Department of Electronic Engineering
City University of Hong Kong

EE5415 Mobile Applications Design and Development

Lab01: Android Development Environment (Java)

I. Introduction:

The aim of this lab is to learn the basic skills of using Android Studio as an Android Application Development Environment, which include software installation and project creation on MS Windows-10 machines. Students will setup their Android Studio, create Android Virtual Device (AVD) and go through all steps to create a "My Clock" app and run it on the AVD.

II. Objectives

- Setup the Android Studio 3.5.3
- Create "My Calendar" Android App
- Understand the structure of an Android Project
- Use the Android Emulator

III. Setting up the Development Environment

Starting from 10th December 2014, the official Integrated Development Environment (IDE) for Android app development is Android Studio. Thus, it will be the most popular way to develop Android Apps. Android Studio is built on IntelliJ IDEA Community Edition from Jet Brain, which is one of the popular Java IDEs. At the core of Android Studio is an intelligent code editor capable of advanced code completion, refactoring, and code analysis. The powerful code editor helps us be a more productive Android app developer. In addition, Android Studio comes with an optimized emulator image. Thus, we can test our apps by running them on an Android Virtual Device (AVD).

To setup your Android Studio, you can follow steps as listed below. Detailed installation instructions are available here:
https://developer.android.com/studio/

YouTube Tutorial: How to Download and Install Android Studio 3.4.2 on Windows 10:
https://www.youtube.com/watch?v=RVP5kJiwfDo

3.1 Installation of Android Studio

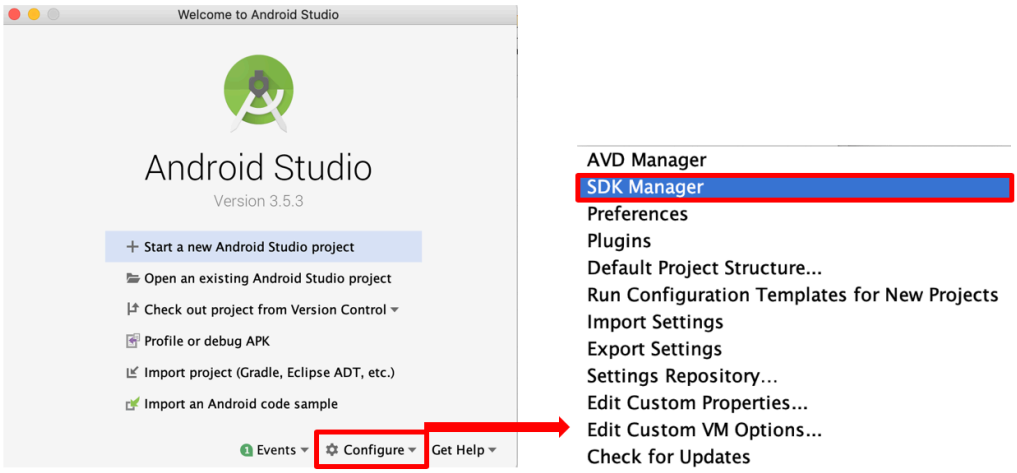
- Go to http://developer.android.com/sdk/index.html and download Android Studio installation package for MS-Windows, Mac or Linux platforms.
- Unpack it to a convenient location of the hard drive.
- Double click the package file to install the Android Studio into your computer.

3.2 Adding SDK Packages

By default, the Android SDK does not include everything you need to start your app development. The SDK separates tools, platforms, and other components into packages, you can download them as needed using the Android SDK Manager. Before starting your project development, there are a few packages you should add to your Android SDK.

To start adding packages, launch the Android SDK Manager in one of the following ways:

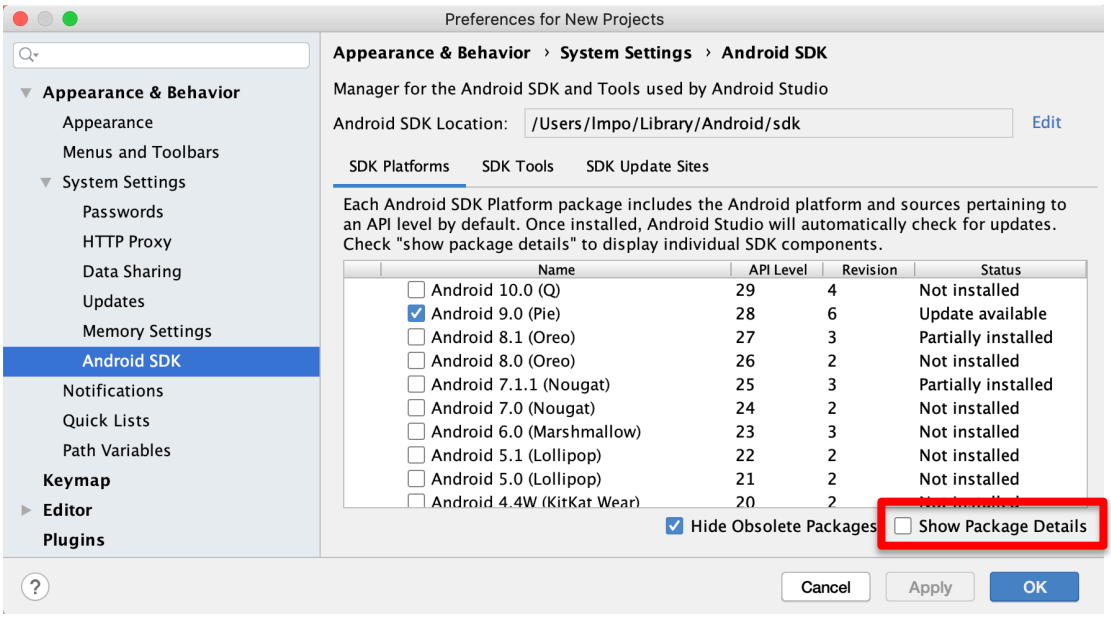
(i) In the Android Studio startup window, click Configure and then SDK Manager:



