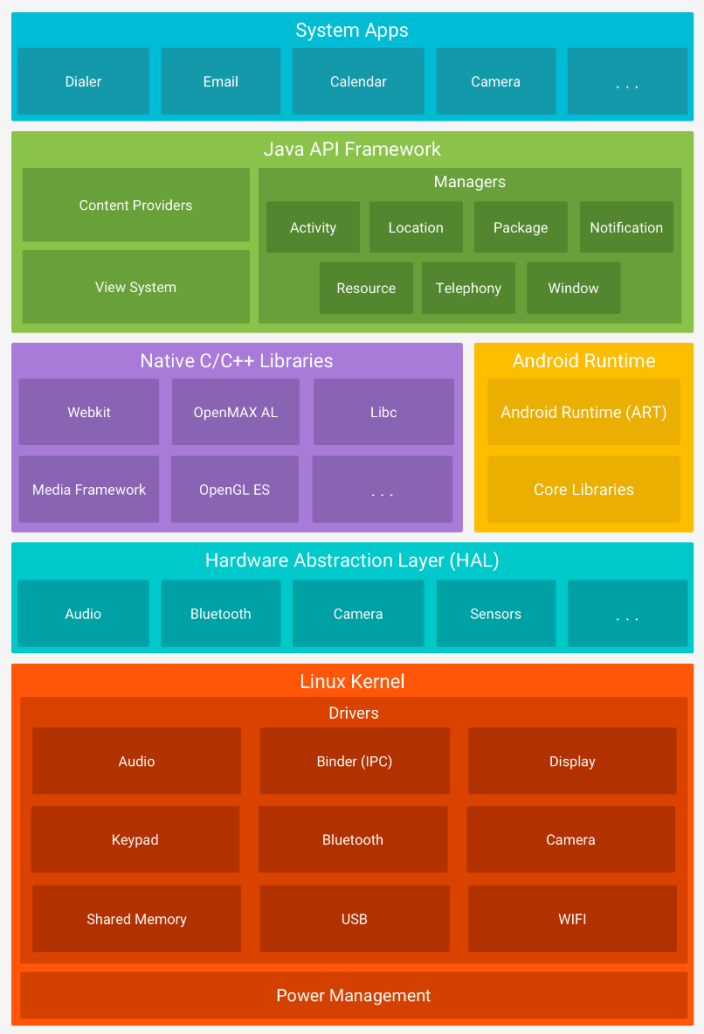


Figure 4: Android Studio Detailing Percentage can Run Your App

1.4 Components of the Android Platform

Android is an open-source, Linux-based software stack created for a wide array of devices. The following diagram shows the major components of the Android platform:



The foundation of the Android platform is the Linux kernel. For example, the Android Runtime (ART) relies on the Linux kernel for underlying functionalities such as threading and low-level memory management. Using a Linux kernel allows Android to take advantage of key security features and allows device manufacturers to develop hardware drivers for a well-known kernel.

The hardware abstraction layer (HAL) provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework. The HAL consists of multiple library modules, each of which implements an interface for a specific type of hardware component, such as the camera or blue-tooth module. When a framework API makes a call to access device hardware, the Android system loads the library module for that hardware component.

Figure 6: List of Android Versions and Corresponding API Levels

The entire feature-set of the Android OS is available to you through APIs written in the Java language. These APIs form the building blocks you need to create Android apps by simplifying the reuse of core, modular system components and services.

Android comes with a set of core apps for email, SMS messaging, calendars, internet browsing, contacts, and more. Apps included with the platform have no special status among the apps the user chooses to install. Hence, a third-party app can become the user's default web browser, SMS messenger, or even the default keyboard. The system applications function both as apps for users and to provide key capabilities that developers can access from their own app.

1.5 Building an Android Application

Before we start learning how to build apps using Android Studio proper, we need to appreciate what goes on in the backend of the building process.

So how do your Android apps compile and run? Ever since Android 7.0, Android apps use the ART as runtime, which includes a just-in-time (JIT) compiler with code profiling that continually improves the performance of Android apps as they run. The JIT compiler complements ART's current ahead-of-time (AOT) compiler and improves runtime performance, saves storage space, and speeds up application and system updates.

As the app code is compiled, the Java bytecode generated (**`.class`**) will be put through the Dalvik compiler creating the Dalvik bytecode (**`.dex`**). Originally, for Android applications running on 4.4 or below, the Dalvik Virtual Machine (DVM) is used to run the **`.dex`** files. However, in the current workflow, when the user runs the app, it triggers the ART to load the **`.dex`** files. The **`.oat`** files, which are the AOT binary for the respective **`.dex`** files, if available will be put through the ART directly then the app will run. If the **`.oat`** files are not available, the workflow will depend on whether it is “hot” code (executed frequently) or “cold” code (executed at most once). The JIT compiler will be more focused for use on “hot” code.

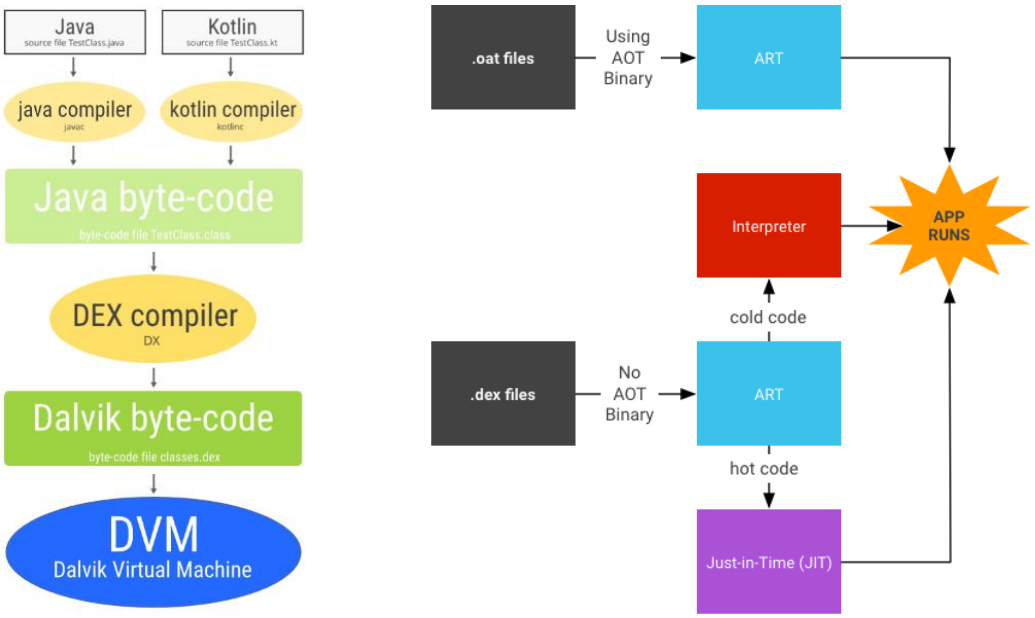


Figure 7: DVM Compilation Workflow (left); AOT and JIT Compilation Workflow

To build Android apps, we primarily need the Android Software Development Kit (SDK), which contains the necessary tools to create, compile and package Android apps. The Android SDK also contains the Android Debug Bridge (ADB), which is a tool that allows you to connect to a virtual or real Android device, for the primary purpose of debugging your app.

