CREATING COMPOSITE VIEWS

Beyond The Basics – Parts 1 & 2

James Harmon
jamesharmon@gmail.com
www.ObjectTrainingGroup.com
LinkedIn http://www.linkedin.com/pub/james-harmon/0/298/4bb

Topics

- What is a Composite Views
- Building the layout
- Creating the constructors
- Adding custom attributes
- Adding properties
- Making Composite Views extensible
- Proper Encapsulations
- Using Composite Views
- Tools Especially Useful for Composite Views
- Packaging Composite Views for Reuse

WHAT IS A COMPOSITE VIEW

A view is a visual component that can be displayed on the screen using either a declarative or programmatic technique

OR

```
TextView tv = new TextView(context);
tv.setText("Hello World");
```

Composite View
– a view “composed” of other views

TextView

TextView

ImageView

- Not a subclass of View
- Not new functionality (?)
- Not a performance improvement

- When not to create a composite view
  - Only used once
  - Too many options

Framework View

Enterprise View

Open
(Framework)

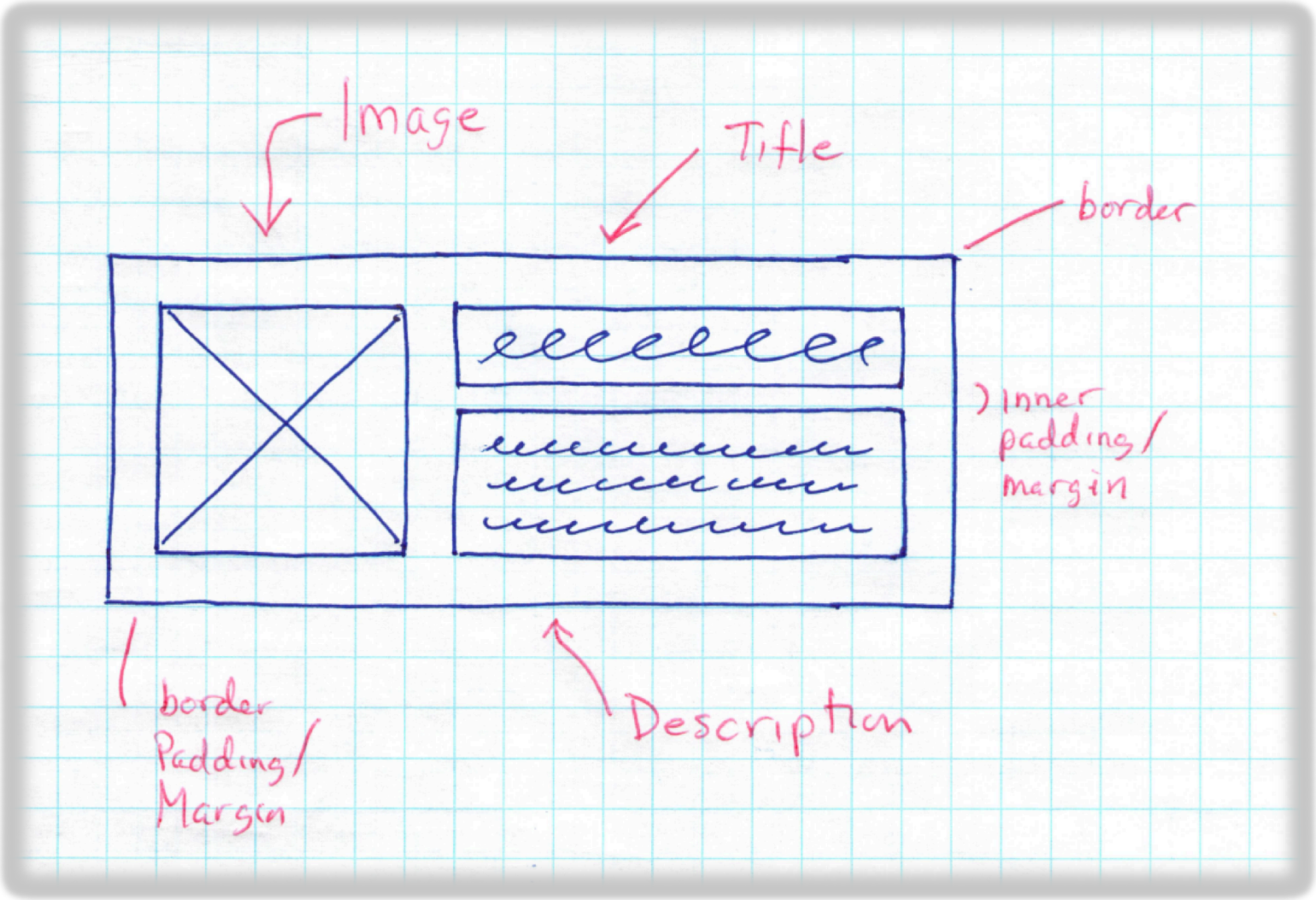
Closed
(Enterprise)

