Coding Standards Android

Change History

Date	Ver	Change Description	Prepared By	Reviewed By	Approved By
26-08-2016	1.0	Standards have been added	Rajendhiran	Kumaran	Harihara G
07-02-2017	1.1	Updated the document ID	Deepa	Deepa	Harihara.G
01-02-19	1.2	Updated the document ID	Document Controller	MR	CISO

CodeQuality & BestPractice for Android - (rudiments)

Agenda

1. Quality Measurement
2. Code Standards - Basic
3. Android Practices – Basic

1. Quality Measurement

a. It's not that much easy to measure the code quality????
b. Because the best part of programming lays at all the edges of the sphere, so first we need to find those edges of the sphere… !@#!@%
c. Quality has been determined only based on low rate in the No. of WTFs/Minute….
d. This can be resolved by following some basic standards and there are some code analysis tools help us lot on this

1. Quality Measurement – Code Analysis Tools

FindBugs"...is a static code analysis tool that analyses Java byte code and detects a wide range of problems"

PMD

"...scans source code and looks for potential problems possible bugs, unused and sub-optimal code and over-complicated expressions."

CheckStyle

"...is a development tool to help programmers write Java code that adheres to a coding standard."

Android Lint

"...is a static code analysis tool that checks your Android project source files for potential bugs and optimization improvements for correctness, security, performance, usability, accessibility, and internationalization."

1. Quality Measurement – Code Analysis Tools – Comparison

• Finds real bugs • Low false detection rate • Fast (works on byte code)	• Finds bad practices • Finds duplicate code with CPD	• Checks code against some standards (e.g. Sun Code Conventions) • Can not find real bugs	• Dedicated to Android, built-in on Android Studio • Mostly finds problems about resources • Detects only some Java bad practices

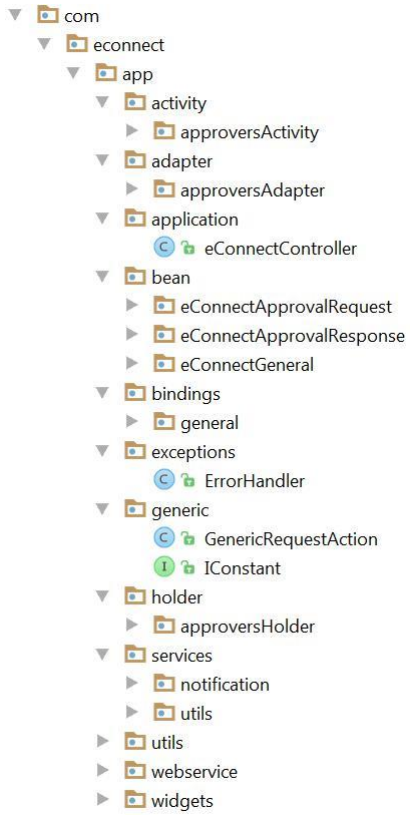
Use Gradle plugins for Findbugs, PMD, Checkstyle, and applying the built-in Lint checks. Combine them with gradle tasks and run them at once. A good read by Vincent Brison: http://goo.gl/hqQdfN

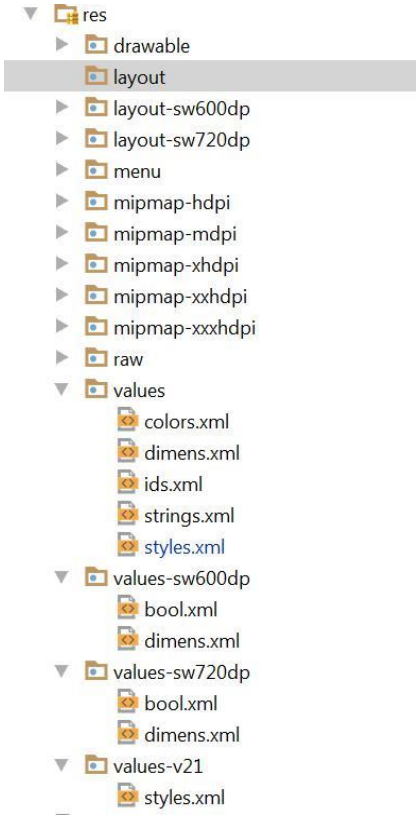
2. Basic Code Standard

a. Organized Files
b. Naming Conventions
c. Anonymous or Nested Class
d. Clean-up Your Code
e. Less Complexity vs Better Readability
f. Nested Control Flows
g. Class Size & Dependencies
h. Avoid Complex Conditions
i. Misc

Organise the code & resource files, It will help the user on understanding the code base.

