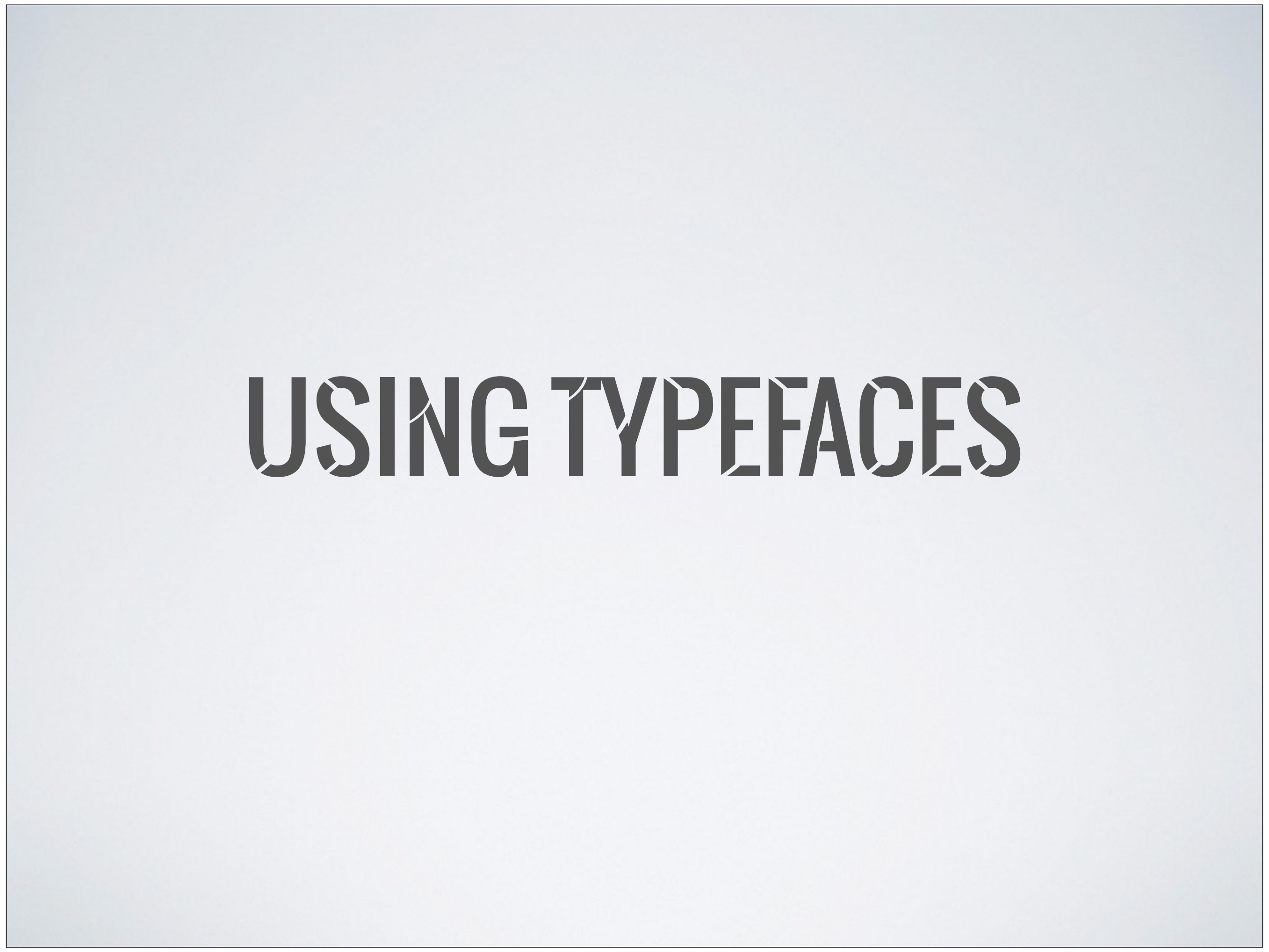


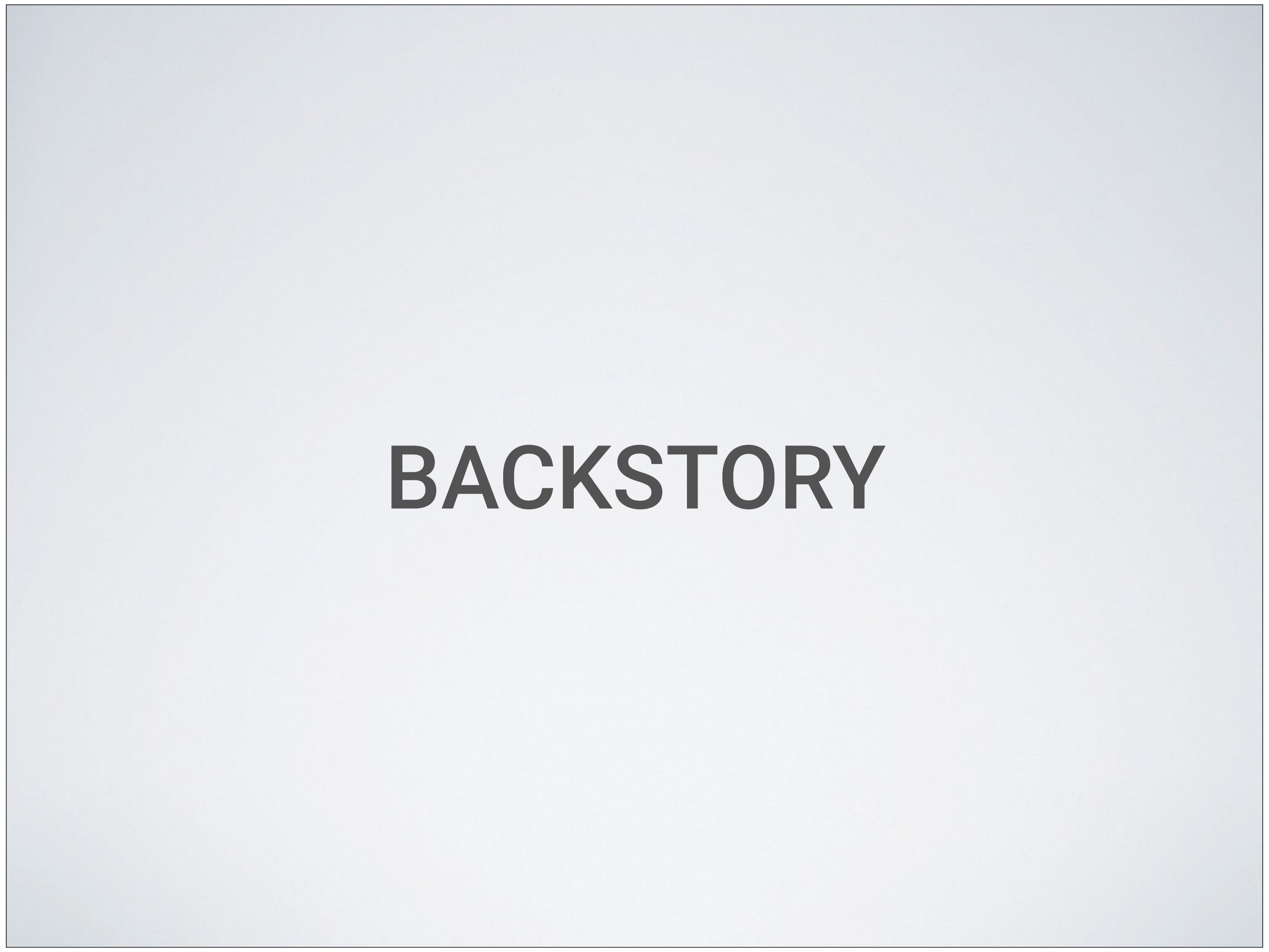
USING TYPEFACES

Thank you Screen Interaction and Stockholm meet up for having me.

SHAMELESS PLUG

- Work/Live in London
- Android Dev 5+ Years
- Current: OWLR - Best IP Camera Software Viewer
- Built - YOYO, LoveFlutter, Argos, OnTrack, MetOffice, SunGoals, blah blah blah…

@chrisjenx

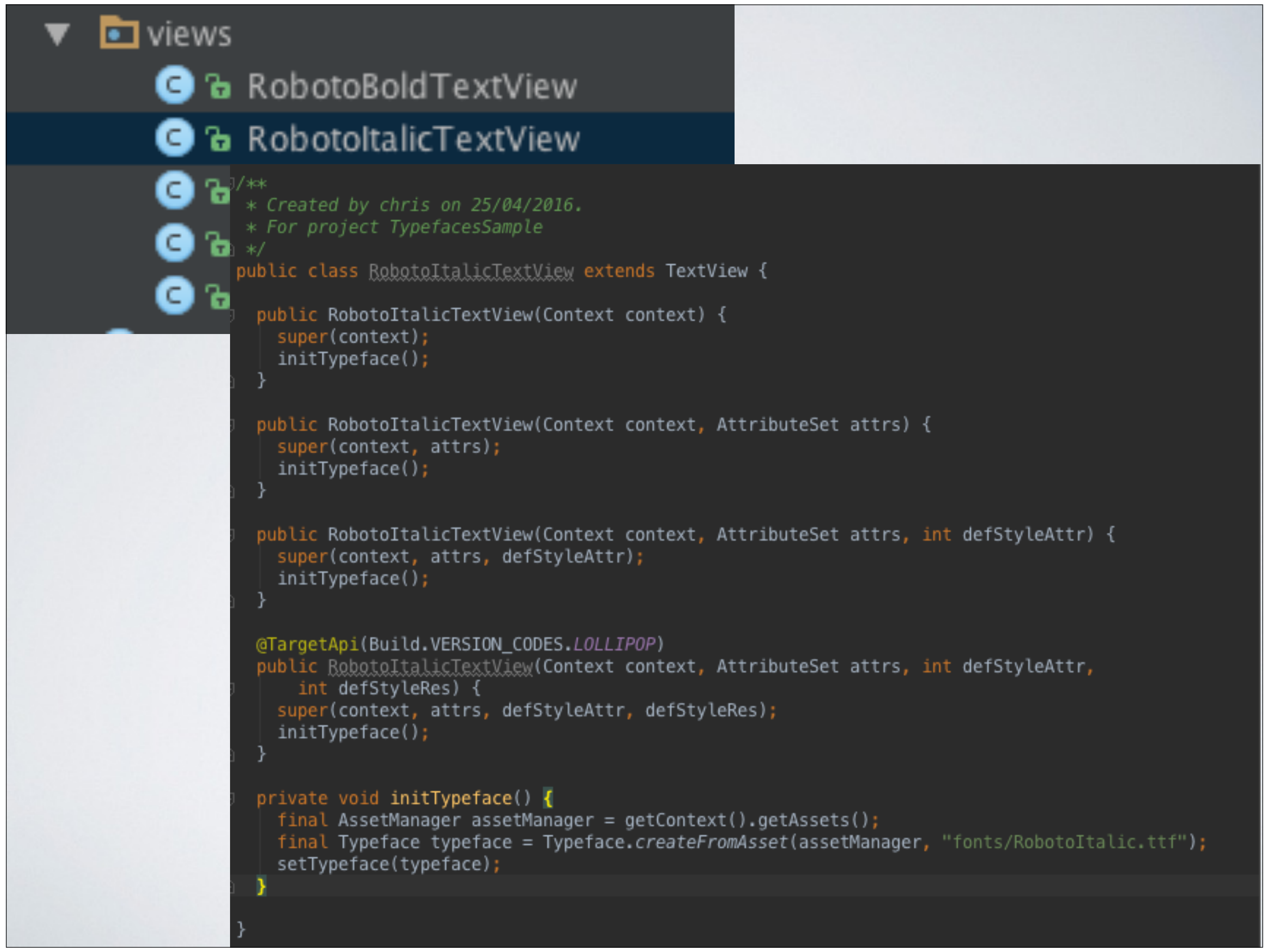


BACKSTORY

So bit of a back story, tried to ‘storify' this as lets be honest typefaces aren’t the most thrilling of topics.

mubaloo®
enterprise mobility

Mubaloo 2011, Fresh Android Developer, maybe built 10+ apps.



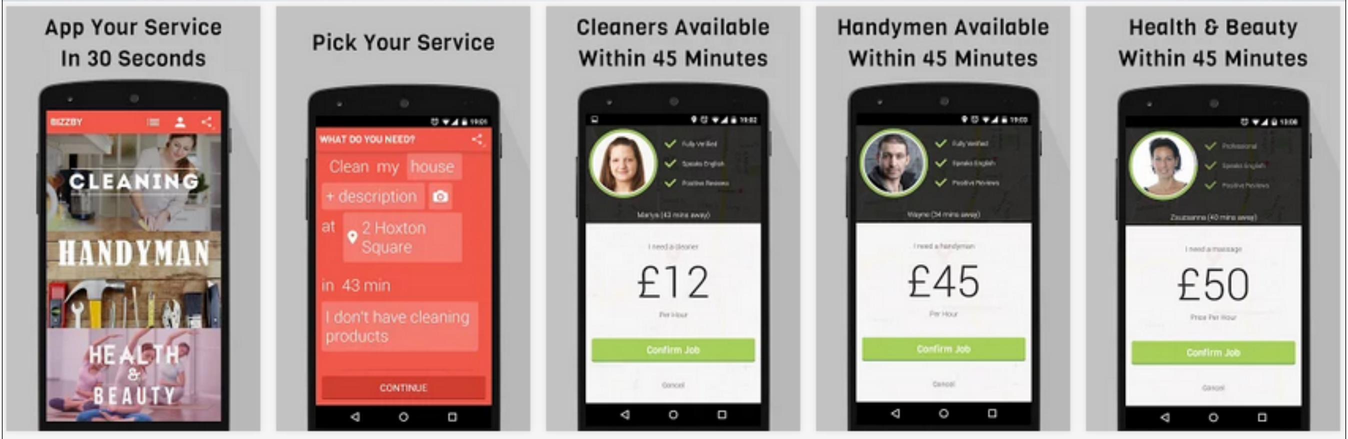
```java
/**
 * Created by chris on 25/04/2016.
 * For project TypefacesSample
 */
public class RobotoItalicTextView extends TextView {

  public RobotoItalicTextView(Context context) {
    super(context);
    initTypeface();
  }

  public RobotoItalicTextView(Context context, AttributeSet attrs) {
    super(context, attrs);
    initTypeface();
  }

  public RobotoItalicTextView(Context context, AttributeSet attrs, int defStyleAttr) {
    super(context, attrs, defStyleAttr);
    initTypeface();
  }

  @TargetApi(Build.VERSION_CODES.LOLLIPOP)
  public RobotoItalicTextView(Context context, AttributeSet attrs, int defStyleAttr,
      int defStyleRes) {
    super(context, attrs, defStyleAttr, defStyleRes);
    initTypeface();
  }

  private void initTypeface() {
    final AssetManager assetManager = getContext().getAssets();
    final Typeface typeface = Typeface.createFromAsset(assetManager, "fonts/RobotoItalic.ttf");
    setTypeface(typeface);
  }

}
```

Started to notice this in allot of projects.
Didn’t think any of that back then.
Nothing inherently wrong with this, but doesn’t scale…

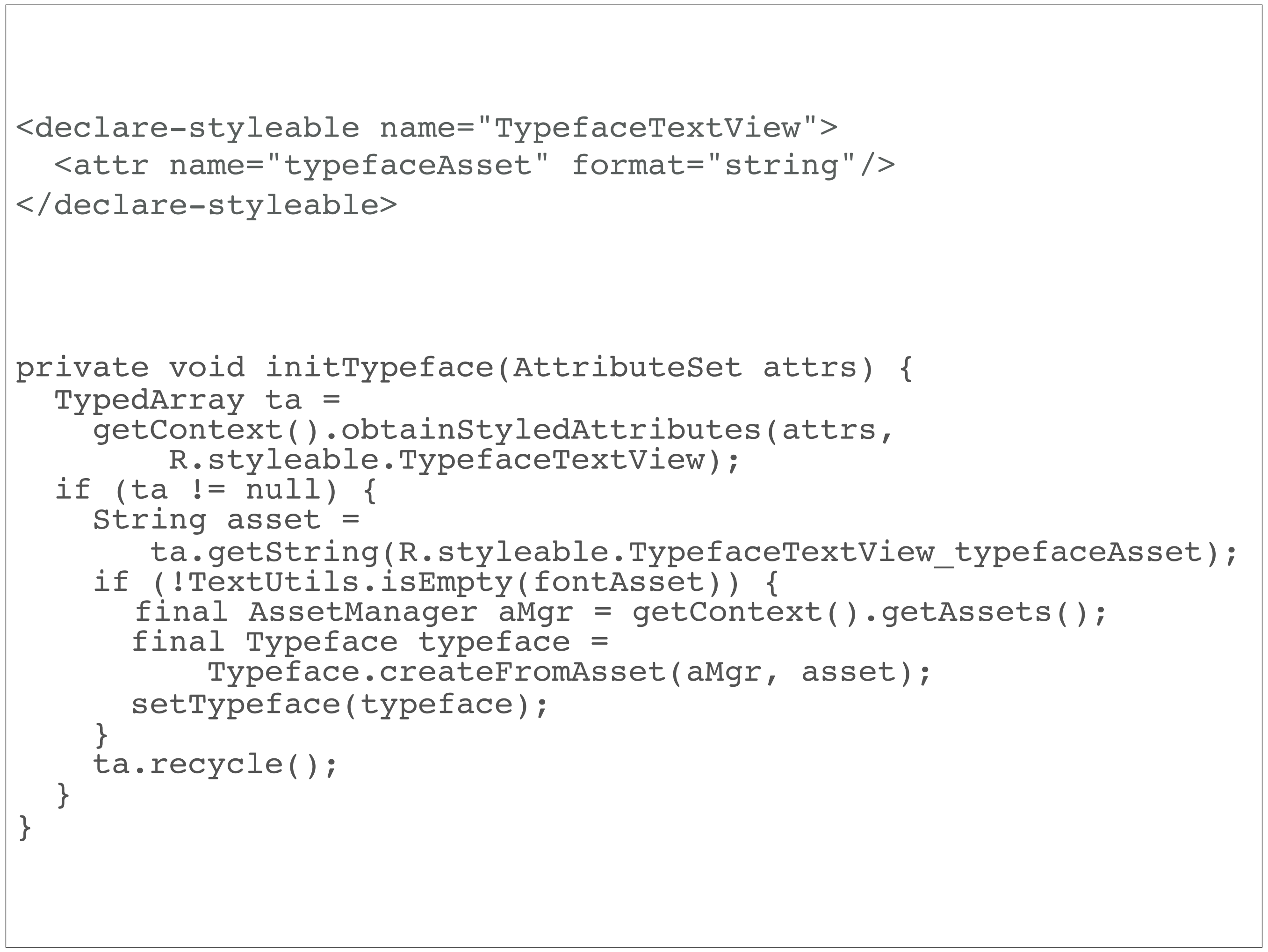
Recursive:

```
private void iterateViews(ViewGroup group, Typeface typeface) {
  for (int i = 0; i < group.getChildCount(); i++) {
    final View view = group.getChildAt(i);
    if (view instanceof ViewGroup) {
      iterateViews((ViewGroup) view, typeface); continue;
    }
    if (view instanceof TextView) {
      ((TextView) view).setTypeface(typeface);
    }
  }
}
```

```
final Typeface typeface =
    TypefaceUtils.load(getAssets(), "fonts/Roboto-Bold.ttf");
final View view = getWindow().getDecorView();
iterateViews((ViewGroup) view, typeface);
```


App Your Service In 30 Seconds
Pick Your Service
Cleaners Available Within 45 Minutes
Handymen Available Within 45 Minutes
Health & Beauty Within 45 Minutes
CLEANING
HANDYMAN
HEALTH & BEAUTY
Clean my house
+ description
at 2 Hoxton Square
in 43 min
I don't have cleaning products
CONTINUE
£12
£45
£50
Confirm Job

2013 moved to London to join Bizzby.
Everything was new/exciting. We were going to change the world!
Key thing: I met Dana Nedamaldeen - Design and Product mattered.
We spent so much time on the product - it never launched (true story)

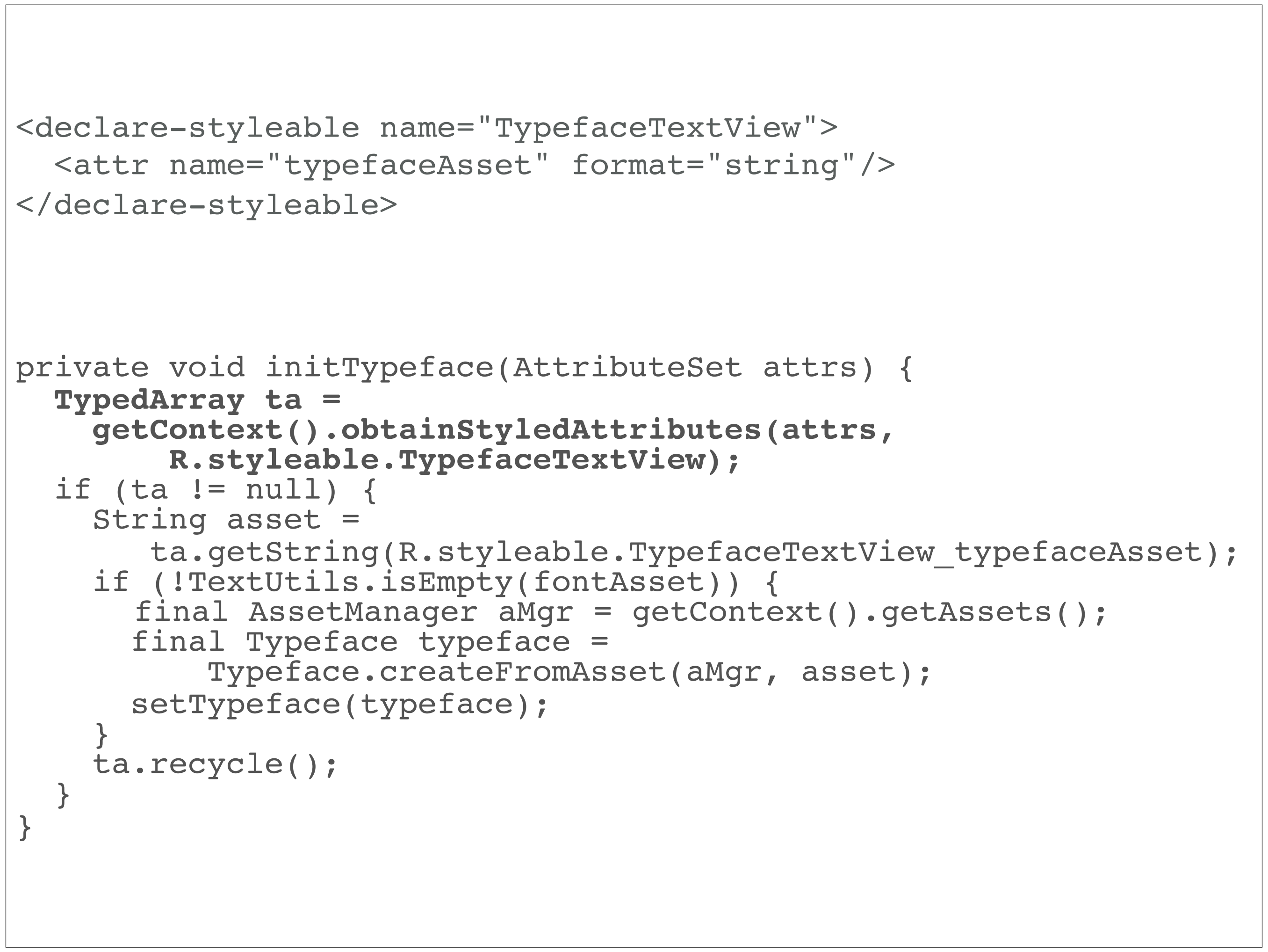


```
<declare-styleable name="TypefaceTextView">
  <attr name="typefaceAsset" format="string"/>
</declare-styleable>
```

```
private void initTypeface(AttributeSet attrs) {
  TypedArray ta =
    getContext().obtainStyledAttributes(attrs,
        R.styleable.TypefaceTextView);
  if (ta != null) {
    String asset =
       ta.getString(R.styleable.TypefaceTextView_typefaceAsset);
    if (!TextUtils.isEmpty(fontAsset)) {
      final AssetManager aMgr = getContext().getAssets();
      final Typeface typeface =
          Typeface.createFromAsset(aMgr, asset);
      setTypeface(typeface);
    }
    ta.recycle();
  }
}
```

Something better…

You’ve probably done something similar to this yourselves.

