Lab 13: Drawables, Styles, and Animations

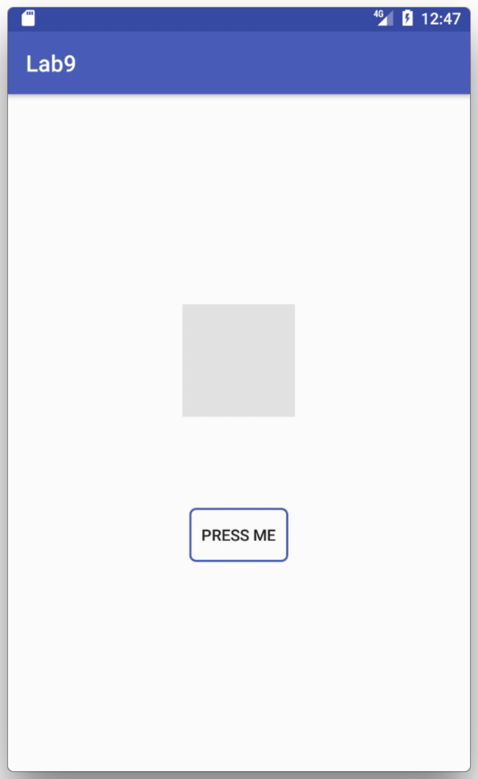
In this lab you will:

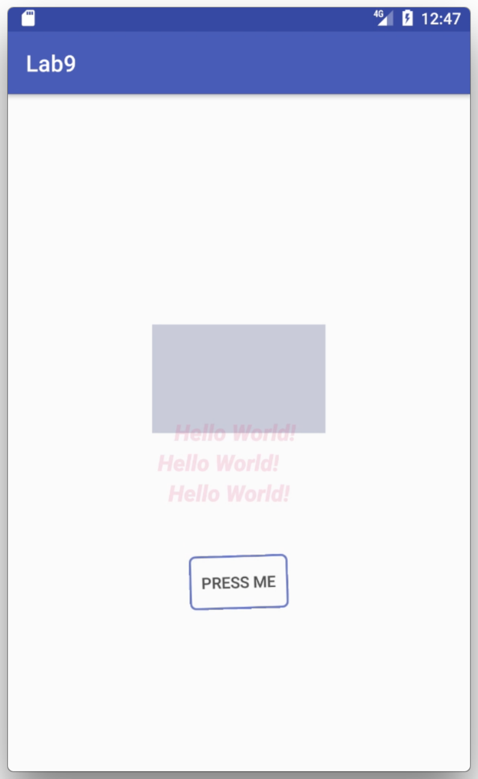
- Create a state-list drawable (out of two shape drawables) to customize a Button.
- Create a style for several TextViews.
- Animate several widgets on button press.

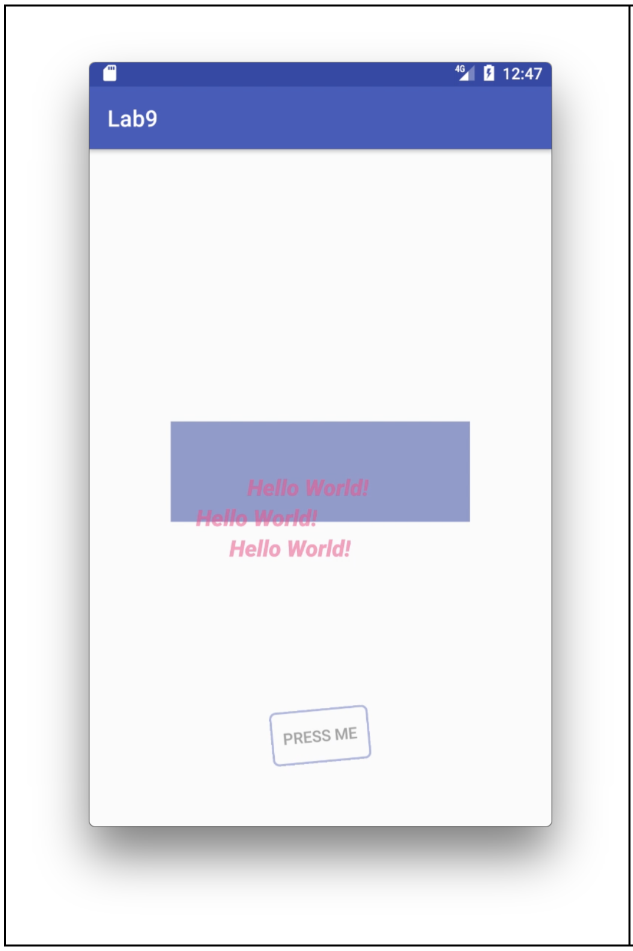
Result:

