Android Course Day 2

Lucio Cossio
Luis Mazoni

Course Agenda

Day 1

- **Development environment and tools**
- **Android project structure**
  - Source, tests , resources, manifest
- **Activity**
  - Lifecycle
  - Layout interaction
  - ActionBar
- **Practice**

Day 2

- **Review Day 1**
- **Resources**
  - String
  - Dimensions
  - Layout
- **Views**
  - TextView, Edit Text, Button
  - String resources
  - View listeners
- **Animation**
- **Practice**

Day 3

- **Review Day 2**
- **Intents**
  - Open new activity
  - Sending data
  - Actions
- **Android Manifest**
  - Overview
  - Add activities
- **Practice**

Day 4

- **Review Day 3**
- **Fragments**
  - Lifecycle
  - Fragment Manager
  - Arguments
- **Practice**

Day 5

- **Review Day 4**
- **ListView**
  - Adapter
  - View Holder
- **Async Tasks**
  - UI Thread
- **Practice**

Review

- The visible lifetime of an activity happens between a call to **onStart**() until a corresponding call to **onStop**()
- The foreground lifetime of an activity happens between a call to **onResume**() until a corresponding call to **onPause**()
- **onPause**() is always called when an activity is being placed in the background or on its way to destruction. One example of when **onPause**() and **onStop**() is called and **onSaveInstanceState**(Bundle) is not is when a user navigates back from activity B to activity A: there is no need to call **onSaveInstanceState**(Bundle) on B because that particular instance will never be restored, so the system avoids calling it.
- **onStop**() may never be called, in low memory situations where the system does not have enough memory to keep your activity's process running after its onPause() method is called.

Resources

Located under res folder

There are several types of resources, such as:

- Animation
- Color State List
- Drawable
- Layout
- Menu
- String
- Style
- Bool
- Color
- Dimension
- ID
- Integer
- Integer Array
- Typed Array

Resources - Specifying for configurations

- Any resource folder can be appended with a configuration suffix, such as:
- Language and Region (en, fr, pt-rBR…)
- Screen size (small, normal, large, xlarge)
- Orientation (port, land)
- UI mode (car, desk, television, appliance and watch)
- Screen pixel density (ldpi, mdpi, hdpi, xhdpi, xxhdpi, xxxhdpi, nodpi and tvdpi)

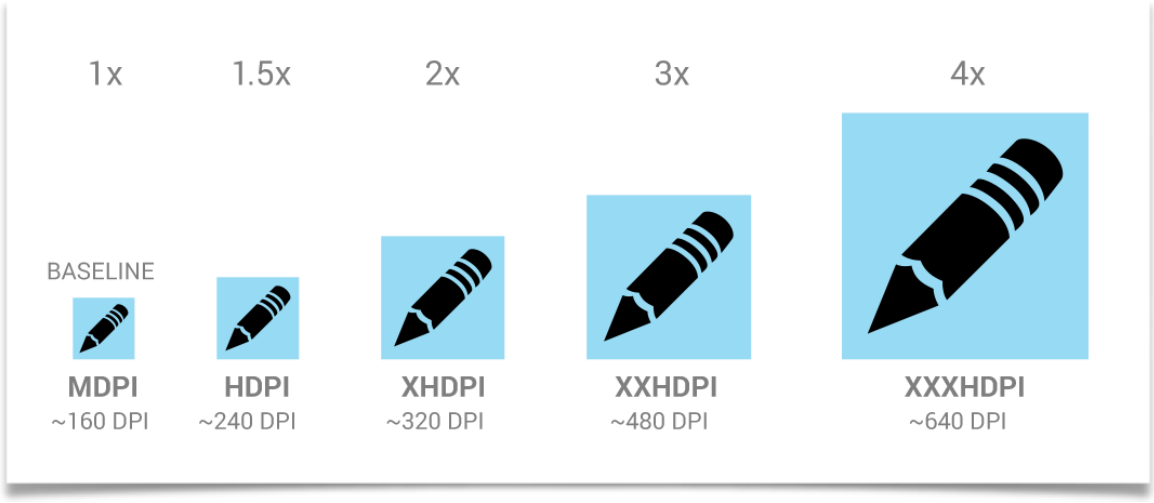
We'll be using the following

- Drawable
- String
- Dimension
- Color
- Layout

Drawable Resources

A Drawable can be a bitmap file, like .png, .jpg, or .gif file. Android creates a Drawable resource for any of these files when you save them in the res/drawable/ directory.