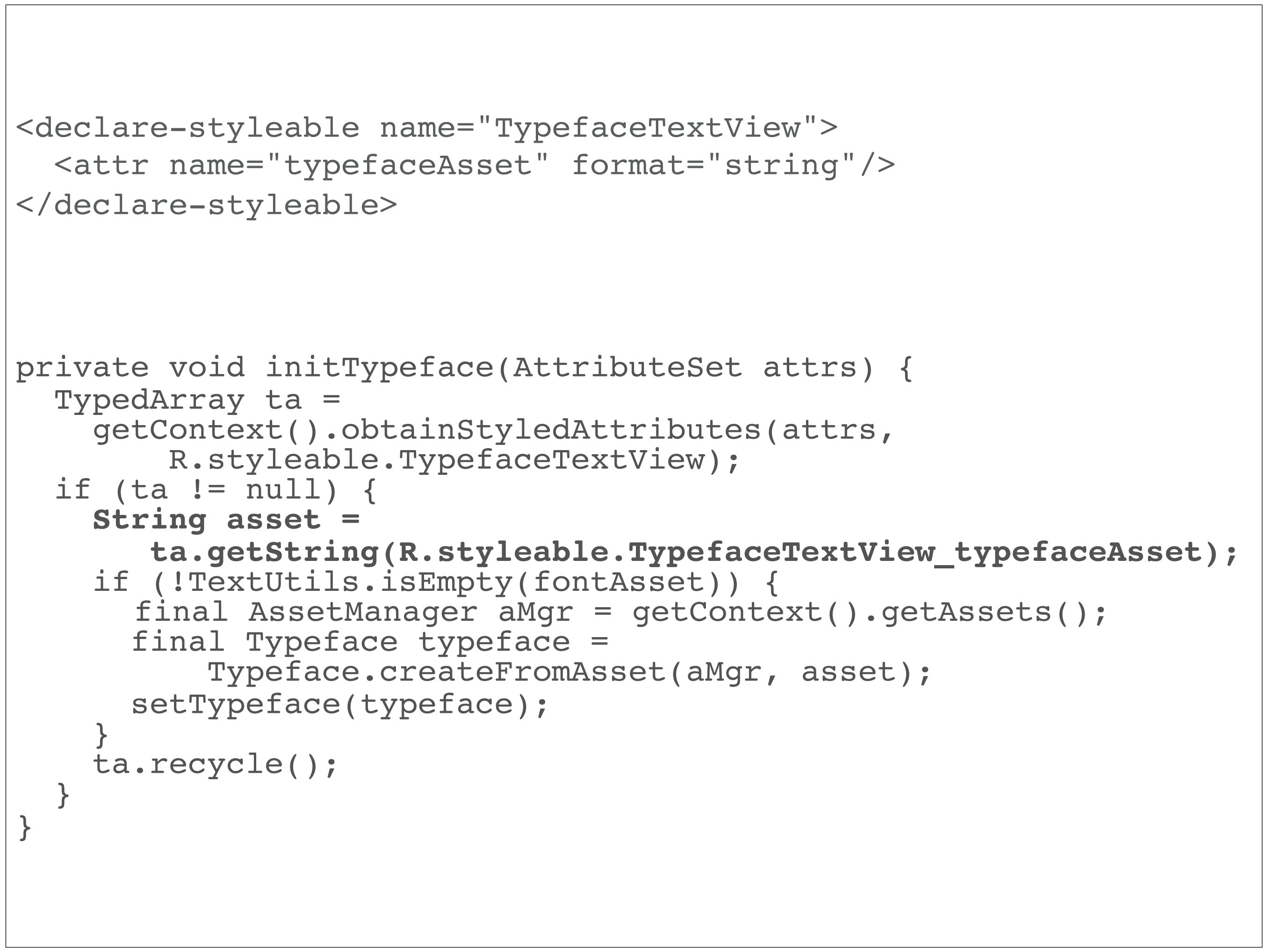


```
<declare-styleable name="TypefaceTextView">
  <attr name="typefaceAsset" format="string"/>
</declare-styleable>
```

```
private void initTypeface(AttributeSet attrs) {
  TypedArray ta =
    getContext().obtainStyledAttributes(attrs,
        R.styleable.TypefaceTextView);
  if (ta != null) {
    String asset =
       ta.getString(R.styleable.TypefaceTextView_typefaceAsset);
    if (!TextUtils.isEmpty(fontAsset)) {
      final AssetManager aMgr = getContext().getAssets();
      final Typeface typeface =
          Typeface.createFromAsset(aMgr, asset);
      setTypeface(typeface);
    }
    ta.recycle();
  }
}
```

Something better…

You’ve probably done something similar to this yourselves.

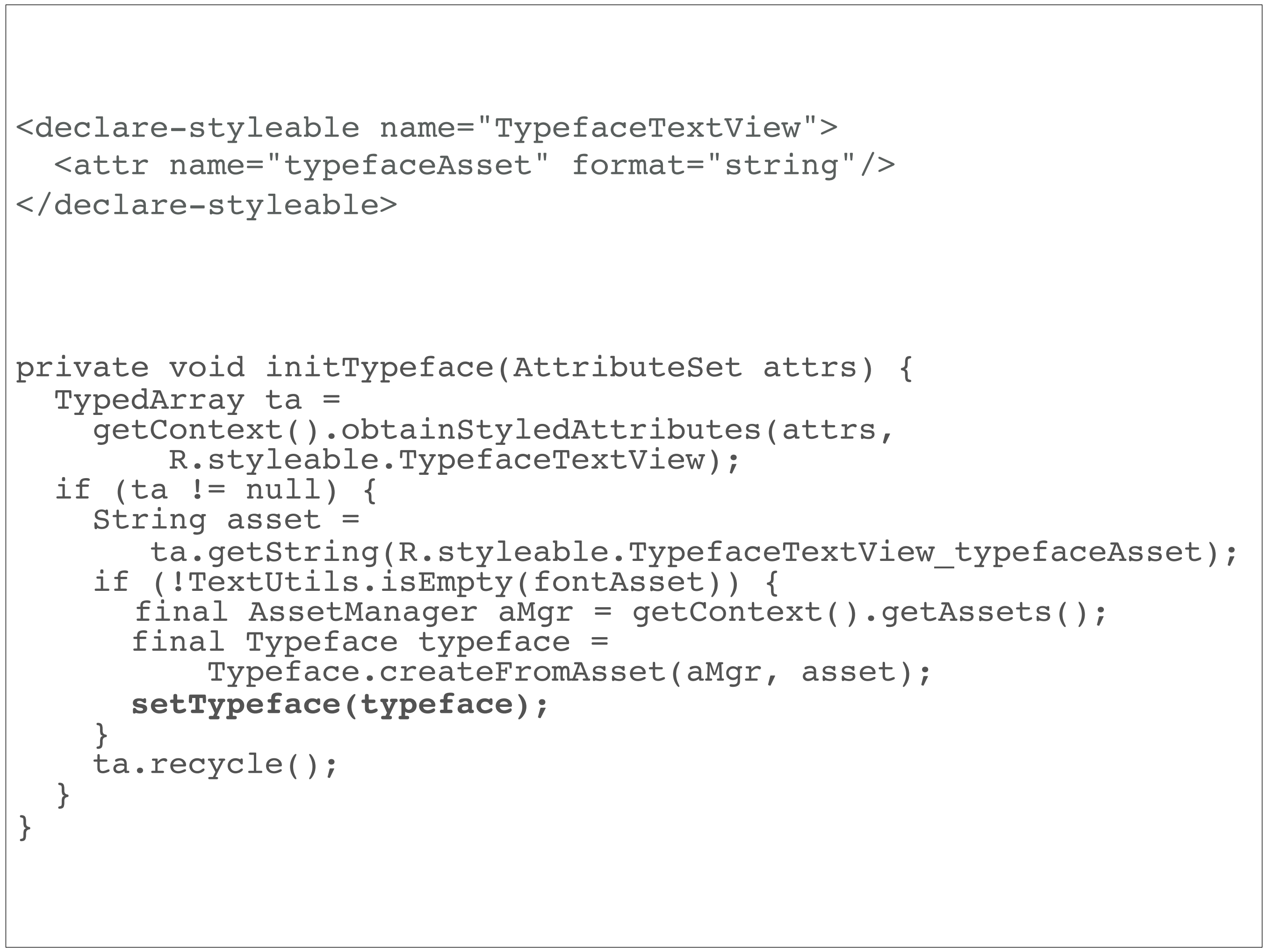


```
<declare-styleable name="TypefaceTextView">
  <attr name="typefaceAsset" format="string"/>
</declare-styleable>
```

```
private void initTypeface(AttributeSet attrs) {
  TypedArray ta =
    getContext().obtainStyledAttributes(attrs,
        R.styleable.TypefaceTextView);
  if (ta != null) {
    String asset =
       ta.getString(R.styleable.TypefaceTextView_typefaceAsset);
    if (!TextUtils.isEmpty(fontAsset)) {
      final AssetManager aMgr = getContext().getAssets();
      final Typeface typeface =
          Typeface.createFromAsset(aMgr, asset);
      setTypeface(typeface);
    }
    ta.recycle();
  }
}
```

Something better…

You've probably done something similar to this yourselves.



```
<declare-styleable name="TypefaceTextView">
  <attr name="typefaceAsset" format="string"/>
</declare-styleable>
```

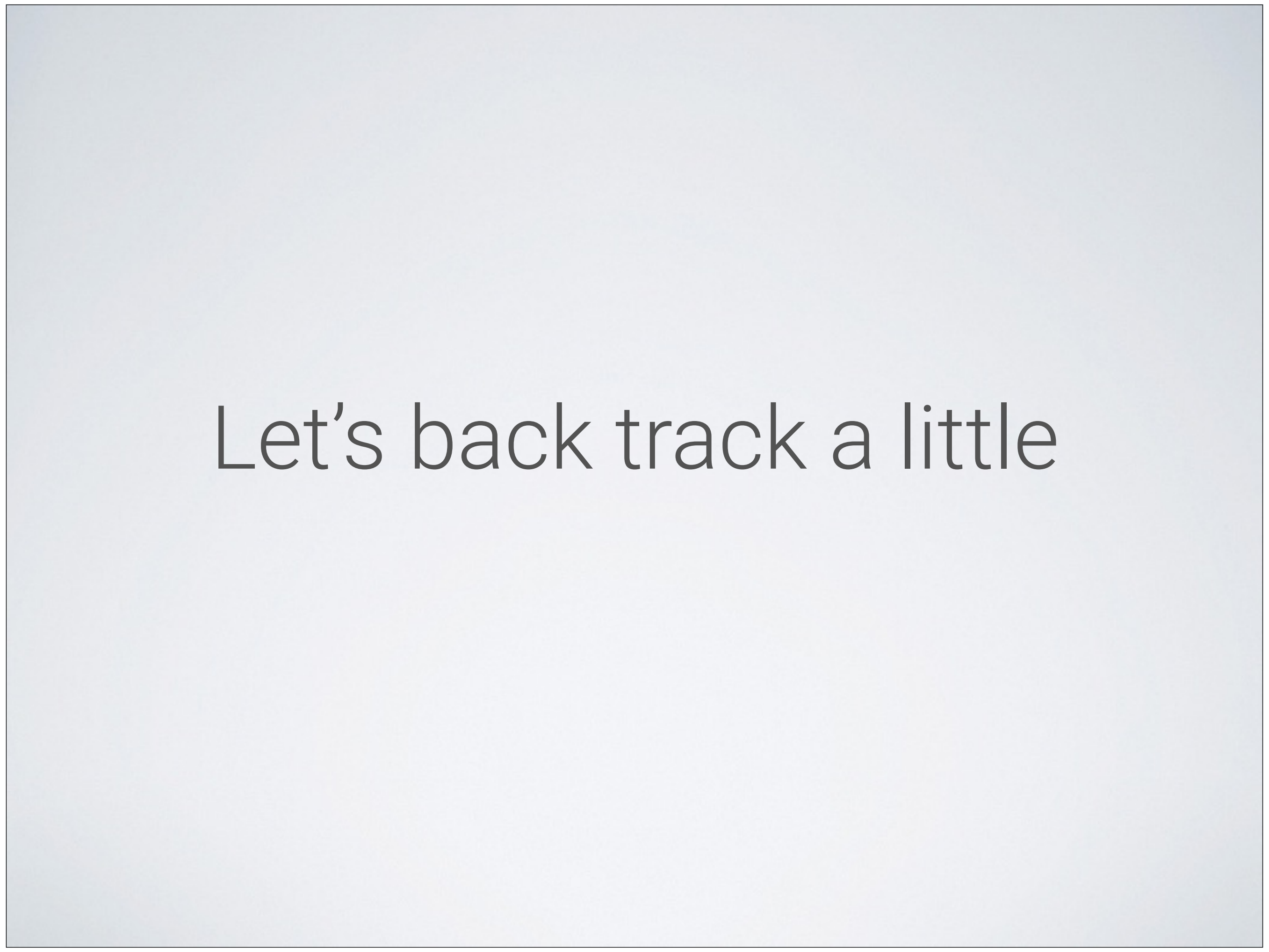
```
private void initTypeface(AttributeSet attrs) {
  TypedArray ta =
    getContext().obtainStyledAttributes(attrs,
        R.styleable.TypefaceTextView);
  if (ta != null) {
    String asset =
       ta.getString(R.styleable.TypefaceTextView_typefaceAsset);
    if (!TextUtils.isEmpty(fontAsset)) {
      final AssetManager aMgr = getContext().getAssets();
      final Typeface typeface =
          Typeface.createFromAsset(aMgr, asset);
      setTypeface(typeface);
    }
    ta.recycle();
  }
}
```

Something better…

You’ve probably done something similar to this yourselves.

```
<typefacessample.TypefaceTextView
      android:text="Hello World!”
      //…
      app:typefaceAsset="Roboto-Bold.ttf"/>
```

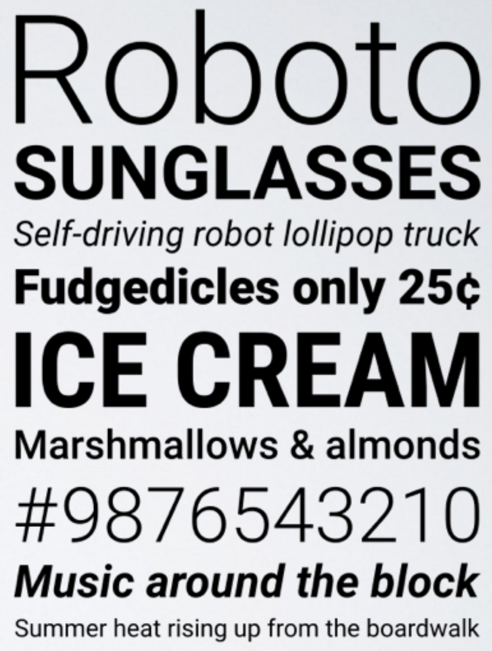
Timelines diverge.



Let’s back track a little

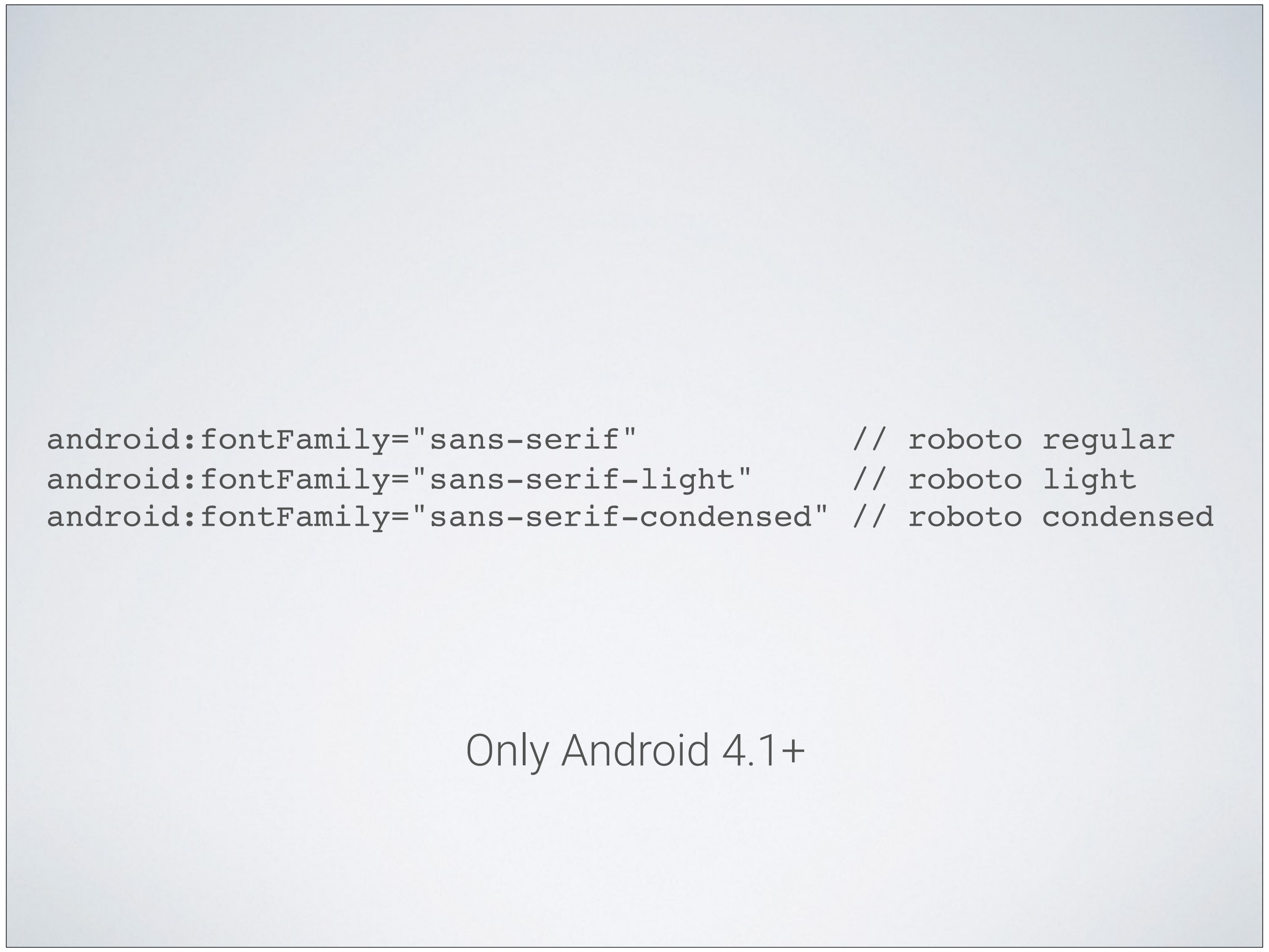
All the way back to 2011

- Introduced as default font in Android 4.0 (Holo)
- First standard font-family in Android



Roboto
SUNGLASSES
Self-driving robot lollipop truck
Fudgedicles only 25¢
ICE CREAM
Marshmallows & almonds
#9876543210
Music around the block
Summer heat rising up from the boardwalk

Roboto introduced in 2011~
Great a standard typeface.

