Special Interest Group on Innovative Software 2011-2012

Workshop on Android app development

Session 1: Application Fundamentals

Contents

Prepared by Mr. Felix Tam, Committee Member of the Special Interest Group (SIG) on Innovative Software 2011-2012, Department of Computer Science, Hong Kong Baptist University.

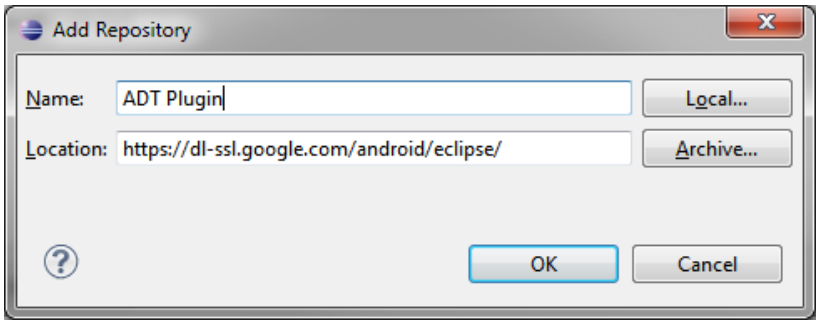
1 Setup development environment and emulator

Tools that you will need for Android app development:

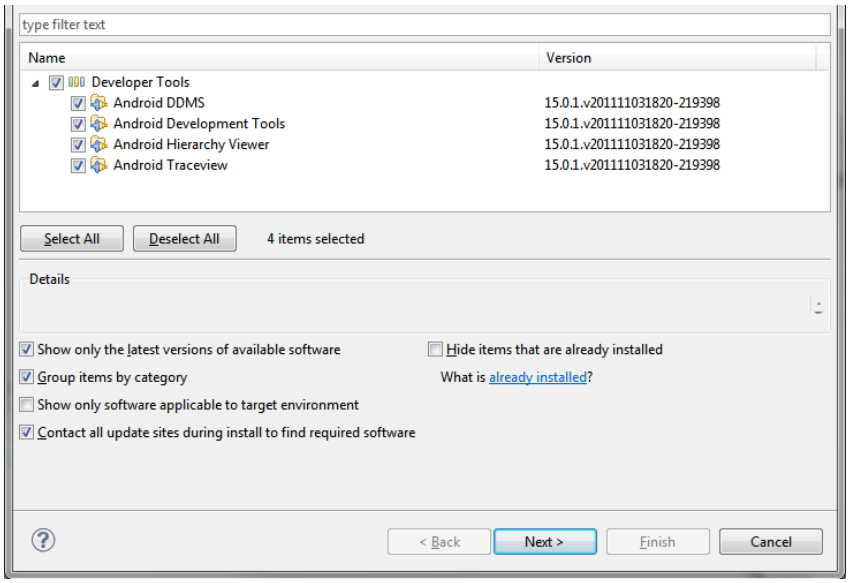
1. Eclipse IDE for Java Developers
2. Java SDK
3. ADT plugin for Eclipse
4. Android SDK

1.1. Download and install the ADT

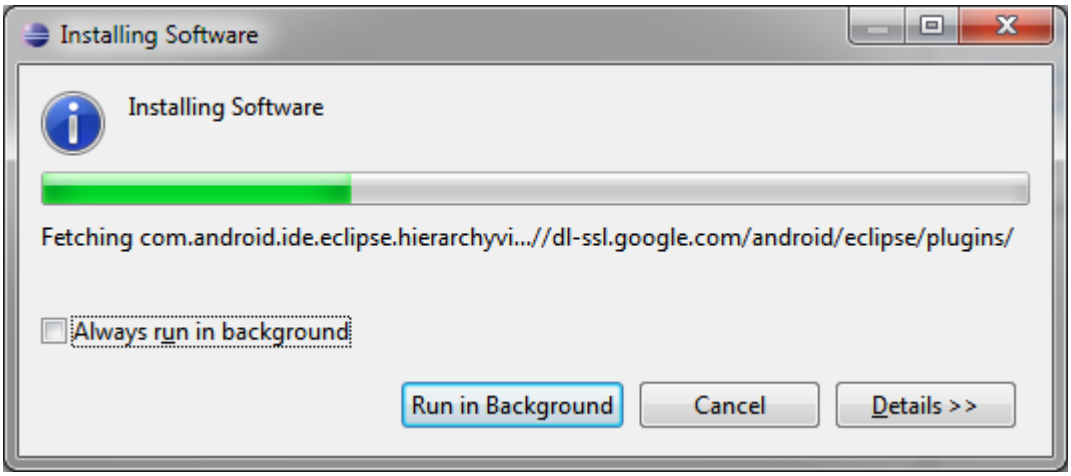
1. Make sure you have the JDK installed on your computer. If no, please download it from http://www.oracle.com/technetwork/java/javase/downloads/index.html and select the version according to your operating system.
2. Start Eclipse, go to **Help > Install New Software**, click on the **Add** in the top-right corner. (You may get Eclipse IDE for Java Developers at http://www.eclipse.org/downloads/)
3. In the **Add Repository** dialog, type **ADT Plugin** for the name and **https://dl-ssl.google.com/android/eclipse/** for the location respectively.



4. After a while you will see an option called **Developer Tools**. Select the checkbox next to those options and click **Next** to continue.



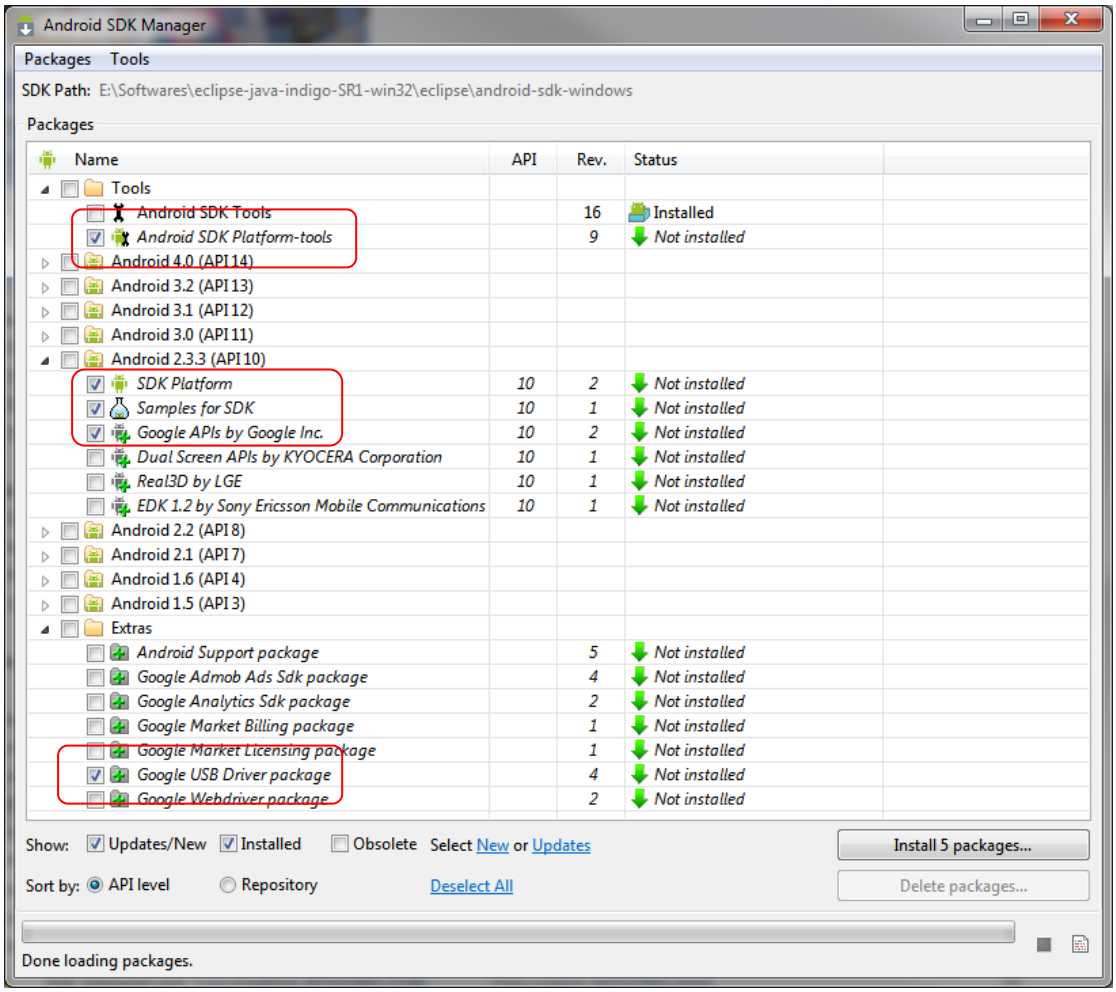
5. Click **Next**, **Finish** and accept the terms of the license agreements in order to get the ADT plugin from the remote site. When you are asked for unsigned content, click **OK** to accept. You have to restart Eclipse after the installation is finished.



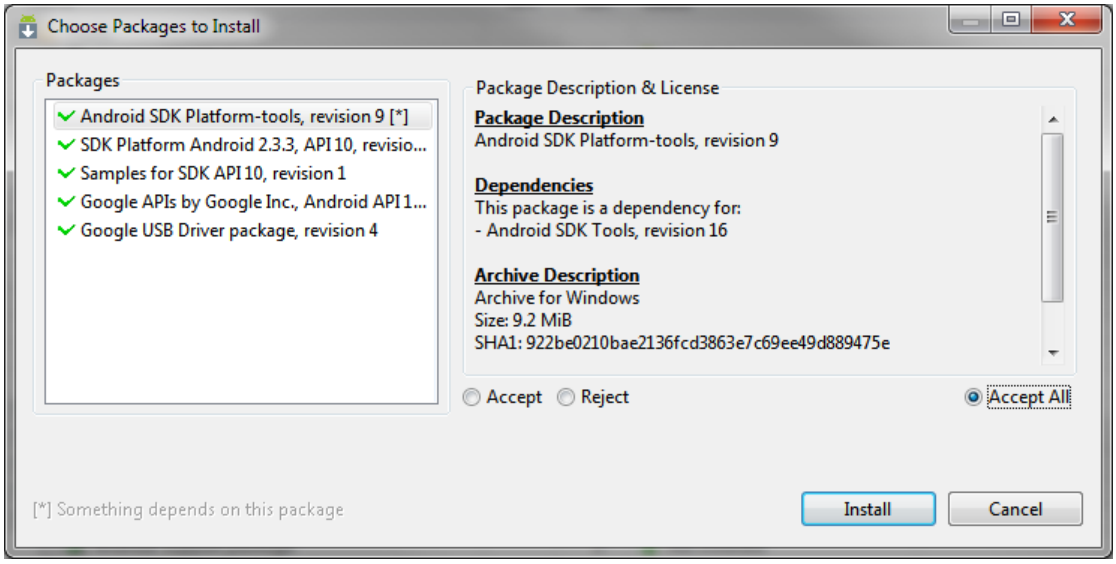
6. After restarting Eclipse, you will be asked to configure an Android SDK as well. Click **Cancel** at this moment since we will download and configure the SDK manually.

1.2. Download the Android SDK

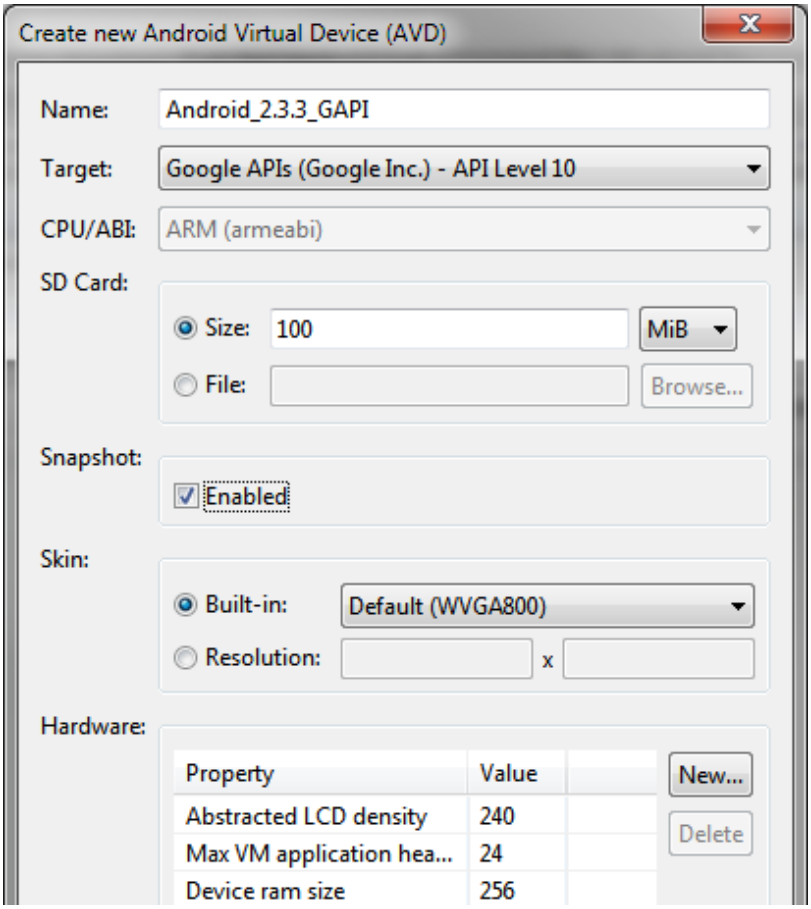
1. Go to **http://developer.android.com/sdk/index.html** and download the latest SDK in zip format.
2. Unzip the zip file and you will get the folder **android-sdk-windows**. Put the SDK folder to your favorite location. In this tutorial, we will put it under the Eclipse root folder.
3. The SDK does not contain any Android platform binary initially. If you need to develop an app which bases on Android 4.0, you have to download the Android 4.0 platform binary and other related stuff. In this workshop, all of our tutorials and examples will base on Android 2.3.3. To download Android 2.3.3 platform binary, please navigate to your SDK folder and start **SDK Manager.exe**.
4. Download and install the packages according to the screenshots:



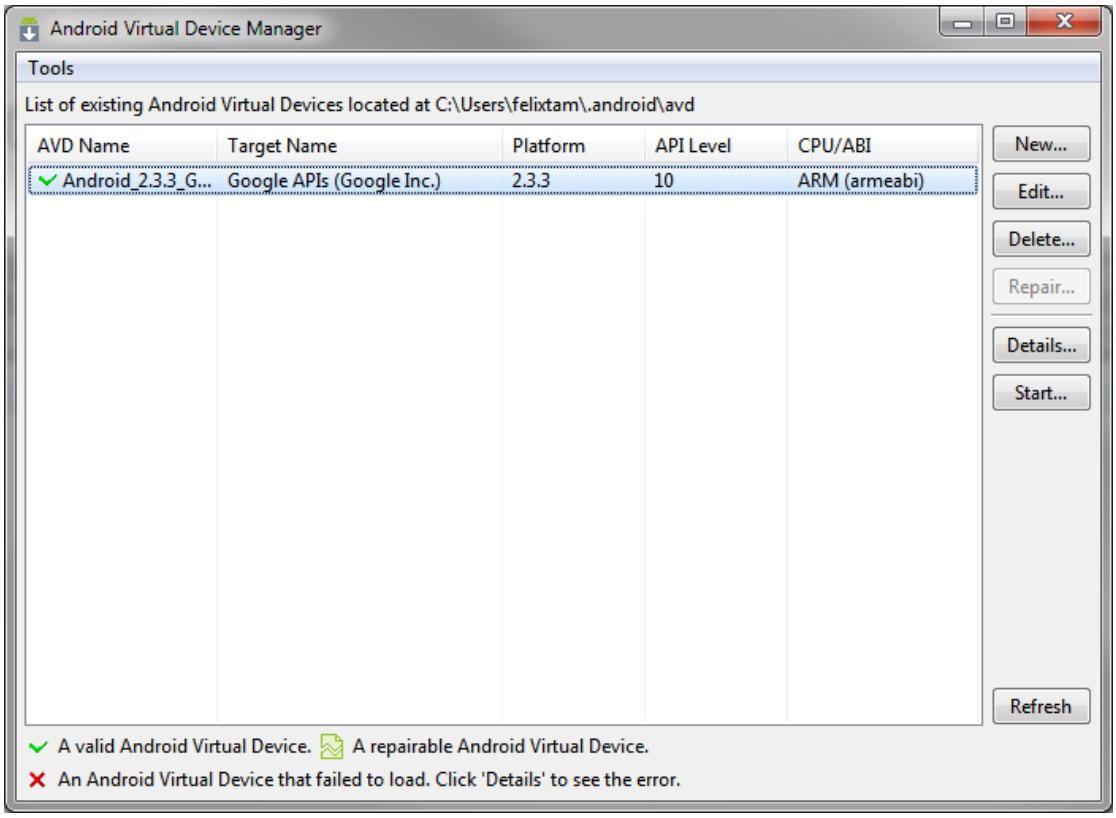
5. Choose **Accept All** and click **Install**. This should take a few minutes.



