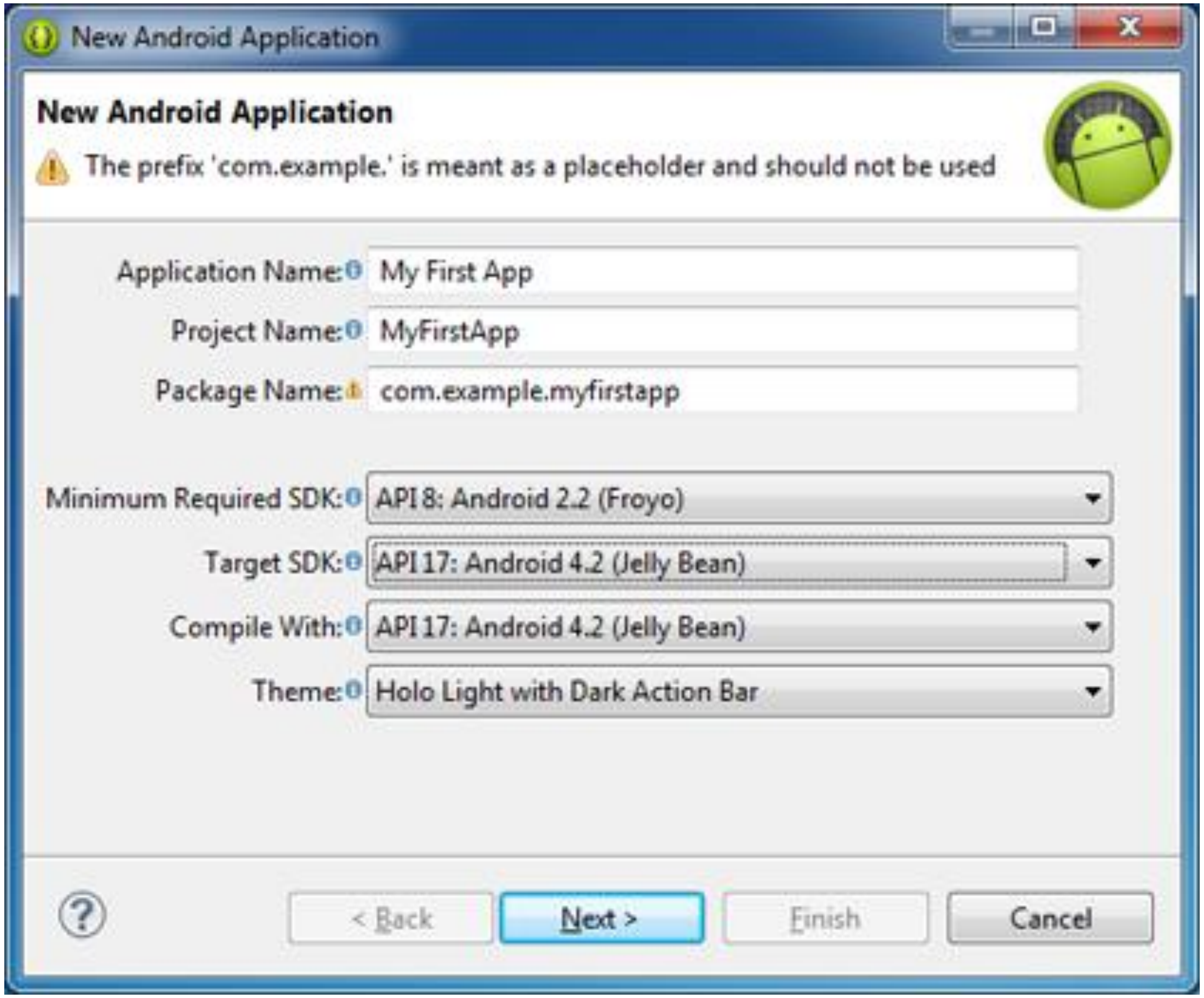
App1

- **Create a Project with Eclipse**
- Click **New** in the toolbar.
- In the window that appears, open the **Android** folder, select **Android Application Project**, and click **Next**.
- Fill in the form that appears:
  - **Application Name** is the app name that appears to users. For this project, use "My First App."
  - **Project Name** is the name of your project directory and the name visible in Eclipse.
  - **Package Name** is the package namespace for your app (following the same rules as packages in the Java programming language). Your package name must be unique across all packages installed on the Android system. For this reason, it's generally best if you use a name that begins with the reverse domain name of your organization or publisher entity. For this project, you can use something like "com.example.myfirstapp." However, you cannot publish your app on Google Play using the "com.example" namespace.
  - **Minimum Required SDK** is the lowest version of Android that your app supports, indicated using the API level. To support as many devices as possible, you should set this to the lowest version available that allows your app to provide its core feature set. If any feature of your app is possible only on newer versions of Android and it's not critical to the app's core feature set, you can enable the feature only when running on the versions that support it (as discussed in Supporting Different Platform Versions). Leave this set to the default value for this project.