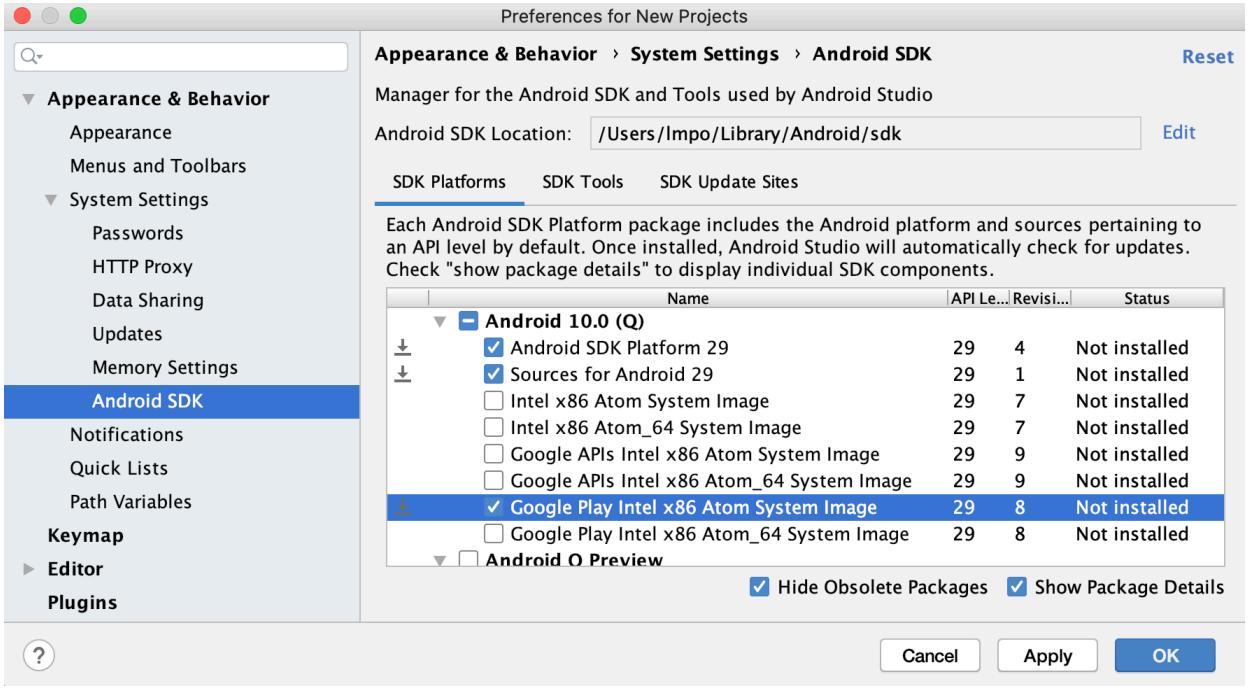
(ii) In the Configure sub-window, click **SDK Manager** in the toolbar.

When you open the SDK Manager for the first time, several packages are selected by default. Leave these selected, but be sure you have everything you need to get started by following these steps:

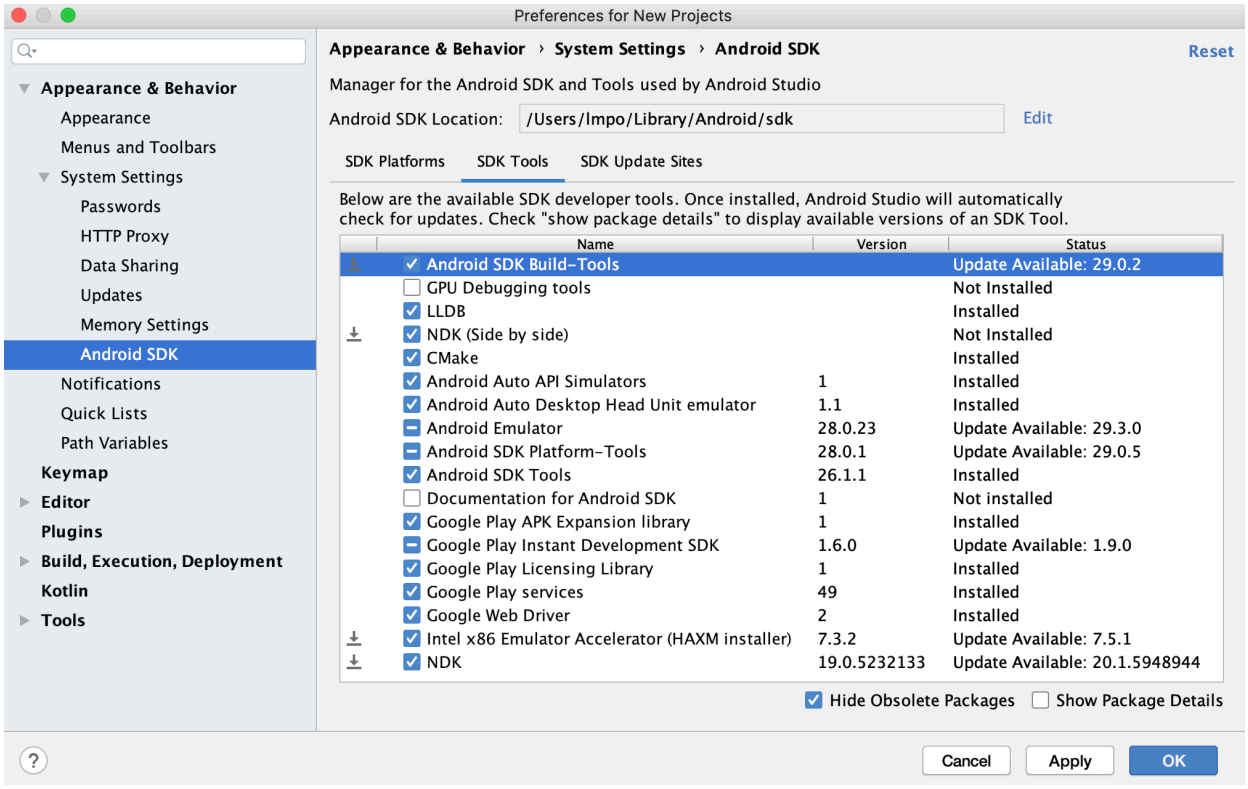
3.2.1. Get the Latest SDK Tools



In the Default Preferences window, select "**Show Package Details**" to show the package details of each platform such that you can select the packages for installation as shown below:



You should select the latest **Android SDK platform 29** and **Google Play Intel x86 Atom System Image**. In addition, you also need to install some essential Android SDK tools by click on the **SDK Tools** button for installing the latest tools as shown below:



You can also select the "Show Package Details" to get more details and here are some essential packages you should selected for setting your development environment.

- **Android SDK Build-tools** (highest version)
- **Android Emulator**
- **Android SDK Platform-Tools**
- **Android SDK Tools**
- **Intel x86 Emulator Accelerator (HAXM installer)**
- **Support Repository**
- **Google USB Driver**
- **Android Support Library**

The Android Support Library provides an extended set of APIs that are compatible with most versions of Android. While Google USB Driver could let use run your apps on physical device by connecting the device with the USB cable. In addition, the HAXM could speed up the performance of the AVD.

Once you've selected all the desired packages, click **OK** button to start the packages installation.

3.3. Create "My Calendar" Android Project to show "Hello World"

As a developer, you know that the first impression of a development framework is how easy it is to write "Hello World." On Android, it's pretty easy if you are using Android Studio as your IDE, because it has a great plugin that handles your project creation and management to greatly speed-up your development cycles.

