(We can also do JUnit unit test in Android Studio, see my course notes on Junit)

每次用了 Emulator 後不要關(鎖屏也可以開著, 即不鎖), 下次要用 Emulator 時可以直接用, 這樣可以省掉 launch Emulator 的時間!

Android Studio 快捷鍵:
Undo: Cril+Z, 或點上面的彎箭頭
使用它的提示: 用上下方向鍵選中某個提示的, 按回車
按 Tab 鍵可以自動补全, 跟 Terminal 中一樣.
若要在 Emulator 中 run, 則按上面那個指向右的三角形, 它是 save, compile 和 run 都弄了. 視頻中 Emulater 選的 Nexus 5 API 23 x86(Android 6.0, API 23), 我不一定要選這個.

Notes from Udacity nanodegree video:
Compile:
./gradlew assembleDebug
Then an apk file will be built: app/build/outputs/apk/app-debug-unaligned.apk
Add a custom app icon: Right-click on the app folder, New, Image Asset, Image file.

Android Studio notes from https://youtu.be/nuc3H4KhSFY:

Android Studio → File → New → New Project

Then in the window of New Project,
Application name: Converter
Company Domain: edu.gatech.seclass
Package name: seclass.gatech.edu.converter
→ Next

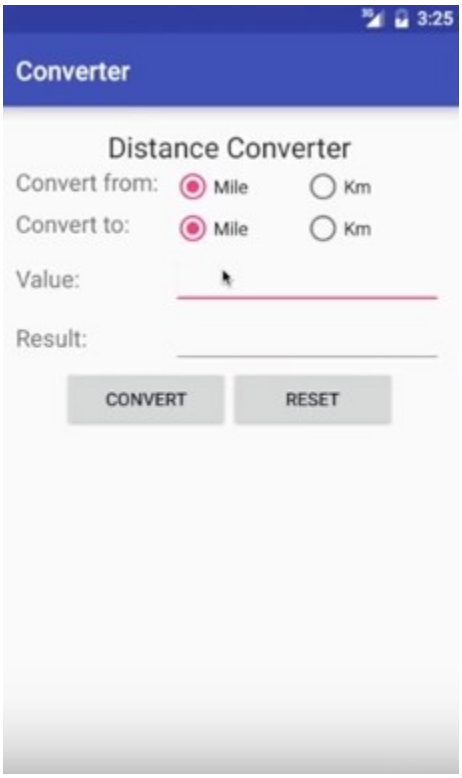
Then in the window of Target Android Devices,
Minimum SDK: API 15: Android 4.0.3(IceCreamSandwich)
→ Next

Then in the window of Add an Activity to Mobile,
Choose Empty Activity
→ Next

Then in the window of Customize the Activiey,
Use all default settings (Activity Name: MainActivity, Check Generate Layout File, Layout Name: activity_main)
→ Finish

Then comes to the code window.
注意在左邊目錄的上方要選 Project, 而不是默認的 Android.
Two things to point out here: one is the MainActivity(in the java directory), where the MainActivity is basically the main window of your application in this case. And the second thing that you wanna pay attention to is the layout(in the res directory). That defines basically le guin for this activity.

要做的是如下的 app:



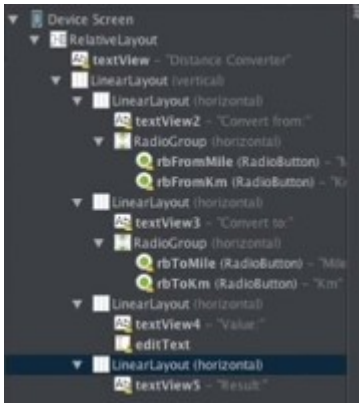
以下就是寫 activity_main.xml 和 MainActivity.java 兩個文件.

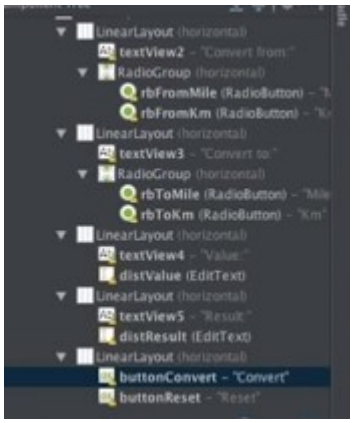
寫 activity_main.xml(其實全是拖, activity_main.xml 的代碼見本 note 最後):

點左邊的 res/layout/activity_main.xml on the left. In the middle-bottom, switch from Text to Design. 然後看到一個手機, 可以把 Widget 等拖到手機裡去. 在手機頭上還可以選用哪款手機.
然後將手機中的 Hello Word 刪掉, 但 Hello Word 所在的那個 RelativeLayout 不刪.
->拖一個 Widgets 中的 Large Text 到手機中(放入之前 Hello Word 所在的 RelativeLayout 中), 作為這個 activity 的 title.
->雙擊這個 Large Text, 填入其文字內容: Distance Converter.
->拖一個 Layouts 中的 LinearLayout(Vertical)到之前 Hello Word 所在的 RelativeLayout 中, so that we can put all the different widgets vertically.
->在那個 Vertical Layout 的框還在顯示時(若要手動讓它顯示, 可在右邊的 Componet Tree 中點這個 Layout, 其它的如 text view 也一樣這樣點出來. 一般新建一個東西時, 都要先在 Componet Tree 中將它的上一級的東點亮), 拖一個 Layouts 中的 LinearLayout(Horizontal)到 Vertical Layout 裡面, one for each basically set of lable and controls.
->在那個 Horizontal Layout 的框還在顯示時, 拖一個 Widgets 中的 Medium Text 到裡面作為 label, 文字內容改為 Convert from: .

->拖一個 Containers 中的 RadioGroup 到 Convert from 所在的 Horizontal Layout 中, that is ganna allow me to put the two check button together.
->拖兩個 Widgets 中的 RadioButton 到 RadioGroup 中. 可以在右上角的 Componet Tree 中看到這些東西的 hierarchy, 若某個東西的 hierarchy 不對, 可在這裡將它拖到正確的地方.
->將第一個 RadioButton 的 text 改為 Mile, 並選中 checked(意思是這個 button 是 initially checked), 並將 id 改為 rbFromMile
->將第二個 RadioButton 的 text 改為 Km, 並將 id 改為 rbFromKm
->將兩個 RadioButton 水平排列: 在右上角的 Componet Tree 中選中 RadioGroup, 在右下角的 Properties 中點 orientation, 然後選 horizontal.
->讓兩個 RadionButton 間隔遠點: 在右上角的 Componet Tree 中選中 rbFromMile, 在右下角的 Properties 中, 將 layout:width 的值改為 100 (改了後點下它下面的 layout:height 就定好了).
->再拖一個 LinearLayout(Horizontal)到之前的下面, 然後按前面一樣的步驟, 在這個 layout 裡建一個 Medium Text(內容為 Convert to:), 再建一個 RadioGroup, 兩個 RadioButton: Mile(rbToMile, checked)和 Km(rbToKm), 水平排列, 並將 rbToMile 的 width 改為 100.
->將 Convert from 和 Convert to 的 width 都改為 120.
->再拖一個 LinearLayout(Horizontal)到之前的下面, 拖一個 Medium Text 到這個 layout 裡, 文字內容為 Value: , width 設為 120. 拖一個 Text Fields 中的 Number(Decimal)到這個 layout 裡, id 設為 distValue.
->再拖一個 LinearLayout(Horizontal)到之前的下面, ,拖一個 Medium Text 到這個 layout 裡, 文字內容為 Result:, width 設為 120. 拖一個 Text Fields 中的 Number(Decimal)到這個 layout 裡, id 設為 distResult.
->再拖一個 LinearLayout(Horizontal)到之前的下面, 拖兩個 Widget 中的 Button 到這個 layout 裡, 文字內容分別為 Convert 和 Reset, id 分別設為 buttonConvert 和 buttonReset, 並將它們的 weight(不是 width)都設為 1, so that 它們兩個 have the same weight and take up the whole space. 將它們的 properties 中的 onClick 都設為 handleClick , 即 assign a method that is going to be invoked, so called basically, every time that the button, the widget, is clicked. 在 handleClick 函數中, 會判斷是哪個 button invoked 的它.
->在 Componet Tree 中選中這兩個 button 所在的 layout, 然後在手機中用鼠標調整它的寬度. 如果它不是默認居中的, 還要在 Componet Tree 中將它的 layout:gravity(不是後面那個單獨的 gravity)中的 center 改為 horizontal.