When creating icons, it's important to keep in mind that your app may be installed on a variety of devices that offer a range of pixel densities. But you can make your icons look great on all devices by providing each icon in multiple sizes.



String Resources

There are three kind of string resources, that are all described as XML:

- **String**
- String Array
- Quantity String (Plurals)

String Resource - String

Defining it

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello!</string>
</resources>
```

Using it

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

or

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

Dimension

A dimension value defined in XML. A dimension is specified with a number followed by a unit of measure. For example: 10px, 2in, 5sp, 8dp

Defining it

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen
        name="padding_top"
        >5dp</dimen>
</resources>
```

Using it

```
<TextView android:layout_width="match_parent"
    android:gravity="center"
    android:layout_height="wrap_content"
    android:textSize="100dp"
    android:id="@+id/time"
    android:paddingTop="@dimen/padding_top"
    android:text="00:00"/>
```

Color

A color value defined in XML. The color is specified with an RGB value and alpha channel. You can use a color resource any place that accepts a hexadecimal color value. You can also use a color resource when a drawable resource is expected in XML (for example, android:drawable="@color/green").

Defining it

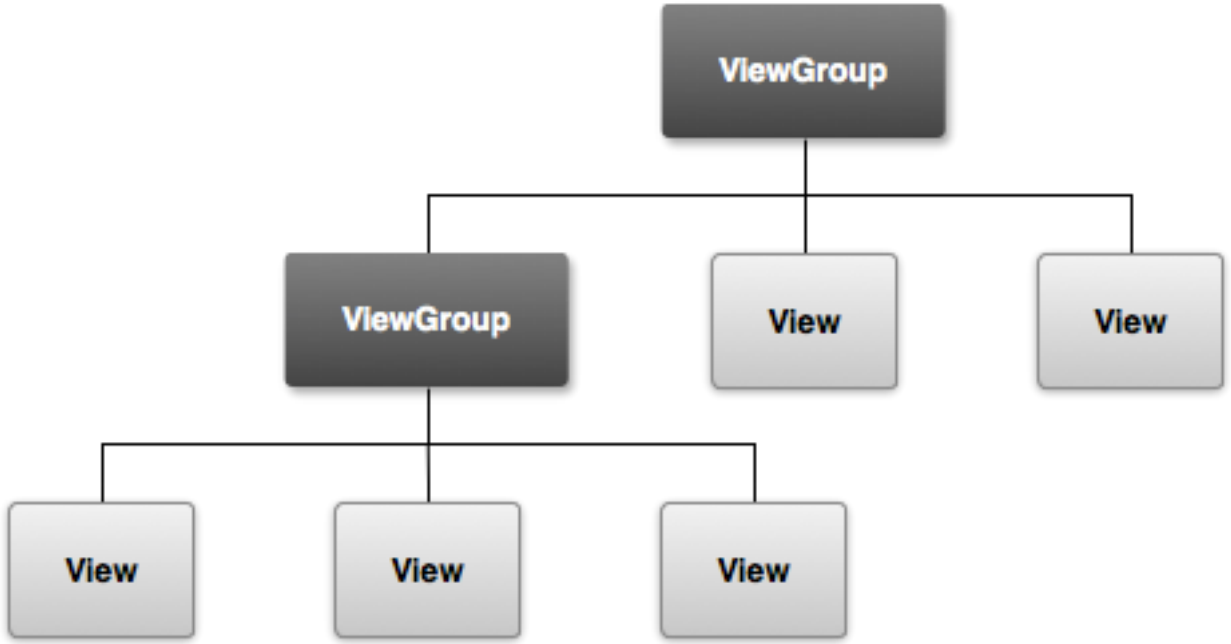
```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="green">#ff00FF00</color>
</resources>
```

Using it

```xml
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="@color/green"/>
```

Layout

The graphical user interface for an Android app is built using a hierarchy of View and ViewGroup objects.

