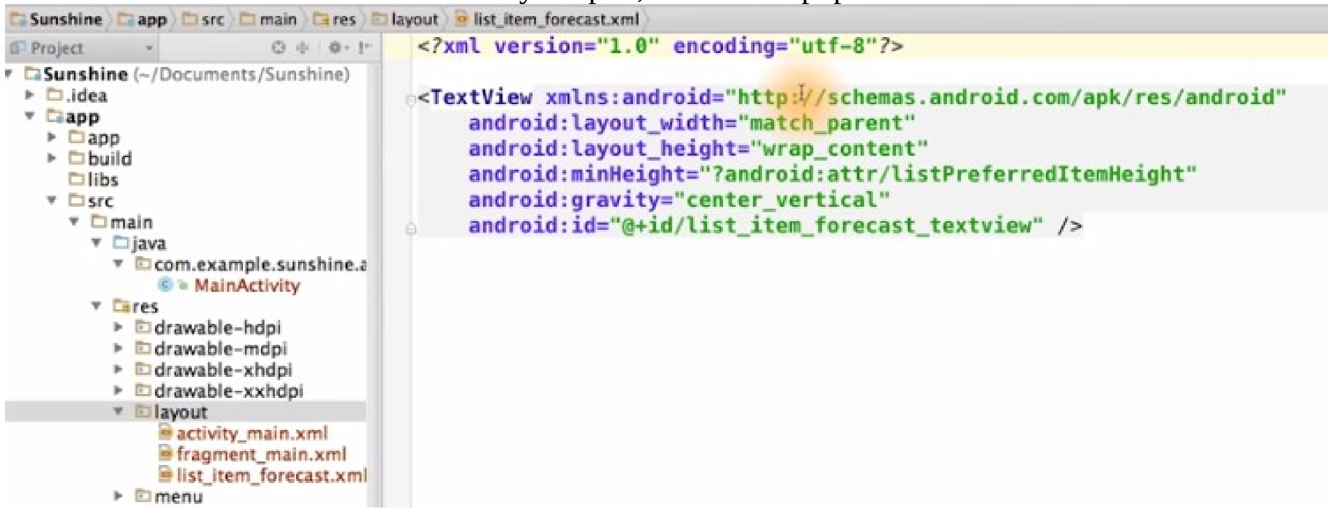
Lesson 1: Create Project Sunshine with a Simple UI

Procedures in Lesson 1:
Start Android Studio
Start a new Android Studio project
Configure your new project: Application name: Sunshine, Company Domain: android.example.com
Select the form factors: choose API 10: Android 2.3.3 (Gingerbread)
Add an activity to Mobile: Blank Activity, next, check “Use a Fragment”
Now everything is open. Choose “Project” above the catalog on the top-left.
Add a custom app icon: Right-click on the app folder, New, Image Asset, Image file
Connect phone to computer, Settings, Scroll down to About Phone, Build Number, Tap on that seven times, then when you go back to settings, you'll see the Developer options menu appear.
Hit the green run button
Choose a running device / Launch emulator

Compile and run in phone by command line (there is a way to write codes in terminal, record it later):
go to the project folder, like Sushine$
chmod +x gradlew (only for first time)
./gradlew assembleDebug
(an apk file will be built in app/build/outputs/apk folder)
adb install -r app/build/outputs/apk/app-debug-unaligned.apk
adb shell am start -n com.example.android.sunshine/com.example.android.sunshine.MainActivity

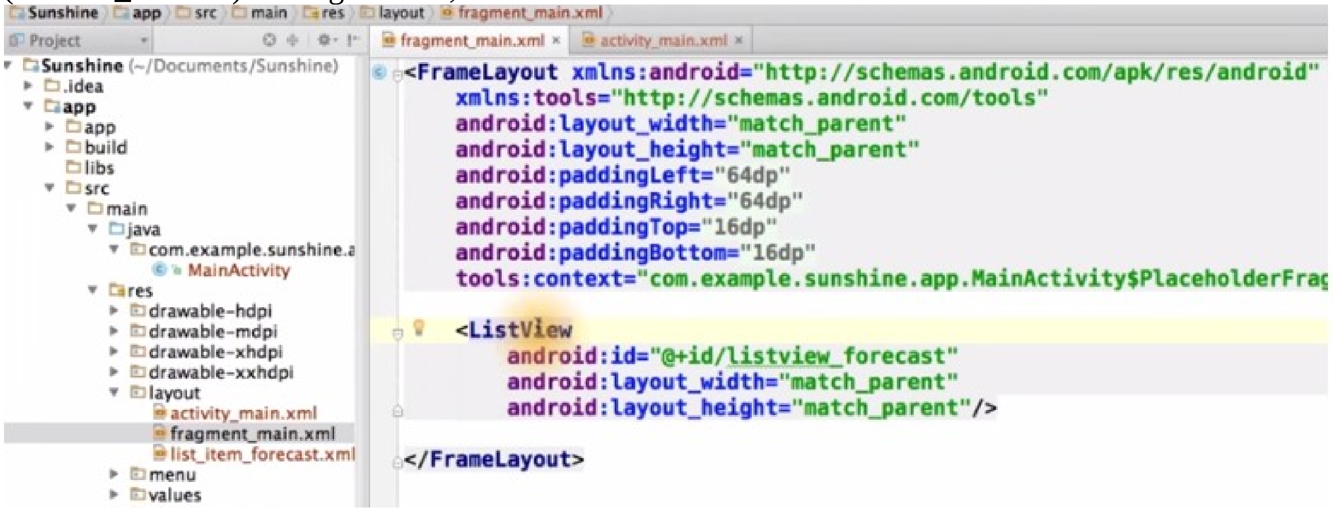
MainActivity is launched when you start the app. At the bottom of MainActivity, we have PlaceholderFragment (i.e., MainActivityFragment.java out of MainActivity in my Android Studio, see more explanations later)
Go to fragment_main.xml, choose the Design lab and drag new UI elements to the virtual phone, or choose the Text tab and edit the xml (there is a TextView there). Drag more elements to the phone.