最終的 hierarchy 是這樣的:





寫 MainActivity.java:
//A view is basically a widget
// switch(view.getId()): 若是點的 Convert 這個 button, 則 view 等於 Convert 這個 button, view.getID()就等於 Convert 這個 button 的 id, 即 buttonConvert.
// Toast is a window that appears, telling you something and then disappears automatically without any need to click.

```
package com.example.android.myapplication;
import android.content.Context;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.EditText;
import android.widget.RadioButton;
import android.widget.Toast;
import java.text.DecimalFormat;
public class MainActivity extends AppCompatActivity {
    private RadioButton rbFromMile;
    private RadioButton rbFromKm;
    private RadioButton rbToMile;
    private RadioButton rbToKm;
    private EditText distValue;
    private EditText distResult;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        rbFromMile = (RadioButton) findViewById(R.id.rbFromMile);
        rbFromKm = (RadioButton) findViewById(R.id.rbFromKm);
        rbToMile = (RadioButton) findViewById(R.id.rbToMile);
        rbToKm = (RadioButton) findViewById(R.id.rbToKm);
        distValue = (EditText) findViewById(R.id.distValue);
        distResult = (EditText) findViewById(R.id.distResult);
    }
    public void handleClick(View view) {
        String unitFrom = "Mile";
        String unitTo = "Mile";
        switch(view.getId()) {
            case R.id.buttonConvert:
                String value = distValue.getText().toString();
                if(value.length() > 0) {
                    if(rbFromKm.isChecked()) {
                        unitFrom = "Km";
                    }
                    if (rbToKm.isChecked()) {
                        unitTo = "Km";
                    }
                    if (unitFrom.contentEquals(unitTo)) {
                        distResult.setText(value);
                    } else {
                        if (unitFrom.contentEquals("Mile")) {
                            distResult.setText(milesToKm(value));
                        } else {
                            distResult.setText(kmToMiles(value));
                        }
                    }
```

```
            } else {
                Context context =  getApplicationContext();
                CharSequence text =  "Empty value!";
                int duration = Toast.LENGTH_SHORT;
                Toast toast = Toast.makeText(context, text, duration);
                toast.show();
            }
            break;
        case R.id.buttonReset:
            distValue.setText("");
            distResult.setText("");
            rbFromMile.setChecked(true);
            rbToMile.setChecked(true);
            break;
    }
  }
  public String milesToKm(String strMiles) {
    double miles = Double.parseDouble(strMiles);
    double km = miles * 1.60934;
    DecimalFormat format = new DecimalFormat("#.##");
    return String.valueOf(format.format(km));
  }
  public String kmToMiles(String strKm) {
    double km = Double.parseDouble(strKm);
    double miles = km / 1.60934;
    DecimalFormat format = new DecimalFormat("#.##");
    return String.valueOf(format.format(miles));
  }
}
```

From 本 note 最後:
R is basically a static class that gets associated to your Android app, and contains references to all of your resources. So for example in this case we're using R to access the IDs and within these IDs the text one ID which is the one corresponding to our text entry.

From stackoverflow:
getId() returns the android:id value, or the value you set via setId(). In your XML, they(指 the IDs) may look like @+id/field1 and @+id/field2. Those are allocating ID resources, which are turned into numbers at compile time. These are the numbers you refer to in Java code as R.id.field1 and R.id.field2.

我：如果 getID()返回的是 String 的話，那代碼中就不用寫成 R.id.buttonReset 了，而會直接寫成 "buttonReset". 實際上，getID()一個數更 make sense, 因為數作為一個 unique ID 才能常見. Android Studio 中若寫出 R.id, 後面自動是示的 buttonReset 顯示的也是一個 int.

And obviously, we could make this much more fancy by, for example, associating actions also to the Value so if somebody enters some value something happens. But as I said, we want to keep it simple for now.

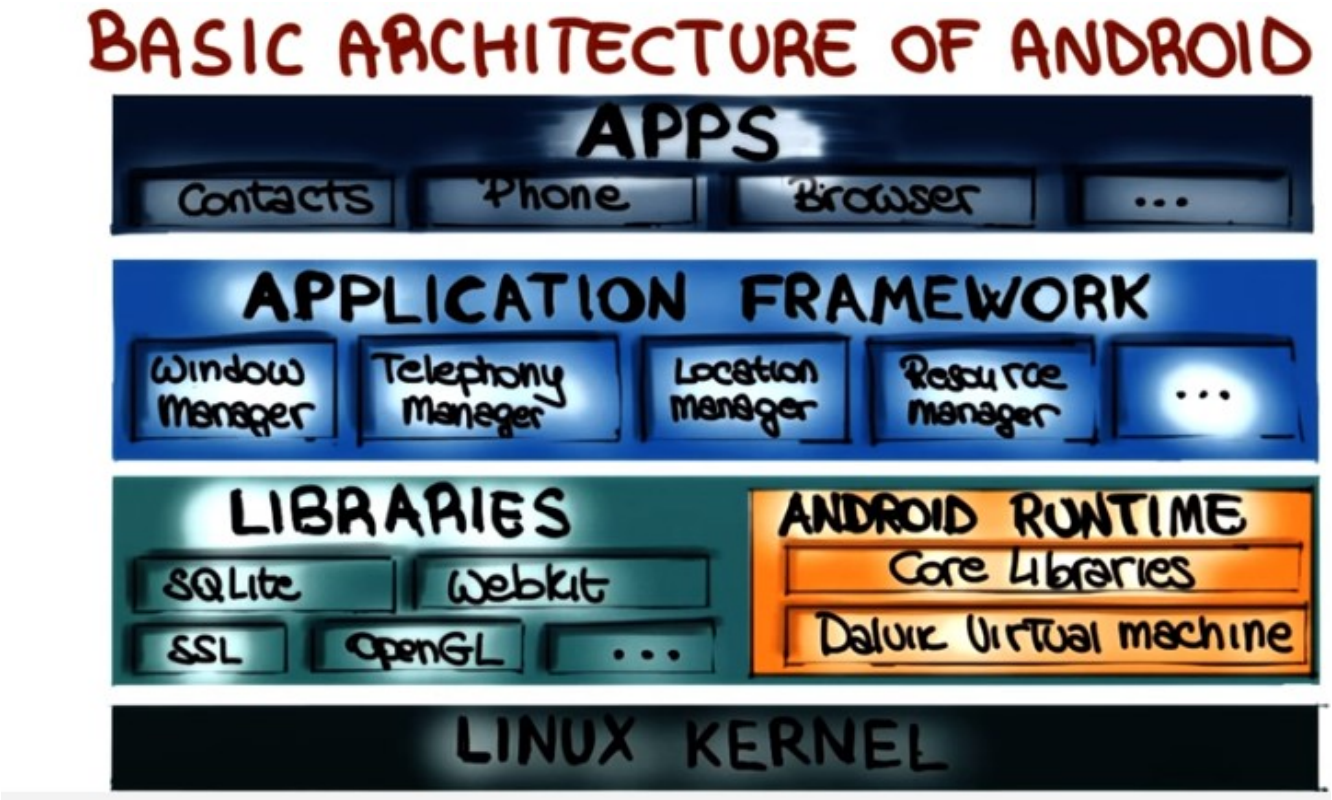
Normal CS6300 notes:

1. Hi. And welcome the third lesson on tools of the trade. In the previous two lessons, we talked about integrated development environments and version control systems. Today, we're going to talk about the android system. An operating system designed for mobile devices. Such as phones, and, tablets. And as we did for our two previous tools of the trade lessons. We will discuss both the conceptual characteristics of android, and also its practical aspects, through a demo. So let's go and see what this is all about.

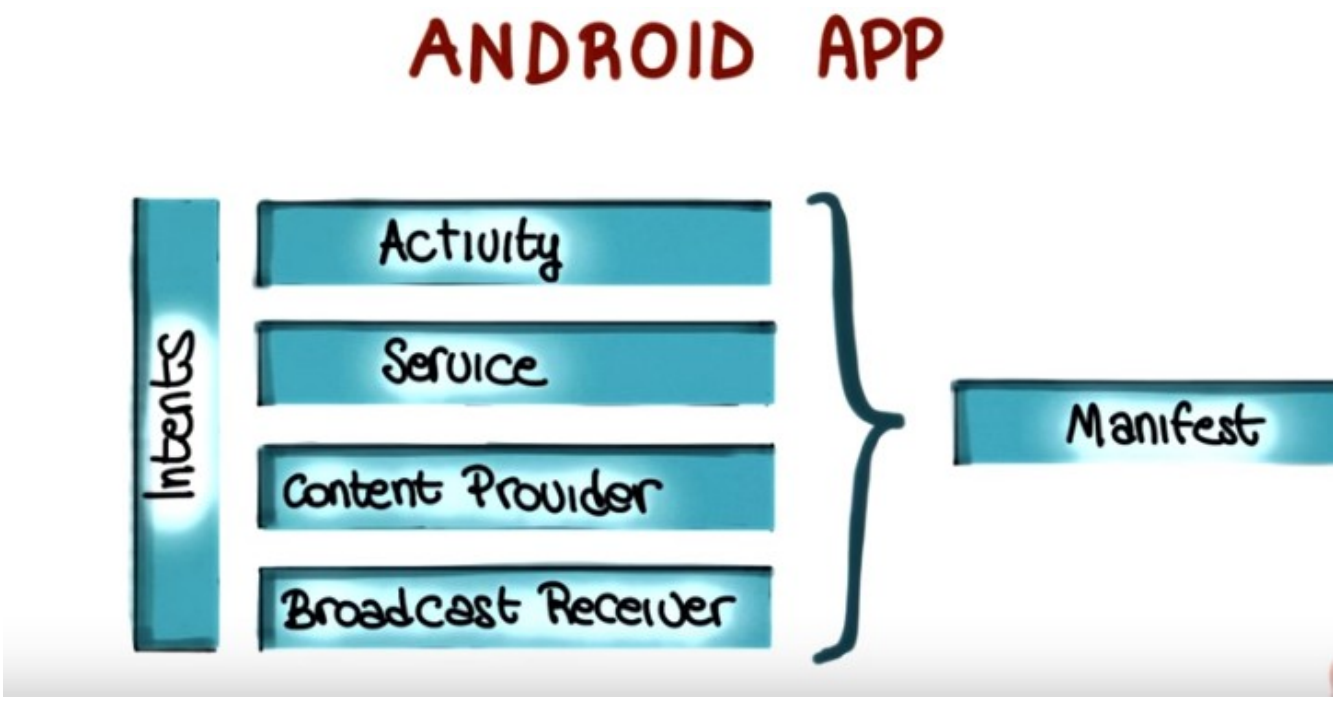
2. So what is Android? As many of you probably already know, Android is an operating system designed for mobile devices. Where mobile devices are devices that are, of course, mobile, and that are characterized by the fact of having touch screens and sensors. Typical examples would be smart phones, like the one that you probably have in your pocket, and tablets, that are very commonly used nowadays. An important characteristic of the Android operating system, it is based on the Linux kernel. So it is running customized version of the Linux kernel. And in order to be able to run application, it is powered by a Java-based virtual machine, which is called the Dalvik VM. The Dalvik VM is a type of JVM which is used in Android devices to run applications. An important point about the Dalvik VM is that it is optimized for mobile devices. What that means is that it is optimized for low processing power and low memory environments. Therefore, Android applications or Android apps, as we normally call them, are Java-based applications that run on Dalvik.

