

# CLEAN CODE BY CONVENTION?!

AN EXPERIENCE REPORT BY THE  
EXAMPLE OF ANDROID LINT.



# ANDRÉ DIERMANN

Software Architect @ [it-objects](#)

# AGENDA

- Introduction
- Android Lint
- Clean Code  
by Convention?

# INTRODUCTION

- Motivation
- Clean Code
- Android Lint

# OUR MOTIVATION

## CODE QUALITY FLAWS DUE TO

- heterogenous team
- working distributed over Europe
- extreme agility
- framework limits
- huge code base

# OUR MOTIVATION

HOW TO MAKE THE CODE CLEAN(ER)?

- harmonize the understanding
  - common conventions
- provide automated support
  - Static Code Analysis

# CLEAN CODE

- no unique definition
- "School of Thought"

# CLEAN CODE

*[...] Clean code always looks like it was written by someone who cares. [...]*

Michael Feathers

# ANDROID LINT

- tool for command-line and IDE
- scans all kind of development artifacts
- reports potential bugs, bad coding habits, broken conventions, ...
- features more than 200 built-in checks (October 2015)

# EXAMPLE

The screenshot shows the Android Studio IDE with the following components:

- Toolbar:** Includes icons for file operations, navigation, lint, and VCS.
- Breadcrumbs:** Shows the path: iosched > android > src > main > res > layout > activity\_session\_detail.xml.
- Editor:** Displays the XML code for `activity_session_detail.xml`. The code includes a comment `<!-- Session subtitle -->` and an XML element `<TextView` with attributes: `android:layout_height="wrap_content"`, `android:layout_width="match_parent"`, `android:layout_marginLeft="@dimen/keyline_2"`, `android:layout_marginRight="16dp"`, `android:text="Trevor Johns and Roman Nurik"`, `android:id="@+id/session_subtitle"`, `android:maxLines="2"`, `android:ellipsize="end"`, `android:textSize="16sp"`, `android:textColor="@color/body_text_2_inverse"`, and `android:textAlignment="viewStart"`.
- Warning:** A yellow tooltip points to the `android:textAlignment="viewStart"` attribute, stating: "To support older versions than API 17 (project specifies 14) you must also specify gravity or layout\_gravity="start" more... (%F1)".
- Terminal:** Shows the command prompt: `andre-ito @ MacBookPro : ~/Documents/workspace-android-studio/iosched $`.
- Bottom Bar:** Contains tabs for Terminal, Version Control, Android, Messages, Run, TODO, Event Log, and Gradle Console. A status bar at the bottom indicates: "Gradle build finished with 2 warnings(s) in 15s 45ms (12 minutes ago) 129:1 LF UTF-8 Git: master".

# EXAMPLE

Lint Report

Check performed at Wed May 27 06:54:33 CEST 2015.  
45 errors and 421 warnings found:

Correctness

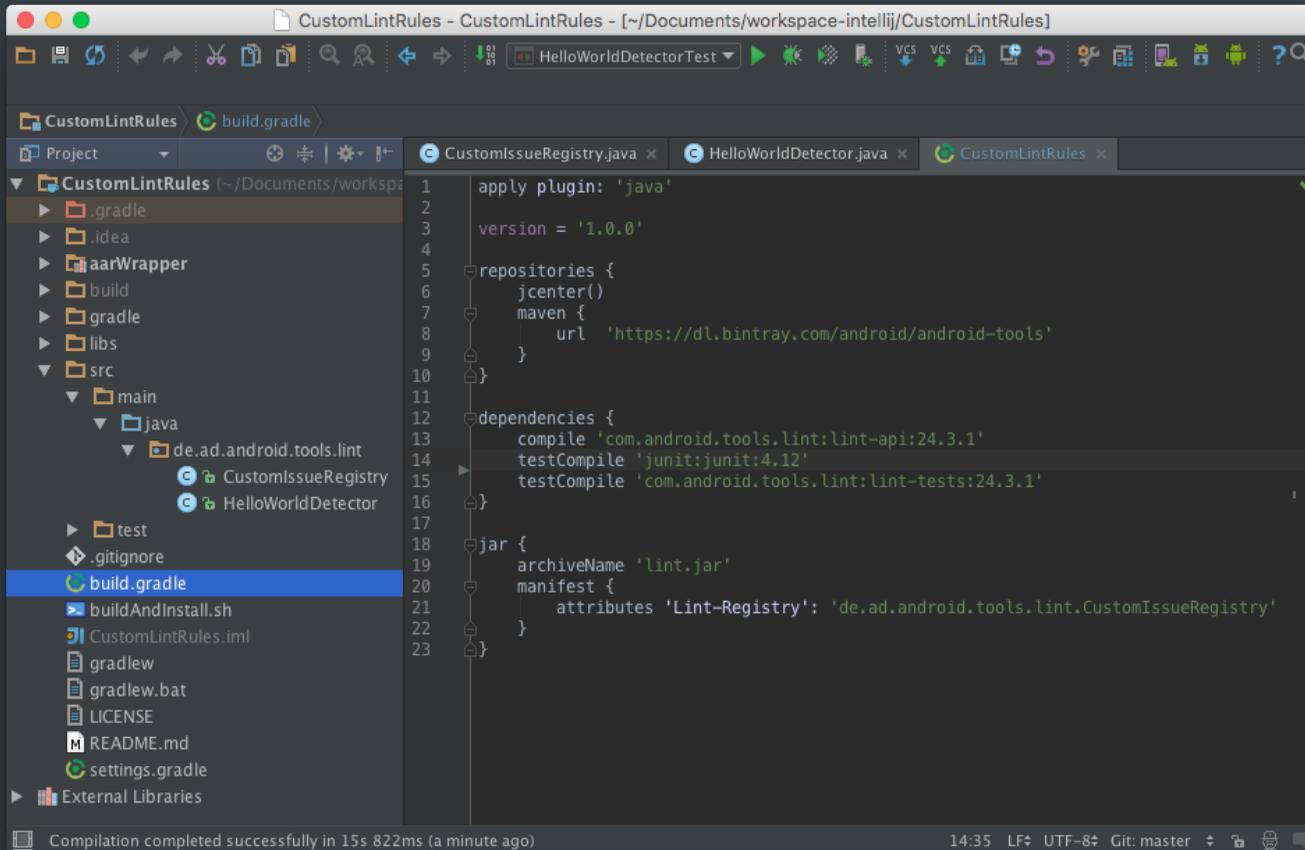
1	⚠ SuspiciousImport: 'import android.R' statement
2	❗ MissingRegistered: Missing registered class
6	⚠ AppCompatActivity: Using Wrong AppCompatActivity Method
29	⚠ CommitPrefEdits: Missing <code>commit()</code> ON <code>SharedPreferences</code> editor
1	⚠ CutPasteId: Likely cut & paste mistakes
4	⚠ DefaultLocale: Implied default locale in case conversion
7	⚠ InconsistentLayout: Inconsistent Layouts

# ANDROID LINT

- Creation + Verification
- Operation + Continuous Integration
- Real world examples

# CREATION

<https://github.com/a11n/CustomLintRules>



The screenshot shows an IDE window titled "CustomLintRules - CustomLintRules - [~/Documents/workspace-intellij/CustomLintRules]". The interface includes a toolbar at the top, a project explorer on the left, and a code editor on the right. The project explorer shows a directory structure for "CustomLintRules" with subdirectories like ".gradle", ".idea", "aarWrapper", "build", "gradle", "libs", "src", "test", ".gitignore", "build.gradle", "buildAndInstall.sh", "CustomLintRules.iml", "gradlew", "gradlew.bat", "LICENSE", "README.md", "settings.gradle", and "External Libraries". The code editor displays the content of "build.gradle":

```
1  apply plugin: 'java'
2
3  version = '1.0.0'
4
5  repositories {
6      jcenter()
7      maven {
8          url 'https://dl.bintray.com/android/android-tools'
9      }
10 }
11
12 dependencies {
13     compile 'com.android.tools.lint:lint-api:24.3.1'
14     testCompile 'junit:junit:4.12'
15     testCompile 'com.android.tools.lint:lint-tests:24.3.1'
16 }
17
18 jar {
19     archiveName 'lint.jar'
20     manifest {
21         attributes 'Lint-Registry': 'de.ad.android.tools.lint.CustomIssueRegistry'
22     }
23 }
```

The status bar at the bottom indicates "Compilation completed successfully in 15s 822ms (a minute ago)" and "14:35 LF+ UTF-8+ Git: master".

# CREATION

## CORE PRINCIPLES

```
21 import ...
22
23 /**
24  * Check which determines if application title equals "Hello world"
25  */
26 public class HelloWorldDetector extends Detector
27     implements Detector.XmlScanner {
28
29     public static final Issue ISSUE = Issue.create(
30         "HelloWorld", //ID
31         "Unexpected application title", //brief description
32         "The application title should state 'Hello world'", //explanation
33         Category.CORRECTNESS, //category
34         5, //priority
35         Severity.INFORMATIONAL, //severity
36         new Implementation( //implementation
37             HelloWorldDetector.class, //detector
38             Scope.MANIFEST_SCOPE //scope
39         );
40
41     private static final String TITLE = "Hello world";
42
43     @Override
44     public boolean appliesTo(@NonNull Context context, @NonNull File file) {...}
45
46     @Override public Collection<String> getApplicableAttributes() {...}
47
48     @Override public void visitAttribute(@NonNull XmlContext context,
49         @NonNull Attr attribute) {...}
50
51
52
53
```

