Android Programming Basics

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/android-tutorial/

For live Android training, please see courses at http://courses.coreservlets.com/.

Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this Android tutorial. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
  - JSF 2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 6 or 7 programming, custom mix of topics
  - Ajax courses can concentrate on 1 library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo, etc.) or survey several
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, EJB3, GWT, Hadoop, SOAP-based and RESTful Web Services

Contact hall@coreservlets.com for details

Topics in This Section

- Making and testing Android projects
- Basic program structure
- Java-based layout
- XML-based layout
- Eclipse ADT visual layout editor
- Hybrid layout
- Project structure summary

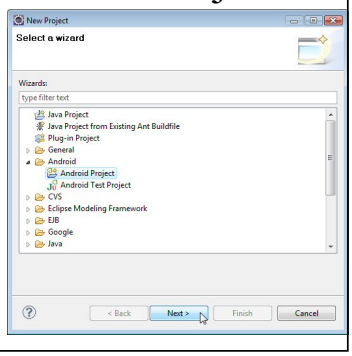
Making an Android Project

Review from Previous Section

- **Already installed**
  - Java 6
  - Eclipse
  - Android SDK
  - Eclipse ADT Plugin
- **Already configured**
  - Android SDK components updated
  - Eclipse preferences
    - Android SDK location set
    - At least one AVD (Android Virtual Device) defined
  - Documentation
    - http://developer.android.com/guide/developing/index.html
    - http://developer.android.com/reference/packages.html

Making Your Own Android App: Basics

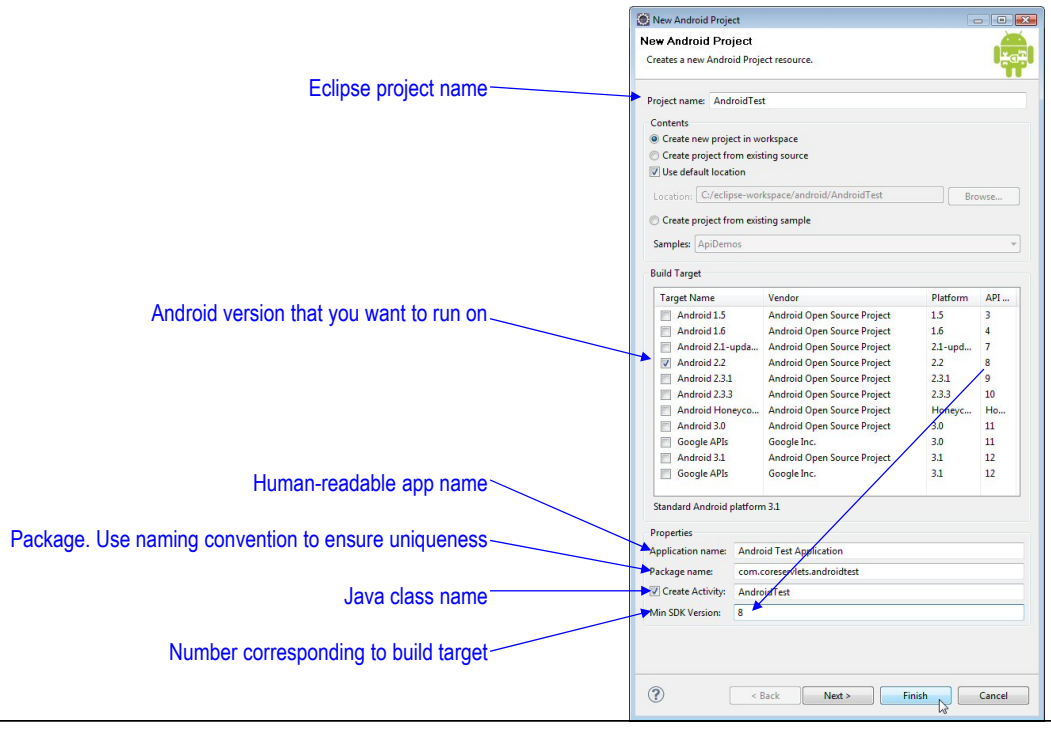
- **Idea**
  - When you create a new app, it has simple "Hello World" functionality built in.
    - So, you can create and test an app without knowing syntax (which is not discussed until next tutorial section)
- **Steps**
  - File → New → Project → Android → Android Project
    - Once you do this once, next time you can do File → New → Android Project
  - Fill in options as shown on next page
  - Run new project as shown previously
    - R-click → Run As → Android Application



