Supplementary Note of Android Studio

1. Click “**File**” → “**New**” → “**New Project…**”. Select “**Empty Activity**” → “**Next**” → Name your project “**MyClock**” → package name is “**com.example.myclock**” → select “**Java**” under “Language” → make sure “Minimum SDK” is “**API 17: Android 4.2 (Jelly Bean)**” → “**Finish**”.

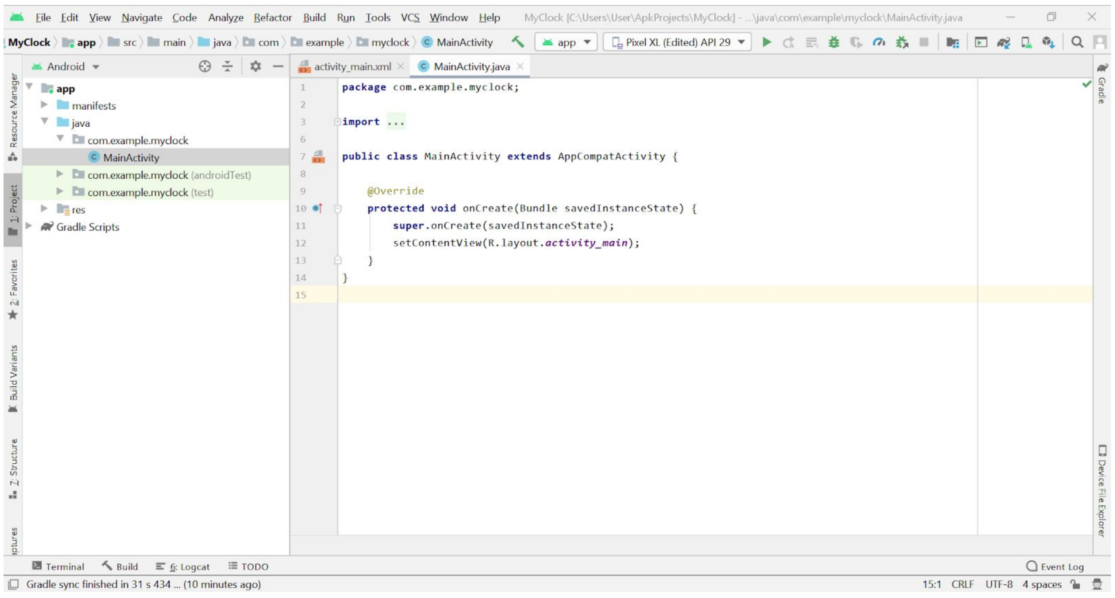
   Here is a description of each field:

   Application Name is the app name that appears to users. For this project, use “MyClock”

   Package name is used to create package name of your app. Basically, it is the package namespace for your app (following the same rules as packages in the Java programming language). Your package name must be unique across all packages installed on the Android system. For this reason, it's generally best if you use a name that begins with the reverse domain name of your organization or publisher entity. For this project, you can use something like "com.example.myclock." However, you cannot publish your app on Google Play using the "com.example" namespace.

   Minimum SDK is the lowest version of Android that your app supports, indicated using the API level. To support as many devices as possible, you should set this to the lowest version available that allows your app to provide its core feature set. If any feature of your app is possible only on newer versions of Android and it's not critical to the app's core feature set, you can enable the feature only when running on the versions that support it (as discussed in Supporting Different Platform Versions). Leave this set to the default value for this project.