The primary IDE used for development of Android apps will be Android Studio, coupled with Gradle as an automated build tool to assist in the development and deployment of the app. Android’s configuration files, for both the development and UI, are all based on XML and the main language for development is primarily either Java or Kotlin.

1.6 Anatomy of an Android App

First, we will need to get familiar with different terminology used in the development process of an Android app.

Android Manifest

Before Android can start an application component, it needs to learn that the component exists. Hence, apps declare their components in a manifest file that is bundled into the Android package. The manifest file is a structured XML file which lists all the required permissions required for the application to work properly as well as define the minimum API level required, hardware and software requirements, etc.

On Android Studio, the Android manifest file is within the compulsory manifest package and always named AndroidManifest.xml for any application. As your app is developed with different UI components or packages drawn from the API for use, Gradle will automatically update the manifest file with the configurations and links required. Manual edits may be made to the manifest file in the case of declaring more activities, services or linking multiple activities together. **Note that you find if any of your defined additional activities or services do not run, it is most likely because you did not declare the additional activities or services in the manifest.**

Application Context

While working with Android app development, you will come across the Context class or articles which talk about the context of the application. The application context refers to the application environment and the process within which all its components are running. It allows applications to share the data and resources between various building blocks. The application context is uniquely identified on a device based on the package name of the application. **In other words, you can think of the Context as the current "device" your built apps are running on.**

Instances of the class android.content.Context provides the connection to the Android system which executes the application. It also gives access to the resources of the project and the global information about the application environment. For example, you can check the size of the current device display through the Context. Activities and Services (which will be touched upon next) all extend the Context class, and hence, can be directly used to access the Context. You can easulty obtain a reference to the context by calling **`Context.getApplicationContext()`** in Java or simply **`applicationContext`** in Kotlin.

Activities

An activity is a **single screen** of an app with a user interface, containing UI elements for the user to interact with the app. Each activity is assigned a window, where it will use up all the available screen space. However, windows can also be made smaller when deemed suitable such that the screen can display several activities simultaneously. Unlike programming paradigms where apps are launched with a main() method, the Android system initiates code in an Activity instance by invoking specific callback methods that correspond to specific stages of the **Activity Life Cycle**. ***More on activities and the activity life cycle will be covered in Lab 1.***

Service

A service is a background operation which does not require interaction with the user (hence, no UI supporting any given service). Services are useful for actions that you want to perform for a while, regardless of what is on the screen. Some examples including downloading a file or playing music while flipping through other apps. By default, services and activities run on the same main application thread. If a service is doing some processing that takes a while to complete, you would invoke a separate thread to run it. Hence, it is important to note that a service is neither a separate process (*it runs in the same process as the app it is part of),* nor a thread (*it is not a means to work off the main thread*).

Services can be either classified as bounded or unbounded. Unbounded services run in the background indefinitely until it is stopped by itself or a client, while bounded services are bounded by a component or client and can only be stopped until all clients unbind the service. A flowchart depicting the life cycle of bounded / unbounded services is seen on the next page:

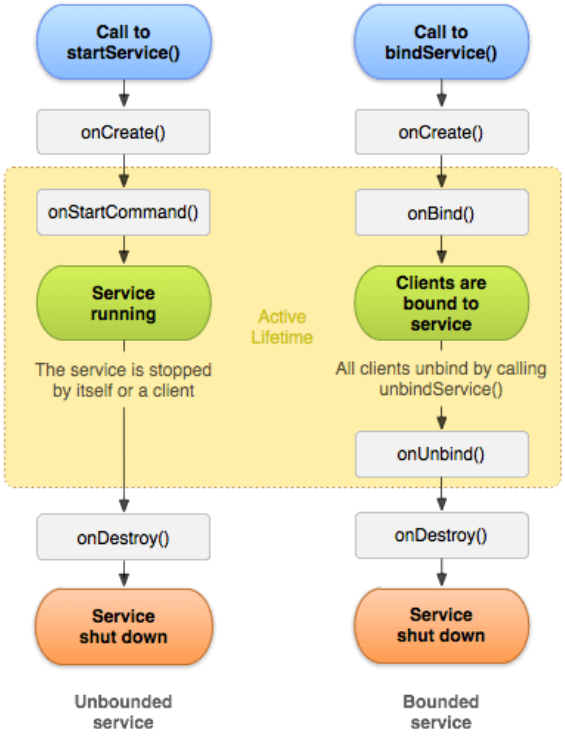
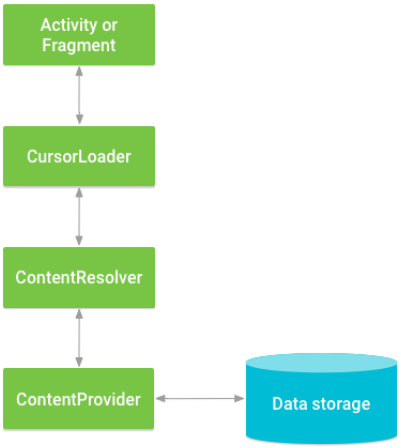


Figure 8: Bounded / Unbounded Service Lifecycle

Content Providers

A content provider defines a structured interface to application data. A provider can be used for accessing data within one application but can also be used to share data with other applications. Typically, you as a programmer will work with content providers in the scenario to access an existing content provider in another application or create a new content provider in your application to share persistent data with other applications.



Broadcast

A broadcast is a message sent or received from the Android system (like incoming SMS / calls, low battery notification, change of network notification, etc) or other Android apps. Each app can receive broadcasts through broadcast receivers, and apps can register to receive specific broadcasts, where the system will automatically route broadcasts to apps that have subscribed to receive a particular type of broadcast.