3. Now that we saw some high level information on what Android is, let's discuss the basic architecture of the Android system. Let's start at the highest level were we have the Android apps which are applications, as we just saw, that are written in Java, and that get compiled into bicode, packaged and installed on the Android system. Basically, any applications that you ever used on any Android-powered phone or tablet is an app. Typical examples will be the Contacts app, so your address book. The Phone app in your smartphone, this is the app that allows you to make and receive phone calls. The browser, and of course games are also apps. Like for example, Angry Birds or Candy Crush, just in

case you ever played any of those. Right underneath the apps, we have the application framework. The application framework is that layer that provides the services that are needed by Android apps to run. Such as the Window manager, the telephony manager, the location manager and the resource managers and there's many more. So basically, the apps rely on this layer to get information and to be able to work, to operate and to interact with the underlying phone. Going down one more layer, we have the Android native libraries which provide APIs for various functionalities. For example, the DB APIs, the webkit APIs, the SLL APIs, OpenGL and so on. For instance, SQLite is the library that provides DB application for the DB APIs that your applications can use to store information. This layer, in addition to the libraries, also contains the Android runtime which consists of the Dalvik virtual machine which we just discussed, and also the core Java libraries. Basically the core Java libraries are the Android version of the standard Java libraries, also called JDKs. Finally, the basic layer, the layer on which all of this runs, is the Linux kernel, which is the core of the Android operating system and it is a standard Linux kernel, as we said, with some customizations made by Google. The Kernel interacts with the hardware and includes all the essential hardware drivers to access the network, the file system, the screen, and so on. So basically, these layers give you an idea of how the whole system works, starting from the apps, which are at the highest level, down to the lowest level, which is the Linux Kernel that interacts with the hardware.

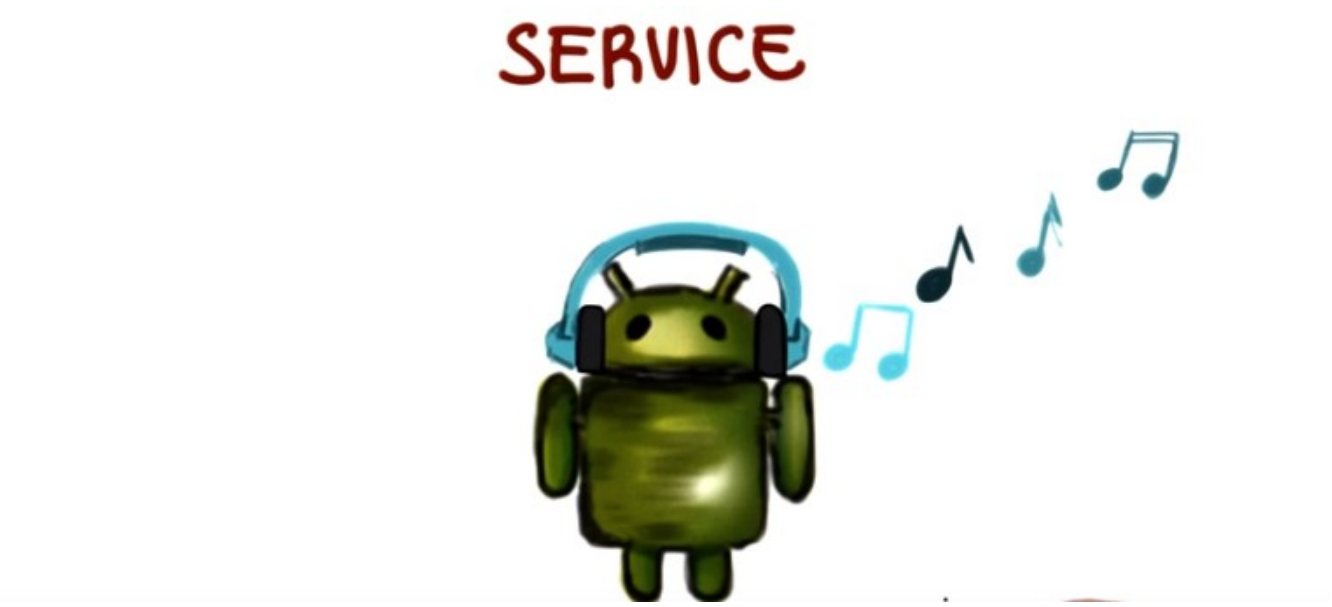


4. So now that we saw the general structure of the Android system, let's move our focus to the Android apps, which is, after all, what's we're interested in developing. An Android application is slightly different from the traditional application, as it is a collection of several components of different types. We have activities, services, content providers, and broadcast receivers. And all these four components are connected by intents and tied together by what we call a android Manifest, which is a xml file that declares the components and some properties of your application. So, now let's look at each one of those elements individually.

