Android

development environment

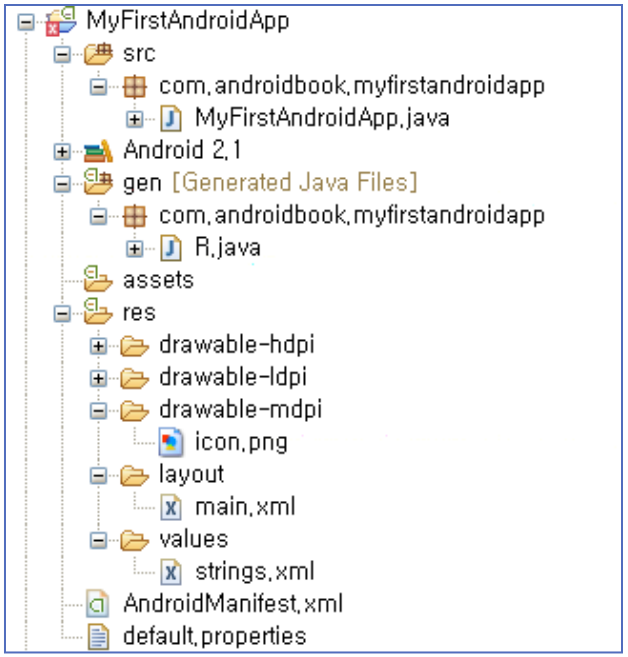
2010.03.19

Database Laboratory
Hanyang Univ.

Android Project Files(1/2)

❖ Android File

- AndroidManifest.xml
  - a global application description file. It defines your application's capabilities and permissions and how it runs.
- Default.properties
  - an automatically created project file. It defines your application's build target and other build system options, as required.



Android Project Files(2/2)

Android File

- src folder
  - Required folder where all source code for the application resides.
- gen folder
  - Required folder where auto-generated resource files for the application reside.
- res folder
  - Required folder where all application resources are managed. Application resources include animations, drawable image assets, layout files, XML files, data resources like strings, and raw files.
- res/drawable/icon.png
  - Application icon displayed on the launcher screen.
- res/layout/main.xml
  - Single screen layout file.
- res/values/strings.xml
  - Application string resources.

Debugging Android App(1/3)

❖ Debugging Android Application

- Before we go any further, you need to become familiar with debugging in the emulator
- In your project, edit the file and create a new method called *forceError()* in your class
- The *forceError()* method forces a new unhandled error in your application.

```
public class MyFirstAndroidApp extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        forceError();
    }

    public void forceError() {
        if(true){
            throw new Error("whoops");
        }
    }
}
```

- It’s probably helpful at this point to run the application and watch what happens.

Debugging Android App(2/3)

