LayoutInflater

A dive in

Who am I?

- CTO of @OWLR
- Android Engineer 6+ years
  - Met Office
  - Yoyo Wallet
- Social Stuff:
  - Web: http://chrisjenx.com
  - Twitter: @chrisjenx

Whats the Story?
物語は何ですか?

Winter 2013

I wanted to do this:

```
<TextView
   font="RobotoBold.ttf"
   />
```

And this:

```xml
<style name="MyTheme">
  <!-- Default Font -->
  <item name="font">Roboto-Regular.ttf</item>
</style>
```

Stuck with this:

```
<com.myapp.RobotoBoldTextView />
```

```
public class RobotoBoldTextView extends TextView {
  public RobotoTextView(Context context, @Nullable AttributeSet attrs) {
    super(context, attrs);
    setTypeface(Typeface.createFromFile("assets/fonts/Roboto-Bold.ttf"));
  }
}
```

Problems?

- Difficult to extend (Now can't extend `AppCompat !extend RobotoTextView`)
- Definitely not composeable
- Difficult to change/update
- Mixing design with implementation detail.

Custom Attributes?

```
<declare-styleable name="CustomTextView">
  <attr name="font" format="string" />
</declare-styleable>
```

```
public class CustomTextView extends TextView {
  public CustomTextView(Context context,
      @Nullable AttributeSet attrs, int defStyle) {
    super(context, attrs);
    TypedArray a = context.obtainStyledAttributes(
        attrs, R.styleable.CustomTextView, defStyle, 0);
    setTypeface(Typeface.createFromFile(
        a.getString(R.styleable.CustomTextView_font)));
    a.recycle();
  }
}
```

Same Problems

- Difficult to extend (Still can't extend. `AppCompat !extend CustomTextView`)
- Still not composeable

Is this possible?

```
<AnyTextView
   font="RobotoBold.ttf"
   />
```

<u>Calligraphy</u> was born.

LayoutInflation Injection

What? 何!

Layout Inflation

- First, How do we use it.

Layout Inflation

- ~~First, How do we use it.~~
- First, How do we use it, **correctly**.
- How does it work? どのように機能するのですか?
- Lets "ViewPump" - えっ?!

Using the LayoutInflater, correctly!

The **only** way to get the `LayoutInflater`:

```
LayoutInflater inflater =
    LayoutInflater.from(Context context);
```

Using the LayoutInflater, correctly!

The ~~only~~ **safest** way to get an instance:

```
LayoutInflater inflater =
    LayoutInflater.from(Context context);
```

Using the LayoutInflater, correctly!

The ~~only~~ **safest** way to get an instance:

```
LayoutInflater inflater =
    LayoutInflater.from(Context context);
```

It does this:

```
(LayoutInflater) context
      .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
```

Let's be clear!

Unsafe: `getActivity()` or `getLayoutInflater()`.

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {
    return getActivity().getLayoutInflater()
        .inflate(fragment_order_list, container, false);
}
```

Correct:

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {
    return inflater
        .inflate(fragment_order_list, container, false);
}
```

But, You just said!

Use:

```
LayoutInflater.from(Context context);
```

Correct, this is the only known exception.

```
Fragment::onCreateView(LayoutInflater inflater, ...)
```

Examples

MyListAdapter.java

```
@Override
public View getView(int position, View convertView,
    ViewGroup parent) {
  // Use context directly above us. i.e "Parent"
  LayoutInflater inflater =
      LayoutInflater.from(parent.getContext());
  //...
  View view = inflater
      .inflate(R.layout.my_list_item, parent, false);
  return view;
}
```

Examples Cont.

RecycleView.MyAdapter.java

```
@Override
public MyAdapter.ViewHolder onCreateViewHolder(
    ViewGroup parent, int viewType) {
  // Use context directly above us. i.e. "Parent"
  View v = LayoutInflater.from(parent.getContext())
      .inflate(R.layout.my_text_view, parent, false);
  //...
  return new ViewHolder(v);
}
```

Examples Cont.

MyView.java

```
public MyView(Context context) {
  LayoutInflater.from(context, this, true);
}
```

Why is this important?

Two examples:

```
class MyApplication extends Application {
  public void onCreate() {
    // Application LayoutInflater
    LayoutInflater.from(MyApplication.this)
  }
}

class MyFirstActivity extends Activity {
  public void onCreate(Bundle savedInstanceState) {
    // Activity LayoutInflater
    LayoutInflater.from(MyFirstActivity.this)
  }
}
```

Application Context and Theme

```
class MyApplication extends Application {
  public void onCreate() {
    // Application LayoutInflater
    LayoutInflater.from(MyApplication.this)
  }
}
```

Uses:

```
<manifest>
<application
  android:theme="@android:style/Theme.DeviceDefault.Light"
  >
</manifest>
```

Vs. Activity

```java
class MyDialogActivity extends Activity {
  public void onCreate(Bundle savedInstanceState) {
    LayoutInflater.from(MyDialogActivity.this) // Activity LayoutInflater
  }
}
```

Uses:

```xml
<manifest>
  <activity
    android:theme="@android:style/Theme.DeviceDefault.Dialog" />
</manifest>
```

`LayoutInflater.from(application)` will use `Theme.DeviceDefault.Light`!

But, I use the same theme.

But, I use the same theme.

You do now, but future proof.

We can also extend `Contexts`

```
<FrameLayout
    xmlns:android="..."
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:theme="@style/Theme.DeviceDefault.Overlay">
  <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
</FrameLayout>
```

Extend `Context` **not** `Theme`**?**

Everything is a `ContextWrapper`!

- `Application extends ContextWrapper`
- `Activity extends ContextThemeWrapper`
- `ContextThemeWrapper extends ContextWrapper`

Creating `ContextThemeWrappers`

```
<FrameLayout
    android:theme="@style/Theme.DeviceDefault.Overlay">
```

Simple equivalent:

```
Context context = parent.getContext().
if(hasThemeAttr) {
  int themeRes = attrs.getResourceId(THEME_ATTR, 0);
  context = new ContextThemeWrapper(parent, themeRes);
}
view = new FrameLayout(context, attrs);
```

FrameLayout and children will now get styled attributes from `Theme.DeviceDefault.Overlay`

Overkill?

Last week:

This issue depends on the Context used to get the LayoutInflater instance. In My PagerAdapter I was injecting Context via Dagger2 which was ApplicationContext', and I instantiated LayoutInflater in constructor using thatContext' and Calligraphy had no impact there.

