

# CUSTOM LINT RULES

## A JOURNEY TOWARDS CLEANER CODE



# ANDRÉ DIERMANN

Software Architect @ [it-objects](#)

# AGENDA

- Introduction
- Creation
- Verification
- Operation
- Q & A

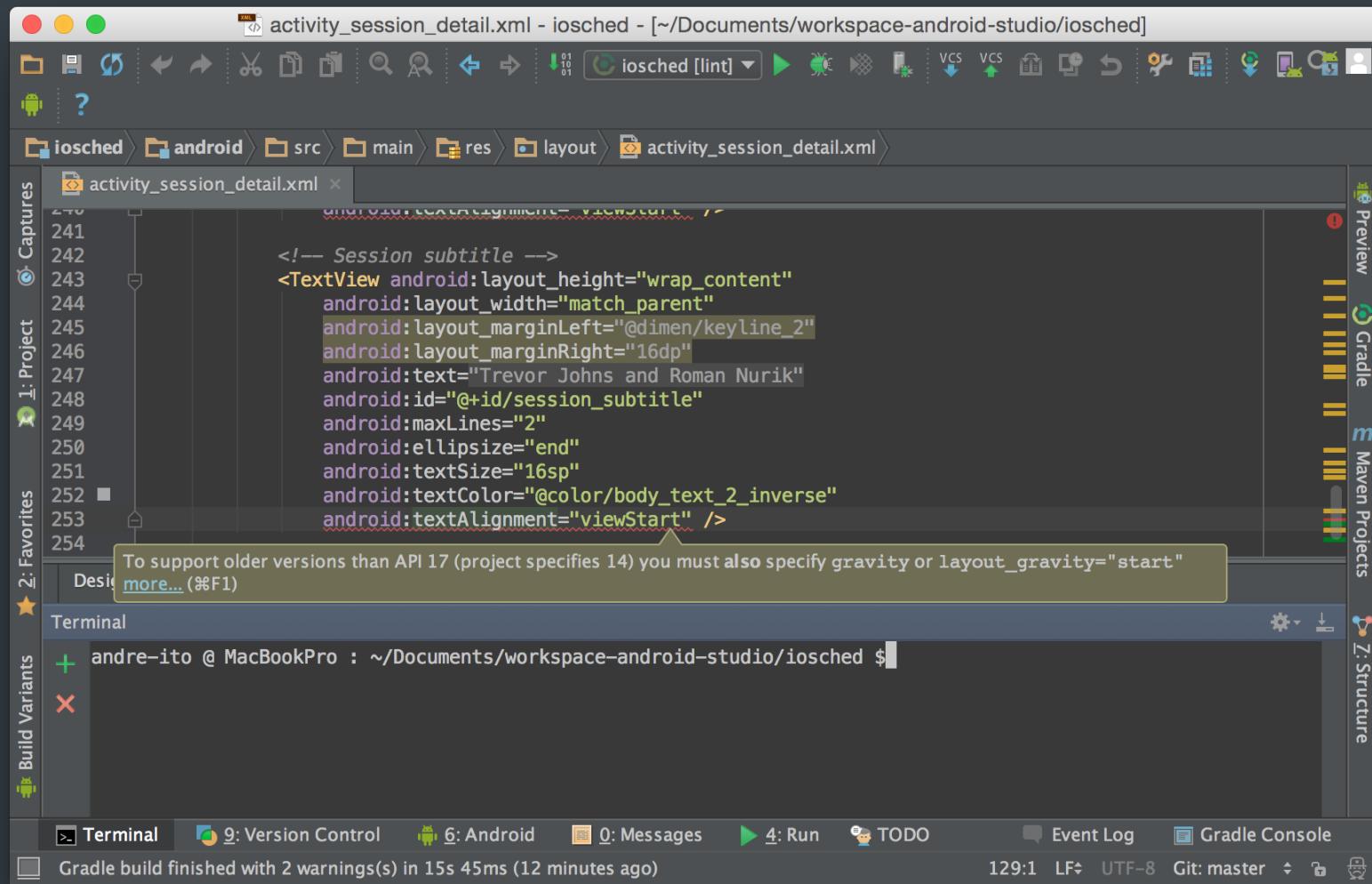
# INTRODUCTION

- Lint
- Custom rules

# 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



# EXAMPLE

A screenshot of a web browser window displaying an Lint Report. The browser has a dark theme with light-colored buttons. The address bar shows the URL `/build/outputs/lint-results.html`. The main content area has a black background with white text. It starts with the title "Lint Report" in large font, followed by a horizontal line. Below that, it says "Check performed at Wed May 27 06:54:33 CEST 2015." and "45 errors and 421 warnings found:". Another horizontal line follows. Then there is a section titled "Correctness" with a list of issues:

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

# CUSTOM RULES

## MOTIVATION

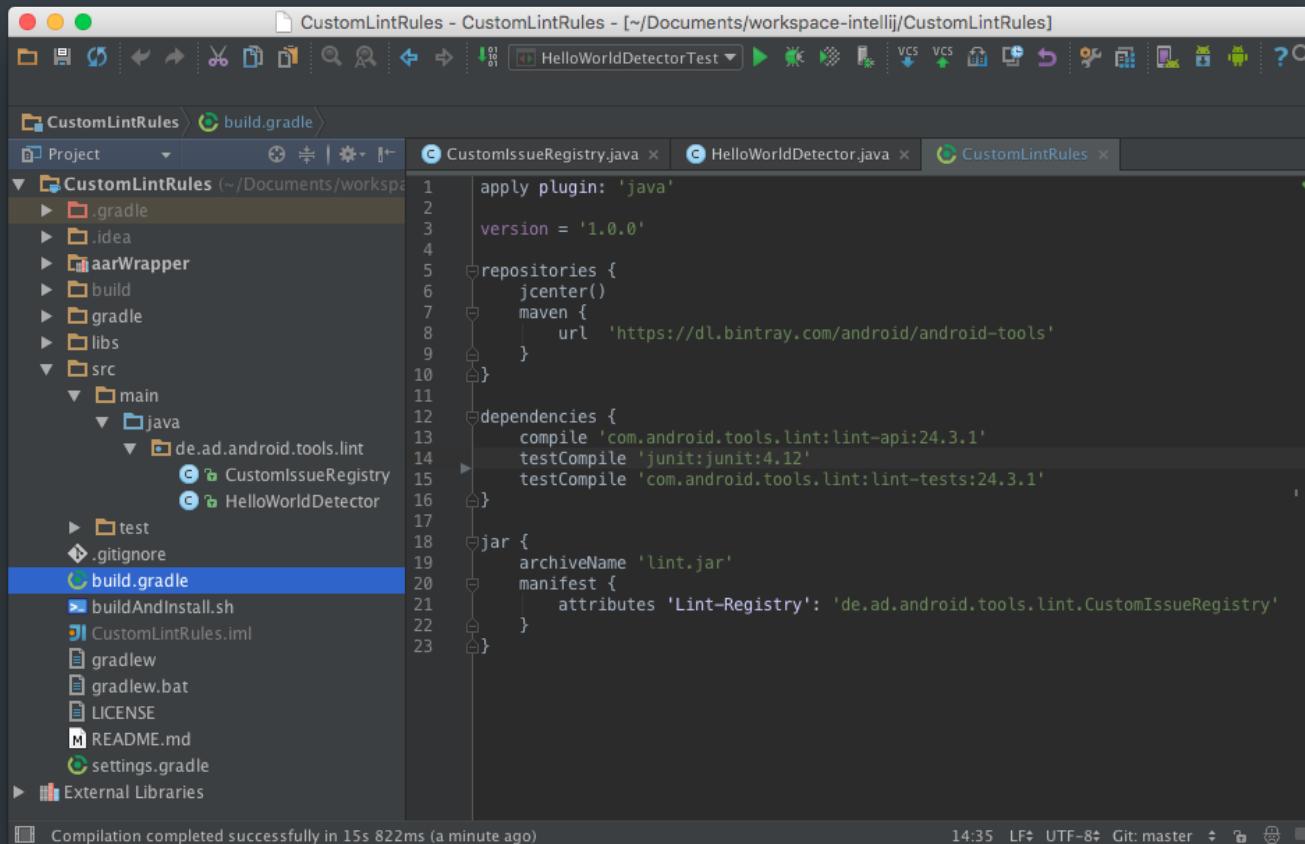
- work in large and distributed teams requires dedicated conventions
- huge code bases require automated checks
- need for 'Android specific' validations  
(vs. checkstyle, FindBugs, ...)
- ...