Generally, broadcasts can be used as a messaging system across apps and outside of normal user flow for the respective apps to perform certain tasks. However, careful management needs to be made to not abuse the opportunity to respond to broadcasts and run jobs in the background that can contribute to slow system performance.

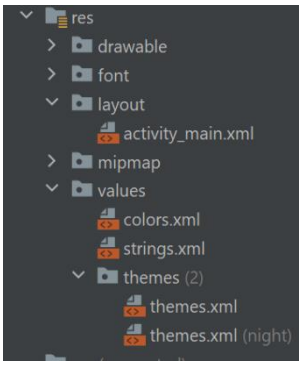
Intents

An intent is a messaging object you can use to request an action from another app component. Although intents facilitate communication between components in several ways, there are three fundamental usages:

1) Starting an activity by passing an intent which describes the activity to start and carries any necessary data required. This will allow the transfer of information, including objects, from one activity to another.
2) Starting a service by passing an intent describing the service to start and carries any necessary data required. An example of a service is downloading a file.
3) Delivering a broadcast by passing an intent which contains the specific message to send.

Resources

To maintain some form of ***strong consistency of elements*** such as colour, fonts and even strings throughout the entire app, Android Studio has a dedicated folder to place resource information. The enforcement of strong consistency is to allow the developer to edit the the "front-end" (interface and interface elements) of the app **without having to touch the application code** itself, and this will ultimately impact the user experience of the app itself, making it more appealing and intuitive to use.



Within Android Studio, there is a folder named "res", containing XML files with information on all these resources. You may also place media files such as MP3 or MP4 files into the res folder if some form of media is to be played (you will see an example in the next section). Even configurations for different devices or different languages can be placed into this folder to support the wide range of screen sizes and internationalization of the app.

1.7 Example Android Apps

To better explain the different components within the Android anatomy, you may refer to the following examples with their codes to get a hands-on understanding on how the different components of Android apps work. Note that these example code can help to provide a base for you to reuse to develop your own apps in the future, which is a common practice for developers today to use coding snippets contributed by a coding community.

It is strongly recommended that you develop the following Android projects with the code given, then play around with the app itself to get a better sense and understanding of how the different Android components explained in the previous section actually work.

Example App 1: ActivityToActivity

This app is mainly to demonstrate the use of Intents to pass data from one Activity to another, which some of you may recall was a big problem when coding desktop applications with JavaFX.

Each of the screens represent a separate Activity. When the quantity is increased and the "ORDER" button is pressed, the price is updated and the **data for the price and quantity is sent as an Intent to the next Activity** for the output.

You may also notice a "pop-up" like text saying "Moving to another Activity" This is known as a Toast object which can display pop-up like text temporarily when an event occurs.

strings.xml

```
<resources>
    <string name="app_name">Lecture 1: ActivityToActivity</string>
    <string name="textviewtext">Quantity</string>
    <string name="addbuttontext">+</string>
    <string name="subbuttontext">-</string>
    <string name="quantityTextViewtext">0</string>
    <string name="priceLabeltext">Price</string>
    <string name="priceTextViewtext">$0</string>
    <string name="orderButtontext">Order</string>
</resources>
```

MainActivity.kt

```
1  import android.content.Intent
2  import androidx.appcompat.app.AppCompatActivity
3  import android.os.Bundle
4  import android.view.View
5  import android.widget.TextView
6  import android.widget.Toast
7  import java.text.NumberFormat
8
9  class MainActivity : AppCompatActivity() {
10     var currentOrder = 0
11     var price = 4.50
12
13     override fun onCreate(savedInstanceState: Bundle?) {
14         super.onCreate(savedInstanceState)
15         setContentView(R.layout.activity_main)
16     }
17     fun changeOrder(view: View) {
18         when (view.id) {
19             R.id.addButton -> currentOrder++
20             R.id.subButton -> currentOrder--
21         }
22         val qtyTextView: TextView = findViewById(R.id.quantityTextView)
23         qtyTextView?.setText("" + currentOrder)
24     }
25     fun submitOrder(view: View) {
26         val priceTextView : TextView = findViewById(R.id.priceTextView)
27         priceTextView?.setText("" +
28        NumberFormat.getCurrencyInstance().format(
29                                        price * currentOrder))
30         val toast = Toast.makeText(applicationContext,
31                        "Moving to another Activity", Toast.LENGTH_SHORT)
32         toast.show()
33         val intent = Intent(this, Main2Activity::class.java)
34         intent.putExtra("order", "" + currentOrder)
35         intent.putExtra("total", "" + price * currentOrder)
36         startActivity(intent)
37     }
38 }
```

Explanation:

Lines 17 to 23	The method changeOrder is linked to both the + button and the – button, and will edit the currentOrder variable according to the view (or the control) detected. The qtyTextView will be changed to reflect the value of the currentOrder variable.
Lines 25 to 37	The method submitOrder is linked to the ORDER button. Upon tapping on the button, the priceTextView will show the price paid. A Toast object will display a message to the user stating the move to another Activity. An intent will be set up such that the Intent's destination is Main2Activity, with the currentOrder information tagged to "order" and the price information tagged to "total". This will allow the next Activity to retrieve the information based on the respective tags. The next Activity will be run with the Intent passed to the next Activity.

MainActivity2.kt

```
1  import android.os.Bundle
2  import android.widget.TextView
3  import androidx.appcompat.app.AppCompatActivity
4  import java.text.NumberFormat
5
```

```
6   class Main2Activity : AppCompatActivity() {
7       override fun onCreate(savedInstanceState: Bundle?) {
8           super.onCreate(savedInstanceState)
9           setContentView(R.layout.activity_main2)
10          val intent = intent
11          val order = intent.getStringExtra("order")
12          val total = intent.getStringExtra("total")
13                  (findViewById<TextView>(R.id.activity_main2_textView1)).setText(
14                          "You have ordered " + order + " coffee.\nTotal is " +
15                  NumberFormat.getCurrencyInstance()
16              .format(total!!.toDouble()))
17      }
18  }
19
```

Explanation:

Lines 10 to 16	The order and total variables will not contain information from the Intent by making use of the set-up tags to retrieve the respective information from the Intent. The activity_main2_textView1 will then display the concatenated string with the information.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/textviewtext"
        android:textAllCaps="true"
        android:textColor="@android:color/black"
        android:textSize="24sp" />