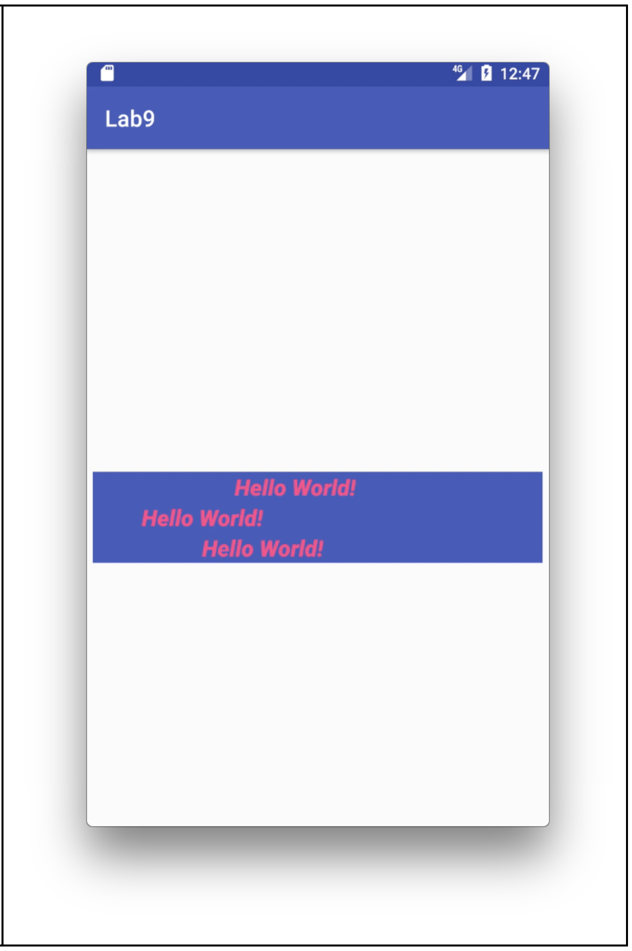
On tapping the button, this animation occurs, where

- The Button moves off the bottom of the screen, rotates slightly, and fades out.
- The gray square changes vertical position (to be aligned with the TextViews), changes width to fill the screen, and changes color.
- The TextViews fade in, and move to a random horizontal offset.