Create a new layout resource file: res, layout (right click), New, Layout resource file, File name set to list_item_forecast, Root Element set to TextView, then switch to the Text tab to show list_item_forecast.xml. Tao: why do we want to create this list_item_forecast.xml file, and write a TextView in it? Becasue later in the ArrayAdapter, we want to populate the list view into this TextView.



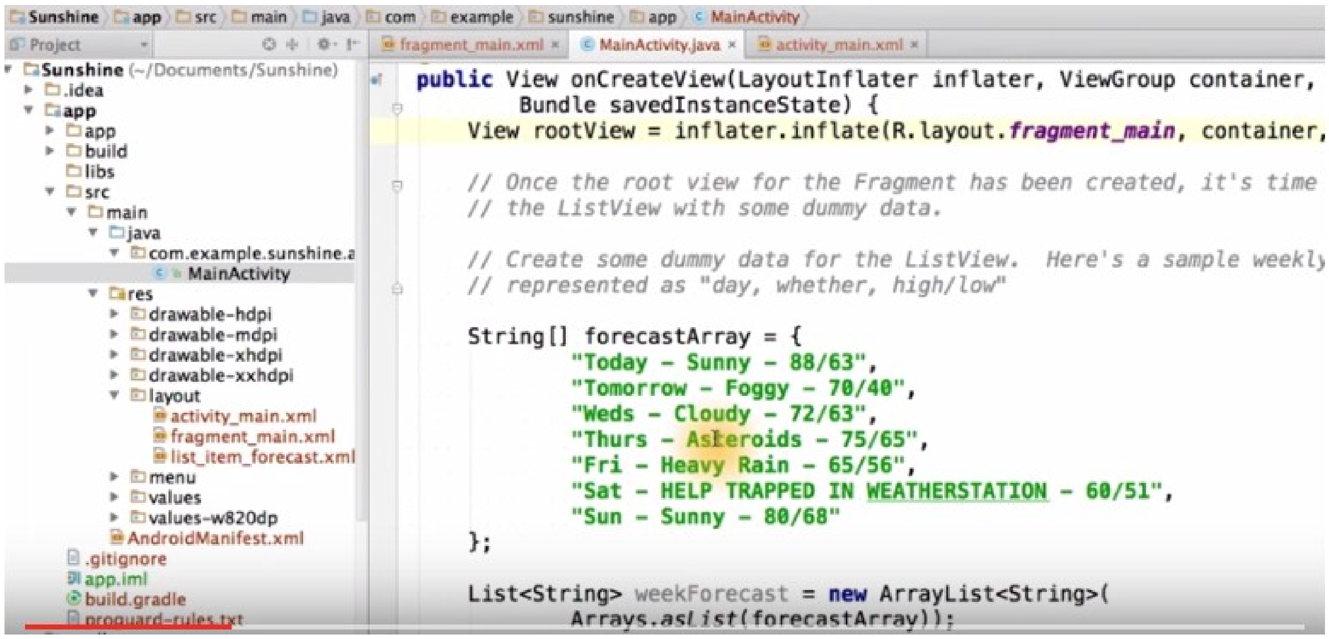
```
<?xml version="1.0" encoding="utf-8"?>

<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:minHeight="?android:attr/listPreferredItemHeight"
    android:gravity="center_vertical"
    android:id="@+id/list_item_forecast_textview" />
```

Add the list view directly to the fragment:
Open up the fragment_main.xml file, change TextView to ListView, set layout_width and layout_height to match_parent, the parent of this RelativeLayout is in activity_main.xml because the fragment is contained within the activity (should mean their corresponding java file). In activity_main.xml, we see that this view is also match_parent for height and width, so now we can confirm that the ListView actually will take up the full screnn. Going back to the fragment_main file, we can add the id onto the ListView. And then, since this layout only contains one child, we can simplifty the layout by changing this into a FrameLayout. Tao: in fragment_main.xml, a ListView is defined and an id (listview_forecast) is assigned to it, which we will use later.