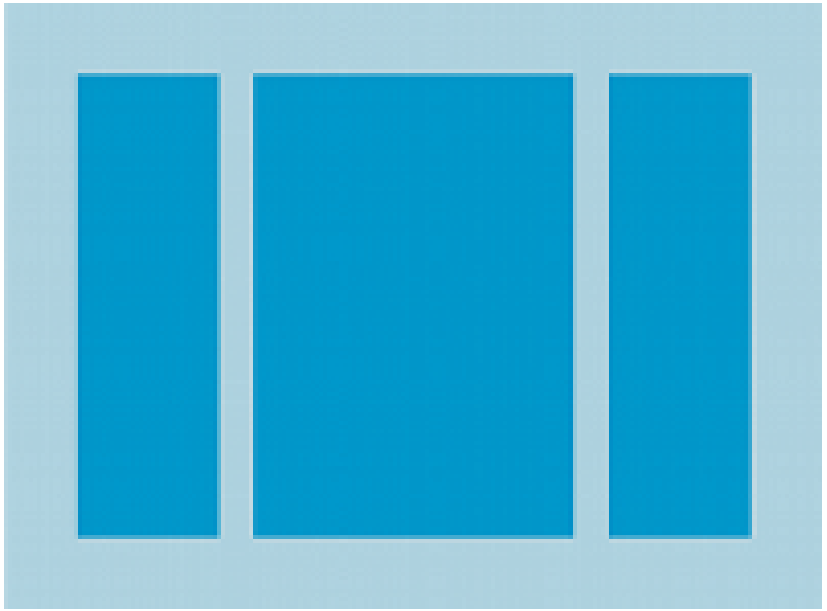


Layout

Common View Groups

Linear Layout

organizes its children into a single horizontal or vertical row. Creates a scrollbar if the length of the window exceeds the length of the screen



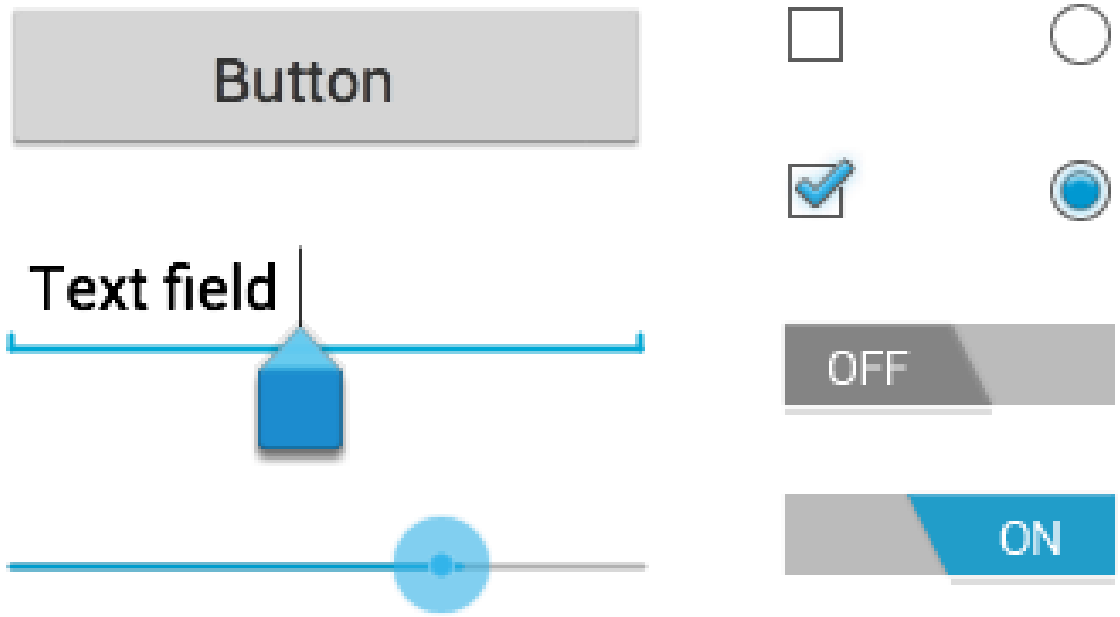
Relative Layout

Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).