3.3.1. Create a New Android Project

In Android Studio, to start a new Android project as follows:

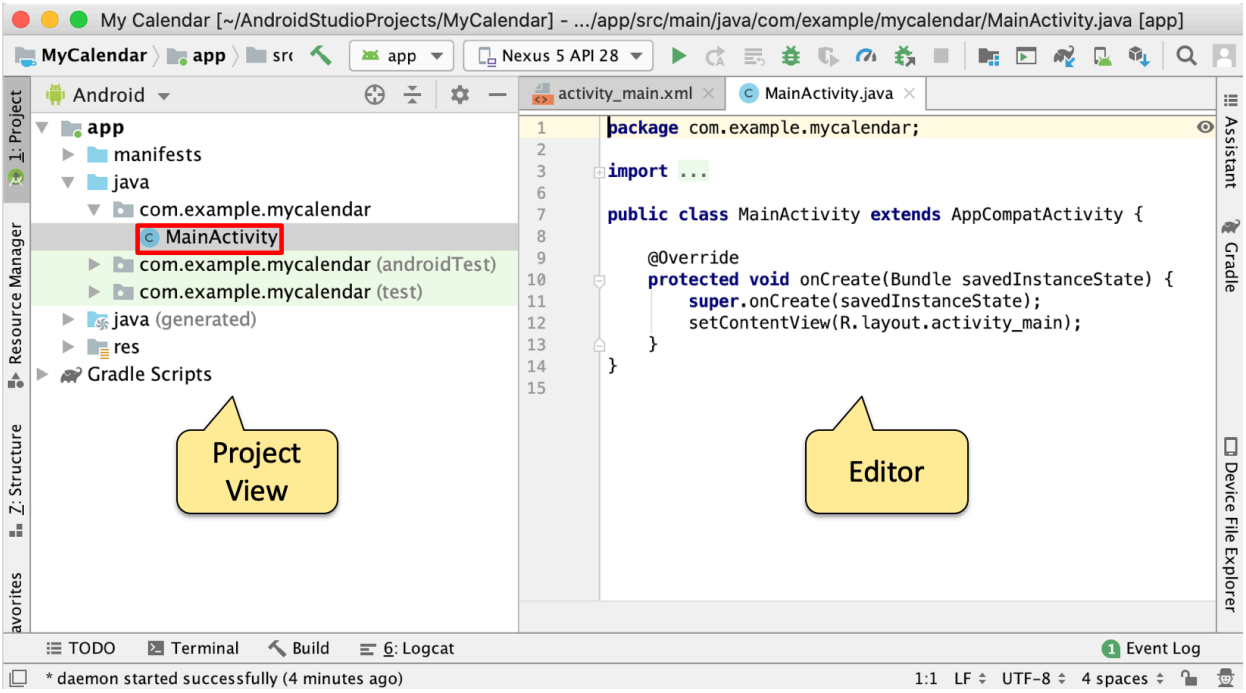
- From Android Studio Startup Menu, select **Start a new Android Studio Project**, then a "New Android Application" window will pop up
- Choose your project : "**Empty Activity**"
- Fill in the project details with the following values:
  - Application name : **My Calendar**
  - Package name: **com.example.mycalendar**
  - Project location: use the default setting
- Language: **Java**
- Minimum API Level: **API 19: Android 4.4 (KitKat)**
- Click **Finish**
- A new project will be created with a Main Activity with following files:
  - Activity Name : **MainActivity**
  - Layout Name : **activity_main**

Here is a description of each field:

- **Application Name** is the app name that appears to users. For this project, use "My Clock"
- **Company Domain** 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.mycalendar." 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.
- **Activity Name** – This is the name for the class stub that will be generated by the plugin. This will be a subclass of Android's Activity class. An Activity is simply a class that can run and do work. It can create a UI if it chooses, but it doesn't need to. As the checkbox suggests, this is optional, but an Activity is almost always used as the basis for an application.

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

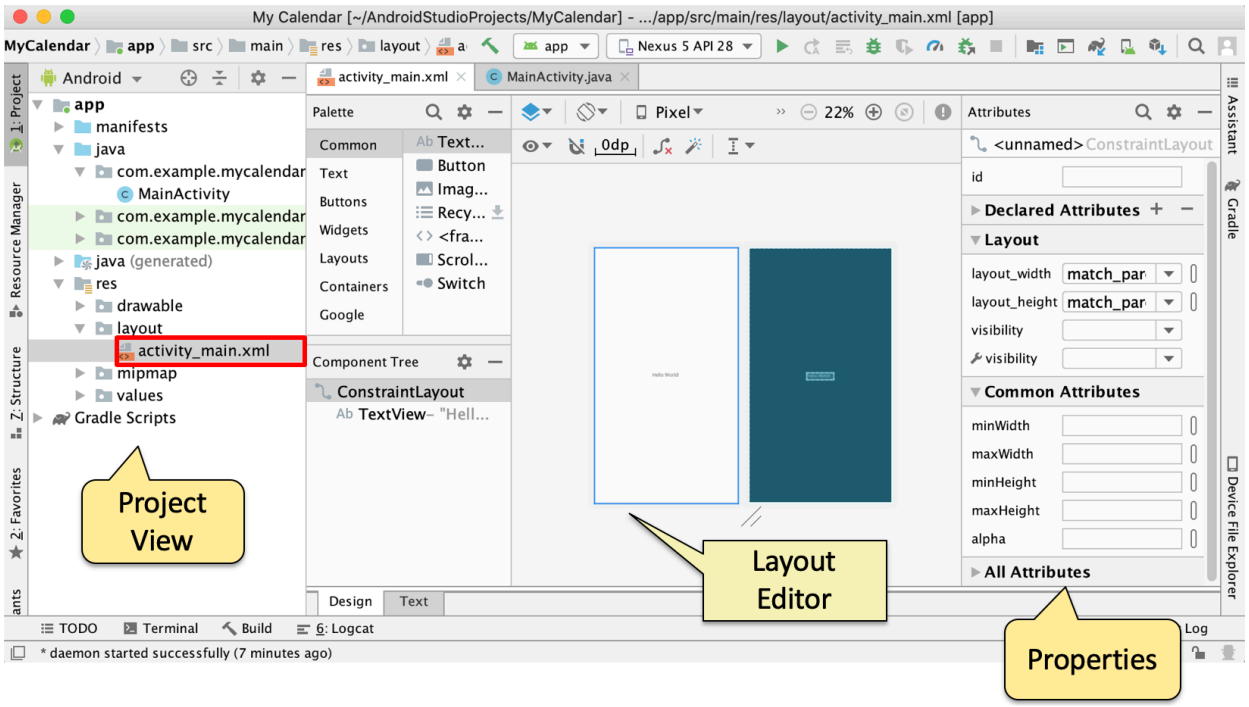


3.3.2. Familiar with the Features of Android Studio IDE

At the core of Android Studio is an intelligent code editor capable of advanced code completion, refactoring, and code analysis. The powerful code editor helps you be a more productive Android app developer.