LayoutInflater
`attachToRoot` / `parent`

```
inflate(layoutResource,
 parent, attachToRoot)
```

`parent`

@parent A view group that will be the parent. Used to properly inflate the layout_* parameters.

parent

```
<LinearLayout android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical">
  <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:weight="1"/>
</LinearLayout>

inflater.inflate(layoutRes, parent, false);
// Does this:
viewGroup = createView(activity.context, ...); // FrameLayout
view = createView(viewGroup.context, ...) // TextView
// Attrs only available at inflation time!
params = viewGroup.generateLayoutParams(viewAttrs);
// Can now tell parent how to be laid out.
view.setLayoutParams(params);
```

attachToRoot

```
inflater.inflate(R.layout.activity, parent, true);

// Does this:

// FrameLayout
viewGroup = createView(activity.context, ...);
// TextView
view = createView(viewGroup.context, ...)
// Attrs only available at inflation time!
params = viewGroup.generateLayoutParams(viewAttrs);
// Add to parent and layout.
parent.addView(view, params);
```

How to use attachToRoot

```
// ListViewAdapter.java

public View getView(int position, View convertView,
      ViewGroup parent) {
  LayoutInflater inflater = LayoutInflater.from(convertView)
  // Inflate Row, parent == convertView (ListView), do not attach.
  View rowView = inflater
    .inflate(R.layout.rowlayout, parent, false);
  //..
  return rowView;
}
```

How to use `attachToRoot`

```java
// CustomLayout.java

public class CustomLayout extends FrameLayout {
  public CustomLayout(Context context,
        AttributeSet attrs, int defStyle) {
    super(context, attrs);
    // inflate inner layout, parent == this, attach == true.
    LayoutInflater.from(context)
      .inflate(R.layout.custom_layout_include, this, true);
  }
}
```

How to use attachToRoot

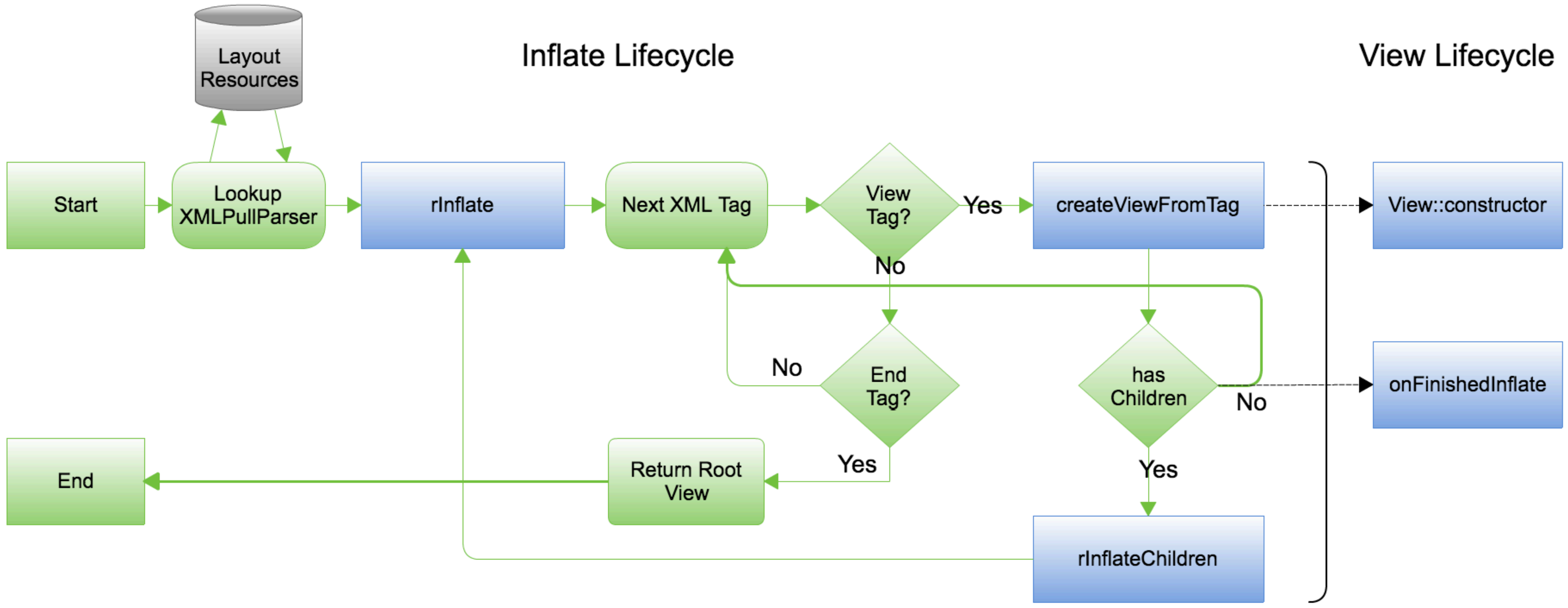
```
// MyFragment.java:
public View onCreateView(LayoutInflater inflater,
    ViewGroup container, Bundle savedInstanceState) {

  // Inflate the layout for this fragment, don't attach
  return inflater
    .inflate(R.layout.article_view, container, false);
}
```

How does it work?

Compile

Layout XML -> AAPT -> Optimised Layout xml -> .apk

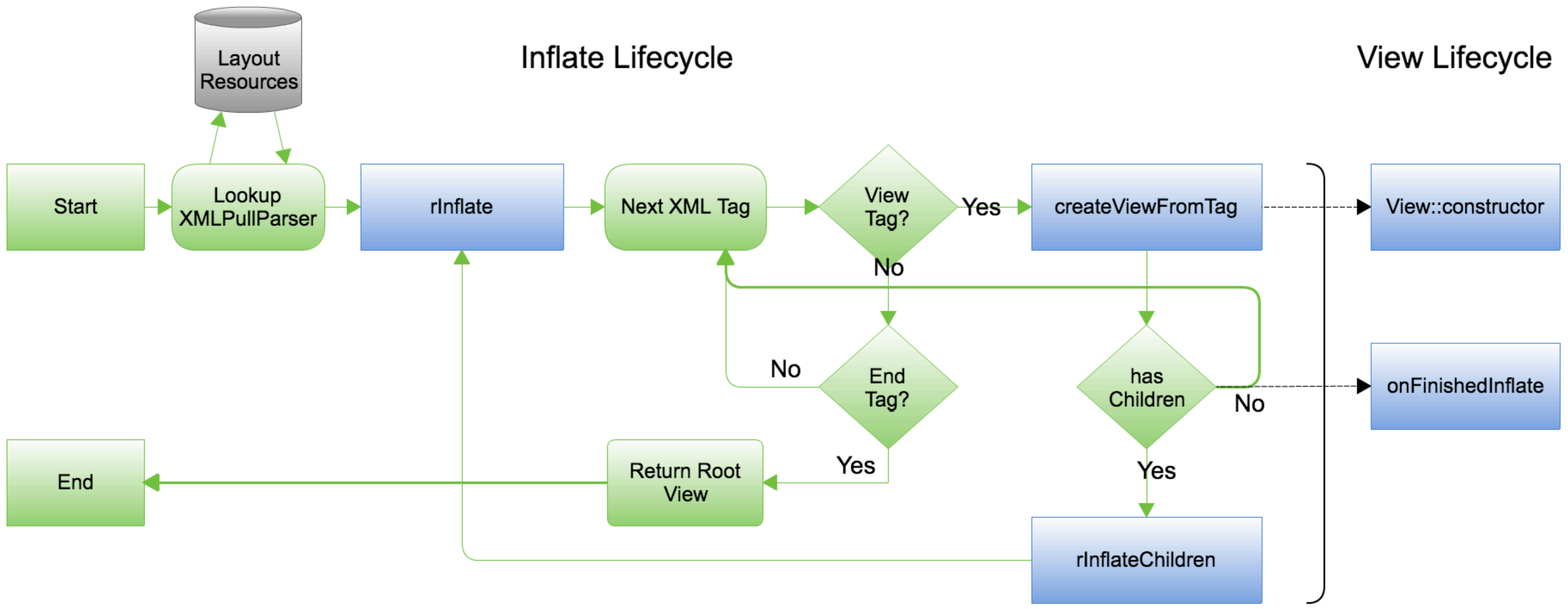
Layout files can not be read by a custom XMLPullParser at runtime.