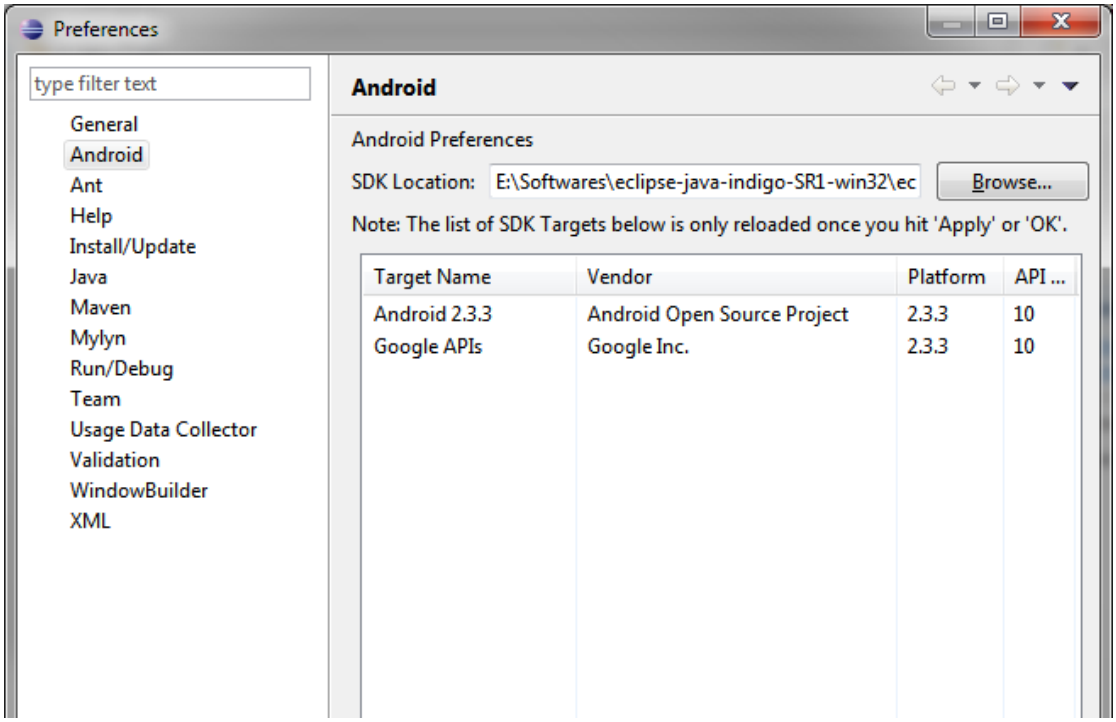
6. When you are asked to restart ADB, choose **Yes**.
7. We can now create an Android 2.3.3 emulator for testing and debugging. Go to the SDK folder and start **AVD Manager.exe**.
8. In the Android Virtual Device Manager, click **New** to create a new Android emulator.
9. Please refer to the screenshot below for the options in creating the emulator.



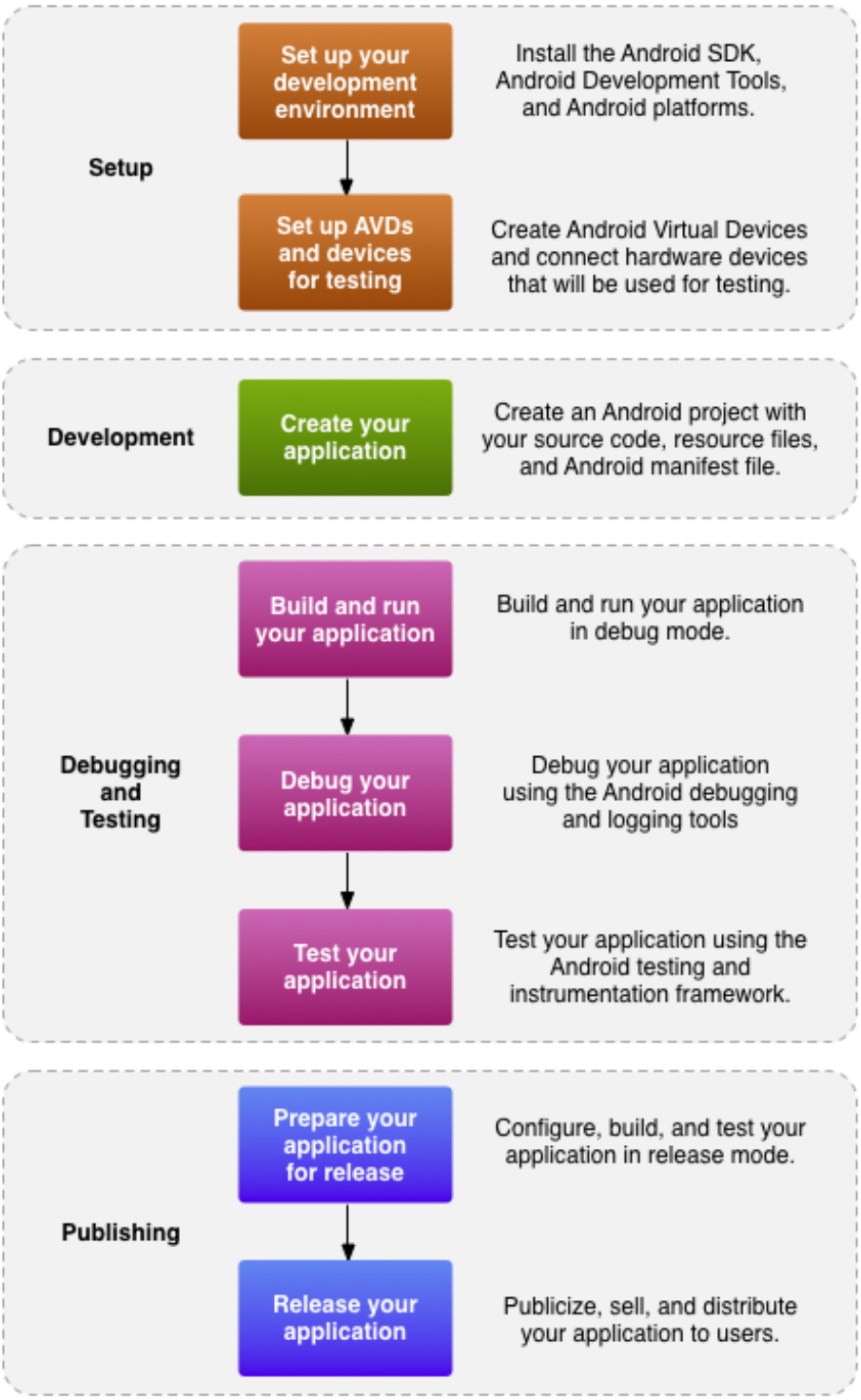
10. You can create emulators as many as you want. Please make sure you have downloaded the corresponding Android platform using the SDK Manager before making an emulator.



11. We also need to specify the SDK directory in Eclipse. Inside Eclipse, select **Window > Preferences > Android**, click **Browse** to choose the location of your SDK directory.



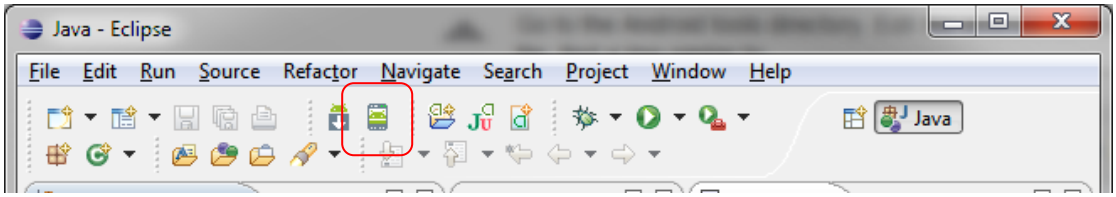
2 Steps for developing Android apps



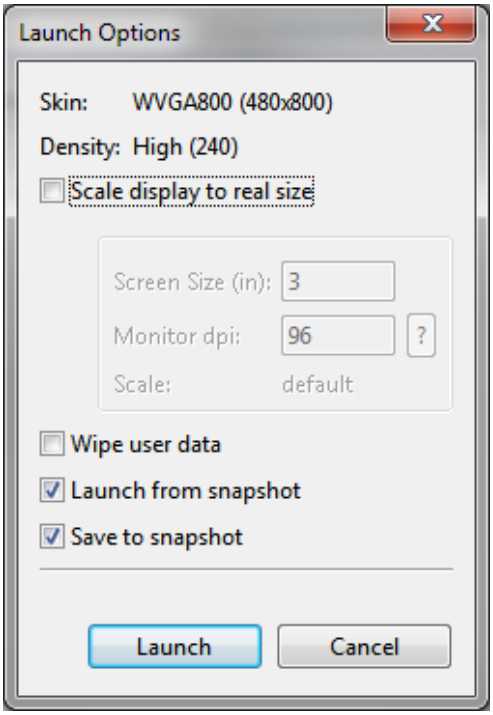
Source: Google Inc., 461px x 738px, http://developer.android.com

3 Control the emulator

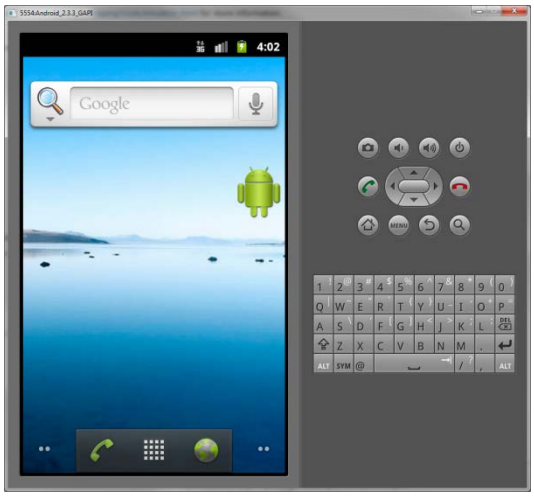
1. To start the emulator, go to the **Android Virtual Device Manager**. You can start the manager by double clicking the **AVD Manager.exe** in the SDK directory or the icon in Eclipse toolbar.



2. Choose the emulator, click **Start** and **Launch**.



3. After the emulator has successfully booted, you will see the following the screen. (It may take several minutes to boot up the emulator for the first time.)



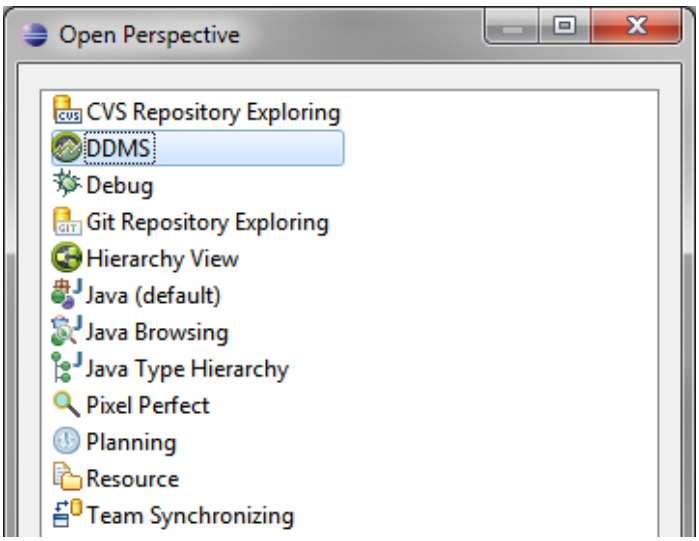
4. Here is a list of shortcut that you may use it during your development:

Emulated Device Key	Keyboard Key
Home	HOME
Menu	F2 or Page-up button
Back	ESC
Call/dial button	F3
Hangup/end call button	F4
Search	F5
Power button	F7
Switch to previous layout orientation (for example, portrait, landscape)	KEYPAD_7, Ctrl-F11
Switch to next layout orientation (for example, portrait, landscape)	KEYPAD_9, Ctrl-F12
Toggle cell networking on/off	F8
Toggle full screen mode	Alt-Enter
Toggle trackball mode	F6
Enter trackball mode temporarily (while key is pressed)	Delete

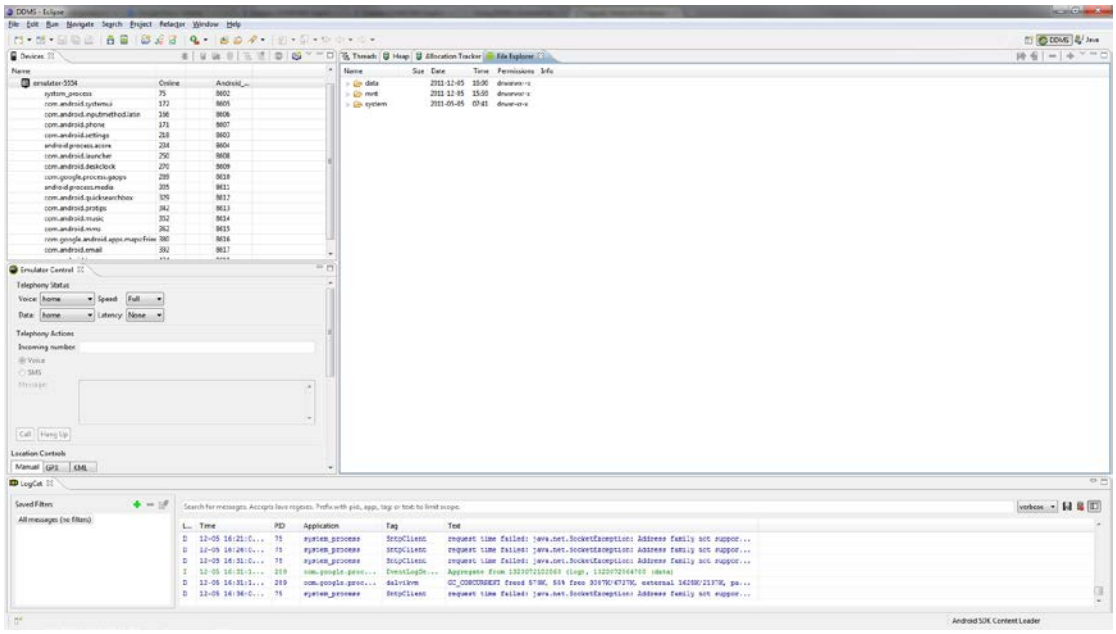
Source: Google Inc., http://developer.android.com

Please refer to **http://developer.android.com/guide/developing/tools/emulator.html** for more information.