# CREATION

- Project setup
- Lint API

# PROJECT SETUP

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



The screenshot shows the IntelliJ IDEA IDE interface. The title bar reads "CustomLintRules - CustomLintRules - [~/Documents/workspace-intellij/CustomLintRules]". The left sidebar displays the project structure under "CustomLintRules (~/Documents/workspace-intellij/CustomLintRules)". The "build.gradle" file is selected and open in the main editor area. The code in the build.gradle file is as follows:

```
apply plugin: 'java'
version = '1.0.0'

repositories {
    jcenter()
    maven {
        url 'https://dl.bintray.com/android/android-tools'
    }
}

dependencies {
    compile 'com.android.tools.lint:lint-api:24.3.1'
    testCompile 'junit:junit:4.12'
    testCompile 'com.android.tools.lint:lint-tests:24.3.1'
}

jar {
    archiveName 'lint.jar'
    manifest {
        attributes 'Lint-Registry': 'de.ad.android.tools.lint.CustomIssueRegistry'
    }
}
```

The status bar at the bottom indicates "Compilation completed successfully in 15s 822ms (a minute ago)" and shows the current time as "14:35".

# LINT API

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

# LINT API

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

The screenshot shows the IntelliJ IDEA interface with the project 'CustomLintRules' open. The 'CustomIssueRegistry.java' file is the active editor. Several words in the code are highlighted with red circles and bolded text:

- 'Scanner' is circled and bolded.
- 'Detector' is circled and bolded.
- 'Issue' appears twice in the code, both instances are circled and bolded.
- 'Title' is circled and bolded.
- 'Context' is circled and bolded.
- 'File' is circled and bolded.
- 'Attributes' is circled and bolded.
- 'Attr' is circled and bolded.

```
/*
 * 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(@NotNull Context context, @NotNull File file) {...}
    @Override
    public Collection<String> getApplicableAttributes() {...}
    @Override
    public void visitAttribute(@NotNull XmlContext context,
        @NotNull Attr attribute) {...}
}
```

# LINT API ISSUE

```
public static final Issue ISSUE = Issue.create(  
    "HelloWorld", //ID  
    "Unexpected application title", //brief description  
    "The application title should" //explanation  
        + " state 'Hello world'",  
    Category.CORRECTNESS, //category  
    5, //priority  
    Severity.INFORMATIONAL, //severity  
    new Implementation( //implementation  
        HelloWorldDetector.class, //detector  
        Scope.MANIFEST_SCOPE //scope  
    )  
);
```

# LINT API DETECTOR

```
public class HelloWorldDetector extends Detector
    implements XmlScanner {

    public static final Issue ISSUE = Issue.create(...);

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

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

    @Override public void visitElement(@NonNull XmlContext context,
        @NonNull Element element) {...}

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

# LINT API

## SCANNER

- JavaScanner
- ClassScanner
- BinaryResourceScanner
- ResourceFolderScanner
- XmlScanner
- GradleScanner
- OtherFileScanner

# LINT API SCANNER

**JavaScanner**

---

applicableSuperClasses( )

checkClass( . . . )

---

getApplicableMethodNames( )

---

visitMethod( . . . )

...

**XmlScanner**

---

getApplicableEle

visitElement( . . . )

---

getApplicableAtt

---

visitAttribute( . . . )

...

# LINT API

## ISSUEREGISTRY

```
public class CustomIssueRegistry extends IssueRegistry {  
    @Override  
    public List<Issue> getIssues() {  
        return Arrays.asList(  
            MyCustomCheck.ISSUE, //Note:  
            MyAdvancedCheck.AN_ISSUE, //A check actually is a detector.  
            MyAdvancedCheck.ANOTHER_ISSUE //One detector can report  
        ); //multiple types of issues.  
    }  
}
```

# VERIFICATION

- Testing
- Debugging

# TESTING

- Test
- Assert
- JUnit4

# 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(FILES));  
    }  
}
```

# ASSERT

Lint test library only allows assertion of Strings.

```
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"));  
}
```

**#ProTip:** *Let the test fail first and copy the assertion error to get the expected output.*

# ASSERT

Two options for test data

- Inline resources

```
assertEquals("No warnings.",  
    lintProject(  
        xml("AndroidManifest.xml", "..."),  
        java("src/main/java/test/MainActivity.java", "..."),  
        ...));
```

- External resources

```
assertEquals("No warnings.",  
    lintFiles(  
        "ValidAndroidManifest.xml=>AndroidManifest.xml", ...));
```

# JUNIT4

The official Android Lint test library only  
supports JUnit3 style.