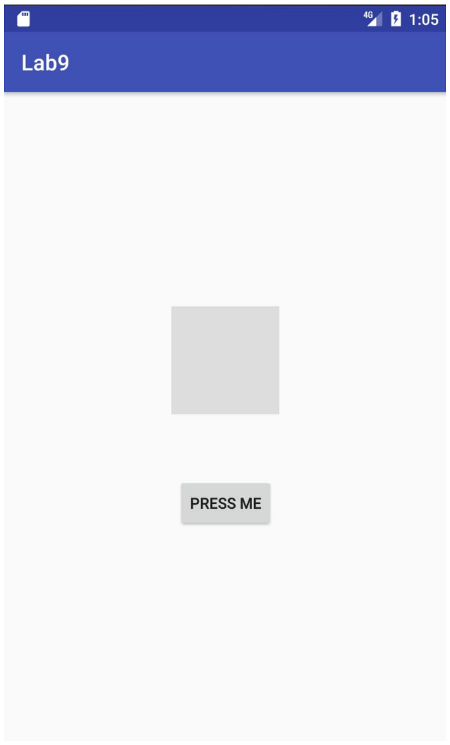
Steps

- Create a new project (with the Empty Activity template). Give it a package name `com.techexchange.mobileapps.lab13`.
- Go to colors.xml and add a new color called "shapeStartColor", i.e.
  `<color name="shapeStartColor">#DDDDDD</color>`
  (I've made it a light gray, but you can specify any color you wish!)
- We will cheat and not use a Fragment this time; all our widgets will be directly in the MainActivity. Open activity_main.xml and overwrite it with the attached code below (you may copy and paste this). Take a look at it, as there is a new attribute "alpha", signifiying the translucency of the view. Run the code, you should get this:

```
<?xml version="1.0" encoding="utf-8"?>
  <LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:id="@+id/rootLayout"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center">

    <View
        android:id="@+id/shapeView"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:background="@color/shapeStartColor"/>
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:alpha="0"/>
    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:alpha="0"/>
    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:alpha="0"/>
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="press me"/>
</LinearLayout>
```



(The text views are set to completely transparent)

Part 1, customize button with drawable

- Create a new drawable resource file named "button_background_default". Have it be a rectangular outline with rounded corners:

```
<shape xmlns:android="http://schemas.android.com/apk/res/android" android:shape="rectangle" >
    <stroke android:width="2dp" android:color="@color/colorPrimary"/>
    <corners android:radius="5dp"/>
</shape>
```

- Create another drawable resource file named "button_background_pressed". Have it be a similar shape, but with a thicker line and a different color:

```
<shape xmlns:android="http://schemas.android.com/apk/res/android" android:shape="rectangle" >
    <stroke android:width="5dp" android:color="@color/colorAccent"/>
    <corners android:radius="5dp"/>
</shape>
```

- Create yet another drawable resource file named "button_background". Instead of a single shape, this will be a list of references to other drawables, which will be used by a button as it changes state.

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/button_background_pressed"
          android:state_pressed="true"/>
    <item android:drawable="@drawable/button_background_default"/>
</selector>
```

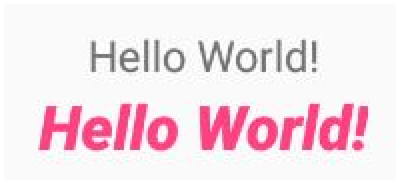
- Go back to activity_main.xml, and set the background attribute of the button. Add this line within the button widget:

```
android:background="@drawable/button_background"
```

- Run the app, your button should now look like the button in the mock, with a rounded border (instead of a solid gray rectangle). Clicking the button should turn the border thicker and pink.

Part 2, use shared style on TextViews

- Rather than drab gray small text, we want our TextViews to have more...well, style. Here's before and after:



Set the alpha to one of the TextViews to "1" temporariliy, so that it is visible. Add the following attributes to it:

```
    android:textSize="20dp"
   android:textColor="@color/colorAccent"
   android:textStyle="bold|italic"
   android:fontFamily="sans-serif-medium"
```

- This is what we want, but we don't want to have to paste these same 4 lines on every TextView. Remove these lines that you just added. Instead, we will put these into a "style" element. Open styles.xml and add this within the "resources" tag (i.e. a sibling of the existing style). Note the different syntax.

```
<style name="lab9TextStyle">
        <item name="android:textSize">20dp</item>
        <item name="android:textColor">@color/colorAccent</item>
        <item name="android:textStyle">bold|italic</item>
        <item name="android:fontFamily">sans-serif-medium</item>
</style>
```

- Now return to activity_main.xml and add this line to each TextView:

```
style="@style/lab9TextStyle"
```

Turn the alphas to "1" temporarily so that you can confirm that this style is applied to each TextView.

Part 3, Animate!

What seems like a simple animation turns out to have many things going on. On button press:

- The gray shape changes position, shape, and color.
- The three TextViews change alpha (to fade in) and horizontal position.
- The Button changes position, alpha (to fade out) and rotation.

Every one of these changes will be controlled by a property animation. (In case you look online for examples, note that there are other types of animations in the Android world, e.g. ViewAnimations, etc., which are a different pattern). Property animations basically call a particular method on a view many times in succession, with a gradually changing value. The general pattern for them looks like this