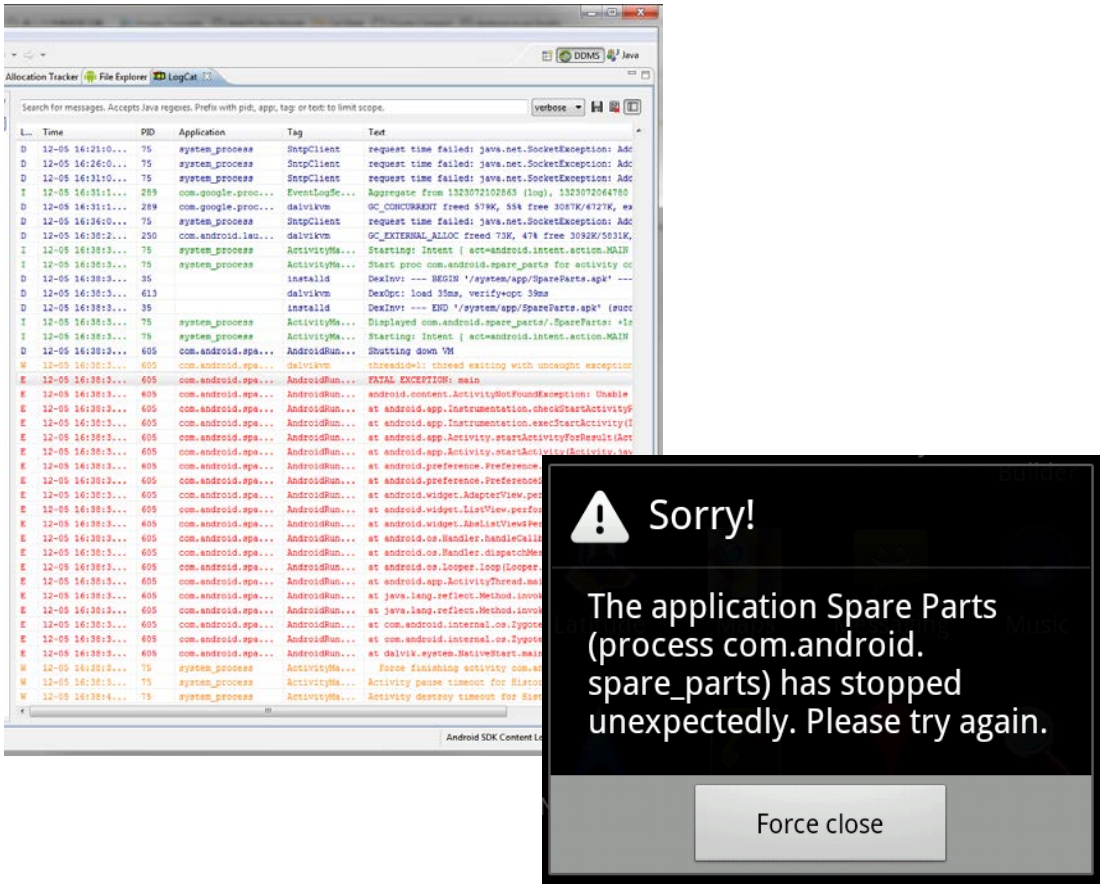
5. We can use DDMS (Dalvik Debug Monitor Server) to control and monitor emulators and actual devices. You will find DDMS in the **tools/** directory of the SDK and we can also start it in Eclipse. To do this, click **Window > Open Perspective > Other > DDMS**.



6. By using DDMS, you will able to simulate phone calls, network status, GPS coordinate for the emulator. You can also browse the file inside the emulator and using **Logcat**. **Logcat** is very important towards your development as it provides you a view for collecting and viewing the system debug output.

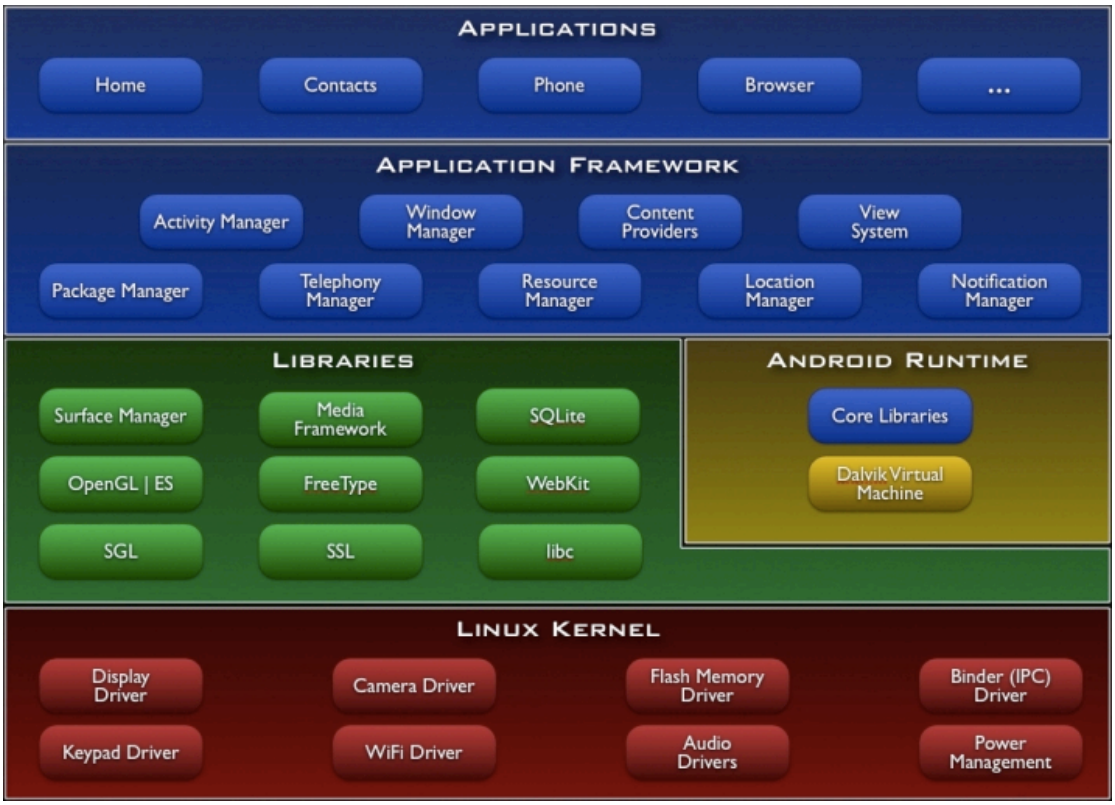


7. For example, if your application is force closed, then you are required take a look at the logcat to see the exceptions thrown by the Android system



4 Understand fundamental concepts

4.1. Android architecture



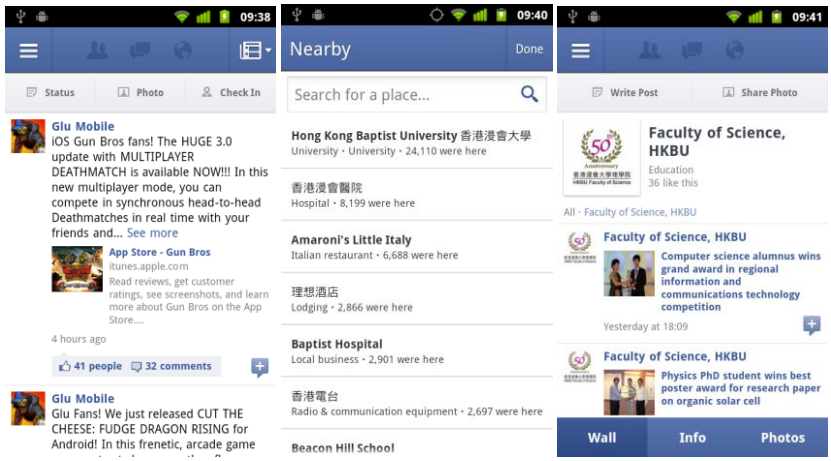
Source: Google Inc., 713px x 512px, http://developer.android.com

For details, please refer to

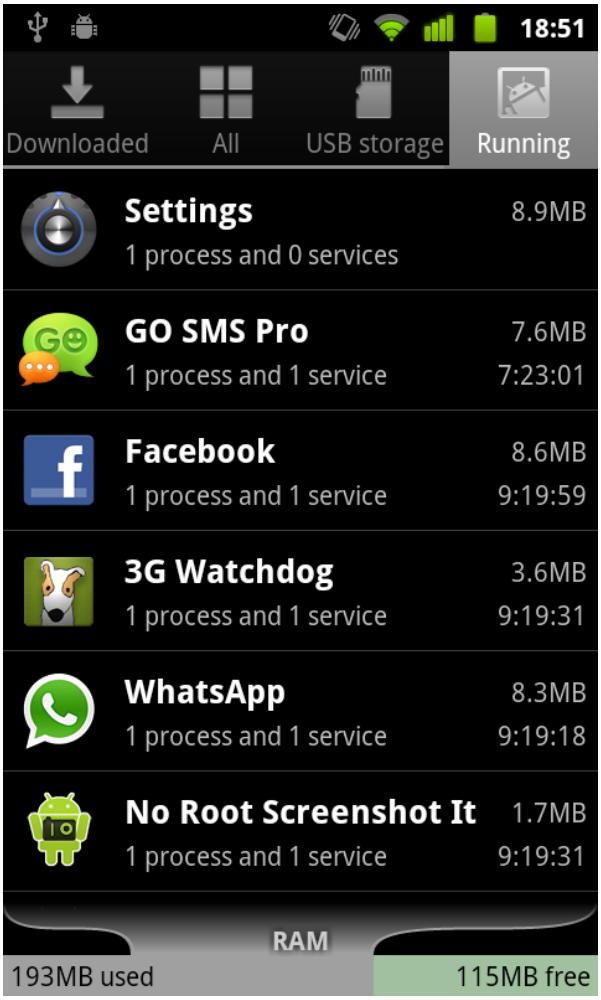
http://developer.android.com/guide/basics/what-is-android.html

4.2. Application components

1. Activities
   - Each activity is a single screen with user interface
   - Every screen is implemented as a subclass of **android.app.Activity**



2. Services
   - Run in background
   - No user interface
   - Designed for performing long-running tasks
   - Every service is implemented as a subclass of **android.app.Service**

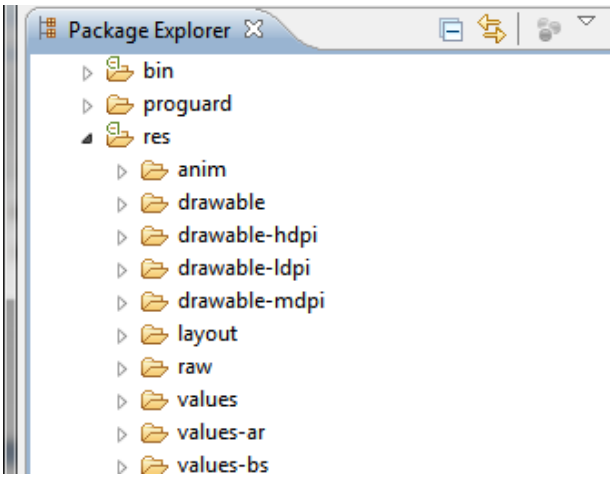


3. Content providers
   - A shared set of application data
   - E.g. Contacts, Calendar (Android 4.0)
4. Broadcast receivers
   - A component designed to response to global broadcast announcements
   - System broadcast announcements E.g. change of battery status/network status

4.3. Application resources

- Provide other resources to your apps. E.g. Image, Video, Audio, Text etc…
- Resource are put under **res** directory of your Android project
- Every resource is identified by a resource ID

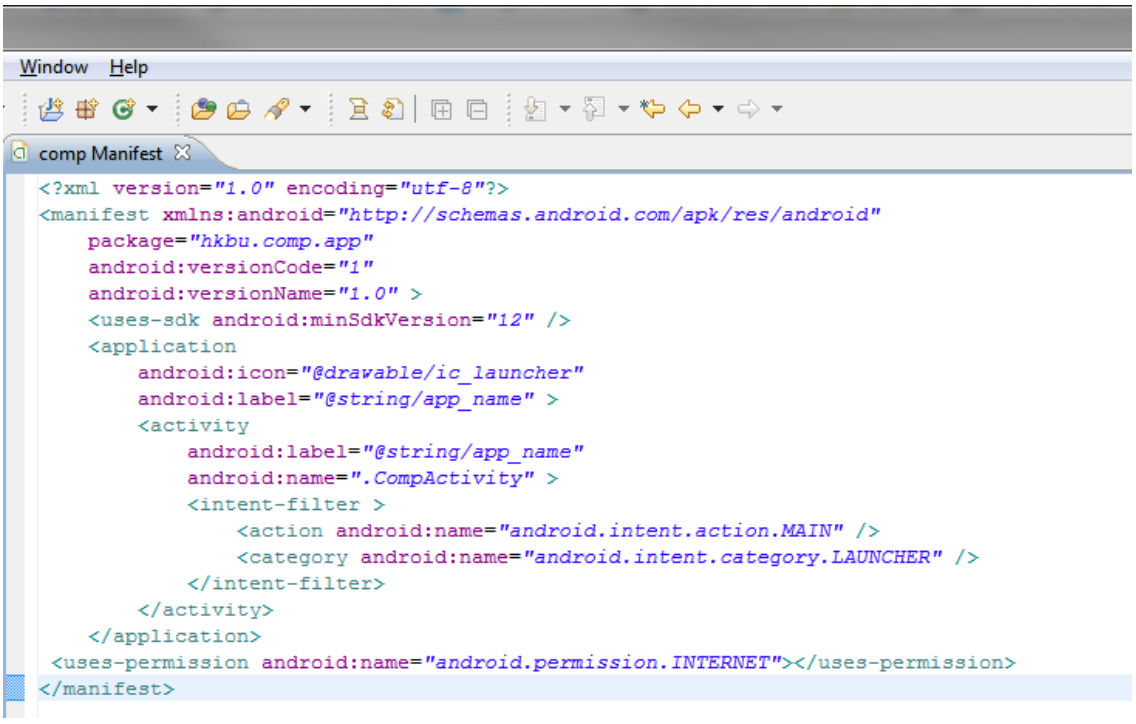
- Resources can have qualifier names for alternative E.g. layout-**land**, values-**ja**, drawable-**hdpi** etc