App1



New Android Application

The prefix 'com.example.' is meant as a placeholder and should not be used

Application Name:	My First App
Project Name:	MyFirstApp
Package Name:	com.example.myfirstapp
Minimum Required SDK:	API 8: Android 2.2 (Froyo)
Target SDK:	API 17: Android 4.2 (Jelly Bean)
Compile With:	API 17: Android 4.2 (Jelly Bean)
Theme:	Holo Light with Dark Action Bar

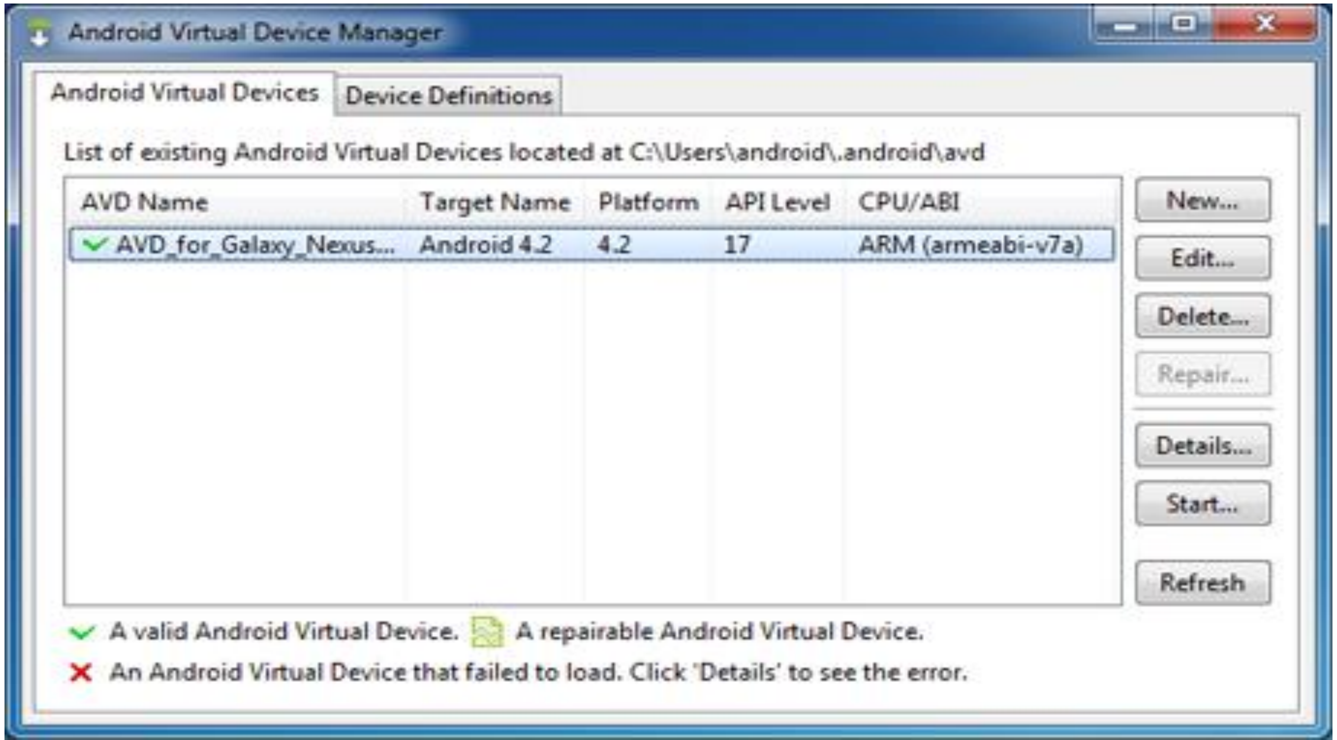
App1

  - **Target SDK** indicates the highest version of Android (also using the API level) with which you have tested with your application.As new versions of Android become available, you should test your app on the new version and update this value to match the latest API level in order to take advantage of new platform features.
  - **Compile With** is the platform version against which you will compile your app. By default, this is set to the latest version of Android available in your SDK. (It should be Android 4.1 or greater; if you don't have such a version available, you must install one using the SDK Manager). You can still build your app to support older versions, but setting the build target to the latest version allows you to enable new features and optimize your app for a great user experience on the latest devices.
  - **Theme** specifies the Android UI style to apply for your app. You can leave this alone.
- Click **Next**.
- On the next screen to configure the project, leave the default selections and click **Next**.
- The next screen can help you create a launcher icon for your app.You can customize an icon in several ways and the tool generates an icon for all screen densities. Before you publish your app, you should be sure your icon meets the specifications defined in the Iconography design guide.
- Click **Next**.
- Now you can select an activity template from which to begin building your app.For this project, select **BlankActivity** and click **Next**.
- Leave all the details for the activity in their default state and click **Finish**.

App1

Run on the Emulator

- Whether you're using Eclipse or the command line, to run your app on the emulator you need to first create anAndroid Virtual Device (AVD). An AVD is a device configuration for the Android emulator that allows you to model different devices.
- To create an AVD:
- Launch the Android Virtual Device Manager:
  - In Eclipse, click Android Virtual Device Manager from the toolbar.
- In the *Android Virtual Device Manager* panel, click **New**.