Layout - Views

Common Controls

Control Type	Related Classes
Button	Button
Text field	EditText,AutoCompleteTextView
Checkbox	CheckBox
Radio button	RadioGroup RadioButton
Toggle button	ToggleButton
Spinner	Spinner
Pickers	DatePicker,TimePicker



Layout - XML Definition

```
<?xml version="1.0" encoding="utf-8"?>
<ViewGroup xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@[+][package:]id/resource_name"
    android:layout_height=["dimension" | "fill_parent" | "wrap_content"]
    android:layout_width=["dimension" | "fill_parent" | "wrap_content"]
    [ViewGroup-specific attributes] >
    <View
        android:id="@[+][package:]id/resource_name"
        android:layout_height=["dimension" | "fill_parent" | "wrap_content"]
        android:layout_width=["dimension" | "fill_parent" | "wrap_content"]
        [View-specific attributes] >
        <requestFocus/>
    </View>
    <ViewGroup >
        <View />
    </ViewGroup>
    <include layout="@layout/layout_resource"/>
</ViewGroup>
```

Layout - Using your XML

- As an Activity

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

- With layout inflater

```
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    View paymentView = inflater.inflate(R.layout.payment_page_fragment, container);
    return paymentView;
}
```

Practice:
Simple login form

Listeners

All Views have the following events that could be listen to:

- onClick()
- onLongClick()
- onFocusChange()
- onKey()
- onTouch()
- onCreateContextMenu()