Use it like an existing view

- programmatically or declaratively
- functionality is configured and modified through properties and methods

Simple reuse

- Create a separate file
- use import

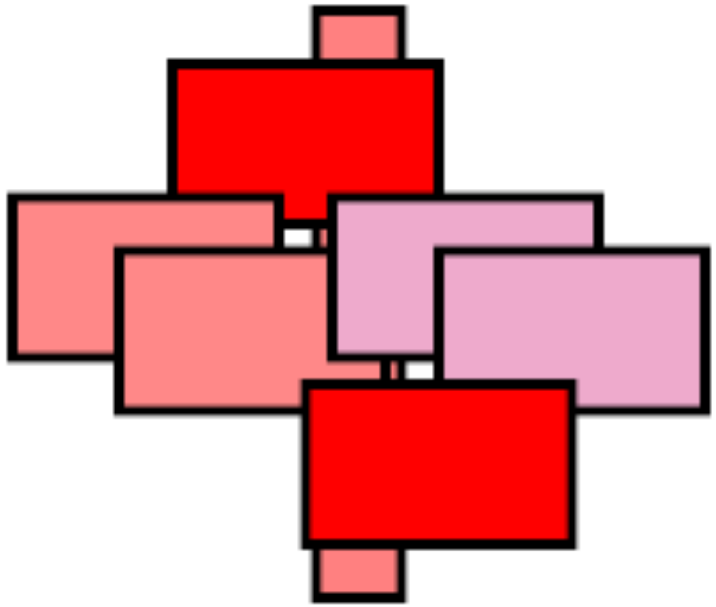


Here’s an example

Let’s see the example

BUILDING THE LAYOUT

- Make the layout reusable
- You won't always know you want to make a CV in advance
- So use good layout principles
  - Encapsulation
  - DRY



- Encapsulate views in a ViewGroup
  - RelativeLayout is the best

- Assume no knowledge of “outside” views

What about “onDraw”, “onMeasure”, “onLayout”

Composite views consist of views that already know how to measure / draw / layout

So pick a view container (ViewGroup) that knows how to do the same thing

RelativeLayout is your friend

Code

Write good OO code – this makes building the CV easier

Start by putting the code in the Fragment or Activity

Encapsulate the code in the CV

BUILDING THE VIEW

The Process

- Select the appropriate Super class
- Create the View Class
- Create the constructors