Android Studio IDE provides multi-screen app develop:

- Build apps for Android phones, tablets, Android Wear, Android TV, Android Auto and Google Glass.
- With the new Android Project View and module support in Android Studio, it's easier to manage app projects and resources, refactoring, and code analysis. The powerful code editor helps us be a more productive Android app developer.



3.4. Android Virtual Device Manager (AVD Manager)

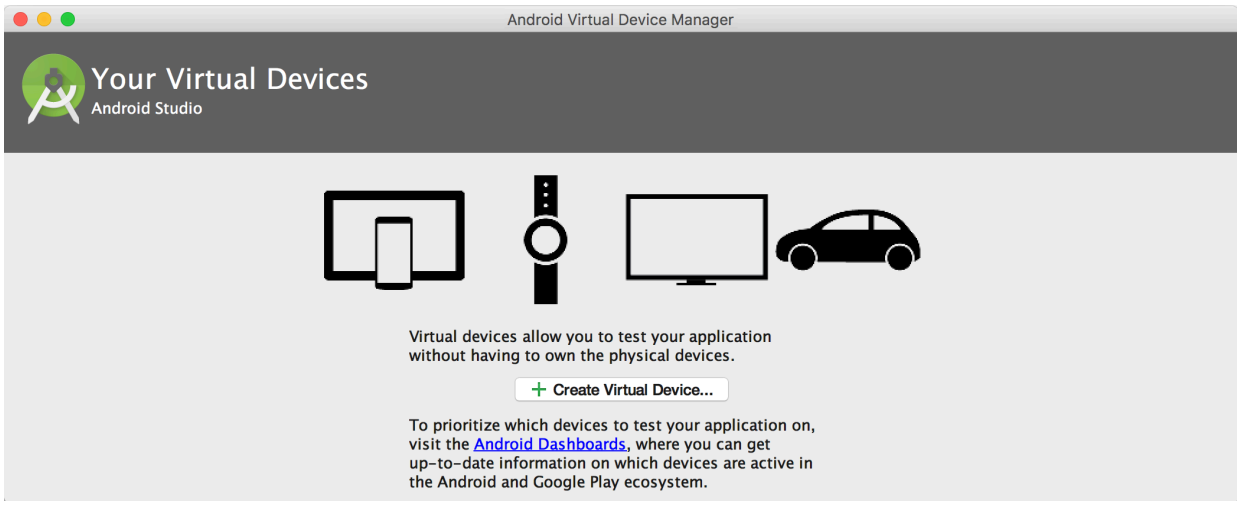
Android Studio IDE provides virtual devices for all shapes and sizes

- Android Studio comes pre-configured with an optimized emulator image.
- The updated and streamlined Virtual Device Manager provides pre-defined device profiles for common Android devices.

The AVD Manager provides a graphical user interface in which you can create and manage Android Virtual Devices (AVDs), which are required by the Android Emulator. For more information, see Managing AVDs with AVD Manager.

You can launch the AVD Manager in Android Studio by selecting **Tools > AVD Manager**, or click the AVD Manager icon in the toolbar.

The AVD Manager main screen shows your current virtual devices if you have created some, otherwise it will show below. Press the "Create Virtual Device …" button to create a device for your lab exercise.