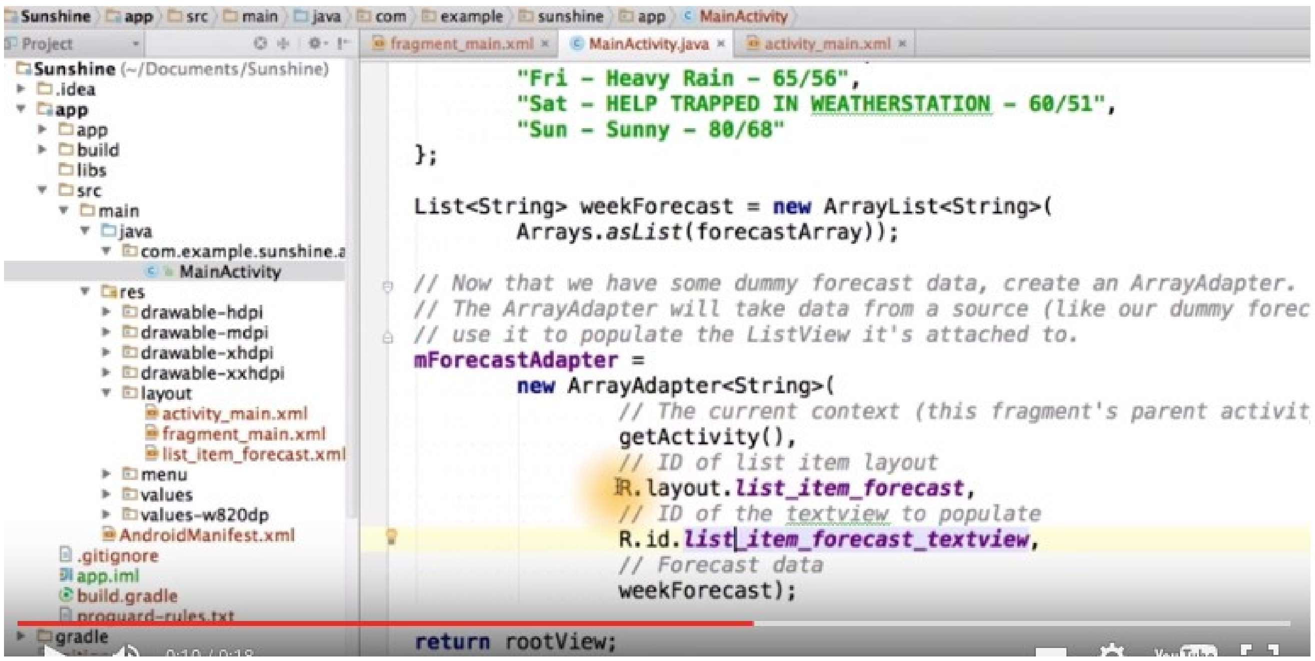


```xml
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="64dp"
    android:paddingRight="64dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context="com.example.sunshine.app.MainActivity$PlaceholderFrag

    <ListView
        android:id="@+id/listview_forecast"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</FrameLayout>
```