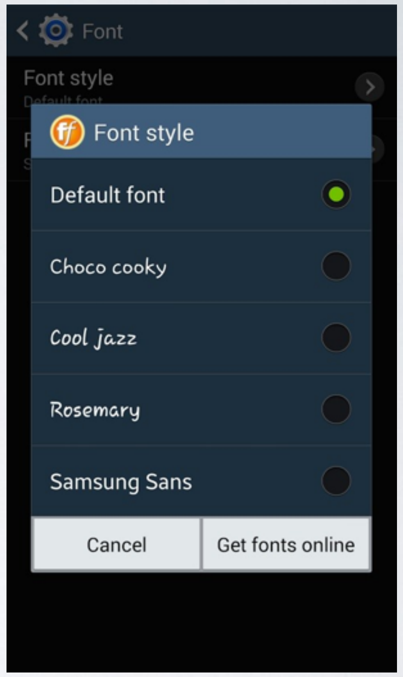


```
android:fontFamily="sans-serif"           // roboto regular
android:fontFamily="sans-serif-light"     // roboto light
android:fontFamily="sans-serif-condensed" // roboto condensed
```

Only Android 4.1+

We can use it like this!
But this only on Android 4+

- Samsung - Can change font. Some devices only use Samsung Sans
- LG - Similar issue use can change font
- Other manufacturers inconsistent


Font

Font style

Default font

Font style

Default font [✓]

Choco cooky

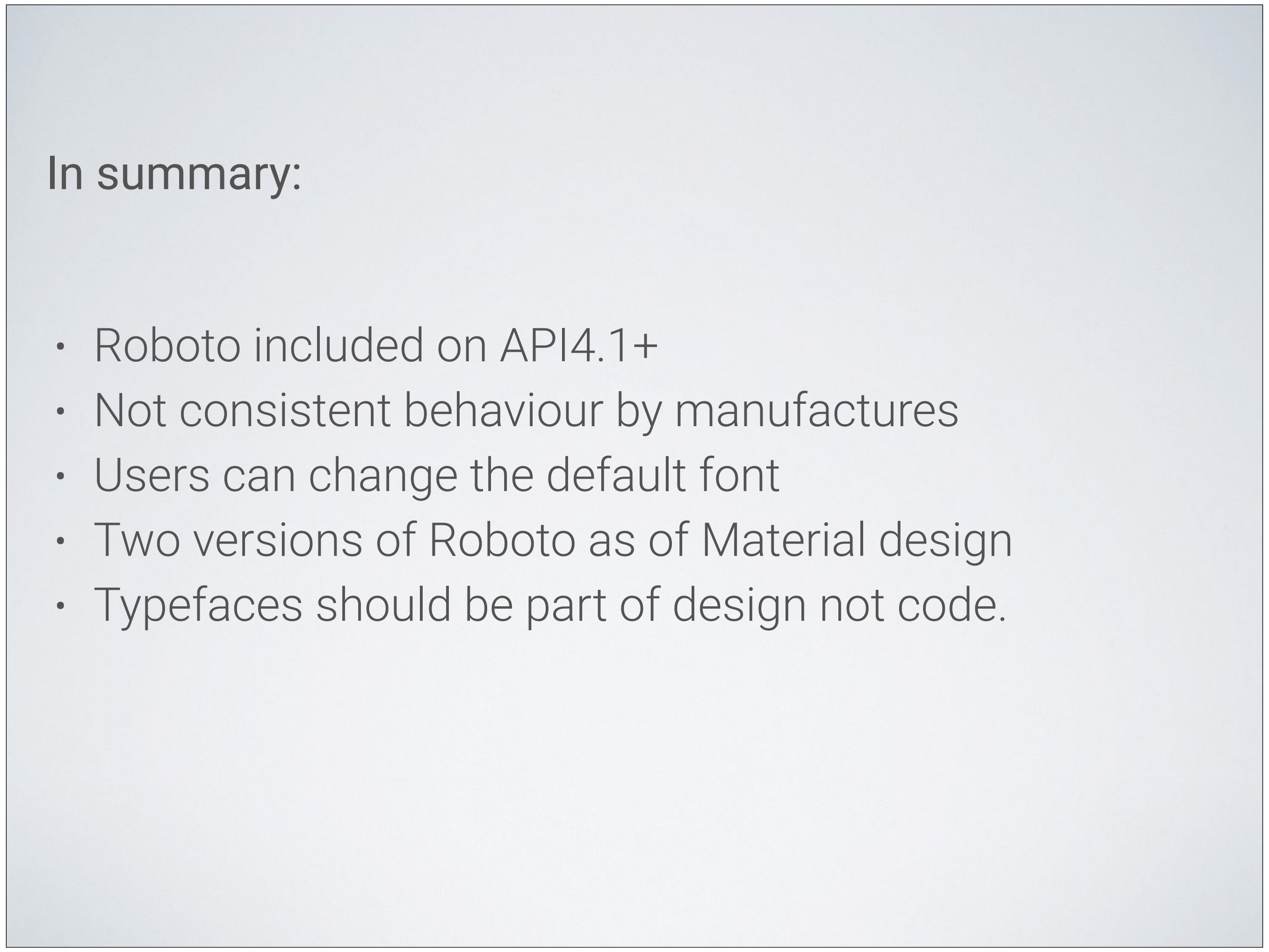
Cool jazz

Rosemary

Samsung Sans

Cancel | Get fonts online

Then manufactures go and mess this up!

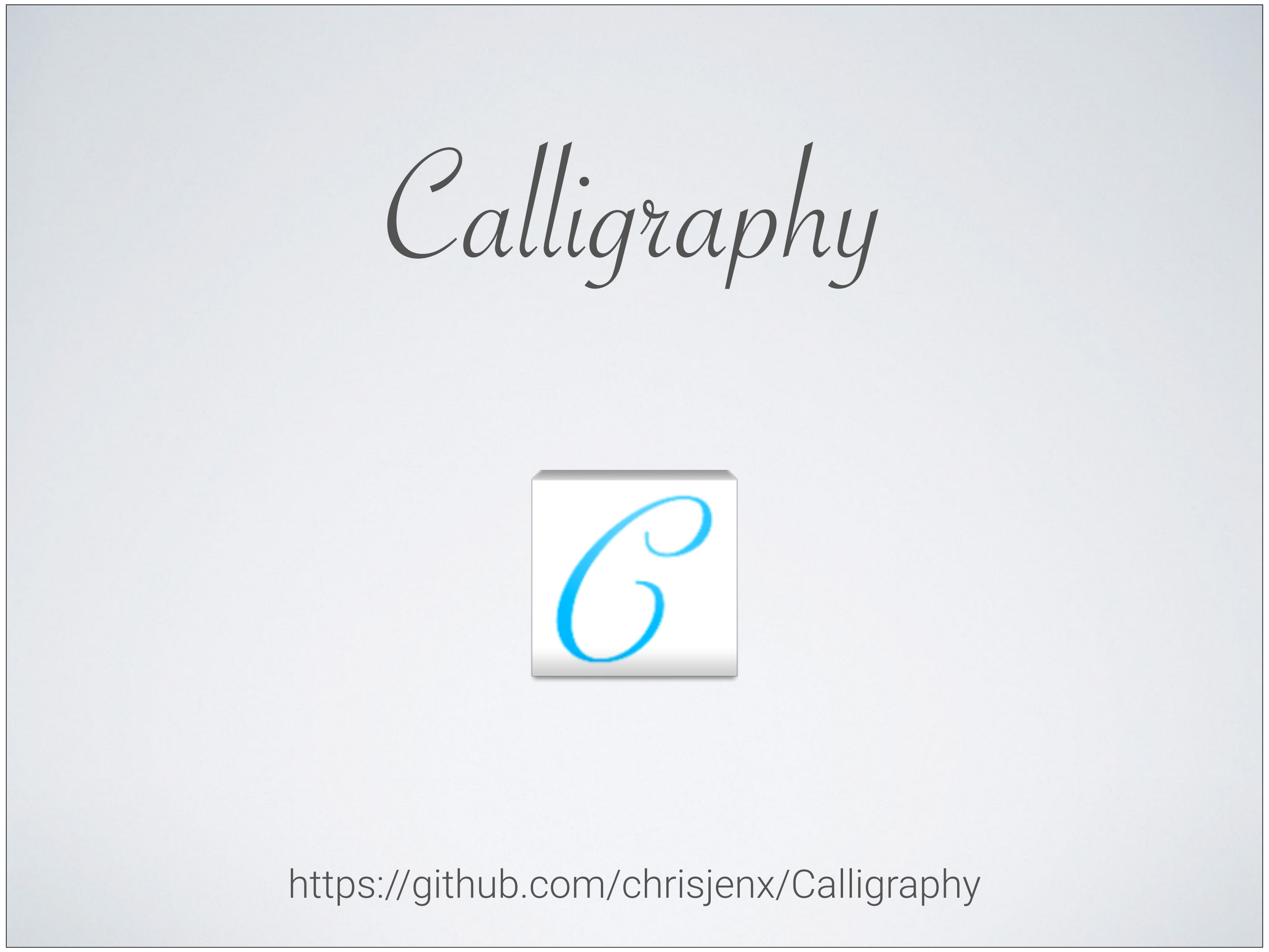


In summary:

- Roboto included on API4.1+
- Not consistent behaviour by manufactures
- Users can change the default font
- Two versions of Roboto as of Material design
- Typefaces should be part of design not code.

In summary

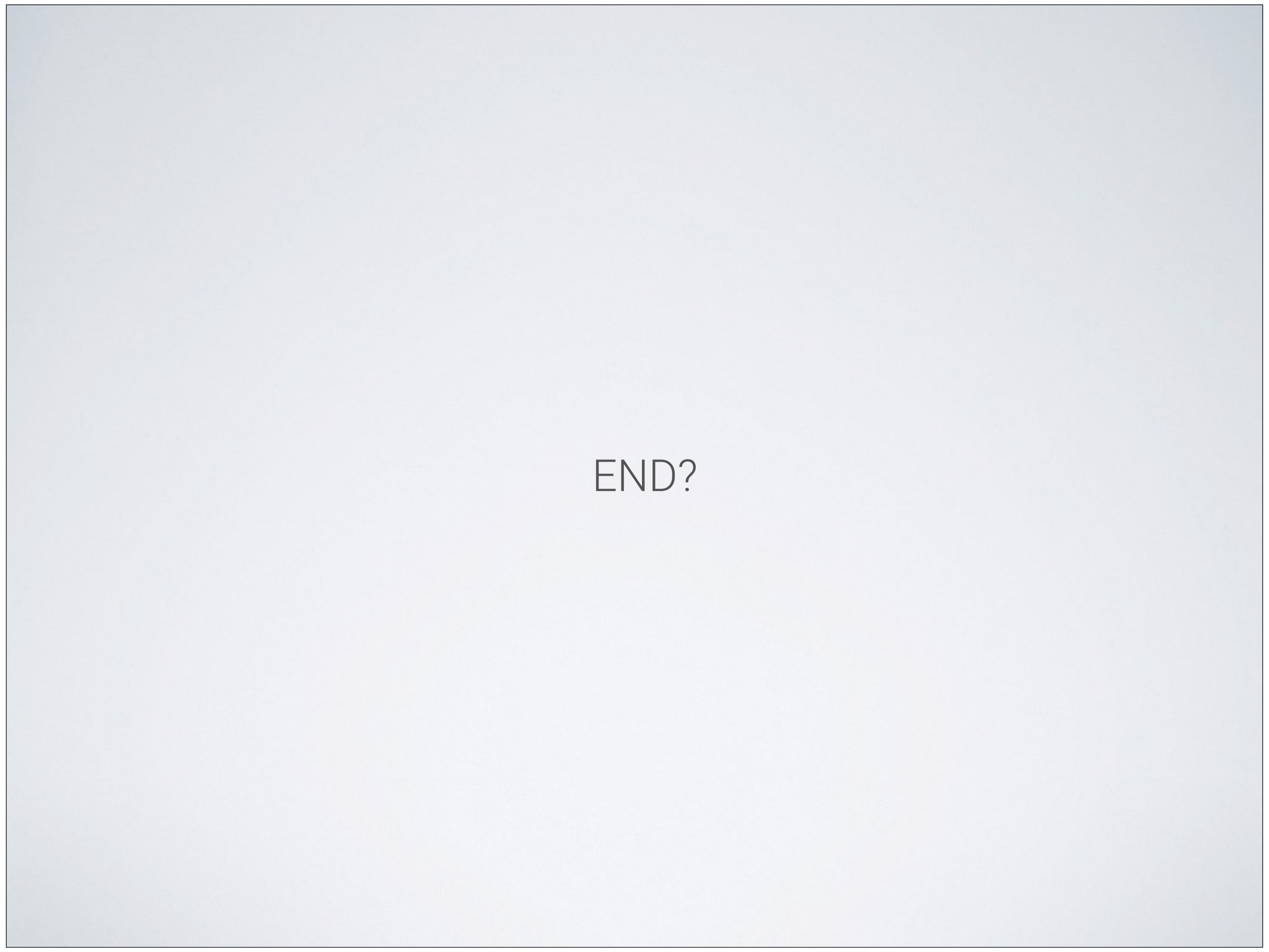
I went home and hacked something together in a few hours! Boom.



Calligraphy

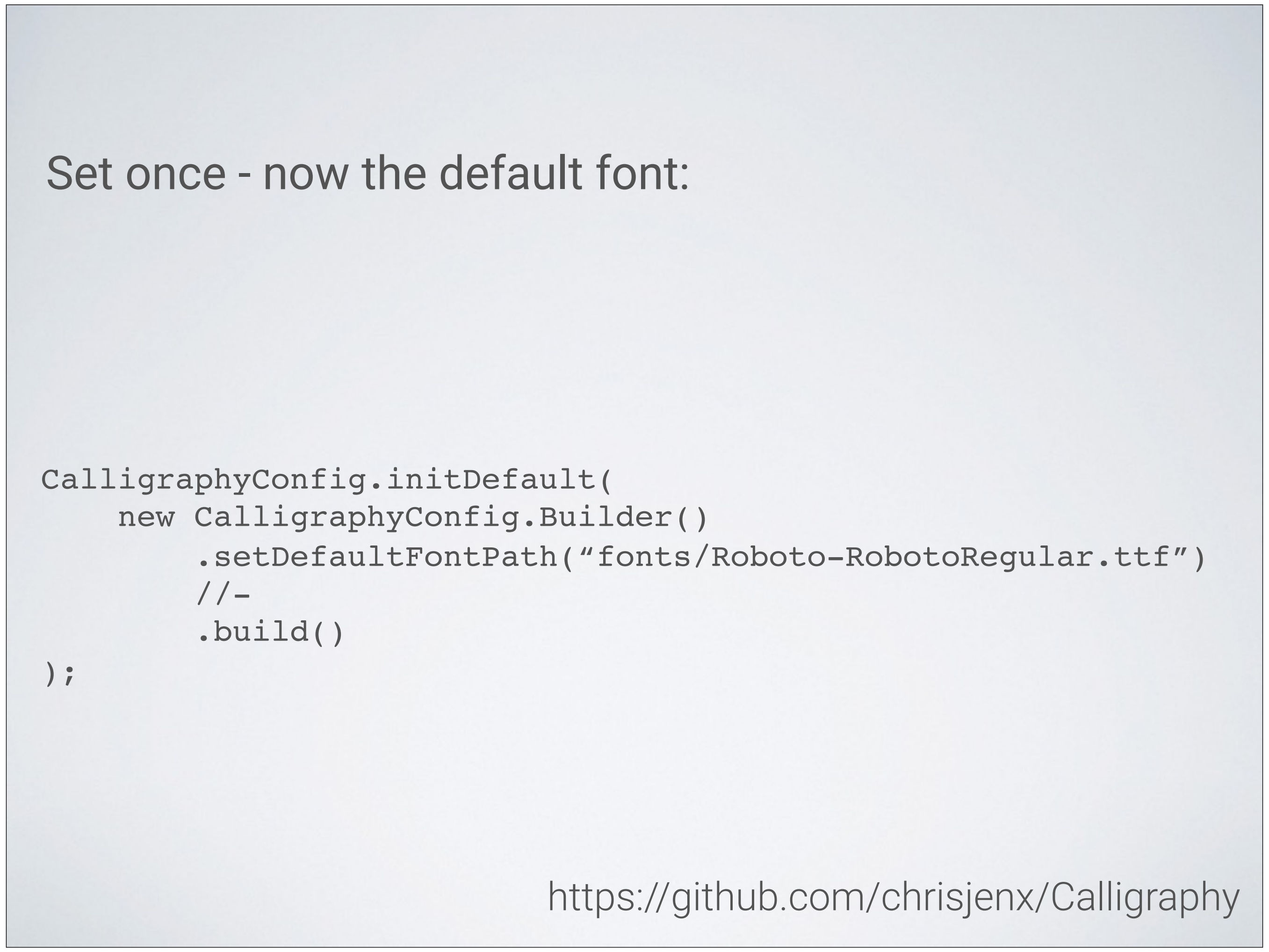
https://github.com/chrisjenx/Calligraphy

Convinced the name is about 90% of the success…



END?

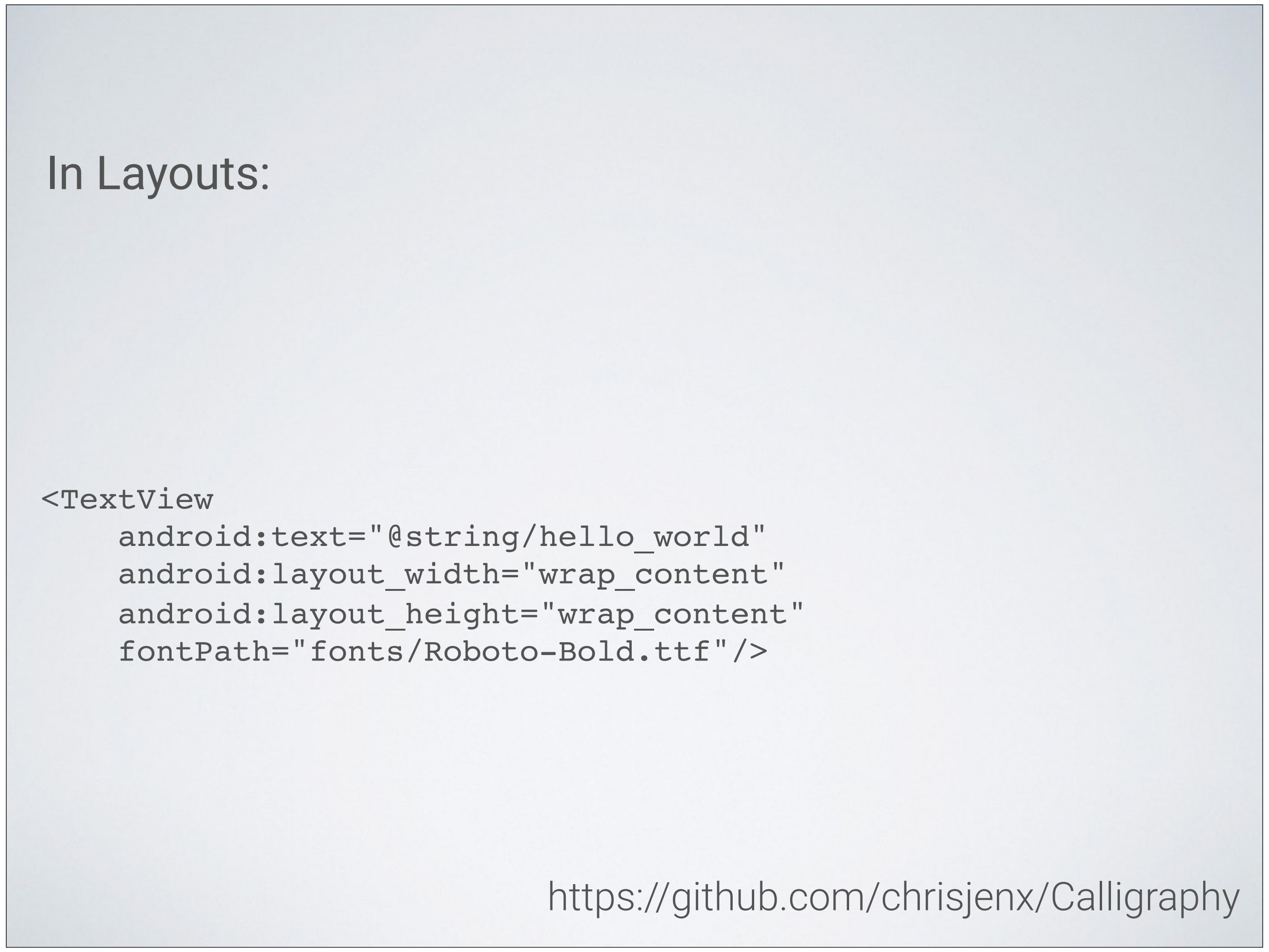
I would end the story here but more to come. So why do you care?
Brief overview of how to use it.