    <Button
        android:id="@+id/addButton"
        android:layout_width="96dp"
        android:layout_height="96dp"
        android:onClick="changeOrder"
        android:text="@string/addbuttontext"
        android:textSize="34sp" />

    <TextView
        android:id="@+id/quantityTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="24dp"
        android:layout_marginTop="24dp"
        android:text="@string/quantityTextViewtext"
        android:textSize="24sp" />

    <Button
        android:id="@+id/subButton"
```

```
        android:layout_width="96dp"
        android:layout_height="96dp"
        android:onClick="changeOrder"
        android:text="@string/subbuttontext"
        android:textSize="48sp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/priceLabeltext"
        android:textAllCaps="true"
        android:textColor="@android:color/black"
        android:textSize="24sp" />

    <TextView
        android:id="@+id/priceTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="24dp"
        android:layout_marginTop="24dp"
        android:text="@string/priceTextViewtext"
        android:textSize="24sp" />

    <Button
        android:id="@+id/orderButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="submitOrder"
        android:text="@string/orderButtontext"
        android:textSize="34sp" />

</LinearLayout>
```

activity_main2.xml

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

    <TextView
        android:id="@+id/activity_main2_textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="24sp" />
</RelativeLayout>
```

Note that in the Android Manifest, you need to manually add the declaration for Main2Activity. The following shows the last few lines of the Android manifest. The part which is highlighted is the part that should be added in.

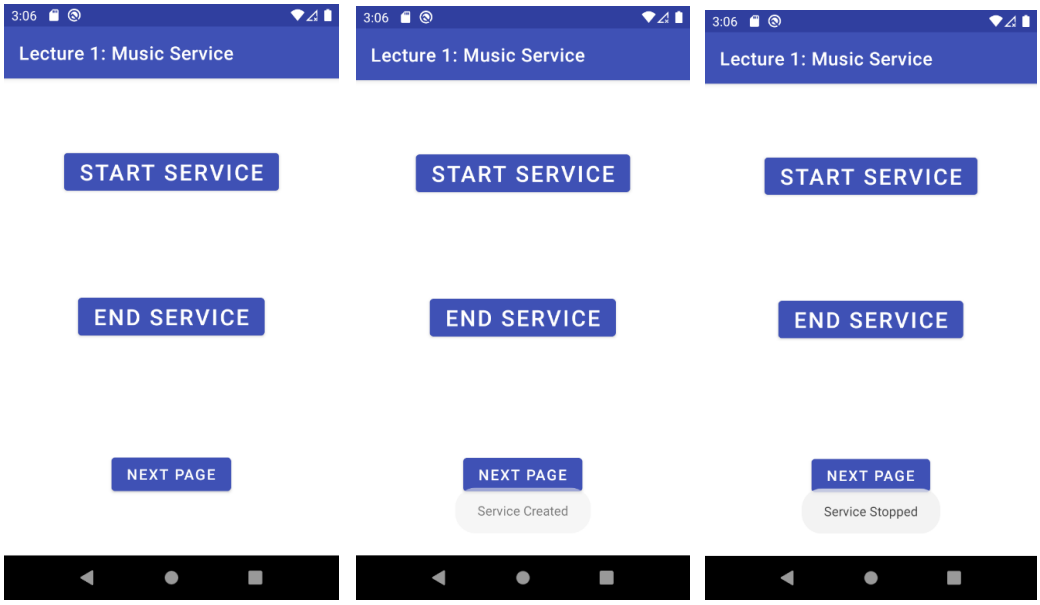
AndroidManifest.xml

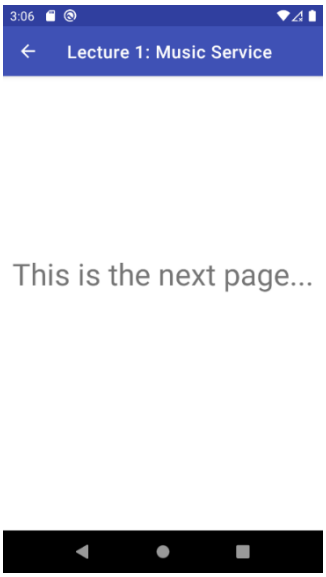
```
...
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".Main2Activity"
                  android:parentActivityName=".MainActivity"/>
    </application>
</manifest>
```

Note that the android:parentActivityName field indicates the parent of Main2Activity, which will place a “back” button on the app bar which goes back to MainActivity when it is touched (see the screenshots of the app)

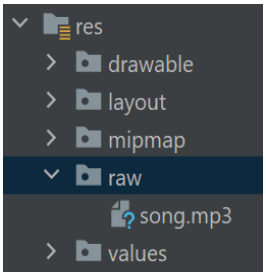
Example App 2: MusicService

This app is to demonstrate how a service bounded to buttons will work and that the service will keep running even if you minimize the app or move on to another activity within the same app.





When the “START SERVICE” button is tapped, a song from an MP3 file will be played **as a service**. This MP3 file is stored in the res folder under the folder named “raw” (you cannot name folders with multimedia elements as any other name) as seen on the right.



The service will be unbounded, meaning say that **the song will play, even in the background after minimising the app or after moving to a new Activity**, until the Service is terminated with the tap of the “END SERVICE” button.

strings.xml

```
<resources>
    <string name="app_name">Lecture 1: Music Service</string>
    <string name="startBtn">Start Service</string>
    <string name="endBtn">End Service</string>
    <string name="notRunning">Service Not Running</string>
    <string name="nextPg">Next Page</string>
    <string name="runningMsg">Service is Running</string>
</resources>
```

MainActivity.kt

```
1  import androidx.appcompat.app.AppCompatActivity
2  import android.os.Bundle
3  import android.content.Intent
4  import android.view.View
5  import android.widget.Button
6
7  class MainActivity : AppCompatActivity(){
8      var buttonStart: Button? = null
9      var buttonStop: Button? = null
10     var buttonNext: Button? = null
11     override fun onCreate(savedInstanceState: Bundle?) {
12         super.onCreate(savedInstanceState)
13         setContentView(R.layout.activity_main)
14         buttonStart = findViewById(R.id.buttonStart)
15         buttonStop = findViewById(R.id.buttonStop)
16         buttonNext = findViewById(R.id.buttonNext)
17     }
18     fun onClickBtn(view: View) {
19         when (view.getId()) {
20             R.id.buttonStart -> startService(
21                                                 Intent(this, MyService::class.java))
22             R.id.buttonStop -> stopService(
23                                                 Intent(this, MyService::class.java))
24             R.id.buttonNext -> {
25                 val intent = Intent(this, NextPage::class.java)
26                 startActivity(intent)
27             }
28         }
29      }
30 }
```

Explanation:

Lines 11 to 17	The method onCreate is run straight after the Activity is run. The attributes are set to the respective buttons.
Lines 18 to 29	The method onClickBtn is linked to all of the buttons in the Activity. The Service will be started / stopped depending on which button is tapped on. Note that an Intent must be used as a means to store the information required to startService(), stopService() or startActivity().