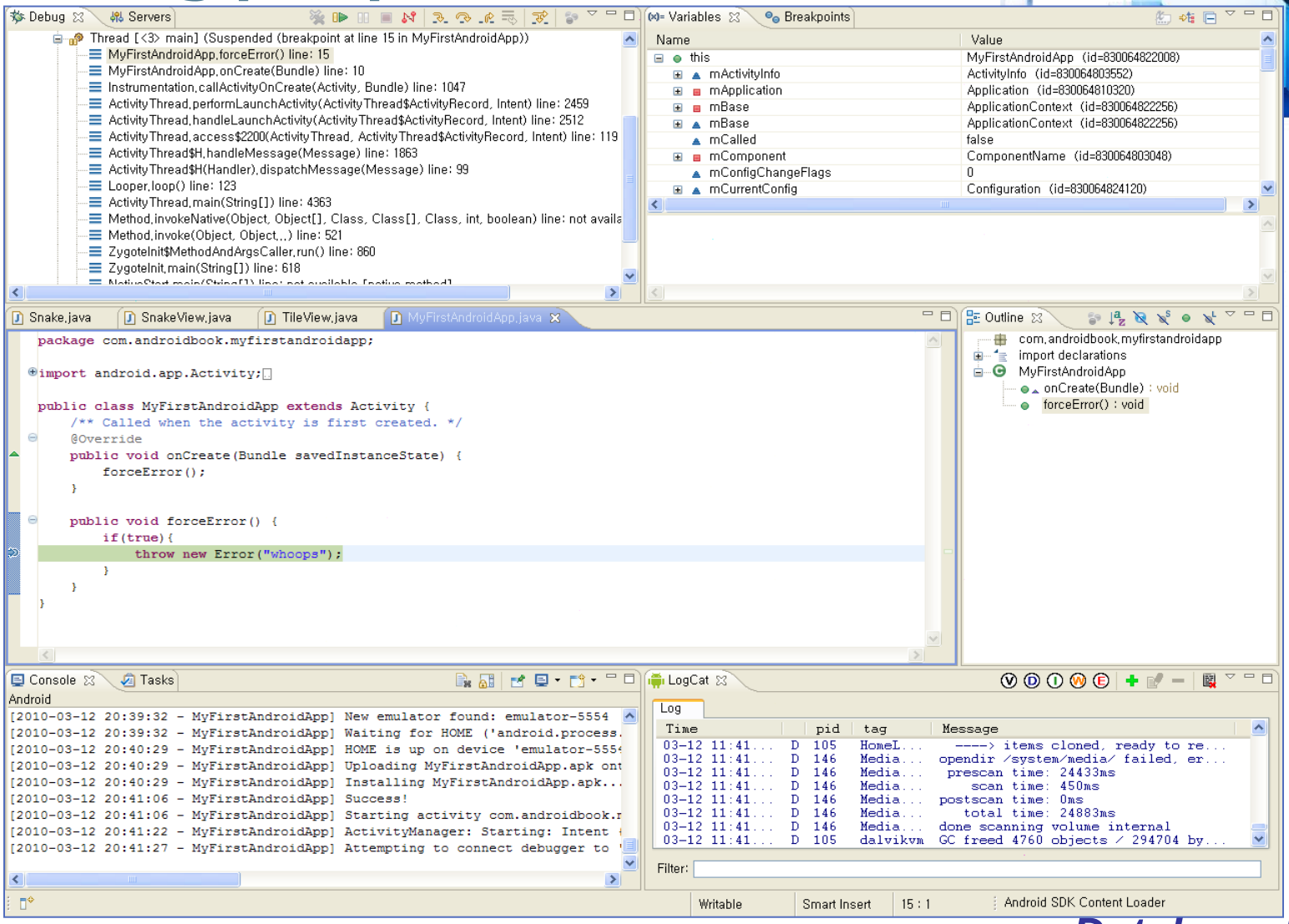
❖ Debugging Android Application

- In the emulator, you see that App has stopped unexpectedly

- choose to Debug your application.
  To do this, click on the little bug drop-down list
- In Eclipse, use the Debug perspective to
  set breakpoints, step through code, and watch the LogCat logging information about your application

Debugging Android App(3/3)

❖ Debug perspectives



Logging Support(1/2)

❖ Logging Support to App

- Need to familiarize yourself with logging

1. First, you must add the appropriate import statement for the Log class.

```
import android.util.Log;
```

2. declare a constant string that you use to tag all logging messages

Method	Purpose
Log.e()	Log errors
Log.w()	Log warnings
Log.i()	Log informational messages
Log.d()	Log Debug messages
Log.v()	Log Verbose messages