- Viable for other to work on our code at our absence.
- It help the user to quickly navigate among the modules and start work on the codes.
- This will reduce the uncessary confusions like searching up the codes on different files.





2. Basic Code Standard – Naming Conventions

Identifier Type	Rules for naming	Example
Packages	The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, currently com, edu, gov, mil, net, org, or one of the English two-letter codes identifying countries as specified in ISO Standard 3166, 1981. Subsequent components of the package name vary according to an organization's own internal naming conventions. Such conventions might specify that certain directory name components be division, department, project, machine, or login names.	com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese
Classes	Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML). – PascalCase	class Raster; class ImageSprite;
Interfaces	Interface names should be capitalized like class names.	interface RasterDelegate; interface Storing
Methods	Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized. - CamelCase	run(); runFast(); getBackground();
Variables	Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should not start with underscore _ or dollar sign $ characters, even though both	int i; char c; float myWidth

	are allowed. Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.	

Constants	The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging.)	static final int MIN_WIDTH = 4;
		static final int MAX_WIDTH = 999;
		static final int GET_THE_CPU = 1;

Basic Code Standard – Anonymous or Nested Class

```
@Override
protected void onCreate(Bundle savedInstanceState)
{ super .onCreate(savedInstanceState);
...
   final String someValue = getSomeValue();
button.setOnClickListener(new View.OnClickListener()
{
@Override
public void onClick(View v) {
           doSomething();
doSomethingElse();
           if (someCondition) {
textView.setText(someValue);
}
       }
   }) ;
...
}
```

```
@Override
protected void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState);
...
String someValue = getSomeValue();
button.setOnClickListener(new
MyButtonClickListener(someValue)); ...
}
private class MyButtonClickListener
implements View.OnClickListener {
private final String someValue;
   public MyButtonClickListener(String someValue)
{
this.someValue = someValue;
}
@Override
public void onClick(View v) {
       doSomething();
doSomethingElse();
       if (someCondition) {
textView.setText(someValue);
}
   }
```

Basic Code Standard – Clean-up Your Code

Remove all Unused

- Fields
- Methods
- Method parameters
- Classes
- Future proof code

```java
public class Calculator {

    private int result;

    public int add(int a, int b) {
        return a + b;
    }

    private int multiply(int a, int b, String dummy) {
        int result = 0;
        for (int i = 0; i < b; i++) {
            result = add(result, a);
        }
        return result;
    }
}
```

Basic Code Standard – Less Complexity vs Better Readability

```java
public intgetStringResourceForStatus(final String
itemStatus) {
if (ACTIVE.equalsIgnoreCase(itemStatus)) {
return R.string.status_active;
}
if (PAUSED.equalsIgnoreCase(itemStatus)) {
return R.string.status_paused;
}
if (DELETED.equalsIgnoreCase(itemStatus)) {
return R.string.status_deleted;
}
if (DELAYED.equalsIgnoreCase(itemStatus)) {
return R.string.status_delayed;
}
if (PENDING.equalsIgnoreCase(itemStatus)) {
return R.string.status_pending;
}
if (SOLD.equalsIgnoreCase(itemStatus)) {
return R.string.status_sold;
}
if (BOUGHT.equalsIgnoreCase(itemStatus)) {
return R.string.status_bought;
} else {
return R.string.status_unknown;
     }
```

```java
public intgetStringResourceForStatus(final String
itemStatus) {
int resourceId;
if (ACTIVE.equalsIgnoreCase(itemStatus)) {
        resourceId =R.string.status_active;
} else if (PAUSED.equalsIgnoreCase(itemStatus)) {
        resourceId =R.string.status_paused;
} else if (DELETED.equalsIgnoreCase(itemStatus)) {
        resourceId =R.string.status_deleted;
} else if (DELAYED.equalsIgnoreCase(itemStatus)) {
        resourceId =R.string.status_delayed;
} else if (PENDING.equalsIgnoreCase(itemStatus)) {
        resourceId =R.string.status_pending;
} else if (SOLD.equalsIgnoreCase(itemStatus)) {
        resourceId =R.string.status_sold;
} else if (BOUGHT.equalsIgnoreCase(itemStatus)) {
        resourceId =R.string.status_bought;
} else {
        resourceId =R.string.status_unknown;
     }
return resourceId;
}
```

The Cyclomatic Complexity of this method is 16 which is greater than 10 authorized.