Set once - now the default font:

```
CalligraphyConfig.initDefault(
    new CalligraphyConfig.Builder()
        .setDefaultFontPath(“fonts/Roboto-RobotoRegular.ttf”)
        //-
        .build()
);
```

https://github.com/chrisjenx/Calligraphy

Key feature is:

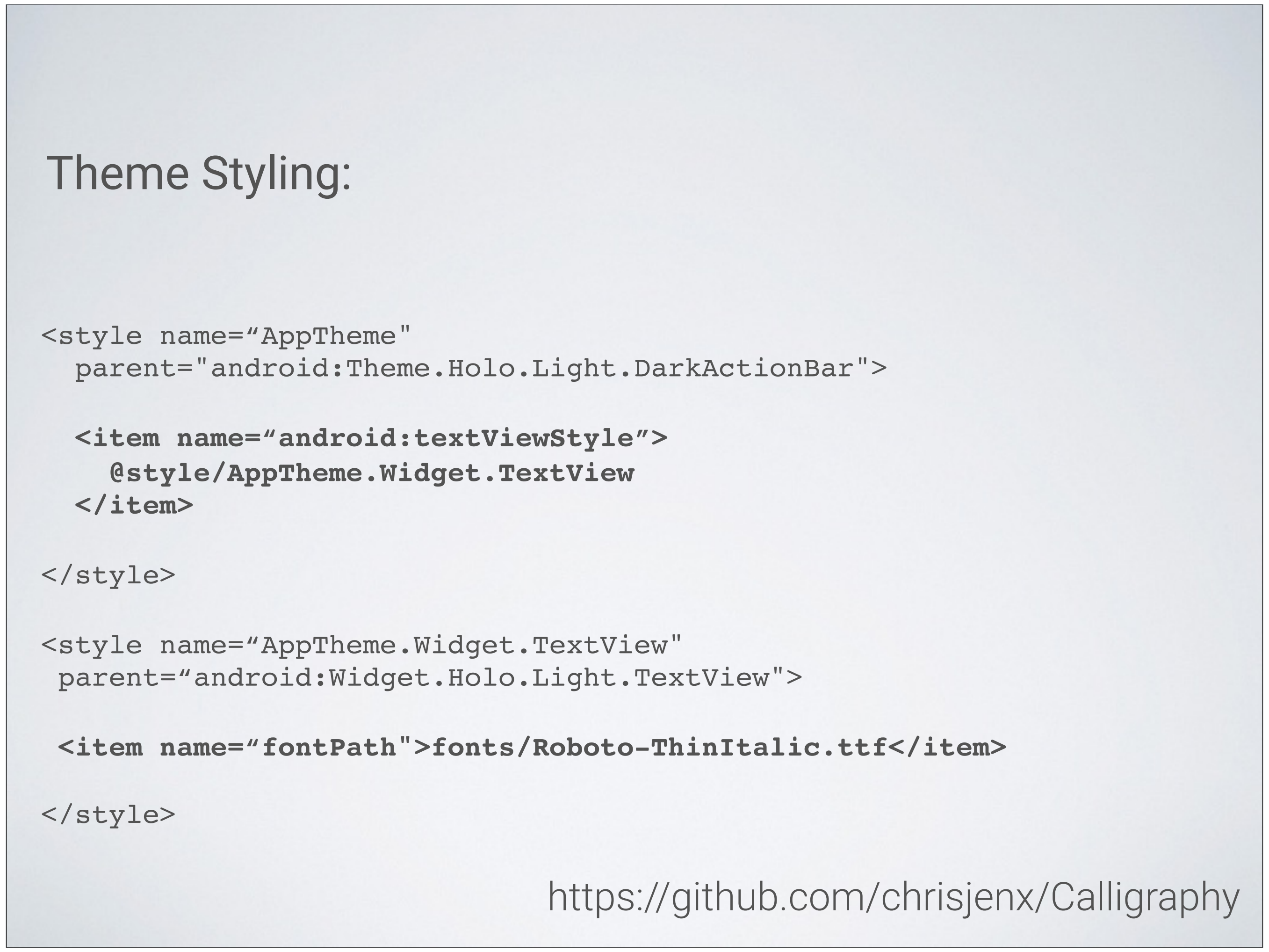


In Layouts:

```
<TextView
    android:text="@string/hello_world"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    fontPath="fonts/Roboto-Bold.ttf"/>
```

https://github.com/chrisjenx/Calligraphy

No more custom views, also saves you extension hell.
Note the missing namespace (thats intentional)

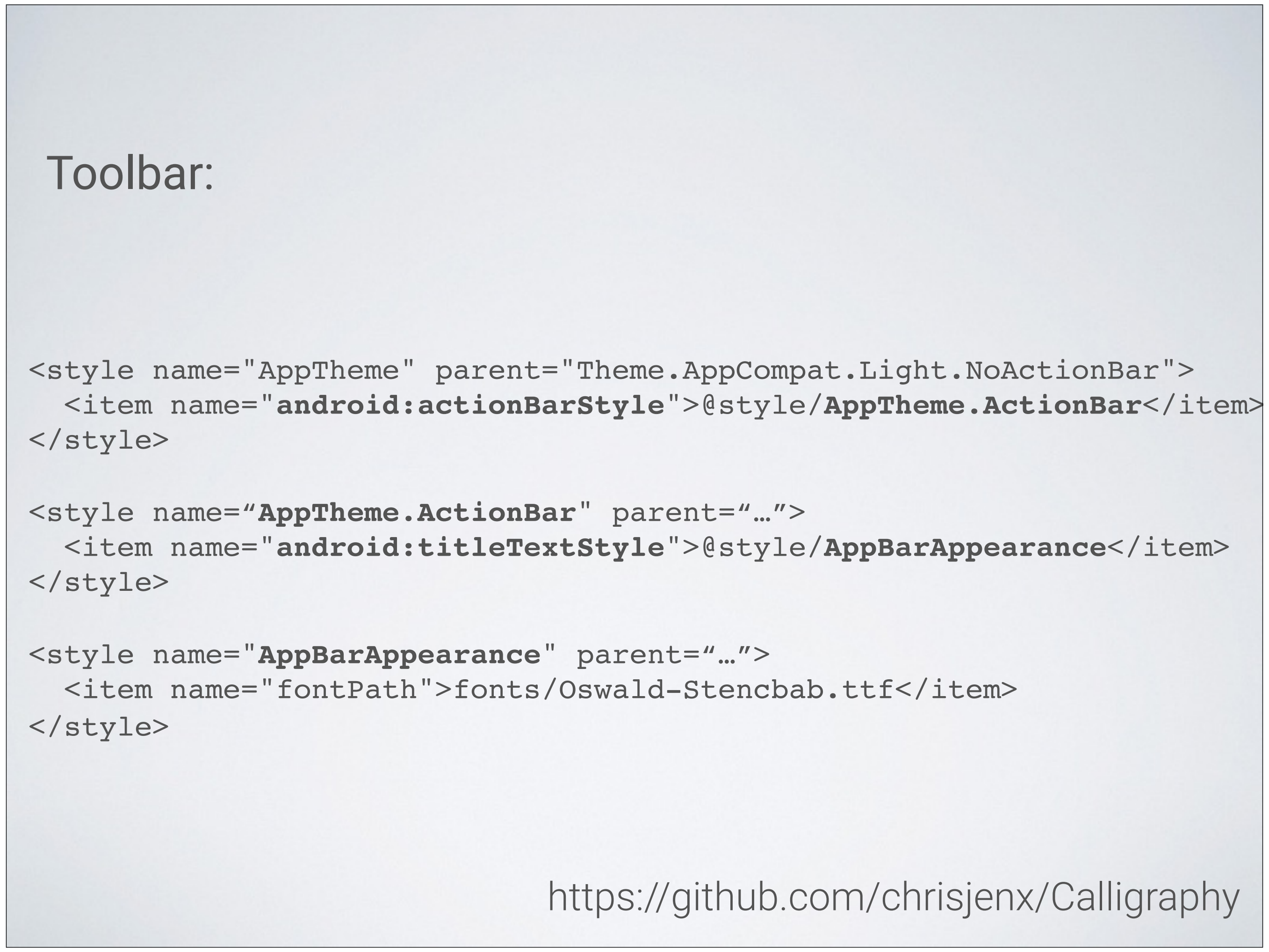


Theme Styling:

```
<style name=“AppTheme"
  parent="android:Theme.Holo.Light.DarkActionBar">

  <item name=“android:textViewStyle”>
    @style/AppTheme.Widget.TextView
  </item>

</style>

<style name=“AppTheme.Widget.TextView"
 parent=“android:Widget.Holo.Light.TextView">

 <item name=“fontPath">fonts/Roboto-ThinItalic.ttf</item>

</style>
```

https://github.com/chrisjenx/Calligraphy

Can also do this for editTextViewStyle etc.

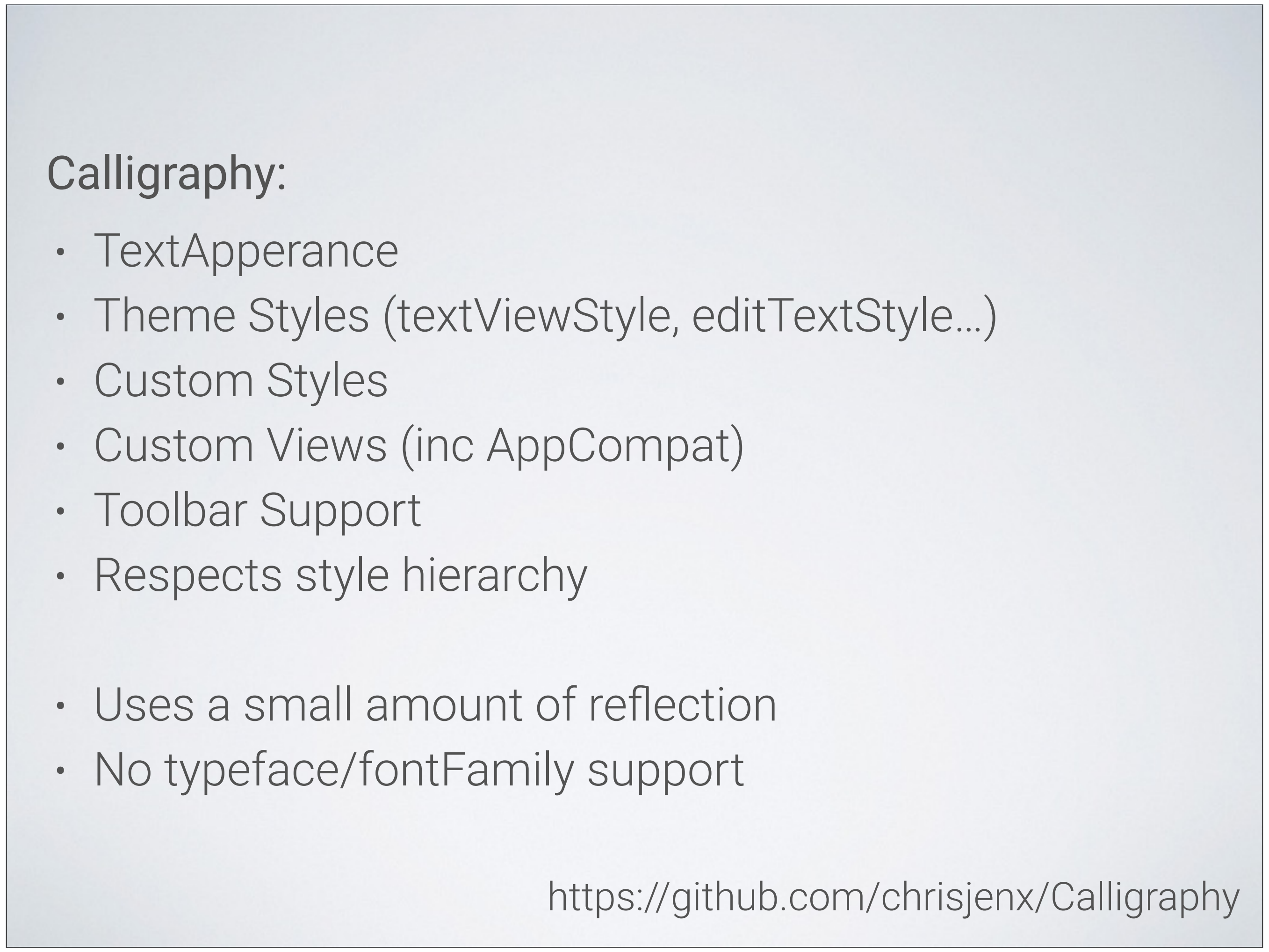


Toolbar:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
  <item name="android:actionBarStyle">@style/AppTheme.ActionBar</item>
</style>

<style name=“AppTheme.ActionBar" parent=“…”>
  <item name="android:titleTextStyle">@style/AppBarAppearance</item>
</style>

<style name="AppBarAppearance" parent=“…”>
  <item name="fontPath">fonts/Oswald-Stencbab.ttf</item>
</style>
```

https://github.com/chrisjenx/Calligraphy

AppBar Appearance -> ActionBar Style -> App Theme.


Calligraphy:

- TextApperance
- Theme Styles (textViewStyle, editTextStyle…)
- Custom Styles
- Custom Views (inc AppCompat)
- Toolbar Support
- Respects style hierarchy

- Uses a small amount of reflection
- No typeface/fontFamily support

https://github.com/chrisjenx/Calligraphy

Some pros and cons

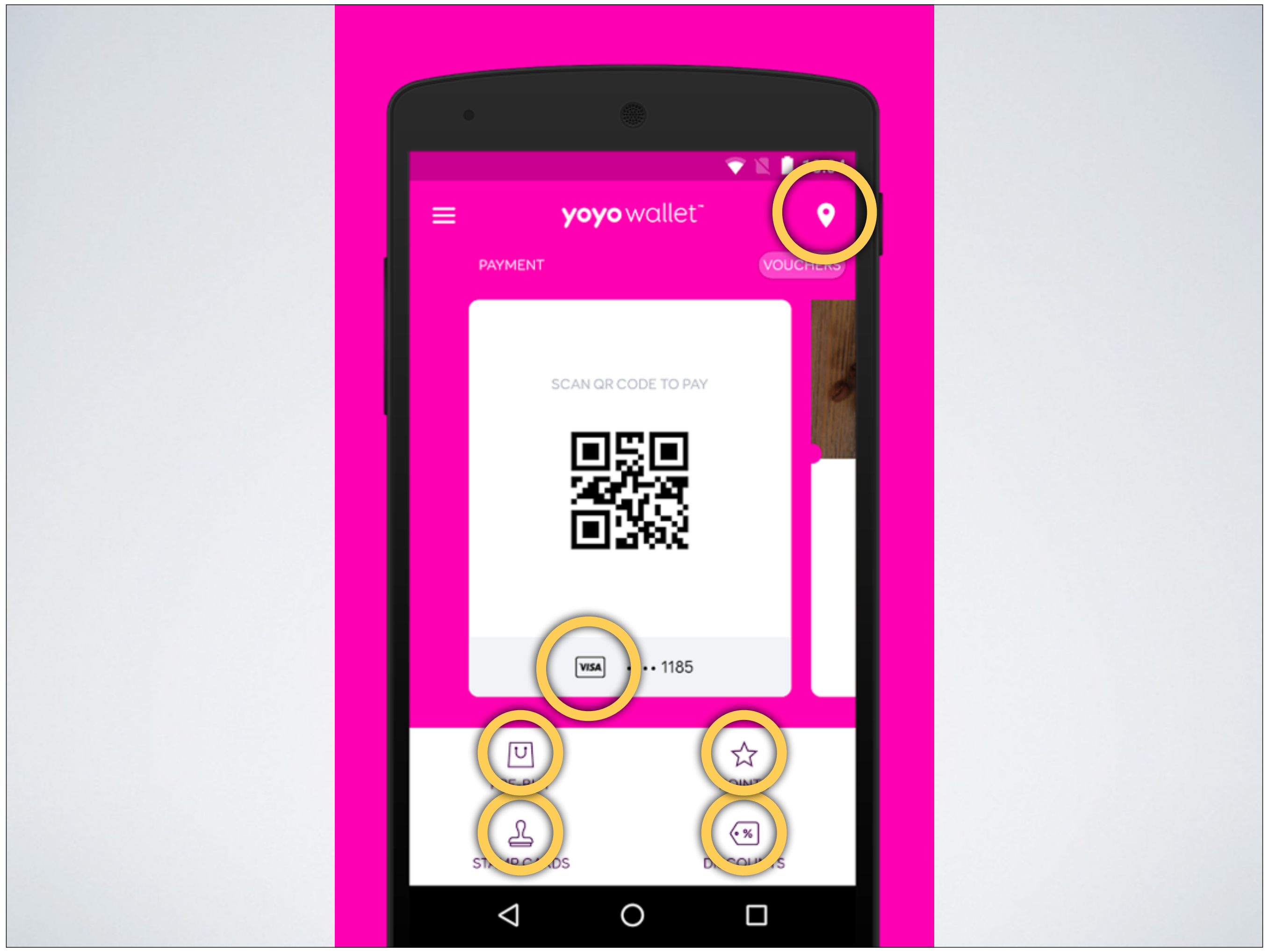
http://bit.ly/android-dialogs

Not getting into how it works, see link.

Back to the story. Moved to being a contractor for while.

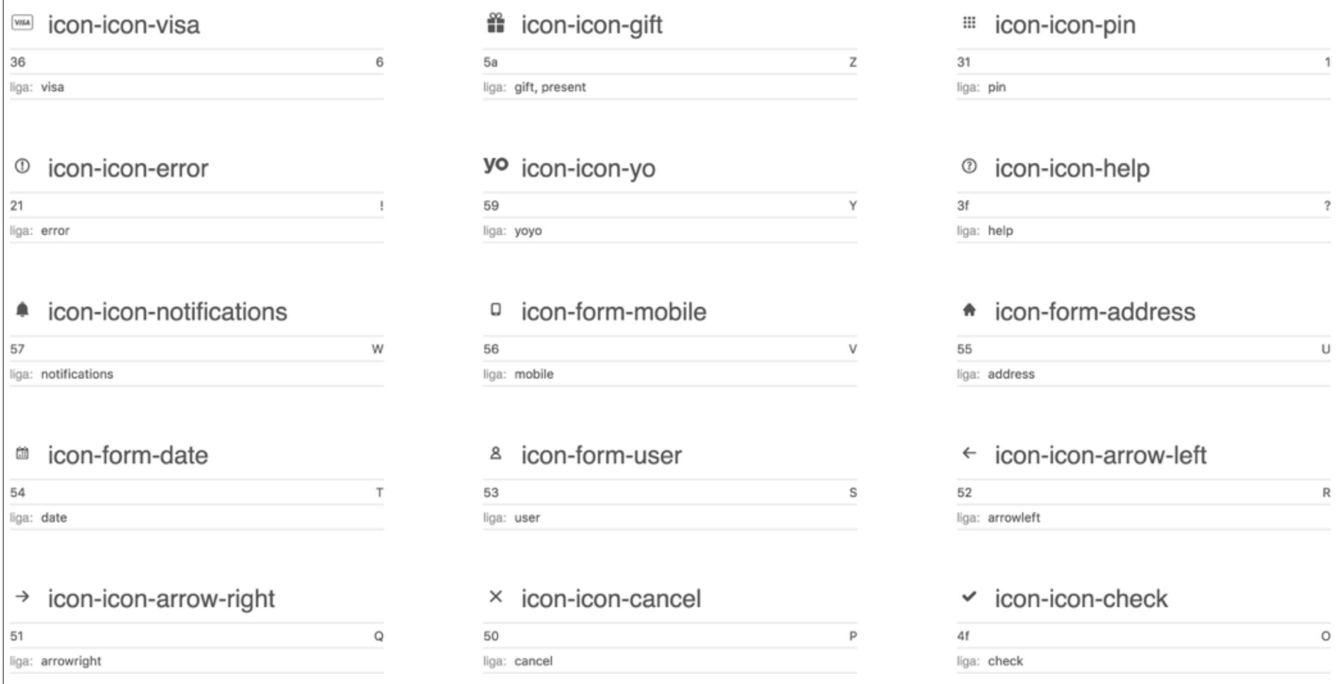
yoyo wallet™

I joined Yoyo. Yet again started working with another superb designer (Who now works for Apple).
Really high requirements.



Before vector icons

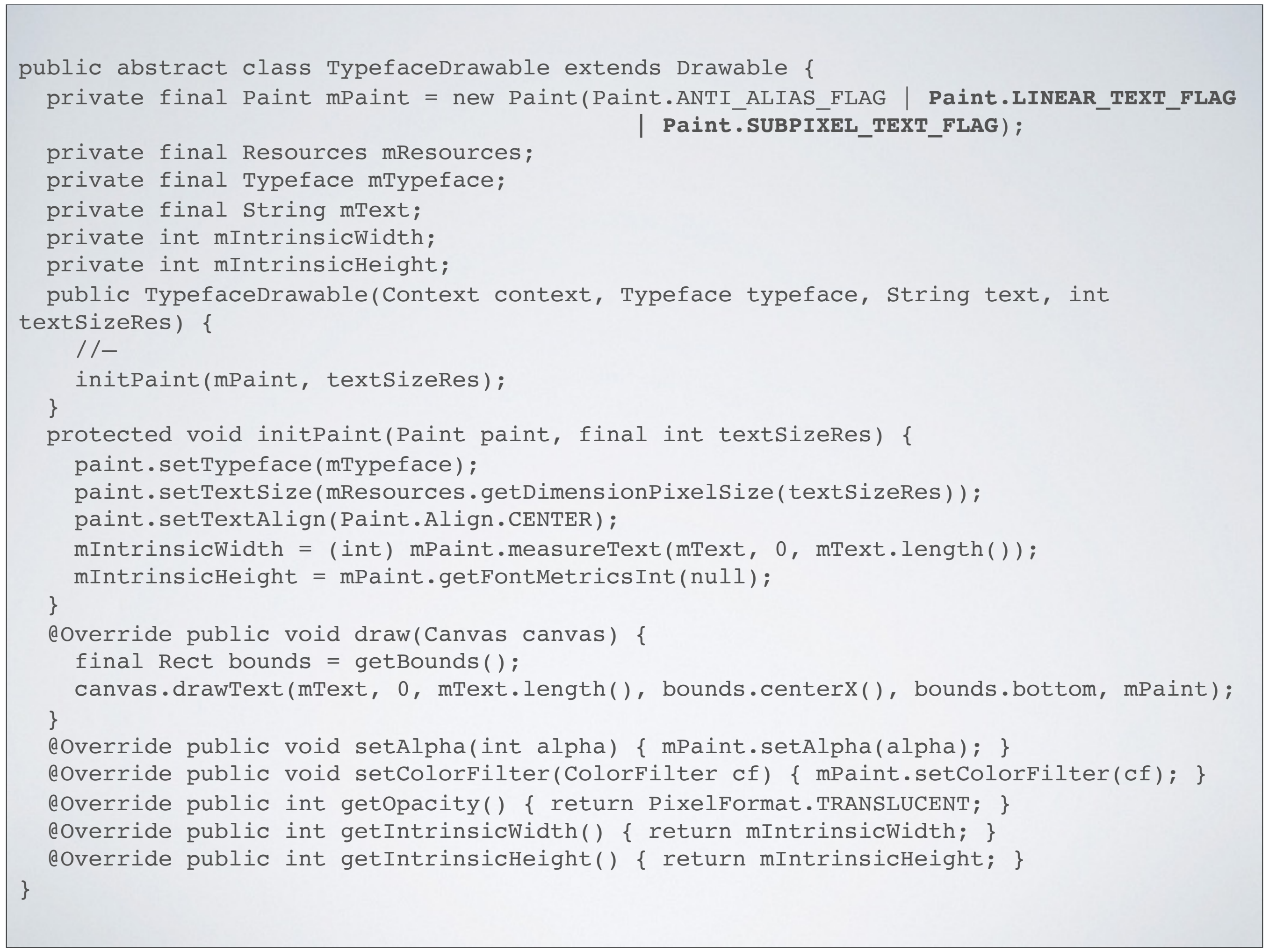
Asset Board:



icon-icon-visa	36	6	liga: visa
icon-icon-gift	5a	Z	liga: gift, present
icon-icon-pin	31	1	liga: pin
icon-icon-error	21	!	liga: error
icon-icon-yo	59	Y	liga: yoyo
icon-icon-help	3f	?	liga: help
icon-icon-notifications	57	W	liga: notifications
icon-form-mobile	56	V	liga: mobile
icon-form-address	55	U	liga: address
icon-form-date	54	T	liga: date
icon-form-user	53	S	liga: user
icon-icon-arrow-left	52	R	liga: arrowleft
icon-icon-arrow-right	51	Q	liga: arrowright
icon-icon-cancel	50	P	liga: cancel
icon-icon-check	4f	O	liga: check

Asset board, everything is mapped to a letter.
If android properly supported ligatures we could use words instead.