# CREATION

## CORE PRINCIPLES

```
import java.util.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import com.android.tools.lint.*;
import com.android.tools.lint.detector.*;

/**
 * Check which determines if application title equals "Hello world"
 */
public class HelloWorldDetector extends Detector
    implements Detector.XmlScanner {

    public static final Issue ISSUE = Issue.create(
        "HelloWorld",
        "Unexpected application title",
        "The application title should state 'Hello world'",
        Category.CORRECTNESS,
        5,
        Severity.INFORMATIONAL,
        new Implementation(
            HelloWorldDetector.class,
            Scope.MANIFEST_SCOPE
        )
    );

    private static final String TITLE = "Hello world";

    @Override
    public boolean appliesTo(@NonNull Context context, @NonNull File file) {...}

    @Override
    public Collection<String> getApplicableAttributes() {...}

    @Override
    public void visitAttribute(@NonNull XmlContext context,
        @NonNull Attr attribute) {...}
}
```

The screenshot shows the IDE interface with the following elements:

- Project View:** Shows the project structure with folders like `src`, `main`, `java`, `de`, `ad`, `android`, `tools`, `lint`, and `HelloWorldDetector`.
- Code Editor:** Displays the `IssueRegistry` file with the `Issue` class definition. Red circles highlight the `Scanner`, `Detector`, and `Issue` annotations.
- Toolbar:** Includes icons for Ant build, Gradle, Maven Projects, and Z-Structure.
- Status Bar:** Shows the current file encoding as UTF-8 and the Git branch as master.

# CREATION

## CORE PRINCIPLES OF THE LINT API

- Issue
- Detector
- Scanner
- IssueRegistry

*Reference guide:*

<https://github.com/a11n/android-lint>

# VERIFICATION

## TEST

```
public class HelloWorldDetectorTest extends LintDetectorTest {
    //Specify the detector under test
    @Override protected Detector getDetector() { ... }

    //Specify the issues to report
    @Override protected List<Issue> getIssues() { ... }

    //Perform test
    public void test() throws Exception {
        //assert that linting a given set of files
        //returns the expected output
        assertEquals(EXPECTED_OUTPUT, lintFiles(FILE));
    }
}
```

# VERIFICATION

## ASSERT

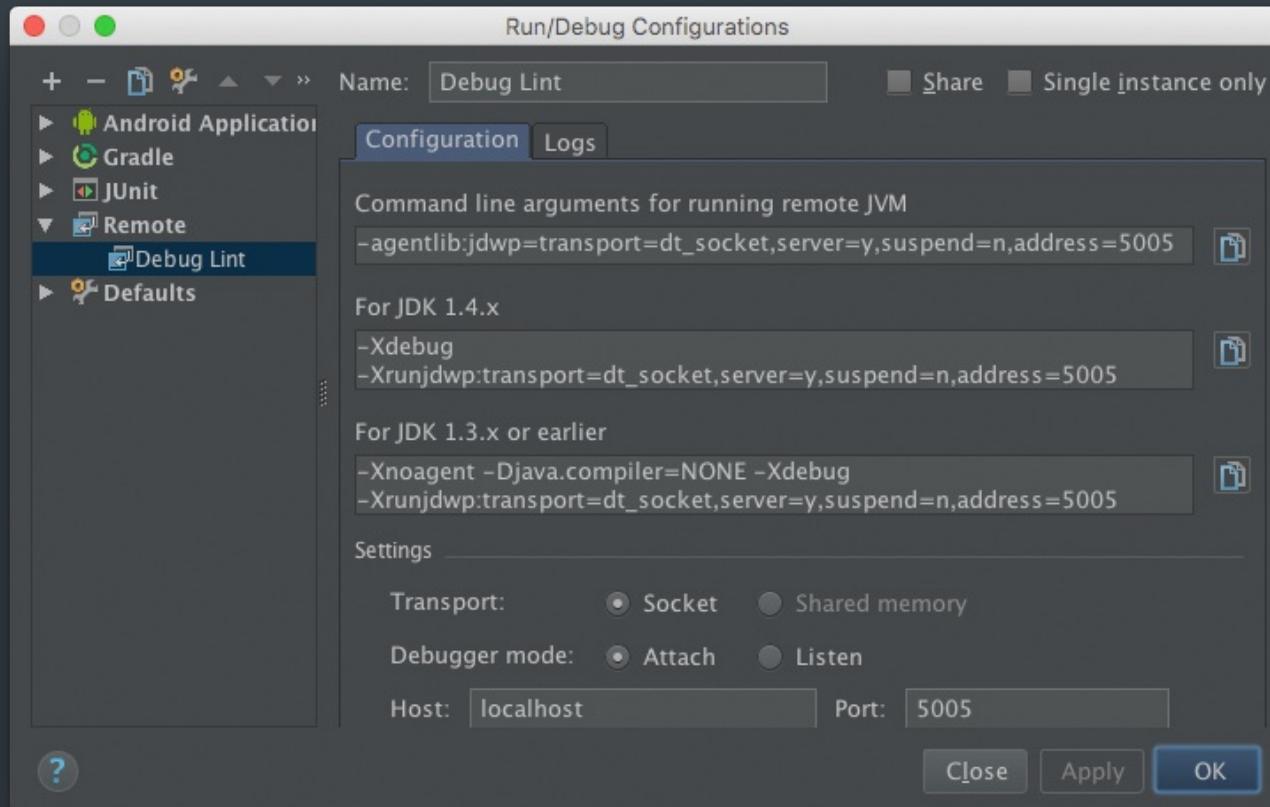
```
public void testShouldDetectWarning() throws Exception {
    assertEquals(
        "AndroidManifest.xml:8: Information: Unexpected title \""@string
+ "        android:label=\"@string/app_name\"\\n"
+ "        ~~~~~\\n"
+ "0 errors, 1 warnings\\n",
        lintFiles("InvalidAndroidManifest.xml=>AndroidManifest.xml"));
}
```