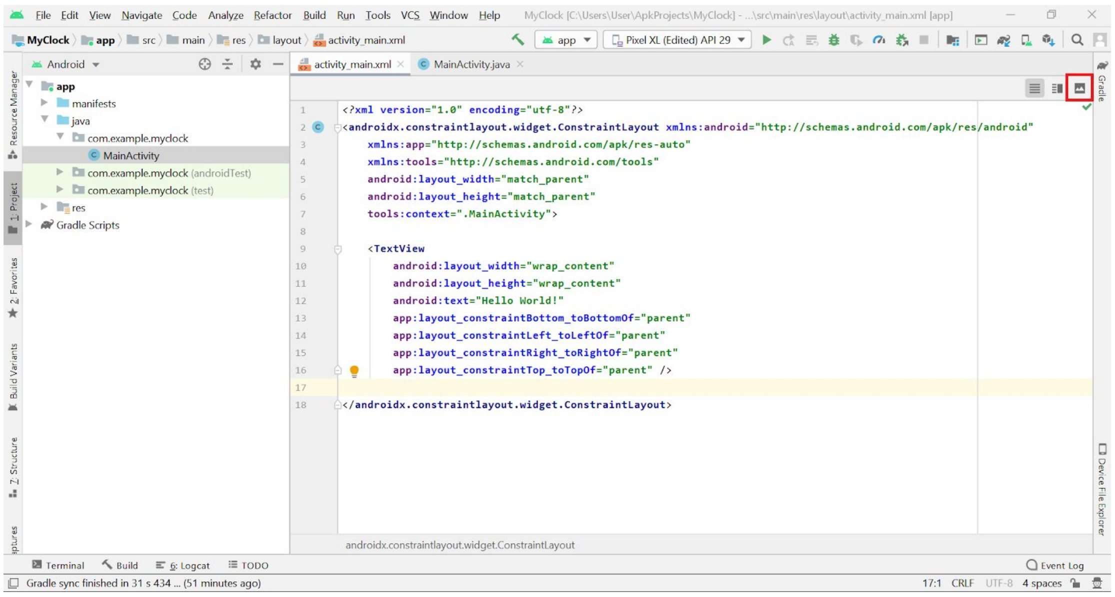
2. Your Android project is now ready. It should be visible in the Project Window on the left of the Android Studio User Interface:


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
MyClock [C:\Users\User\ApkProjects\MyClock] - ...\java\com\example\myclock\MainActivity.java
MyClock 〉 app 〉 src 〉 main 〉 java 〉 com 〉 example 〉 myclock 〉 MainActivity
app
Pixel XL (Edited) API 29
Android
app
manifests
java
com.example.myclock
MainActivity
com.example.myclock (androidTest)
com.example.myclock (test)
res
Gradle Scripts
activity_main.xml
MainActivity.java
Resource Manager
1: Project
2: Favorites
Build Variants
7: Structure
Gradle
Device File Explorer
Terminal
Build
6: Logcat
TODO
Event Log
15:1 CRLF UTF-8 4 spaces

```
package com.example.myclock;

import ...

public class MainActivity extends AppCompatActivity {

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

Gradle sync finished in 31 s 434 ... (10 minutes ago)

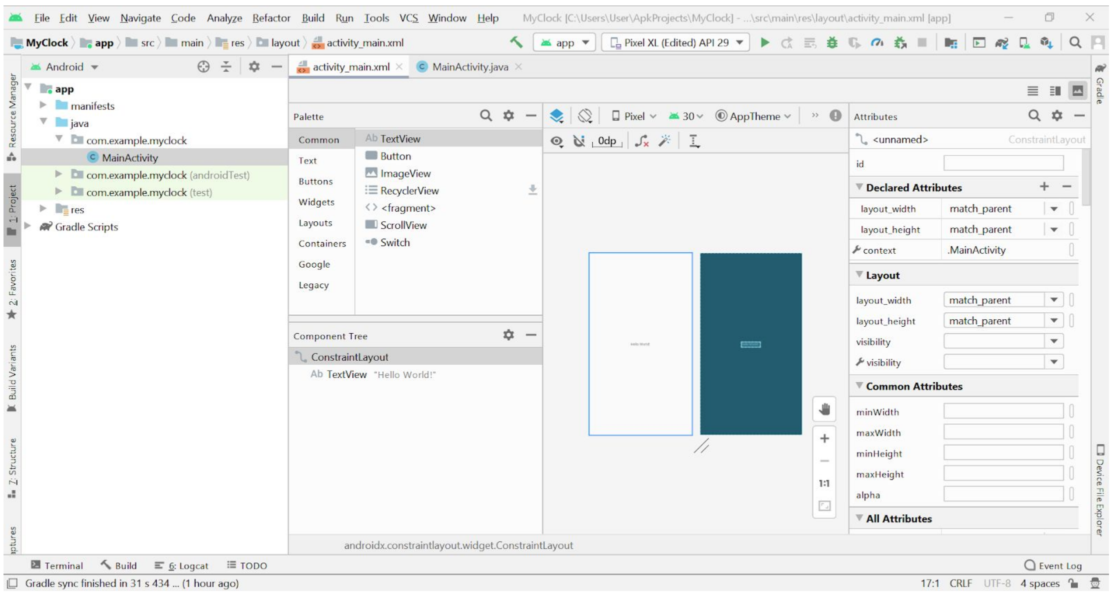
3. This is the default layout of the app. Click the **Design icon** at the top right corner, you will see the GUI of this layout.


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
MyClock [C:\Users\User\ApkProjects\MyClock] - ...\src\main\res\layout\activity_main.xml [app]
MyClock 〉 app 〉 src 〉 main 〉 res 〉 layout 〉 activity_main.xml
app
Pixel XL (Edited) API 29
Android
app
manifests
java
com.example.myclock
MainActivity
com.example.myclock (androidTest)
com.example.myclock (test)
res
Gradle Scripts
activity_main.xml
MainActivity.java
Resource Manager
1: Project
2: Favorites
Build Variants
7: Structure
Gradle
Device File Explorer
Terminal
Build
6: Logcat