```
public abstract class TypefaceDrawable extends Drawable {
  private final Paint mPaint = new Paint(Paint.ANTI_ALIAS_FLAG | Paint.LINEAR_TEXT_FLAG
                                             | Paint.SUBPIXEL_TEXT_FLAG);
  private final Resources mResources;
  private final Typeface mTypeface;
  private final String mText;
  private int mIntrinsicWidth;
  private int mIntrinsicHeight;
  public TypefaceDrawable(Context context, Typeface typeface, String text, int
textSizeRes) {
    //–
    initPaint(mPaint, textSizeRes);
  }
  protected void initPaint(Paint paint, final int textSizeRes) {
    paint.setTypeface(mTypeface);
    paint.setTextSize(mResources.getDimensionPixelSize(textSizeRes));
    paint.setTextAlign(Paint.Align.CENTER);
    mIntrinsicWidth = (int) mPaint.measureText(mText, 0, mText.length());
    mIntrinsicHeight = mPaint.getFontMetricsInt(null);
  }
  @Override public void draw(Canvas canvas) {
    final Rect bounds = getBounds();
    canvas.drawText(mText, 0, mText.length(), bounds.centerX(), bounds.bottom, mPaint);
  }
  @Override public void setAlpha(int alpha) { mPaint.setAlpha(alpha); }
  @Override public void setColorFilter(ColorFilter cf) { mPaint.setColorFilter(cf); }
  @Override public int getOpacity() { return PixelFormat.TRANSLUCENT; }
  @Override public int getIntrinsicWidth() { return mIntrinsicWidth; }
  @Override public int getIntrinsicHeight() { return mIntrinsicHeight; }
}
```

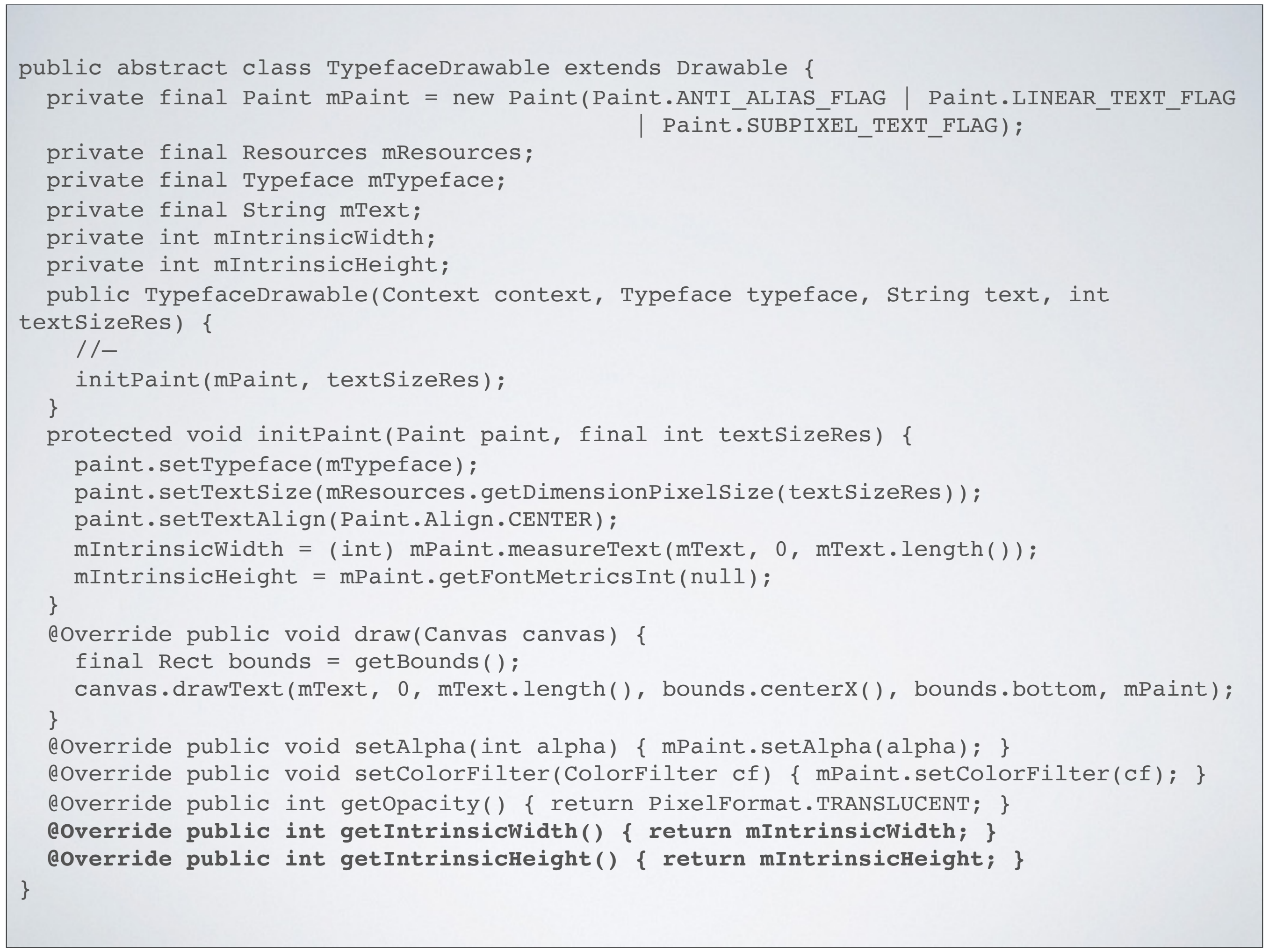


```
public abstract class TypefaceDrawable extends Drawable {
  private final Paint mPaint = new Paint(Paint.ANTI_ALIAS_FLAG | Paint.LINEAR_TEXT_FLAG
                                             | Paint.SUBPIXEL_TEXT_FLAG);
  private final Resources mResources;
  private final Typeface mTypeface;
  private final String mText;
  private int mIntrinsicWidth;
  private int mIntrinsicHeight;
  public TypefaceDrawable(Context context, Typeface typeface, String text, int
textSizeRes) {
    //…
    initPaint(mPaint, textSizeRes);
  }
  protected void initPaint(Paint paint, final int textSizeRes) {
    paint.setTypeface(mTypeface);
    paint.setTextSize(mResources.getDimensionPixelSize(textSizeRes));
    paint.setTextAlign(Paint.Align.CENTER);
    mIntrinsicWidth = (int) mPaint.measureText(mText, 0, mText.length());
    mIntrinsicHeight = mPaint.getFontMetricsInt(null);
  }
  @Override public void draw(Canvas canvas) {
    final Rect bounds = getBounds();
    canvas.drawText(mText, 0, mText.length(), bounds.centerX(), bounds.bottom, mPaint);
  }
  @Override public void setAlpha(int alpha) { mPaint.setAlpha(alpha); }
  @Override public void setColorFilter(ColorFilter cf) { mPaint.setColorFilter(cf); }
  @Override public int getOpacity() { return PixelFormat.TRANSLUCENT; }
  @Override public int getIntrinsicWidth() { return mIntrinsicWidth; }
  @Override public int getIntrinsicHeight() { return mIntrinsicHeight; }
}
```

This turns of Glyph caching so that assets are not scaled, sub pixel basically turns off pixel snapping if your glyph would be between two pixels.

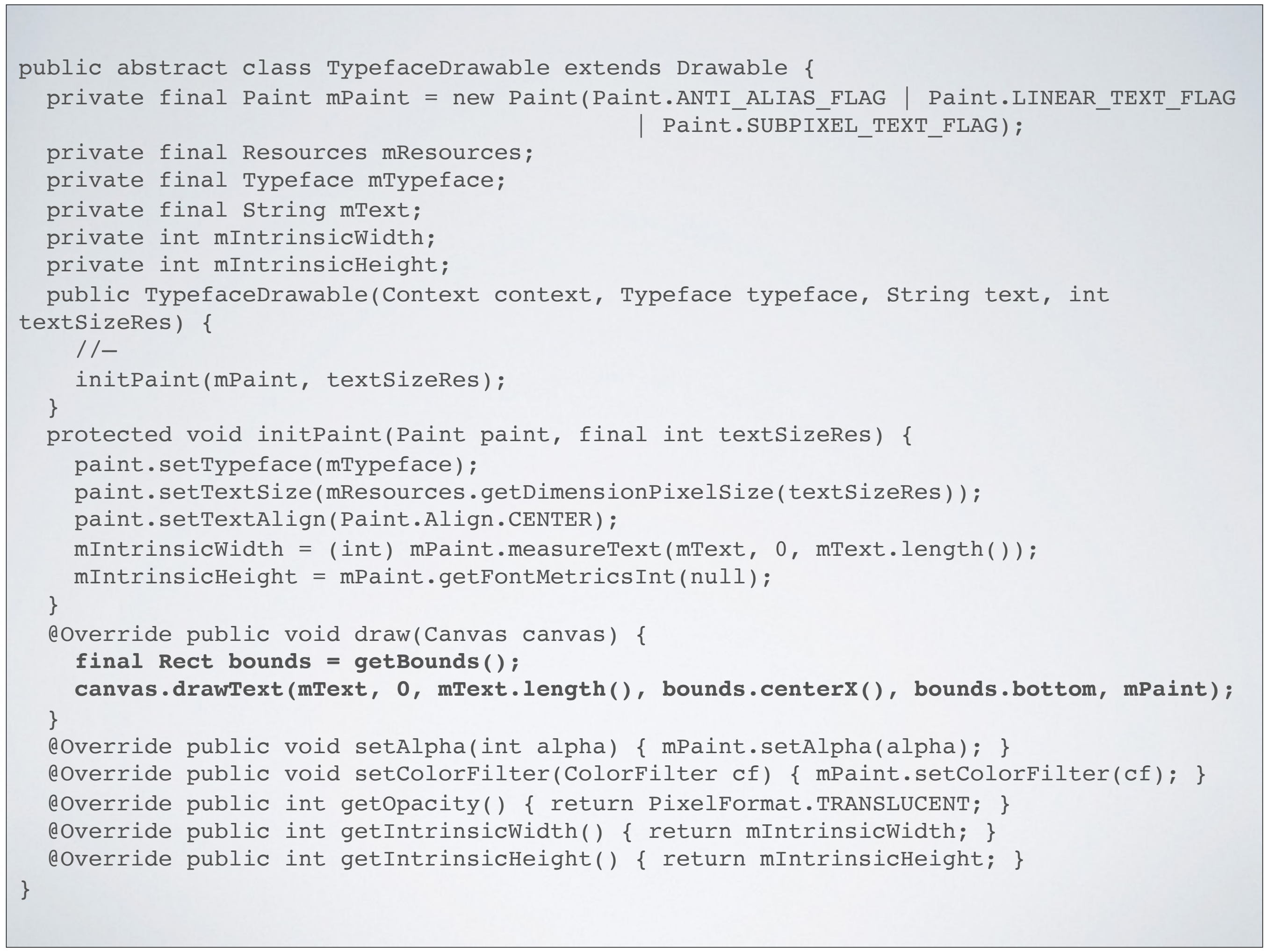
```
public abstract class TypefaceDrawable extends Drawable {
  private final Paint mPaint = new Paint(Paint.ANTI_ALIAS_FLAG | Paint.LINEAR_TEXT_FLAG
                                             | Paint.SUBPIXEL_TEXT_FLAG);
  private final Resources mResources;
  private final Typeface mTypeface;
  private final String mText;
  private int mIntrinsicWidth;
  private int mIntrinsicHeight;
  public TypefaceDrawable(Context context, Typeface typeface, String text, int
textSizeRes) {
    //–
    initPaint(mPaint, textSizeRes);
  }
  protected void initPaint(Paint paint, final int textSizeRes) {
    paint.setTypeface(mTypeface);
    paint.setTextSize(mResources.getDimensionPixelSize(textSizeRes));
    paint.setTextAlign(Paint.Align.CENTER);
    mIntrinsicWidth = (int) mPaint.measureText(mText, 0, mText.length());
    mIntrinsicHeight = mPaint.getFontMetricsInt(null);
  }
  @Override public void draw(Canvas canvas) {
    final Rect bounds = getBounds();
    canvas.drawText(mText, 0, mText.length(), bounds.centerX(), bounds.bottom, mPaint);
  }
  @Override public void setAlpha(int alpha) { mPaint.setAlpha(alpha); }
  @Override public void setColorFilter(ColorFilter cf) { mPaint.setColorFilter(cf); }
  @Override public int getOpacity() { return PixelFormat.TRANSLUCENT; }
  @Override public int getIntrinsicWidth() { return mIntrinsicWidth; }
  @Override public int getIntrinsicHeight() { return mIntrinsicHeight; }
}
```

Make sure to set the typeface before measuring.



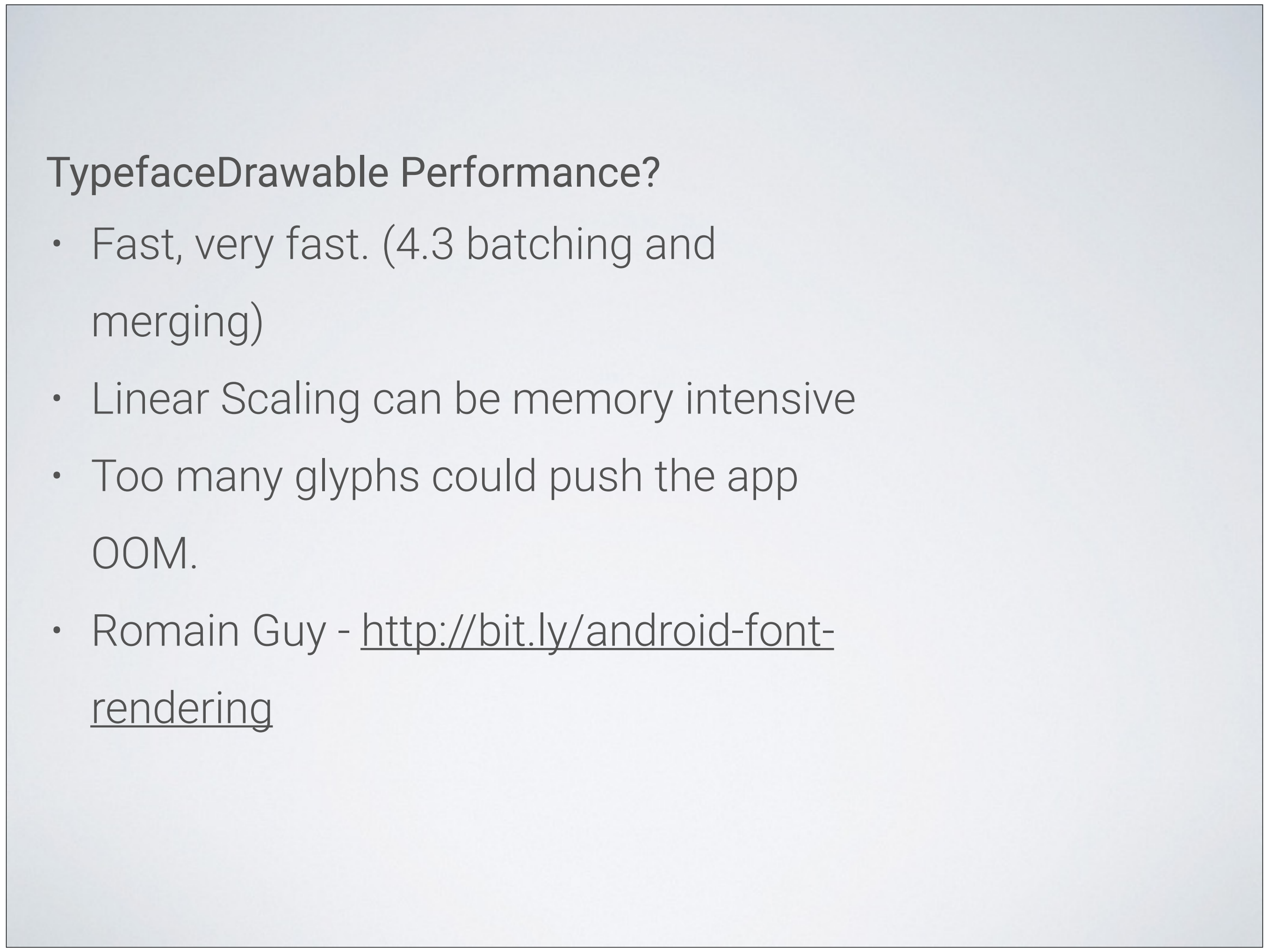
```
public abstract class TypefaceDrawable extends Drawable {
  private final Paint mPaint = new Paint(Paint.ANTI_ALIAS_FLAG | Paint.LINEAR_TEXT_FLAG
                                            | Paint.SUBPIXEL_TEXT_FLAG);
  private final Resources mResources;
  private final Typeface mTypeface;
  private final String mText;
  private int mIntrinsicWidth;
  private int mIntrinsicHeight;
  public TypefaceDrawable(Context context, Typeface typeface, String text, int
textSizeRes) {
    //–
    initPaint(mPaint, textSizeRes);
  }
  protected void initPaint(Paint paint, final int textSizeRes) {
    paint.setTypeface(mTypeface);
    paint.setTextSize(mResources.getDimensionPixelSize(textSizeRes));
    paint.setTextAlign(Paint.Align.CENTER);
    mIntrinsicWidth = (int) mPaint.measureText(mText, 0, mText.length());
    mIntrinsicHeight = mPaint.getFontMetricsInt(null);
  }
  @Override public void draw(Canvas canvas) {
    final Rect bounds = getBounds();
    canvas.drawText(mText, 0, mText.length(), bounds.centerX(), bounds.bottom, mPaint);
  }
  @Override public void setAlpha(int alpha) { mPaint.setAlpha(alpha); }
  @Override public void setColorFilter(ColorFilter cf) { mPaint.setColorFilter(cf); }
  @Override public int getOpacity() { return PixelFormat.TRANSLUCENT; }
  @Override public int getIntrinsicWidth() { return mIntrinsicWidth; }
  @Override public int getIntrinsicHeight() { return mIntrinsicHeight; }
}
```

This gives the layout system a size to work with.