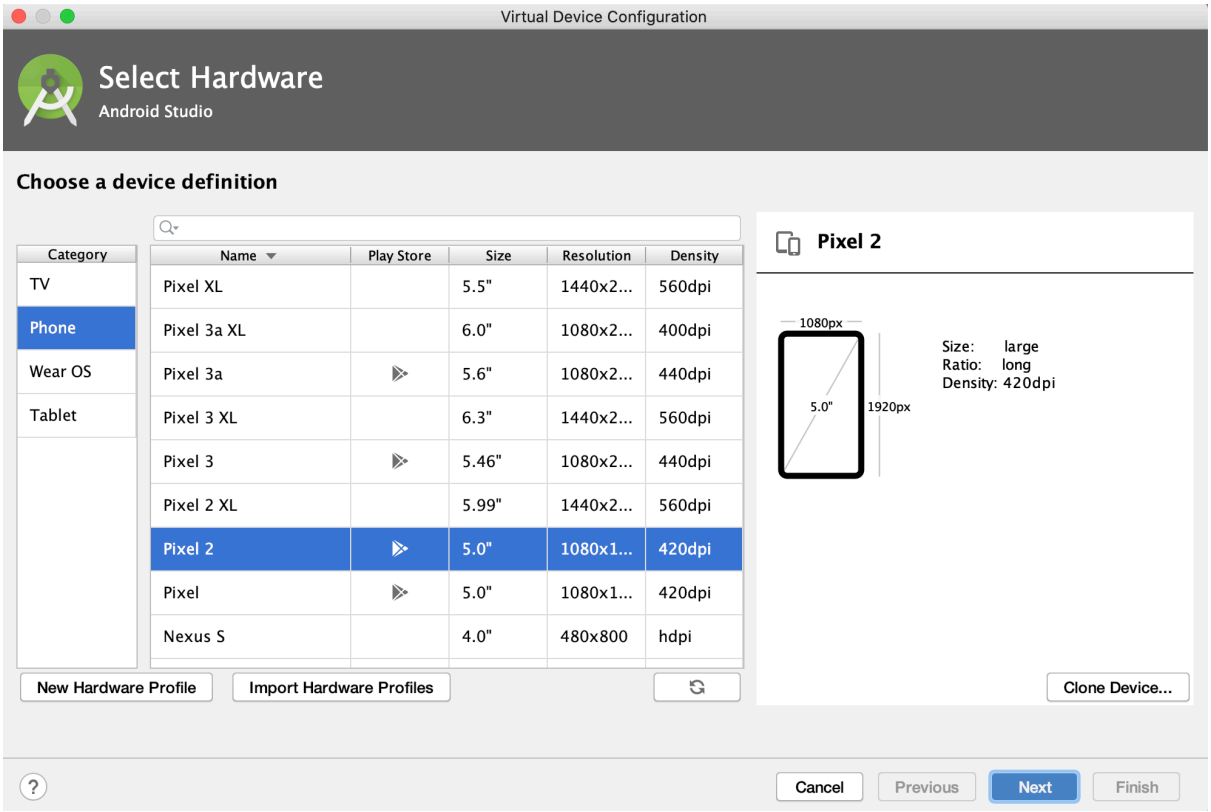


3.4.1. Creating an AVD

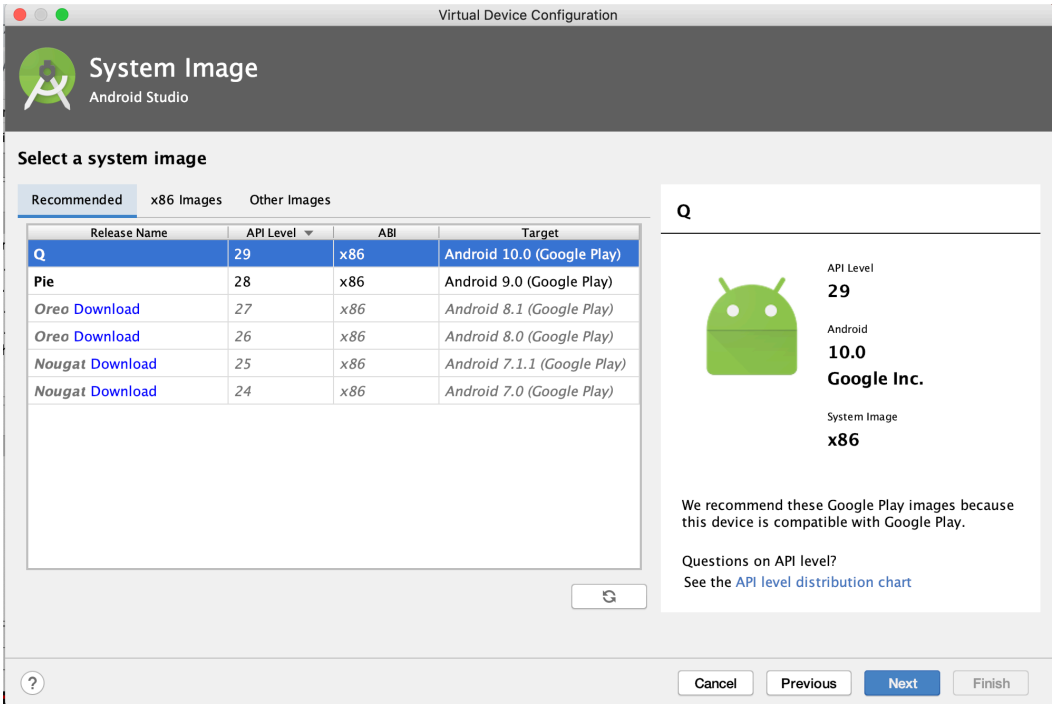
You can create as many AVDs as you would like to use with the Android Emulator. To effectively test your app, you should create an AVD that models each device type for which you have designed your app to support. For instance, you should create an AVD for each API level equal to and higher than the minimum version you've specified in your manifest `<uses-sdk>` tag.

To create an AVD based on an existing device definition:

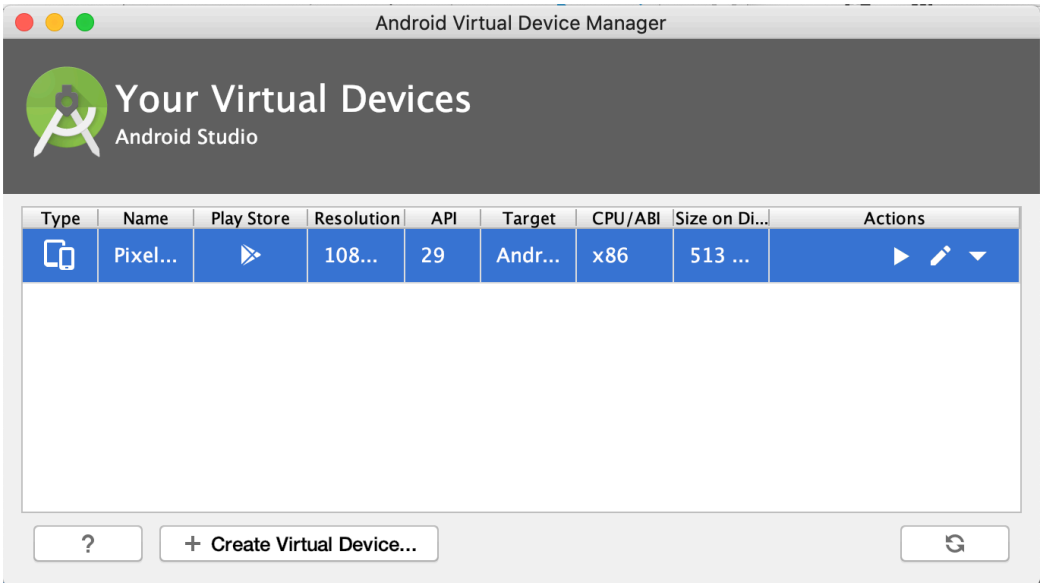
1. From the main screen as shown in above figure, click **Create Virtual Device**.
2. In the Select Hardware window, select a device configuration, such as **Pixel 2**, then click **Next**.



3. Select the desired system version for the AVD and click **Next**.
4. Select System Image such as **Q with API 29 level, ABI x86**.



5. Verify the configuration settings, then click **Finish**.

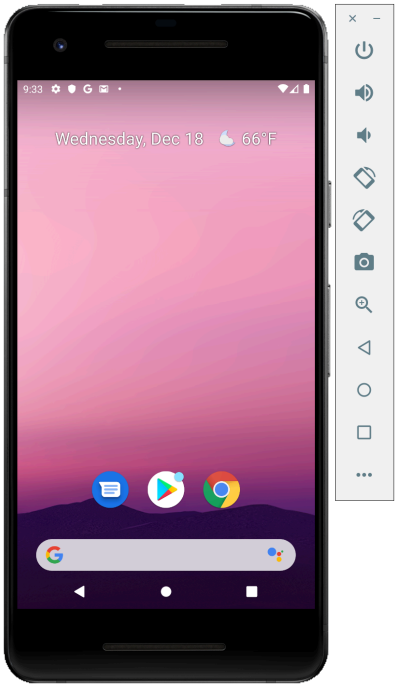


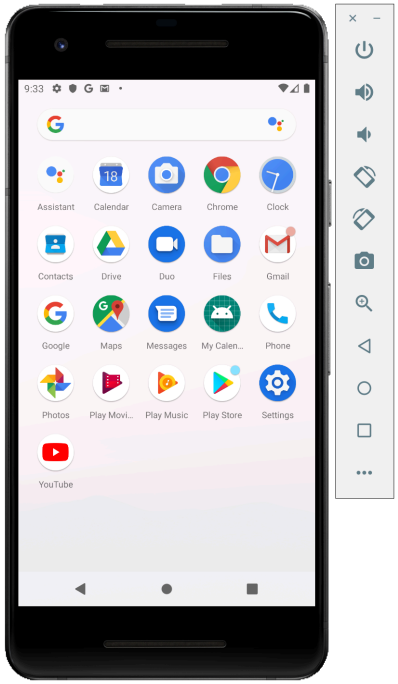
If necessary, click **Show Advanced Settings** to select a custom skin for the hardware profile and adjust other hardware settings.