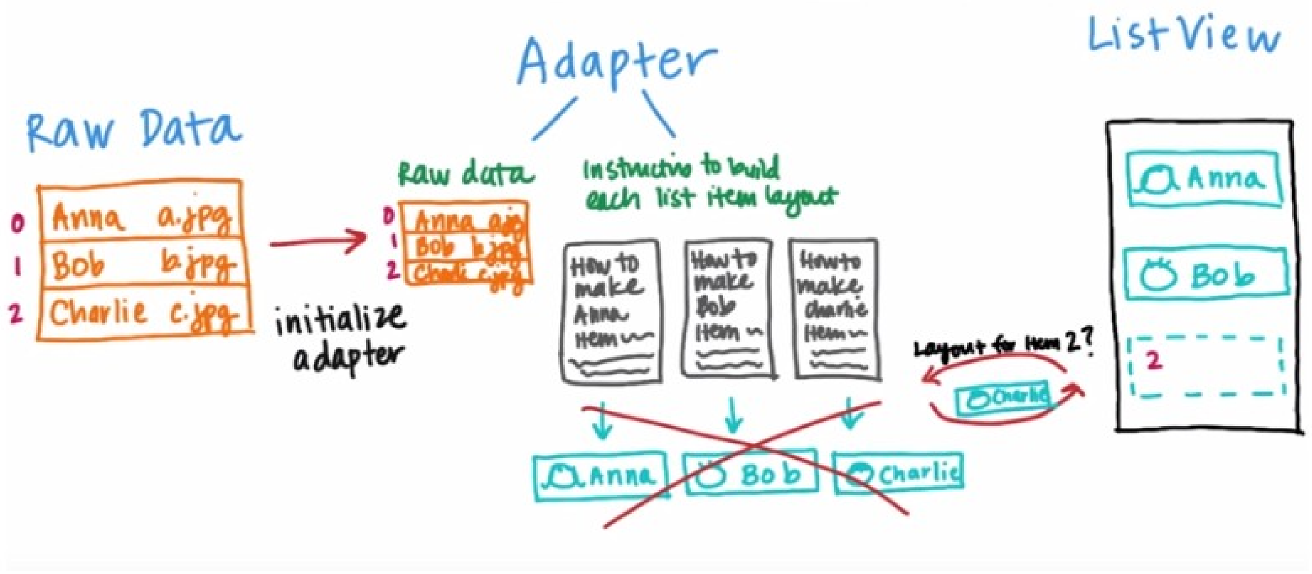
With our ListView ready to go, we'll want to create some fake data to populate it. Open up MainActivity.java and then scroll down to the bottom where the PlaceholderFragment class is (Tao: in my computer it is MainActivityFragment.java). Within the onCreatView method, create an array list of strings to represent each weather forecast item and then we turned it into an array list. Then declare your ArrayAdapter of Strings. We pass in (Tao: to the ArrayAdapter) the context, which is this fragment's parent activity, as well as the ID of the list item layout (Tao: R.layout.list_item_forecast), and the ID of the TextView we want to populate (Tao: R.id.list_item_forecast_textview, note that list_item_forecast_textview is defined in list_item_forecast.xml, so these two R are from the same file). And lastly, we pass in the forecast data (Tao: i.e. the List<String> weekForecast).



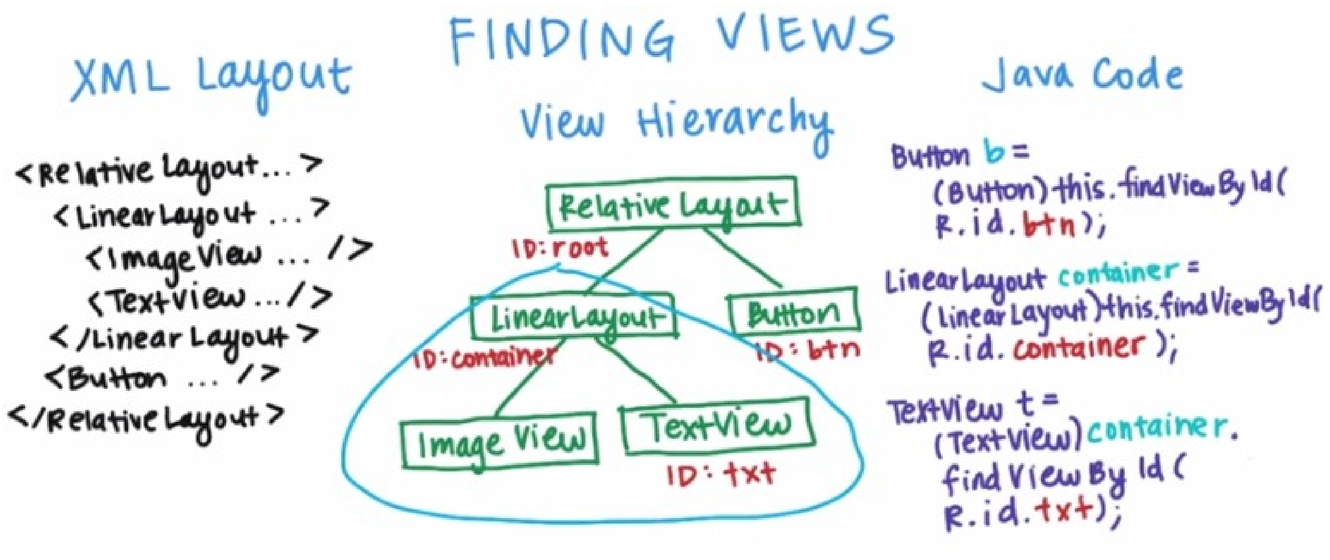
```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    View rootView = inflater.inflate(R.layout.fragment_main, container,

    // Once the root view for the Fragment has been created, it's time
    // the ListView with some dummy data.

    // Create some dummy data for the ListView.  Here's a sample weekly
    // represented as "day, whether, high/low"

    String[] forecastArray = {
            "Today - Sunny - 88/63",
            "Tomorrow - Foggy - 70/40",
            "Weds - Cloudy - 72/63",
            "Thurs - Asteroids - 75/65",
            "Fri - Heavy Rain - 65/56",
            "Sat - HELP TRAPPED IN WEATHERSTATION - 60/51",
            "Sun - Sunny - 80/68"
    };

    List<String> weekForecast = new ArrayList<String>(
            Arrays.asList(forecastArray));
```



