|  |
| --- |
|  |
| ISC Integration with CheckStyle |
|  |
|  |
| **Ali Bidabadi** |
| **6/19/2015** |

|  |
| --- |
|  |

Contents

[Overview: 3](#_Toc422903090)

[Document Organization: 3](#_Toc422903091)

[File Level Checks: 3](#_Toc422903092)

[CheckStyle Module: FileTabCharacter 3](#_Toc422903093)

[CheckStyle Module: FileLength 3](#_Toc422903094)

[CheckStyle Module: JavadocPackage 4](#_Toc422903095)

[Checks for size violations: 4](#_Toc422903096)

[CheckStyle Module: MethodLength 4](#_Toc422903097)

[CheckStyle Module: ParameterNumber 4](#_Toc422903098)

[CheckStyle Module: LineLength 4](#_Toc422903099)

[Checks for imports: 5](#_Toc422903100)

[CheckStyle Module: AvoidStarImport 5](#_Toc422903101)

[CheckStyle Module: IllegalImport 5](#_Toc422903102)

[CheckStyle Module: RedundantImport 5](#_Toc422903103)

[CheckStyle Module: UnusedImports 6](#_Toc422903104)

[Checks for blocks: 6](#_Toc422903105)

[CheckStyle Module: AvoidNestedBlocks 6](#_Toc422903106)

[CheckStyle Module: EmptyBlocks 6](#_Toc422903107)

[CheckStyle Module: LeftCurly 7](#_Toc422903108)

[CheckStyle Module: RightCurly 7](#_Toc422903109)

[CheckStyle Module: NeedBraces 7](#_Toc422903110)

[Checks for common whitespace: 7](#_Toc422903111)

[CheckStyle Modules: NoWhitespaceBefore, MethodParamPad, OperatorWrap, ParenPad, TypecastParenPad, WhitespaceAround 7](#_Toc422903112)

[Checks for common coding problems: 8](#_Toc422903113)

[CheckStyle Module: AvoidInlineConditionals 8](#_Toc422903114)

[CheckStyle Module: EmptyStatement 8](#_Toc422903115)

[CheckStyle Module: EqualsHashCode 8](#_Toc422903116)

[CheckStyle Module: HiddenField 9](#_Toc422903117)

[CheckStyle Module: IllegalInstantiation 9](#_Toc422903118)

[CheckStyle Module: InnerAssignment 10](#_Toc422903119)

[CheckStyle Module: MagicNumber 10](#_Toc422903120)

[CheckStyle Module: MissingSwitchDefault 10](#_Toc422903121)

[CheckStyle Module: SimplifyBooleanExpression 11](#_Toc422903122)

[CheckStyle Module: SimplifyBooleanReturn 11](#_Toc422903123)

[Checks for class design: 11](#_Toc422903124)

[CheckStyle Module: FinalClass 11](#_Toc422903125)

[CheckStyle Module: HideUtilityClassConstructor 11](#_Toc422903126)

[CheckStyle Module: InterfaceIsType 12](#_Toc422903127)

[CheckStyle Module: VisibilityModifier 12](#_Toc422903128)

[CheckStyle Module: DesignForExtension 12](#_Toc422903129)

[Checks for naming conventions: 13](#_Toc422903130)

[CheckStyle Module: ConstantName 13](#_Toc422903131)

[CheckStyle Modules: MemberName, LocalVariableName, LocalFinalVariableName, MethodName, ParameterName, StaticVariableName, TypeName 13](#_Toc422903132)

[Checks for Javadoc comments: 13](#_Toc422903133)

[CheckStyle Module: JavadocMethod 14](#_Toc422903134)

[CheckStyle Module: JavadocType 14](#_Toc422903135)

[CheckStyle Module: JavadocVariable 14](#_Toc422903136)

[CheckStyle Module: JavadocStyle 14](#_Toc422903137)

[Checks for modifiers: 15](#_Toc422903138)

[CheckStyle Module: ModifierOrder 15](#_Toc422903139)

[CheckStyle Module: RedundantModifier 15](#_Toc422903140)

[Miscellaneous other checks: 15](#_Toc422903141)

[CheckStyle Module: ArrayTypeStyle 16](#_Toc422903142)

[CheckStyle Module: TodoComment 16](#_Toc422903143)

[CheckStyle Module: FinalParameters 16](#_Toc422903144)

[CheckStyle Module: UpperEll 16](#_Toc422903145)

[Proposed additional rules: 17](#_Toc422903146)

[CheckStyle Module: MultipleVariableDeclarations 17](#_Toc422903147)

[CheckStyle Module: OneStatementPerLine 17](#_Toc422903148)

[What should establish critical violations? 17](#_Toc422903149)

[Maven CheckStyle plugin properties: 17](#_Toc422903150)

[Additional Resources: 18](#_Toc422903151)

# Overview:

CheckStyle is a static code analysis tool used in software development for checking if Java source code complies with coding rules. In this document default rulesets based on Sun Code Conventions are elaborated. This document also contains suggestions as to how to tweak some of the rules to be more in line with ISC code style practices.