3.4.2. Launch the AVD

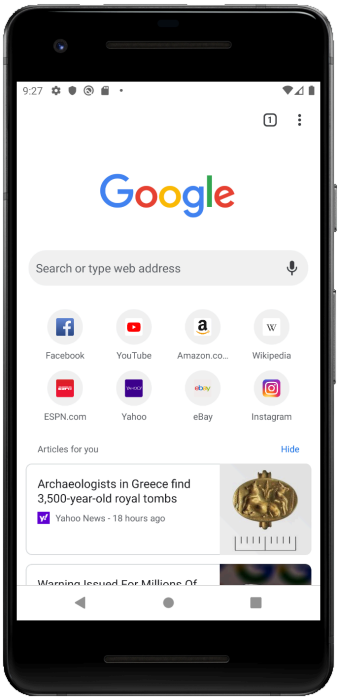
To launch the **Pixel 2** AVD in the Android Emulator, click the launch button ▶ in the list of AVDs.

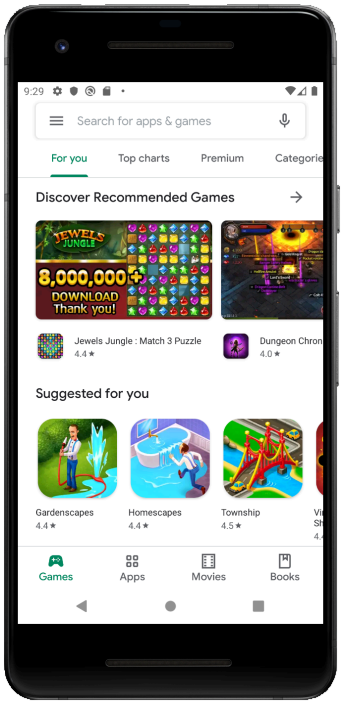
- Click “GOT IT” for going to the Home screen as shown below:





- Swipe up to show all the installed apps and then click **Chrome** app icon to browse the web and click Play Store app icon to login your Google account to download other apps for your Homework 1.





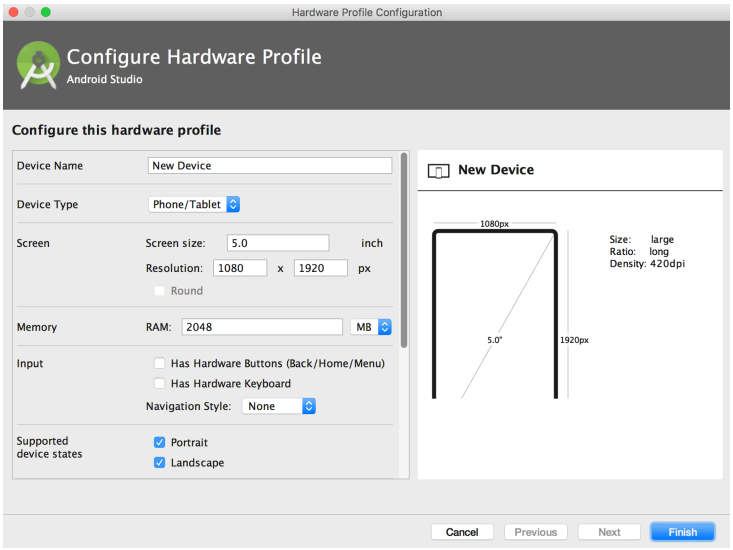
3.4.2. Creating a Device Definition

In case the available device definitions do not match the device type you'd like to emulate, you can create a custom device definition for your AVD:

1. From the main screen of AVD Manager, click **Create Virtual Device**.
2. To begin your custom device by using an existing device profile as a template, select a device profile then click **Clone Device**.

   Or, to start from scratch, click **New Hardware Profile**.

3. The following Configure Hardware Profile window as shown below allows you to specify various configurations such as the screen size, memory options, input type, and sensors. When you're done configuring the device, click **Finish**.



4. Your custom device configuration is now available in the list of device definitions (shown after you click **Create Virtual Device**). To continue preparing an AVD with your custom device configuration, select the new configuration and follow the instructions above to create an AVD with an existing device definition (and select your new definition).

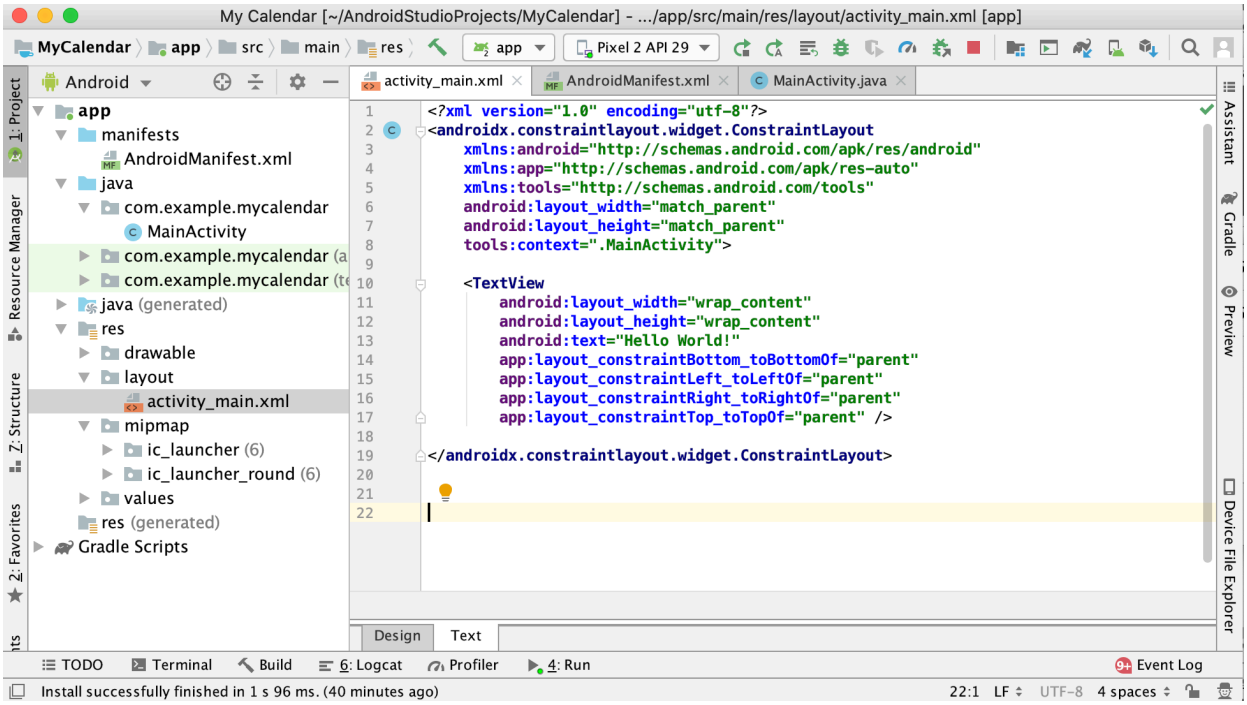
In this lab, you will run your app in the Android Emulator. Before you can launch the emulator, you must create an Android Virtual Device (AVD). An AVD defines the system image and device settings used by the emulator.