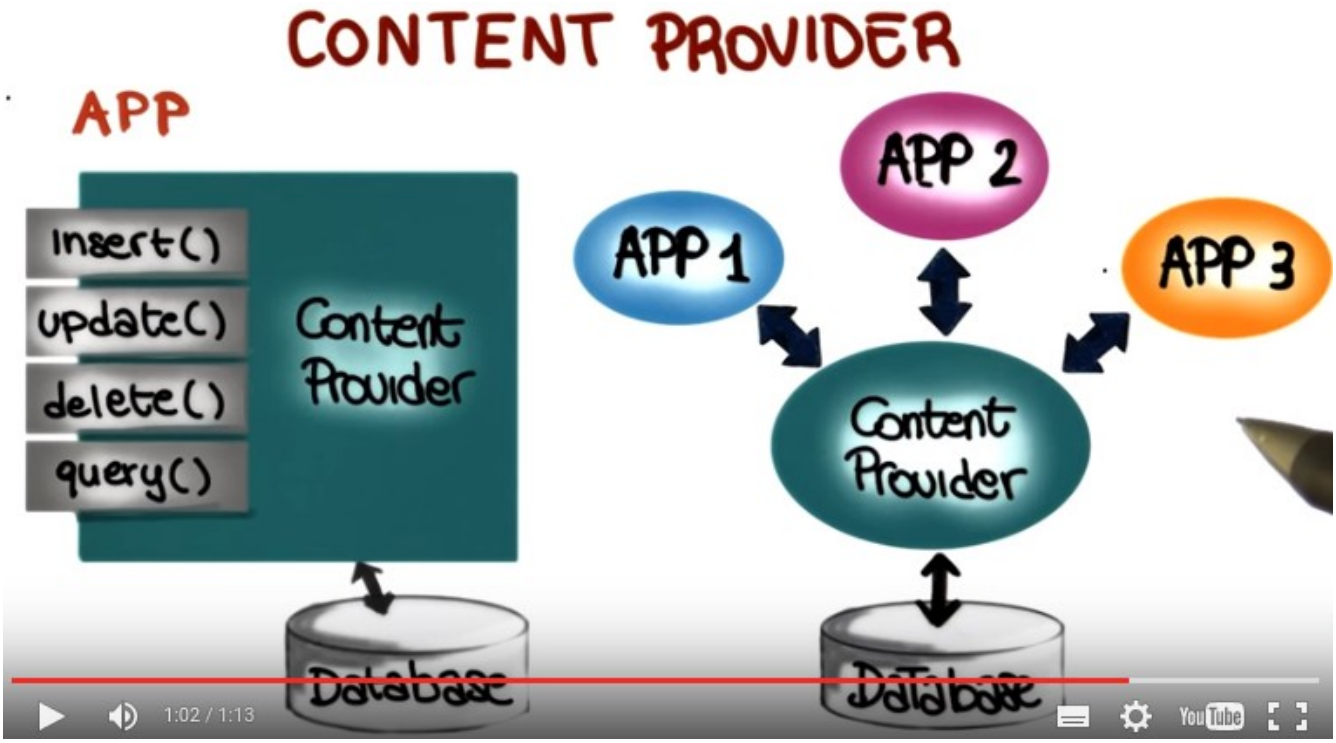


5. Let's start with activities, in Android and activity is a single screen with a user interface. What I'm shown here is the phone dialer, that your both pretty familiar with, because it is, you know, in one form or another, present in all smart phones. And this corresponds to an activity. There's another few things to know about activities. First of all, activities are independent units. But they can work together to form a cohesive whole and we'll see what that means. And one of the nice things about activities, is that they can be invoked by other applications. For example, assume that you're a runner and you're using the Run Keeper to keep track of your runs. Run Keeper can invoke a camera activity. When you're taking a picture during a run. So even though the camera activity is an independent one, it can be used by another application. Finally, from the programmatic standpoint, and as we will see in our demo, we can create an activity by simply extending the activity class, which is part of the Android system.

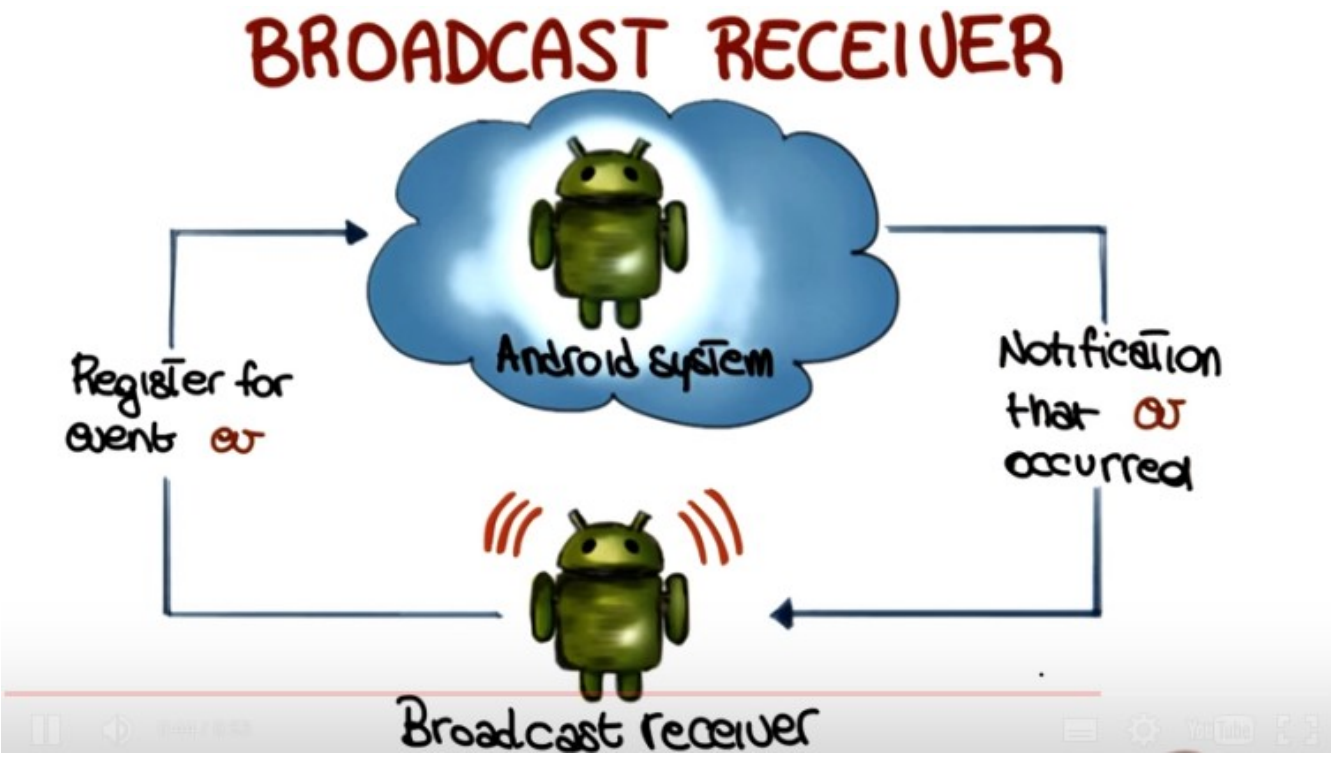
6. The next component that I want to discuss it's a service. A service is an application component that performs a usually long running operation in the background while not interacting with the user. A typical example of a service, is a service for playing music which will run in the background even when we are not using the music app directly. Another typical example is a download service, such as the one that takes care of downloading updates in the background. Once more services, they run in the background and they don't interact with the user. So they perform some long running function when not interacting with the user. Therefore, unlike an activity, services do not provide a user interface. Similar to activities from the programmatic standpoint, the way to create a service is simply to extend the service class, and we'll talk more about this. And there's just a few things that I want to mention about services, just to further clarify what they are. A service is not a way to offload some work that should be logically done by the application. But the service is a way for the app to tell the system about something that it wants to continue doing in the background, even when the user leaves the application. In addition, a service is also a way for the application to expose some of this functionality, for example the music playing functionality. By doing so, applications can bind to the service and using the service. For example, imagine a turn by turn navigation app, such an app will bind to the music player service to be able to stop the audio temporarily when making an announcement. For example, when a turn is up coming.



7. The third kind of component that I want to mention is the content provider. The content provider provides a structure interface to a set of data such as for example a set of pictures or media files in general. As we saw earlier, Android contains an implementation of the SQ Lite Database manager. So, very often, apps rely on a SQ Lite database to store the data and a provider (即 content provider 啊) to give it access to such data. In some cases, a provider can be used for accessing data within an application. In which case the data will be application data, such as a list of items to be bought for a shopping list management application. There is a much more interesting case, though. More generally a provider can be used by an application also to share data with other applications. In the case represent here for instance, this content provider is being accessed by three different application. If you want an example, a typical one will be an address book, the contact app in your smartphone, for instance. The address book could use a provider to give access to the user contacts to other applications such as, for example, a telephone application, a voice over IP application, a messaging application, and so on. So in this case, all these applications will be able to access the same data through a common, well-structured interface, which is the one provided by the content provider.