TODO
Event Log
androidx.constraintlayout.widget.ConstraintLayout
17:1 CRLF UTF-8 4 spaces

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

Gradle sync finished in 31 s 434 ... (51 minutes ago)

You should be able to see this GUI:



<unnamed> ConstraintLayout

id

Declared Attributes

layout_width	match_parent
layout_height	match_parent
context	.MainActivity

Layout

layout_width	match_parent
layout_height	match_parent
visibility	
visibility	

Common Attributes

minWidth	
maxWidth	
minHeight	
maxHeight	
alpha	

All Attributes

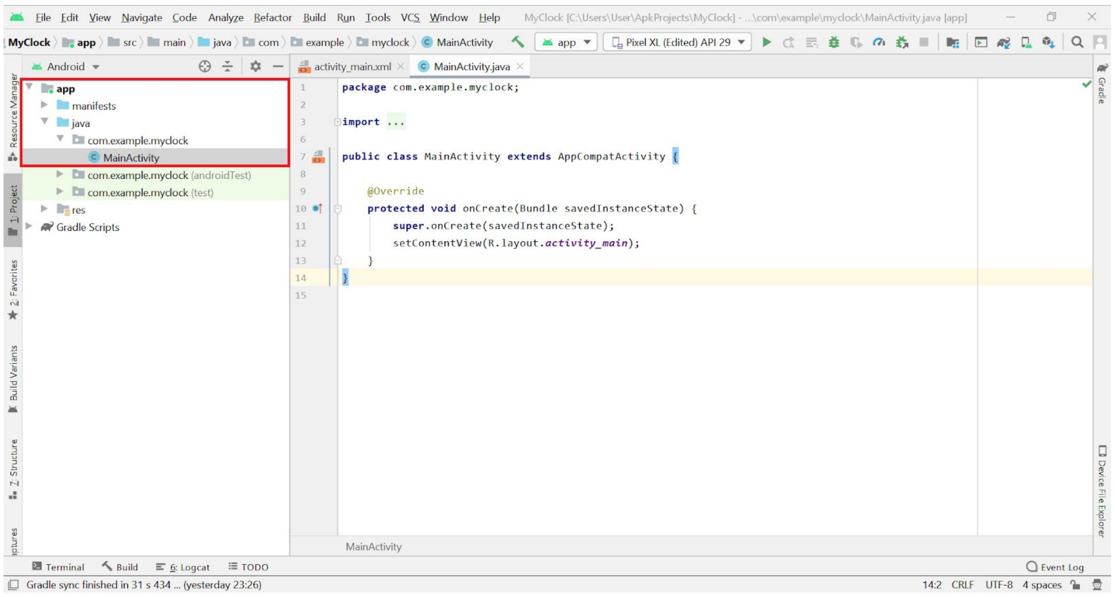
Gradle sync finished in 31 s 434 ... (1 hour ago)

17:1 CRLF UTF-8 4 spaces

From this “What You See What You Get” (WYSWYG) interface, you can realize that the **activity_main.xml** file only defined to display the “Hello World” text on the center.

4. This XML layout file is loaded by a java class located at left-side project window under: **app → java → com.example.myclock → MainActivity**.

   You should be able to see the following:


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
MyClock [C:\Users\User\ApkProjects\MyClock] - ...\com\example\myclock\MainActivity.java [app]
MyClock 〉 app 〉 src 〉 main 〉 java 〉 com 〉 example 〉 myclock 〉 MainActivity
app
Pixel XL (Edited) API 29
Android
app
manifests
java
com.example.myclock
MainActivity
com.example.myclock (androidTest)
com.example.myclock (test)
res
Gradle Scripts
activity_main.xml
MainActivity.java
MainActivity
Terminal
Build
6: Logcat
TODO
Event Log