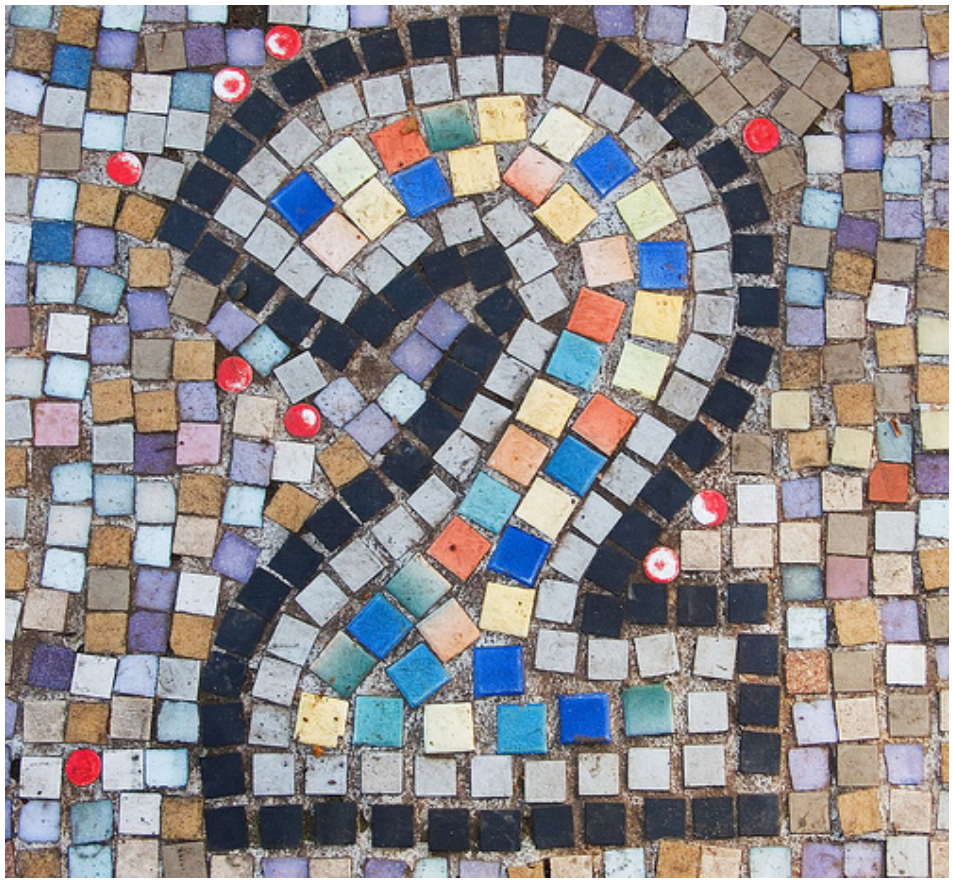
Which Super Class?

- ViewGroup
- LinearLayout
- RelativeLayout

What are our constructor choices? When and how are they called.

- 3 Constructors
- Called from view
- Follow the chaining

Use the 2 argument constructor



ADDING CUSTOM ATTRIBUTES

Decorate the CV with custom attributes

- Expose as attributes

Create a custom name space

- How “big” should the name space be?

Support expected attributes

- Most attributes are included
- Others you need to define

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <!-- Attributes for ImageCaptionView -->
    <declare-styleable name="ImageCaptionView">
        <!-- Image caption -->
        <attr name="caption" format="string" />
        <!-- Image description -->
        <attr name="description" format="string" />
    </declare-styleable>

</resources>
```

Pull out the attributes in the constructor and assign them to the children

```
TypedArray a = getContext().
    obtainStyledAttributes(attrs, R.styleable.ImageCaptionView, 0, 0);

int resId = a.getResourceId(R.styleable.ImageCaptionView_image, 0);
// only assign image if attribute is set
if (resId != 0) {
    setImage(resId);
}

String captionText = a.getString(R.styleable.ImageCaptionView_caption);
caption.setText(captionText);

String descriptionText =
    a.getString(R.styleable.ImageCaptionView_description);

description.setText(descriptionText);

a.recycle();
```

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
                xmlns:bullet="http://schemas.android.com/apk/res-auto"
```

ADDING PROPERTIES PROPERLY

Support properties that enhance the functionality of the CV

- setters and getters
- Proper naming conventions
- Use constants

MAKING COMPOSITE VIEWS EXTENSIBLE

Make CVs Extensible

- Don’t hardcode things
- Use styles to avoid hard coding

Make CVs Usable

- JavaDoc
- Naming conventions

Make CVs Flexible

- Dynamically changing text size
- Enforcing aspect ratio

PROPER ENCAPSULATION

Child Views

When appropriate allow for the possibility of additional child views or view replacements

Encapsulation Example – using Volley for image retrieval

- Hide Volley inside view

USING COMPOSITE VIEWS

- Add to layouts
- Support programmatic usage

TOOLS FOR IMPROVING COMPOSITE VIEWS

- Hierarchy Viewer
- Droid Inspector

PACKAGING COMPOSITE VIEWS FOR REUSE

- Create a separate library project
- Configure library project
  - Android properties
  - is library
- Configure using project
  - Android properties
  - add library

- Remove composite view items from using project
- Remove unneeded artifacts from library project

- Create “aar” files with Gradle
- Use “aar” files in apps
- Publish “aar” files to public or private repositories

Links & Contact Info

- Slides
  - https://github.com/jamesharmon/image-caption-view/slides.pdf
- Source Code
  - https://github.com/jamesharmon/image-caption-view
- jamesharmon@gmail.com