Making Your Own Android App: Setting Project Options

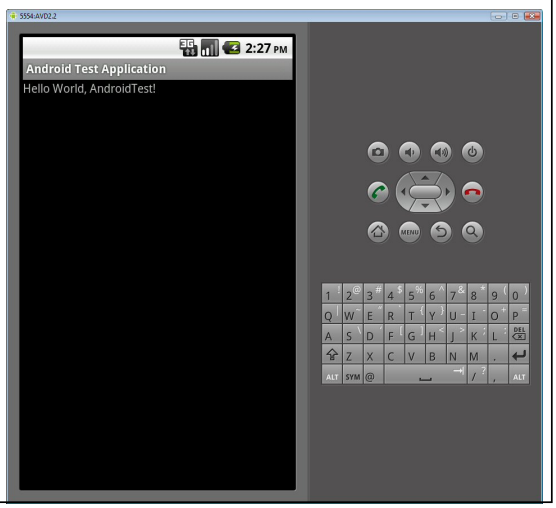
- **New Android Project Settings**
  - Project Name
    - Eclipse project name. Follow naming convention you use for Eclipse.
  - Build Target
    - The Android version that you want to use. For most phone apps, choose 2.2, since that is the most common version in use worldwide.
  - Application name
    - Human-readable app name – title will be shown on Android title bar.
  - Package name
    - Apps on a particular Android device must have unique packages, so use com.yourCompany.project
  - Create Activity
    - The name of the top-level Java class
  - Min SDK Version
    - Number to match the Build Target. Summarized in the Eclipse dialog, but for details, see http://developer.android.com/guide/appendix/api-levels.html

Making Your Own Android App: Setting Project Options



Running New App on Emulator

- **Builtin functionality**
  - Newly created projects automatically have simple "Hello World" behavior
- **Execution steps**
  - Same as with any project
    - R-click → Run As → Android Applicaton
      - Reminder: do not close emulator after testing. Emulator takes a long time to start initially, but it is relatively fast to deploy a new or a changed project to the emulator.

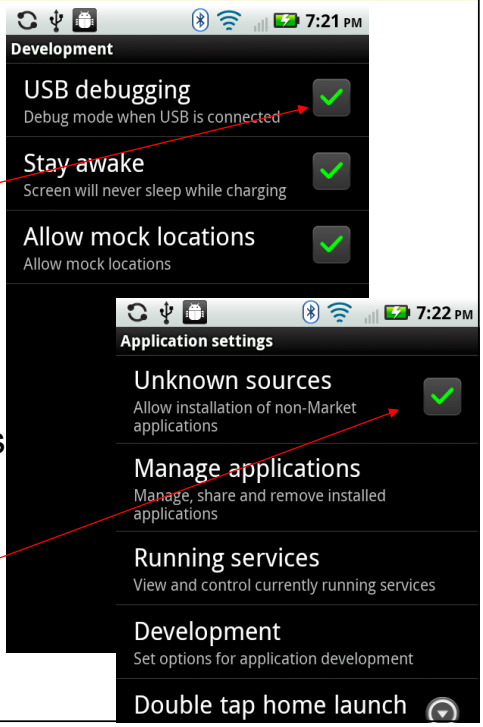


Running New App on Physical Android Device (Phone)

- **Unsigned apps are trivial**
  - Just plug in phone and do normal process from Eclipse
- **Steps**
  - Configure phone to allow untrusted apps
    - Once only. See next page.
  - Shut down emulator
  - Plug in phone
  - R-click project
  - Run As → Android Application
    - This installs and runs it. But it is left installed after you unplug phone, and you can run it on phone in normal manner.

