Coding standards – Java/Android.

This document is meant as a guideline for code practices for using Java/Android. It's not intended to tell you how to write everything but merely suggests a few best practices and techniques to use so that the code is easier to interpret and maintain. This is contrary to the source from android, who say that the standards they suggest are a strict set of rules.

One rule to keep in mind is to be consistent. If someone has adopted a slightly different standard for their parts of the code, try to keep to theirs instead of writing small parts / extra parts in your own version. Inconsistency is worse than not following rules!

This document has suggestions intended mainly at developing for android and as such doesn't contain some of the more basic Java conventions. As such I won't cover them explicitly in this document but if you want to read up on them, a good source can be found here: http://www.oracle.com/technetwork/java/codeconv-138413.html

Also the link for the source used to write this is: http://source.android.com/source/code-style.html#java-language-rules

Table of Contents

Java Style Rules.

Field Naming Conventions.

- Non-public, non-static field names start with m
- Static field names start with s
- other fields start with lower case letters.
- Public static final fields (constants) should be ALL_CAPS_&_UNDERSCORES, for example:

```
public class MyClass {
```

```
public static final int SOME_CONSTANT = 42;
public int publicField;
private static MyClass sSingleton;
int mPackagePrivate;
private int mPrivate;
protected int mProtected;
}
```

Javadoc Conventions.

- Every file should have a copyright at the top.
- Package statement should follow after this.
- Blocks to be separated by lines.
- If there's an inheritance / class declaration it should explain what the class does.
- Every non-trivial public method **must** contain a javadoc comment with at least one sentence describing what it does.
- Don't bother for things like getters, setters or other trivial methods.
- There's no android style so use the styles set out by sun/oracle in the link above.

Method Conventions.

- Methods should be short and focused
- No hard limit is placed on length
- Suggested length is at 40 lines, any longer and consider breaking it up if possible.

Brace Style.

- Braces do not go on their own lines, they're on the same as the code before them.

```
if (something) {
// …
} else if (somethingElse) {
```

- Fit it all onto one line if possible (advised but not required) where applicable.

```
if (condition)
body(); // bad!
```

Line length.

- 100 Characters long (Max).
- URLs can be longer than 100 for ease of copy paste.
- Import lines are an exception

Annotations.

Annotations should be used and should precede other modifiers for the same element.
The below is copied directly from the source mentioned above:

Android standard practices for the three predefined annotations in Java are:

- @Deprecated: The @Deprecated annotation must be used whenever the use of the annotated element is discouraged. If you use the @Deprecated annotation, you must also have a @deprecated Javadoc tag and it should name an alternate implementation. In addition, remember that a @Deprecated method is *still supposed to work*.

If you see old code that has a @deprecated Javadoc tag, please add the @Deprecated annotation.

- @Override: The @Override annotation must be used whenever a method overrides the declaration or implementation from a super-class.

For example, if you use the @inheritdocs Javadoc tag, and derive from a class (not an interface), you must also annotate that the method @Overrides the parent class's method.

- @SuppressWarnings: The @SuppressWarnings annotation should only be used under circumstances where it is impossible to eliminate a warning. If a warning passes this "impossible to eliminate" test, the @SuppressWarnings annotation *must* be used, so as to ensure that all warnings reflect actual problems in the code.

When a @SuppressWarnings annotation is necessary, it must be prefixed with a TODO comment that explains the "impossible to eliminate" condition. This will normally identify an offending class that has an awkward interface. For example:

```
// TODO: The third-party class com.third.useful.Utility.rotate() needs generics

@SuppressWarnings("generic-cast")

List<String> blix = Utility.rotate(blax);
```

When a @SuppressWarnings annotation is required, the code should be refactored to isolate the software elements where the annotation applies.

On a final note, you should use TODO comments!

Variable scope.

- Keep variable scope to a minimum.
- Declare them in the innermost blocks that covers all uses of the variables.
- Declared at the first point they are used.
- Should have an initializer (postpone if you don't have enough information at that point).
- Try-Catch is an exception to this!
- Loop variables should be declared in the loop, unless there's a good reason to not do it.

Indentation.

- Use 4 space indents or tabs, whichever you prefer but be consistent.

The standards mention that they never use tabs but they use whatever the code around them does. There appears to be no real reason why they don't use tabs so if you do, feel free to continue.

Java Language Rules.

Some rules about exception handling have been omitted from the document as I would have just had to copy paste nearly a page in about how to handle exceptions correctly. Please do refer to the document linked above to read about exceptions as they seem to be quite important!

Reminder link: http://source.android.com/source/code-style.html#dont-ignore-exceptions

Finalizers.

The standard says to not user finalizers for the following main reason:

- No guarantees to when a finalizer will be called or even that it will be called at all.

They can be useful if they work but from the sounds of it and they way they are described it's better off to not use them unless you're %100 sure that they're going to work.

Imports.

The ordering of imports are as follows:

- Android Imports
- Imports from 3rd parties
- Java and Javax
- They should be alphabetical with capitals first
- Blank lines between major groupings