- You have to extend `LintDetectorTest` and
- prefix your test methods with `test`.

# LINT-JUNIT-RULE

## MOTIVATION

- usage of JUnit4
- better assertions

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

# EXAMPLE

```
//Setup Detector and Issues
@Rule public Lint lint =
    new Lint(new MyCustomDetector(), MyCustomDetector.ISSUE);

@Test
public void test() throws Exception {
    //Lint the specified files from your test resources
    List<Warning> lintResult =
        lint.files("AndroidManifest.xml", "res/values/strings.xml");

    //Assertions
    //...
}
```

# EXAMPLE

```
//AssertJ
assertThat(lintResult)
    .hasWarnings(2)
    .in("AndroidManifest.xml", "strings.xml")
    .atLine(8, 14)
    .withMessage("MyCustomDetector warning message.",
                 "MyCustomDetector warning message.");

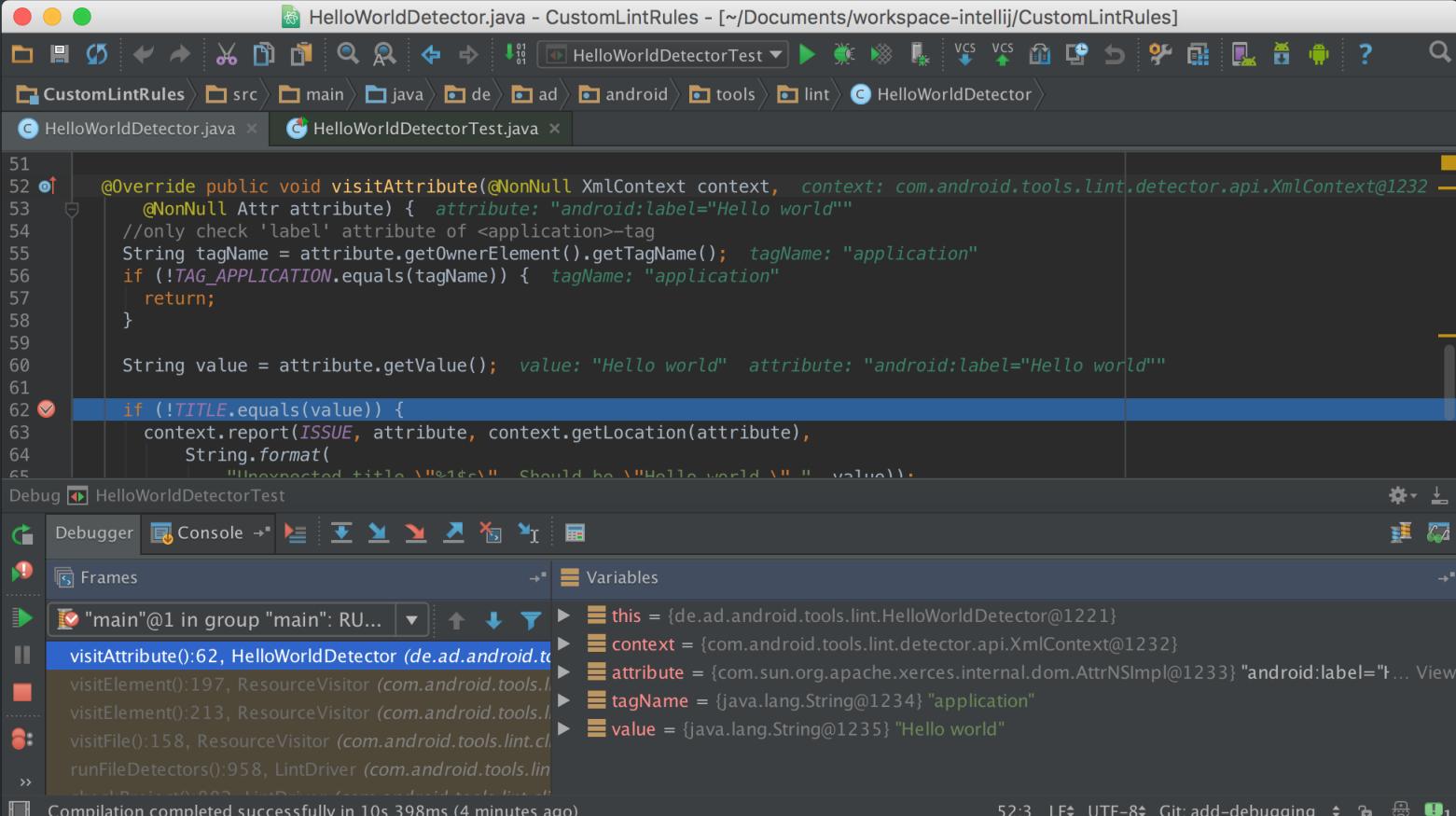
//Hamcrest
assertThat(lintResult,
           hasWarnings(
               in("AndroidManifest.xml", "strings.xml"),
               atLine(8, 14),
               withMessage("MyCustomDetector warning message.",
                           "MyCustomDetector warning message."))));
```