Layout Resources
Inflate Lifecycle
View Lifecycle
Start
Lookup XMLPullParser
rInflate
Next XML Tag
View Tag?
Yes
createViewFromTag
View::constructor
No
No
End Tag?
has Children
No
onFinishedInflate
End
Return Root View
Yes
Yes
rInflateChildren

Example Inflation Flow

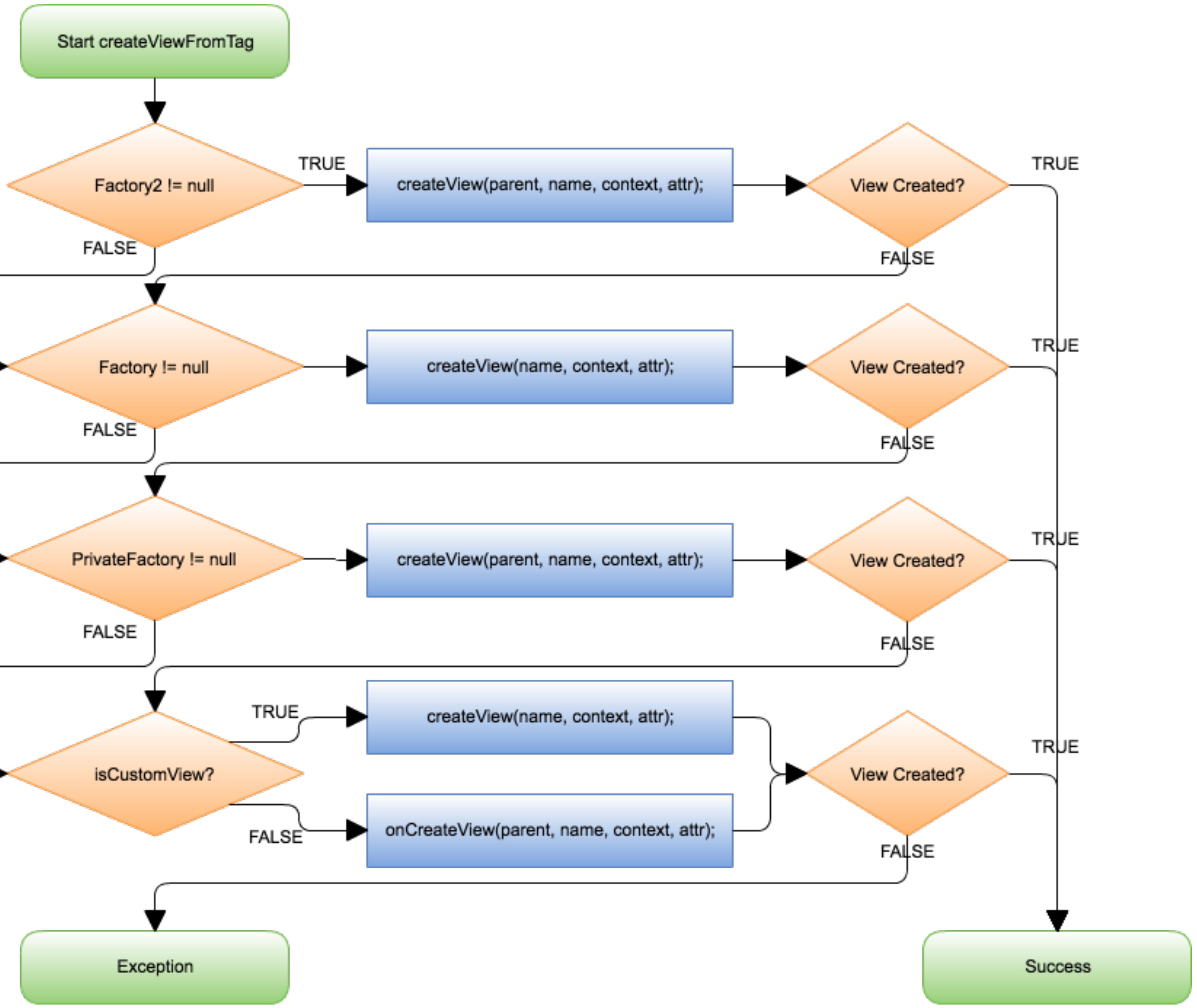
```
<FrameLayout>
  <TextView />
  <ImageView />
</FrameLayout>
```

```
//1. Inflates root view
root = new FrameLayout(...);
//2. look for children
view = new TextView(...);
//3. has children?
view.onFinishedInflate();
//4. next tag
view = new ImageView(...);
//5. has children?
view.onFinishedInflate();
//6. next tag? end? step out of recursion.
root.onFinishedInflate();
// next tag? end? return
return root;
```


Inflate Lifecycle
View Lifecycle
Layout Resources
Start
Lookup XMLPullParser
rInflate
Next XML Tag
View Tag?
Yes
No
createViewFromTag
View::constructor
End Tag?
No
Yes
has Children
No
Yes
onFinishedInflate
End
Return Root View
rInflateChildren

createViewFromTag

Things get weird. 物事は奇妙になる


Start createViewFromTag
Factory2 != null
TRUE
FALSE
createView(parent, name, context, attr);
View Created?
TRUE
FALSE
Factory != null
FALSE
createView(name, context, attr);
View Created?
TRUE
FALSE
PrivateFactory != null
FALSE
createView(parent, name, context, attr);
View Created?
TRUE
FALSE
isCustomView?
TRUE
FALSE
createView(name, context, attr);
onCreateView(parent, name, context, attr);
View Created?
TRUE
FALSE
Exception
Success

