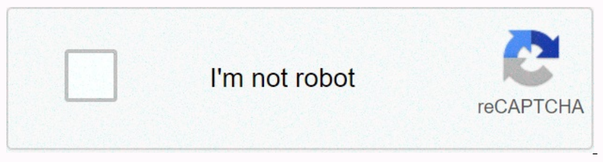

Continue

Gradle checkstyle report xml

I'll show here how to use checkstyle plugins in gradle-based Java projects. Checkstyle is a static code quality tool used to find the complex fan-out of a class, complexity of methods, utility layers, errors due to non-follow encryption standards. So you can remove the complexity from your Java source code and make the code easier to read and maintain for the future. You may also want to read quality codes using SonarQube or Jacoco in Gradle Projects.What is style checking? Checkstyle is a static code analysis tool used in software development to check if Java source code complies with encryption rules. The programming style adopted by a software development project can help adhere to good programming practice that improves code quality, readability, reusability, and reduces development costs. The test performed primarily limits itself to presentation and non-analysis of content, and does not confirm the accuracy or completeity of the program. In fact, it can be tedious to adhere to all style limitations, some of which can be harmful to the dynamics of the programming stages; Therefore, it may be useful to determine the level of testing that is required for a certain type of program. Checkstyle defines a set of available modules, each of which provides a rule check with a configurable level of stringentness (required, optional...). Each rule can increase notifications, alerts, and errors. For example, the test style can test:Javadoc comments for classes, attributes and methods; Naming conventions of properties and methods; Limit the number of function parameters, line length; The presence of mandatory headlines; The use of import packages, classes, modifying the scope and guiding blocks; The distance between some characters; Good classroom building practice; Many measurements are complex, including expressions. What is a checkstyle plugin? The Checkstyle plugin performs quality checks on the project's Java source files using Checkstyle and generates reports from these tests. Here we will see how to use the checkstyle plugin in gradle-based projects. Apply checkstyle PluginThe Checkstyle plugin adds the following tasks to the project. To use the test style in a gradle-based Java project, you need to use the test type plugin in the following way: plugin { id 'checkstyle' }Orapply plugin: 'checkstyle'To check such rules, you first define such rules usually in the type of check.xml files and if you want to prevent or avoid any rules defined in the check style.xml file as An exception to any class or module then you can define the suppression rules in checkstyle-suppressions.xml files and include these files into the check.xml-file style. By default, the Checkstyle plugin expects the configuration files to be placed in the original project, but this can be change and you need to configure in the build.gradle script. The default location of .xml is <root project= directory=>/config/checkstyle/checkstyle.xml. If you have a check-suppressions.xml or suppression style.xml file </root> </root> you can also place this file under the configuration directory/checkstyle. If you don't want to use the default location of the test type configuration file then you can create a check type configuration folder and place the file that configures your check type. Assuming you've created quality/style check folders under the original directory of the project, you've set your test style.xml and .xml goals. In this case you need the following configurations in your build.gradle script: def configDir = ${project.rootDir}/quality style check { toolVersion '8.36.1' configFile file($configDir/checkstyle/checkstyle.xml) configProperties.checkstyleSuppressionsPath = file($configDir/checkstyle/suppressions.xml).absolutePath } checkstyleMain { source ='src/main/java' } checkstyleTest { source ='src/test/java' }In the above configurations, I suppose you do not include suppressions.xml files in checkstyle files.xml. If you include .xml files into test style.xml then you don't need to configure it separately in build.gradle script. Please find in the section below how to include suppression.xml or checkstyle-suppressions.xml the .xml check. Include other configuration files into the test style.xml fileI will use sun test style files in this example and can be downloaded later from the source code section. Add the following XML tag at the bottom of the content in a test style.xml but before the last line (),<module name=SuppressionFilter> <property name=file value=quality/checkstyle/suppressions.xml>></property> </module>where res on resance measures.xml<module name=SuppressionFilter> <property name=file value=${config_loc}/suppressions.xml>>property> </module>is your suppression configuration file: or if you have a configuration file in the default location, you can use it below: Where is the error report created? Error reports are generated at the <roor project's= directory=>/build/reports/checkstyle location and you'll find the generated HTML and XML files, where you'll find all the errors listed. You can check them one by one and solve them. Let's move on to the example... Set up a project Prerequisite parcel you must first set up a project environment so that you can write the code needed to use the type of check in your application. Build.gradle scenarios are given with the necessary configurations as shown below. I have specified the JDK version I will use for this application. Note I've used the checkstyle plugin and I'm using the default location of the checkstyle file, so I don't need to configure the checkstyle location in the building file explicitly.plugins { id 'java-library' id 'checkstyle' } sourceCompatibility = 12 targetCompatibility = 12 reps { jcenter() } dependencies { } Recommended reading: Utility classes should not have a finished public builder or defaultJava Classl shape the checkstyle plugin and create gradle based on the Java project. Now I need to</roor> must</roor> a Java class to check how checkstyle rules are applied to the Java class and which rules are being violated and under which I need to correct the violations.package com.roytuts.java.project.checkstyle; public class CheckStyleApp { public static void main(String[] args) { System.out.println(Hello, CheckStyle); } } The ApplicationNow test navigates to the original project directory using cmd prompts and performs clean gradle commands for building or clean gradlew builds. You'll see some errors in the report. You can open the error reporting file (java-project-checkstyle/build/reports/checkstyle/main.html) and check what errors you've encountered in your Java file. When you open the main report file.html you see the following error:Missing package information.java file. Utility classes should not have a public or default builder. The file contains tab characters (this is the first case). Missing a Javadoc comment. The args parameters should be final. Now it's time to fix the problem. First, add the document above the layer as shown below: package com.roytuts.java.project.checkstyle; /*** *@author Soumitra **/ Final Class Public CheckStyleApp {Next add the package below information.java files in the same package of the project. Make sure you don't create any Java layers for this, but you should create normal files./** **@author Soumitra**/pack com.roytuts.java.project.checkstyle; The next fix is to make the final grade as you won't create any cases from it and it will fix the second problem (class utilities should not have a public builder or default.). So in the example above I made the final class. Now I've also added JavaDoc but you'll still get this error if you build the app. To completely eliminate this error, you also need to add JavaDoc to the main method. The complete class with repairs is given below: package com.roytuts.java.project.checkstyle; /*** * @author Soumitra * **/ Final class Public CheckStyleApp { checkstyleapp private() { } /** * @param args */ public void main (final String[] args) { System.out.println (Hello, CheckStyle); } } So I added JavaDoc above the main() method, and I also created the main() method parameters. The only problem now left is that the file contains tab characters (this is the first case). Now you need to go to your IDE and change the style format from tab to space. For Eclipse, you can do so by visiting navigation: Preference -> Java -> Code Style -> Formatter. Edit Active profile. (If you don't want to edit the built-in profile, create a new one.) In Active Profile: there is a drop-down dialog box.. Next to it, click Edit... opens the edit dialog box. In the Indents tab, you'll find Tab Policy. Change Tab Policy from Hybrid to Space & apply only changes. Now test applications from cmd prompts by performing the same clean gradle command construction. Now you won't see any errors in the cmd dashboard or in the report file. Fox. The report file now looks similar to the one below: That's all. You can use the checkstyle plugin in the same way I explained above for any of your Java-based applications. Source CodeSum Down Thank you for reading. Read.