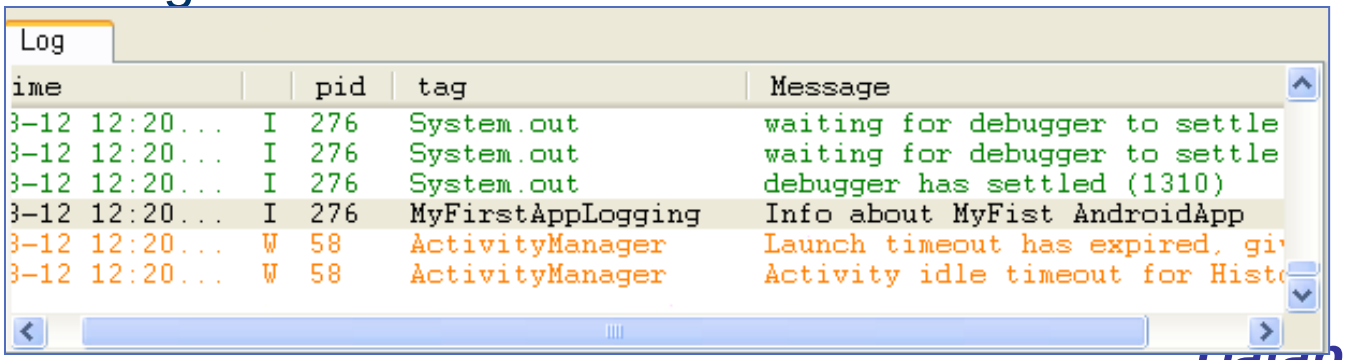
Logging Support(2/2)

❖ Logging Support to App

- You can use the LogCat utility within Eclipse to filter your logging messages to this debug tag

```
public class MyFirstAndroidApp extends Activity {
    private static final String DEBUG_TAG = "MyFirstAppLogging";
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
//        super.onCreate(savedInstanceState);
//        setContentView(R.layout.main);
        Log.i(DEBUG_TAG, "Info about MyFist AndroidApp");
    }
}
```

3. Save your work and debug it in the emulator
4. notice that your logging messages appear in the LogCat listing



Resource Value Types

- **Android applications rely on many different types of resources**
  - text labels, image graphics, and color schemes, for user interface design.
  - These resources are stored in the /res directory of your Android project
  - The resource types supported by the Android SDK
  - The aapt traverses all properly formatted files in the /res directory hierarchy and generates the class file R.java in your source code directory /src

Resource Value Types

Resource Type of	Required Directory	Filename	Key XML Element, if applicable
Strings	/res/values/	**strings.xml (suggested)**	<string>
Arrays of Strings	/res/values/	**arrays.xml (suggested)**	<string-array>
Color Values	/res/values/	**colors.xml (suggested)**	<color>
Dimensions	/res/values/	**dimens.xml (suggested)**	<dimen>
Simple Drawables (Paintable)	/res/values/	**drawables.xml (suggested)**	<drawable>
Bitmap Graphics	/res/drawable/	Examples: **img.png, img.jpg, img.9.png, img.gif, red_oval.xml**	Supported graphics files or Drawable definition XML files like shapes.
Animation Sequences (Tweening)	/res/anim/	Examples: **fancy_anim1.xml, fancy_anim2.xml**	<set>, <alpha>, <scale>, <translate>, <rotate>
Menu Files	/res/menu/	Examples: **my_menu1.xml, more_options.xml**	<menu>
XML Files	/res/xml/	Examples: **some.xml, more.xml**	Defined by the developer.
Raw Files	/res/raw/	Examples: **some_audio.mp3, some_video.mp4, some_text.txt**	
Layout Files	/res/layout/	Examples: **start_screen.xml, main_screen.xml, help_screen.xml**	Varies. Must be a layout element.
Styles and Themes	/res/values/	**styles.xml, themes.xml** (Suggested)	<style>

Working with Resources

Working with String Resources

- String resources are defined in XML under the /res/values project directory
- compiled into the application package at build time.
- Here's an example of a simple string resource file /res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Resource Viewer</string>
    <string name="test_string">Testing 1,2,3</string>
    <string name="test_string2">Testing 4,5,6</string>
</resources>
```

- Bold, Italic, and Underlined Strings

```
<string
  name="txt"><b>Bold</b>,<i>Italic</i>,<u>Line</u></string>
```

- Using String Resources as Format Strings

```
<string
    name="winLose">Score: %1$d of %2$d! You %3$s.</string>
```

Working with Resources

❖ Using String Resources Programmatically

- The following code accesses your application's string resource named hello, returning only the string

```
String myStrHello =
    getResources().getString(R.string.hello);
```

- You can also access the string and preserve the formatting by using this other method