Comment | Open | Confirm | Resolve | False Positive | Assign [to me] | Plan | Change Severity | Debt: 11min

Rule | Changelog

Methods should not be too complex

The cyclomatic complexity of methods should not exceed a defined threshold. Complex code can perform poorly and will in any case be difficult to understand and therefore to maintain.

squid:MethodCyclomaticComplexity | Testability > Unit level

Basic Code Standard – Nested Control Flows

```
public void myMethod() {
if (condition1) {
   /* ... */
if (condition2) {
            /* ... */
for (int i = 0; i <10; i++) {
                /* ... */

if (condition4) {

if (condition5) {
                          /* ... */
}
return;
}
            }
        }
    }
}
```

=

```
public void myMethod() {
if (condition1) {
   /* ... */
if (condition2) {
            /* ... */
for (int i = 0; i <10; i++) {

                /* ... */

if (condition4) {
if (condition5) {
                          /* ... */
}
return;
}
            }
        }
    }
}
```

Basic Code Standard – Class Size & Dependencies

```
public class UserProfile implements Parcelable {
   private String userId;
   private String name;
   private String surname;
   private String initials;
   private String email;
   private Time joinDate;
   private String phoneNumber ;
   private String locationId;
   private String street;
   private String streetNumber;
   private String city;
   private String region;
   private String country;
   private String zipCode;
   private AccountType accountType;

   ...
```

```
public class UserProfile implements Parcelable {
   private String userId;
   private String name; private
   String surname; private String
   initials; private String email;
   private Time joinDate; private
   String phoneNumber; private
   UserAddress address; private
   AccountType accountType; ...
```

```
public class UserAddress implements Parcelable {
   private String locationId;
   private String street;
   private String streetNumber;
   private String city;
   private String region;
   private String country;
   private String zipCode;
   ...
```

Basic Code Standard – Avoid Complex Conditions

```
if (((condition1 || condition2) && condition3 && condition4) || (condition5
&& condition6)) {
// Do something
} else {
// Do something else
}
```

```
if (firstCondition() || thirdCondition()) {
// Do something
} else {
// Do something else
}
private boolean firstCondition() {
return (condition1 || condition2) && secondCondition();
}
private boolean secondCondition() {
return condition3 && condition4;
}
private boolean thirdCondition() {
return condition5 && condition6;
}
```

Basic Code Standard – Misc. Override for a Reason

```
public class MainActivity
extends AppCompatActivity {
@Override
   protected void onCreate(Bundle
savedInstanceState) {
super.onCreate(savedInstanceState);
...
   }
@Override
protected void onResume() {
super .onResume();
doSomething();
}
@Override
protected void onPause() {
super.onPause();
doSomethingElse();
}
}
```

```
public class MainActivity extends AppCompatActivity
{ @Override
   protected void onCreate(Bundle savedInstanceState)
{ super.onCreate(savedInstanceState);
...
   }
@Override
protected void onResume() {
super.onResume();
// doSomething();
}
@Override
protected void onPause()
{ super.onPause();
// doSomethingElse();
}
}
```

Basic Code Standard – Misc. String Operations

```
public class Price {
private int value;
   …
public StringgetAsString() {
return value + ""; // bad!
}
```

->

```
new
StringBuilder().append(Integer.toString(value)).append("
).toString();
```

```
public StringgetAsString() {
    return String.valueOf(value); // good
    // OR
return Integer.toString(value); // even
better!
}
```

Basic Code Standard – Misc. Adding Something to String

```
public class Item {
private int price;
   …
public int getPrice() {
return price;
}
}
// In some Activity
textView.setText("Price: " + item.getPrice()); // bad!
```

```
textView.setText("Price: " + Integer.toString(item.getPrice())); // good
textView.setText(String.format("Price: %d", item.getPrice())); // even better!
```

Android Way...

```
<string name="price_text">Price: %1$d</string>
textView.setText(getString(R.string.price_text, item.getPrice()));
```