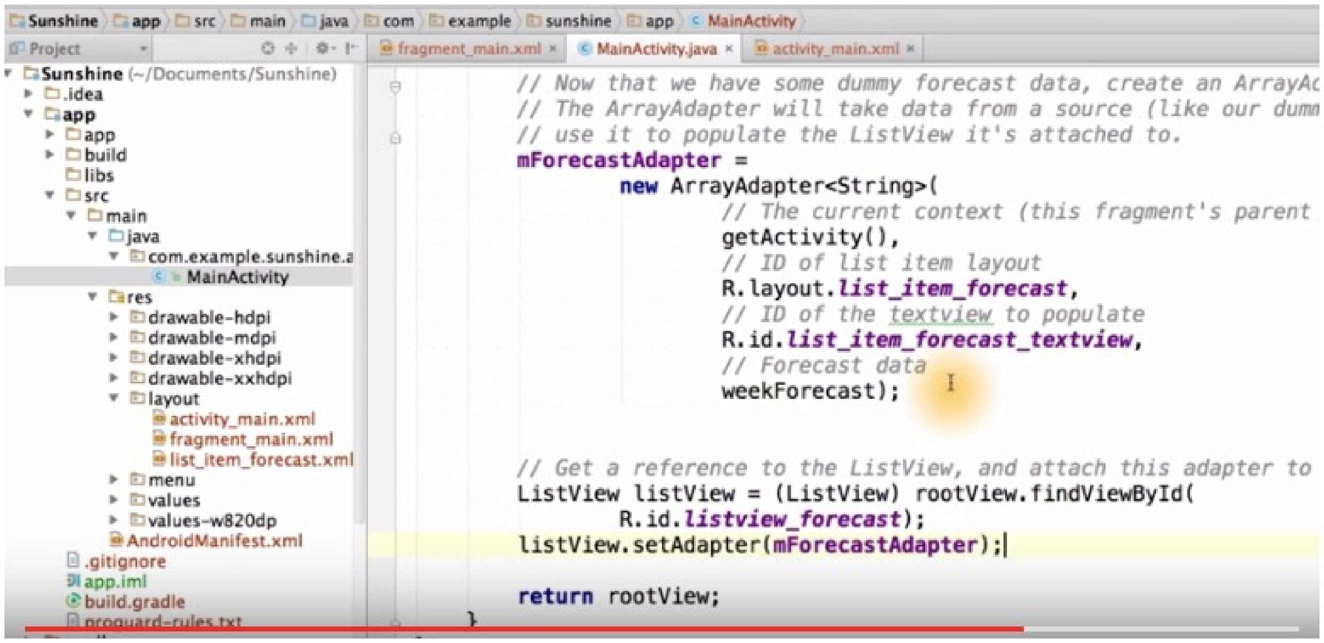
```
            "Fri - Heavy Rain - 65/56",
            "Sat - HELP TRAPPED IN WEATHERSTATION - 60/51",
            "Sun - Sunny - 80/68"
    };

    List<String> weekForecast = new ArrayList<String>(
            Arrays.asList(forecastArray));

    // Now that we have some dummy forecast data, create an ArrayAdapter.
    // The ArrayAdapter will take data from a source (like our dummy forec
    // use it to populate the ListView it's attached to.
    mForecastAdapter =
            new ArrayAdapter<String>(
                    // The current context (this fragment's parent activit
                    getActivity(),
                    // ID of list item layout
                    R.layout.list_item_forecast,
                    // ID of the textview to populate
                    R.id.list_item_forecast_textview,
                    // Forecast data
                    weekForecast);

    return rootView;
```