```
CharSequence myBoldStr =
    getResources().getText(R.string.boldhello)
```

- To load a format string, One way to do this is to use the *TextUtils.htmlEncode()* method:

```
import android.text.Html;
import android.text.TextUtils;
...
String myStyledWinString;
myStyledWinString =
    getResources().getString(R.string. winLoseStyled);
String escapedWin = TextUtils.htmlEncode("Won");
String resultText =
    String.format(myStyledWinString, 5, 5, escapedWin);
CharSequence styledResults = Html.fromHtml(resultText);
```

Working with Resources

Working with String Arrays

- You can specify lists of strings in resource files
- String arrays are defined in XML under the /res/values project directory

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="flavors">
        <item>Vanilla Bean</item>
        <item>Chocolate Fudge Brownie</item>
        <item>Strawberry Cheesecake</item>
        <item>Coffee, Coffee, Buzz Buzz Buzz</item>
        <item>Americone Dream</item>
    </string-array>
    <string-array name="soups">
        <item>Vegetable minestrone</item>
        <item>New England clam chowder</item>
        <item>Organic chicken noodle</item>
    </string-array>
</resources>
```

- accessing string arrays resources

```
String[] aFlavors =
    getResources().getStringArray(R.array.flavors);
```

Working with Resources

❖ Working with Colors

- Android applications can store RGB color values, which can then be applied to other screen elements

- #RGB (Example: #F00 is 12-bit color, red)
- #ARGB (Example: #8F00 is 12-bit color, red with alpha 50 percent)
- #RRGGBB (Example: #FF00FF is 24-bit color, magenta)
- #AARRGGBB (Example: #80FF00FF is 24-bit color, magenta with alpha 50 percent)

- Here's an example of a simple color resource file: /res/values/colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="background_color">#006400</color>
    <color name="text_color">#FFE4C4</color>
</resources>
```

- Accessing a color resource

```
int myResourceColor =
    getResources().getColor(R.color.prettyTextColor);
```

Working with Resources

Working with Dimensions

- dimensions can be stored as resources

Unit of Measurement	Description	Resource Tag Required	Example
Pixels	Actual screen pixels	px	20px
Inches	Physical measurement	in	1in
Millimeters	Physical measurement	mm	1mm
Points	Common font Measurement	pt	14pt
Density-Independent Pixels	Pixels relative to 160dpi screen	dp	1dp
Scale-Independent Pixels	Best for scalable font display	sp	14sp

- Here's an example of a simple dimension resource file /res/values/dimens.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="FourteenPt">14pt</dimen>
    <dimen name="OneInch">1in</dimen>
    <dimen name="TenMillimeters">10mm</dimen>
    <dimen name="TenPixels">10px</dimen>
</resources>
```

- accessing dimension resource

```
float myDimension =
    getResources().getDimension(R.dimen.textPointSize);
```

Working with Resources

❖ Working with Menus(1/2)

- Each menu resource is stored as a XML files in the /res/menu directory
- Here's an example of a simple menu resource file /res/menu/speed.xml

```
<menu xmlns:android
    ="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/start"
        android:title="Start!"
        android:orderInCategory="1"></item>
    <item
        android:id="@+id/stop"
        android:title="Stop!"
        android:orderInCategory="4"></item>
    <item
        android:id="@+id/accel"
        android:title="Vroom! Accelerate!"
        android:orderInCategory="2"></item>
    <item
        android:id="@+id/decel"
        android:title="Decelerate!"
        android:orderInCategory="3"></item>
</menu>
```

Working with Resources

Working with Menus(2/2)

- You can create menus using the Eclipse plug-in
- Plug-in can access the various configuration attributes for each menu item
- /res/menu/speed.xml, simply override the method *onCreateOptionsMenu()* in your application

```
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.speed, menu);
    return true;
}
```

To-Do List Example(1/5)