MyService.kt

```
1  import android.app.Service
2  import android.widget.Toast
3  import android.content.Intent
4  import android.media.MediaPlayer
5  import android.os.IBinder
6
7  class MyService : Service() {
8      lateinit var myPlayer: MediaPlayer
9      override fun onBind(intent: Intent?): IBinder? {
10         return null
11     }
12     override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
13         Toast.makeText(applicationContext,
14                                "Service Created", Toast.LENGTH_LONG).show()
15         myPlayer = MediaPlayer.create(this, R.raw.song)
16         myPlayer?.setLooping(true)
17         myPlayer?.start()
18         return START_STICKY
19     }
20     override fun onDestroy() {
21         Toast.makeText(applicationContext,
22                                "Service Stopped", Toast.LENGTH_LONG).show()
23         super.onDestroy()
```

```
24          myPlayer.stop()
25      }
26  }
```

Explanation:

Lines 8 to 11	myPlayer is given the lateinit modifier to tell the compiler that there is no assign the value to the vaiable the moment the variable is declared. Note that in Kotlin, all variables declared must be initialized immediately. The onBind method in this context does nothing and has no impact on the app.
Lines 12 to 19	When the service is started, a Toast object will display that the service is created, which will create the MediaPlayer object and start playing. START_STICKY will tell the system to run the Service in the background even if the Activity is destroyed due to a memory problem or some other cases.
Lines 20 to 25	When the service is stopped (through the END SERVICE button), a Toast object will display that the service is stopped, and subsequently stop the service and MediaPlayer.

MainActivity.kt

```
1   import android.os.Bundle
2   import androidx.appcompat.app.AppCompatActivity
3
4   class NextPage : AppCompatActivity(){
5       override fun onCreate(savedInstanceState: Bundle?) {
6           super.onCreate(savedInstanceState)
7           setContentView(R.layout.activity_next)
8       }
9   }
```

activity_main.xml

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

    <Button
        android:id="@+id/buttonStart"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="74dp"
        android:onClick="onClickBtn"
        android:text="@string/startBtn"
        android:textSize="24sp" />

    <Button
        android:id="@+id/buttonStop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:onClick="onClickBtn"
        android:text="@string/endBtn"
        android:textSize="24sp" />

    <Button
        android:id="@+id/buttonNext"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="63dp"
        android:onClick="onClickBtn"
        android:text="@string/nextPg"
        android:textSize="16sp" />
</RelativeLayout>
```

activity_next.xml

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

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="200dp"
        android:layout_marginEnd="8dp"
        android:text="This is the next page..."
        android:textSize="34sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Note that in the Android Manifest, you need to manually add the declaration for MyService. The following shows the last few lines of the Android manifest. The part which is highlighted is the part that should be added in.

AndroidManifest.xml

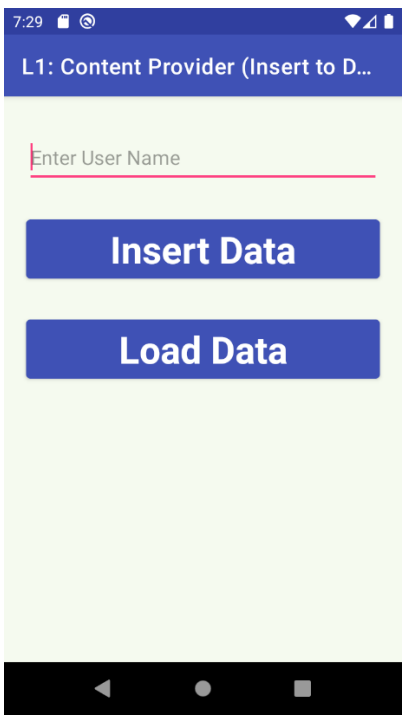
```
...
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".NextPage"
                        android:parentActivityName=".MainActivity" />
        <!-- Mention the service name here -->
        <service android:name=".MyService"/>
    </application>