Basic Code Standard – Misc. String Comparison

```
public boolean someMethod(String value)
   { String someOtherString;
   // ...

if (value == someOtherString) {// bad!
return true;
} else {
return false;
}
```

```
public boolean someMethod(String value)
   { String someOtherString;
   // ...

if (value.equals(someOtherString)) { // good!
return true;
} else {
return false;
}
```

```
/**
* Compares the given object to this string and
returns true if they are
* equal. The object must be an instance of
{@code String} with the same length,
* where for every index, {@code charAt} on each string
returns the same value.
*/
@Override public boolean equals(Object other) {
if (other == this) {
return true;
}
if (other instanceof String) {
        String s = (String)other;
        int count = this.count; if
        (s.count != count) {
return false;
}
// TODO: we want to avoid many boundchecks in the
loop below
// for long Strings until we have array
equality intrinsic.
        // Bad benchmarks just push .equals without
first getting a
        // hashCode hit (unlike real world use in a
Hashtable). Filter
        // out these long strings here. When we get the
array equality
        // intrinsic then remove this use of
hashCode. if (hashCode() != s.hashCode()) {
return false;
}
for (int i = 0; i < count; ++i)
{ if (charAt(i) != s.charAt(i))
{ return false;
```

Basic Code Standard – Misc. Literal Boolean Values

```
public boolean someMethod(String value) {
   String someOtherString;
   // ...

if (value.equals(someOtherString)) { // good!
return true;
} else {
return false;
}
```

=

```
public boolean someMethod(String value) {
   String someOtherString;
   // ...

   return value.equals(someOtherString); // better!
}
```

```
public void someMethod(String value) {
boolean isValid = isValid(value);
   // ...

if (isValid==true){ //bad!
        doSomething();
} else {
        doSomethingElse();
}
```

```
public void someMethod(String value) {
boolean isValid = isValid(value);
   // ...

if (isValid) { // good!
        doSomething();
} else {
        doSomethingElse();
}
```

```
}
```

=

Basic Code Standard – Misc. Android Way AsyncTaskLoader

12. Use an AsyncTaskLoader instead of an AsyncTask

A caveat while using an AsyncTask is that if the Activity gets destroyed before the AsyncTask has completed, it will still keep running and deliver the result in it's onPostExecute() method, which could cause unexpected behaviour. A typical example of this situation is when a device is rotated while an AsyncTask is loading content.

Loaders were introduced in Honeycomb but can also be used in pre-Honeycomb versions using the support library.

Loaders are managed by a LoaderManager which is tied to the lifecycle of it's Activity or Fragment. Each Activity or Fragment contains an instance of LoaderManager. If the Activity/Fragment is destroyed, the LoaderManager destroys the Loaders and frees up resources. In case of a configuration change, it retains it's Loaders.

We can get a LoaderManger instance and initialize a Loader in the following way:

```
getLoaderManager().initLoader(LOADER_ID, null, this);
```

A simple AsyncTaskLoader can be created in the following way:

```
class CustomLoader extends AsyncTaskLoader<String> {

    public CustomLoader(Context context) {
        super(context);
    }

    public String loadInBackground() {
        String result = null;
        // Load result string
        return result;
    }
}
```

You might also want to override onStartLoading(), onForceLoad(), onReset(), onCancelled(), onStopLoading(), onAbandon(), cancelLoadInBackground(), onCancelLoad() according to your needs.

3. Basic Android Practices

a. Reuse Styles & Resources
b. Include Layouts

c. Merge – Include - Layouts
d. Colors

e. Dimensions

f. strings.xml
g. Activity or Fragment?
h. Build Variant
i. Signing Configs

Before

```
<TextView
android:id="@+id/my_text"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
.
.
.
```

Extracted Styles

```
<style name="Wrap">
<item
name="android:layout_width">wrap_content
</item>
<item
name="android:layout_height">wrap_conten
t</item>
</style>
```

After

```
<TextView
android:id="@+id/my_text"
style="@style/Wrap"
.
.
.
/>
```

```
<ImageView
android:id="@+id/my_image"
android:layout_width="match_parent"
android:layout_height="wrap_content"
.
.
.
```

```
<style name="MatchWidth">
<item
name="android:layout_width">match_paren
t</item>
<item
name="android:layout_height">wrap_conte
nt</item>
</style>
```

```
<ImageView
android:id="@+id/my_image"
style="@style/MatchWidth"
.
.
.
/>
```

```
<LinearLayout
android:id="@+id/my_linear_layout"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:orientation="vertical"> .

.
```

```
<style name="MatchWidthVertical">
<item
name="android:layout_width">match_paren
t</item>
<item
name="android:layout_height">wrap_conte
nt</item>
<item
name="android:orientation">vertical</it
em>
</style>
```

```
<LinearLayout
android:id="@+id/my_linear_layo
ut"
style="@style/MatchWidthVertica
l">
.
.
```