# DEBUGGING

- during testing
- during Gradle run

# DEBUGGING DURING TESTING

Works out of the box in IntelliJ and Android Studio.



The screenshot shows the IntelliJ IDEA interface during a debugging session. The top bar displays the project name "HelloWorldDetector.java - CustomLintRules" and the file path "~/Documents/workspace-intellij/CustomLintRules". The navigation bar shows the current file "HelloWorldDetectorTest.java" is selected. The code editor displays Java code for a custom lint rule, specifically the `visitAttribute` method. A breakpoint is set at line 62, which is highlighted in blue. The code checks if the attribute value is "Hello world" and reports an issue if it's not. The bottom status bar indicates "Compilation completed successfully in 10s 398ms (4 minutes ago)".

```
51
52     @Override public void visitAttribute(@NotNull XmlContext context, Context context: com.android.tools.lint.detector.api.XmlContext@1232
53             @NotNull Attr attribute) { attribute: "android:label=\"Hello world\""
54             //only check 'label' attribute of <application>-tag
55             String tagName = attribute.getOwnerElement().getTagName(); tagName: "application"
56             if (!TAG_APPLICATION.equals(tagName)) { tagName: "application"
57                 return;
58             }
59
60             String value = attribute.getValue(); value: "Hello world" attribute: "android:label=\"Hello world\""
61
62             if (!TITLE.equals(value)) {
63                 context.report(ISSUE, attribute, context.getLocation(attribute),
64                     String.format(
65                         "%nUnexpected title %s! Should be \"%s\".", value));
66         }
67     }
68 }
```

The debugger tool window is open, showing the call stack and variable values. The call stack shows the current frame is "visitAttribute():62, HelloWorldDetector (de.ad.android.to...)" and the previous frame is "visitElement():197, ResourceVisitor (com.android.tools.lint.detector.api.ResourceVisitor)". The variables pane shows the following:

- this = {de.ad.android.tools.lint.HelloWorldDetector@1221}
- context = {com.android.tools.lint.detector.api.XmlContext@1232}
- attribute = {com.sun.org.apache.xerces.internal.dom.AttributeImpl@1233} "android:label="Hello world"
- tagName = {java.lang.String@1234} "application"
- value = {java.lang.String@1235} "Hello world"

# DEBUGGING DURING GRADLE RUN

Credits to [Marc Prengemann](#)