```java
public abstract class TypefaceDrawable extends Drawable {
  private final Paint mPaint = new Paint(Paint.ANTI_ALIAS_FLAG | Paint.LINEAR_TEXT_FLAG
                                            | Paint.SUBPIXEL_TEXT_FLAG);
  private final Resources mResources;
  private final Typeface mTypeface;
  private final String mText;
  private int mIntrinsicWidth;
  private int mIntrinsicHeight;
  public TypefaceDrawable(Context context, Typeface typeface, String text, int
textSizeRes) {
    //—
    initPaint(mPaint, textSizeRes);
  }
  protected void initPaint(Paint paint, final int textSizeRes) {
    paint.setTypeface(mTypeface);
    paint.setTextSize(mResources.getDimensionPixelSize(textSizeRes));
    paint.setTextAlign(Paint.Align.CENTER);
    mIntrinsicWidth = (int) mPaint.measureText(mText, 0, mText.length());
    mIntrinsicHeight = mPaint.getFontMetricsInt(null);
  }
  @Override public void draw(Canvas canvas) {
    final Rect bounds = getBounds();
    canvas.drawText(mText, 0, mText.length(), bounds.centerX(), bounds.bottom, mPaint);
  }
  @Override public void setAlpha(int alpha) { mPaint.setAlpha(alpha); }
  @Override public void setColorFilter(ColorFilter cf) { mPaint.setColorFilter(cf); }
  @Override public int getOpacity() { return PixelFormat.TRANSLUCENT; }
  @Override public int getIntrinsicWidth() { return mIntrinsicWidth; }
  @Override public int getIntrinsicHeight() { return mIntrinsicHeight; }
}
```

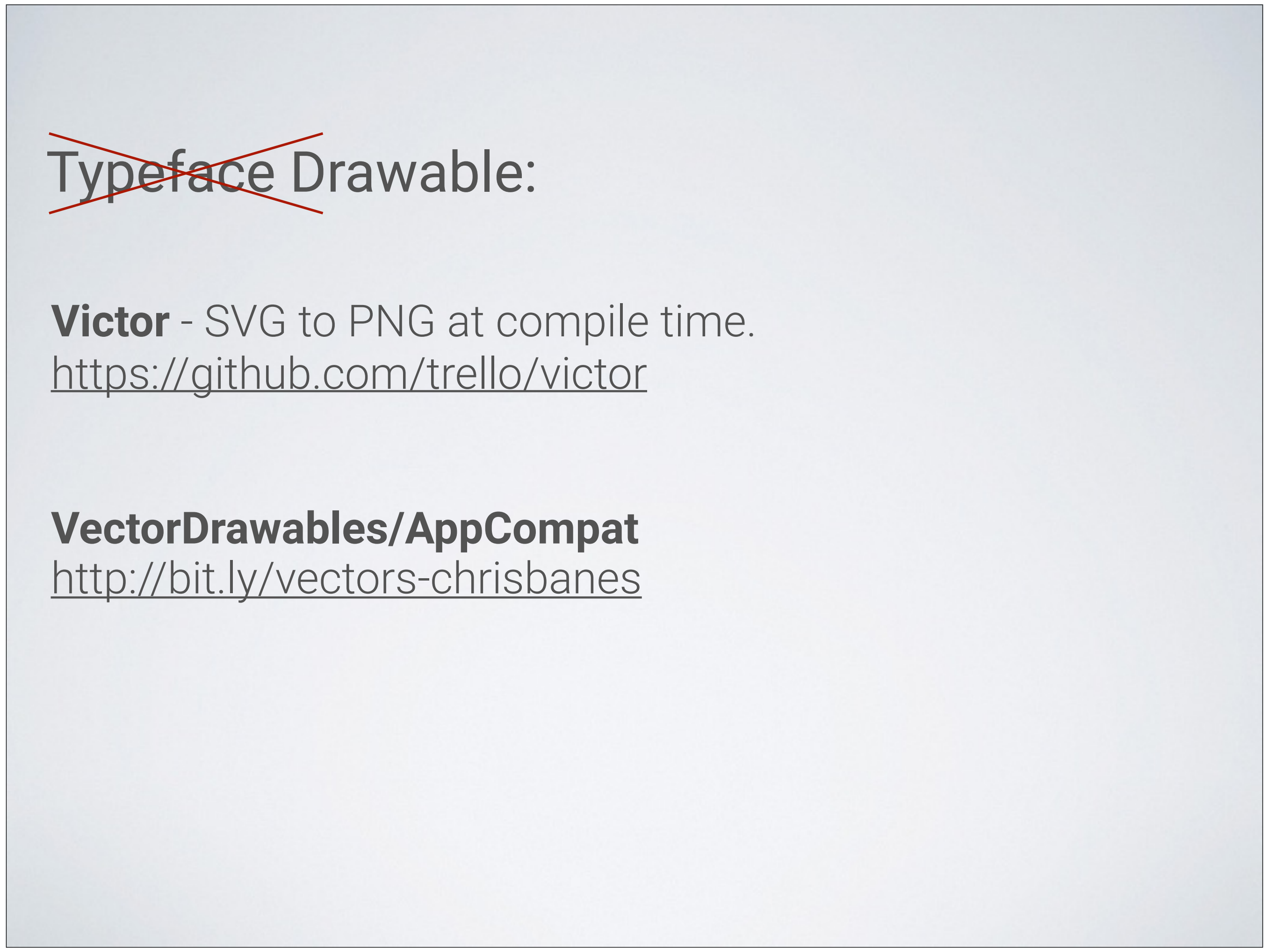
This gives the layout system a size to work with.



TypefaceDrawable Performance?

- Fast, very fast. (4.3 batching and merging)
- Linear Scaling can be memory intensive
- Too many glyphs could push the app OOM.
- Romain Guy - http://bit.ly/android-font-rendering

- Assets are rendered native level, rendered by Harfbuzz - Used to be Skia
- Linear scaling won't use any caches - accurate typefaces, rending time.
- Lots of glyphs would take up a lot of space, LRU so should purge out older glyphs. (Still memory churn)
- Guy talks about this.

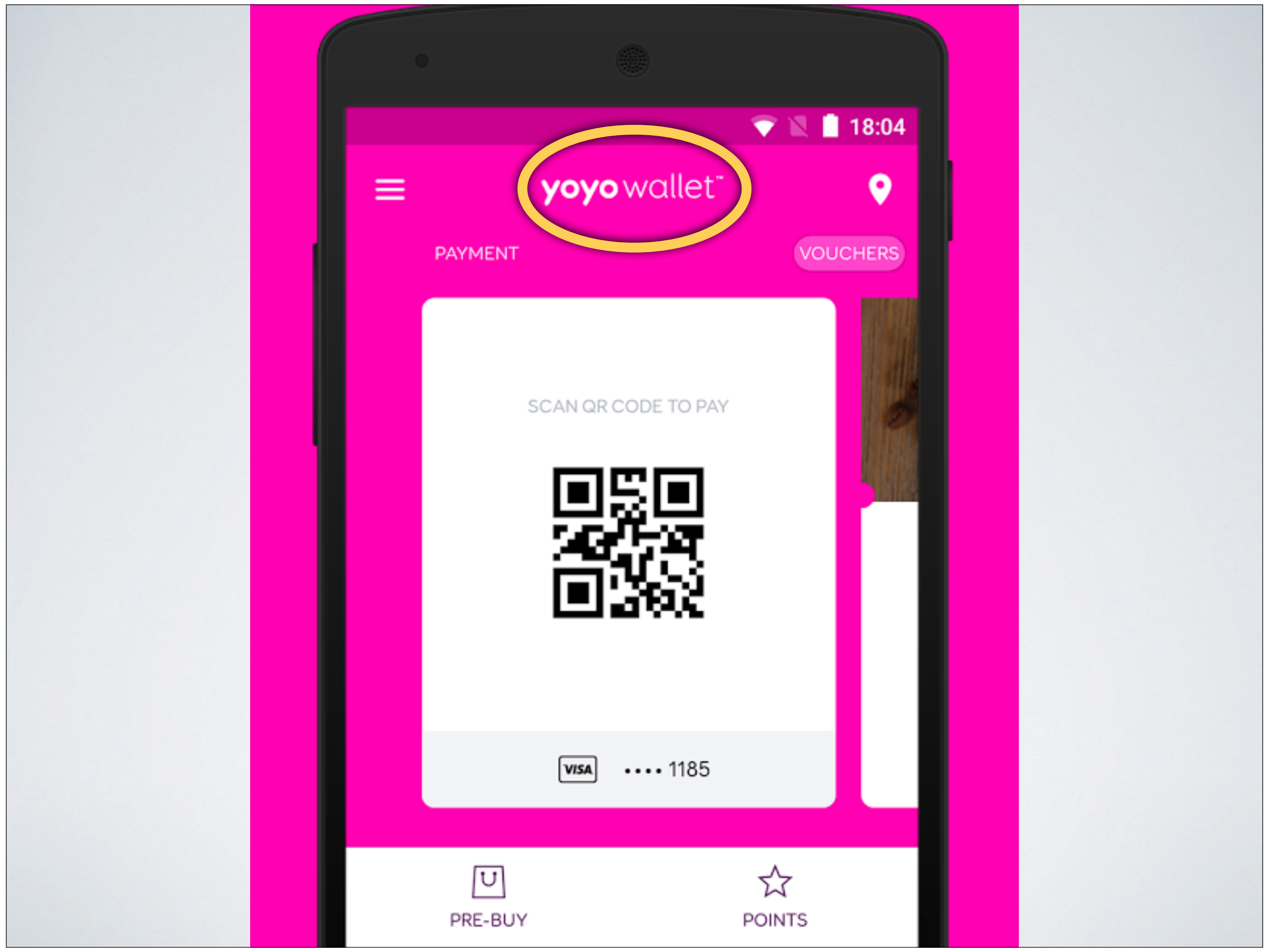


~~Typeface~~ Drawable:

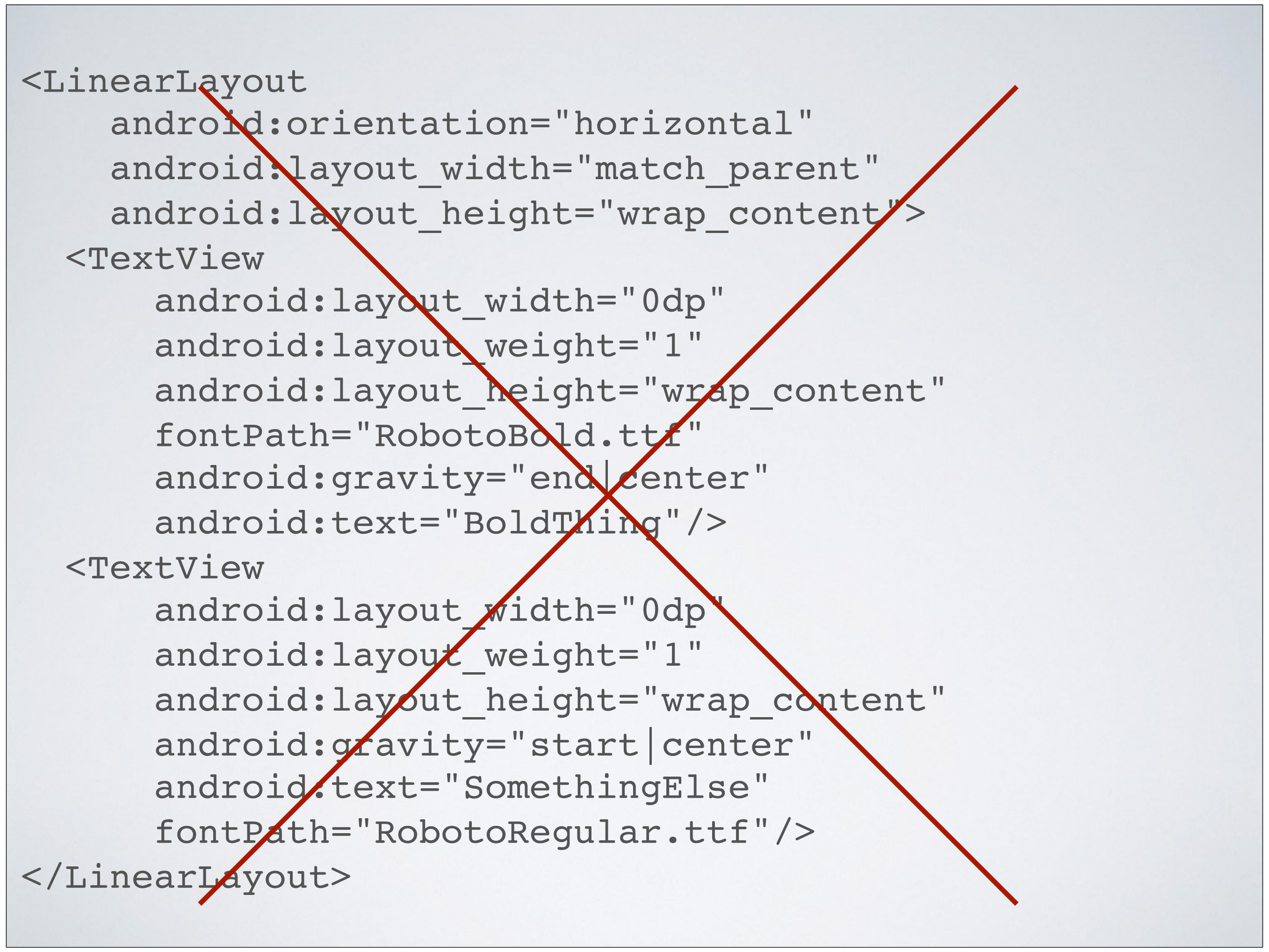
Victor - SVG to PNG at compile time.
https://github.com/trello/victor

VectorDrawables/AppCompat
http://bit.ly/vectors-chrisbanes

Now, I wouldn't use TypefaceDrawables



Would be tricky to do without code, Combining fonts.



```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
  <TextView
      android:layout_width="0dp"
      android:layout_weight="1"
      android:layout_height="wrap_content"
      fontPath="RobotoBold.ttf"
      android:gravity="end|center"
      android:text="BoldThing"/>
  <TextView
      android:layout_width="0dp"
      android:layout_weight="1"
      android:layout_height="wrap_content"
      android:gravity="start|center"
      android:text="SomethingElse"
      fontPath="RobotoRegular.ttf"/>
</LinearLayout>
```

You could do this.
Don’t.

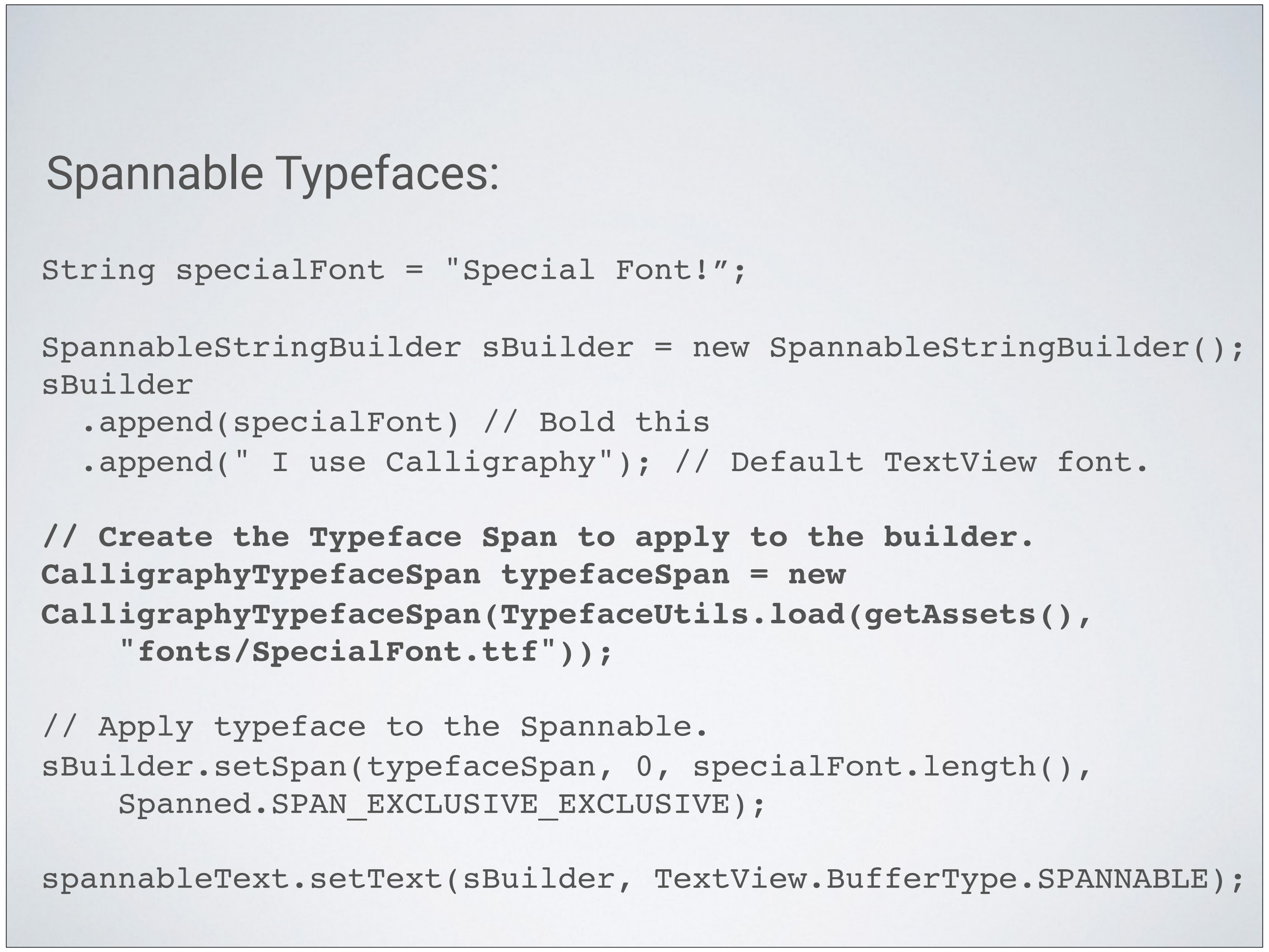
Spannable Typefaces:

Spannable Typefaces:

```
String specialFont = "Special Font!”;

SpannableStringBuilder sBuilder = new SpannableStringBuilder();
sBuilder
  .append(specialFont) // Bold this
  .append(" I use Calligraphy"); // Default TextView font.

// Create the Typeface Span to apply to the builder.
CalligraphyTypefaceSpan typefaceSpan = new
CalligraphyTypefaceSpan(TypefaceUtils.load(getAssets(),
    "fonts/SpecialFont.ttf"));

// Apply typeface to the Spannable.
sBuilder.setSpan(typefaceSpan, 0, specialFont.length(),
    Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);

spannableText.setText(sBuilder, TextView.BufferType.SPANNABLE);
```

Spannable Typefaces:

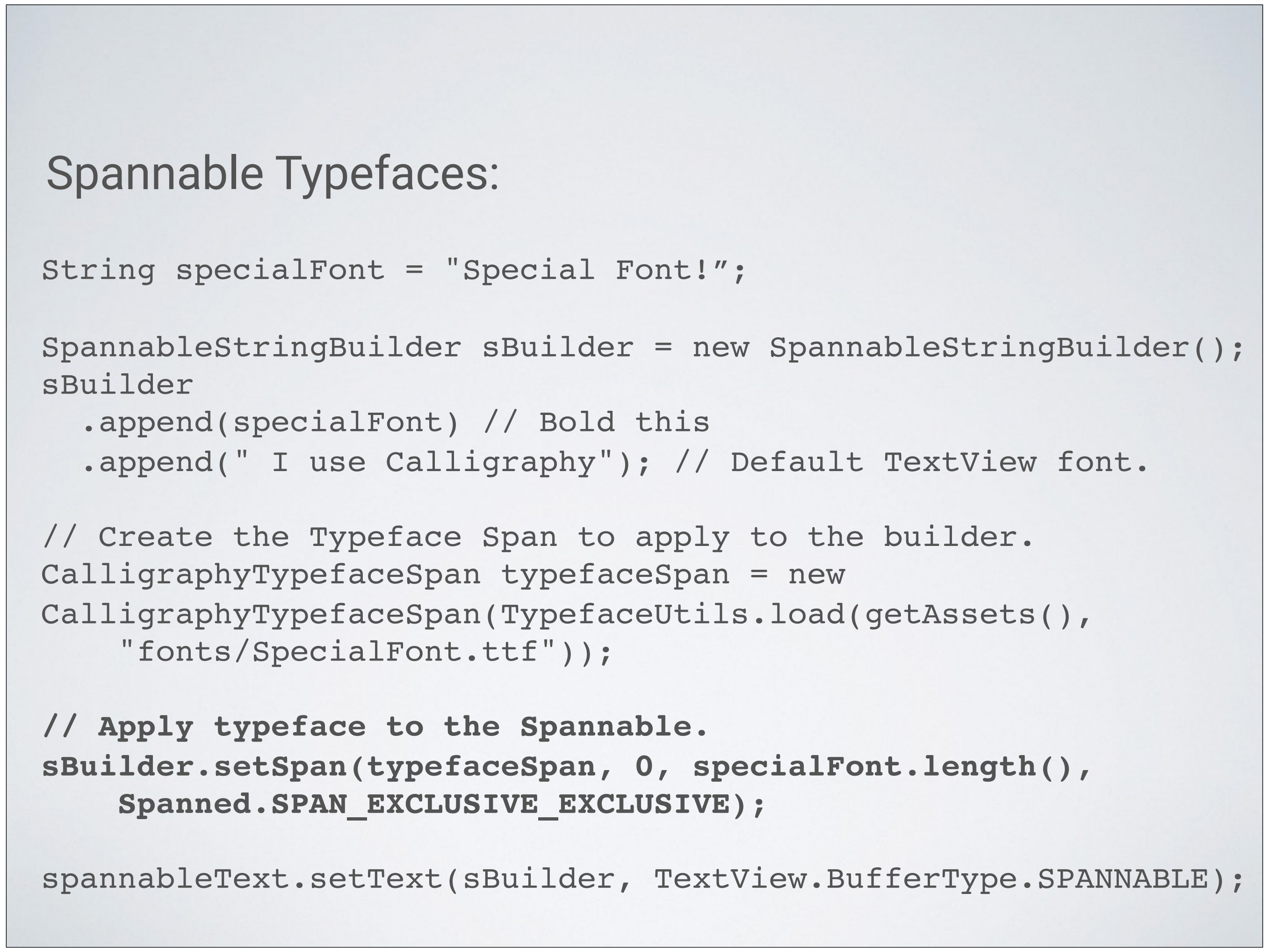
```
String specialFont = "Special Font!”;

SpannableStringBuilder sBuilder = new SpannableStringBuilder();
sBuilder
  .append(specialFont) // Bold this
  .append(" I use Calligraphy"); // Default TextView font.

// Create the Typeface Span to apply to the builder.
CalligraphyTypefaceSpan typefaceSpan = new
CalligraphyTypefaceSpan(TypefaceUtils.load(getAssets(),
    "fonts/SpecialFont.ttf"));

// Apply typeface to the Spannable.
sBuilder.setSpan(typefaceSpan, 0, specialFont.length(),
    Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);

spannableText.setText(sBuilder, TextView.BufferType.SPANNABLE);
```



Spannable Typefaces:

```
String specialFont = "Special Font!”;

SpannableStringBuilder sBuilder = new SpannableStringBuilder();
sBuilder
  .append(specialFont) // Bold this
  .append(" I use Calligraphy"); // Default TextView font.