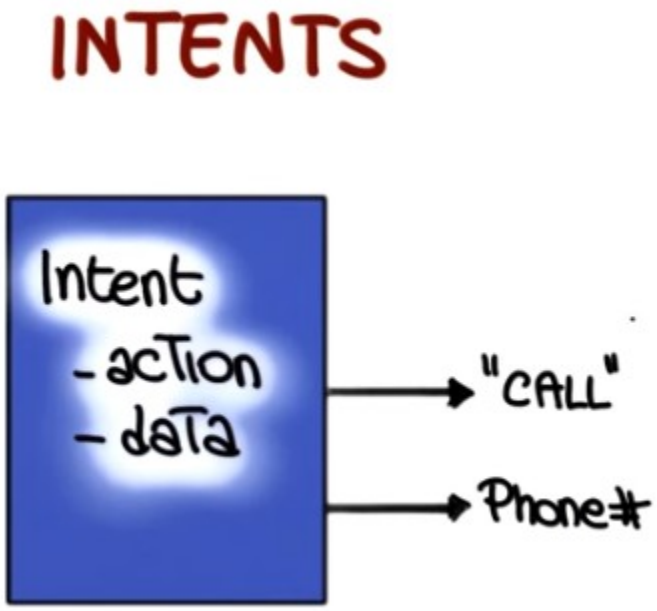


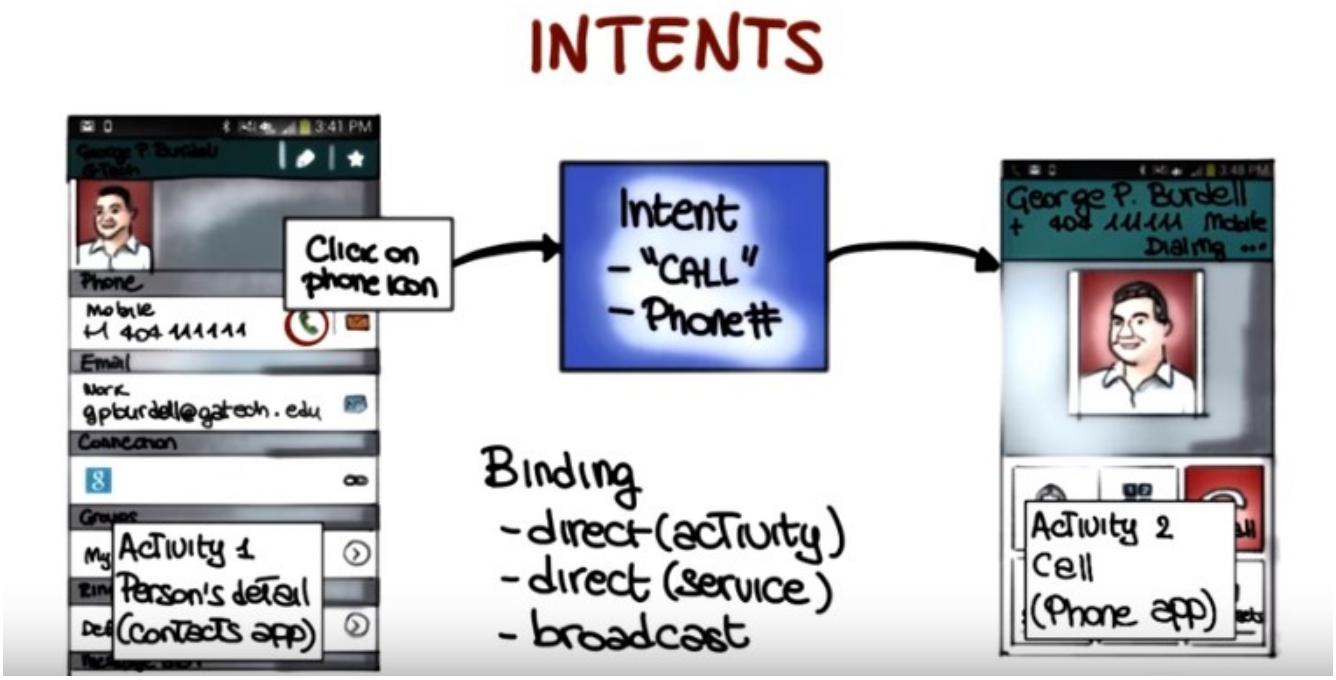
8. The fourth and final component that we are going to cover is the broadcast receiver, which is a receiver, as the name says, that can be registered to receive system or application events. To illustrate, consider here, my broadcast receiver. Where as here, we have the Android system. What the broadcast receiver can do is to register with the Android system for specific event of interest. And the Android system will notify the broadcast receiver every time an event of that kind will occur. For an example, consider a music player. A music player could use a broadcast receiver to register for the event of incoming phone calls, because it wants to suspend playing music every time that there is a phone call incoming. Similarly, it could also register for the end of the phone call because it might want to resume playing music every time a phone call is ended. So broadcast receivers are extremely useful and can be used in many, many applications.



9. As we said at the beginning, the four components that we just saw can be connected by using intents. So, what is an intent? An intent is an abstract description of an operation to be performed and it consists of two main parts, an action and a set of data. The action is the action to be performed, which is indicated as a string. And the data is the data on which the action will operate. For example, consider a phone call intent. Such an intent might have call an action. And, information from the contact database, in particular a phone number, as data. So, why are intents useful? Well, because they provide the developers with a way to perform late binding between different applications to connect applications that were not initially meant to be connected. That is, they allow for binding at run time to otherwise completely decoupled applications. So let's see an example of how this works. To do that, let's use the same example as before. So imagine that you're browsing your contacts, so activity one will be the contact application. So an activity in the contacts application. So you use that activity and you get to the person that you want to call. At this point you click on the phone icon and the result is the generation of an intent. And the intent is exactly the one that we saw before. It's an intent in which the action is call, and the data is the phone number for the person that you want to call. When this intent is generated what happens is that the call activity of the phone app is being activated with the final result that the person that you were looking up will be called by your phone. So why does that happen? How can it be that I perform an action in one activity and another activity is started as a consequence? To answer that, let's see how the binding between these two activities happens through intents. And there's different ways in which that can happen. The first possibility is a direct binding. In this case the activity or the app generating the intent will directly start an activity from another application. Similarly, the app generating the intent could directly start the service. For example, by requesting an audio player service to play a given audio file or a stream. Finally, and this is the most interesting type of binding, the activity generating the intent, so in this case the contacts application, could broadcast the event. And in this case the intent will be delivered to all interested broadcaster receivers. So we just saw the broadcast receivers and this is the way in which some information can be broadcasted. So in this case clearly it would not be a system event but it will be an application event. Going back to our example of the phone call, because the phone application previously notified the Android system that it

was supporting the action call, the system will notify the phone app that a call intent has been generated. So that's why this works. The phone app says, I can handle calls. When a call intent is broadcasted, the activity is bound to the intent. The activity is notified about the intent, and can react to it. And in case you're curious, this also the reason why for instance if you have more than one app that can make phone calls, for example you have a voice over IP application or multiple ones, and you have your regular phone application, when you select the phone icon in your contacts you are provided with a choice of which app to use to call. That's because there are multiple parties, multiple activities, that could handle that intend and therefore unless you define some default behavior you will have to choose which one should be activated as a consequence of the generation of the intent. So, similar to broadcast receivers, intents are also extremely useful in many situations to bind the activities that would be otherwise completely decoupled, as we said at the beginning.