```java
package com.example.myclock;

import ...

public class MainActivity extends AppCompatActivity {

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

Gradle sync finished in 31 s 434 ... (yesterday 23:26)

14:2 CRLF UTF-8 4 spaces

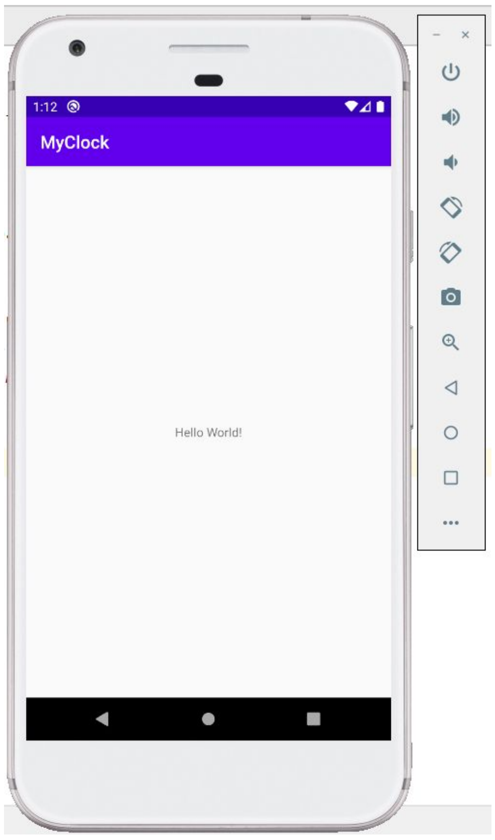
This is the default program.

```
package com.example.myclock;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Notice that the class is based on the AppCompatActivity class, which is a super class of Activity class. An Activity is a single application entity that is used to perform actions. An application may have many separate activities, but the user interacts with them one at a time. The onCreate() method will be called by the Android system when your Activity starts — it is where you should perform all initialization and UI setup.

5. Basically, the “Hello World” is really to run on your AVD. The Android Studio makes it easy to run your applications, just click **Run ‘App’ icon** in the toolbar.

   The Android Studio automatically creates a new run configuration for your project and then launches the Android Emulator. Depending on your environment, the Android emulator might take several minutes to boot fully, so please be patient. When the emulator is booted, the Android Studio installs your application and launches the default Activity. You should be able to see the following:



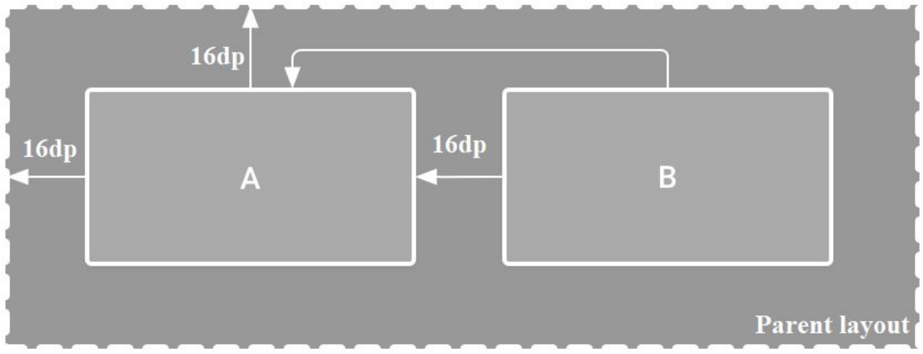
6. Go back to the **activity_main.xml** file and click **Code icon** (next to Design icon), you can find the content of this XML file as shown below:

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

   <TextView
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Hello World!"
       app:layout_constraintBottom_toBottomOf="parent"
       app:layout_constraintLeft_toLeftOf="parent"
       app:layout_constraintRight_toRightOf="parent"
       app:layout_constraintTop_toTopOf="parent" />

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

In this layout file, it used a `ConstraintLayout` and `TextView` objects to define the structure of the UI. Basically, the general structure of an Android XML layout file is simple: it's a tree of XML elements, wherein each node is the name of a View class (this example, however, is just one View element). You can use the name of any class that extends View as an element in your XML layouts, including custom View classes you define in your own code. This structure makes it easy to quickly build up UIs, using a simpler structure and syntax than you would use in a programmatic layout. This model is inspired by the web development model, wherein you can separate the presentation of your application (its UI) from the application logic used to fetch and fill in data.

The Component Tree window on the bottom-left side shows the layout hierarchy of views. In this case, the root view is a `ConstraintLayout`, containing just one `TextView` object. `ConstraintLayout` is a layout that defines the position for each view based on constraints to sibling views and the parent layout. In this way, you can create both simple and complex layouts with a flat view hierarchy. That is, it avoids the need for nested layouts, which can increase the time required to draw the UI.

For example, you can declare the following layout:
• View A appears 16dp from the top of the parent layout.
• View A appears 16dp from the left of the parent layout.
• View B appears 16dp to the right of view A.
• View B is aligned to the top of view A.



7. In the above XML example, there's just one View element: the TextView, which has five XML attributes. Here's a summary of what they mean:

   xmlns:android is an XML namespace declaration that tells the Android tools that you are going to refer to common attributes defined in the Android namespace. The outermost tag in every Android layout file must have this attribute.

   android:id attribute assigns a unique identifier to the TextView element. You can use the assigned ID to reference this View from your source code or from other XML resource declarations.

   android:layout_width attribute defines how much of the available width on the screen this View should consume. In this case, it's the only View so you want it to take up the entire screen, which is what a value of "match_parent" means.

   android:layout_height is just like android:layout_width, except that it refers to available screen height.