// Create the Typeface Span to apply to the builder.
CalligraphyTypefaceSpan typefaceSpan = new
CalligraphyTypefaceSpan(TypefaceUtils.load(getAssets(),
    "fonts/SpecialFont.ttf"));

// Apply typeface to the Spannable.
sBuilder.setSpan(typefaceSpan, 0, specialFont.length(),
    Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);

spannableText.setText(sBuilder, TextView.BufferType.SPANNABLE);
```

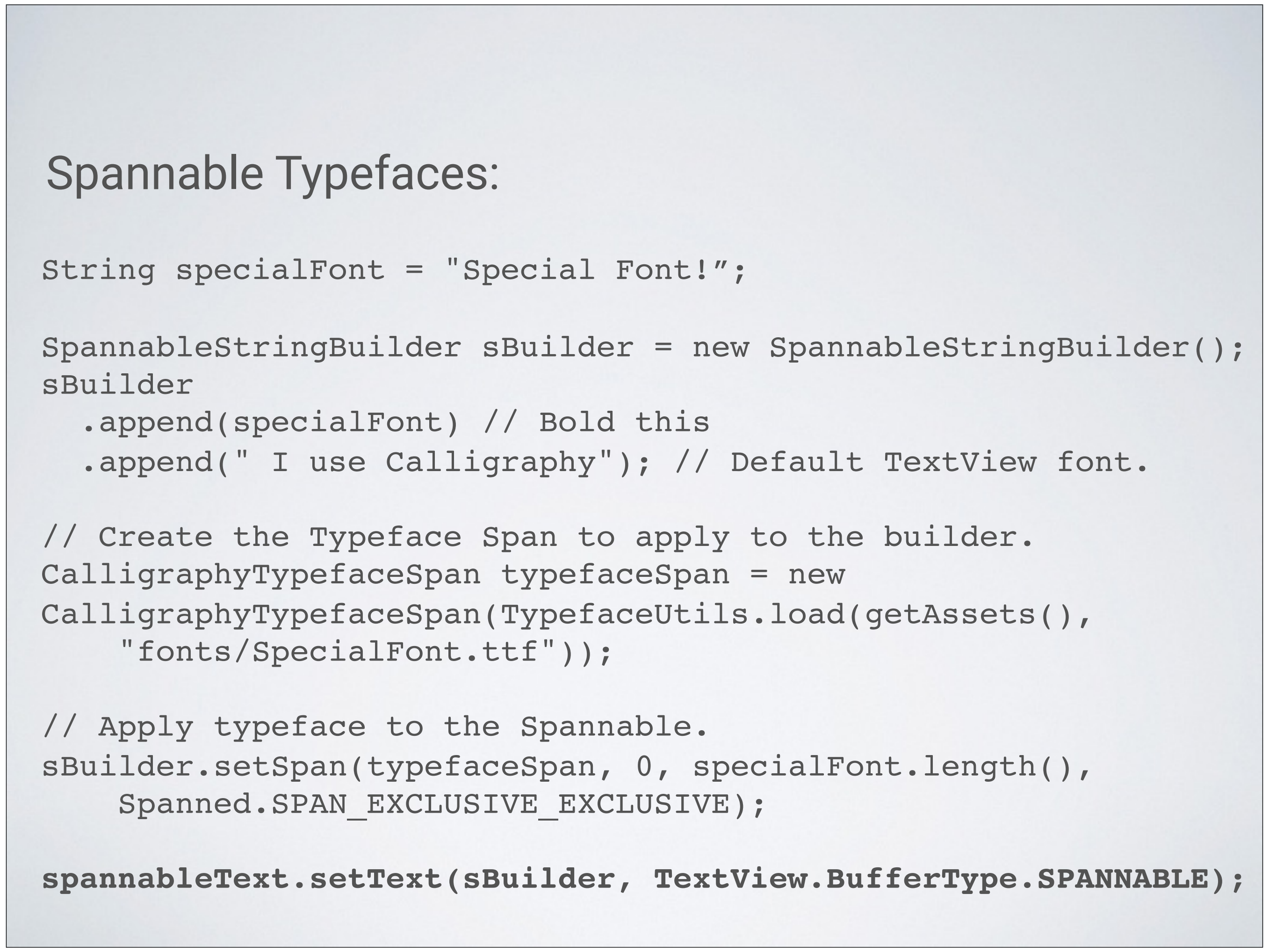
Create the Typeface span, I’ll go through the code of the TypefaceSpannable later.



Spannable Typefaces:

```
String specialFont = "Special Font!”;

SpannableStringBuilder sBuilder = new SpannableStringBuilder();
sBuilder
  .append(specialFont) // Bold this
  .append(" I use Calligraphy"); // Default TextView font.

// Create the Typeface Span to apply to the builder.
CalligraphyTypefaceSpan typefaceSpan = new
CalligraphyTypefaceSpan(TypefaceUtils.load(getAssets(),
    "fonts/SpecialFont.ttf"));

// Apply typeface to the Spannable.
sBuilder.setSpan(typefaceSpan, 0, specialFont.length(),
    Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);

spannableText.setText(sBuilder, TextView.BufferType.SPANNABLE);
```

Set the Span over the specific text areas.

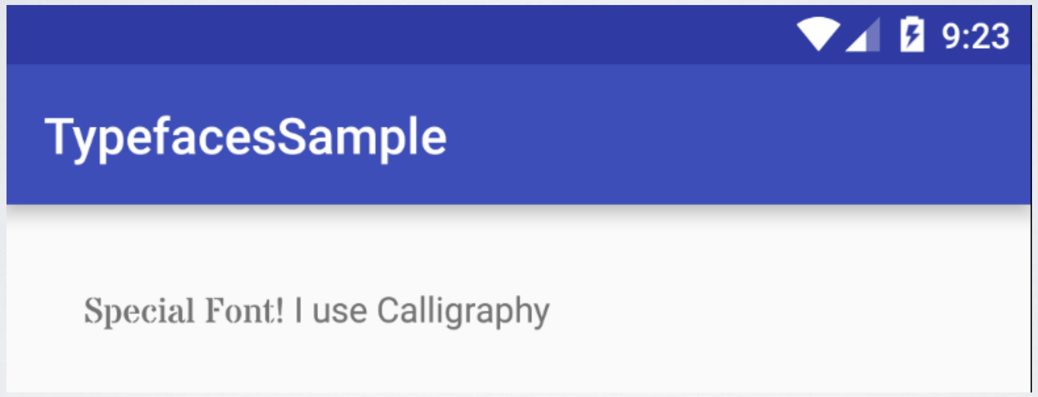


Spannable Typefaces:

```
String specialFont = "Special Font!”;

SpannableStringBuilder sBuilder = new SpannableStringBuilder();
sBuilder
  .append(specialFont) // Bold this
  .append(" I use Calligraphy"); // Default TextView font.

// Create the Typeface Span to apply to the builder.
CalligraphyTypefaceSpan typefaceSpan = new
CalligraphyTypefaceSpan(TypefaceUtils.load(getAssets(),
    "fonts/SpecialFont.ttf"));

// Apply typeface to the Spannable.
sBuilder.setSpan(typefaceSpan, 0, specialFont.length(),
    Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);

spannableText.setText(sBuilder, TextView.BufferType.SPANNABLE);
```

Most important line! BufferType!

Spannable Typefaces:



TypefacesSample

Special Font! I use Calligraphy

Which of course gives you this!

```java
public class CalligraphyTypefaceSpan extends MetricAffectingSpan {
    private final Typeface typeface;
    public CalligraphyTypefaceSpan(final Typeface typeface) {
        //—
        this.typeface = typeface;
    }
    @Override public void updateDrawState(final TextPaint drawState) {
      apply(drawState);
    }
    @Override public void updateMeasureState(final TextPaint paint) {
      apply(paint);
    }
    private void apply(final Paint paint) {
        final Typeface oldTypeface = paint.getTypeface();
        final int oldStyle = oldTypeface != null ?
            oldTypeface.getStyle() : 0;

        final int fakeStyle = oldStyle & ~typeface.getStyle();
        if ((fakeStyle & Typeface.BOLD) != 0) {
            paint.setFakeBoldText(true);
        }
        if ((fakeStyle & Typeface.ITALIC) != 0) {
            paint.setTextSkewX(-0.25f);
        }
        paint.setTypeface(typeface);
    }
}
```

Basically if the current typeface style was bold, apply bold to this one, if it was italic set the new one italic.

```
public class CalligraphyTypefaceSpan extends MetricAffectingSpan {
    private final Typeface typeface;
    public CalligraphyTypefaceSpan(final Typeface typeface) {
        //–
        this.typeface = typeface;
    }
    @Override public void updateDrawState(final TextPaint drawState) {
      apply(drawState);
    }
    @Override public void updateMeasureState(final TextPaint paint) {
      apply(paint);
    }
    private void apply(final Paint paint) {
        final Typeface oldTypeface = paint.getTypeface();
        final int oldStyle = oldTypeface != null ?
            oldTypeface.getStyle() : 0;

        final int fakeStyle = oldStyle & ~typeface.getStyle();
        if ((fakeStyle & Typeface.BOLD) != 0) {
            paint.setFakeBoldText(true);
        }
        if ((fakeStyle & Typeface.ITALIC) != 0) {
            paint.setTextSkewX(-0.25f);
        }
        paint.setTypeface(typeface);
    }
}
```

Basically if the current typeface style was bold, apply bold to this one, if it was italic set the new one italic.

```java
public class CalligraphyTypefaceSpan extends MetricAffectingSpan {
    private final Typeface typeface;
    public CalligraphyTypefaceSpan(final Typeface typeface) {
        //—
        this.typeface = typeface;
    }
    @Override public void updateDrawState(final TextPaint drawState) {
      apply(drawState);
    }
    @Override public void updateMeasureState(final TextPaint paint) {
      apply(paint);
    }
    private void apply(final Paint paint) {
        final Typeface oldTypeface = paint.getTypeface();
        final int oldStyle = oldTypeface != null ?
            oldTypeface.getStyle() : 0;

        final int fakeStyle = oldStyle & ~typeface.getStyle();
        if ((fakeStyle & Typeface.BOLD) != 0) {
            paint.setFakeBoldText(true);
        }
        if ((fakeStyle & Typeface.ITALIC) != 0) {
            paint.setTextSkewX(-0.25f);
        }
        paint.setTypeface(typeface);
    }
}
```

Basically if the current typeface style was bold, apply bold to this one, if it was italic set the new one italic.

```java
public class CalligraphyTypefaceSpan extends MetricAffectingSpan {
    private final Typeface typeface;
    public CalligraphyTypefaceSpan(final Typeface typeface) {
        //—
        this.typeface = typeface;
    }
    @Override public void updateDrawState(final TextPaint drawState) {
      apply(drawState);
    }
    @Override public void updateMeasureState(final TextPaint paint) {
      apply(paint);
    }
    private void apply(final Paint paint) {
        final Typeface oldTypeface = paint.getTypeface();
        final int oldStyle = oldTypeface != null ?
            oldTypeface.getStyle() : 0;

        final int fakeStyle = oldStyle & ~typeface.getStyle();
        if ((fakeStyle & Typeface.BOLD) != 0) {
            paint.setFakeBoldText(true);
        }
        if ((fakeStyle & Typeface.ITALIC) != 0) {
            paint.setTextSkewX(-0.25f);
        }
        paint.setTypeface(typeface);
    }
}
```

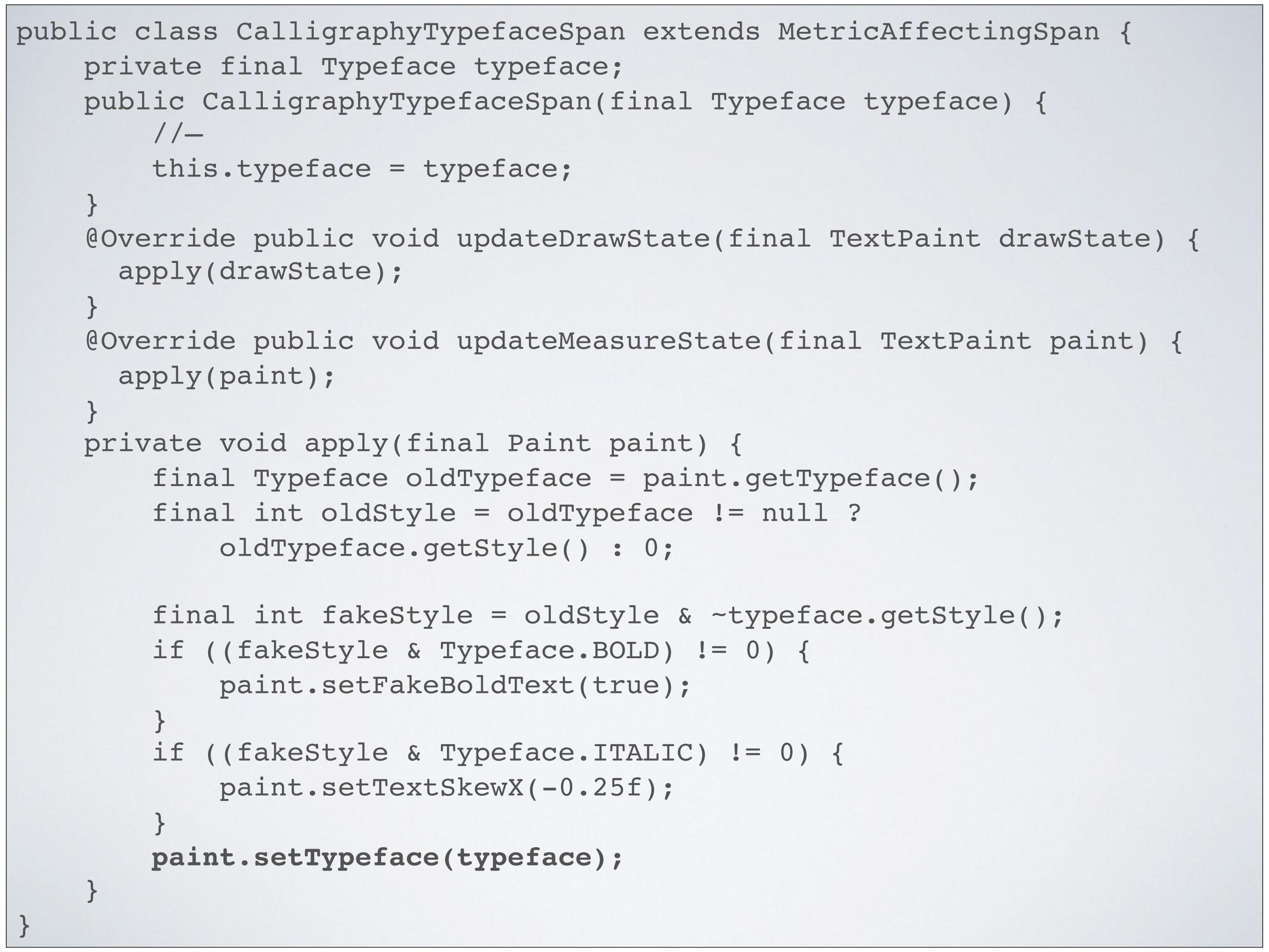
Basically if the current typeface style was bold, apply bold to this one, if it was italic set the new one italic.

```
public class CalligraphyTypefaceSpan extends MetricAffectingSpan {
    private final Typeface typeface;
    public CalligraphyTypefaceSpan(final Typeface typeface) {
        //–
        this.typeface = typeface;
    }
    @Override public void updateDrawState(final TextPaint drawState) {
      apply(drawState);
    }
    @Override public void updateMeasureState(final TextPaint paint) {
      apply(paint);
    }
    private void apply(final Paint paint) {
        final Typeface oldTypeface = paint.getTypeface();
        final int oldStyle = oldTypeface != null ?
            oldTypeface.getStyle() : 0;

        final int fakeStyle = oldStyle & ~typeface.getStyle();
        if ((fakeStyle & Typeface.BOLD) != 0) {
            paint.setFakeBoldText(true);
        }
        if ((fakeStyle & Typeface.ITALIC) != 0) {
            paint.setTextSkewX(-0.25f);
        }
        paint.setTypeface(typeface);
    }
}
```

Basically if the current typeface style was bold, apply bold to this one, if it was italic set the new one italic.

```java
public class CalligraphyTypefaceSpan extends MetricAffectingSpan {
    private final Typeface typeface;
    public CalligraphyTypefaceSpan(final Typeface typeface) {
        //—
        this.typeface = typeface;
    }
    @Override public void updateDrawState(final TextPaint drawState) {
      apply(drawState);
    }
    @Override public void updateMeasureState(final TextPaint paint) {
      apply(paint);
    }
    private void apply(final Paint paint) {
        final Typeface oldTypeface = paint.getTypeface();
        final int oldStyle = oldTypeface != null ?
            oldTypeface.getStyle() : 0;

        final int fakeStyle = oldStyle & ~typeface.getStyle();
        if ((fakeStyle & Typeface.BOLD) != 0) {
            paint.setFakeBoldText(true);
        }
        if ((fakeStyle & Typeface.ITALIC) != 0) {
            paint.setTextSkewX(-0.25f);
        }
        paint.setTypeface(typeface);
    }
}
```

Basically if the current typeface style was bold, apply bold to this one, if it was italic set the new one italic.



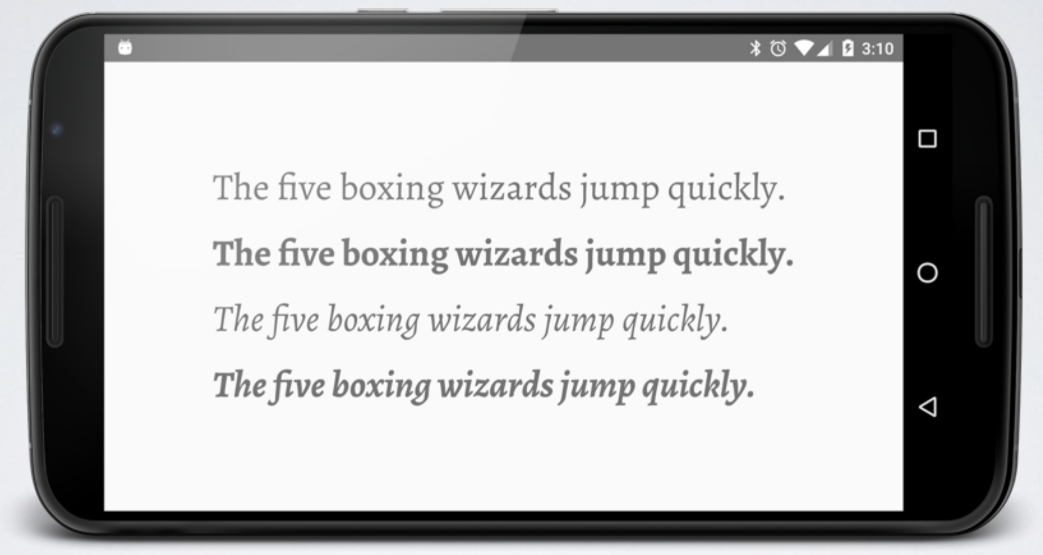
```java
public class CalligraphyTypefaceSpan extends MetricAffectingSpan {
    private final Typeface typeface;
    public CalligraphyTypefaceSpan(final Typeface typeface) {
        //–
        this.typeface = typeface;
    }
    @Override public void updateDrawState(final TextPaint drawState) {
      apply(drawState);
    }
    @Override public void updateMeasureState(final TextPaint paint) {
      apply(paint);
    }
    private void apply(final Paint paint) {
        final Typeface oldTypeface = paint.getTypeface();
        final int oldStyle = oldTypeface != null ?
            oldTypeface.getStyle() : 0;