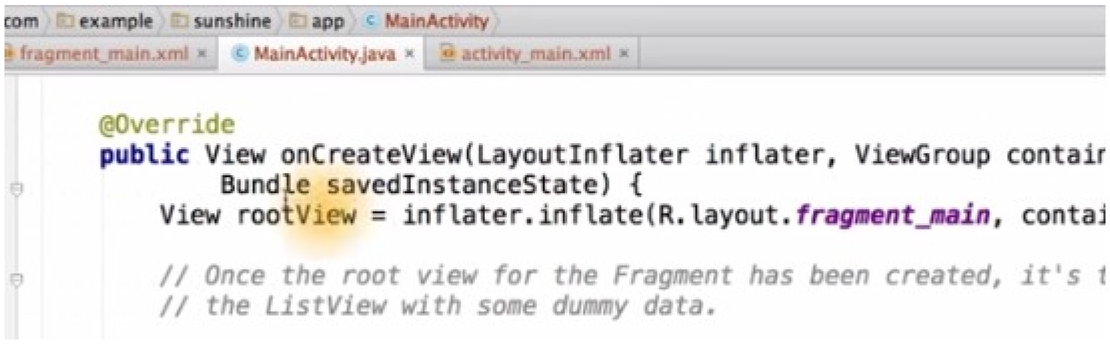
Once the adapter is initialized, let's bind it to the ListView. But you may notice that we don't have a reference to the ListView in our fragment. It was only defined in the layout XML. The system takes and inflates layout XML files, and turns them into a full view hierarchy with a root layout of the main activity at the very top of the view tree. We can also assgin IDs to each view in the tree, but it's not required if you don't need a reference to an individual view. Within the Java code of the associated activity or fragement, if we need a reference to the button, we can simply call findViewById, which will traverse down the view hierarchy until it finds a view with an ID button, and then it will return that.



Now in the Placeholder Fragment class, bind the adapter to the ListView. Luckily, we did assign an ID to the ListView earlier, so we can find it easily now. First, we find the ListView in the view hierarch by using the findViewById call and then, we set the adapter to it. The adapter will supply list item layouts to the ListView based on the weekForecast data. Note that the rootView here referes to the root veiw of the fragment, which we just inflated up above here.



```
// Now that we have some dummy forecast data, create an ArrayAc
// The ArrayAdapter will take data from a source (like our dumm
// use it to populate the ListView it's attached to.
mForecastAdapter =
        new ArrayAdapter<String>(
                // The current context (this fragment's parent
                getActivity(),
                // ID of list item layout
                R.layout.list_item_forecast,
                // ID of the textview to populate
                R.id.list_item_forecast_textview,
                // Forecast data
                weekForecast);

// Get a reference to the ListView, and attach this adapter to
ListView listView = (ListView) rootView.findViewById(
        R.id.listview_forecast);
listView.setAdapter(mForecastAdapter);

return rootView;
}
```



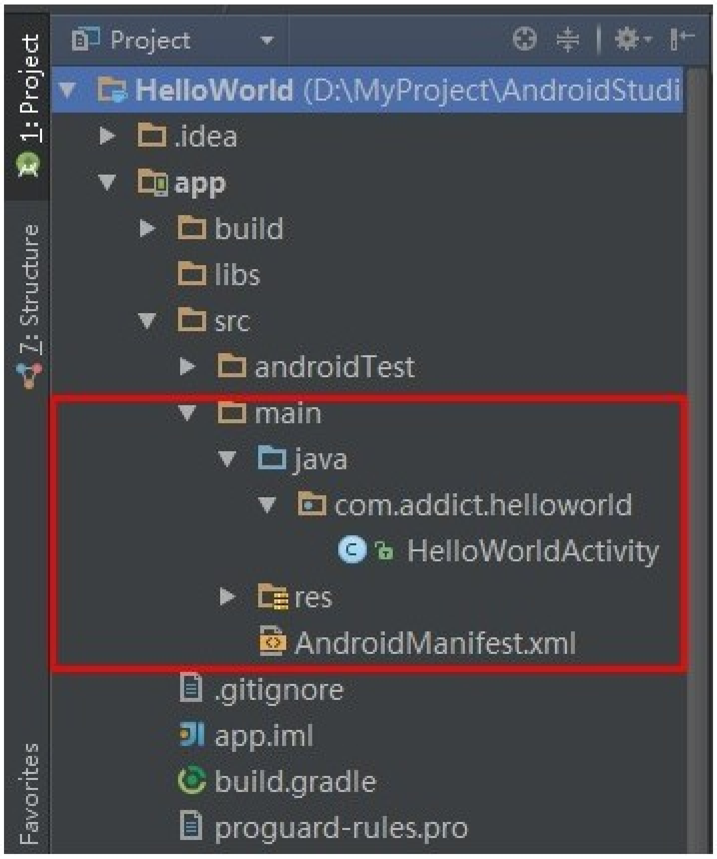
```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup contai
        Bundle savedInstanceState) {
    View rootView = inflater.inflate(R.layout.fragment_main, contai

    // Once the root view for the Fragment has been created, it's t
    // the ListView with some dummy data.
```

More explanations:

When you create a new project, you need to define a minimum SDK, or Software Development Kit. The SDK is a set of tools that you download that allows you to compile and create Android Projects. SDKs match API levels, when a new API level is released, a new SDK is released to make projects for that level. A current SDK is automatically downloaded when you install Android Studio.