Running New App on Phone: Configuring Android Device

- **Enable USB debugging**
  - Settings → Applications → Development
    - Required: USB debugging
      - Allows PC to send commands via USB
    - Optional: Stay awake
      - Phone/device won't sleep when connected via USB
    - Optional: Allow mock locations
      - Let PC send fake GPS locations
- **Allow unknown sources**
  - Settings → Applications → Unknown sources



Basic Program Structure

General Structure (Common to All Approaches)

```
package com.companyname.projectname;

import android.app.Activity;
import android.os.Bundle;
import android.widget.SomeLayoutOrView;

public class SomeName extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        SomeLayoutOrView view = createOrGetView();
        ...
        setContentView(view);
    }
    ...
}
```

There is no need to type the import statements by hand. Just use the classes in your code, and when Eclipse marks the line as an error, click on the light bulb at the left, or hit Control-1, then choose to have Eclipse insert the import statements for you.

Apps are frequently shut down by the device. This lets you remember some info about the previous invocation. Covered in later lectures, but for now, just know that you should always call super.onCreate as first line of onCreate.

I also follow a few official Android coding conventions here (4-space indentation, no *'s in imports, {'s on same line as previous code, @Override where appropriate). Conventions are strictly enforced in official code, and are used in all examples and tutorials. So, you might as well follow the conventions from the beginning. Follow these simple ones for now, and a later lecture will give coding convention details and provide an Eclipse preferences file to help with them.

Three Main Approaches

- **Java-based**
  - Use Java to define Strings, lay out window, create GUI controls, and assign event handlers. Like Swing programming.
- **XML-based**
  - Use XML files to define Strings, lay out window, create GUI controls, and assign event handlers. The Java method will read the layout from XML file and pass it to setContentView.
- **Hybrid**
  - Use an XML file to define Strings, lay out window and create GUI controls. Use Java to assign event handlers.
- **Examples in this tutorial section**
  - Button that says "Show Greeting". Small popup message appears when button is pressed.
  - Implemented each of the three ways.

Java-Based Approach: Template

```
public class SomeName extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        String message = "...";
        LinearLayout window = new LinearLayout(this);
        window.setVariousAttributes(…);
        Button b = new Button(this);
        b.setText("Button Label");
        b.setOnClickListener(new SomeHandler());
        mainWindow.addView(b);
        ...
        setContentView(window);
    }
    private class SomeHandler implements OnClickListener {
       @Override
        public void onClick(View clickedButton) {
            doSomething(...);
        }
    } }
```

OnClickListener is a public inner class inside View. But, as long as you import android.view.View.OnClickListener, you use it just like a normal class. And, remember that Eclipse helps you with imports: just type in the class name, then either click on the light bulb or hit Control-1 to have Eclipse insert the proper import statements for you.

XML-Based Approach: Template

- **Java**

```
public class SomeClass extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
    public void handlerMethod(View clickedButton) {
        String someName = getString(R.string.some_name);
        doSomethingWith(someName);
    }  }
```

- **XML**

res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="some_name">…</string>
  …
</resources>
```

res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout …>
   <TextView … />
   <Button … android:onClick="handlerMethod" />
</LinearLayout>
```

Hybrid Approach: Template

- **Java**

```
public class SomeClass extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button b = (Button)findViewById(R.id.button_id);
        b.setOnClickListener(new SomeHandler());
    }
    private class SomeHandler implements OnClickListener {
       @Override
        public void onClick(View clickedButton) {
            doSomething(...);
        }
    } }
```

- **XML**
  - Controls that need handlers are given IDs
  - You do *not* use android:onClick to assign handler

Java-Based Layout

Big Idea

- **Approach**
  - Use Java to define Strings, lay out window, create GUI controls, and assign event handlers.
- **Advantages**
  - Familiar to Java desktop developers. Like approach used for Swing, SWT, and AWT.
  - Good for layouts that are dynamic (i.e., that change based on program logic).
- **Disadvantages**
  - Harder to maintain (arguable, but general consensus)
  - Works poorly with I18N
  - *Not* generally recommended except for dynamic layouts
    - But still acceptable for App Store. Whatever works best for your programmers and your app. No code police.