`createView*` parameters

```
<FrameLayout android:background="@drawable/background">
  <TextView android:text="@string/my_text"/>
  <com.example.MyTextView android:text="@string/my_text" />
</FrameLayout>

// Creating the FrameLayout
createView(parent, name, context, attrs);
// Would be equivalent to:
createView(
    null,
    "FrameLayout",
    activity.getContext(),
    new AttributeSet(){ background = "@drawable/background" }
)
```

createView* parameters

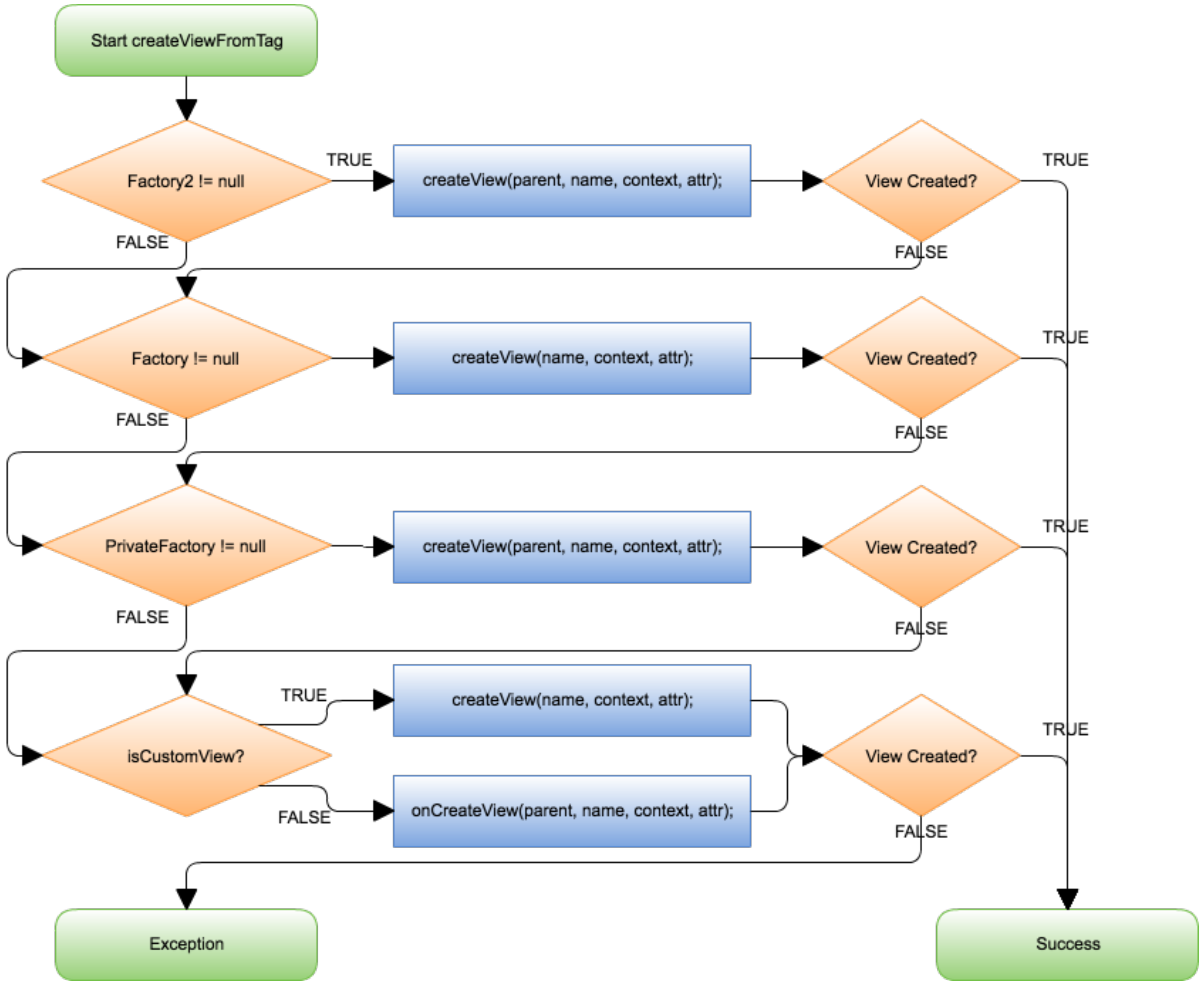
```
<FrameLayout android:background="@drawable/background">
  <TextView android:text="@string/my_text"/>
  <com.example.MyTextView android:text="@string/my_text" />
</FrameLayout>
```

```
// Creating the TextView
createView(parent, name, context, attrs);
// Would be equivalent to:
createView(
    FrameLayout::this,
    "TextView",
    FrameLayout::getContext(),
    new AttributeSet(){ text = "@string/my_text" }
)
```

createView* parameters

```
<FrameLayout android:background="@drawable/background">
  <TextView android:text="@string/my_text"/>
  <com.example.MyTextView android:text="@string/my_text" />
</FrameLayout>
```

```
// Creating the com.example.MyTextView
createView(parent, name, context, attrs);
// Would be equivalent to:
createView(
    FrameLayout::this,
    "com.example.MyTextView",
    FrameLayout::getContext(),
    new AttributeSet(){ text = "@string/my_text" }
)
```


Start createViewFromTag
Factory2 != null
TRUE
createView(parent, name, context, attr);
View Created?
TRUE
FALSE
FALSE
Factory != null
createView(name, context, attr);
View Created?
TRUE
FALSE
FALSE
PrivateFactory != null
createView(parent, name, context, attr);
View Created?
TRUE
FALSE
FALSE
TRUE
createView(name, context, attr);
isCustomView?
View Created?
TRUE
FALSE
onCreateView(parent, name, context, attr);
FALSE
Exception
Success

What are the Factories?

Hook you can supply that is called when inflating from a LayoutInflater.
You can use this to customize the tag names available in your XML layout files.

"Customize the tag names?"

Hook you can supply that is called when inflating from a LayoutInflater.
You can use this to customize the tag names available in your XML layout files.

- This is misleading. 誤解を招く

Means this?

```
public String createView(name) {
  return  "com.chrisjenx." + name;
}
```

No. いいえ

Factory definition.

A hook into the LayoutInflater that intercepts tag names from your layout files.
Then return null or a View you have created for that Tag Name.

Factory2

```
// MyActivity.java:
public void onCreate(Bundle savedInstanceState) {
  LayoutInflater inflater = LayoutInflater.from(this);
  inflater.setFactory2(new LayoutInflater.Factory2() {
      @Override
      public View onCreateView(View parent, String name,
            Context context, AttributeSet attrs) {
        if("FrameLayout".equals(name))
          return new FrameLayout(context);
        return null;
      }
      @Override public View onCreateView(String name, Context context,
            AttributeSet attrs) {
        if("FrameLayout".equals(name))
          return new FrameLayout(context);
        return null;
      }
    });
}
```

Factory

```
// MyActivity.java:

public void onCreate(Bundle savedInstanceState) {
  LayoutInflater inflater = LayoutInflater.from(this);
  inflater.setFactory(new LayoutInflater.Factory() {
    @Override public View onCreateView(String name, Context context,
          AttributeSet attrs) {
      if("FrameLayout".equals(name)) return new FrameLayout(context);
      return null;
    }
  });
}
```

Which one should I use?