- Add to `gradle.properties`

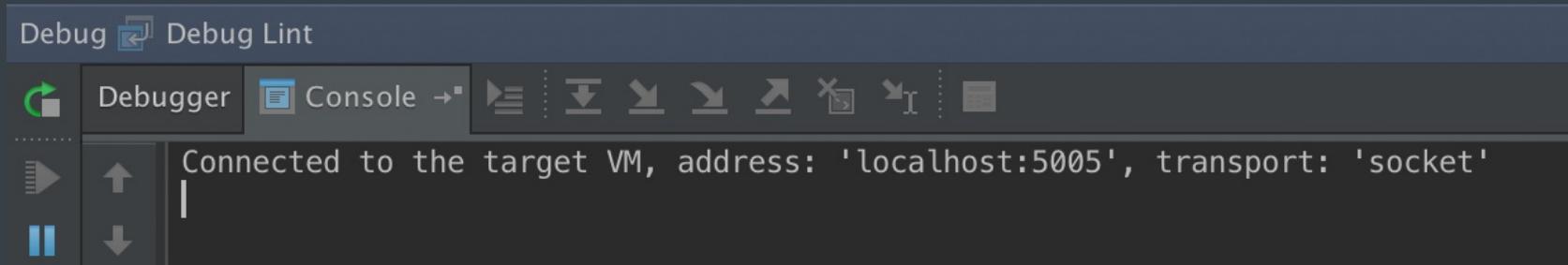
```
org.gradle.jvmargs=
  '-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5005'
```

- Start Gradle Daemon

```
./gradlew --daemon
```

# DEBUGGING DURING GRADLE RUN

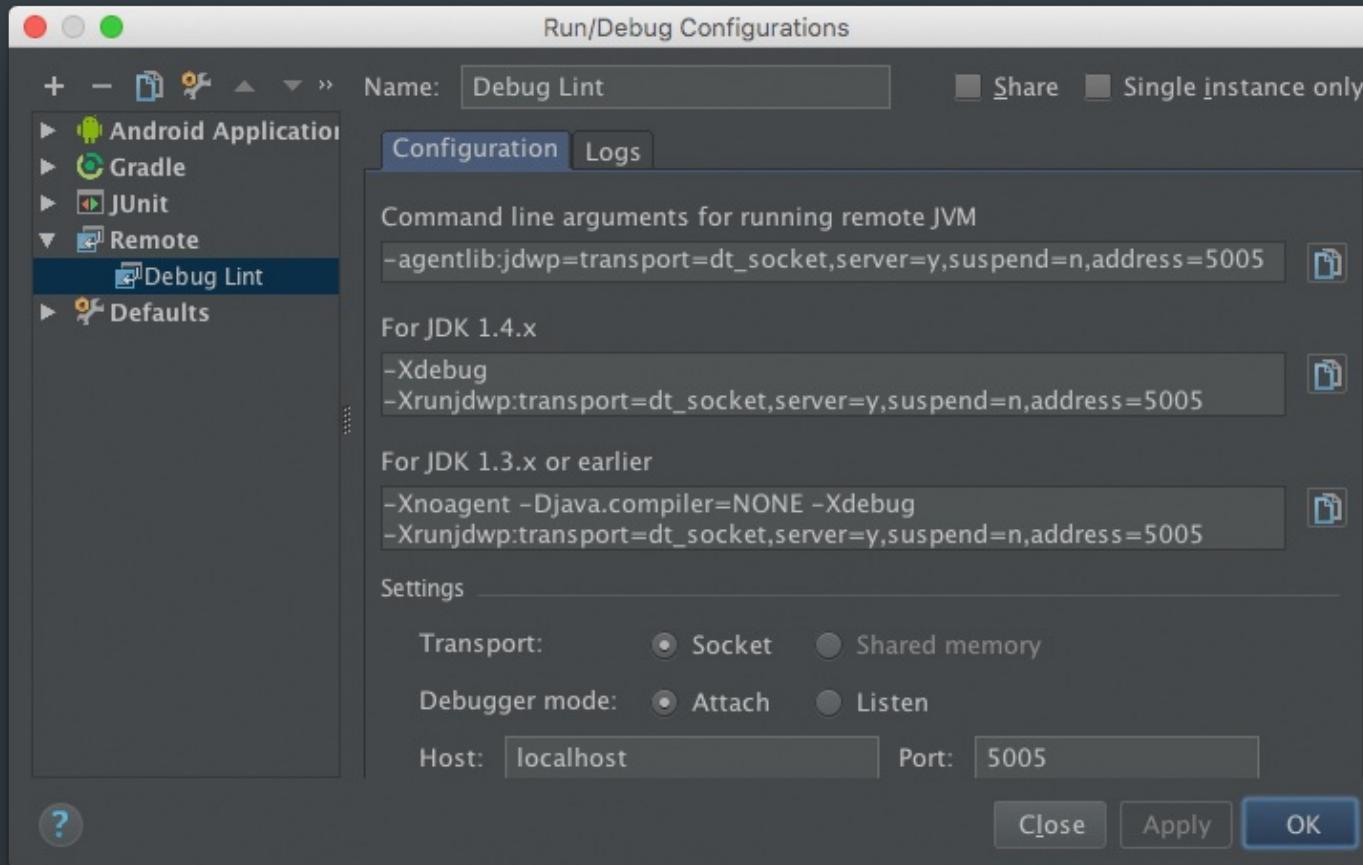
- Add and run new remote debug configuration