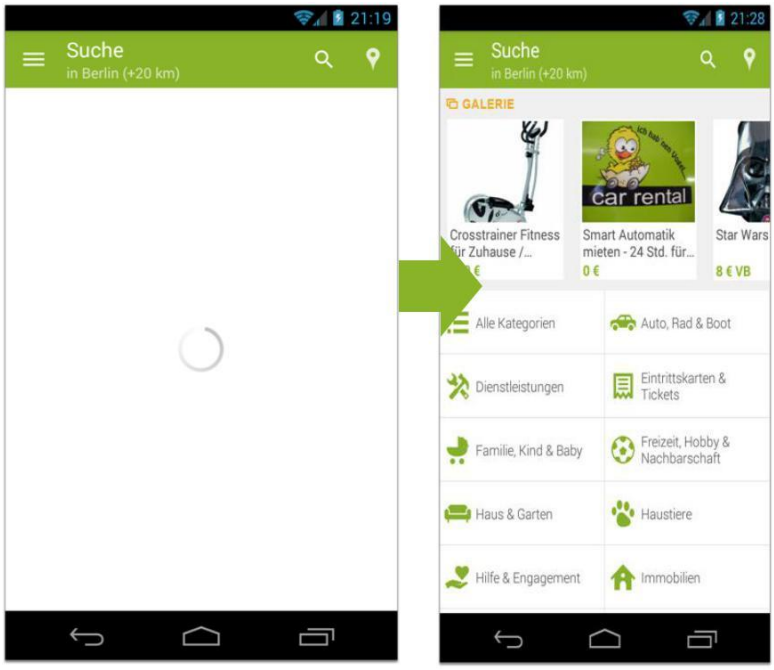
Basic Android Practices - Parent Styles

```
<style name="EbayEditText.Base"
parent="@style/Widget.AppCompat.EditText">
<item
name="android:background">?attr/editTextBackground</item>
<item name="android:textColor">@color/dark_grey</item>
<item
name="android:textColorHint"GD/SDM/CodingStandards>@color/midtoneAndroid
grey</item> V1.1 <item
name="android:textAppearance">@style/TextMedium</item>
</style>
<style name="LoginField"
```

```
parent="@style/EbayEditText.B
ase"> <item
```

```
<LinearLayout
style="@style/MatchVertical"
android:gravity="center_horizontal"
android:paddingLeft="@dimen/gutter_double"
android:paddingRight="@dimen/gutter_double">
<EditText
android:id="@+id/auth_email"
style="@style/Username" />
<EditText
android:id="@+id/auth_password"
```

Basic Android Practices - Include Layouts




```
activity_home.xml
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/
res/android"
xmlns:tools="http://schemas.android.com/tools"
style="@style/Match">
    .
    .
    .
    .
<include
android:id="@+id/loading"
style="@style/Match"
layout="@layout/layout_loading"
android:visibility="gone" />
</RelativeLayout>
```

```
layout_loading.xml
<RelativeLayout
xmlns:android= "http://schemas.android.com/ap
k/res/android"
xmlns:tools="http://schemas.android.com/tool
s"
style="@style/Match"
android:background="@android:color/white">
<ProgressBar
android:layout_width="60dp"
android:layout_height="60dp"
android:layout_centerInParent="true"
android:indeterminate="true" />
</RelativeLayout>
```

Basic Android Practices - Merge – Include – Layouts

Another handy tag is the *<merge />* tag. It acts as a pseudo parent and helps get rid of an unneeded root *ViewGroup*.