Use Factory2, or better:

```
// MyActivity.java
public void onCreate(Bundle savedInstanceState) {
  LayoutInflater inflater = LayoutInflater.from(this);
  LayoutInflaterCompat.setFactory(inflater,
    new LayoutInflaterFactory() {
      @Override
      public View onCreateView(View parent, String name,
            Context context, AttributeSet attrs) {
        // Wraps both depending on API Level.
        return null;
      }
  });
}
```

PrivateFactory?

PrivateFactory

Implements `LayoutInflater.Factory2` **interface.**

Activity == PrivateFactory

```
class MyActivity extends Activity {
  @Override
  public View onCreateView(View parent, String name,
        Context context, AttributeSet attrs) {
    return super.onCreateView(parent, name, context, attrs);
  }

  @Override public View onCreateView(String name,
        Context context, AttributeSet attrs) {
    return super.onCreateView(name, context, attrs);
  }
}
```

Activity == PrivateFactory

```
// android Activity.java source

public class Activity extends ContextThemeWrapper
        implements LayoutInflater.Factory2, //...
```

`LayoutInflater.from(context)`. Context is important!

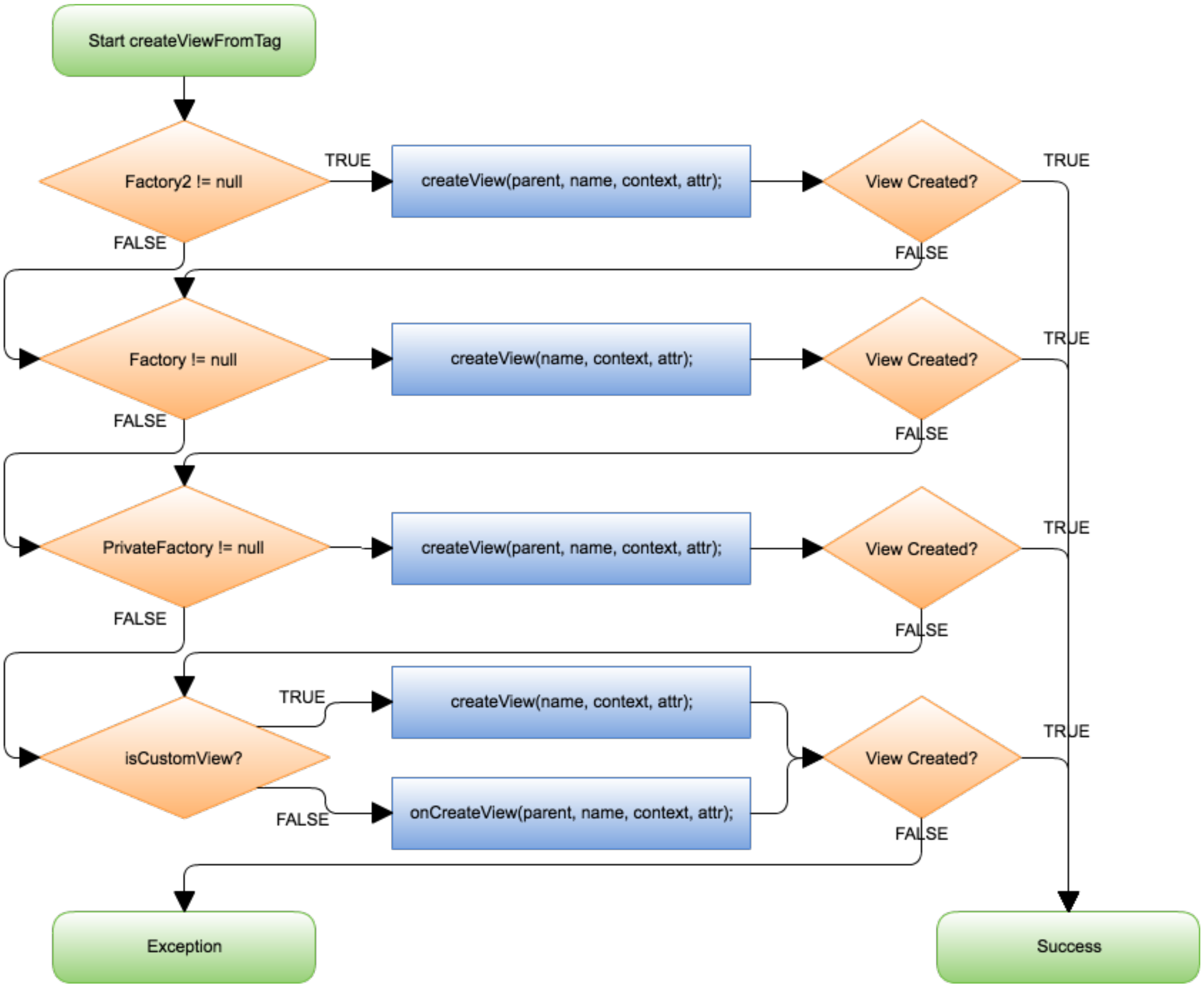
Fragments

```
<!-- Fragments From XML -->
<fragment
      android:id="@+id/fragment_one"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      class="com.example.fragments.MyFragmentOne" />

// FragmentActivity.java slice
abstract class FragmentActivity {
  @Override
  public View onCreateView(View parent, String name,
        Context context, AttributeSet attrs) {
    // Is this Tag a fragment?
    if (!"fragment".equals(name)) return null;
    Fragment fragment = // createFragment from attr[class]
    // Call fragment onCreateView logic
    fragment.onInflate(FragmentActivity::this,
          attrs, fragment.savedInstanceState);
    return fragment.mView;
  }
}
```

More importantly, AppCompat

AppCompatActivity

```
// AppCompatActivity simplified (alot).
public final View createView(View parent, final String name,
        @NonNull Context context, @NonNull AttributeSet attrs) {
  View view = null;
  switch (name) {
     case "TextView":
         view = new AppCompatTextView(context, attrs);
         break;
     case "ImageView":
         view = new AppCompatImageView(context, attrs);
         break;
     //... All AppCompatViews
  }
  return view;
}
```


Start createViewFromTag
Factory2 != null
TRUE
FALSE
createView(parent, name, context, attr);
View Created?
TRUE
FALSE
Factory != null
FALSE
createView(name, context, attr);
View Created?
TRUE
FALSE
PrivateFactory != null
FALSE
createView(parent, name, context, attr);
View Created?
TRUE
FALSE
isCustomView?
TRUE
FALSE
createView(name, context, attr);
onCreateView(parent, name, context, attr);
View Created?
TRUE
FALSE
Exception
Success

LayoutInflater has `onCreateView`/`createView`

Android Views -> `onCreateView()`

e.g. `<FrameLayout />`

Custom views -> `createView()`

e.g. `<com.example.CustomView />`

To be exact

If there is a `.` in the class name:

```
if (-1 == name.indexOf('.')) {
    view = onCreateView(parent, name, attrs);
} else {
    view = createView(name, null, attrs);
}
```

Why?

- I don't know.

What does `createView()` do?