List of existing Android Virtual Devices located at C:\Users\android\.android\avd

AVD Name	Target Name	Platform	API Level	CPU/ABI
✓ AVD_for_Galaxy_Nexus...	Android 4.2	4.2	17	ARM (armeabi-v7a)

✓ A valid Android Virtual Device. ▨ A repairable Android Virtual Device.
✗ An Android Virtual Device that failed to load. Click 'Details' to see the error.

App1

- Fill in the details for the AVD. Give it a name, a platform target, an SD card size, and a skin (HVGA is default).
- Click **Create AVD**.
- Select the new AVD from the *Android Virtual Device Manager* and click **Start.**
- After the emulator boots up, unlock the emulator screen.

To run the app from Eclipse:

- click **Run** from the toolbar.
- In the **Run as** window that appears, select **Android Application** and click **OK**.

App2

Create a Linear Layout

In this lesson, you'll create a layout in XML that includes a text field and a button.

- Open the activity_main.xml file from the res/layout/ directory.
- The BlankActivity template you chose when you created this project includes the activity_main.xml file with a RelativeLayout root view and a TextView child view.
- First, delete the <TextView> element and change the <RelativeLayout> element to <LinearLayout>. Then add the android:orientation attribute and set it to "horizontal". The result looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
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:orientation="horizontal" >
</LinearLayout>
```

App2

- LinearLayout is a view group (a subclass of ViewGroup) that lays out child views in either a vertical or horizontal orientation, as specified by the android:orientation attribute. Each child of a LinearLayoutappears on the screen in the order in which it appears in the XML.
- The other two attributes, android:layout_width and android:layout_height, are required for all views in order to specify their size.
- Because the LinearLayout is the root view in the layout, it should fill the entire screen area that's available to the app by setting the width and height to "match_parent". This value declares that the view should expand its width or height to *match* the width or height of the parent view.

App2

Add a Text Field (activity_main.xml)

- To create a user-editable text field, add an <EditText> element inside the <LinearLayout>.
- Like every View object, you must define certain XML attributes to specify the EditText object's properties. Here's how you should declare it inside the <LinearLayout> element:

```
<EditText android:id="@+id/edit_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message" />
```

App2

Add String Resources

- When you need to add text in the user interface, you should always specify each string as a resource. String resources allow you to manage all UI text in a single location, which makes it easier to find and update text. Externalizing the strings also allows you to localize your app to different languages by providing alternative definitions for each string resource.
- By default, your Android project includes a string resource file at res/values/strings.xml. Add a new string named "edit_message" and set the value to "Enter a message." (You can delete the "hello_world" string.)
- While you're in this file, also add a "Send" string for the button you'll soon add, called "button_send".
- The result for strings.xml looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
    <resources>
      <string name="app_name">My First App</string>
      <string name="edit_message">Enter a message</string>
      <string name="button_send">Send</string>
      <string name="action_settings">Settings</string>
      <string name="title_activity_main">MainActivity</string>
    </resources>
```

- In Eclipse, click Run from the toolbar.

App2

Respond to the Send Button

- To respond to the button's on-click event, open theactivity_main.xml layout file and add theandroid:onClick attribute to the <Button> element:
-
```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

Start the Second Activity

- To start an activity, call startActivity() and pass it your Intent. The system receives this call and starts an instance of the Activity specified by the Intent.
- With this new code, the complete sendMessage() method that's invoked by the Send button now looks like this:

```
/** Called when the user clicks the Send button */
    public void sendMessage(View view) {
        Intent intent = new Intent(this, DisplayMessageActivity.class);
        EditText editText = (EditText) findViewById(R.id.edit_message);
        String message = editText.getText().toString();
        intent.putExtra(EXTRA_MESSAGE, message);
        startActivity(intent);
    }
```

App2

Create the Second Activity

- To create a new activity using Eclipse:
- Click **New** in the toolbar.
- In the window that appears, open the **Android** folder and select **Android Activity**. Click **Next**.
- Select **BlankActivity** and click **Next**.
- Fill in the activity details:
  - **Project**: MyFirstApp
  - **Activity Name**: DisplayMessageActivity
  - **Layout Name**: activity_display_message
  - **Title**: My Message
  - **Hierarchial Parent**: com.example.myfirstapp.MainActivity
  - **Navigation Type**: None
- Click **Finish**.

App2

Display the Message

- To show the message on the screen, create a TextView widget and set the text using setText(). Then add theTextView as the root view of the activity's layout by passing it to setContentView().
- The complete onCreate() method for DisplayMessageActivity now looks like this:
-
```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);

   // Get the message from the intent
   Intent intent = getIntent();
   String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);

   // Create the text view
   TextView textView = new TextView(this);
   textView.setTextSize(40);
   textView.setText(message);

   // Set the text view as the activity layout
   setContentView(textView);
}
```