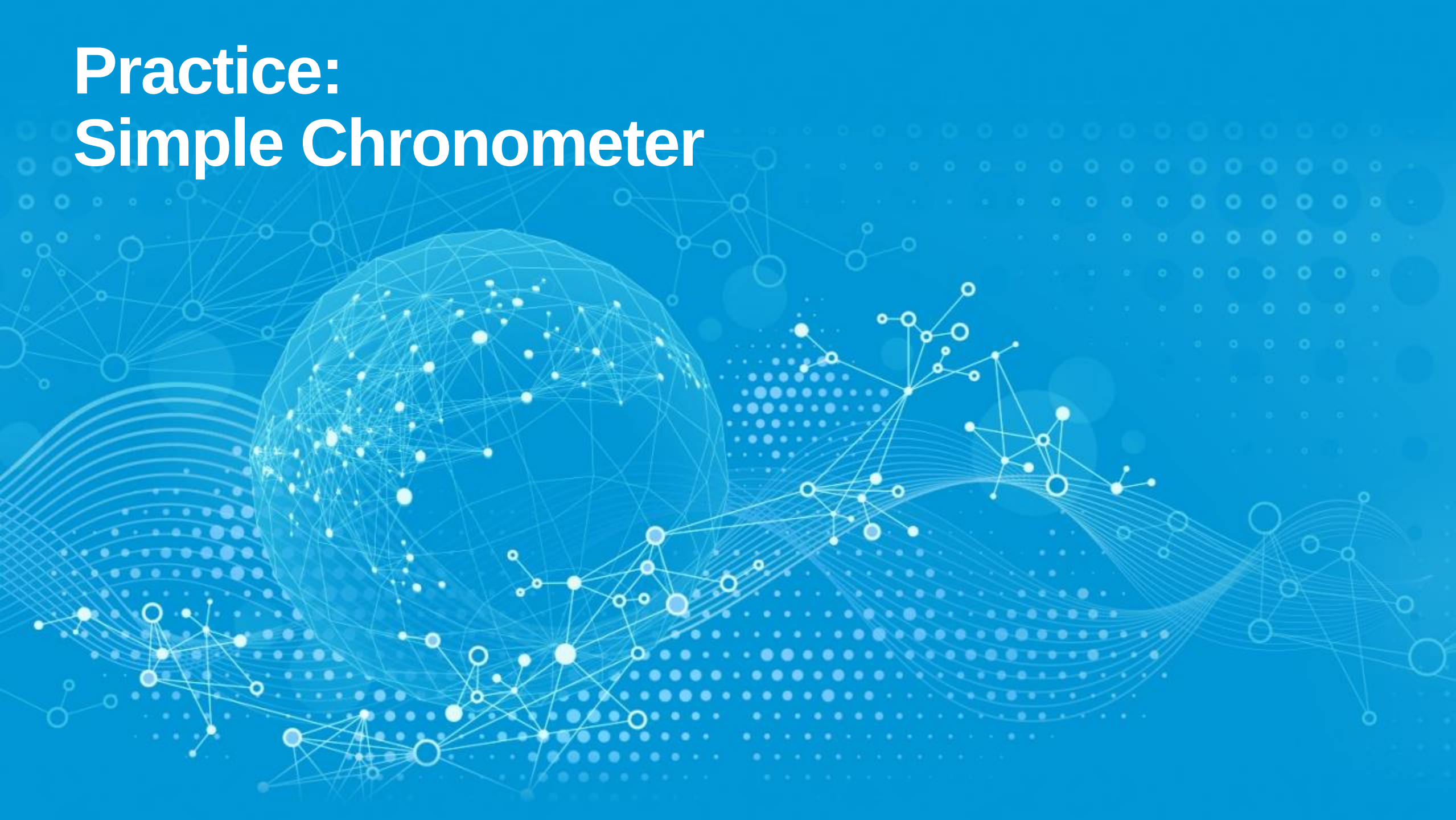
Listeners

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Stop"
    android:id="@+id/stop"
    android:layout_weight="1"
    android:layout_below="@+id/pause"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:onClick="onClick"/>
```

or

```
protected void onCreate(Bundle savedValues) {
    ...
    // Capture our button from layout
    Button button = (Button)findViewById(R.id.corky);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(mCorkyListener);
    ...
}
```

Practice:
Simple Chronometer



Animation

The Android framework provides two animation systems: property animation (introduced in Android 3.0) and view animation. Both animation systems are viable options, but the property animation system, in general, is the preferred method to use, because it is more flexible and offers more features. In addition to these two systems, you can utilize Drawable animation, which allows you to load drawable resources and display them one frame after another.

- Drawable Animation
- View Animation
- Property Animation

Animation – Drawable Animation – Defining it

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="true">
    <item android:drawable="@drawable/rocket_thrust1" android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust2" android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust3" android:duration="200" />
</animation-list>
```

Animation – Drawable Animation – Using it

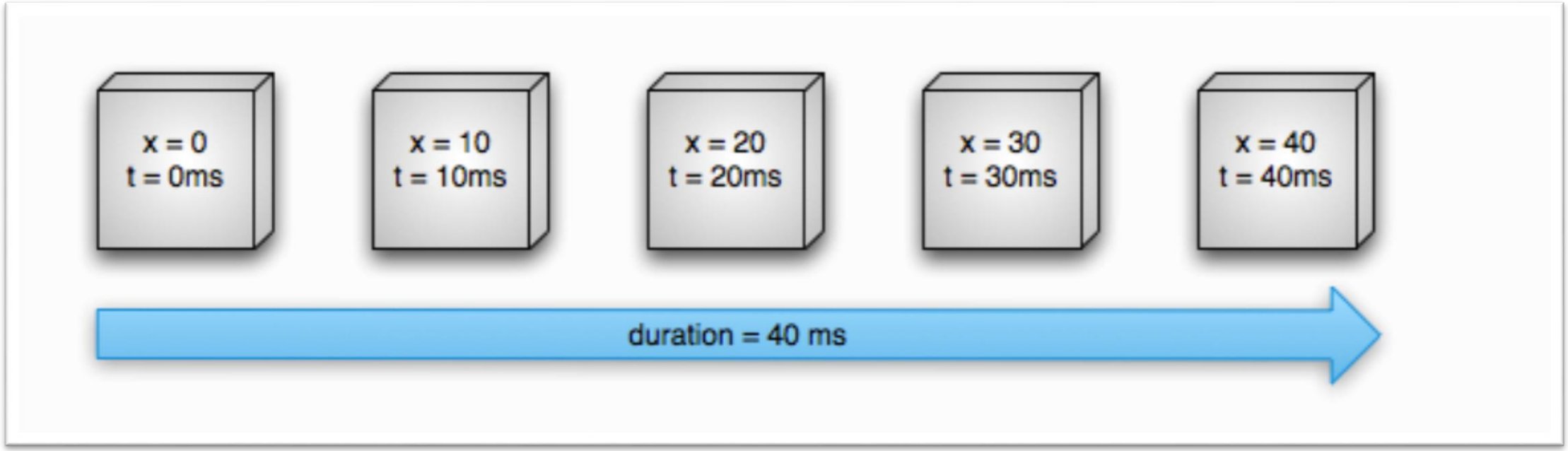
```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  ImageView rocketImage = (ImageView) findViewById(R.id.rocket_image);
  rocketImage.setBackgroundResource(R.drawable.rocket_thrust);
  rocketAnimation = (AnimationDrawable) rocketImage.getBackground();
}

public boolean onTouchEvent(MotionEvent event) {
  if (event.getAction() == MotionEvent.ACTION_DOWN) {
    rocketAnimation.start();
    return true;
  }
  return super.onTouchEvent(event);
}
```