</manifest>
```

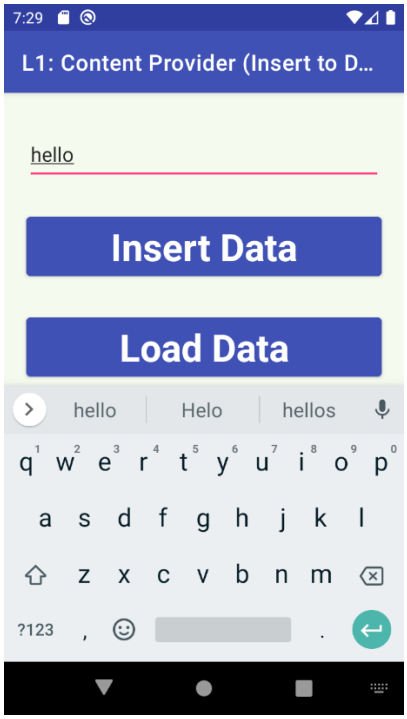
Note that the service component indicates Service which you wish to run in the app. Also note the syntax for a comment in XML which is: **<!-- ... -->**

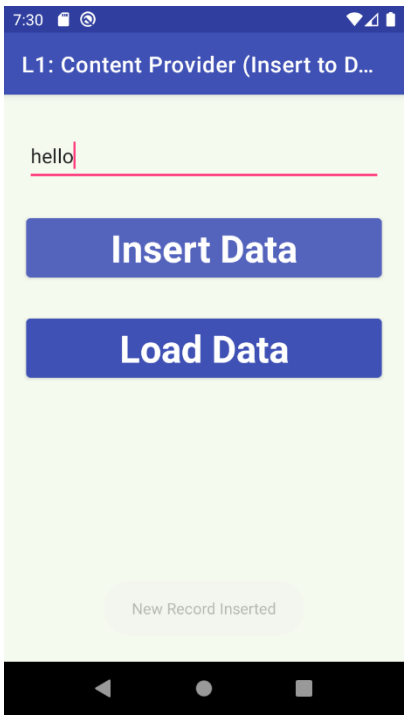
Example App 3 and 4: StudentContentProvider

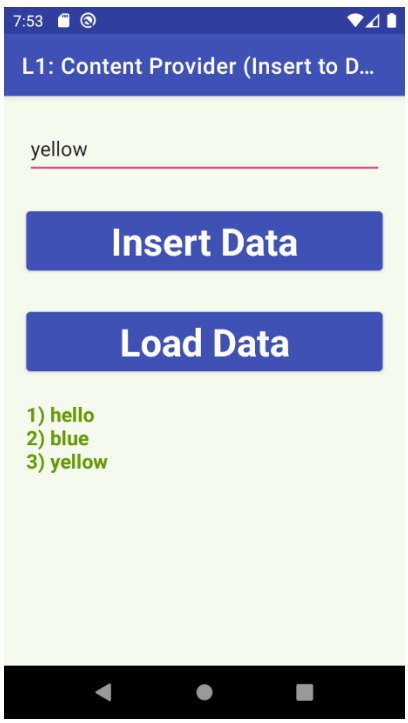
This app is mainly to demonstrate how content providers serve as a central location to provide information across different apps.

Screenshots for StudentContentProvider App (Enter items into Database)





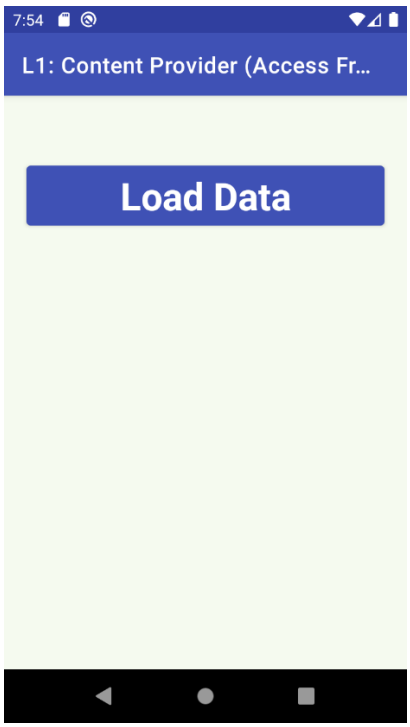


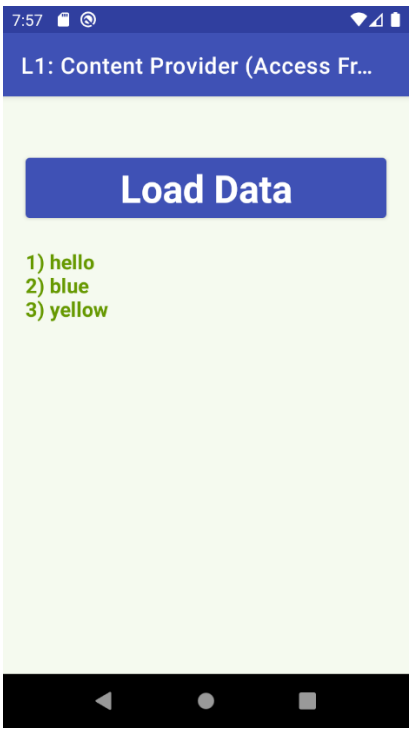


When the user taps on the TextView field, the Android keyboard will appear and will allow for typing of text. The app is coded is such a way that tapping anywhere else on the screen will cause the keyboard to close. When “Insert Data” is tapped on, the text typed in the TextView is saved into the database. When “Load Data” is tapped on, a query is run internally to fetch the database and the data will be processed for display.

Note that this data will be **persistent**, which means that the database content will stay even after the app is closed and reopened, or even after the phone is turned off then on. The only way the data will be wiped is that if an intentional effort is made to wipe the database from the phone.

Screenshots for StudentContentProviderTwo App (Access Data from Database)





Note that this app is a completely different app from the one before. When the app is run, the internal code will load the database then a query will be run internally to fetch the database and the data will be processed for display, similar to the previous app.

This also proves that the content provider provides a means for persistent data which can be shared across different apps.

Code for StudentContentProvider App (Enter items into Database)

strings.xml

```
<resources>
    <string name="app_name">L1: Content Provider (Insert to DB)</string>
    <string name="hintText">Enter User Name</string>
    <string name="heading">Content Provider In Android</string>
    <string name="insertButtontext">Insert Data</string>
    <string name="loadButtonText">Load Data</string>
    <string name="noRecords">No Records Found!</string>
</resources>
```

MainActivity.kt

```
1   import android.annotation.SuppressLint
2   import androidx.appcompat.app.AppCompatActivity
3   import android.os.Bundle
4   import android.widget.Toast
5   import android.widget.EditText
6
7   import android.content.ContentValues
8   import android.content.Context
9   import android.net.Uri
10  import android.view.MotionEvent
11  import android.view.View
12  import android.view.inputmethod.InputMethodManager
13  import android.widget.TextView
14
15  class MainActivity : AppCompatActivity() {
16      override fun onCreate(savedInstanceState: Bundle?) {
17          super.onCreate(savedInstanceState)
18          setContentView(R.layout.activity_main)
19      }
20
21      // Force the touch keyboard to hide when user touches anywhere on screen
22      override fun onTouchEvent(event: MotionEvent?): Boolean {
```