Please note that this lab assumes you are building applications for minimum version of Android 4.4 platform (KitKat, API level 19). Android apps are forward compatible. That means an app built against API level 19 can run on Android 4.4 platform and further releases. If you compile the project in your own computer, you should make sure you have the correct SDK version installed.

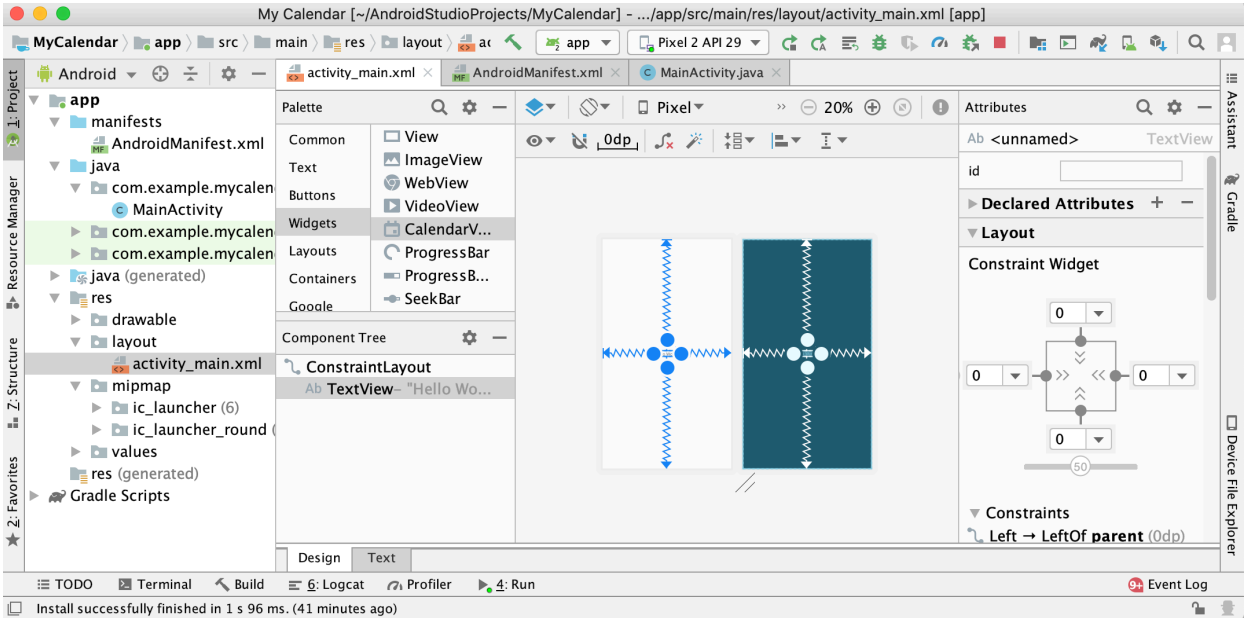
Now your environment is ready for development. When you are working in our laboratory, the computers may have already configured properly and installed the software you need. You may also want to set up the same development environment in your own computer using these procedures

IV. Create “My Calendar” App with a CalendarView Wedget

Go back to your created “My Calendar” Android project. The default layout of your Android Studio should be as shown below:



It is currently opening the activity_main.xml file, located inside the Android View **app > res > layout > activity_main.xml.** This is an XML-based layout file for building the user interface (UI) of the Android app. You can click on the “**Design**” button to the GUI of this layout as shown below:



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 in the centre.

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

It should look like this:

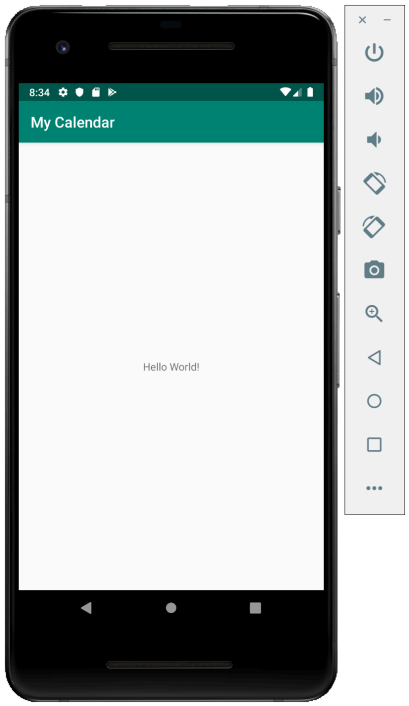
```
package com.example.mycalendar;

import android.support.v7.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.

4.1 Run the "My Calendar" App on AVD

Basically, the "Hello World" is really to run on your AVD. The Android Studio makes it easy to run your applications: Select **Run > Run 'App'**.

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 now see something like this:



The "Hello World" you see in the grey bar is actually the application title. The Android Studio creates this automatically (the string is defined in the `res/values/strings.xml` file and referenced by your `AndroidManifest.xml` file). The text below the title is the actual text that you have created in the TextView object.

4.2. Modify the Layout XML and the String of the TextView

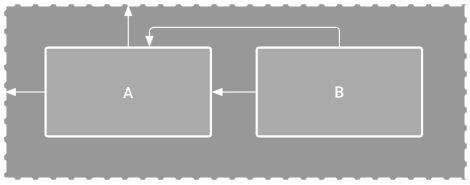
If you open the `activity_main.xml` file again and click on the button of "**Text**", you can find the content of this XML file as shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<android.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" />

</android.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's 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.

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** - This 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** - This 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** - This 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** - This is just like android:layout_width, except that it refers to available screen height.
- **android:text** - This 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.

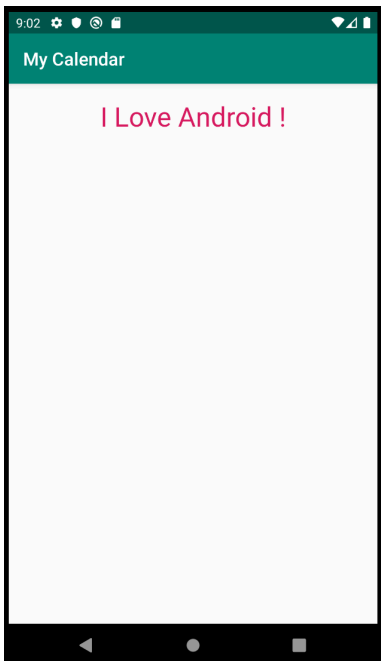
These XML layout files belong in the **res/layout/** directory of your project. The "**res**" is short for "resources" and the directory contains all the non-code assets that your application requires. In addition to layout files, resources also include assets such as images, sounds, and localized strings.

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.

