1. Naming

1.1 Package

Should be in lowercase letter
e.g. java, lang, sql, util etc.

1.2 Constants

Should be in uppercase letter
e.g. RED, YELLOW, MAX_PRIORITY etc.

1.3 Class

Should be in UpperCamelCase and be a noun
e.g. String, Color, Button, System, Thread etc.

1.4 Interface

Should be in UpperCamelCase and be an adjective
e.g. Runnable, Remote, ActionListener etc.

1.5 Method

Should be lowerCamelCase and be a verb
e.g. actionPerformed(), main(), print(), println() etc.

1.6 Variable

Should be in lowerCamelCase
e.g. firstName, orderNumber, btnLoad etc.

1.6.2 member variable

prefix with an underscore e.g. _firstName

1.7 Items in XML

IDs need to make sense e.g. a login button should not be named face
Words should be separated by underscores, do not use camelcase (within Java files follow 1.6)

If there is more than one of an element on the screen, do not number them, name them
e.g. btn_load and btn_clear NOT btn_1 and btn_2

If there is only one of an element it is allowed to be named what it is e.g. list_view

1.7.1 Buttons

btn_thing

1.7.2 Text View

txt_thing

1.7.3 Edit Text

etxt_thing

2. Text

All text should be included in the strings.xml file, and not hardcoded

3. Braces

3.1 Conditionals

Braces should be used on conditional statements (if, while, for etc) even when there is only one contained line to prevent confusion

3.2 General

No line break before the opening brace.
Line break after the opening brace.
Line break before the closing brace.
Line break after the closing brace, only if that brace terminates a statement or terminates the body of a method, constructor, or named class. For example, there is no line break after the brace if it is followed by else or a comma.

Figure 3.1

```
void function() {
      If (true) {
            // does some stuff
      } else
            //does some other stuff
      }
}

void function2 () {

}
```

3.3 Empty blocks

Empty blocks may follow above conventions or may be closed immediately after opening

Figure 3.2 Both these formats are acceptable for empty blocks

```
void emptyFunction() {
}

void emptyFunction() {}
```

4. Whitespace

4.1 Vertical whitespace

Single blank line:

1. Between consecutive members or initializers of a class: fields, constructors, methods, nested classes, static initializers, and instance initializers.

    Exception: A blank line between two consecutive fields (having no other code between them) is optional. Such blank lines are used as needed to create logical groupings of fields.

2. Between statements, as needed to organize the code into logical subsections.

4.2 Horizontal Whitespace

See Google Java style guide for additional guidelines

Single space between

1. Separating any reserved word, such as if, for or catch, from an open parenthesis (() that follows it on that line
2. Separating any reserved word, such as else or catch, from a closing curly brace (}) that precedes it on that line
3. Around operators: e.g. `i ++, num = 5`
4. Between the type and variable of a declaration: `List<String> list`

5. Coding Practices

5.1 Variable Declarations

Multiple declarations permitted per line if no assignments are being made.

Figure 5.1

```
// permitted
int a, b;
int c = 10;

// not permitted
int a = 0, b, c = 10;
```

5.2 @Override: always used

A method is marked with the @Override annotation whenever it is legal. This includes a class method overriding a superclass method, a class method implementing an interface method, and an interface method respecifying a superinterface method.

Exception: @Override may be omitted when the parent method is @Deprecated.

6. Javadoc

6.1 General formatting

The basic formatting of Javadoc blocks is as seen in this example:

Figure 6.1 Javadoc Formatting

```
/**
 * Multiple lines of Javadoc text are written here,
 * wrapped normally...
 */
public int method(String p1) { ... }
```

... or in this single-line example:

```
/** An especially short bit of Javadoc. */
```

The basic form is always acceptable. The single-line form may be substituted when the entirety of the Javadoc block (including comment markers) can fit on a single line. Note that this only applies when there are no block tags such as @return.

6.2 Block tags

Any of the standard "block tags" that are used appear in the order @param, @return, @throws, @deprecated, and these four types never appear with an empty description. When a block tag doesn't fit on a single line, continuation lines are indented four (or more) spaces from the position of the @.

6.3 The summary fragment

Each Javadoc block begins with a brief summary fragment. This fragment is very important: it is the only part of the text that appears in certain contexts such as class and method indexes. This is a fragment—a noun phrase or verb phrase, not a complete sentence.

Tip: A common mistake is to write simple Javadoc in the form `/** @return the customer ID */`. This is incorrect, and should be changed to `/** Returns the customer ID. */`.

6.4 Where Javadoc is used

At the minimum, Javadoc is present for every public class, and every public or protected member of such a class, with a few exceptions noted below.

6.5 Exceptions

6.3.1 Exception: self-explanatory methods

Javadoc is optional for "simple, obvious" methods like getFoo, in cases where there really and truly is nothing else worthwhile to say but "Returns the foo".

Important: it is not appropriate to cite this exception to justify omitting relevant information that a typical reader might need to know. For example, for a method named getCanonicalName, don't omit its documentation (with the rationale that it would say only /** Returns the canonical name. */) if a typical reader may have no idea what the term "canonical name" means!

6.3.2 Exception: overrides

Javadoc is not always present on a method that overrides a supertype method.

Resources

https://www.javatpoint.com/java-naming-conventions
https://google.github.io/styleguide/javaguide.html