*JUnit4 & better assertions:*

<https://github.com/a11n/lint-junit-rule>

# VERIFICATION

## DEBUG



# OPERATION

## CONFIGURATION

- build.gradle

```
android {  
    lintOptions { ... }  
}
```

- lint.xml

```
<lint>  
    <issue id="IconMissingDensityFolder" severity="ignore" />  
</lint>
```

- Java / Resource

```
@SuppressWarnings("TheIssueYouWantToSuppress")
```

```
tools:ignore="TheIssueYouWantToSuppress"
```

# OPERATION

## APPLICATION

- Have a Java module for the Lint rules and an Android library module as wrapper

```
Android application project
--app          //default Android application module
--lint        //Android library, acts as wrapper for the Lint ru
--lintrules   //Java module with your custom Lint rules
```

```
project.afterEvaluate {
def compileLint = project.tasks.getByPath(':lint:compileLint')
compileLint.dependsOn ':lintrules:jar'
compileLint << {
    copy{
        from '../lintrules/build/libs'
        into 'build/intermediates/lint'
    }
}
}
```

# CONTINUOUS INTEGRATION

<https://github.com/a11n/docker-jenkins-android-lint>

The screenshot displays the Jenkins web interface for a project named 'Build and lint Android application'. The main content area features a 'Lint Trend' graph showing a decreasing trend in lint issues over four builds. The y-axis is labeled 'count' and ranges from 0 to 16. The x-axis is labeled with build numbers #1, #2, #3, and #4. The graph shows a yellow area representing the total count of lint issues, which starts at approximately 16 for build #1 and decreases to about 7 for build #4. A blue area at the bottom of the graph represents the number of lint failures, which remains very low throughout the builds.

Build Number	Time
#4	20.10.2015 07:35
#3	20.10.2015 07:17
#2	20.10.2015 07:11
#1	20.10.2015 06:38

Additional interface elements include a sidebar with navigation options like 'Back to Dashboard', 'Status', 'Changes', 'Workspace', 'Build Now', 'Delete Project', 'Configure', and 'Lint Issues'. The top navigation bar includes a search bar and a 'Disable Project' button. The bottom of the interface features links for 'Workspace', 'Recent Changes', 'RSS for all', and 'RSS for failures'.

# REAL WORLD EXAMPLES

- Naming conventions
- Logging
- Smell detection

# CLEAN CODE BY CONVENTION?

- Challenges
- Conventions
- Chances + Limits
- Discussion

# CHALLENGES

## INTRODUCING CLEAN CODE

- Where to start?
- How to convince team members?
- How to support team members?
- How to evolve team culture?
- ...

# CONVENTIONS

*Conventions are agreements on  
how to do certain things.*

*Synonyms: Guidelines, Principles, Best Practices, ...*

# CONVENTIONS

## RELATION TO CLEAN CODE

- "Conventions are part of Clean Code."
- "Clean Code is expressed by conventions."
- "Clean Code is a set of conventions."

# CHANCES

- easy to understand, communicate and validate
- trigger awareness for Clean Code

# CHANCES

CLEAN CODE



CONVENTIONS

# LIMITS

CONVENTIONS

≠

CLEAN CODE

# DISCUSSION

- Do you have dedicated conventions in your team, project or company?
- How do you explain/teach clean code to a Junior Developer?

THANK YOU FOR YOUR ATTENTION.

Q&A



q2ad



a11n