# Document Organization:

This document discusses the file level checks and is followed by all the default Sun convention modules. Some extra modules are proposed and at the end of this document a few rules for which non-compliance should trigger build failure have been suggested. Throughout this document notes and suggestions are mostly highlighted with the RED color.

# File Level Checks:

## CheckStyle Module: FileTabCharacter

***Check:*** Checks that there are no tab characters ('\t') in the source code.

***Reason***: Use of tabs is not recommended by Apache.

**Magnitude:** Large

## CheckStyle Module: FileLength

***Check:*** Checks for long source files (longer than 2000 lines).

***Reason***: If a source file becomes very long it is hard to understand. Therefore long classes should usually be refactored into several individual classes that focus on a specific task.

***ISC Code Status***: Adheres to this rule.

## CheckStyle Module: JavadocPackage

***Check:*** Checks that each Java package has a Javadoc file used for commenting. By default it only allows a package-info.java file, but can be configured to allow a package.html file.

***ISC Code Status***: Does not follow this rule.

**Magnitude:** Large

**Is this needed?**

# Checks for size violations:

## CheckStyle Module: MethodLength

***Check***: Checks for long methods and constructors

***Reason***: If a source file becomes very long it is hard to understand. Therefore long classes should usually be refactored into several individual classes that focus on a specific task.

***ISC Code Status***: Does not adhere to this rule.

**Magnitude:** Medium

**Note: The default maximum allowable number of lines per method is 150 lines.**

## CheckStyle Module: ParameterNumber

***Check***: Checks the number of parameters of a method or constructor

***Reason***: It helps readability and maintenance of the code.

***ISC Code Status***: Does not adhere to this rule.

**Magnitude:** Small

**Note: The default maximum allowable number of parameter is 7. However we can increase this to 10 to satisfy our current ISC code. (There is really no specific reason mentioned in the documentation as to why max 7 is the default value.)**

## CheckStyle Module: LineLength

***Check***: Checks for long lines.

***Reason***: Long lines are hard to read in printouts or if developers have limited screen space for the source code, e.g. if the IDE displays additional information like project tree, class hierarchy, etc.

**Note: The default maximum allowable line length is 80. However we can increase this to 120 to accommodate our preference in ISC code.**

**Note2: Is this check going to be useful? In my opinion Eclipse Code Style is more flexible and has more granularities associated with it.**

# Checks for imports:

## CheckStyle Module: AvoidStarImport

***Check***: Checks that there are no import statements that use the \* notation.

***Reason***: Importing all classes from a package or static members from a class leads to tight coupling between packages or classes and might lead to problems when a new version of a library introduces name clashes.

***ISC Code Status***: Adheres to this rule.

## CheckStyle Module: IllegalImport

***Check***: Checks for imports from a set of illegal packages.

***Reason***: By default, the check rejects all sun.\* packages since programs that contain direct calls to the sun.\* packages are not guaranteed to work on all Java-compatible platforms

***ISC Code Status***: Adheres to this rule.

## CheckStyle Module: RedundantImport

***Check***: Checks for redundant import statements.

***Reason***: Obviously we do not want to this duplicate import statements in our code base.

***ISC Current Code Status***: Does not adhere to this rule. (It should but if a developer forgot to apply ctrl+shft+o, then this could potentially happen)

**Magnitude:** Large

## CheckStyle Module: UnusedImports

***Check:*** Checks for unused import statements.

***Reason:*** Avoid unnecessary lines of code.

***ISC Current Code Status:*** Does not adhere to this rule. (It should but if a developer forgot to apply ctrl+shft+o, then this could potentially happen)

**Magnitude:** Large

# Checks for blocks:

## CheckStyle Module: AvoidNestedBlocks

***Check***: Finds nested blocks, i.e. blocks that are used freely in the code.

***Reason***: Nested blocks are often leftovers from the debugging process, they confuse the reader. For instance the following code snippet will be flagged by CheckStyle:

// if (conditionThatIsNotUsedAnyLonger)

{

System.out.println("unconditional");

}

***ISC Current Code Status***: Adheres to this rule.

## CheckStyle Module: EmptyBlocks

***Check***: Checks for empty blocks.

***Reason***: It is generally not recommended to have empty blocks. However some exceptions may apply.

***ISC Current Code Status***: Adheres to this rule.

**Magnitude:** Large

**Proposal: To cover special cases I propose to tweak this rule so that it does not report blocks where we intentionally left empty with some TODO comments. Developers should ensure that they always leave some comments stating the reason as to why an empty block is needed (such as a TODO or any other comment).**