Min SDK acts as your low pass filter, the lower the min SDK is, the more devices can be used, but the cost of using older features will increase. Min SDK 按字面意思理解就可以了, 你在寫軟件時指定了 min SDK, 則這個軟件只能在比 min SDK 高的版本的安卓係統中運行. Low pass filter 這個詞用來形容它, 應該是說反了(他說的那個 google play 的例子就是證据)
Target SDK is not a high pass filter. Only declare which platform you've tested. 意思就是指你開發軟件時以 target SDK 為目標, 一般選最新的版本, 如果版本更新了, target SDK 也跟着更新, 加更多的功能.

http://www.cnblogs.com/DebugLife/p/4341702.html :

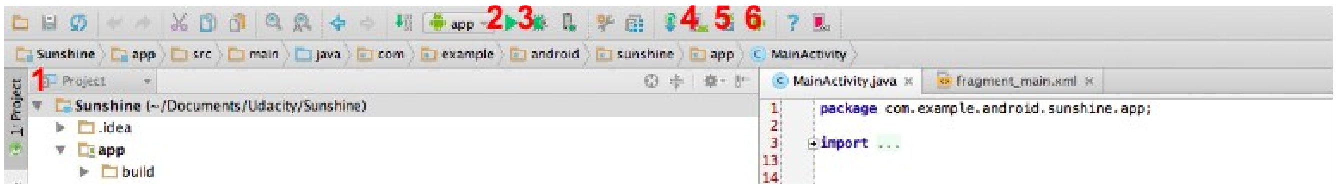


上图(Tao: 注意左上角要選 Project, 而不是默認的 Android)目录中将 java 代码和资源文件（图片、布局文件等）全部归结为 src，在 src 目录下有一个 main 的目录，同时划分出 java 和 res 两个目录。

•java 目录：放置所有 java 代码，展开后可以看到刚才创建的 HelloWorldActivity 文件。
•res 目录：放置项目中使用到的所有图片、布局、字符串等资源，后面再详细说明。
•libs 目录：放置项目中使用的第三方包，后面用到的时候再详细说明。
•AndroidManifest.xml：整个项目的配置文件，在程序中定义的所有四大组件都需要在这个文件里注册。

Ways to run a project: run in a phone by clicking buttons in Android Studio, run in a phone by command lines, run in an emulator.

Emulators are great, but they can be a bit slow. If at all possible, we suggest you use a real device. 用手機開發比用 emulator 更加 faster and smoother.



```
package com.example.android.sunshine.app;

import ...
```

1. This determines how the project structure will be shown. In the course we use the **Project** view but the **Android** view is also a handy way to organize your files.

2. This is the **run** button. It will save and compile the current state of your project as an apk and then launch it on either an Android phone connected to your computer or an emulator.

3. This is the **debug** button. It runs your code as above, but attaches the Android debugger, which allows you to do things like stop at breakpoints and step through the code.

Gradle

Gradle is the build system that packages up and compiles Android Apps. Android Studio automatically generates Gradle files for your application, including the build.gradle for your app and module and the settings.gradle for your app. You do not need to create these files. You can run Gradle from a terminal (described in this and this video), but you can also use the *run* button which will automatically run the Gradle scripts in your project.
Look at fragment_main.xml, switch to Design (rather than Text) on the bottom

Three most common layouts: frame layout, linear layout, relative layout
Frame layout: useful for simple layouts, with a single view or stack or views. Views are all aligned

against the frame boundaries only.
Linear layout: perfect for stacking vies vertically or horizontally, one after another. It is also the only way to break up the display proportionately.
Relative layout: Sophisticated layout that allows the positioning of views relative to other views or the boundaries of the view.

Activity

Activities are the components of Android apps that the user interacts with and a core class in Android. When you create an app with Android Studio, it will create an initial activity class that will start when the app is launched. The default name of this activity is `MainActivity` . An activity is a single, focused thing that the user can do and roughly maps to one screen of the app. 我: activity 就是一個 Java 程序, 可以理解為 main 函數.

Fragment

Activities can contain one or more Fragments. Fragments are modular sections of an activity, usually meant to display UI. Two activities can have the same fragment and fragments can be added or removed from an Activity. An Activity with blank Fragment is what we created for Sunshine.
The `PlaceholderFragment` is automatically generated as an inner class of the activity, but fragments don’t need to be inner classes.