```
<?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:layout_marginTop="32dp"
        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>
```

This 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.

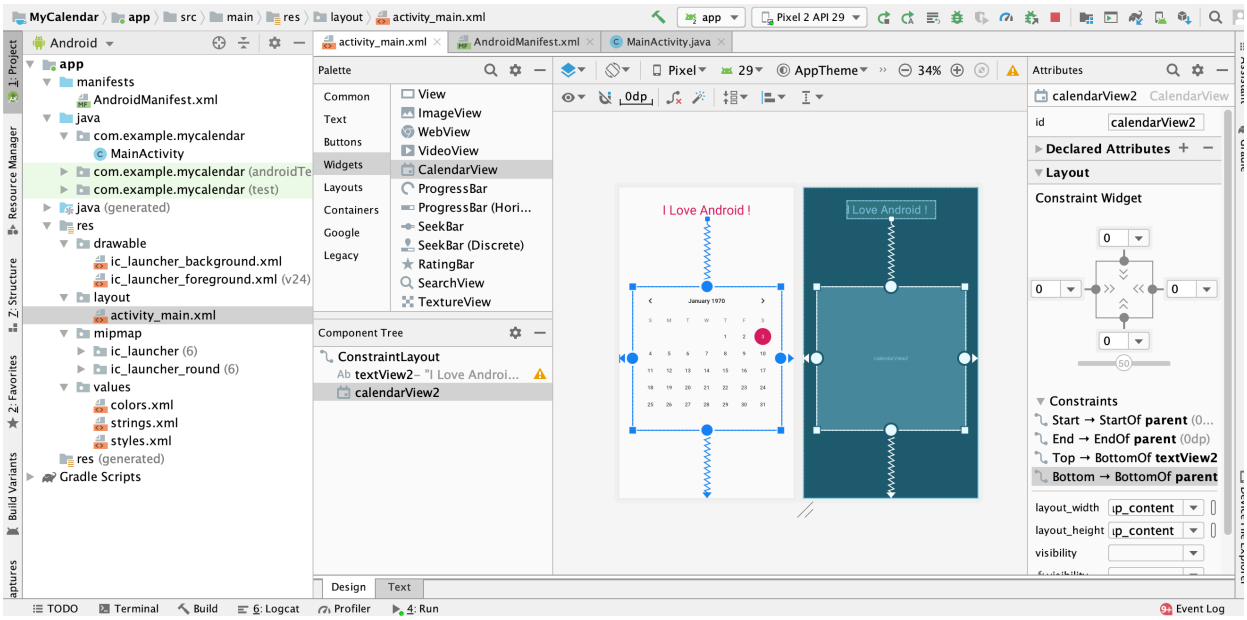
Save the project and **re-run** your application by clicking the green arrow icon to run, or select Run > Run 'App'



In the following sections, you'll build a layout similar to this.

4.3. Add a Calendar to UI

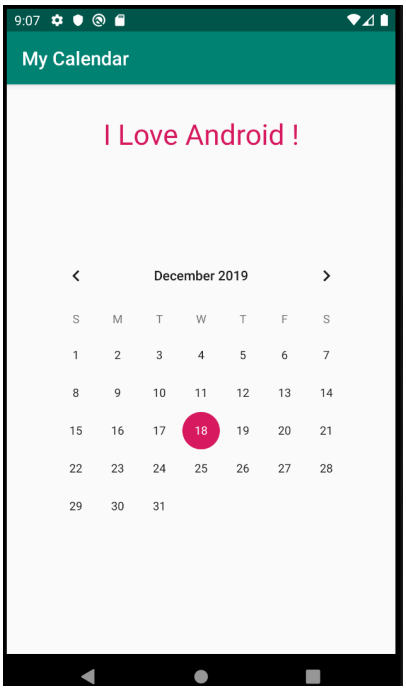
Open the `activity_main.xml` file again and click on the button of "**Design**" to GUI. From the Palette window, drag and drop a **CalendarView** object to the layout interface as shown below:



Try to add layout constraint to the calendar object as shown in the above figure. After that, click the "Text" button to view the XML content of the activity_main.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:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        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" />

    <CalendarView
        android:id="@+id/calendarView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView2" />

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

Save your modified code and run it.



V. Take a Tour of the Application

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 contians 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.propeties**: Ths 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'.

VI. Android Studio Keyboard Commands

This section lists just a few 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.

Programming key commands

Action	Android Studio Key Command
Command look-up (autocomplete command name)	CTRL + SHIFT + A
Project quick fix	ALT + ENTER (Win) OPTION + ENTER (macOS)
Reformat code	CTRL + ALT + L (Win) OPTION + CMD + L (macOS)
Show docs for selected API	CTRL + Q (Win) F1 (macOS)
Show parameters for selected method	CTRL + P
Generate method	ALT + Insert (Win) CMD + N (macOS)
Jump to source	F4 (Win) CMD + down-arrow (macOS)
Delete line	CTRL + Y (Win) CMD + Backspace (macOS)
Search by symbol name	CTRL + ALT + SHIFT + N (Win) OPTION + CMD + O (macOS)

Project and editor key commands

Action	Android Studio Key Command
Build	CTRL + F9 (Win) CMD + F9 (macOS)
Build and run	SHIFT + F10 (Win) CTRL + R (macOS)
Toggle project visibility	ALT + 1 (Win) CMD + 1 (macOS)
Navigate open tabs	ALT + left-arrow; ALT + right-arrow (Win) CTRL + left-arrow; CTRL + right-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.

EE5415 Mobile Applications Development and Design

Lab01 Homework: Android App Evaluation

Select an Android app from the Google Play (http://play.google.com), which may be similar to an app that you want to develop in your course group project. The purpose of this is to get your thinking about the project early, to force you to dig into the Google Play and to look at other apps just to see what can be done in Android. Critically evaluate the selected app. Submit **a 4-page evaluation report** with following items:

1. The name of the app evaluated.
2. A summary paragraph for the selected app, which identifies key features of the app.
3. At one page for identifying the target audience of the app, evaluating the profit and/or humanitarian potential of the app (e.g. the expected market for the app, estimated number of paid downloads or estimated number of impressions/clicks per month if ad-based), and functionality and user interface design evaluations.
4. A list of positive characteristics (e.g. high-quality graphics, fun, indispensable tool)
5. A list of negative characteristics (e.g. force close, slow, confusing menu titles).

Submit your evaluation in MS-Words format as attachment to EE5415 Gmail account at ee5415@gmail.com on or before **11.00pm of 5th Feb 2020**. The subject of your email should satisfy the following format:

EE5415 Lab01 Homework – Student Name (Student Number)
Example: EE5415 Lab01 Homework – Chan Chi Man (51234567)