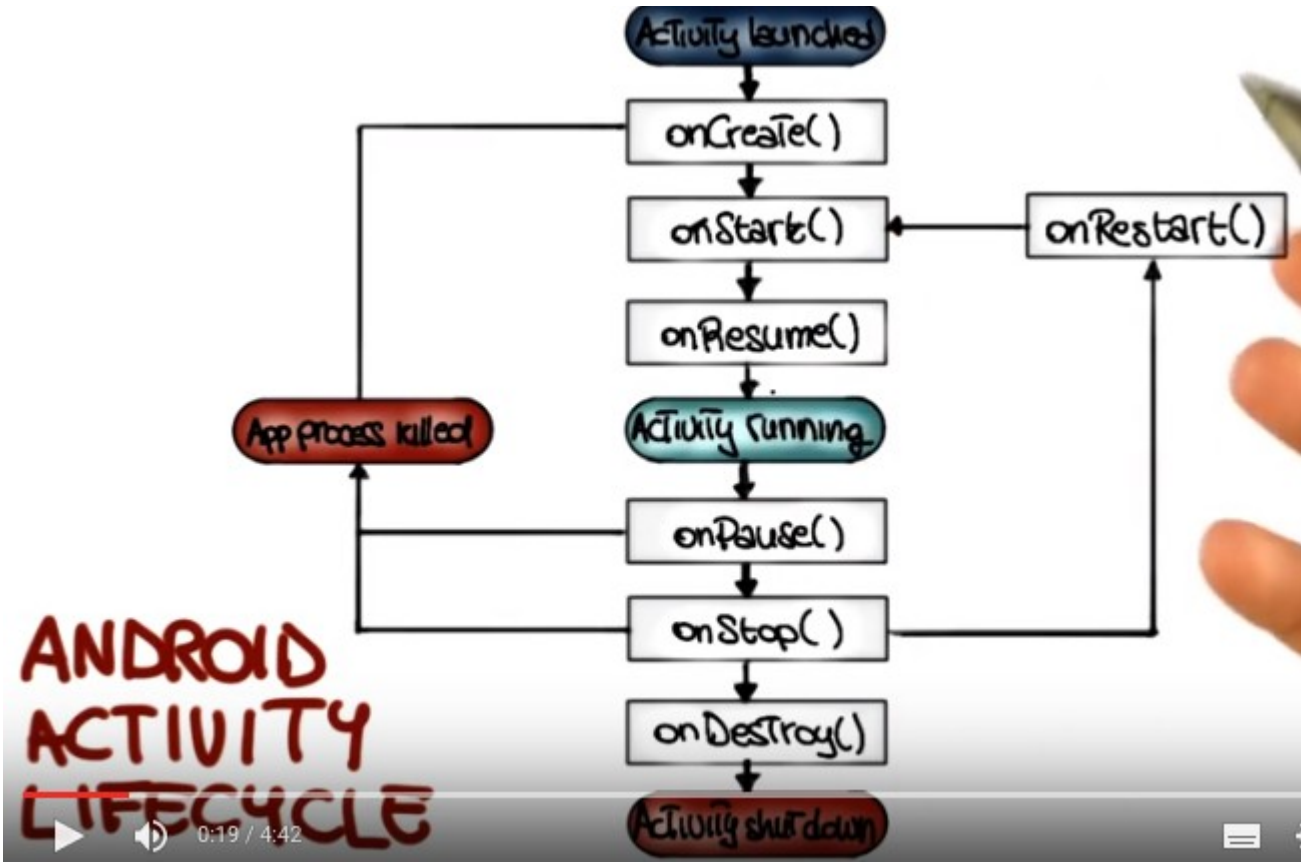


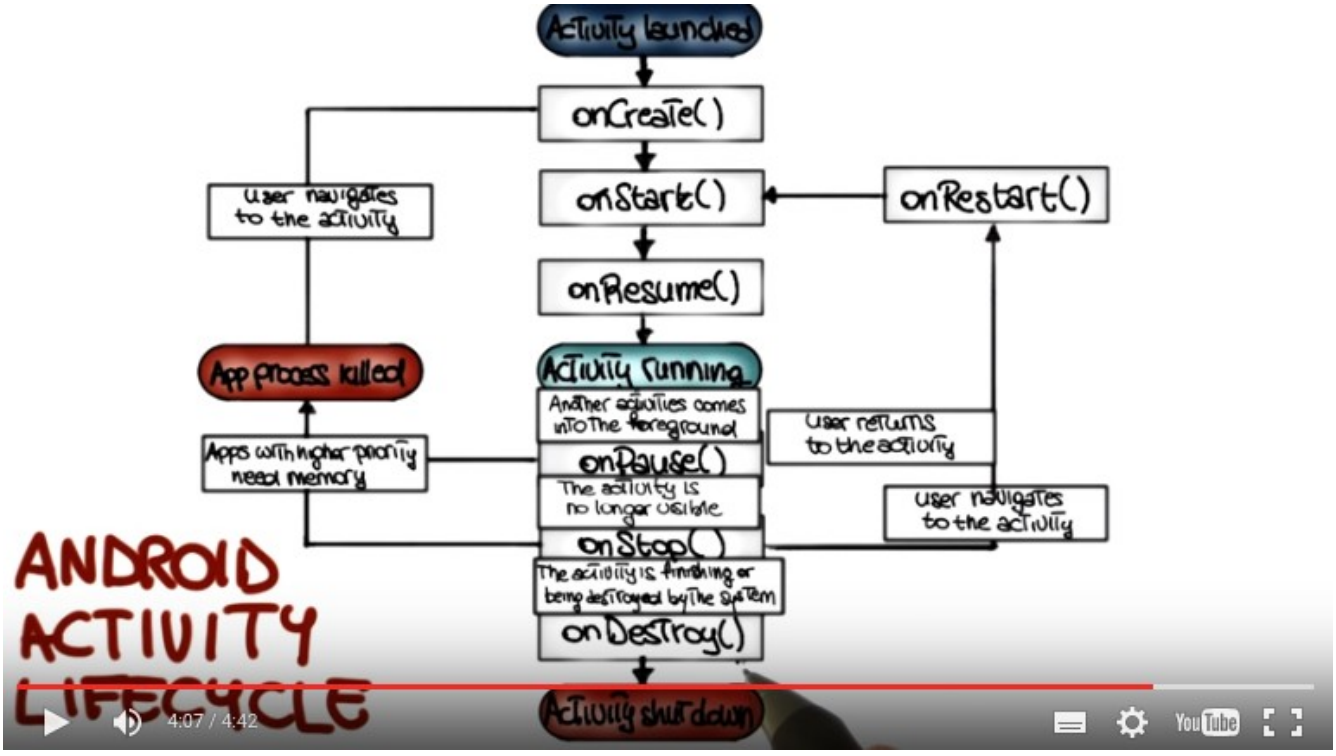
10. If you remember the beginning of the lesson, I said the Manifest is what keeps an Android application together. So what is a Manifest exactly? In a Manifest is a XML file that declares, among other things, all of the steady components of your app: activities, services and content providers. Broadcast receivers could be defined in either statically in the Manifest or dynamically in run time. So they don't necessarily have to be in the Manifest, but they might be. The second thing. That the manifest declares are all the permissions required for your app to work. For example, if the application requires network access or access to calling and messaging capabilities, that must be specified in the manifest. The manifest also specifies the entry point for the application. That is, which activity should be launched when the application is executed. So what that means is that when you click on an app to start the app up the system will look into the Manifest to see which activity has to be started as a consequence of that. So which one is the entry point for your application as we were saying. The Manifest also declares the version of the app, and this is used for updates. Because the system is able to check which version you have and which one is the latest version available for that application. And finally, the manifest also declares the lowest Android SDK version for which is app is valid. So this is something that is checked at installation time to make sure that the application can run on your version of the operating system. So again, nothing too exciting, but information that is necessary for the system to understand how your application is structured and what the different parts are and what is needed for the application to operate correctly.

11. Now that we saw the main components of an Android application, there is one last, but very important aspect of Android that we need to cover, which is the Android activity lifecycle. So I'm going to use this classic diagram to use such lifecycle. But before getting to describe the different parts, let's see why we have a lifecycle in the first place. The point here is that because Android apps run in a multitasking environment, the runtime system routinely suspends, stops, and even kills applications and activities. If you want an example, just think of the case of a phone call that arrives while you're writing an email message. In such cases, what happens is that the phone application replaces your email client and gets in the forefront. So apps must be able to react to these occurrences and suitably handle them. Luckily, the Android system is nice enough to notify activities when it needs to change their state, for example, when it needs to suspend, or kill them. And it does so by calling some special methods in the activities. Therefore, when you develop an application, an Android app, you must be aware of what these methods are, and suitably implement them. We will therefore now have a look at these methods, and see what they mean when they get called, and how this relates to the different states of an activity. And we're going to do that using this diagram that I introduced here. In order to better illustrate how this all works I'm going to complicate the picture a little bit by adding information about what happens when going from one phase of the lifecycle to another one. Because this will have to make the diagram more self-explanatory. Let's start from here, from the moment when an activity, or an app is launched. We will call our activity A, and what happens is that when A is started, the system will call different methods on the activity, and will put the activity in the foreground. What that means is that the activity will stay on the screen, visible, the user will be able to interact with the activity, and the activity will be in the running state. And so this is when you're looking at your device and you are seeing in the foreground your activity and you can interact with it. At this point two things might happen if another activity is launched. ** First thing starts ** In the first case A my lose is focus but still be visible. This is the case in which the new activity that is being created is maybe an activity that doesn't take up the whole screen, so my original activity is still visible in the background. Or a case in which the

activity(即新的 activity) is a transparent activity, so it has focus but I can still see the original activity behind it. In this case the system will call onPause on Activity A. And the activity will be in the paused state. A paused activity is completely alive in the sense that it maintains all state information and remains attached to the window manager. But what can happen is that the activity might be killed or destroyed for example in case in which there is not enough memory to run both the new activity and the original activity. ** First thing ends **

** Second thing starts ** So the second case is a case in which the new activity that is being launched is completely obscuring(隐藏, 使陰暗) my original activity. Which is a kind of more standard case. So, for example, you receive a phone call while you're editing a message. The phone app completely replaces the email client on your device, and becomes the new running activity, the one in the foreground. In this case the system will call onStop on the original activity and the activity will be in the stop state. Similar to the pause state, also in this case the activity A still retains all of state information but it is more likely that it will be killed or destroyed by the system. So lets look at what can happen from these two states in a little more detail. As we said there are cases in which an app with higher priority might need memory from the system, resources from the system. And the activity that's paused or stopped will be killed. In this case, if the user happens to go back to the activity, the system will recreate it basically, so it will restart it from scratch. So this is the reason why it is very important that when your application goes into the paused or stopped state, which means when the system calls onPause or onStop in your app, you suitably save the state of the application so that the state can be restored if the activity is killed and recreated afterwards. A second possibility when your activity is paused or stopped is that the activity is finishing or is being completely destroyed by the system. In this case, the onDestroy method is called and the activity is shut down for good. The third possibility is that the activity is not killed, is not destroyed, and the user simply returns to the activity. For example, you go back to editing your email after finishing your phone call. In this case, the system will call onRestart() on the activity and the activity will go back to the running state. So what I just covered are kind of the main paths through an activity lifecycle. But I'll strongly encourage you to read more about this, and to make sure to go through this lifecycle until it is clear to you what the different states are, and what you should do for each of them when the system invokes state changing methods on your activity.


Activity launched
onCreate()
onStart()
onRestart()
onResume()
Activity running
App process killed
onPause()
onStop()
onDestroy()
Activity shut down
ANDROID
ACTIVITY
LIFECYCLE
0:19 / 4:42


Activity launched
onCreate()
User navigates to the activity
onStart()
onRestart()
onResume()
App process killed
Activity running
Another activities comes into The foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
user navigates to the activity
onStop()
The activity is finishing or being destroyed by the System
onDestroy()
Activity shut down
ANDROID
ACTIVITY
LIFECYCLE
4:07 / 4:42

12. Now that we covered all the main conceptual aspects of the Android system, let's look at how these aspects work in practice through a demo. Let's start our Android demo by downloading the Android SDK (SDK 即 Software Development Kit), which will allow us to build Android apps. To do that, we need to go to the site developer.android.com. Here, we click on Develop. Then we click on Tools. And finally, we click on Download here on the left. This brings us to this page in which we can get the Android SDK. And we can get it as a bundle, where the bundle includes various things such as the Eclipse AD, plus the ADT plugin, Android SDK Tools and other parts. So for now, what we're going to do, we're just going to download the Android SDK. So we're going to skip that and we're going to go to DOWNLOAD FOR OTHER PLATFORMS. And here where it says SDK Tools Only, you can pick the version of the Android SDK for your platform. So if you work on a Mac, you will pick this one and, otherwise, you can pick the Linux version or the Windows version. And once you click on that, you will get the SDK package. Now that you have downloaded the Android SDK for your platform, you can unzip it and you can do that graphically or as I'm doing here from the command line. So I'm just going to run unzip and pass the file as a parameter. And I'm doing the command line version because that's, you know, a little more challenging so the, the GUI one, you'll basically just click on the archive. And as soon as we have unpacked the SDK, we can move into the android-sdk directory. We can look at the content of the directory. And as you can see, there are several files and directories inside. And the one that we're interested in right now is the tools directory. So that's where we're going to go next. And again, if you're doing this through a GUI, we'll just click on the tools directory. And here, we have a bunch of applications. These are all part of the Android SDK. And what we're interested in is this android command. So we're going to run it, which you can do again from the command line or just by clicking on it:

./android

You'll probably get an error here but don't worry about it. We're going to fix it in a minute. So when we run the command, we're going to get the Android SDK Manager, which allows us to manage the, the packages, different packages that are part of the Android SDK so to see which ones are installed to install new ones. And as you can see, you can install different versions of Android. And for each version, there is a corresponding API version. And you can also see that the, the one selected by default is the latest version, which, in this case, is Android 4.4 and API version 19. So all the corresponding parts are selected for installation and, similarly, the platform tools and the build tools are also selected. And you can see on the right that none of these packages is currently installed. So we click Install to install these nine packages. Next, we have to accept the license in order to be able to install the packages. So we do that. We click Install again. And if you want, you can also look at the log here to see what's happening during the installation. And you can see that there's an error there referring to the fact that, you know, stopping the ADB server failed and that's okay because we haven't installed the server yet. So you might see exactly the same error. And at this point, you know, just you know, you just have to wait until the installation finishes. So we're going to stop here and we're going to go to the next step.

13. Now that we have installed the Android SDK, the next thing I want to do is to install the Eclipse plug-in for the Android SDK.

Instructor Notes:
The Eclipse ADT Plugin is no longer supported, so we recommend that you use Android Studio instead.

And the Eclipse plugin will allow us to develop Android apps from within Eclipse. To do that we are going to go back to the page where we were, so the download page for the, the developer.android.com website. So as you can see, here in the download section there is an option that is Setting Up An Existing IDE, which is what we want to do because we want to set up Eclipse. And you can see that there's also another entry which is called Android Studio. And this one is a different IDE, it's kind of a more experimental one, it's a newer one, and so we're not going to use it for now, but just for your information, this is another possibility. So, if you don't want to develop from within Eclipse using the plugin you can also use the Android Studio. But for most of the examples in the class and for the project, we're going to be working with Eclipse. So, for now, it's recommended that you use the Eclipse plugin instead of Android Studio. So, we're going to ignore the Android Studio for now. Okay, so when we click on Setting Up an Existing IDE and we click on installing the Eclipse Plugin. This will bring up a page that gives us all the information that we need to install that plugin. In particular, what we're going to do, we're going to take this URL down here, we're going to copy it, and we're going to use it to download the plugin from within Eclipse. So let's switch back to Eclipse now. So, to install the plugin in Eclipse, we're going to go to the help menu, install new software, down there. And this will bring up a page in which we can add a new site. Let's just give a name to the site. Let's call it Android Eclipse plugin. And let's use the URL that we copied here, which is the location of the plugin. And after we have done that, we can click, okay. And the clips will go and download the description of the plug in from the site and as you can see there are two different parts of the plugin. There is developer tools and we just going to select all of them. And similarly there is a native development tools. That we're also going to select. And at this point, we're just going to click Next and install everything. If you installed plugins before, you know, that now you're going to be presented with a set of dependencies. We just accept the dependencies. We'll have to also accept the license for the software. Which we do we click Finish and then Eclipse starts installing the software. And just something I want to mention while the installation proceeds is that the some of the software in the plugin it's unsigned. Which means that it's you know theoretically untrusted so when, when Eclipse will get to that point, it will raise a warning, and it will notify you that you are installing unsigned software. This is exactly what is happening here. It is telling you that you are installing software that contains unsigned content. And that you can look at the details if you want, you can see which software is that. But in this case, you can safely install the software anyways. So you want to go and click okay, which would continue the installation process. And when you're done with the installation, Eclipse will let you know that you need to restart Eclipse for the software to actually install, will click yes. After restarting Eclipse, you should now see two buttons, up in the toolbar of Eclipse. The first one is the Android SDK manager. And the second button is the Android virtual device manager. But before looking at what these buttons do, let's go to Eclipse preferences up here, and we're going to look at the Android preferences that were also added by the plugin. So we go to this window, we click on Android, which will open the preferences for the Android platform. And after that, you should see the same, information that I have here on this window. So you should see that there's a version of Android installed. And also a version of the Google, API's. So now, if you expand the Android entry here on the left, you can see that there's other specific configurations that you can set. But we're not going to worry about that for now. So we're just going to close this window, and we're going to go back to the two buttons that we mentioned before. And we're going to bring bring up the SDK Manager. And this is exactly the same Android SDK Manager that we saw in a previous part of the, of the demo. But here, you can notice that the packages that we installed before, so the Android SDK Tools, the Platform-tools and the Build-tools and the Android version 4.4 and the corresponding API, are now marked as installed as indicated on the right.

14. So now, let's start the Android virtual device manager by clicking on this button, and this is the place where you can create and configure Android virtual devices, which is what we normally call emulators. So if you click on the second tab, you can see some standard pre-defined configurations.

Just as an example let's pick one, for example the Nexus 4 configuration. So this will emulate a Nexus 4 and now when we click on the Create AVD, this will create the device, and the configuration and a lot of the information is already filled out as you can see. And now we can pick the Target API and version of Android. And the next thing I will recommend to select is Snapshot in the Emulation Options because that makes the boot up time for the emulator considerably faster. So after we've done that, we'll click on OK, which will create the actual virtual device, and this is going to take just a little bit of time because the configurations being set up. And after we're done, we can see that now we have the newly created virtual device for the Nexus 4 right here in our list. So now we can select our emulator and start it. And when we click on Start we're presented with this window, in which we can select some launch options. And the one I'm going to select here is this Scale Display to Real Size, which means that we're going to see the emulated device in real size instead of full screen which will give us an idea how big is the actual display. So we click Launch. And this might take some time the first time you do it, but then it'll be faster after that. So what happens is basically, the emulator's being created and as soon as it is created, it is launched. So after the device is started, it'll start booting the operating system, the Android operating system. And when that is done you will be presented with the opening screen for the operating system, because this is the first time that we run the emulator. So the only thing I want to point out here, is that here, on the right, you have a set of buttons which correspond to the buttons that you normally have in in any of your mobile devices. So you have the volume buttons, the power button, the home button, and other navigation buttons, and you can use those to interact with other applications in the emulator. So what we're going to do next is we're going to click OK here then and we'll get to the initial screen of Android. And at this point we have our fully functional, fully configured emulator and we can run our apps on this emulator, which is what we're going to see next. We're going to see how to write applications, compile them, and run them on the emulator.

15. So let's create our first Android app. To do that, we go to the Package Explore>Contextual Menu>New>Other. This brings up the window in which we can select different types of applications and in particular Android applications. As you can see, we have several options. And the one that we want to pick is Android application project, so we select it. We click Next. So this will start this widget in which we can choose different properties of the application, in particular the name. I'm going to call the application converter because the application I want to write is a simple application that ocnverts between kilometers and miles. The next thing I can select is the Minimal Required SDK. So this is the information for backward compatibility. So it's saying that my application should run for any version of Android between 2.2 and 4.4, which is the current version. And the next thing I want to do is to change the package name, so I'm going to use the package name that we usually use for our example, so edu.caltech. Now we click Next to continue. This brings up this, new window in which we can select our configuration options. We're just not going to change any of those, so we just click Next again. More options here. So for example, you can select the foreground. You can select whether you want to trim the surrounding black space for your application. You can select the icons again. We're just going to keep the default values. And, we're going to move forward. The next thing we can do is to choose the type of activity that we want to create. And, in this case, wer'e going to go with a blank activity. And, this leads us to the last screen of the wizard in which we can pick the activity name. We're fine with Main Activity. You can choose the name for the layout and the navigation type. We're just going to keep none for the navigation type. And we click Finish. So at this point our app initial activity gets created. So now you can see that our Android project has been created, so we can click here and expand the project and see the content of the project. As you can see here, we have a source folder that contains our package and main activity. We have the libraries required for Android, an assets folder, the binary folder where our compiled code will go. Then the libraries, in particular the Android support library. And finally, a directory with all the resources for my project. So, just for example, the icons for the project, and here want to point out a few things. So, I'm going to open the layout folder and here