Reflection.

```
View createView(String name, String prefix, AttributeSet attrs) {
  //..
  clazz = mContext.getClassLoader()
      .loadClass(prefix != null ? (prefix + name) : name)
      .asSubclass(View.class);
  //..
  constructor = clazz.getConstructor(mConstructorSignature);
  //..
  return constructor.newInstance(mConstructorArgs);
}
```

Recap:

How do I do this?

```
<TextView
   font="RobotoBold.ttf"
   />
```

Is there a callback like this?

```
View onViewCreated(View view, AttributeSet attrs);
```

You can't and No.

Enter Calligraphy

Goal, create:

- Enable font injection everywhere.
- Expose `onViewCreated(View view, AttributeSet atts)`

Calligraphy v1 and v2

Enabled:

```
<TextView
  font="RobotoBold.ttf"
  />

<style name="MyTheme">
  <!-- Default Font -->
  <item name="font">Roboto-Regular.ttf</item>
</style>
```

Calligraphy

- Stable but still flawed.
- Limited functionally, only handles fonts.
- Wanted to expose: `onViewCreated(View view, AttributeSet atts)`

Goal, create:

- ~~Enable font injection everywhere.~~
- Expose `onViewCreated(View view, AttributeSet atts)`

ViewPump was born.

ViewPump & Calligraphy

- Interceptor based View Inflation. (Based on Square OKHttp Interceptors)
- James Barr, Twitter: @jbarr21
- Took Calligraphy and turned it into ViewPump
- See InflationX/ViewPump
- See InflationX/Calligraphy
- Still pre-release (1.0.0-SNAPSHOT).

Goal, create:

- ~~Enable font injection everywhere.~~
- ~~Expose `onViewCreated(View view, AttributeSet atts)`~~

ViewPump = LayoutInflation Pre/Post Hooks

```
// Post Inflation Hook

public class TextUpdatingInterceptor implements Interceptor {
  @Override
  public InflateResult intercept(Chain chain) {
    InflateResult result = chain.proceed(chain.request());
    if (result.view() instanceof TextView) {
        // Do something to result.view()
        // You have access to result.context() and result.attrs()
        TextView textView = (TextView) result.view();
        textView.setText("[Prefix] " + textView.getText());
    }
    return result;
  }
}
```

ViewPump = LayoutInflation Pre/Post Hooks

```java
// Pre Inflation Hook
public class CustomTextViewInterceptor implements Interceptor {
  private View inflateView(String name, Context context, AttributeSet attrs) {
    if(name.contains("TextView")) return new CustomTextView(context, attr);
    return null;
  }
  @Override
  public InflateResult intercept(Chain chain) {
    InflateRequest request = chain.request();
    View view = inflateView(request.name(), request.context(), request.attrs());
    if (view != null) {
        return InflateResult.builder()
                .view(view)
                .name(view.getClass().getName())
                .context(request.context())
                .attrs(request.attrs())
                .build();
    } else {
        return chain.proceed(request);
    }
  }
}
```

ViewPump = LayoutInflation Pre/Post Hooks

```
ViewPump.init(ViewPump.builder()
    .addInterceptor(new TextUpdatingInterceptor()) // Post Inflation
    .addInterceptor(new CustomTextViewInterceptor()) // Pre Inflation
    .build());
```

- Downwards for Post Inflation
- Upwards for Pre Inflation

How do we do that?

First iteration

```
class CalligraphyLayoutInflater extends LayoutInflater {
  //..
  @Override
  protected View onCreateView(String name, AttributeSet attrs)
        throws ClassNotFoundException {

      for (String prefix : sClassPrefixList) {
          try {
              View view = createView(name, prefix, attrs);
              if (view != null) {
                  interceptView(view, name, attrs);
                  return view;
              }
          } catch (ClassNotFoundException e) {
              // In this case we want to let the base class
              // take a crack at it.
          }
      }
      return super.onCreateView(name, attrs);
  }
  //..
}
```

Worked; badly.

- Only ever intercepted non-custom views
- Would never get intercepted if the view was created by another factory. e.g. AppCompat

Second iteration.

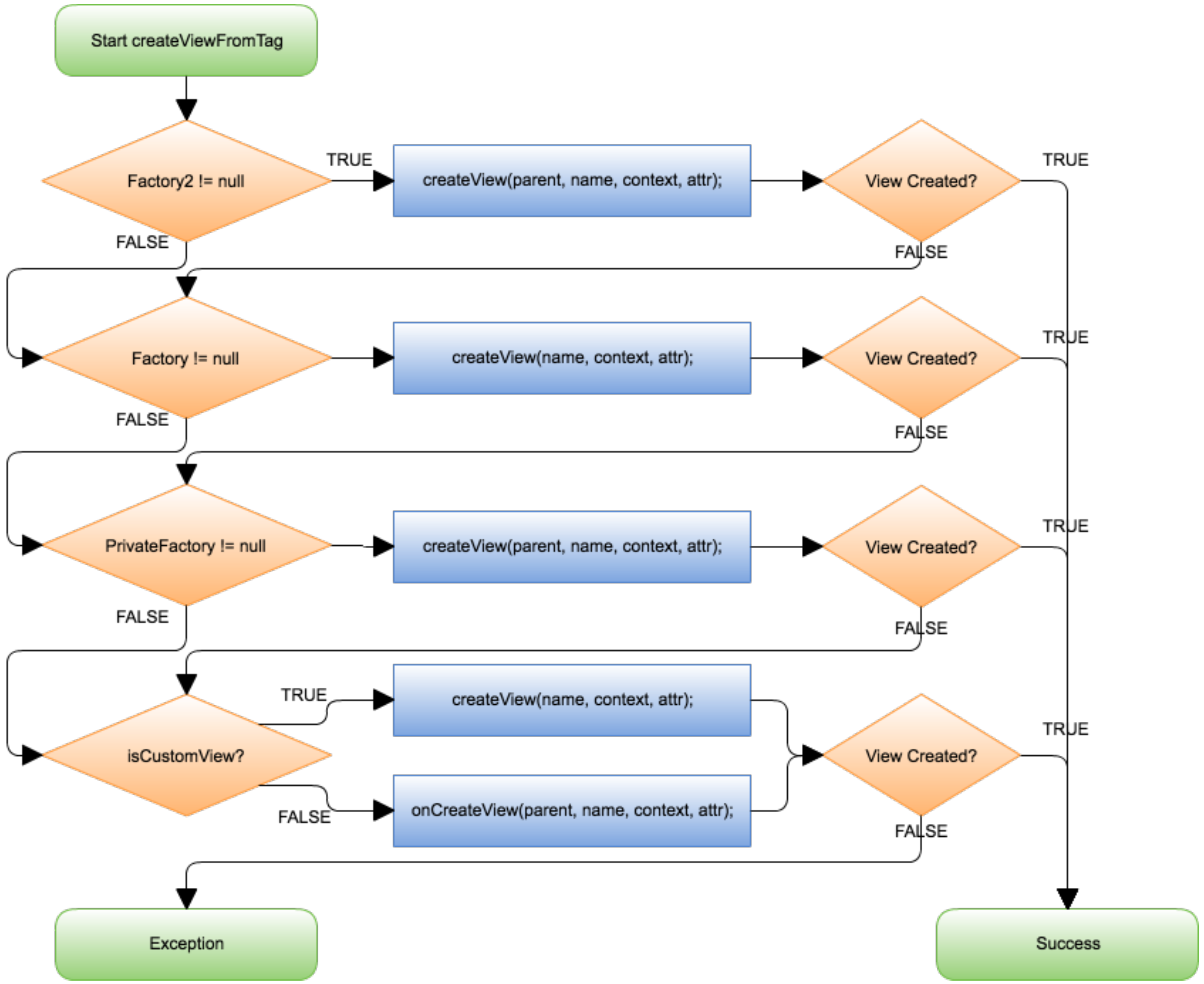
Wrap ALL the Factories:

```
class CalligraphyLayoutInflater extends LayoutInflater {
  {
    setFactory2(new CalligraphyFactory2Wrapper(getFactory2()));
    setPrivateFactory(
        new CalligraphyPrivateFactoryWrapper(getPrivateFactory()));
  }
  @Override
  protected View onCreateView(name, attrs){
    view = super.onCreateView(name, attrs);
    return mCalligraphyFactoryWrapper
        .onViewCreated(view, view.getContext(), attrs);
  }
}
```