Code (Main Method)

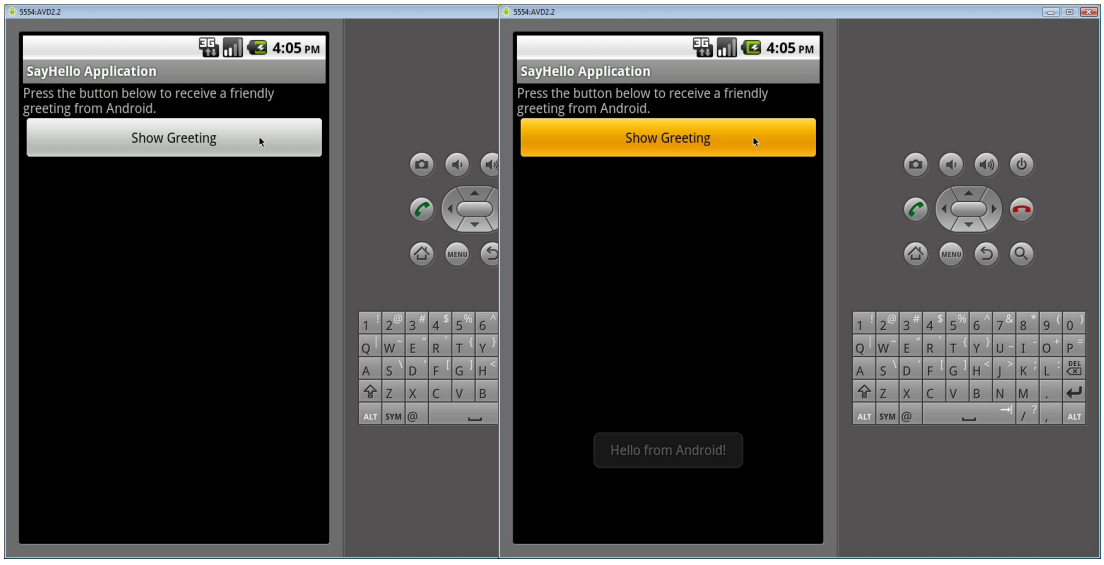
```
public class SayHelloJava extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        String appName = "SayHello Application";
        String windowText =
                "Press the button below to receive " +
                "a friendly greeting from Android.";
        String buttonLabel = "Show Greeting";
        LinearLayout mainWindow = new LinearLayout(this);
        mainWindow.setOrientation(LinearLayout.VERTICAL);
        setTitle(appName);
        TextView label = new TextView(this);
        label.setText(windowText);
        mainWindow.addView(label);
        Button greetingButton = new Button(this);
        greetingButton.setText(buttonLabel);
        greetingButton.setOnClickListener(new Toaster());
        mainWindow.addView(greetingButton);
        setContentView(mainWindow);
    }
```

Code (Event Handler Method)

```
private class Toaster implements OnClickListener {
    @Override
    public void onClick(View clickedButton) {
        String greetingText = "Hello from Android!";
        Toast tempMessage =
                Toast.makeText(SayHelloJava.this,
                               greetingText,
                               Toast.LENGTH_SHORT);
        tempMessage.show();
    }
}
```

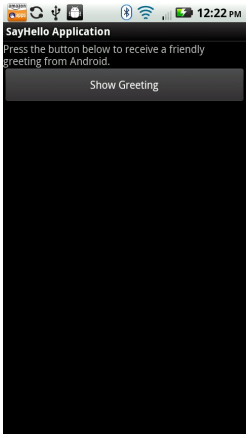

Results on Emulator

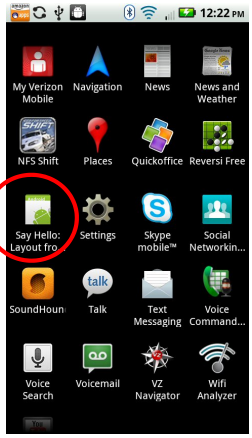
- **Reminder**
  - R-clicked project, Run As → Android Application



Results on Physical Phone

- **Reminder**
  - Configured phone (once only)
  - Shut down emulator, plugged in phone
  - R-clicked project, Run As → Android Application