   android:text sets the text that the TextView should display. In this example, you use a string resource instead of a hard-coded string value. The hello string is defined in the res/values/strings.xml file. This is the recommended practice for inserting strings to your application, because it makes the localization of your application to other languages graceful, without need to hard-code changes to the layout file.

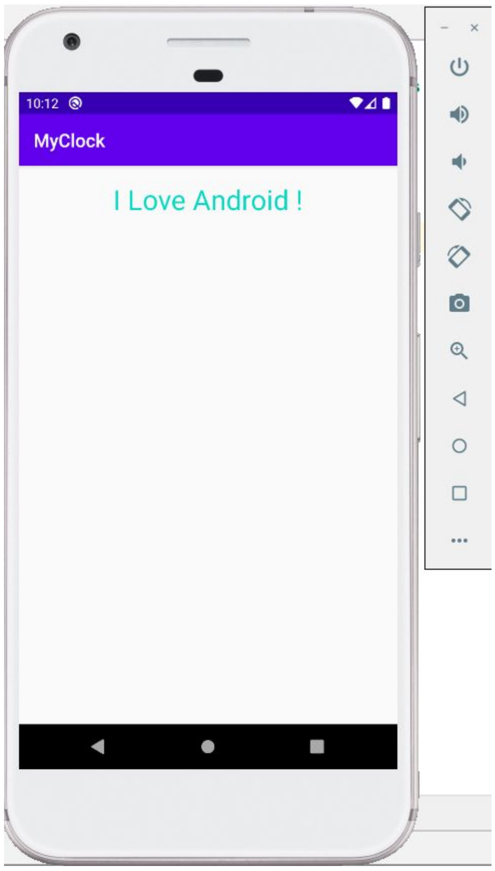
These XML layout files belong in the res/layout/ directory of your project. The "res" is short for "resources" and the directory contains all the non-code assets that your application requires. In addition to layout files, resources also include assets such as images, sounds, and localized strings.

In the Android Project Window, expand the /res/layout/ folder and open activity_main.xml (once opened, you might need to click the "activity_main.xml" tab at the bottom of the window to see the XML source). Replace the contents with the XML layout just defined above.

8. Try to modified layout consists of only one TextView object with `textSize` of 30sp, `textColor` of colorAccent, and text content pointing to “I Love Android !”. In addition, the vertical alignment to the top is 16dp by using the following code:

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

   <TextView
       android:id="@+id/textView"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_marginTop="16dp"
       android:text="I Love Android !"
       android:textColor="@color/colorAccent"
       android:textSize="30sp"
       app:layout_constraintLeft_toLeftOf="parent"
       app:layout_constraintRight_toRightOf="parent"
       app:layout_constraintTop_toTopOf="parent" />

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

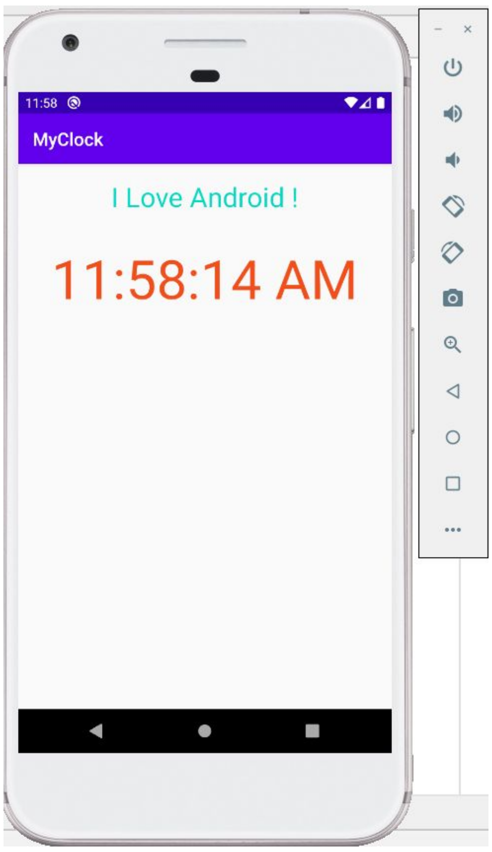
9. Click "**Run**". You should be able to see the following:



10. Try to add a digital clock. Put the following code to the **activity_main.xml** file.

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

   <TextView
       android:id="@+id/textView"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_marginTop="16dp"
       android:text="I Love Android !"
       android:textColor="@color/colorAccent"
       android:textSize="30sp"
       app:layout_constraintLeft_toLeftOf="parent"
       app:layout_constraintRight_toRightOf="parent"
       app:layout_constraintTop_toTopOf="parent" />

   <TextClock
       android:id="@+id/textclock1"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_marginTop="30dp"
       android:textAlignment="center"
       android:format12Hour="hh:mm:ss a"
       android:textColor="#F1511B"
       android:textSize="60sp"
       app:layout_constraintLeft_toLeftOf="parent"
       app:layout_constraintRight_toRightOf="parent"
       app:layout_constraintTop_toBottomOf="@+id/textView" />

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

11. Click “**Run**”. You should be able to see the following:



12. The application you have just created is very similar to other java applications, you may have created in Android Studio. Look in the Project Window on the left side. Notice that the Android Studio has generated a number of folders and files for you:

    AndroidManifest.xml: Every project has a file with this exact name in the root directory. It contains all the information about the application that Android will need to run it:

    - Package name used to identify the application
    - List of Activities, Services, Broadcast Receivers, and Content Provider classes and all of their necessary information, including permissions.
    - System Permissions the application must define in order to make use of various system resources, like GPS.
    - Application defined permissions that other applications must have in order to interact with this application.
    - Application profiling information.
    - Libraries and API levels that the application will use.

java: If you expand this out you will see the package hierarchy you previously entered. This is where your java source code files store.

- **MainActivity.java**: This is the auto-generated stub Activity Class with the name you entered into the project creation wizard. We will add some code to this later.

res: This folder will contain all of the resources (a.k.a. external data files) that your application may need. There are three main types of resources that you will be using and the ADT has created a subdirectory for each.

- **drawable**: This folder will hold image and animations files that you can use in you application. (It already contains a file called icon.png which represents the icon that Android will use for your application once it is installed.)
