Objectives

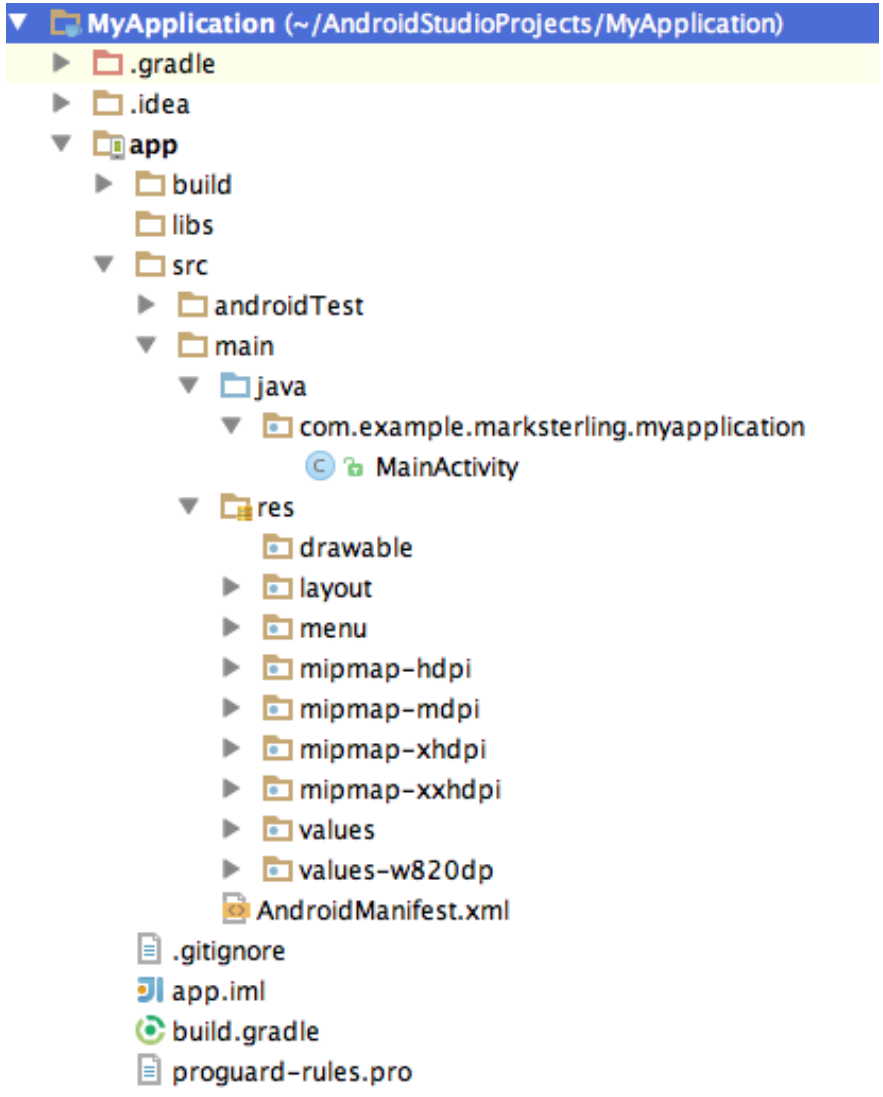
- Discuss Practical Issues of Android vs. iOS Development
- Talk about the Resources Directory
- Show some additional types of `View`
- Go through some coding examples

Device Fragmentation

- Android Positives: larger user base, larger market share
- Android Negatives: greater variety of devices (screen resolutions, screen sizes, localization issues)
- This is what we call *Fragmentation*

Providing External Resources

- Resources: Single place to coordinate issues with Fragmentation
- Drawables: Different images
- Mipmaps: Image resources for different resolutions
- Layouts: Arrangement of Visual Elements on the Screen (May change depending on the device)



Localization

- Value of using indirection in specifying strings
- Make changes in a single place without having to touch our code
- Default and specific behavior is determined by the structure of the resource directory
- Example: `res/values/strings.xml` is the "default" resource for all of our strings, but we can localize to Japanese (for example) by creating a different XML file at `res/values-ja/strings.xml`