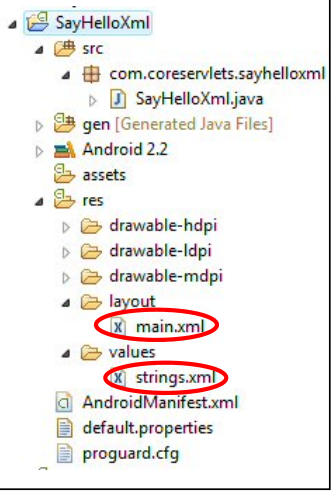




XML-Based Layout

Big Idea

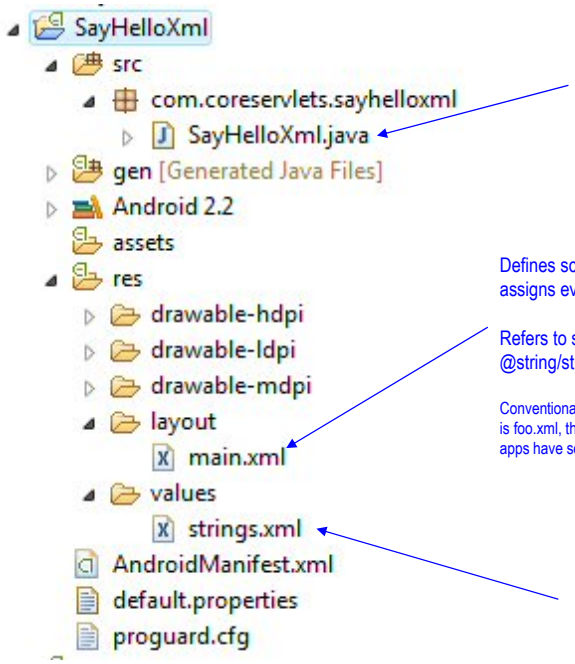
- **Approach**
  - Use XML files to define Strings, lay out window, create GUI controls, and assign event handlers.
    - Define layout and controls in res/layout/main.xml
    - Define Strings in res/values/strings.xml
- **Advantages**
  - Easier to maintain
  - Works well with I18N
  - Can use visual layout editor in Eclipse
  - Standard/recommended approach (along with hybrid)
- **Disadvantages**
  - Works poorly for dynamic layouts



More Details

- **res/layout/main.xml**
  - Define layout and controls with XML description
    - <LinearLayout …>Define controls</LinearLayout>
  - Refer to strings (from strings.xml) with @string/string_name
  - Assign event handler with android:onClick
- **res/values/strings.xml**
  - Define strings used in GUI or that might change with I18N
- **Java code**
  - Refer to layout with R.layout.main
  - Refer to strings with getString(R.string.string_name)
  - Refer to controls with findViewById(R.id.some_id)
- **More info**
  - http://developer.android.com/guide/topics/ui/declaring-layout.html

Project Layout



Refers to layout defined in res/layout/main.xml with R.layout.main.
Refers to strings defined in res/values/strings.xml with getString(R.string.string_name)

Defines screen layout and GUI controls. Optionally assigns event handlers to controls.

Refers to strings defined in res/values/strings.xml with @string/string_name

Conventional for main file to be called main.xml, but not required. If it is foo.xml, then Java uses R.layout.foo. As we will see later, complex apps have several layout files for different screens.

Defines strings that are either used in GUI controls or that might change with internationalization.

Code (res/layout/main.xml)

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

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/window_text"/>
    <Button
        android:text="@string/button_label"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="showToast"/>
</LinearLayout>
```

These attributes (android:orientation, etd.) are defined in JavaDoc API for LinearLayout.

These strings are defined in res/values/strings.xml

This must be a *public* method in main class, have a void return type, and take a View as argument. No interface needs to be implemented, as it does with event handlers referred to in Java code.

Code (res/values/strings.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Say Hello Application</string>
    <string name="window_text">
        Press the button below to receive
        a friendly greeting from Android.
    </string>
    <string name="button_label">Show Greeting</string>
    <string name="greeting_text">Hello from Android!</string>
</resources>
```

app_name is used for the title of the screen. When you create the project, this name is used automatically, but it can be overridden in AndroidManifest.xml. All the rest are developer-specified names.

main.xml refers to this with @string/greeting_text
Java refers to this with getString(R.string.greeting_text)

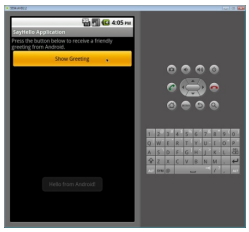
Eclipse auto-completion will recognize the names when editing other files that use them.