For example, if your re-usable layout contains two Buttons placed vertically, you can put them inside a LinearLayout with vertical orientation. But this LinearLayout becomes redundant if the layout is included (using *<include />*) into another LinearLayout. In this case, our re-usable layout can have *<merge />* as the root ViewGroup instead of LinearLayout.

```
<merge xmlns:android="http://schemas.android.com/apk/res/android">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Submit" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Reset" />
</merge>
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter text" />

    <include layout="@layout/buttons.xml" />
</LinearLayout>
```

Basic Android Practices – Colors

Don't do this:

```
<resources>
    <color name="button_foreground">#FFFFFF</color>
    <color name="button_background">#2A91BD</color>
    <color name="comment_background_inactive">#5F5F5F</color>
    <color name="comment_background_active">#939393</color>
    <color name="comment_foreground">#FFFFFF</color>
    <color name="comment_foreground_important">#FF9D2F</color>
    ...
    <color name="comment_shadow">#323232</color>
```

Instead, do this:

```
<resources>

    <!-- grayscale -->
    <color name="white"     >#FFFFFF</color>
    <color name="gray_light">#DBDBDB</color>
    <color name="gray"      >#939393</color>
    <color name="gray_dark" >#5F5F5F</color>
    <color name="black"     >#323232</color>

    <!-- basic colors -->
    <color name="green">#27D34D</color>
    <color name="blue">#2A91BD</color>
    <color name="orange">#FF9D2F</color>
    <color name="red">#FF432F</color>
```

Basic Android Practices – Dimensions

Treat dimens.xml like colors.xml. You should also define a "palette" of typical spacing and font sizes, for basically the same purposes as for colors. A good example of a dimens file:

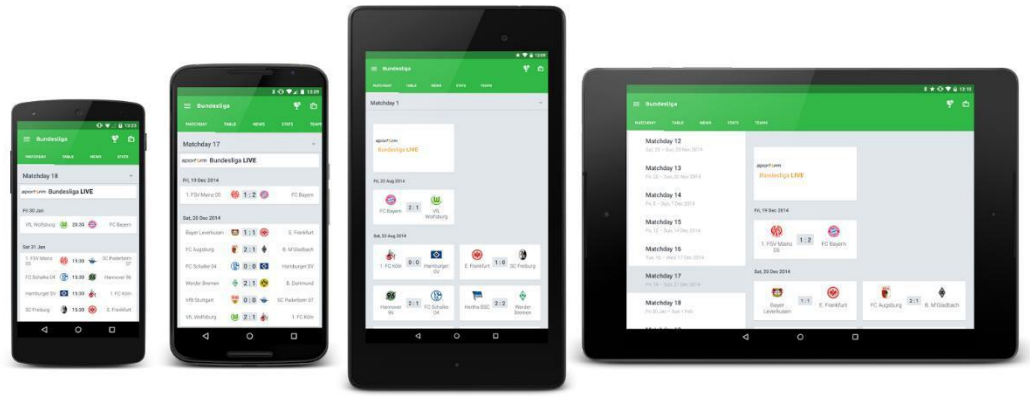
```
<resources>

    <!-- font sizes -->
    <dimen name="font_larger">22sp</dimen>
    <dimen name="font_large">18sp</dimen>
    <dimen name="font_normal">15sp</dimen>
    <dimen name="font_small">12sp</dimen>

    <!-- typical spacing between two views -->
    <dimen name="spacing_huge">40dp</dimen>
    <dimen name="spacing_large">24dp</dimen>
    <dimen name="spacing_normal">14dp</dimen>
    <dimen name="spacing_small">10dp</dimen>
    <dimen name="spacing_tiny">4dp</dimen>

    <!-- typical sizes of views -->
    <dimen name="button_height_tall">60dp</dimen>
    <dimen name="button_height_normal">40dp</dimen>
    <dimen name="button_height_short">32dp</dimen>

</resources>
```



Basic Android Practices – strings.xml

strings.xml

Name your strings with keys that resemble namespaces, and don't be afraid of repeating a value for two or more keys. Languages are complex, so namespaces are necessary to bring context and break ambiguity.

Bad

```
<string name="network_error">Network error</string>
<string name="call_failed">Call failed</string>
<string name="map_failed">Map loading failed</string>
```

Good

```
<string name="error_message_network">Network error</string>
<string name="error_message_call">Call failed</string>
<string name="error_message_map">Map loading failed</string>
```

Don't write string values in all uppercase. Stick to normal text conventions (e.g., capitalize first character). If you need to display the string in all caps, then do that using for instance the attribute `textAllCaps` on a TextView.