        final int fakeStyle = oldStyle & ~typeface.getStyle();
        if ((fakeStyle & Typeface.BOLD) != 0) {
            paint.setFakeBoldText(true);
        }
        if ((fakeStyle & Typeface.ITALIC) != 0) {
            paint.setTextSkewX(-0.25f);
        }
        paint.setTypeface(typeface);
    }
}
```

Basically if the current typeface style was bold, apply bold to this one, if it was italic set the new one italic.

End of the story. Calligraphy works, I use it as do many people and it does enough.

What else?

DataBinding:



The five boxing wizards jump quickly.

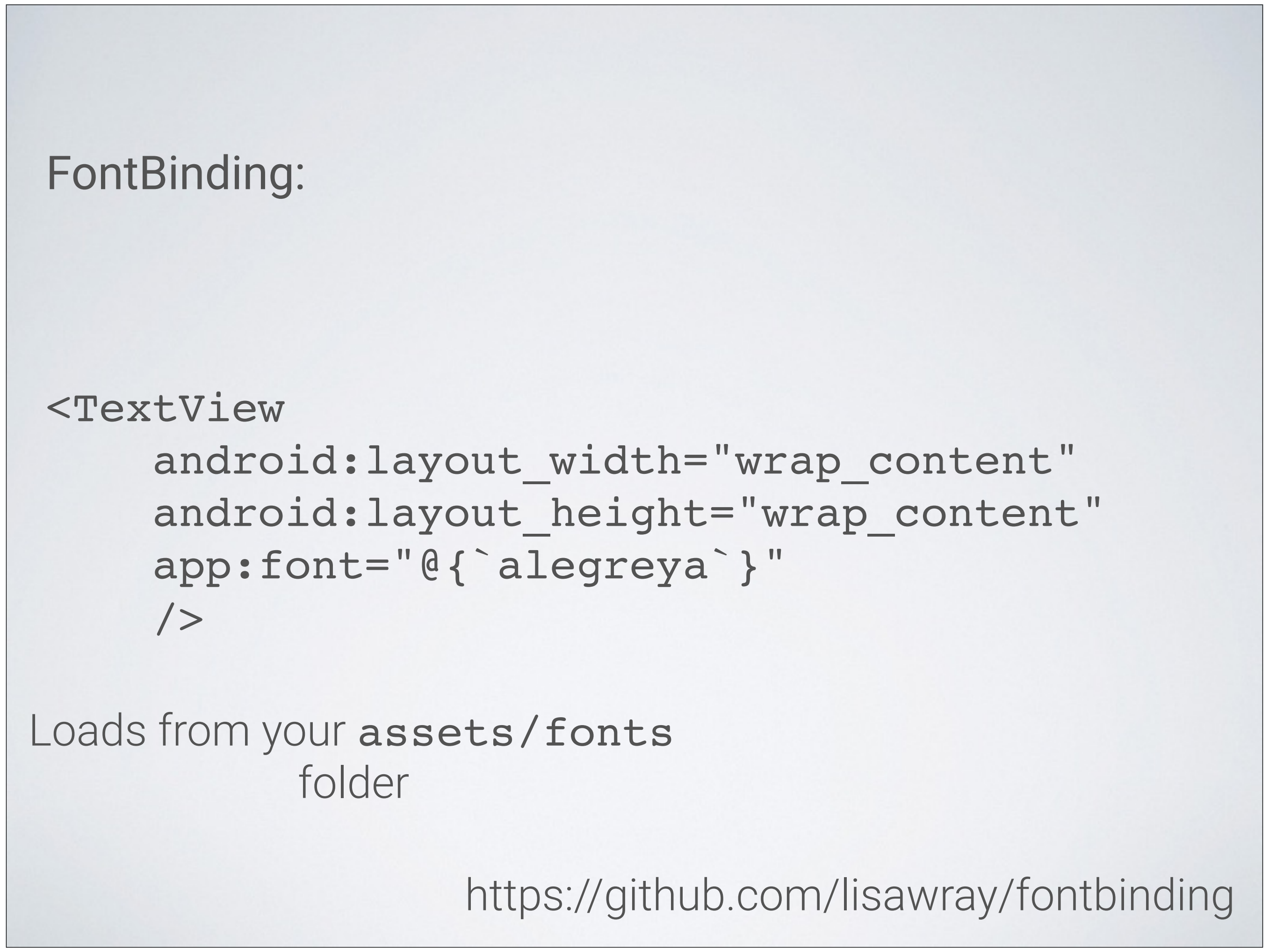
The five boxing wizards jump quickly.

The five boxing wizards jump quickly.

The five boxing wizards jump quickly.

https://github.com/lisawray/fontbinding

Shout out to Lisa Wray.



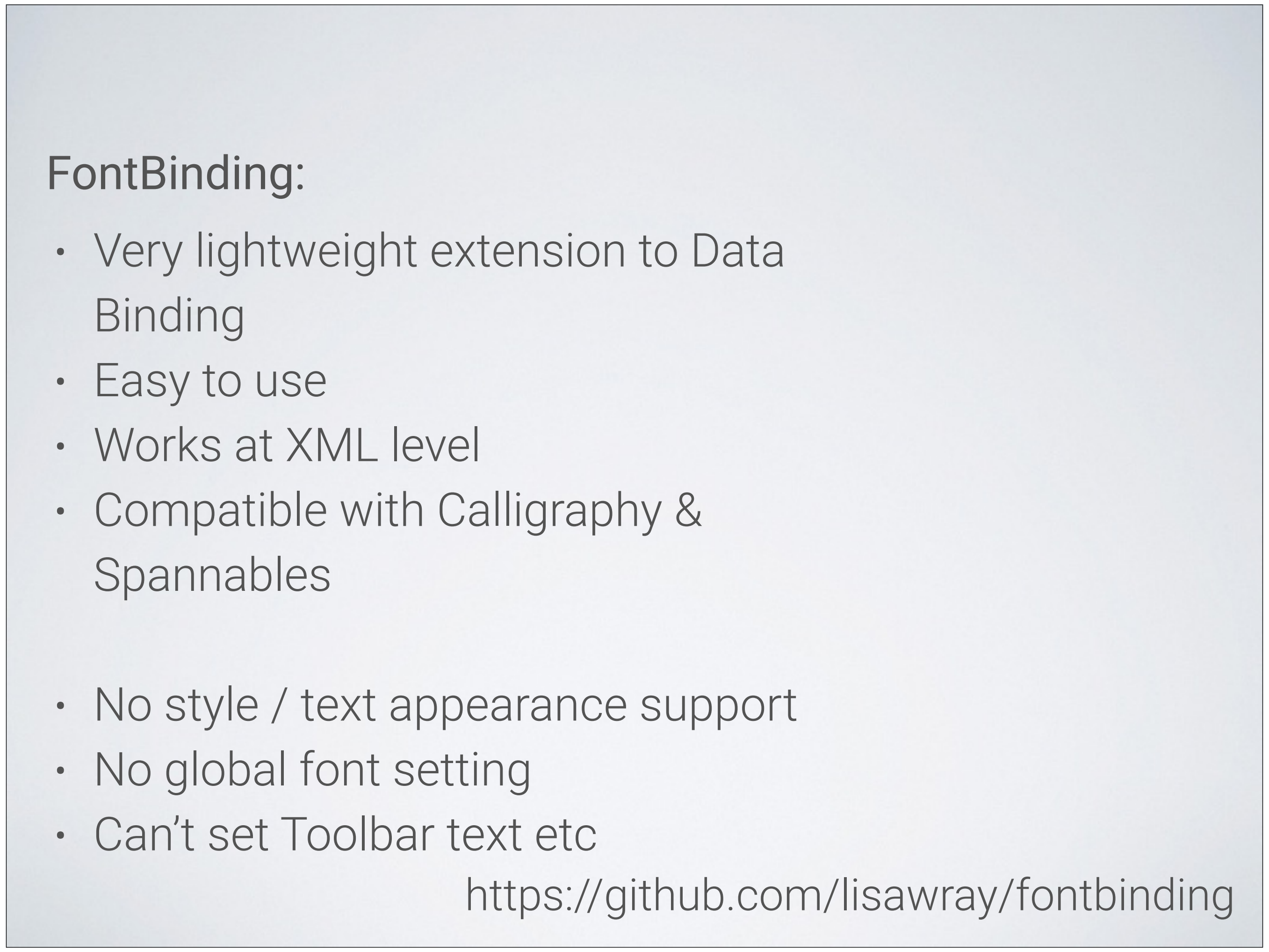
FontBinding:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:font="@{`alegreya`}"
    />
```

Loads from your `assets/fonts` folder

https://github.com/lisawray/fontbinding

Shout out to Lisa Wray. This will find the fonts inside your assets/fonts.

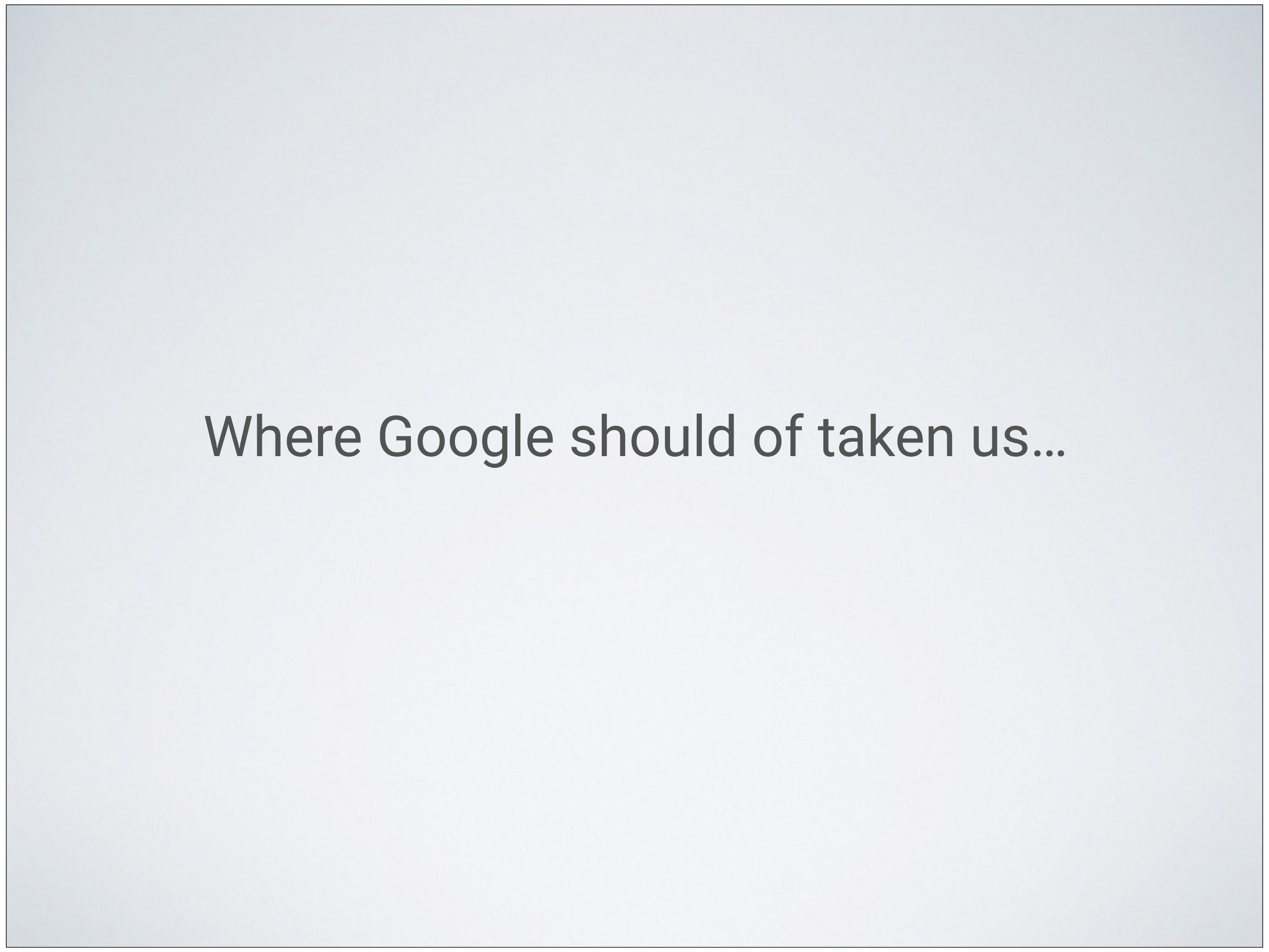

FontBinding:

- Very lightweight extension to Data Binding
- Easy to use
- Works at XML level
- Compatible with Calligraphy & Spannables

- No style / text appearance support
- No global font setting
- Can't set Toolbar text etc

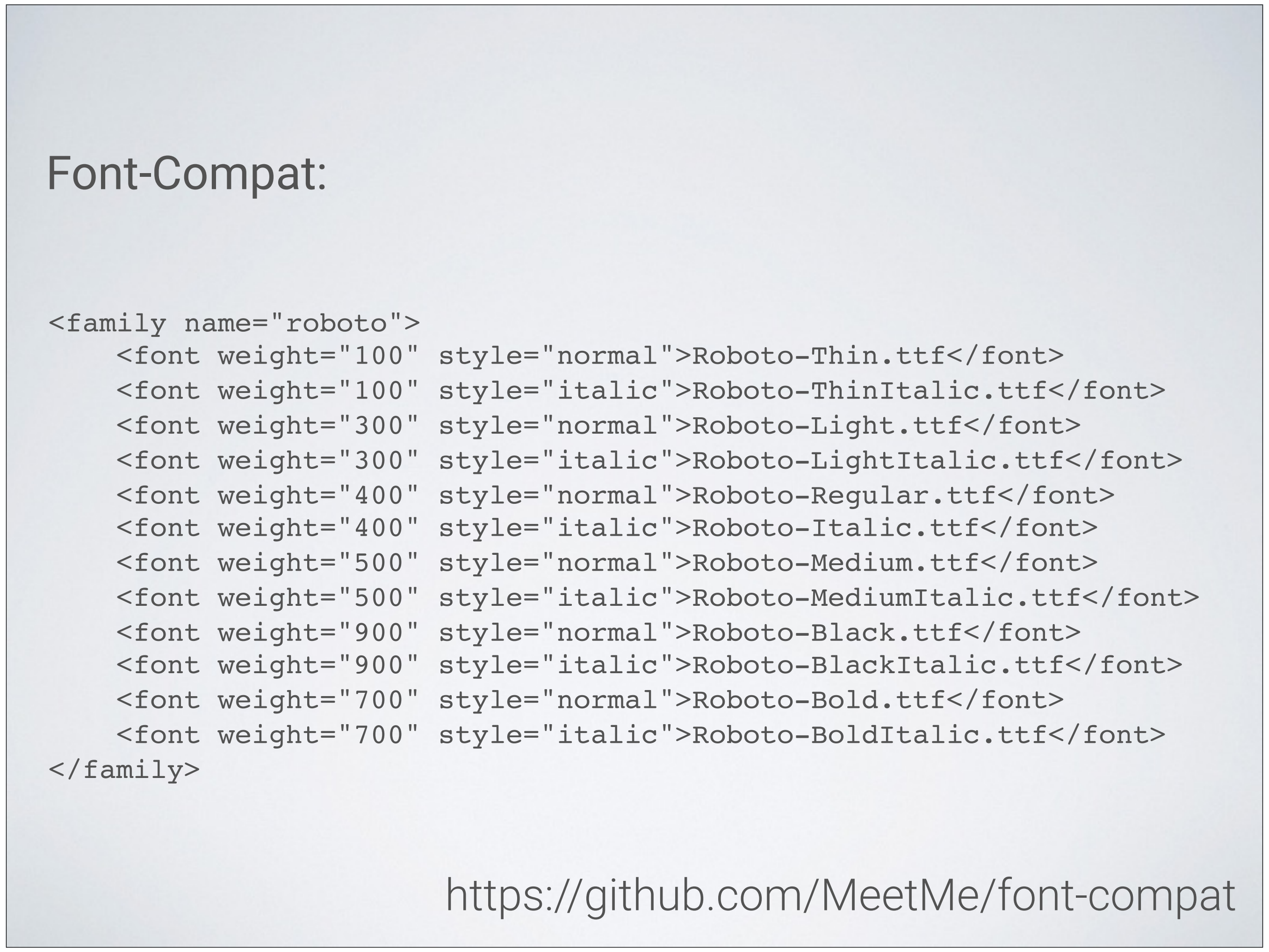
https://github.com/lisawray/fontbinding

Some pros and cons



Where Google should of taken us…

Calligraphy is trying to do what google failed to do.
The next lib is very close.

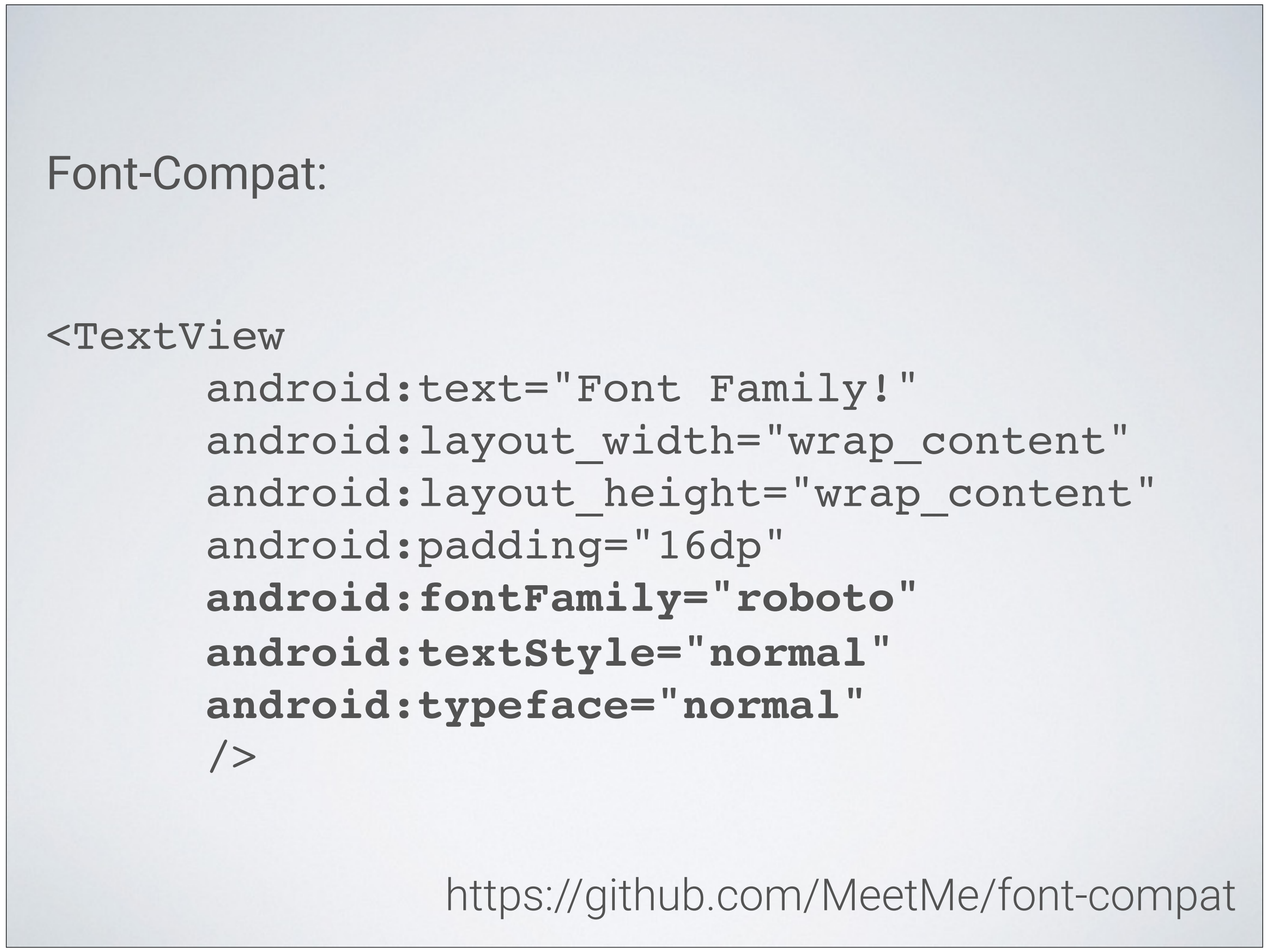


Font-Compat:

```
<family name="roboto">
    <font weight="100" style="normal">Roboto-Thin.ttf</font>
    <font weight="100" style="italic">Roboto-ThinItalic.ttf</font>
    <font weight="300" style="normal">Roboto-Light.ttf</font>
    <font weight="300" style="italic">Roboto-LightItalic.ttf</font>
    <font weight="400" style="normal">Roboto-Regular.ttf</font>
    <font weight="400" style="italic">Roboto-Italic.ttf</font>
    <font weight="500" style="normal">Roboto-Medium.ttf</font>
    <font weight="500" style="italic">Roboto-MediumItalic.ttf</font>
    <font weight="900" style="normal">Roboto-Black.ttf</font>
    <font weight="900" style="italic">Roboto-BlackItalic.ttf</font>
    <font weight="700" style="normal">Roboto-Bold.ttf</font>
    <font weight="700" style="italic">Roboto-BoldItalic.ttf</font>
</family>
```

https://github.com/MeetMe/font-compat

This almost hits the nail on the head. Starts of be defining a font family

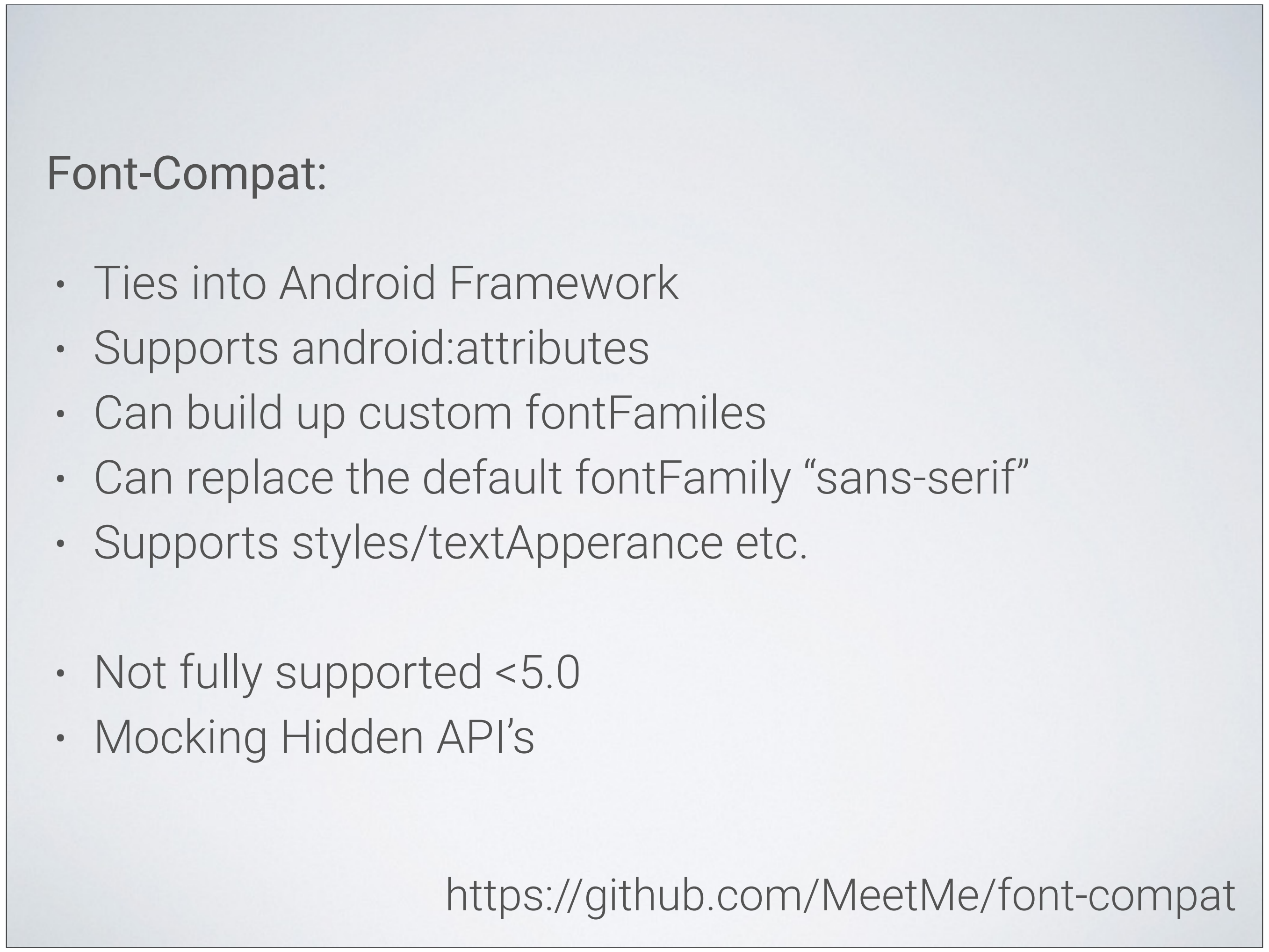


Font-Compat:

```
<TextView
      android:text="Font Family!"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:padding="16dp"
      android:fontFamily="roboto"
      android:textStyle="normal"
      android:typeface="normal"
      />
```

https://github.com/MeetMe/font-compat

Can use fontFamily as you would normally!



Font-Compat:

- Ties into Android Framework
- Supports android:attributes
- Can build up custom fontFamiles
- Can replace the default fontFamily “sans-serif”
- Supports styles/textApperance etc.

- Not fully supported <5.0
- Mocking Hidden API’s

https://github.com/MeetMe/font-compat

Some pros and cons

QUESTIONS?