Portrait vs. Landscape Orientation, Resolution

- Also create different behaviors depending on whether the device is in portrait or landscape mode
- `res/layout-land` and `res/layout-port`
- Different units of measure
  - density-independent pixel
  - scale-independent pixel
  - pixel
  - point
  - millimeter
  - inch

strings.xml

- The strings resource in `values` can also contain descriptions of more complicated data-types such as lists of strings
- `string-array` is a list of strings that we can use for a picker or spinner

```
<resources>
    <string name="app_name">ApplicationMona</string>

    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>

    <string-array name="spinner_list">
        <item>one</item>
        <item>two</item>
        <item>three</item>
    </string-array>

</resources>
```

A Spinner defined in XML

```
<Spinner
        android:id="@+id/my_nice_spinner"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:entries="@array/spinner_list" />
```

- Define the height and width attributes just like we did in the case of a button
- Unique id to reference this View in our activity
- Reference for the “entries”

Simple usage of the Spinner

- After editing the layout the Spinner can be accessed in the same manner as the TextView or Button from the example shown previously
- Spinner has methods that allow us to do more complicated types of user interactions

```
// Get a reference to the spinner
s = (Spinner) findViewById(R.id.my_nice_spinner);

// Assign return value from spinner method to local variable
String myString = (String) s.getSelectedItem();
```

Providing External Resources

- After compiling the project we should get something that looks approximately like the screenshot to the right
- In-Class Exercise: Add some behavior so that the picker can interact with the button

Changing the Button properties

- Once obtaining a reference to the button we can start doing some interaction

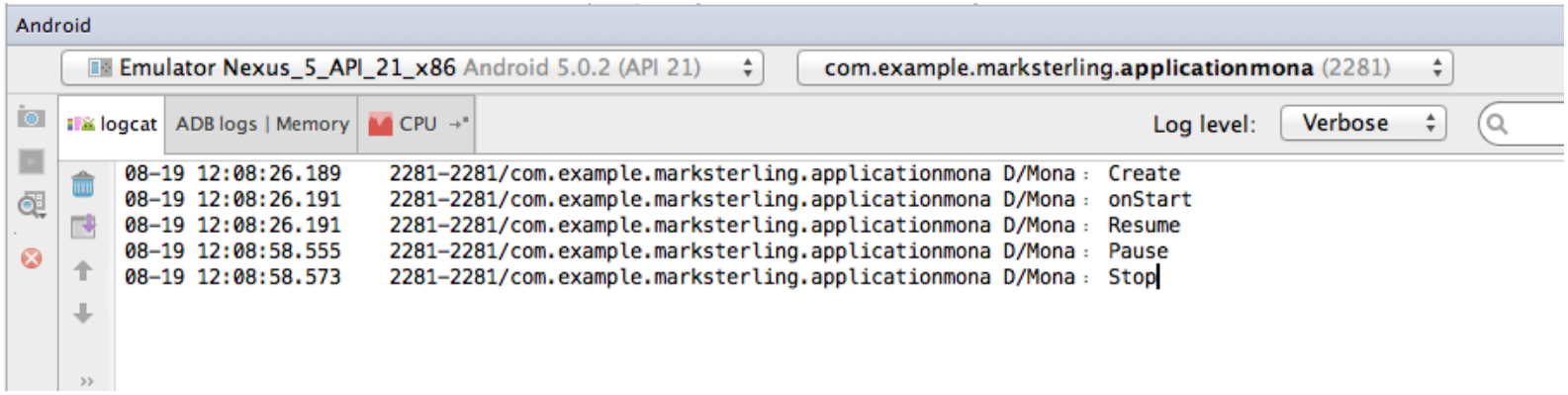
```
public class MainActivity extends Activity {

    private Button b;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        b = (Button) findViewById(R.id.a_nice_button);

        b.setText("New String");

        b.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                b.setText("Button Text");
            }
        });

    }

    // ... additional boilerplate code
}
```

Logging, Viewing the Activity Lifecycle

- The IDE provides a console where we can view debug messages
- Provided with the `Log` class, (`System.out.println()` also works)
- Give a "tag" and the message

```
// ...

    @Override
    protected void onResume() {
        super.onResume();
        Log.d("Mona", "Resume");
    }

    @Override
    protected void onPause() {
        super.onPause();
        Log.d("Mona", "Pause");
    }

    @Override
    protected void onStop() {
        super.onStop();
        Log.d("Mona", "Stop");
    }

// ...
```

Wachting the Activity Lifecycle in the Debugger



- Once we fill in the lifecycle method stubs with Log messages we can watch the activity lifecycle in real-time as we navigate
- This is a good exercise to understand how an application gets launched and what happens, for example, when the user presses the back button or returns to the home screen