4.4. Define the app properties in Android

We use a file called **Manifest.xml** to describe information about your app to the system and Android Market as well. Inside this file, you can:

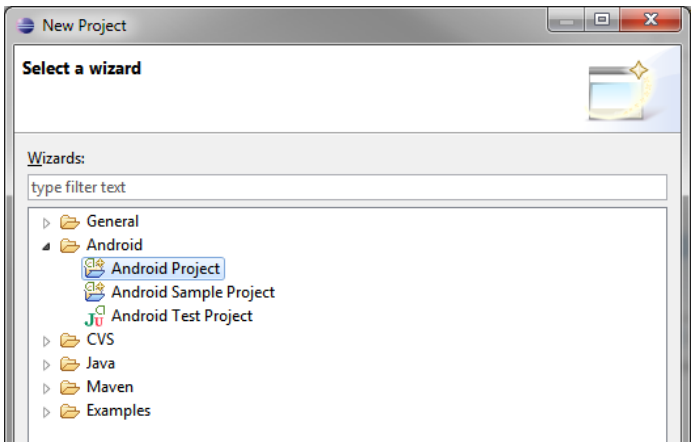
1. Define the package name
2. Define version code and version name
3. Define app icon
4. Define the components (Activities/Services/Receivers etc.)
5. Define the permission(s) should be granted



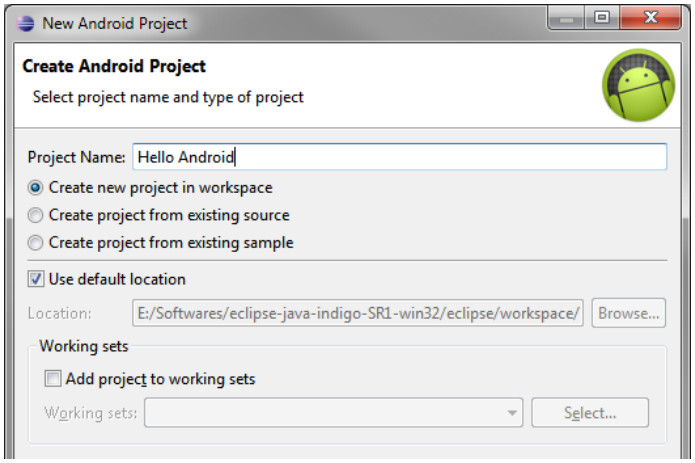
You may refer to http://developer.android.com/guide/topics/manifest/manifest-intro.html for details.

5 Create your first application

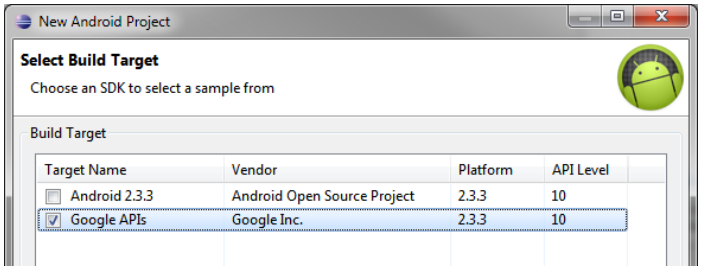
1. Start Eclipse, select **File > New > Project** and choose **Android > Android Project**.



2. Type **Hello Android** in **Project Name**.



3. Select **Google APIs (API level 10)**.

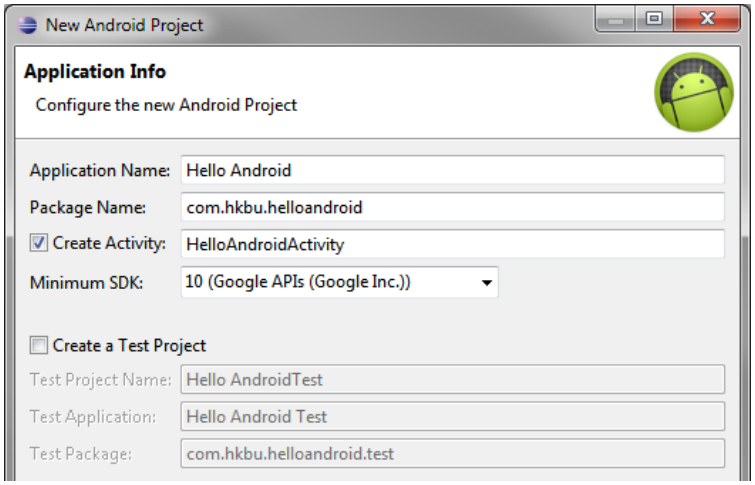


Note:

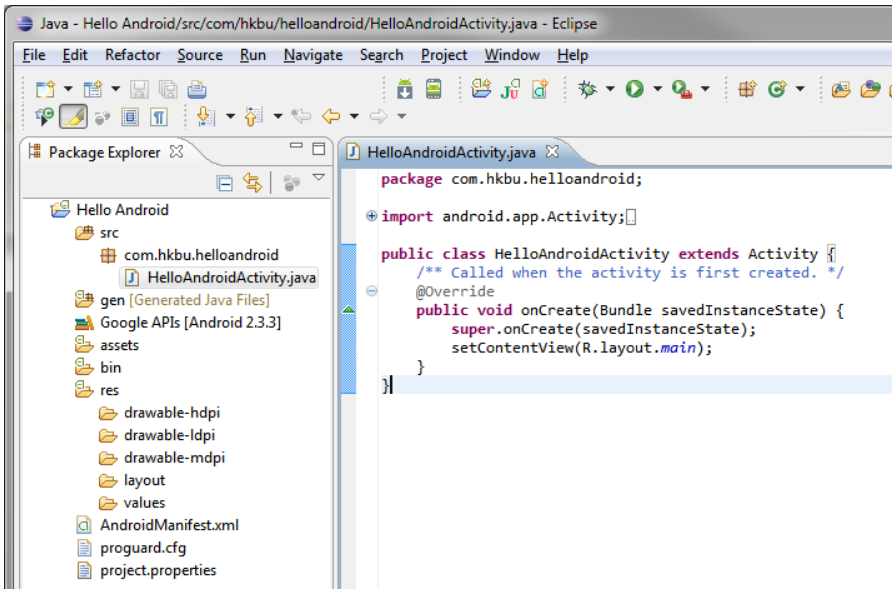
- You will have more **Build Target** once you downloaded more emulator images.
- **Google APIs** includes both standard Android and Google libraries such as Google Maps. Thus, if your application requires the use of Google Maps, you must use Google APIs as the build target.

4. Type **com.hkbu.helloandroid** in **Package Name,** check the box **Create Activity**

and choose **10 (Google APIs (Google Inc.))** as the **Minimum SDK**.

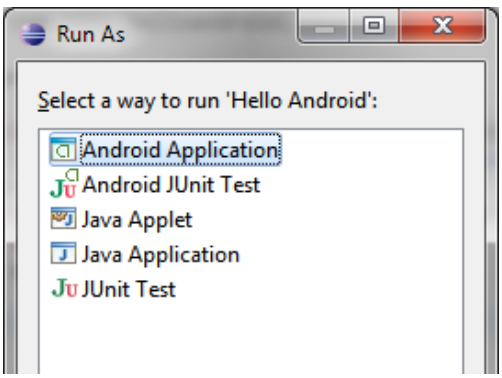


5. Since we have checked the option **"Create Activity"**, the ADT plugin will automatically generate a stub activity and related resources right after the project is created.

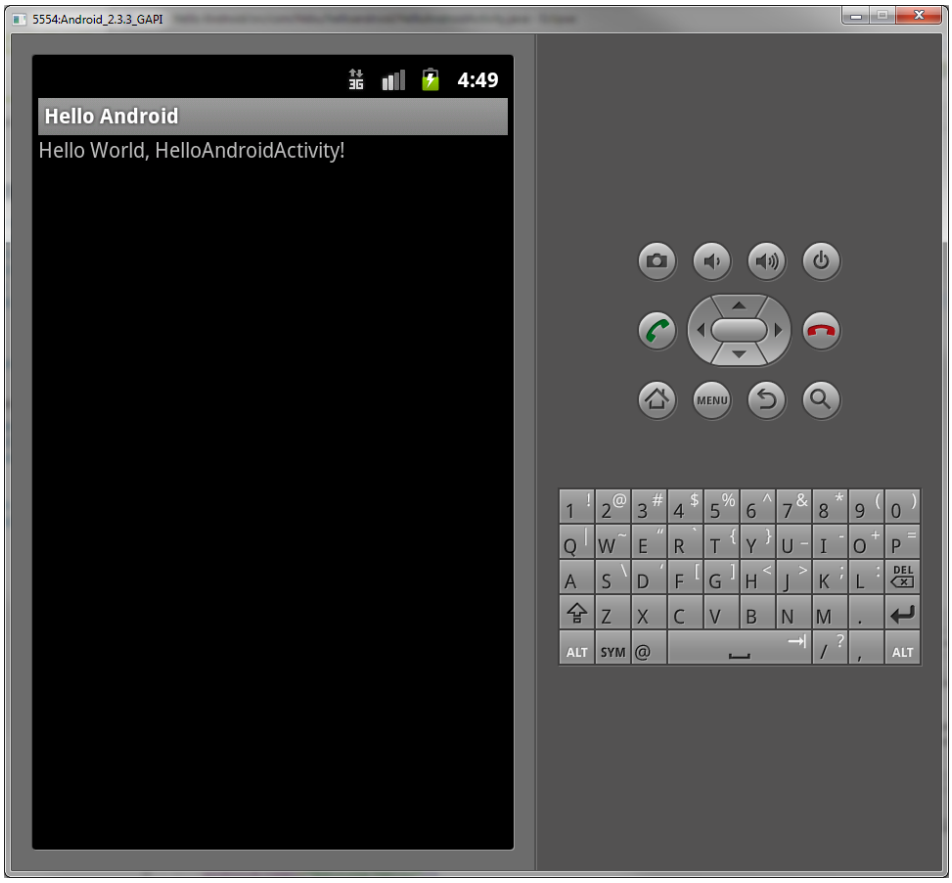


5.1 Run and debug apps in emulator

1. We can try to test the app in the emulator by choosing **"Run"** and select **"Android Application"**.

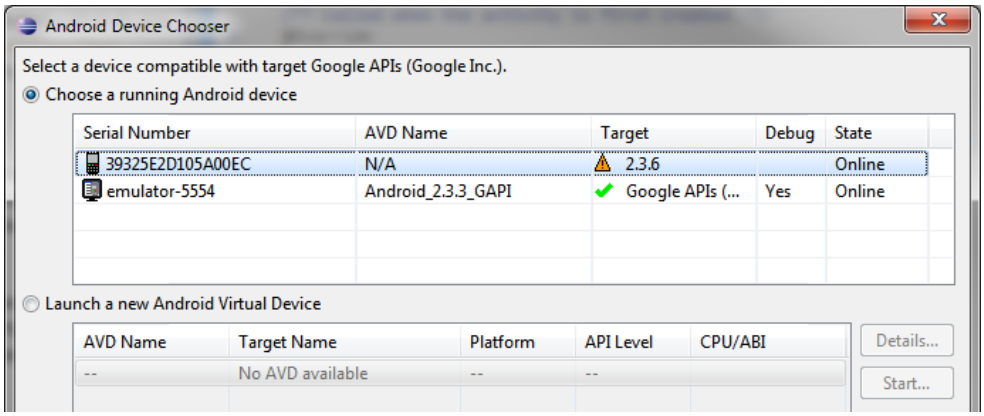


2. The ADT plugin will automatically launch the emulator, install and run the app once the emulator is successfully booted.



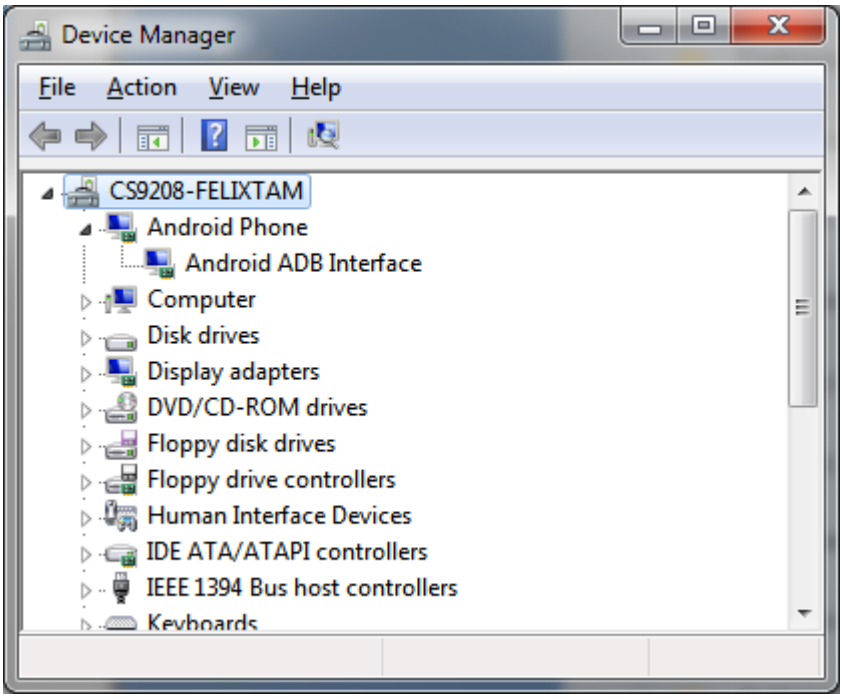
Note:

- If you have more than one emulator created or device connected and satisfied the requirements, you will have a dialog to choose which emulator should be used for testing.

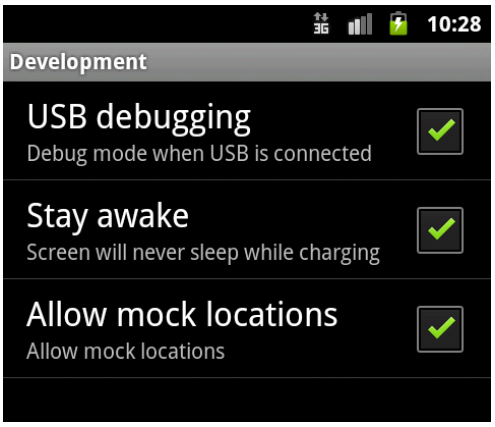


5.2 Run and debug apps in actual device

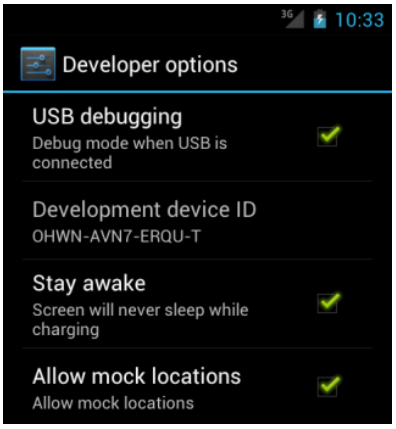
1. Make sure your device driver is properly installed otherwise the ADB will not be able to detect the present of your device.