Bad

```
<string name="error_message_call">CALL FAILED</string>
```

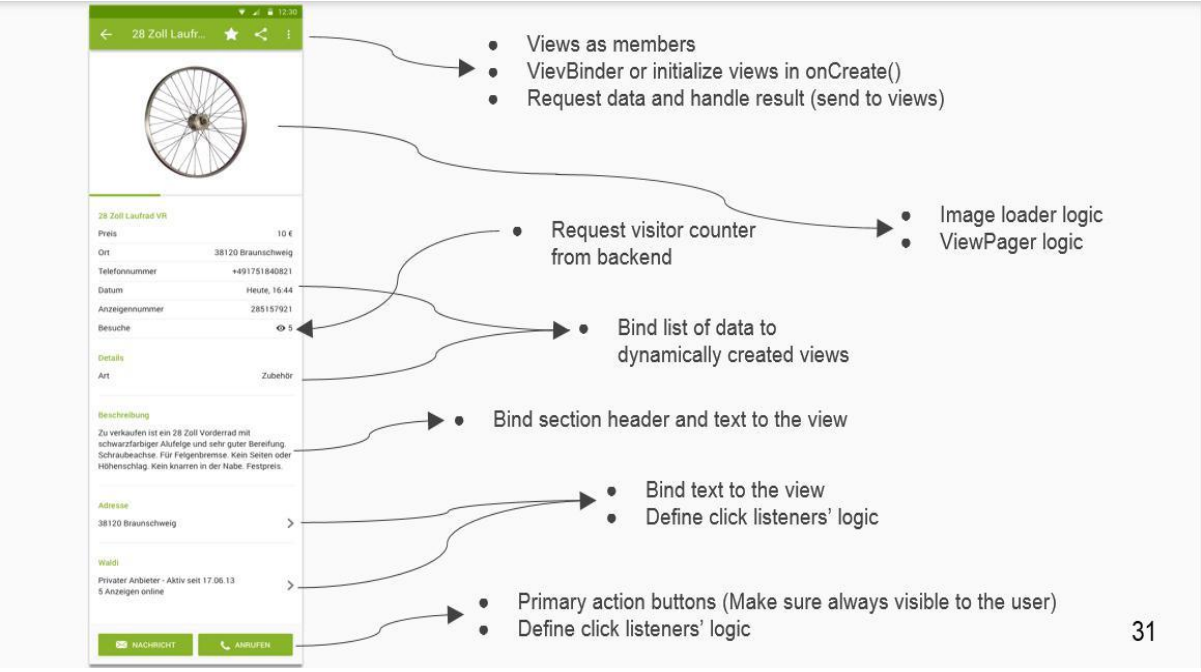
Good

```
<string name="error_message_call">Call failed</string>
```

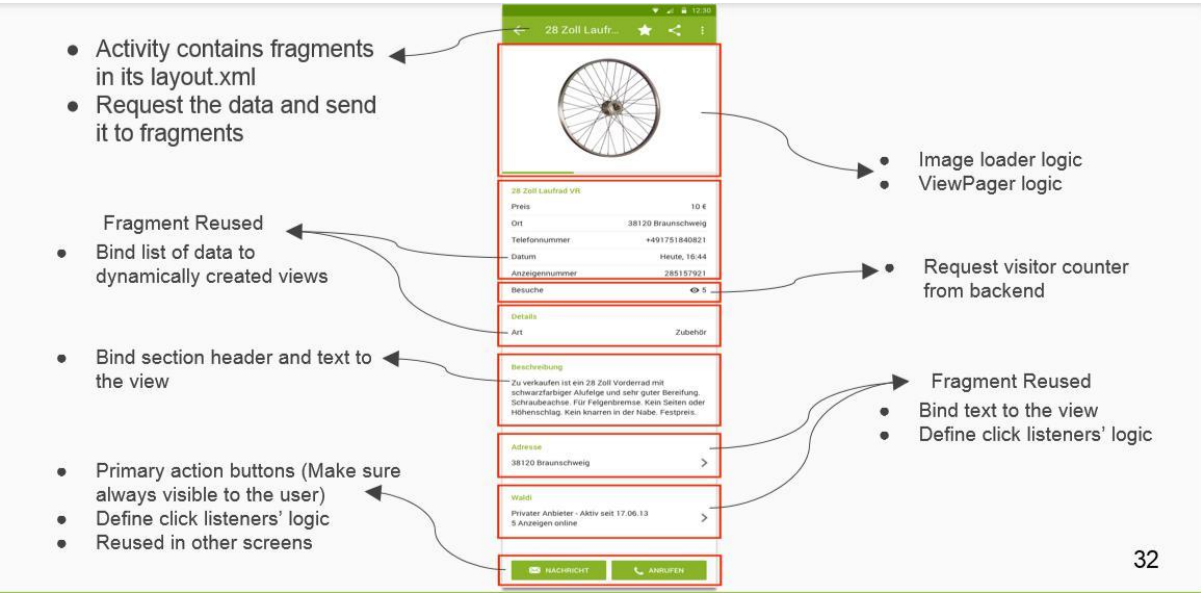
Basic Android Practices – Activity or Fragment?