- Run Lint on your application project

```
./gradlew lint
```

# DEBUGGING DURING GRADLE RUN



# OPERATION

- Configuration
- Application
- Continuous Integration
- Discovery

# CONFIGURATION

Lint can be configured in different ways and at different levels using:

- Gradle
- File
- Other

# GRADLE-BASED CONFIGURATION

```
android {  
    lintOptions {  
        // set to true to turn off analysis progress reporting by l  
        quiet true  
        // if true, stop the gradle build if errors are found  
        abortOnError false  
        // if true, only report errors  
        ignoreWarnings true  
    }  
    ...  
}
```

Complete [list of Lint options](#).

# FILE-BASED CONFIGURATION

lint.xml in the root directory of the Android project

```
<?xml version="1.0" encoding="UTF-8"?>
<lint>
    <!-- Disable the given check in this project -->
    <issue id="IconMissingDensityFolder" severity="ignore" />

    <!-- Ignore the ObsoleteLayoutParams issue in the specified fil
    <issue id="ObsoleteLayoutParams">
        <ignore path="res/layout/activation.xml" />
        <ignore path="res/layout-xlarge/activation.xml" />
    </issue>

    <!-- Change the severity of hardcoded strings to "error" -->
    <issue id="HardcodedText" severity="error" />
</lint>
```

# OTHER

- in Java

```
@SuppressLint("TheIssueYouWantToSuppress")
```

- in XML

```
xmlns:tools="http://schemas.android.com/tools"
tools:ignore="TheIssueYouWantToSuppress"
```

# APPLICATION

- Basic approach
- Integrated approach

# BASIC APPROACH

- utilizes basic Lint extension feature
- two steps setup
  1. assemble custom Lint rules into JAR
  2. copy JAR to ~/.android/lint/

# INTEGRATED APPROACH

Credits to [Cheng Yang](#)

- uses AAR bundle as wrapper
- two steps setup
  1. wrap custom Lint rules into an AAR
  2. make application project depend on that AAR

# CONTINUOUS INTEGRATION

*Lint should be integrated into the  
Continuous Integration process!*

# RECOMMENDATIONS

- Be strict
- Be verbose
- Evolve team culture

# EXAMPLE

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

The screenshot shows a Jenkins project page for "Build and lint Android application".

**Left Sidebar:**

- Back to Dashboard
- Status
- Changes
- Workspace
- Build Now
- Delete Project
- Configure
- Lint Issues

**Build History:**

#	Date
#4	20.10.2015 07:35
#3	20.10.2015 07:17
#2	20.10.2015 07:11
#1	20.10.2015 06:38

[RSS for all](#) [RSS for failures](#)

**Project Build and lint Android application**

**Lint Trend**

A line chart titled "Lint Trend" showing the count of issues over time. The Y-axis is labeled "Count" and ranges from 0 to 16. The X-axis shows four time points. The trend starts at approximately 15.5, dips to about 12.5, then to 10.5, and finally levels off around 7.5.

[Add description](#) [Disable Project](#)

[Enlarge](#) [Configure](#)

**Links:**

- Workspace
- Recent Changes

# EXAMPLE

docker://jenkins/android/lint

## Jenkins

Jenkins Build and lint Android application #2

ENABLE AUTO REFRESH

[Back to Project](#)

[Status](#)

[Changes](#)

[Console Output](#)

[Edit Build Information](#)

[Delete Build](#)

[Git Build Data](#)

[No Tags](#)

[Lint Issues](#)

[Previous Build](#)

[Next Build](#)

### Build #2 (20.10.2015 07:11:44)

Started 33 min ago  
Took 1 min 22 sec

[add description](#)

 Changes

1. Fixed AllowBackup Lint violation. Fixed ContentDescription Lint ([detail](#))

 Revision: 72360643fd0b195ffff20bc3acb54254dfe4a858

- refs/remotes/origin/master

 Lint: [13 issues](#).