2. On your device, go to **Settings > Applications > Development** and enable the **USB debugging** option. If you are using Android 4.0, the path will be **System settings > Developer options** instead.



Android 2.3 or below

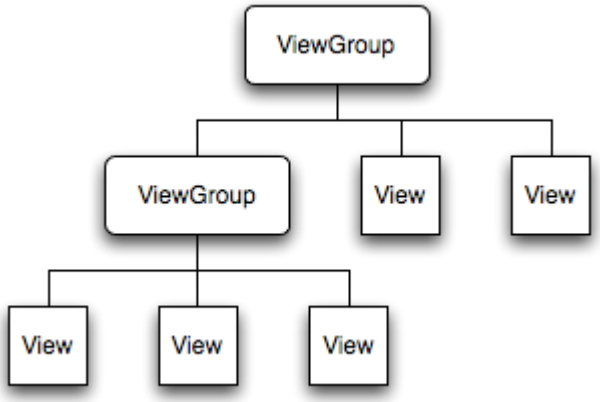


Android 4.0

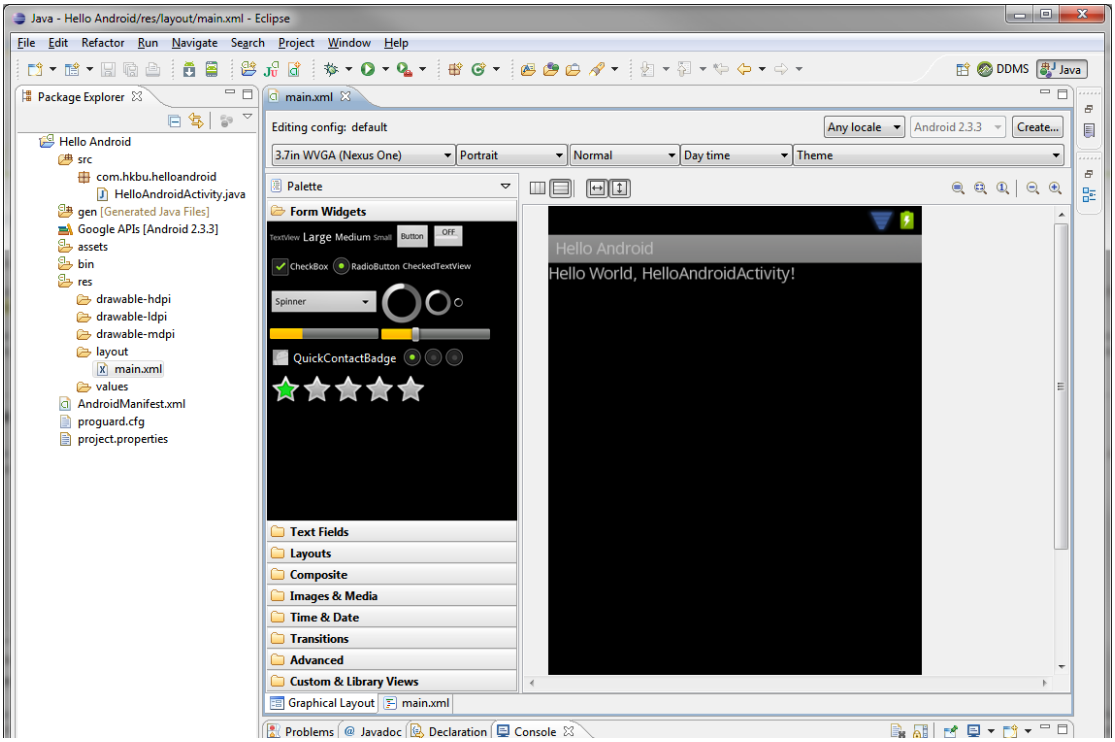
3. Run the application from Eclipse using the same steps shown in 5.1.
4. Select your actual device in **Android Device Chooser** window. (Please refer to the note in 5.1)

6. Create UI element using XML

All UI elements are built using **View** and **ViewGroup** objects in an application. The View can be extended to a **TextView**, **ImageView, WebView** etc, and the ViewGroup serves as layout architecture which describes how the View to be organized in an activity. Apart from building UI directly in the source code, we can also create the interface using XML format. By using the XML, we can simply our code and make debugging easier.



The XML layout file is saved in **res/layout**. By using the Hello Android example, you will find the initial UI is defined in main.xml. When you open the main.xml, Eclipse will show you a graphical UI editor for the interface preview.



However, the graphical UI editor is not sophisticated enough. The final appearance of the interface layout may differ from the preview in the editor. Therefore, the best practice is to create the UI elements by editing the XML code directly. To do this, you can click on the **XXX.xml** tab.

6.1. Understand the XML layout structure

Using the main.xml as an example:

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

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />

</LinearLayout>
```

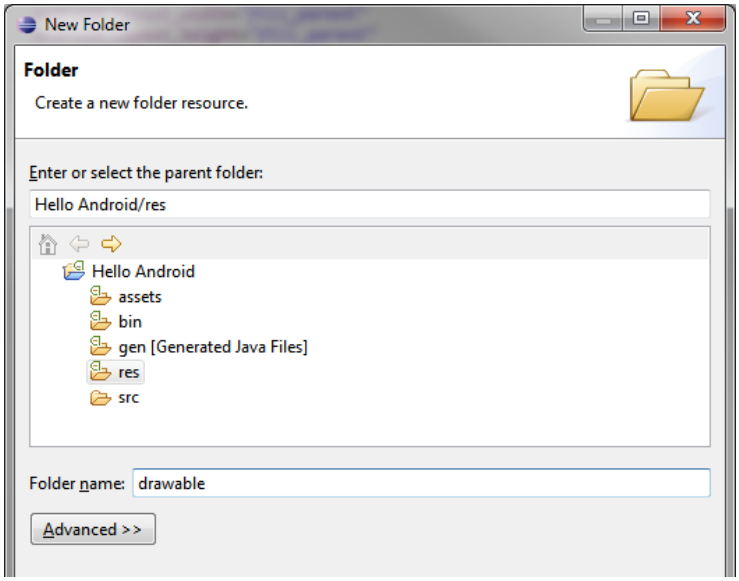
The LinearLayout is a ViewGroup which holds its children in a single column or a single row. The above LinearLayout has only one child – TextView. Android uses TextView to display text to users. For a complete list of ViewGroup and View, please refer to **http://developer.android.com/reference/android/view/ViewGroup.html** and **http://developer.android.com/reference/android/view/View.html** respectively.

Each kind of ViewGroup or View has its own attributes. The attributes usually start with **android:xxx** where the **xxx** is the property name of an attribute. Different ViewGroup and View share some common attributes but some of them are unique. For a complete set of attributes, please refer to the **Reference** section in developer.android.com.

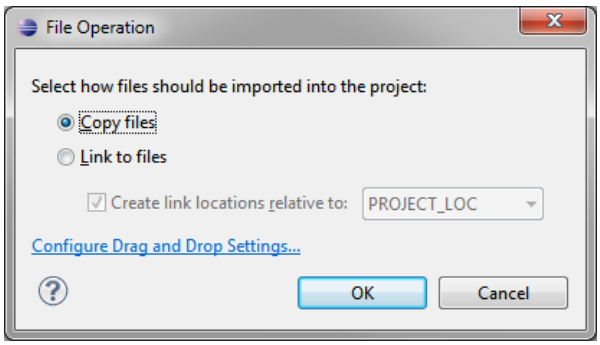
To have a quick outlook on different layouts and UI elements, please go to **http://developer.android.com/resources/tutorials/views/index.html** for the samples. Now, we will see how to add an image just below the text in our main.xml.

6.2. Add text and image to the UI

1. Prepare an image. You may download this **http://developer.android.com/assets/images/home/ics-android.png** as an example.
2. Put the image under **res/drawable**. If this folder is not found, you can create it manually using the **Package Explorer** in Eclipse.



3. Drag the image directly to the **drawable** folder through the **Package Explorer**. In the **File Operation**, choose **Copy files**. You also need to rename the file if the characters are not in (**a-z**), (**0-9**), (**_**) and (**.**) .



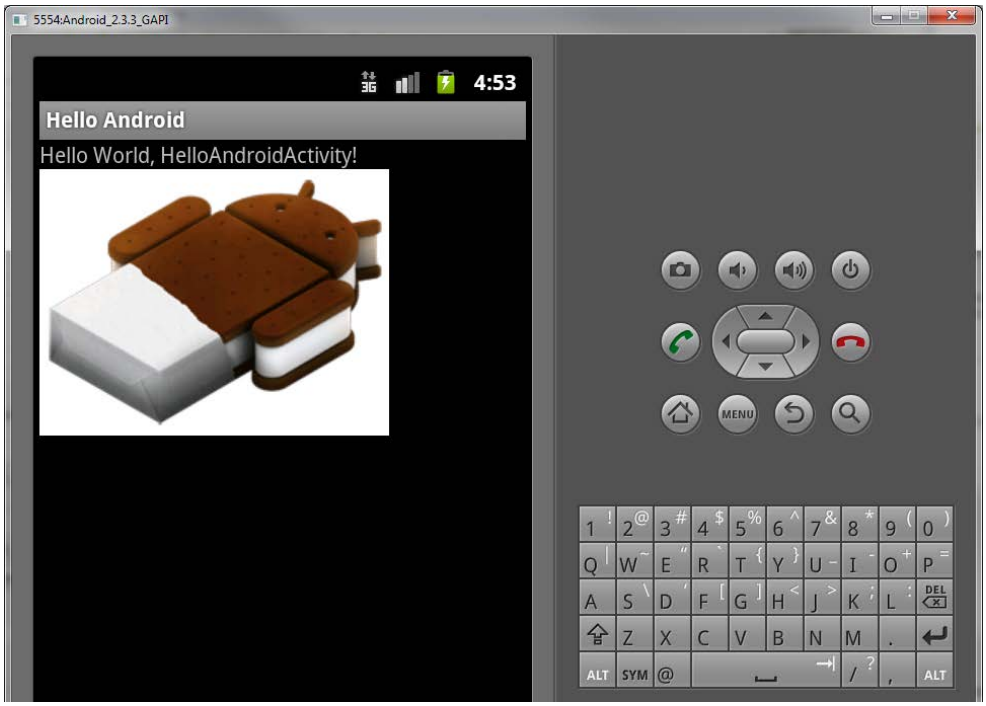
4. In the main.xml, add the following codes below the TextView.

```
<ImageView
        android:src="@drawable/ics_android"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
```

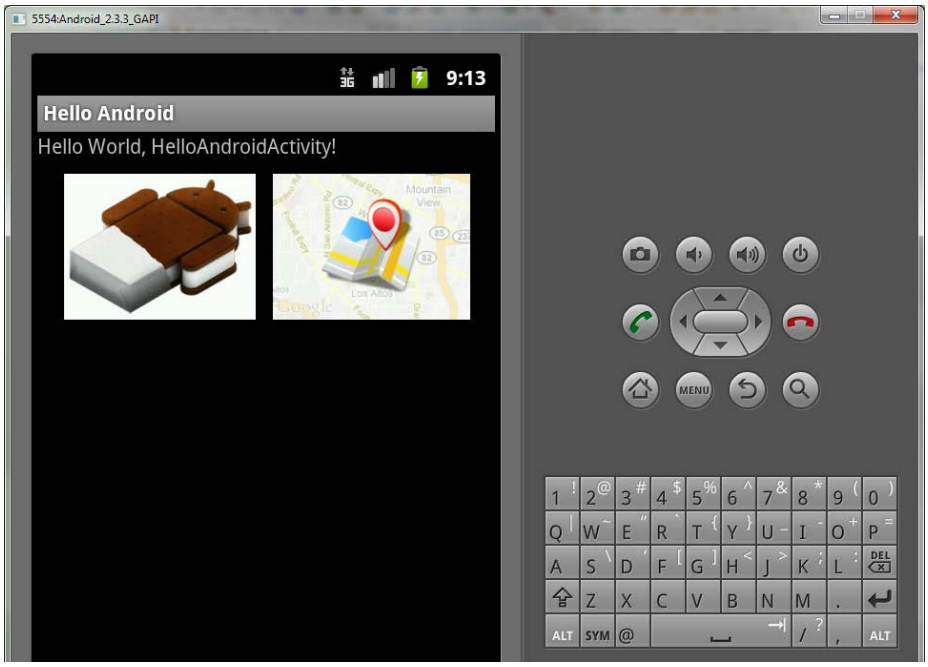
- The **ImageView** is used to show an image in the interface.
- The **android:src** is used to specify the source of the image file. Here the path and the file name are **@drawable/ics_android**. (File extension is omitted)
- The **android:layout_width** and **android:layout_height="wrap_content"** is used to specify the ImageView size which is the same as the image itself.

For the list of complete attributes and values of ImageView, please refer to http://developer.android.com/reference/android/widget/ImageView.html.

5. You can press the Graphical Layout tab to preview the result.
6. Run your app in the emulator and see the actual result. The final appearance should look likes this:

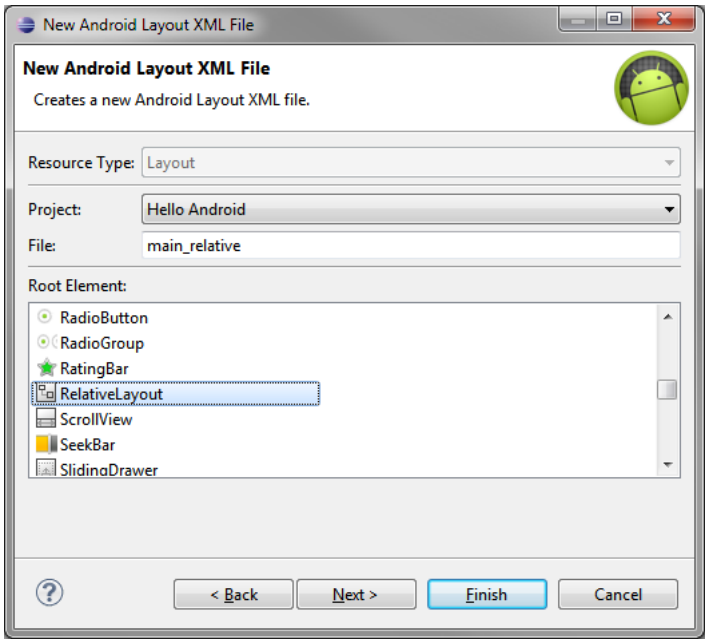


7. The above example uses LinearLayout which is the most basic layout in Android. If we need to achieve the result below, you have to complete the following changes:
   - Create a new layout XML file and use the RelativeLayout
   - Resize the ImageView
   - Add one more ImageView and refer to the new image
   - Add more attributes to the TextView and ImageView



8. Now we need to create a new layout file called **main_relative** inside the **res/layout**. Right Click on the layout folder in Package Explorer and choose **New**

> Other > Android > Android XML Layout File. Type **main_relative** for **File** and select **RelativeLayout** in **Root Element.** Hit **Finish** when you are done.



9. Add a **TextView** to the RelativeLayout by inserting following code:

```
<TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"/>
```

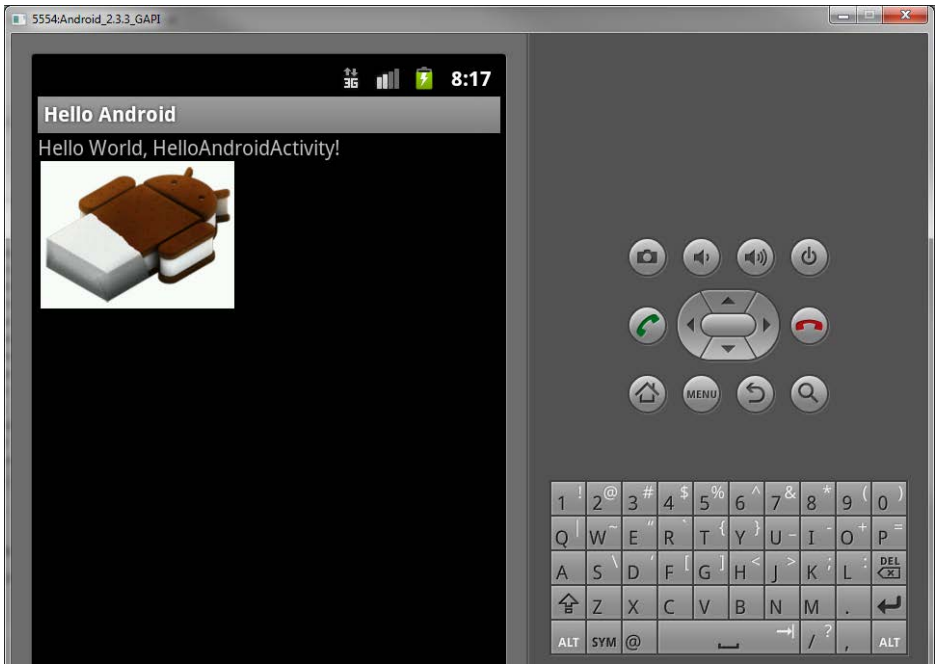
10. Since elements in RelativeLayout are depending to each others. We have to add ID to each element as an identifier. To add an ID for the TextView, we can use **android:id="@+id/MyText"**. (Now the ID of this TextView is MyText)
11. Next, we can add two ImageViews for our two images. The second image can be downloaded via this link (Also put this image in res/drawable, replace the **(-)** with **(_)** for the file name):
    http://developer.android.com/assets/images/home/maps-large.png
12. Insert the following codes for the first ImageView:

```
<ImageView
        android:id="@+id/ImageView1"
        android:layout_width="138dp"
        android:layout_height="102dp"
        android:src="@drawable/ics_android"
        android:layout_below="@id/MyText"/>
```

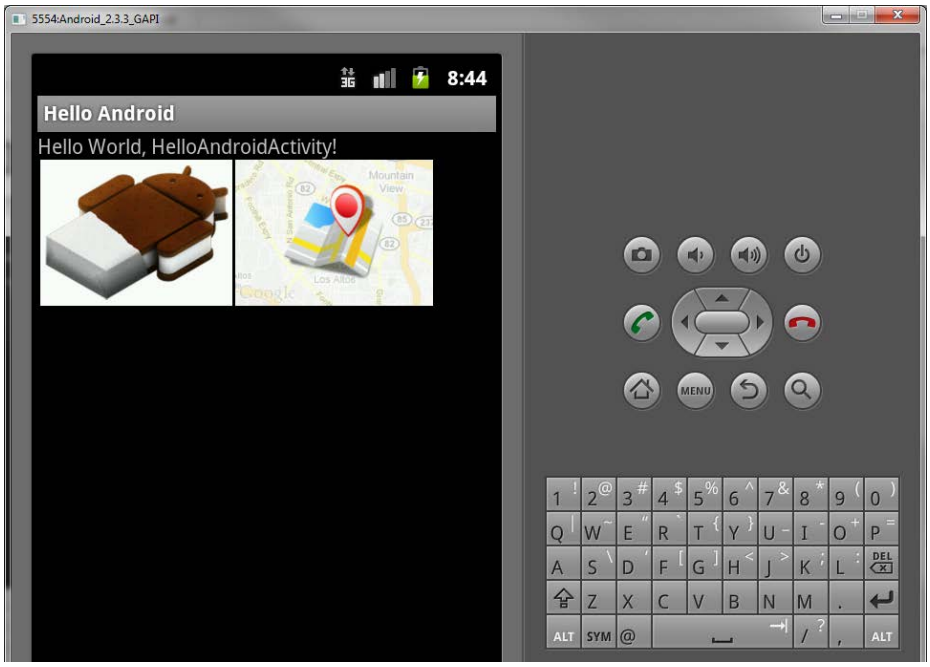
- We specify the width and height in **android:layout_weight** and **android:layout_height**. Please note that we should use **dp** for the unit of

dimension. **dp** stands for **D**ensity-independent **P**ixel which is an abstract unit based on the physical density of the screen. This unit uses a 160 dpi screen as a reference. By using **dp,** we can ensure the same physical image dimension on every device with different screen sizes and resolutions. For details, please refer to: http://developer.android.com/guide/practices/screens_support.html

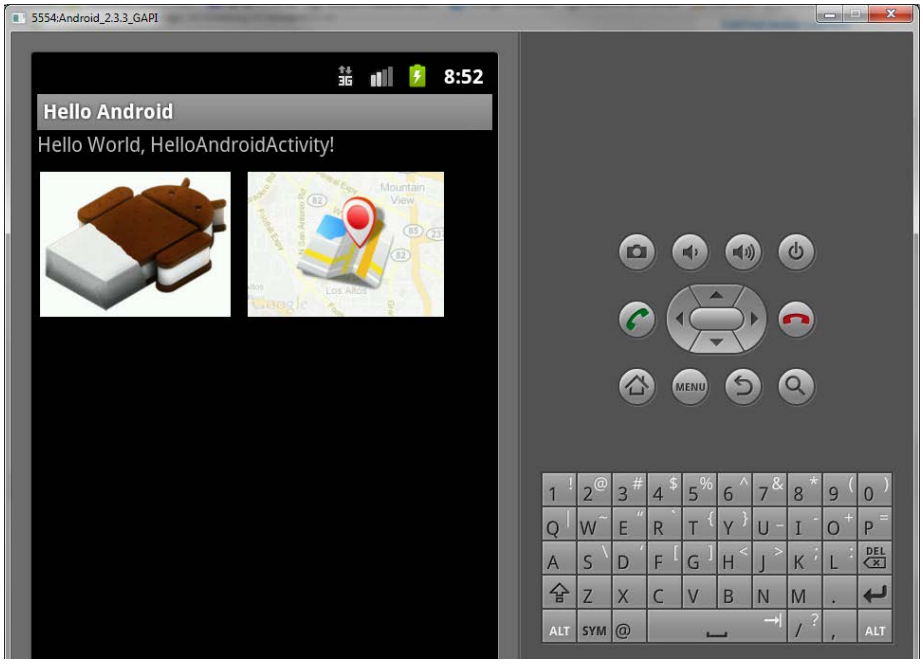
- The **android:layout_below:"@id/MyText"** defines the first ImageView should be placed below MyText (TextView).
- Go to **/src/com.hkbu.helloandroid/HelloAndroidActivity.java**, change **(R.layout.main)** to **(R.layout.main_relative)** for pointing to our new layout file.
- Till now, the interface will look likes this:



13. To add the second image, we can copy the codes of the first ImageView and paste them as the last element of the RelativeLayout. However, you have to modify the following attributes:
    - Change the value of **android:id** to **@+id/ImageView2**
    - Change the value of **android:src** to **@drawable/maps_large**
    - Add **android:layout_toRightOf="@id/ImageView1"**. This means that the second ImageView will be placed on the right-hand side of the first ImageView.
    - Till now, the interface will look likes this:



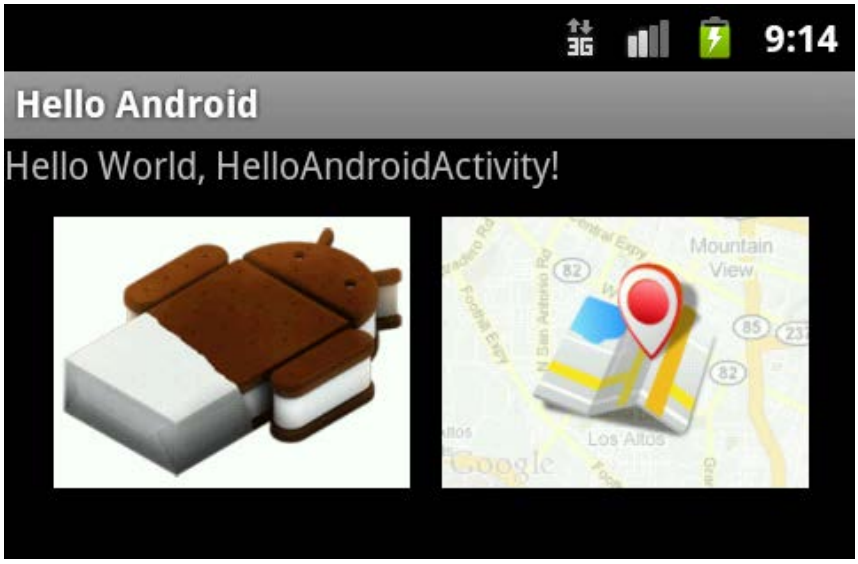
14. We can add margins for the TextView and ImageView to make them not to close to each others.
    - Add **android:layout_marginBottom="10dp"** to the TextView
    - Add **android:layout_marginRight="10dp"** to the first ImageView
    - Till now, interface will look likes this:



15. In the next step, we will align the two images to the center of the screen horizontally. Since RelativeLayout does not support centering more than one child element in the same height, we have to use LinearLayout to encapsulate the two ImageViews. Add the LinearLayout and move the ImageViews into it by using the following code:

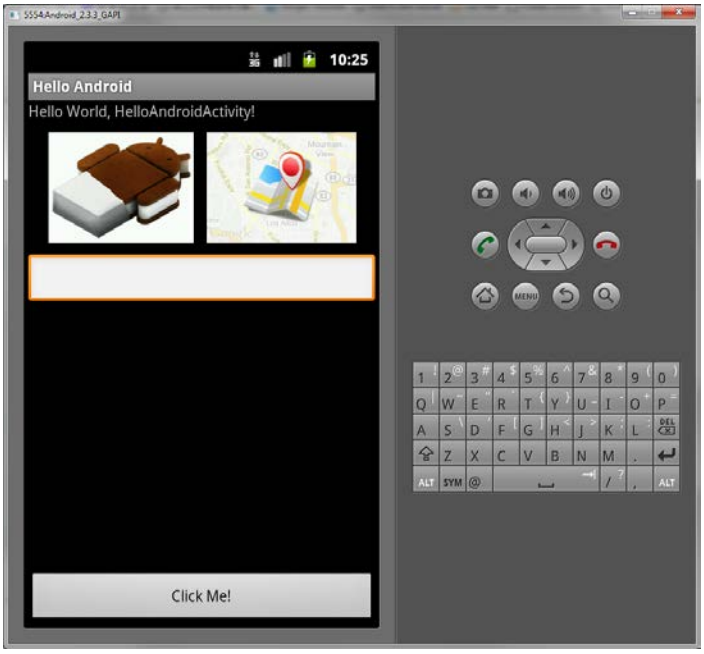
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout … …>
    <LinearLayout android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/MyText"
        android:orientation="horizontal"
        android:gravity="center">
        <ImageView
            android:id="@+id/ImageView1"
            … …/>
        <ImageView
            android:id="@+id/ImageView2"
            … …/>
     </LinearLayout>
</RelativeLayout>
```

- We moved the attribute **android:layout_below="@id/MyText"** from the first ImageView to the LinearLayout to make sure our LinearLayout will be placed under the TextView. Also, **remove** the same attribute for the second ImageView.
- You also need to remove **android:layout_toRightOf="@id/ImageView1"** from the second ImageView since it is only for RelativeLayout before.
- The default direction for aligning elements of LinearLayout is vertical, we have to use **android:orientation="horizontal"** for horizontal alignment. **android:gravity="center"** is used to center ImageViews in the screen.
- The final result:



7. Handle user input and more interface design

In this section we will learn how to handle simple user input and manipulate the UI elements in Java code. We will add a **TextField** and a **Button** under our images.



7.1. Accept user input by adopting EditText and Button

1. Add an ID to the LinearLayout (E.g. android:id="@+id/MyLinearLayout")
2. Add an EditText (For inputting text) **below** the LinearLayout

```
<EditText android:id="@+id/MyTextField"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:layout_below="@+id/MyLinearLayout"/>
```

3. Add a Button at the bottom of the screen.

```
<Button
    android:id="@+id/MyButton"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:text="Click Me!" />
```

4. When users click on this button, the app should perform following actions:
   - The text in the TextView will be change to those users typed in the EditText
   - The images in the two ImageViews will be swapped.

5. From now on, we will write the Java code to listen the action when users click on the button. Add the following code **above** the **"onCreate()"** method in **HelloAndroidActivity.java**.

```
Button myBtn;
TextView myTextView;
EditText myEditText;
ImageView myImgView1;
ImageView myImgView2;
```

Here we create four null references for our Button, TextView and two ImageViews respectively. Import any necessary package suggested by Eclipse.

6. In the **onCreate()** method, add the following code below the **setContentView()** method:

```
myBtn = (Button)findViewById(R.id.MyButton);
myTextView = (TextView)findViewById(R.id.MyText);
myEditText = (EditText)findViewById(R.id.MyTextField);
myImgView1 = (ImageView)findViewById(R.id.ImageView1);
myImgView2 = (ImageView)findViewById(R.id.ImageView2);
```

- We use the **findViewById()** method to get the reference to the UI created using XML format.
- The **R** class will be automatically generated by AAPT (Android Asset Packaging Tool) and updated once you added or updated a resource. We will always use this class to handle all the elements located in the **res/** directory. For details, please refer to http://developer.android.com/guide/topics/resources/accessing-resources.html

7. Add the following code below the onCreate() method to create an anonymous implementation of OnClickListener:

```
private OnClickListener myBtnListener = new OnClickListener() {
        @Override
        public void onClick(View arg0) {
        }
    };
```

We may use Eclipse's quick-fix feature to fix most of errors. Note that for onClickListener you should import android.**view.View**.OnClickListener

8. Inside the onCreate() method, add the following code to register the myBtnListener to myBtn:

```
myBtn.setOnClickListener(myBtnListener);
```

9. Add the following code to the onClick function:

```
myTextView.setText(myEditText.getText());
```

- We use the getText() and setText() method to grab the text from the EditText and then set it to the TextView.