One Activity with lots of views
or
Multiple Fragments in one Activity

Basic Android Practices – One Single Activity



Basic Android Practices – Multiple Fragments in One Activity



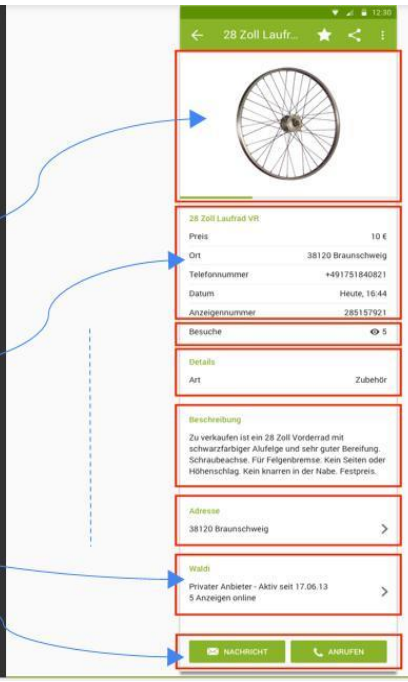
Basic Android Practices – Multiple Fragments in One Activity

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   style="@style/Match">

   <ScrollView
       style="@style/Match"
       android:layout_above="@+id/fragment_vip_contact" >
               <LinearLayout
           style="@style/MatchWidth"
           android:orientation="vertical">
               <fragment
                  android:id="@+id/fragment_vip_gallery"
                  android:name="ebk.vip.fragment.ImageGalleryFragment"
                  style="@style/MatchWidth"
                  android:layout_height="@dimen/vip_image_height"
                  tools:ignore="InconsistentLayout" />

               <fragment
                  android:id="@+id/fragment_vip_ad_basics"
                  android:name="ebk.vip.fragment.KeyValueFragment"
                  style="@style/MatchWidth" />
                              .
                              .
                              .
       </LinearLayout>
   </ScrollView>

   <fragment
       android:id="@id/fragment_vip_contact"
       android:name="ebk.vip.fragment.ContactFragment"
       style="@style/MatchWidth"
       android:layout_alignParentBottom="true" />
</RelativeLayout>
```



33

Basic Android Practices – Build Variant


```
buildTypes {
    debug {
        buildConfigField "String", "BASE_URL", '"http://sandbox.enoahprojects.com/econnectv15/webservicesv1/"';
        buildConfigField "String", "PROFILE_BASE_URL", '"http://sandbox.enoahprojects.com/econnectv15/assets/profileimages/"';
    }

    qa{

    }

    release {
        buildConfigField "String", "BASE_URL", '"http://econnect.enoahisolution.com/webservicesv1/"';
        buildConfigField "String", "PROFILE_BASE_URL", '"http://econnect.enoahisolution.com/assets/profileimages/"';
        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
    }
}
```

34

Basic Android Practices – Signing Configs

Passwords. In your app's `build.gradle` you will need to define the `signingConfigs` for the release build. Here is what you should avoid:

Don't do this. This would appear in the version control system.

```
signingConfigs {
    release {
        storeFile file("myapp.keystore")
        storePassword "password123"
        keyAlias "thekey"
        keyPassword "password789"
    }
}
```

Instead, make a `gradle.properties` file which should *not* be added to the version control system:

```
KEYSTORE_PASSWORD=password123
KEY_PASSWORD=password789
```

That file is automatically imported by Gradle, so you can use it in `build.gradle` as such:

```
signingConfigs {
    release {
        try {
            storeFile file("myapp.keystore")
            storePassword KEYSTORE_PASSWORD
            keyAlias "thekey"
            keyPassword KEY_PASSWORD
        }
        catch (ex) {
            throw new InvalidUserDataException("You should define KEYSTORE_PASSWORD and KEY_PASSWORD in gradle.proper
        }
    }
}
```