```
23          val imm = getSystemService(Context.INPUT_METHOD_SERVICE) as
24                          InputMethodManager
25          if(imm.isAcceptingText)
26               imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
27          return true
28      }
29      fun onClickAddDetails(view: View?) {
30          // Fetching text from user
31          val inputText = (findViewById<View>(R.id.textName)
32                                  as EditText).text.toString()
33          // Class to add values in the database
34          val values = ContentValues()
35          if (inputText.equals(""))
36              Toast.makeText(baseContext, "Please enter a valid username",
37                          Toast.LENGTH_LONG).show()
38          else {
39              values.put(MyContentProvider.name, inputText)
40              // Insert into database through content URI
41              contentResolver.insert(MyContentProvider.CONTENT_URI, values)
42              Toast.makeText(baseContext, "New Record Inserted",
43                          Toast.LENGTH_LONG).show()
44              val imm = getSystemService(Context.INPUT_METHOD_SERVICE) as
45                                  InputMethodManager
46              if(imm.isAcceptingText)
47                   imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
48          }
49      }
50      @SuppressLint("Range")
51      fun onClickShowDetails(view: View?) {
52          val imm = getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager
53          if(imm.isAcceptingText)
54                 imm.hideSoftInputFromWindow(currentFocus!!.windowToken,
55                                             InputMethodManager.RESULT_UNCHANGED_SHOWN )
56          val resultView = findViewById<View>(R.id.res) as TextView
57
58          // Create a cursor object of the content URI
59          val cursor = contentResolver.query(Uri.parse(
60                 "content://com.demo.user.provider/users"), null, null, null, null)
61
62          // Iterate the cursor to print whole table
63          if (cursor!!.moveToFirst()) {
64              val strBuild = StringBuilder()
65              while (!cursor.isAfterLast) {
66                  strBuild.append("""
67                              ${cursor.getString(cursor.getColumnIndex("id"))})
68                                ${cursor.getString(cursor.getColumnIndex("name"))}
69                                  """.trimIndent())
70                  strBuild.append("\n")
71                  cursor.moveToNext()
72              }
73              resultView.text = strBuild
74          } else { resultView.text = getString(R.string.noRecords)}
75      }
76  }
```

Explanation:

Lines 22 to 28	This onTouchEvent method will minimize the Android keyboard when the user taps anywhere on the screen. The Android keyboard is an example of a system service which you can call upon as the Context's INPUT_METHOD_SERVICE. The double exclamation mark (!!) is to indicate to the compiler that the currentFocus cannot return a null so that the compilation will go through without any errors.

Lines 29 to 49	The onClickAddDetails method is linked to the Insert Data button. It will take the input from the TextView, and if it is not an empty string, it will first store the input string into a ContentValues object, then insert the ContentValues object into the database itself through the contentResolver method. A Toast object will tell the user that the record has been inserted
Lines 50 to 76	The onClickShowDetails method is linked to the Load Data button. A Cursor object is created such that it will link to the correct database, **indicated by its content URI**, then the cursor object will iterate through the database, extract the relevant fields and append into a StringBuilder object which will then be used to display the loaded data on the app. Note the SupressLint("Range") on top of the method is to tell the compiler that there is definitely at least 1 record so that the compiler will not throw any "complaints" when compilation is done.

MyContentProvider.kt

```
1   import android.content.*
2   import android.database.Cursor
3   import android.database.sqlite.SQLiteDatabase
4   import android.database.sqlite.SQLiteException
5   import android.database.sqlite.SQLiteOpenHelper
6   import android.database.sqlite.SQLiteQueryBuilder
7   import android.net.Uri
8
9   class MyContentProvider : ContentProvider() {
10      companion object {
11          // Defining authority so that other application can access it
12          const val PROVIDER_NAME = "com.demo.user.provider"
13          // Defining content URI
14          const val URL = "content://$PROVIDER_NAME/users"
15
16          // Parsing the content URI
17          val CONTENT_URI = Uri.parse(URL)
18          const val id = "id"
19          const val name = "name"
20          const val uriCode = 1
21          var uriMatcher: UriMatcher? = null
22          private val values: HashMap<String, String>? = null
23
24          const val DATABASE_NAME = "UserDB"
25          const val TABLE_NAME = "Users"
26          const val DATABASE_VERSION = 1
27          const val CREATE_DB_TABLE =
28              (" CREATE TABLE " + TABLE_NAME
29                      + " (id INTEGER PRIMARY KEY AUTOINCREMENT, "
30                      + " name TEXT NOT NULL);")
31          init {
32              // Match content URI everytime user access table under content provider
33              uriMatcher = UriMatcher(UriMatcher.NO_MATCH)
34              // To access whole table
35              uriMatcher!!.addURI(PROVIDER_NAME, "users", uriCode)
36              // To access a particular row of the table
37              uriMatcher!!.addURI(PROVIDER_NAME,"users/*", uriCode)
38          }
39      }
40
41      override fun getType(uri: Uri): String? {
42          return when (uriMatcher!!.match(uri)) {
43              uriCode -> "vnd.android.cursor.dir/users"
44              else -> throw IllegalArgumentException("Unsupported URI: $uri")
45          }
```

```
46      }
47
48      // Creating the database
49      override fun onCreate(): Boolean {
50          val context = context
51          val dbHelper = DatabaseHelper(context)
52          db = dbHelper.writableDatabase
53          return db != null
54      }
55
56      override fun query(
57          uri: Uri, projection: Array<String>?, selection: String?,
58          selectionArgs: Array<String>?, sortOrder: String?
59      ): Cursor? {
60          var sortOrder = sortOrder
61          val qb = SQLiteQueryBuilder()
62          qb.tables = TABLE_NAME
63          when (uriMatcher!!.match(uri)) {
64              uriCode -> qb.projectionMap = values
65              else -> throw IllegalArgumentException("Unknown URI $uri")
66          }
67          if (sortOrder == null || sortOrder === "") {
68              sortOrder = id
69          }
70          val c = qb.query(db, projection, selection, selectionArgs, null,
71              null, sortOrder)
72          c.setNotificationUri(context!!.contentResolver, uri)
73          return c
74      }
75
76      // Adding data to the database
77      override fun insert(uri: Uri, values: ContentValues?): Uri? {
78          val rowID = db!!.insert(TABLE_NAME, "", values)
79          if (rowID > 0) {
80              val _uri = ContentUris.withAppendedId(CONTENT_URI, rowID)
81              context!!.contentResolver.notifyChange(_uri, null)
82              return _uri
83          }
84          throw SQLiteException("Failed to add a record into $uri")
85      }
86
87      override fun update(uri: Uri, values: ContentValues?, selection: String?,
88                          selectionArgs: Array<String>?): Int {
89          var count = 0
90          count = when (uriMatcher!!.match(uri)) {
91              uriCode -> db!!.update(TABLE_NAME, values, selection, selectionArgs)
92              else -> throw IllegalArgumentException("Unknown URI $uri")
93          }
94          context!!.contentResolver.notifyChange(uri, null)
95          return count
96      }
97
98      override fun delete(uri: Uri, selection: String?,
99                          selectionArgs: Array<String>?): Int {
100         var count = 0
101         count = when (uriMatcher!!.match(uri)) {
102             uriCode -> db!!.delete(TABLE_NAME, selection, selectionArgs)
103             else -> throw IllegalArgumentException("Unknown URI $uri")
104         }
105         context!!.contentResolver.notifyChange(uri, null)
106         return count
107     }
108
109     // Creating object of database to perform query
110     private var db: SQLiteDatabase? = null
111     // Creating a database
```