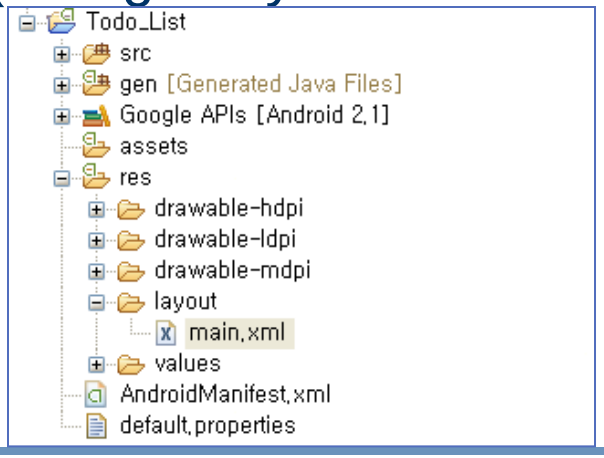
❖To-do list application

- In this example, we want to present users with a list of to-do items and a text entry box to add new ones

1. Start by creating a new Android project

   File> New> Project… , then choose **Android**

2. Enter the details for your project. With the details entered, click Finish to create your new project
3. Open the *main.xml* layout file in the *res/layout* project folder(Using a layout resource file)



To-Do List Example(2/5)

❖To-do list application

4. Modify the main layout to include a *ListView* and an *EditText* within a *LinearLayout.*
   - It's important to give both *EditText* and *ListView* controls IDs so you can get references to them in code

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <EditText
    android:id="@+id/myEditText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="New To Do Item"
  />
  <ListView
     android:id="@+id/myListView"
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
   />
 </LinearLayout>
```

To-Do List Example(3/5)

❖ To-do list application

5. With your user interface defined, open the *ToDoList.java* Activity class from your project's source folder
   - Start by inflating your UI using *setContentView* and then get references to the *ListView* and *EditText* using *findViewByID*.

```
public void onCreate(Bundle icicle) {
  // Inflate your view
  setContentView(R.layout.main);

  // Get references to UI widgets
  ListView myListView = (ListView)findViewById(R.id.myListView);
  final EditText myEditText = (EditText)findViewById(R.id.myEditText);
}
```

6. Still within *onCreate,* define an *ArrayList* of Strings to store each to-do list item.
   - Create a new *ArrayAdapter* instance to bind the to-do item array to the *ListView*.

```
// Create the array list of to do items
final ArrayList<String> todoItems = new ArrayList<String>();
// Create the array adapter to bind the array to the listview
final ArrayAdapter<String> aa;
aa = new ArrayAdapter<String>(this,
                              android.R.layout.simple_list_item_1,
                              todoItems);
// Bind the array adapter to the listview.
myListView.setAdapter(aa);
```

To-Do List Example(4/5)

❖To-do list application

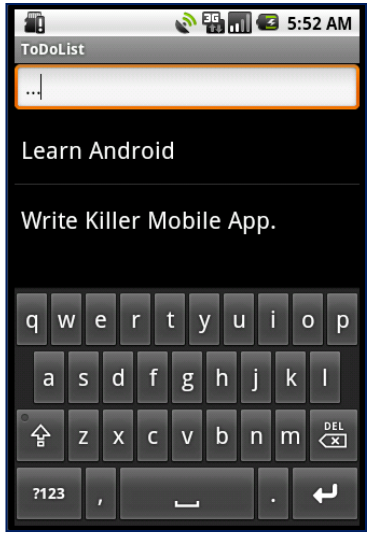
7. Make this to-list functional is to let users add new to-do items.
   - Add an onKeyListerner to the EditText that listens for a “D-pad center button” click before adding the contents of the EditText to the to-do list array and notifying the ArrayAdapter of the change

```
myEditText.setOnKeyListener(new OnKeyListener() {
  public boolean onKey(View v, int keyCode, KeyEvent event) {
    if (event.getAction() == KeyEvent.ACTION_DOWN)
      if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER)
      {
        todoItems.add(0, myEditText.getText().toString());
        aa.notifyDataSetChanged();
        myEditText.setText("");
        return true;
      }
    return false;
  }
});
```

To-Do List Example(5/5)

❖To-do list application

7. Run or debug application, and you'll see a text entry



8. Try adding breakpoints to the code to test the debugger and experiment with the DDMS perspective