Works; better...

- Intercepts all factories.
- Respects factories already set.
- Still broken custom view support.

Warning dragons ahead.
警告ドラゴンズ


Start createViewFromTag
Factory2 != null
TRUE
createView(parent, name, context, attr);
View Created?
TRUE
FALSE
FALSE
Factory != null
createView(name, context, attr);
View Created?
TRUE
FALSE
FALSE
PrivateFactory != null
createView(parent, name, context, attr);
View Created?
TRUE
FALSE
FALSE
isCustomView?
TRUE
createView(name, context, attr);
FALSE
onCreateView(parent, name, context, attr);
View Created?
TRUE
FALSE
Exception
Success

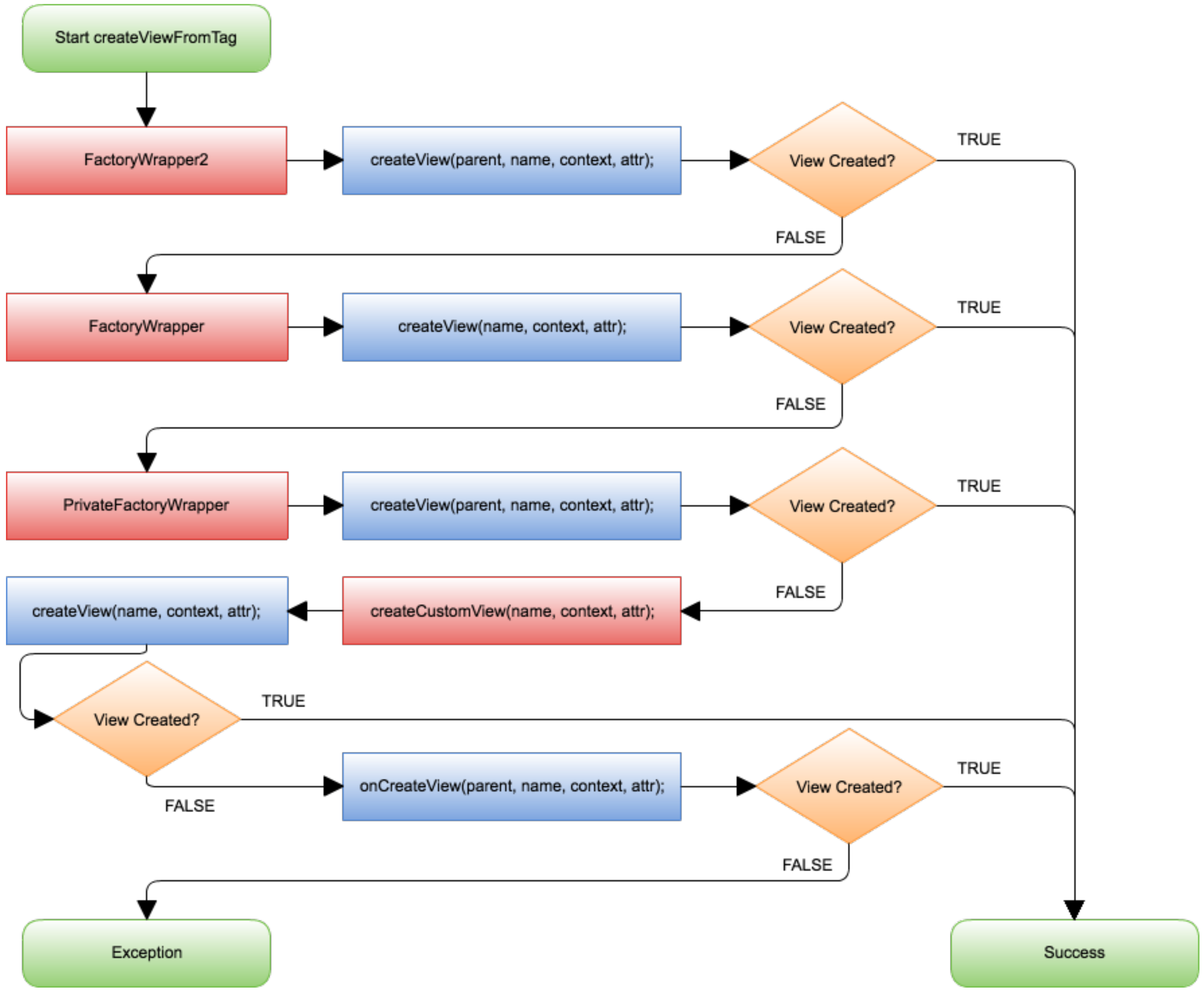
The problem is `final`.