you can see that there's a layout file for the main activity. It's in XML format. So if I click and open this file, I can see that we have two views of this file, a graphical layout view, and, an XML view. So, the nice thing about this is that, with the visual editor I can actually drag and drop widgets into my layout, into my activity. And then those will be automatically reflected in the XML view. So, next I'd like at the strings.xml file in the values folder. So, this is the file that contains all of the strings that are defined within your application. So, it's a centralized place where you can change the strings for your app. So, next I want to look at the android manifest file which is something that we discussed in our lesson. So, basically, this is the file that defines the structure and some configuration options for our app. So, now let's go back to our main activity layout. Let me make this full screen so that we can see better, and zoom in a little bit. Let's go to graphical layout, by default it will just have a hello word string here, so we're just going to delete that. What we need first is a text view So I'm going to take the text view and drag it here. Then I can change the text in this text view by using the property window here on the right and put the text distance here and as you can see that gets reflected in the GUI. Now we're going to add a text field, a numeric text field in particular. Here it is, we want to use one with decimal values. And as you can see here, you can also check what's the distance between the different widgets. And as you can see, each of these elements has got an assigned ID, which you can see here on the right. And if you want, you can go and change that ID. And the ID is then used within the application to refer to this element. In this case, we're going to call it text one. So if you change that, Android will ask you whether you want to update all of the references to this element, because you changed its ID. So let's say Yes in this case, and that will change all the XML and Java resources accordingly. Now that we added this, let's add some radio buttons to our activity. We can do that from our phone widgets set of entries. So in particular here you can see that we have the radio button group, so we just place it at the appropriate distance from the other widgets. So, now let's go an check out our XML view, of the lay out and as you can see, all the changes that we made, all the elements that we added, are now reflected automatically, as I said, into the XML view. So, you can also change the properties and attributes of these elements here and they will be reflected in the graphic layout, as well. So, for example in this case, let's imagine that we want to delete the last radio button. So we can do it here in the XML file. Let's give some names as the first radio button. Let's call it miles and the second radio button we will give it name kilometers. So now, let's see how this is reflected into our graphic layout. As you can see, we have only two radio buttons, as expected. And the labels for these radio buttons are miles and kilometers. At this point, let's try to run this on our emulator. So to do that, we have to go to the menu up there and select the virtual device that we created earlier, the one for the Nexus 4. At this point, to run it, we minimize again our windows so that we can go to the package explorer. We select Converter>Run As>Android Application. And when we do that, the clips will deploy our applicatoin on the emulator and will run it there. And before the emulator is actually launched you will get this option in which you can decide whether you want to monitor log cat or not and what log cat basically does is to show you logs from the emulator within the ID. So normally you might want to say yes to see what's happening inside the emulator. As you can see, now we have the log cap view. Down there we're going to maximize it. You can see there's some information about the emulator. So now let's look at our console. Let's change to the Android console. And if you look at the output here on the console you can see that our activity has been started so we can switch to the emulator to see how our activity running. And in the device, you can see that we have the widgets that we added. So we have the distance, we have the two radio buttons and what we can do here, we can enter an actual value. Like for example, two, three, four, five. And the way we want our application to behave, is that when we click kilometers, the application will do the conversion automatically, and vicea versa. So let's go and implement this functionality in our app. So, to do that, I'm going to minimize the console. going to go back to my ID, and I'm going to select again, Activity Main. In this case, I want to have the XML view. And the way which I can do this, is by connecting my widget to the code and I do that by adding this entry here, in which I say that onClick so the onClick property is linked to the handle click method and

I'm going to copy this and do exactly the same for the second radio button. So what that means is that every time you click the radio button, handleClick will be invoked. So now we need to implement the handleClick method in our activity. So let's go back to the main activity, to the Java code. And as you can see, this is the code that was auto generated. So first I'm going to be writing the signature of my callback on my handle, so the name is handleClick, and the parameter is a parameter of type view. Next, we're going to check the status of a radio button that's been clicked. So we want to see whether it's checked or unchecked, and I do this using the method isChecked of the view object, of the view class. Next we want to get a reference to our text box, and we're going to do this using the TID for the text box that I mentioned before. And we're not looking at all the details of this code, but there's something I want to clarify here, is that we're using this class R here, that we haven't mentioned before. And R is basically a static class that gets associated to your Android app, and contains references to all of your resources. So for example in this case we're using R to access the IDs and within these IDs the text one ID which is the one corresponding to our text entry. And now that we have a reference to the text entry, we can actually use that to read the value that was entered by the user in that entry and we're going to use the method parseDouble of the Double class to do that. So we're going to get from the text entry the text. We're going to convert it to a string, and then we're going to parse it into a double. At this point, what we need to do is to figure out which radio button was actually clicked. If you remember, we associated the same method Handle clicker to both radio buttons. So this method gets invoked when either one is clicked. So in order to be able to distinguish between the two, and in order to know which one was actually clicked, what we need to do is to get the ID of the element that was clicked. And we do that by using the method, Get ID of the view object. Now to match the ID, we use again, class R, as before. And in particular, ID of class R, so we have the first, ID which is RID radio zero, which is the one corresponding to miles. So now we're going to use the value that we set before and if check is true. So if the radio button corresponding to the miles was actually clicked, that means that we are converting the value that was in the text entry from kilometers to miles. So we going to use this method kilometer to miles which we haven't written yet and we going to pass the distance which is the current value in the text entry to this method. This method will return the corresponding value in miles and then we'll use that to update the text in the text field. So at this point our conversion would be complete. So at this point we need to handle the second case. Which is of course the case in which we click the other radio button. So the one with id radio 1, also here, we make sure that the radio button is actually checked. And if it is, in this case we are converting from miles, to kilometers. So we do exactly the same thing that we did before, but we invoke a different method. So in this case we invoke the method miles to kilometers, again passing the distance as an input. At this point we're almost done, so we're just going to clean up the code a bit here, then I'm just going to copy and paste in the two methods, the kilometer to miles, and the miles to kilometer, in the interest of time. So I'm not going to write them in real time. And as you can see, the first method performed the conversion from miles to kilometers by using the proper conversion factor and returning a string value for the converted value and the second method does the opposite. So converts kilometers to miles. So at this point we're ready to run our code. To do that we use the run menu here, which allows to run the convertor application and as we did before, we can go and look at the, our console to figure out what's going on and as you can see the console tells us. That our app is being installed on the emulator, sucessful install, and after it's done we can switch to our emmulator and we can see our application running as we did before., but in this case, with a complete code behind it. So here we can enter, for example, our distance in miles, so let's just put one mile to see how the conversion works in this case and as you can see, the distance gets converted back and forth correctly. So, now let's use a slightly larger number. So for example, let's use the distance between the Earth and the moon and we can convert that from miles to kilometers and back to miles again. So, we have successfully written, deployed and ran in an emulator, our first Android app. And notice that you can run exactly the same app on any of your mobile devices, if you want.

16. Now that you have a basic understanding of Android development, it's time to design and build your own Android app. Please see the instructor notes for details on the project and return to this quiz once you've completed the first deliverable and let us know which get commit you want to use as your submission. Once you've submitted it, please wait for feedback from us before you go on to the next deliverable. Feel free to move on to the next mini course, but make sure you get all the deliverables for this project done on time.

~~~~~~~~~~~~~~~~

第 1 頁 activity_main.xml 的代碼:

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

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textAppearance="?android:attr/textAppearanceLarge"
      android:text="Distance Converter"
      android:id="@+id/textView"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true" />

   <LinearLayout
      android:orientation="vertical"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:layout_below="@+id/textView"
      android:layout_centerHorizontal="true">

      <LinearLayout
         android:orientation="horizontal"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:layout_gravity="center_horizontal">

         <TextView
            android:layout_width="120dp"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceMedium"
            android:text="Convert from:"
            android:id="@+id/textView2" />

         <RadioGroup
```
~~~~~~~~~~~~~~~~

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:orientation="horizontal">

        <RadioButton
            android:layout_width="100dp"
            android:layout_height="wrap_content"
            android:text="Mile"
            android:id="@+id/rbFromMile"
            android:layout_gravity="center_horizontal"
            android:checked="true" />

        <RadioButton
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Km"
            android:id="@+id/rbFromKm"
            android:checked="false" />
    </RadioGroup>
</LinearLayout>

<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <TextView
        android:layout_width="120dp"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Convert to:"
        android:id="@+id/textView3" />

    <RadioGroup
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <RadioButton
            android:layout_width="100dp"
            android:layout_height="wrap_content"
            android:text="Mile"
            android:id="@+id/rbToMile"
            android:checked="true" />

        <RadioButton
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
            android:text="Km"
            android:id="@+id/rbToKm"
            android:checked="false" />
    </RadioGroup>

</LinearLayout>

<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <TextView
        android:layout_width="120dp"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Value:"
        android:id="@+id/textView4" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="numberDecimal"
        android:ems="10"
        android:id="@+id/distValue"
        android:layout_weight="1" />
</LinearLayout>

<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <TextView
        android:layout_width="120dp"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Result:"
        android:id="@+id/textView5" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="numberDecimal"
        android:ems="10"
        android:id="@+id/distResult"
        android:layout_weight="1" />
</LinearLayout>
```

```
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="240dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal">

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Convert"
            android:id="@+id/buttonConvert"
            android:layout_weight="1"
            android:onClick="handleClick" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Reset"
            android:id="@+id/buttonReset"
            android:layout_weight="1"
            android:onClick="handleClick" />

    </LinearLayout>

  </LinearLayout>
</RelativeLayout>
```