```
112     private class DatabaseHelper
113                         internal constructor(context: Context?) : SQLiteOpenHelper(
114         context, DATABASE_NAME,null, DATABASE_VERSION) {
115         override fun onCreate(db: SQLiteDatabase) { db.execSQL(CREATE_DB_TABLE) }
116         override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int,
117                                     newVersion: Int) {
118             // SQL query to drop a table having similar name
119             db.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
120             onCreate(db)
121         }
122     }
123 }
```

For the above code, a line-by-line explanation will not be done, but essentially, it sets up the database by initializing the relevant meta-data for the database and defining the create, query, insert, delete operations, all of which uses SQL to perform the relevant respective operations.

activity_next.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#168BC34A"
    tools:context=".MainActivity">

    <LinearLayout
        android:id="@+id/linearLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:orientation="vertical"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.13"
        tools:ignore="MissingConstraints">

        <EditText
            android:id="@+id/textName"
            android:layout_width="match_parent"
            android:layout_height="50dp"
            android:layout_marginStart="20dp"
            android:layout_marginEnd="20dp"
            android:layout_marginBottom="20dp"
            android:hint="@string/hintText" />

        <Button
            android:id="@+id/insertButton"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_marginStart="20dp"
            android:layout_marginTop="5dp"
            android:layout_marginEnd="20dp"
            android:layout_marginBottom="20dp"
            android:onClick="onClickAddDetails"
            android:text="@string/insertButtontext"
            android:textAlignment="center"
            android:textAppearance="@style/TextAppearance.AppCompat.Display1"
            android:textStyle="bold" />
```

```
        <Button
            android:id="@+id/loadButton"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_marginStart="20dp"
            android:layout_marginTop="5dp"
            android:layout_marginEnd="20dp"
            android:layout_marginBottom="20dp"
            android:onClick="onClickShowDetails"
            android:text="@string/loadButtonText"
            android:textAlignment="center"
            android:textAppearance="@style/TextAppearance.AppCompat.Display1"
            android:textStyle="bold" />

        <TextView
            android:id="@+id/res"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginStart="20dp"
            android:layout_marginEnd="20dp"
            android:clickable="false"
            android:ems="10"
            android:textColor="@android:color/holo_green_dark"
            android:textSize="18sp"
            android:textStyle="bold" />

    </LinearLayout>

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

Code for StudentContentProviderTwo App (Access items into Database)

strings.xml

```
<resources>
    <string name="app_name">L1: Content Provider (Access From DB)</string>
    <string name="loadButtonText">Load Data</string>
    <string name="noRecords">No Records Found!</string>
</resources>
```

MainActivity.kt

```
1   import android.annotation.SuppressLint
2   import android.net.Uri
3   import androidx.appcompat.app.AppCompatActivity
4   import android.os.Bundle
5   import android.view.View
6   import android.widget.TextView
7
8
9   class MainActivity : AppCompatActivity() {
10      var CONTENT_URI = Uri.parse("content://com.demo.user.provider/users")
11
12      override fun onCreate(savedInstanceState: Bundle?) {
13          super.onCreate(savedInstanceState)
14          setContentView(R.layout.activity_main)
15      }
16
17      @SuppressLint("Range")
18      fun onClickShowDetails(view: View?) {
19          val resultView = findViewById<View>(R.id.res) as TextView
20
21          // Create a cursor object of the content URI
22          val cursor = contentResolver.query(Uri.parse(
23                  "content://com.demo.user.provider/users"), null, null, null, null)
24
```

```
25          // Iterate the cursor to print whole table
26          if (cursor!!.moveToFirst()) {
27              val strBuild = StringBuilder()
28              while (!cursor.isAfterLast) {
29                  strBuild.append("""
30                              ${cursor.getString(cursor.getColumnIndex("id"))})
31                                ${cursor.getString(cursor.getColumnIndex("name"))}
32                                  """.trimIndent())
33                  strBuild.append("\n")
34                  cursor.moveToNext()
35              }
36              resultView.text = strBuild
37          } else {
38              resultView.text = getString(R.string.noRecords)
39          }
40      }
41  }
```

The above onClickShowDetails method performs the same functionality as the previous app which loads the data from the database tagged to the relevant URI.

activity_next.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#168BC34A"
    tools:context=".MainActivity">

    <LinearLayout
        android:id="@+id/linearLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:orientation="vertical"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.13"
        tools:ignore="MissingConstraints">

        <Button
            android:id="@+id/loadButton"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_marginStart="20dp"
            android:layout_marginTop="5dp"
            android:layout_marginEnd="20dp"
            android:layout_marginBottom="20dp"
            android:onClick="onClickShowDetails"
            android:text="@string/loadButtonText"
            android:textAlignment="center"
            android:textAppearance="@style/TextAppearance.AppCompat.Display1"
            android:textStyle="bold" />

        <TextView
            android:id="@+id/res"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginStart="20dp"
```

```
            android:layout_marginEnd="20dp"
            android:clickable="false"
            android:ems="10"
            android:textColor="@android:color/holo_green_dark"
            android:textSize="18sp"
            android:textStyle="bold" />

    </LinearLayout>

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

Note that in the Android Manifest **for both apps**, you need to manually add the declaration for the content provider (called the provider). The following shows the last few lines of the Android manifest. The part which is highlighted is the part that should be added in.

AndroidManifest.xml

```
...
        <provider
            android:name="com.example.studentcontentprovider.MyContentProvider"
            android:authorities="com.demo.user.provider"
            android:enabled="true"
            android:exported="true"></provider>
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Note that android:name field will refer to the **instance** of the MyContentProvider class and the android:authorities field can actually be self-defined. However, as observed in the code for both apps, **all apps accessing the same content must reference the same authority path**.

[References]

[1] Global statistics for market share of mobile OS (Aug 2021) https://gs.statcounter.com/os-market-share/mobile/worldwide

[2] Android Platform Architecture: https://developer.android.com/guide/platform

[3] Global statistics for market share of Android versions (Aug 2021) https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide

[4] Android Implementing ART Just-In-Time Compiler https://source.android.com/devices/tech/dalvik/jit-compiler

[5] XML Language Tutorial: https://www.w3schools.com/xml/