10. Then add the following code to swap the images for our ImageViews:

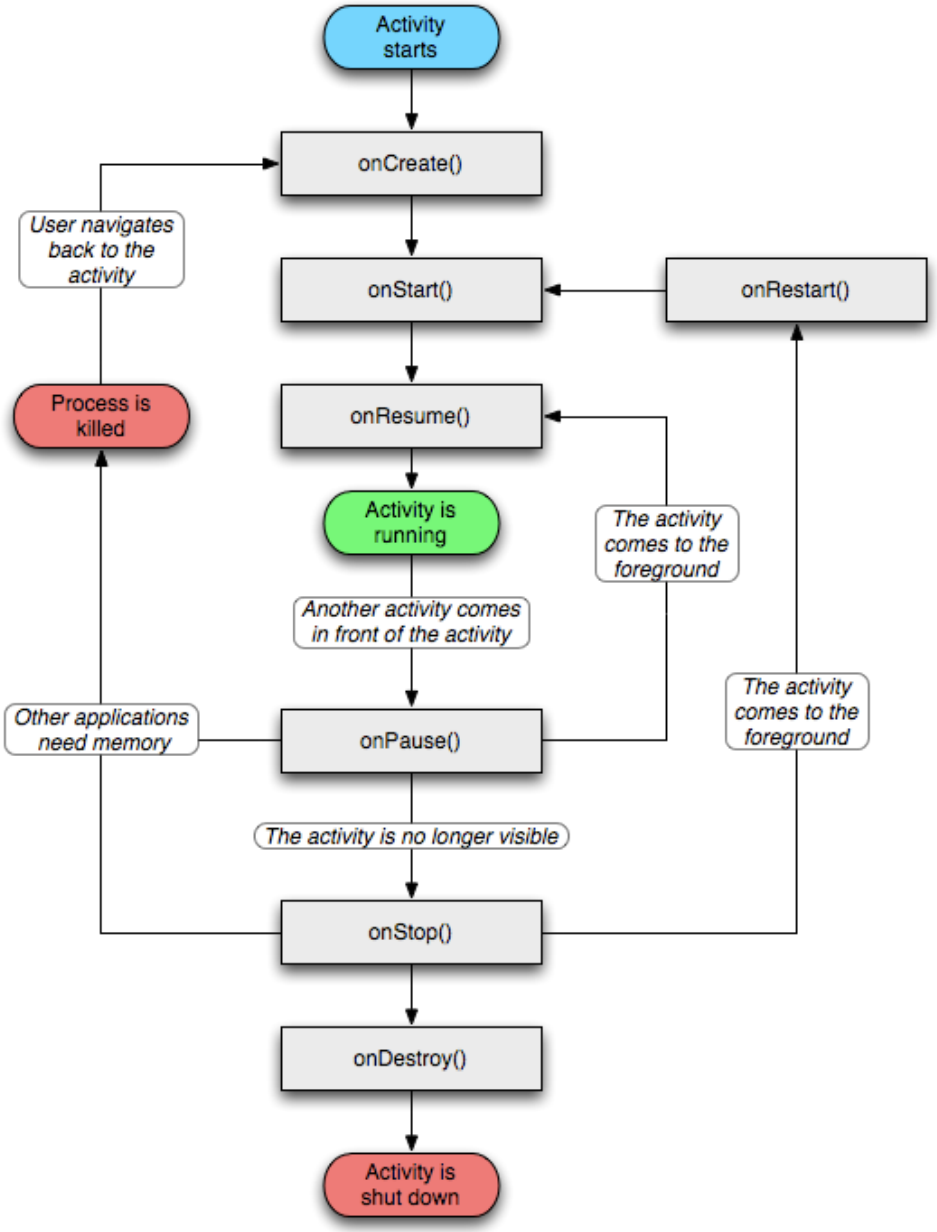
```
myImgView1.setImageResource(R.drawable.maps_large);
myImgView2.setImageResource(R.drawable.ics_android);
```

- We use the **R** class again to refer to our images. The **R** class has different subclasses for each type of resources. For image, it is **R.drawable**.

11. That's it! Try to run it on the emulator or on your device. For a list of R subclasses, please refer to http://developer.android.com/guide/topics/resources/available-resources.html

7.2. Activity lifecycle

In our previous example, we create references and register event listener inside the onCreate() method. It is because the onCreate() method will be called once the activity is started.



Source: Google Inc., 545px x 711px, http://developer.android.com

Things you have to know:

Entire lifetime: Happens between onCreate() and onDestory

Visible lifetime: Happens between onStart() and onStop()

Foreground lifetime: Happens between onResume() and onPause()

For details, please refer to

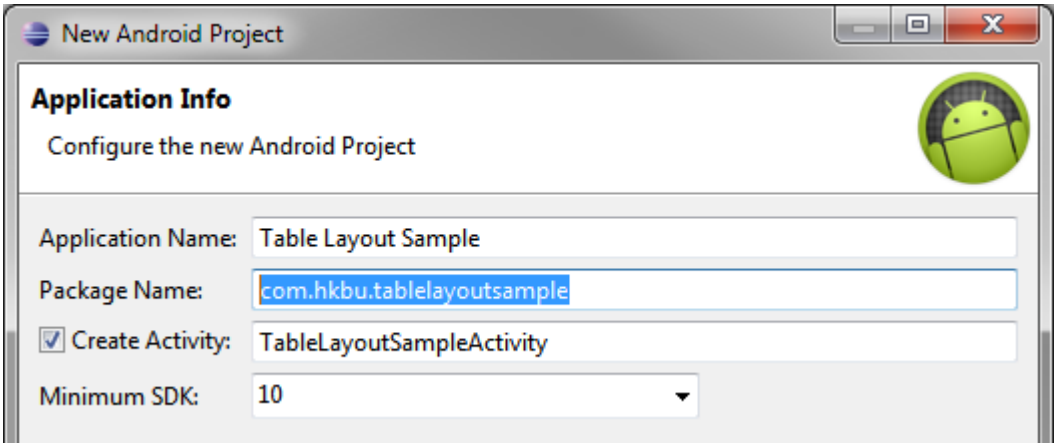
http://developer.android.com/reference/android/app/Activity.html

7.3. Use TableLayout for structural elements and data

Sometimes if you like to position your UI elements or displaying a set of data in rows and columns, using TableLayout will be much more convenient than using LinearLayout or RelativeLayout. Suppose we have the following set of data need to be shown:

Chinese (Traditional):	Chinese - Hong Kong SAR (zh_HK)
Arabic:	Israel (ar_IL)
Bulgarian:	Bulgaria (bg_BG)
Catalan:	Spain (ca_ES)
Czech:	Czech Republic (cs_CZ)
Danish:	Denmark(da_DK)

1. Start a new project in Eclipse using the steps written in section 5. For the **Project Name** and **Package Name**, we will use **Table Layout Sample** and **com.hkbu.tablelayoutsample** respectively.



2. Edit the main.xml, change the LinearLayout to TableLayout. Remove the **android:orientation="vertical"** attribute and the default TextView.

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
</TableLayout>
```

3. To add rows to the TableLayout, we can use **<TableRow></TableRow>** to enclose our UI elements. For example, we can use the following code to construct the first row of text.

```
<TableRow>
    <TextView
        android:layout_width="fill_parent"
```

```
        android:layout_height="wrap_content"
        android:text="Chinese (Traditional):" />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Chinese - Hong Kong SAR (zh_HK)" />
</TableRow>
```

*The number of elements in **<TableRow></TableRow>** equals to the number of columns in a row. We do not need to explicitly specify the column.

4. Please use the same way for inserting the rest of data.
5. Till now, the interface will look likes this:

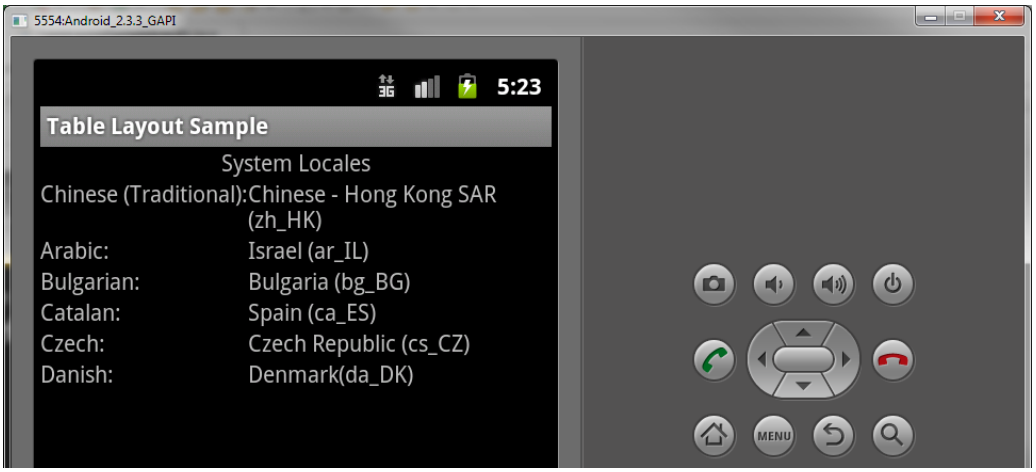
6. As you can see, if the text is too long in a cell, it will not be wrapped automatically. Therefore, we have to add an additional attribute **android:shrinkColumns** to **<TableLayout>**. This attribute will shrink a column based on the index provided. You can shrink more than one column by separated index using comma. Please note that the column is a zero-based index. In our example, add the following attribute to **<TableLayout>**

```
android:shrinkColumns="1"
```

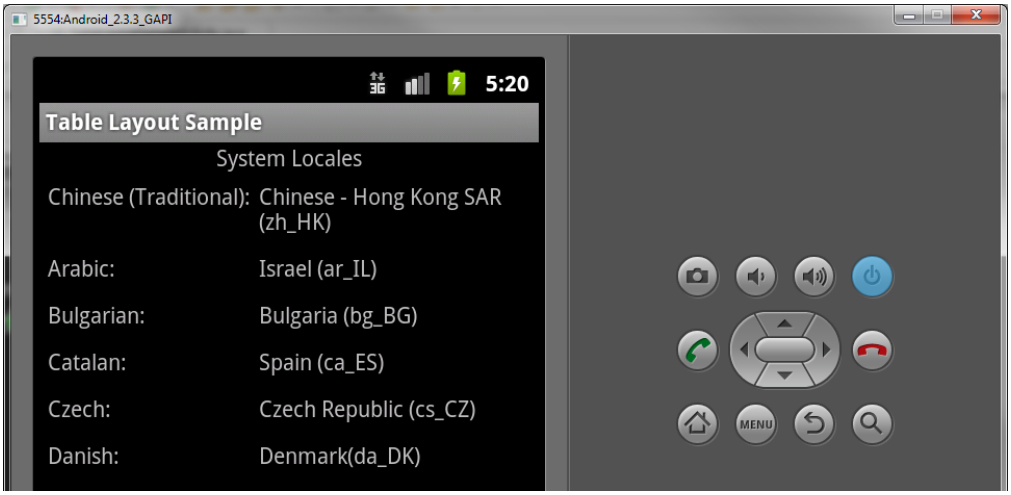
After adding this attribute, the result will be like this:

7. If we need a cell to span over columns, all you need is to add an attribute called **android:layout_span** to the elements in <TableRow>. For example, if we need to add a TextView as a header to the table, you can add the following code to be the first element in <TableLayout>.