Code (Java)

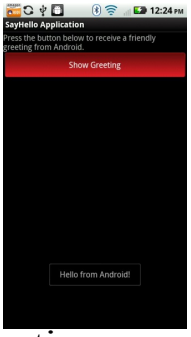
```
public class SayHelloXml extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void showToast(View clickedButton) {
        String greetingText = getString(R.string.greeting_text);
        Toast tempMessage =
            Toast.makeText(this, greetingText,
                           Toast.LENGTH_SHORT);
        tempMessage.show();
    }
}
```

Results

- **On emulator**
  - R-clicked project, Run As → Android Application



  - *Exactly* same look and behavior as previous (Java-based) example



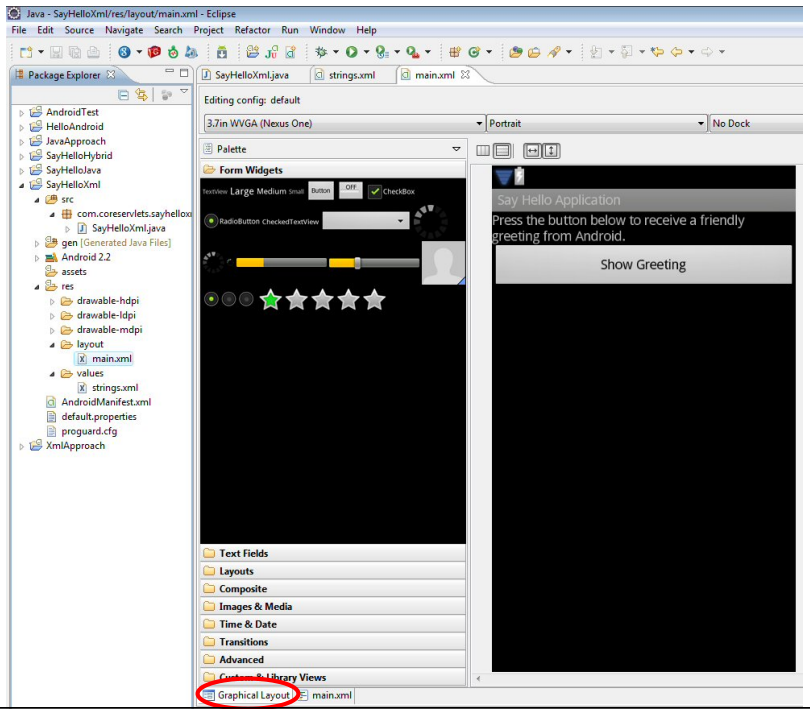
- **On physical phone**
  - Configured phone (once only)
  - Shut down emulator, plugged in phone
  - R-clicked project, Run As → Android Application
  - *Exactly* same look and behavior as previous (Java-based) example

Eclipse ADT Visual Layout Editor

Eclipse Visual GUI Builder and Editor

- **Invoking**
  - When editing main.xml, click Graphical Layout
- **Features**
  - Can interactively change layout attributes (vertical/horizontal, fill characteristics, etc.)
  - Can drag from palette of available GUI controls
  - Can interactively set control characteristics (colors, fill, event handler, etc.)
  - Shows visual preview
- **Warning**
  - Although visual editor is very useful, you should still manually edit XML to fix indentation, order of attributes, use of obsolete attribute names (fill_parent instead of match_parent), and other stylistic things.
- **More info**
  - http://tools.android.com/recent
  - http://www.youtube.com/watch?v=Oq05KqjXTvs

Eclipse Visual Layout Editor



Hybrid Layout

Big Idea

- **Approach**
  - Use XML files to define Strings, lay out window, and create GUI controls.
  - Use Java to assign event handlers.
- **Advantages**
  - Mostly same as XML-based approach
  - But, since event handler needs to be *edited* by Java programmer anyhow, often makes more sense to *assign* it programmatically as well.
- **Disadvantages**
  - Works poorly for dynamic layouts

Code (res/layout/main.xml)

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

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/window_text"/>
    <Button
        android:id="@+id/greeting_button"
        android:text="@string/button_label"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

</LinearLayout>
```

We define an id for the button, so that the button can be referred to in Java code with findViewById(R.id.greeting_button)

We do *not* assign an event handler to the button, as we did in the previous example.

Code (res/values/strings.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Say Hello Application</string>
    <string name="window_text">
        Press the button below to receive
        a friendly greeting from Android.
    </string>
    <string name="button_label">Show Greeting</string>
    <string name="greeting_text">Hello from Android!</string>
</resources>
```

No changes from previous example.