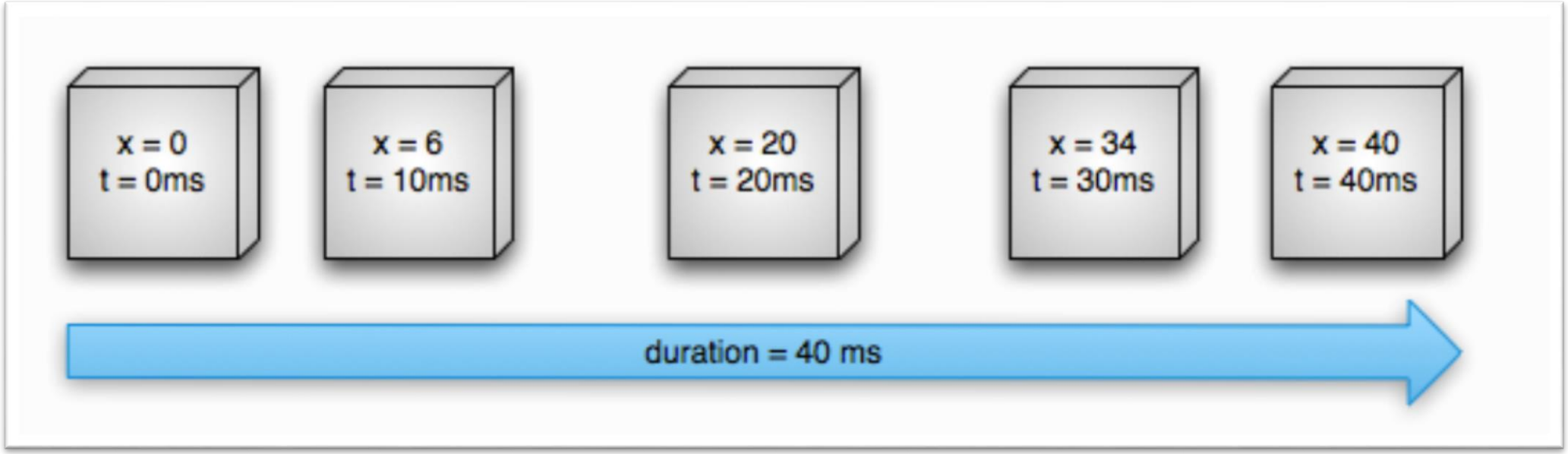
But first! How property animation works?

Linear (interpolator) animation



But first! How property animation works?

Non-Linear (interpolator) animation



Animation – View Animation – Defining it

```
<set android:shareInterpolator="false">
    <scale
        android:interpolator="@android:anim/accelerate_decelerate_interpolator"
        android:fromXScale="1.0"
        android:toXScale="1.4"
        android:fromYScale="1.0"
        android:toYScale="0.6"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fillAfter="false"
        android:duration="700" />
    <set android:interpolator="@android:anim/decelerate_interpolator"

<rotate
            android:fromDegrees="0"
            android:toDegrees="-45"
            android:toYScale="0.0"
            android:pivotX="50%"
            android:pivotY="50%"
            android:startOffset="700"
            android:duration="400" />
    </set>
</set>
```

Animation – View Animation – Using it

On views:

```
ImageView spaceshipImage = (ImageView) findViewById(R.id.spaceshipImage);
Animation hyperspaceJumpAnimation = AnimationUtils.loadAnimation(this, R.anim.hyperspace
spaceshipImage.startAnimation(hyperspaceJumpAnimation);
```

On activities:

```
overridePendingTransition(android.R.anim.fade_in, android.R.anim.fade_out);
```

Animation – Property Animation

There are three main classes used to programmatically execute property animation, those are:

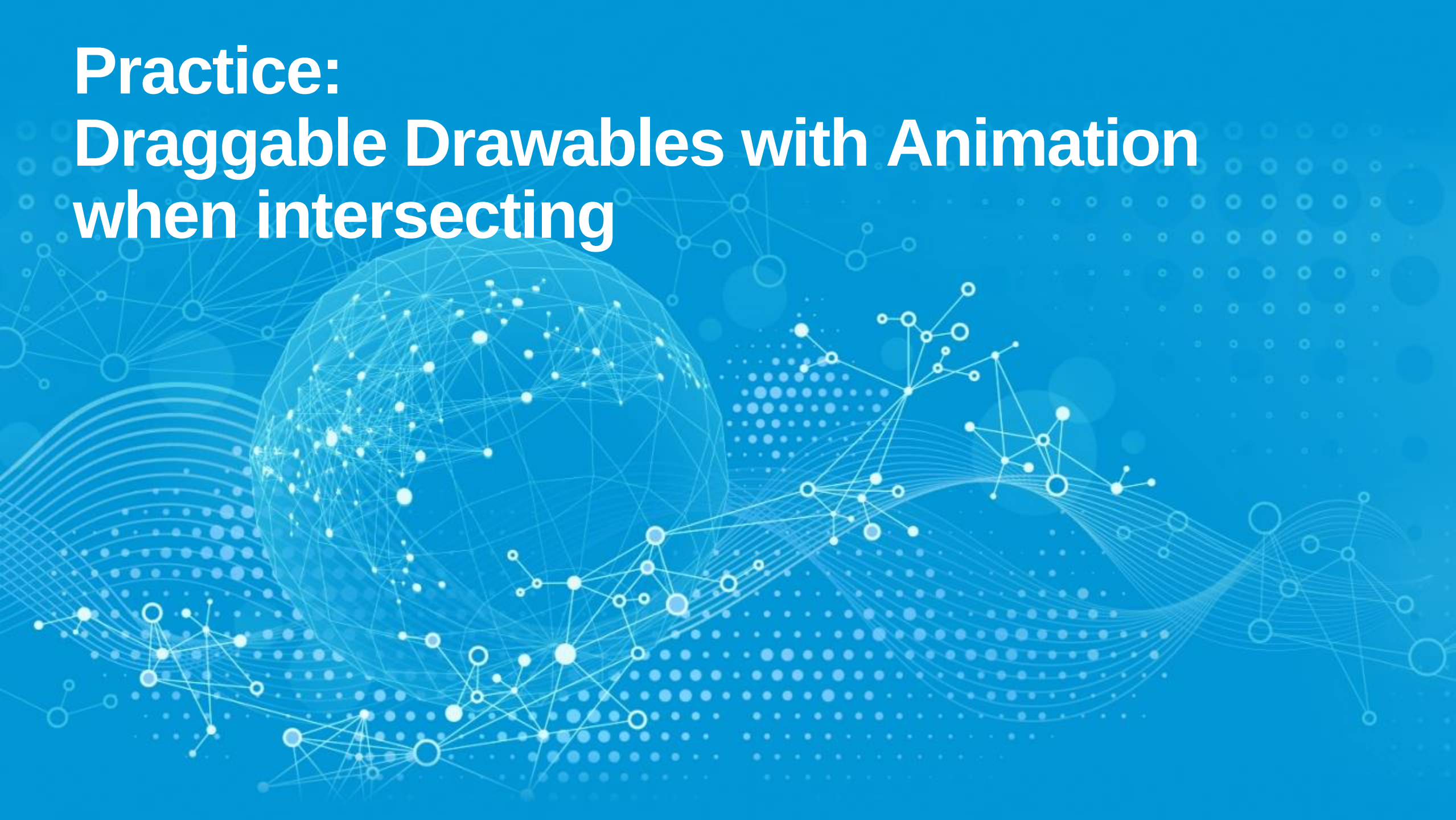
- ValueAnimator
- ObjectAnimator
- AnimatorSet

And this is how we use it:

```
ObjectAnimator anim = ObjectAnimator.ofFloat(foo, "alpha", 0f, 1f);
anim.setDuration(1000);
anim.start();
```

Practice:
Animated Chronometer

Practice: Draggable Drawables with Animation when intersecting



Thank you

Lucio Cossio
Luis Mazoni