- **layout**: This folder will hold xml layout files that the application can use to construct user interfaces. You will learn more about this later, but using a layout resource file is the preferred way to layout out your UI. (It already contains a file called main.xml which defines the user interface for your 'HelloWorld.java' Activity class. Double clicking on this file will open up the Android UI Editor that you can use to help generate the xml layout files.)
- **values**: This folder will hold files that contain value type resources, such as string and integer constants. (It already contains a file called strings.xml. Double clicking on this file will open up the Android Resource Editor. Notice that there are two strings in there already, one of which is named 'app_name'. If you select this value, on the right hand side of the editor you should see the Application Name you entered in the project creation wizard. You can use this editor to add new resources to your application.)

gen: This folder will contain Java files that get auto-generated by ADT. Notice that it already contains one file called “R.java”.

R.java: Is a special static class that is used for referencing the data contained in your resource files. If you open this file you will see a number of static inner classes for each of the resource types, as well as static constant integers within them. Notice that the names of the member variables are the same as the names of the values in your resource files. Each value in a resource file is associated with an integer ID, and that ID is stored in a member variable of the same name, within a static class named after its data type.

assets: This folder is for asset files, which are quite similar to resources. The main difference being that anything stored in the 'assets' folder has to be accessed in the classic 'file' manipulation style. For instance, you would have to use the AssetManager class to open the file, read in a stream of bytes, and process the data. You will not be using assets quite as extensively as you will be using resources.

default.properties: This file contains all of the project settings, such as the build target you chose in the project creation wizard. If you open the file up, you should see 'target=4', which is your build target. You should never edit this file manually. If you wish to edit the project properties, do so by right-clicking the project in the 'Package Explorer'' panel, and selecting 'Properties'.

13. Here is a lists of the code editing practices you should consider using when creating Android Studio apps. For complete user documentation for the IntelliJ IDEA interface (upon which Android Studio is based), refer to the **IntelliJ IDEA documentation**. The following tables list keyboard shortcuts for common operations. The red highlighted ones are most commonly used.

Programming key commands

Action	Android Studio Key Command
Command look-up (autocomplete command name / Find Action)	Control + Shift + A (Win / Linux) Command + Shift + A (MacOS)
Project quick fix (show intention actions and quick fixes)	Alt + Enter (Win / Linux) Option + Enter (MacOS)
Reformat code	Control + Alt + L (Win / Linux) Command + Option + L (MacOS)
Show docs for selected API (Quick documentation lookup)	Control + Q (Win / Linux) Control + J (MacOS)
Show parameters for selected method	Control + P (Win / Linux) Command + P (MacOS)
Generate code (getters, setters, constructors, hashCode/equals, toString, new file, new class)	Alt + Insert (Win / Linux) Command + N (MacOS)
Jump to source	F4 or Control + Enter (Win / Linux) F4 or Command + Down Arrow (MacOS)
Delete line at caret	Control + Y (Win / Linux) Command + Delete (MacOS)
Search by symbol name	Control + Alt +Shift + N (Win / Linux) Command + Option + O (MacOS)

Project and editor key commands

Action	Android Studio Key Command
Build	Control + F9 (Win / Linux) Command+F9 (MacOS)
Build and run	Shift + F10 (Win / Linux) Control + R (MacOS)
Toggle project tool window visibility	Alt + 1 (Win / Linux) Command + 1 (MacOS)
Navigate open tabs	Alt + Right Arrow or Left Arrow (Win / Linux) Control + Right Arrow or Control+Left Arrow (MacOS)

These tables list Android Studio keyboard shortcuts for the default keymap. To change the default keymap on Windows and Linux, go to **File → Settings → Keymap**. To change the default keymap on macOS, go to **Android Studio → Preferences → Keymap**.

If you are using macOS, update your keymap to use the macOS 10.5+ version keymaps under **Android Studio → Preferences → Keymap**.

For a complete keymap reference guide, see the **IntelliJ IDEA documentation**.