- [One new issue](#)
- [4 fixed issues](#)

[Help us localize this page](#)

Page generated: 20.10.2015 07:45:02 REST API Jenkins ver. 1.609.3

# DISCOVERY

## BACKGROUND

- Linting a pure scaffolded application project discovers  
**~50 Lint violations**
- Running Lint on a real world project *discovers*  
**>5000 Lint violations**

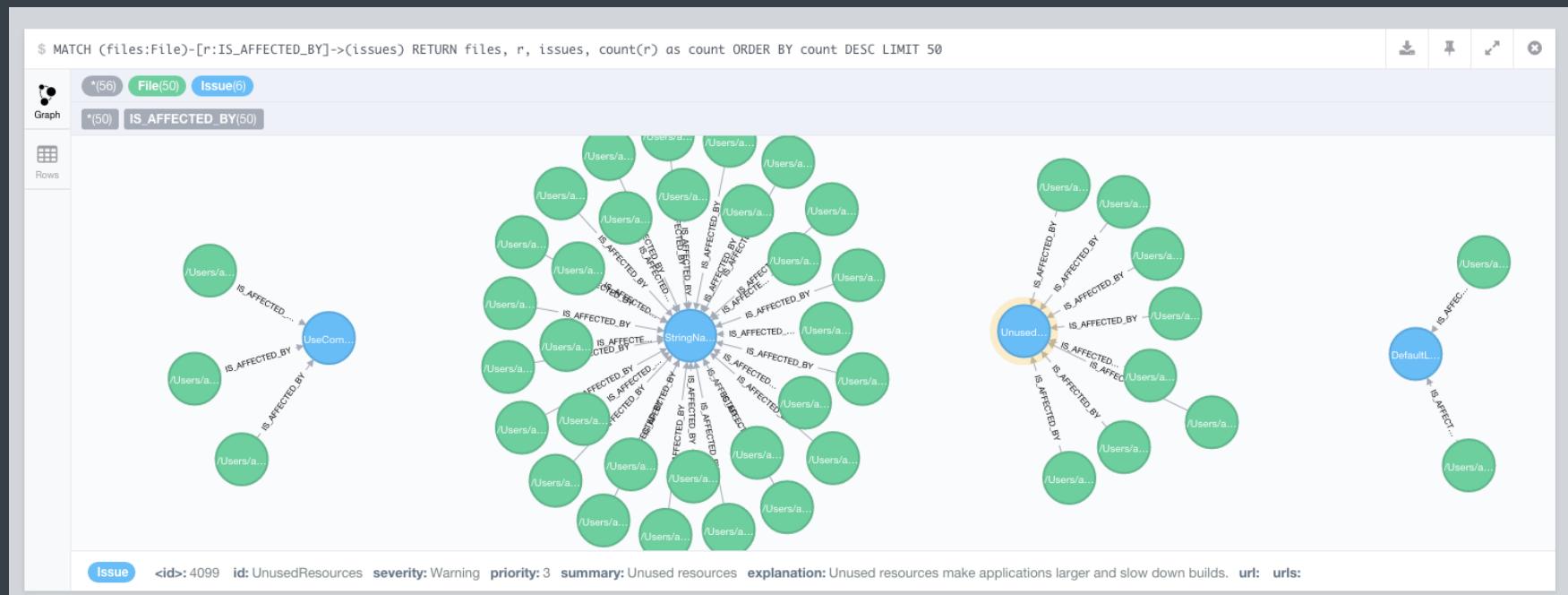
*Challenge:*

*Discover relevant information.*

# DISCOVERY

## LINT-GRAFH

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



# DISCOVERY

## LINT-BROWSER

:( *Not open-sourced yet.* :(

- Angular Material front-end
- parses `lint-results.xml`
- allows grouping, filtering, sorting, ...
- supports discovery of relations, potential sources, ... of issues.

# FUTURE WORK

- Custom SupportAnnotations
- Custom QuickFixes
- Kotlin support
- ...

THANK YOU FOR YOUR ATTENTION.

# Q&A

 q2ad  a11n