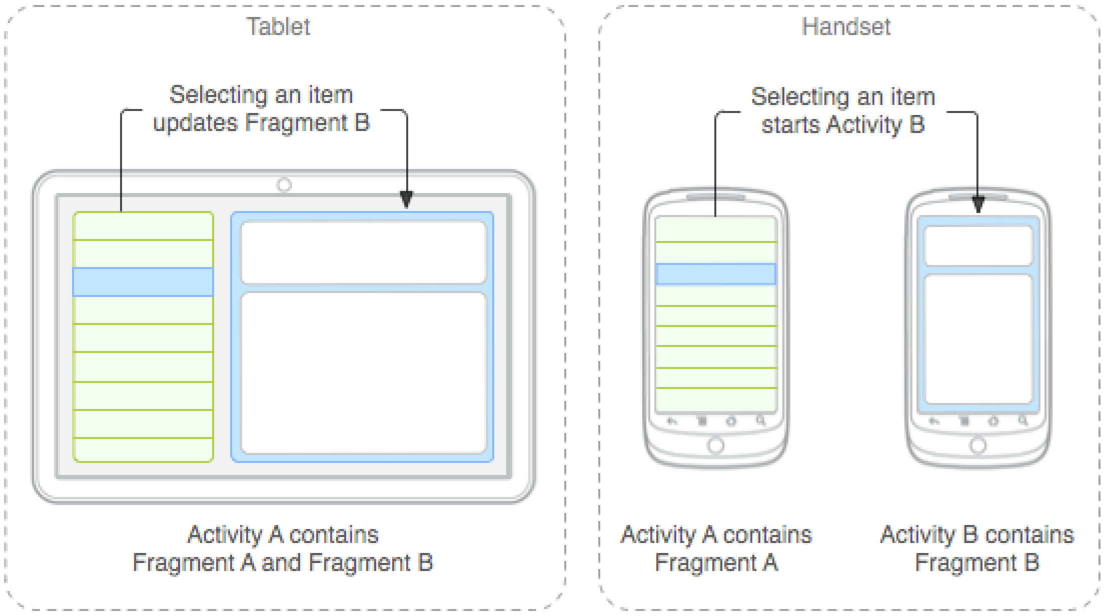
Tao: The folder src/main/java can contain classes (***.java, 注意例如 Solution 這個 class 的文件就是 Solution.java) of Activity and Fragment. The folder src/res/layout can contain xml files of layout (如 activity_main.xml) and fragment layout (如 fragment_main.xml). 所以 xml 文件都叫 layout.

Tao: 視頻中說的 PlaceholderFragment 在我的 Android Studio 中不是 MainActivity 的 inner class, 而是在 MainActivity 之外的一個叫 MainActivityFragment.java 的文件, 它與 PlaceholderFragment 是一樣的, 而且自動就跟 MainActivity 相連, 所以直接在 MainActivityFragment.java 中寫視頻中的那些代碼就可以了.

http://www.cnblogs.com/DebugLife/p/4355669.html :

- Fragment 必须嵌入在 Activity 中使用。
- Fragment 的生命周期 是受 Activity 影响的。
- 可以在一个 Activity 中使用多个 Fragment。
- Fragment 可以在多个 Activity 中重用。
- Android3.0 里出现的。
- 为了更加合理和充分地利用大屏幕的空间。

其实就是为了更好的同时支持手机和平板（我是这么认为的）。看下面这个来自 Android API Guides 中的图就明白了。


Tablet
Selecting an item updates Fragment B
Activity A contains Fragment A and Fragment B
Handset
Selecting an item starts Activity B
Activity A contains Fragment A
Activity B contains Fragment B

Views and ViewGroups

A view is the basic building block for user interface components. A fragment might combine multiple views to define its' layout. Buttons, text and other widgets are subclasses of views and can be combined in ViewGroups to create larger layouts. Common ViewGroups include:

- LinearLayout - For horizontal or vertical collections of elements.
- RelativeLayout - For laying out elements relative to one another.
- FrameLayout - For a single view.

Since views nest within other views, this creates a tree like structure of views for every layout.

Views and XML Layouts

To describe our user interface, we describe layouts using XML. The layout defines a collection of views, view groups and the relationships between them. Our layouts are stored in the app/src/main/res/layout directory. To turn an xml layout into java view objects, we need to **inflate** the layout. After the layout is inflated, we need to associate it with an Activity or Fragment. This process of inflating and associating is a little different depending on whether it's a layout for an Activity or Fragment.

For an Activity

We inflate the layout and associate it with the Activity by calling the `setContentView` method in `onCreate` in our Activity:

```
setContentView(R.layout.activity_main);
```

For a Fragment

In our Fragment classes we inflate the layout in the `onCreateView` method, which includes a `LayoutInflater` as a parameter:

```
View rootView = inflater.inflate(R.layout.fragment_main, container, false);
```

The root view, or view element which contains all the other views, is returned by the `inflate` method of the `LayoutInflater`. We then should return this rootView for the `onCreateView`.

ListView

ListView is a subclass of View optimized for displaying lists by displaying many copies of a single layout. We are going to use a ListView to display our weather information in Sunshine. Each row of weather information is defined by a layout called `list_item_forecast.xml`. The list view contains multiple copies of list_item_forecast.xml, one for each row of weather data.

An `Adapter` is used to populate a ListView.

Adapter

Adapters translate a data source into views for a ListView to display. In our case we used an ArrayAdapter to take an array as our data source and populate our ListView with the data from the array. Here’s the corresponding video about how this process works.