```
<TableRow>
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="System Locales"
        android:layout_span="2"
        android:gravity="center" />
</TableRow>
```

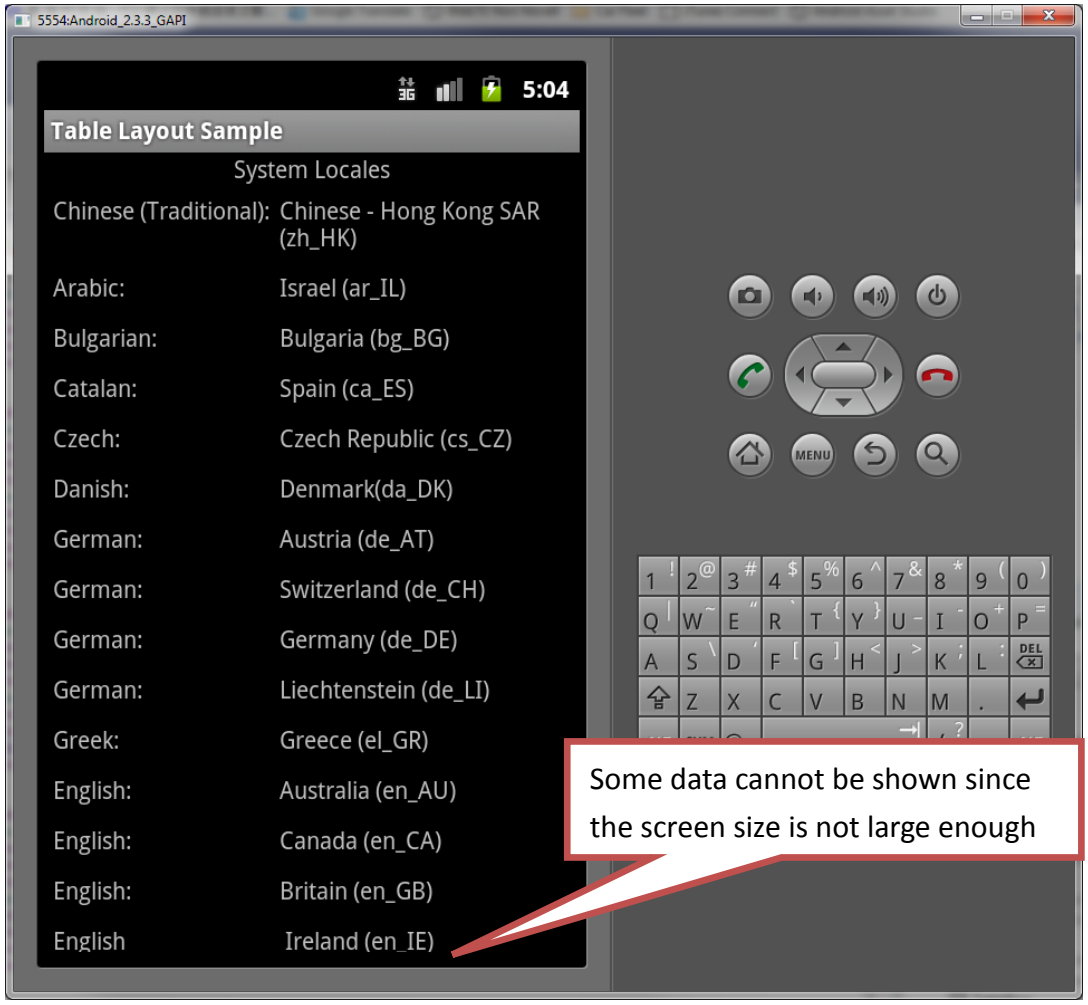


8. To add space between each row, we can add some padding to the **<TableRow>** For example: **android:padding="5dp"**
9. To add space between each column, we can add some margin to the elements (TextViews in this case). For example: **android:layout_marginLeft="5dp"**
10. Till now, the interface will look like this:



7.4. Use ScrollView for large content

When using the TableLayout for showing a large volume of data, the TableLayout may not be shown completely, as it is larger than the screen size.



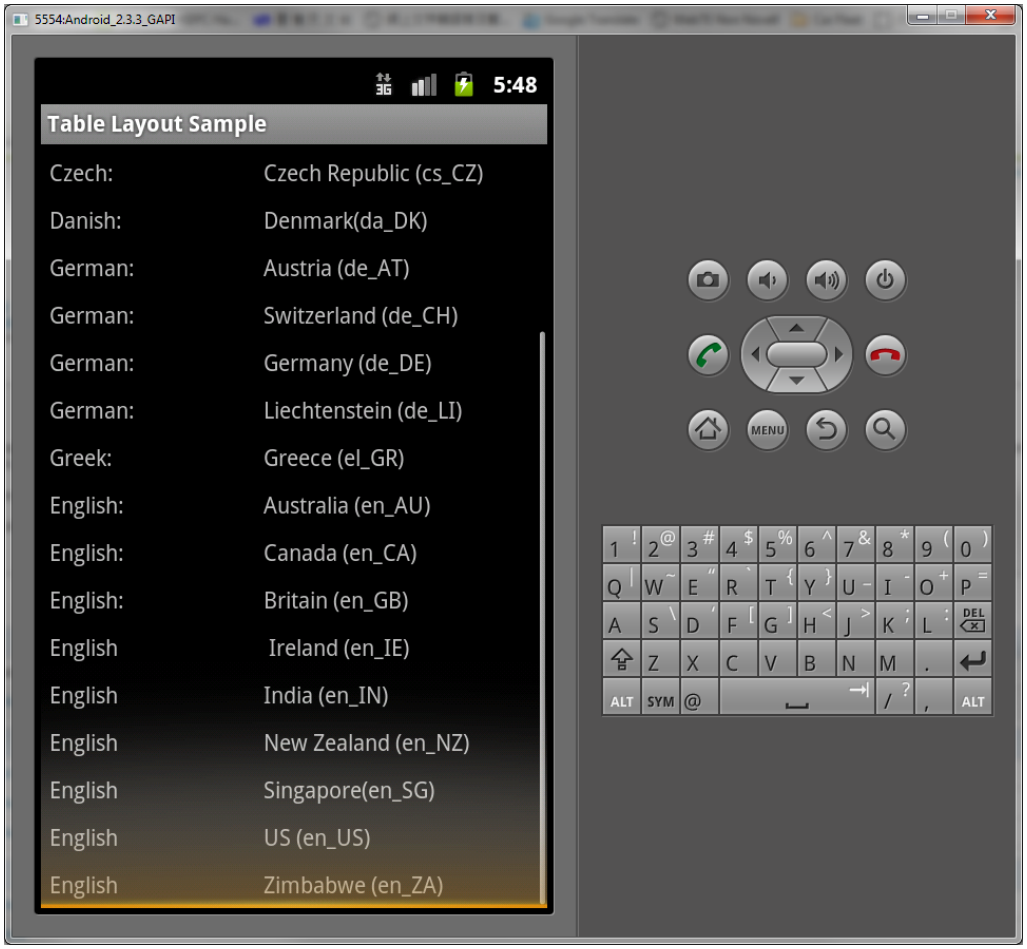
1. To solve this problem, we can use **ScrollView** is to enable scrolling for the TableLayout. The ScrollView extends the FrameLayout and it will enable the scroll functionality of its child. Please note that the ScrollView can only contain one child. If you want to scroll more than one UI elements at the same time, you have to encapsulate UI elements first.
2. Add ScrollView to enclose your <TableLayout>:

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
    android:layout_height="fill_parent">
<TableLayout>… …</TableLayout>
</ScrollView>
```

*Note: You have to insert **xmlns:android="http://schemas.android.com/apk/res/android"** for the outermost

element.

3. Now you can view more data by scrolling the table up and down.



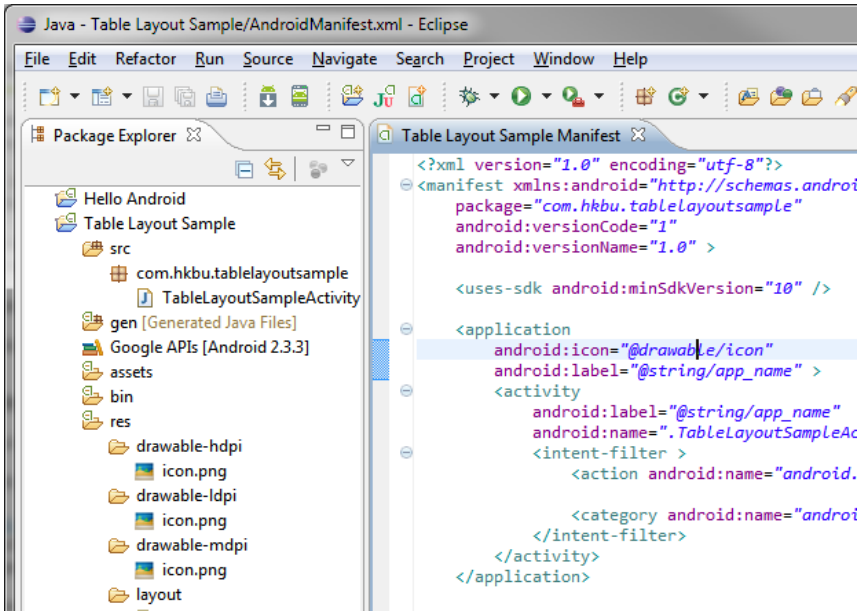
8. Prepare to publish your app

Before you publish the app to Android Market or distribute it to someone else, you need to design an icon for your app instead of using the default one. Also, you may need to change the app name according to your preference later on.

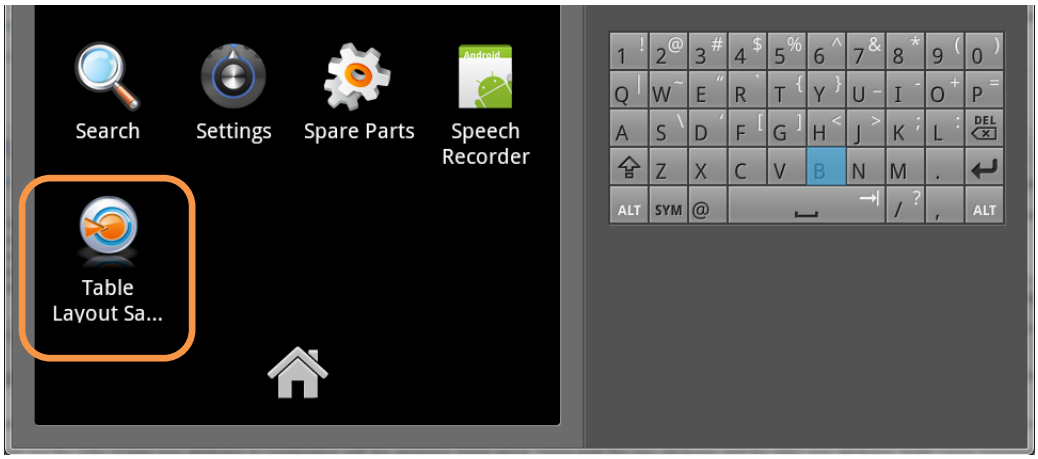
8.1. Prepare the app icon

For the app icon, we have to prepare at least three different sizes for different screen resolutions. Namely: ldpi, mdpi, hdpi. (There are also xhdpi and tvdpi)

1. Please prepare an app icon and resize it to 36x36px for ldpi, 48x48 for mdpi and 72x72 for hdpi. For the app icon design guideline. Please refer to **http://developer.android.com/guide/practices/ui_guidelines/icon_design_launcher.html**. You can also download some free icons with different sizes at www.iconarchive.com
2. Put the icon with the same file name (E.g. icon.png) into drawable-hdpi (72x72), drawable-mdpi (48x48) and drawable-ldpi (36x36) respectively.
3. Open the AndroidManifest.xml, change **android:icon="@drawable/ic_launcher"** to **android:icon="@drawable/icon".** You can also delete the default launcher icon.



4. The new icon will be shown in the launcher after you deploy your app into your emulator or device.



8.2. Change the app name and apply localization

When you look at the AndroidManifest.xml file, you can see an attribute **android:label="@string/app_name"** is already added to the <application> tag. The **@string/app_name** refers to a value of an elements call **app_name** located in **value/strings.xml**. By storing the string in the values/strings.xml, we can easily localize our apps by using different qualifier name (language code this time).
Notes: **zh-rTW** for Traditional Chinese, **zh-rCN** for simplified Chinese

1. Go to **values/strings.xml**, change the app name to **HKBU**.
2. Create a folder called **values-zh-rTW** in **res** directory.
3. Create a **strings.xml** file inside the **values-zh-rTW** and insert the following code:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">香港浸會大學</string>
</resources>
```

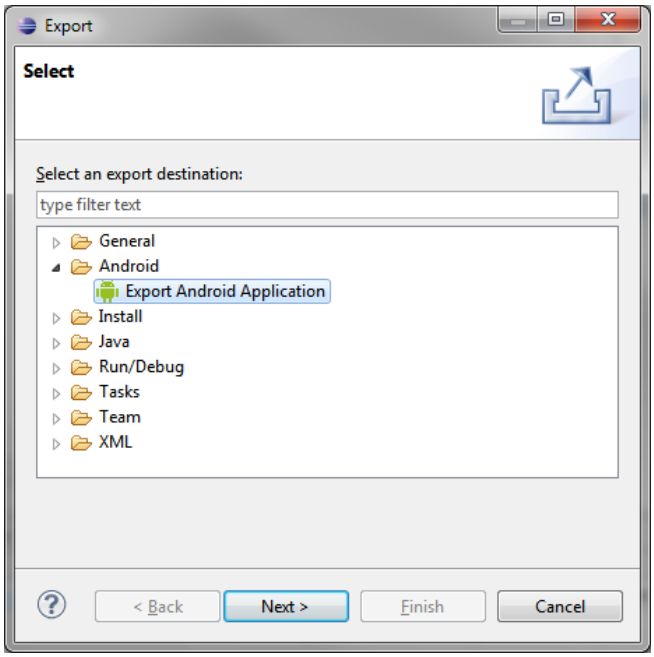
4. Run your app in the emulator or device, you can see the app name is changed.
5. Now, change the locale setting in the emulator or device by going in **Settings > Language and keyboard > Select language > 中文 (繁體).** You should see the name of the app will be automatically changed to Chinese.

*Note: If the launcher cannot display the Chinese app name, try to force close the launcher in **Settings > Applications > Manage applications > All > Launcher > Force stop**

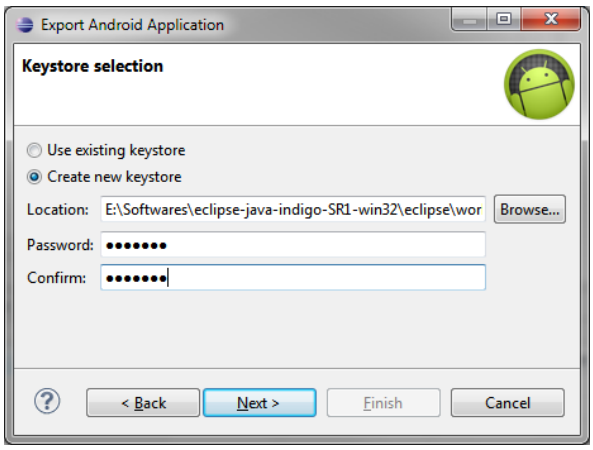
8.3. Sign and export the APK

For every Android app, you have to sign it before uploading it to Android Market or distributing it to other users. The certificate used by ADT for deploying your app in the emulator or device is a debug key only. When exporting the Android app in Eclipse, you can create your own private key if you have not created it before.

1. In Eclipse, highlight your project and select **File > Export > Android > Export Android Application**.

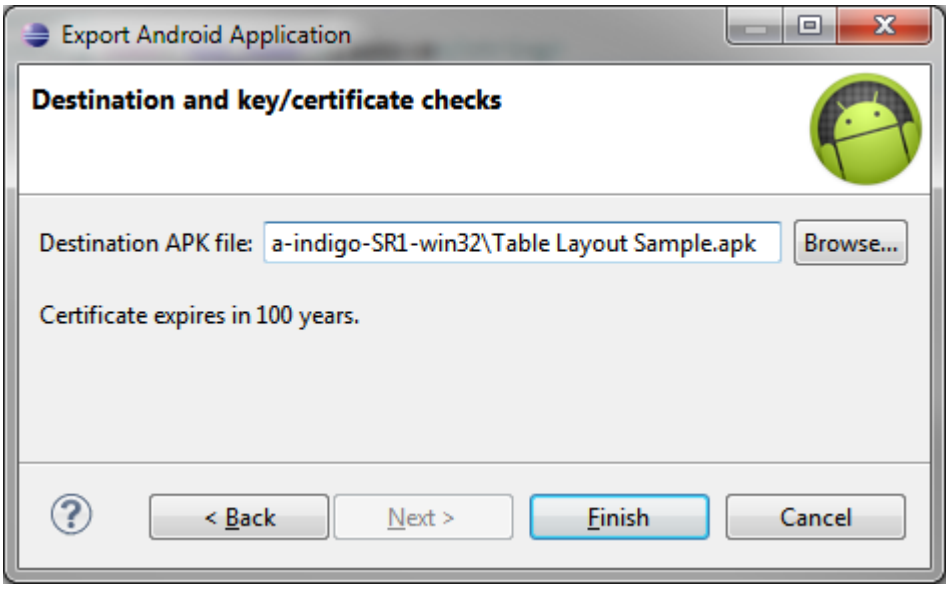


2. Click **Next** to continue if the selected project is correct.
3. Select **Create new keystore**, choose the location for saving the new keystore and enter the password.



4. Enter your personal information in **Key Creation** step. Below is a sample screenshot:

5. Choose your location for saving the published APK file.



6. Now you are ready to distribute your APK file to Android Market or install it directly in any devices.

9. Reference and learning resources:

- Official Android developer website (especially the **Resources** section):
  http://developer.android.com/index.html
- Many common questions raised by other developers previously:
  http://stackoverflow.com/
- Many Android tutorials with complete source code:
  http://www.anddev.org
- Create your app interface online
  http://www.droiddraw.org/
- 3rd party UI development framework:
  https://github.com/cyrilmottier/GreenDroid
- Different ways for implementing listeners in Android app
  http://tseng-blog.nge-web.net/blog/2009/02/14/implementing-listeners-in-your-android-java-application/
- Sams teach yourself Android application development in 24 hours
  Online access from HKBU library:
  http://www.hkbu.edu.hk/lib-cgi/ejour/safari.pl?t=9780132786904)
- Special Interest Group on Innovative Software homepage:
  http://www.comp.hkbu.edu.hk/~sigis/

- End -