LayoutInflater.createView()

```
if (-1 == name.indexOf('.')) {
  view = onCreateView(parent, name, attrs);
} else {
  // We want to intercept this.
  view = createView(name, null, attrs);
}
```

But `createView(name, prefix, attrs)` **is** `final`**.**

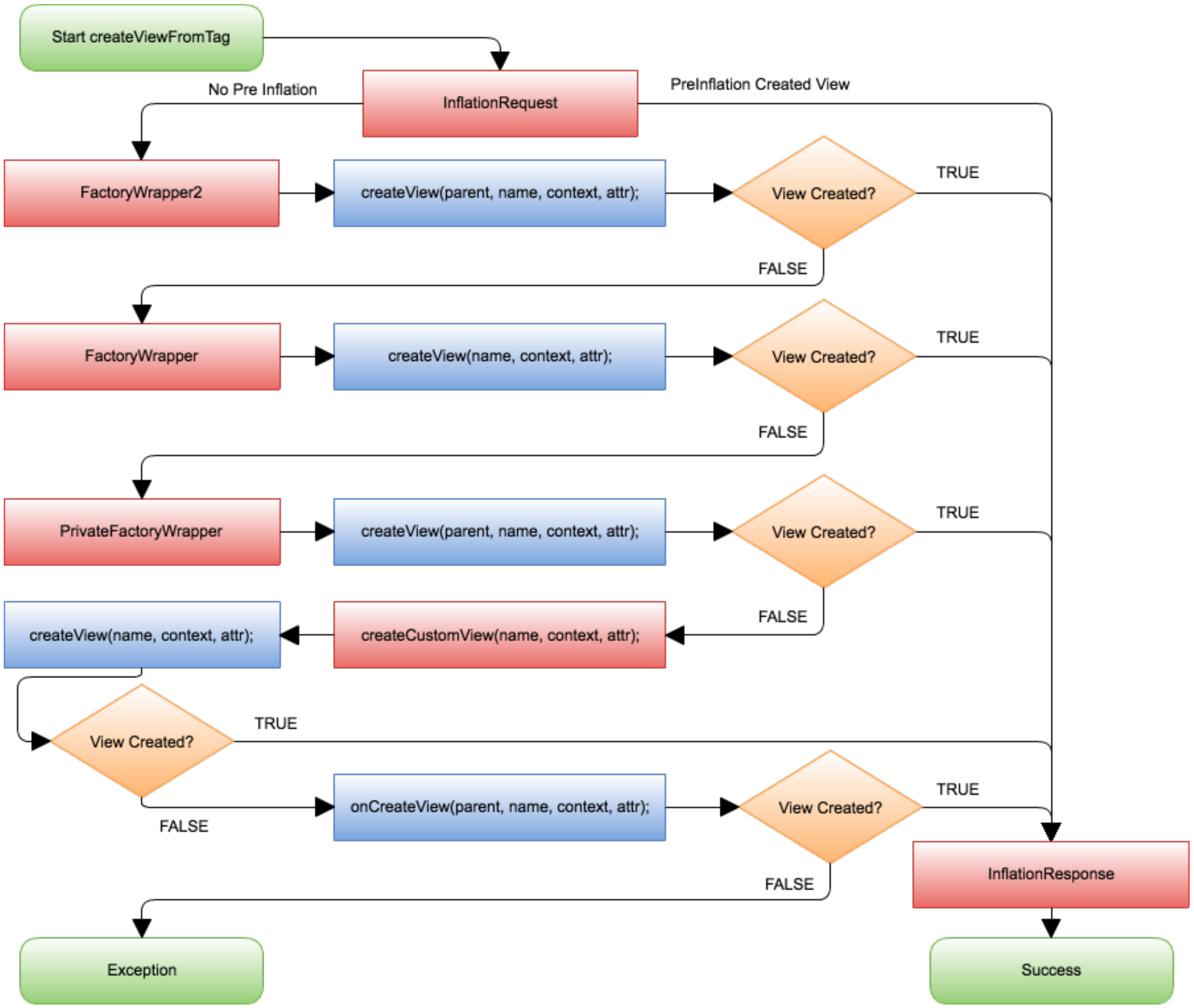
We changed the Inflation flow.


Start createViewFromTag
FactoryWrapper2
createView(parent, name, context, attr);
View Created?
TRUE
FALSE
FactoryWrapper
createView(name, context, attr);
View Created?
TRUE
FALSE
PrivateFactoryWrapper
createView(parent, name, context, attr);
View Created?
TRUE
FALSE
createView(name, context, attr);
createCustomView(name, context, attr);
View Created?
TRUE
FALSE
onCreateView(parent, name, context, attr);
View Created?
TRUE
FALSE
Exception
Success

`createCustomView`

createCustomView

```
private View createCustomView(View parent, View view, String name,
      Context viewContext, AttributeSet attrs) {
  // Is this a custom view?
  if (view == null && name.indexOf('.') > -1) {
      if (mConstructorArgs == null)
          mConstructorArgs = ReflectionUtils
              .getField(LayoutInflater.class, "mConstructorArgs");

      final Object[] mConstructorArgsArr =
            (Object[]) ReflectionUtils.getValue(mConstructorArgs, this);
      final Object lastContext = mConstructorArgsArr[0];
      // Have to set the correct context to the View constructor args, because
      // they can't be passed in.
      mConstructorArgsArr[0] = viewContext;
      ReflectionUtils.setValue(mConstructorArgs, this, mConstructorArgsArr);
      try {
          view = createView(name, null, attrs);
      } catch (ClassNotFoundException ignored) {
      } finally {
          mConstructorArgsArr[0] = lastContext;
          ReflectionUtils.setValue(mConstructorArgs, this, mConstructorArgsArr);
      }
  }
```


Start createViewFromTag
InflationRequest
No Pre Inflation
PreInflation Created View
FactoryWrapper2
createView(parent, name, context, attr);
View Created?
TRUE
FALSE
FactoryWrapper
createView(name, context, attr);
View Created?
TRUE
FALSE
PrivateFactoryWrapper
createView(parent, name, context, attr);
View Created?
TRUE
FALSE
createView(name, context, attr);
createCustomView(name, context, attr);
View Created?
TRUE
FALSE
onCreateView(parent, name, context, attr);
View Created?
TRUE
FALSE
InflationResponse
Exception
Success

ViewPump and Calligraphy

```
ViewPump.init(ViewPump.builder()
    .addInterceptor(new CalligraphyInterceptor(/* .. */))
    .addInterceptor(new TextUpdatingInterceptor())
    .addInterceptor(new CustomTextViewInterceptor())
    .build());
```

Whys?

- Respect Android Styles and Themes set in XML.
- Can still build views visually.
- PreInflation Hooks, Turn XML Tag -> View
- PostInflation Hooks, Interact with View with their attributes after inflation.
- Simpler to interact with than `setFactory` methods.

Use cases:

Replace all TextViews with `ChrisTextView`

```
public InflateResult intercept(Chain chain) {
  InflateRequest request = chain.request();
  View view = inflateChrisTextView(request.name(), request.context(), request.attrs());
  if (view != null) {
      return InflateResult.builder()
              .view(view)
              .name(view.getClass().getName())
              .context(request.context())
              .attrs(request.attrs())
              .build();
  } else {
      return chain.proceed(request);
  }
}
```

Use cases:

Set views without accessibility to GONE!

```
public InflateResult intercept(Chain chain) {
  InflateResult result = chain.proceed(request);
  View view = result.view();
  // Maybe a little aggressive.
  if(view.getContentDescription().isEmpty()) {
    view.setVisibility(View.GONE);
  }
  return result;
}
```

That's all

Chris Jenkins - @chrisjenx
James Barr - @jbarr21

ViewPump - `github.com/InflationX/ViewPump`
Calligraphy - `github.com/InflationX/Calligraphy`

Questions?