```
ObjectAnimator fooAnimator =
    ObjectAnimator.ofFloat(view, "parameter", fromValue, toValue).setDuration(timeInMillis);
fooAnimator.start();
```

Note the use of a factory pattern to vend an animator of your chosen type, with the desired parameters. Also note an unusual pattern, where the method to call is in quotes. Whatever is in quotes (e.g. "foo") will be called as a setter on the view (e.g. the animator will repeatedly call view.setFoo()).

- Go to MainActivity.java declare instance variable for 5 View objects (rootView, textView1,2, and 3, shapreView, an array of Views (List<TextView>), and a Button object. and set the contentView to the main activity.
- Within onCreate() get a reference to the top-level layout and all widgets. Also note that for convenience sake, we pack the three TextViews into a three-element List<TextView>. Wire up a listener for the Button to invoke *onClickAnimate()*

```
rootView = findViewById(R.id.rootLayout);
textView1 = findViewById(R.id.textView1);
textView2 = findViewById(R.id.textView2);
textView3 = findViewById(R.id.textView3);
textViews = Arrays.asList(textView1, textView2, textView3);
shapeView = findViewById(R.id.shapeView);
button = (Button) findViewById(R.id.button);

button.setOnClickListener(this::onClickAnimate);
```

- In the onClickAnimate() method, add ObjectAnimators for all of these properties. Note the type of animator (float, int, or ARGB for color).
- Coordinates are ints because they are actual pixels on the screen; note that we avoid hardcoding any actual pixel/coordinate values (since that would look different on different screen sizes/resolutions), and instead measure the size/position of other views and use those as a value.

```
private void onClickAnimate(View view) {
  int duration = 3000;
  // Animate the shape width, top, bottom, and color.
  float scaleToFillWidth = (float) rootView.getWidth() / shapeView.getWidth();

  ObjectAnimator shape2ScaleXAnimator =
        ObjectAnimator.ofFloat(shapeView, "scaleX", 1f,
scaleToFillWidth).setDuration(duration);
  shape2ScaleXAnimator.start();

  // Set the top to be the top of the first TextView.
  ObjectAnimator shapeTopAnimator =
        ObjectAnimator.ofInt(shapeView, "top", shapeView.getTop(),
textView1.getTop()).setDuration(duration);
  shapeTopAnimator.start();

  // Set the bottom to be the bottom of the third TextView.
  ObjectAnimator shapeBottomAnimator =
        ObjectAnimator.ofInt(shapeView, "bottom", shapeView.getBottom(),
textView3.getBottom()).setDuration(duration);
  shapeBottomAnimator.start();

  int startColor = ContextCompat.getColor(MainActivity.this, R.color.shapeStartColor);
  int endColor = ContextCompat.getColor(MainActivity.this, R.color.colorPrimary);
  ObjectAnimator shape2ColorAnimator =
        ObjectAnimator.ofArgb(shapeView, "backgroundColor", startColor,
endColor).setDuration(duration);
  shape2ColorAnimator.start();

  // Loop through the TextViews, set their alpha and X position (as a random offset).
  for (View v : textViews) {
     ObjectAnimator alphaAnimator =
           ObjectAnimator.ofFloat(v, "alpha", 0f, 1f).setDuration(duration);
     alphaAnimator.start();
```

```
        // Get a random value between negative one-half-screen-width, and positive
one-half-screen-width.
        // Set that as the x position of the TextView.
        float randomOffset = (float) (Math.random() * rootView.getWidth()) - rootView.getWidth() /
2;
        ObjectAnimator xAnimator = ObjectAnimator.ofFloat(v, "x", v.getX(), v.getX() +
randomOffset).setDuration(duration);
        xAnimator.start();


        // Animate Button's Y position, rotation, and alpha.
        // Set button y position to be a random offset from the bottom of the screen.
        ObjectAnimator buttonYAnimator =
                ObjectAnimator.ofFloat(button, "y", button.getY(), rootView.getBottom() +
randomOffset).setDuration(duration);
        buttonYAnimator.start();


        float randomRotation = (float) (Math.random() * 40) - 20;
        ObjectAnimator rotationAnimator =
                ObjectAnimator.ofFloat(button, "rotation", 0f, randomRotation).setDuration(duration);
        rotationAnimator.start();

        alphaAnimator =
                ObjectAnimator.ofFloat(button, "alpha", 1f, 0f).setDuration(duration);
        alphaAnimator.start();
    }
}
```

Play around with manipulating other properties during the animation on your own.