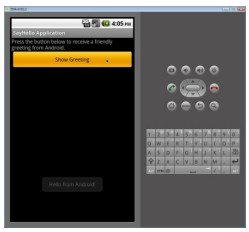
Code (Java)

```
public class SayHelloHybrid extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button greetingButton =
                (Button)findViewById(R.id.greeting_button);
        greetingButton.setOnClickListener(new Toaster());
    }

    private class Toaster implements OnClickListener {
        @Override
        public void onClick(View clickedButton) {
            String greetingText = getString(R.string.greeting_text);
            Toast tempMessage =
                    Toast.makeText(SayHelloHybrid.this,
                                   greetingText,
                                   Toast.LENGTH_SHORT);
            tempMessage.show();
        }
    }}
```

You must call setContentView before calling findViewById. If you call findViewById first, you get null.

Results

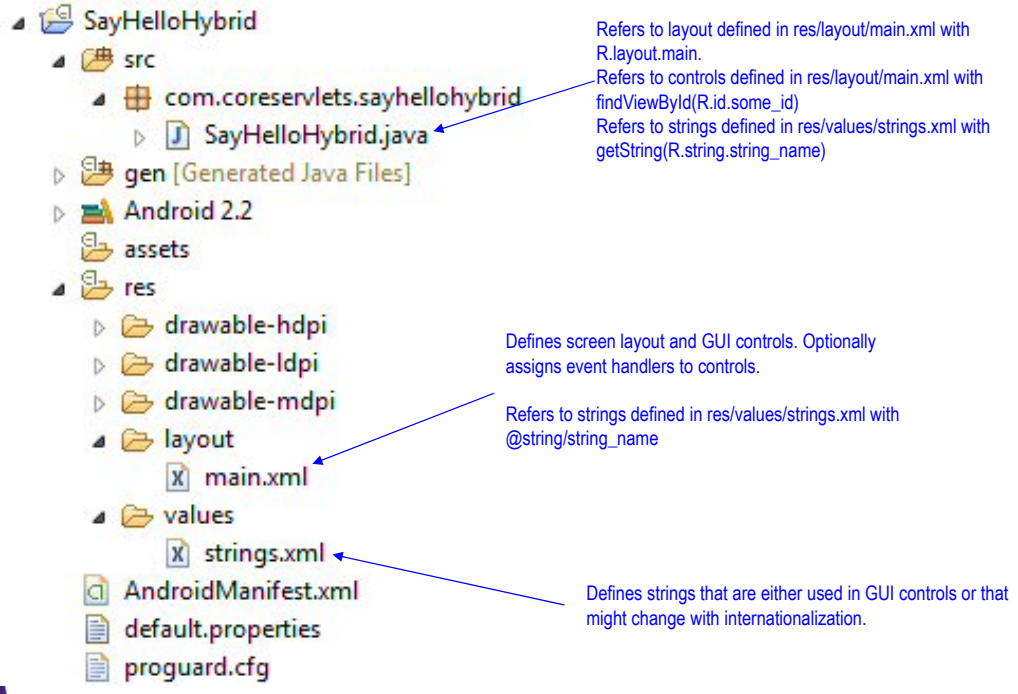
- **On emulator**
  - R-clicked project, Run As → Android Application
  - *Exactly* same look and behavior as previous (Java-based) example



- **On physical phone**
  - Configured phone (once only)
  - Shut down emulator, plugged in phone
  - R-clicked project, Run As → Android Application
  - *Exactly* same look and behavior as previous (Java-based) example

Wrap-Up

Project Layout



Summary

- **XML code**
  - res/layout/main.xml
    - Defines layout properties. Defines GUI controls.
    - Sometimes assigns event handlers to controls
  - res/values/strings.xml
    - Defines Strings used in GUI or for I18N.
- **Java code**
  - Main class extends Action

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    maybeFindControlAndAssignHandler(…);
}
```

Call setContentView *before* calling findViewById.

  - Event handler takes View as argument
    - If assigned programmatically, must implement OnClickListener (or other Listener)

Widget event handling is covered in detail in next tutorial section.

Questions?