## CheckStyle Module: LeftCurly

***Check***: Checks for the placement of left curly braces ('{') for code blocks.

***Reason***: Enhances code readability

***ISC Current Code Status***: Adheres to this rule.

## CheckStyle Module: RightCurly

***Check***: Checks for the placement of right curly braces ('{') that it starts a line.

***Reason***: Enhances code readability

***ISC Current Code Status***: Does not adhere to this rule (Not too many violations!)

**Magnitude:** Small

## CheckStyle Module: NeedBraces

***Check***: Checks for braces around code blocks.

***Reason***: Enhances code readability

***ISC Current Code Status***: Adheres to this rule.

# Checks for common whitespace:

## CheckStyle Modules: NoWhitespaceBefore, MethodParamPad, OperatorWrap, ParenPad, TypecastParenPad, WhitespaceAround

***Check***: Checks against various whitespace policies.

***Reason***: Enhances code readability

***ISC Current Code Status***: Does not adhere to this rule completely

**Magnitude:** Medium

**Note: The above-mentioned modules are in line with ISC code style, hence I would recommend them to be enforced by CheckStyle as well.**

# Checks for common coding problems:

## CheckStyle Module: AvoidInlineConditionals

***Check***: Detects inline conditionals.

***Reason***: Some developers find inline conditionals hard to read.

***ISC Current Code Status***: Does not adhere to this rule.

**Magnitude:** Large

**Proposal: My research shows that many developers in fact continue this practice and do not believe it is really confusing. Particularly I found the following complaint from one of the checkstyle developers very interesting when he decides to skip this check.**

[**https://github.com/checkstyle/checkstyle/issues/1037**](https://github.com/checkstyle/checkstyle/issues/1037)

**So I would propose to disable this check for our ISC code.**

## CheckStyle Module: EmptyStatement

***Check***: Detects empty statements (standalone ";" semicolon).

***Reason***: Enhances code readability

***ISC Current Code Status***: Does not adhere to this rule completely

**Magnitude:** Small

## CheckStyle Module: EqualsHashCode

***Check***: Checks that classes that override equals() also override hashCode().

***Reason***: Equal objects need to have the same hash code, i.e. whenever a.equals(b), then a.hashCode() must be same as b.hashCode(). So in practice whenever equals() method in a class is overridden, hashcode() should be overridden likewise. Check the following link for more insight:

<http://stackoverflow.com/questions/2265503/why-do-i-need-to-override-the-equals-and-hashcode-methods-in-java>

***ISC Current Code Status***: Does not adhere to this rule.

**Magnitude:** Small

## CheckStyle Module: HiddenField

***Check***: Checks that a local variable or a parameter does not shadow a field that is defined in the same class.

***Reason***: Avoid shadowing which is believed to cause less readability and more confusion.

***ISC Current Code Status***: Does not adhere to this rule.

**Magnitude:** Large

**Proposal: Since my investigation shows that a lot of developers actually do not find shadowing to be very confusing and turn off this check and I also believe this check does not go well with our current coding style and is not really cause of lots of confusion, I propose this check to be eliminated from our CheckStyle configuration. I found the following thread useful:**

<http://stackoverflow.com/questions/7776046/checkstyle-how-to-resolve-hidden-field-error>

## CheckStyle Module: IllegalInstantiation

***Check***: Checks for illegal instantiations where a factory method is preferred.

***Reason***: Depending on the project, for some classes it might be preferable to create instances through factory methods rather than calling the constructor. A simple example is the java.lang. Boolean class, to save memory and CPU cycles it is preferable to use the predefined constants TRUE and FALSE. Constructor invocations should be replaced by calls to Boolean.valueOf().

***ISC Current Code Status***: Adheres to this rule.

## CheckStyle Module: InnerAssignment

***Check***: Checks for assignments in subexpressions, such as in String s = Integer.toString(i = 2);

***Reason***: With inner assignments like the one given above, it is difficult to see all places where a variable is set.

***ISC Current Code Status***: Adheres to this rule.

## CheckStyle Module: MagicNumber

***Check***: Checks that there are no ["magic numbers"](http://en.wikipedia.org/wiki/Magic_number_%28programming%29) where a magic number is a numeric literal that is not defined as a constant. By default, -1, 0, 1, and 2 are not considered to be magic numbers.

***Reason***: Generally it is advised to define constant (final) variables for numeric literals.

***ISC Current Code Status***: Does not adhere to this rule.

**Magnitude:** Large

**Note: Although use of constants is preferable, our ISC code in many places has used numeric literals. In defining a simple binary array (such as new byte[8]) to a lot of places in our UI and so forth we can notice these literals.. Therefore there needs to be significant amount time spent to change this pattern which may not be even practical and worth the effort.**

## CheckStyle Module: MissingSwitchDefault

***Check***: Checks that switch statement has a "default" clause.

***Reason***: It's usually a good idea to introduce a default case in every switch statement. Even if the developer is sure that all currently possible cases are covered, this should be expressed in the default branch, e.g. by using an assertion. This way the code is protected against later changes, e.g. introduction of new types in an enumeration type.

***ISC Current Code Status***: Adheres to this rule.

## CheckStyle Module: SimplifyBooleanExpression

***Check***: Checks for over-complicated boolean expressions. Currently flags code like if (b == true).

***Reason***: Complex boolean logic makes code hard to understand and maintain.

***ISC Current Code Status***: Adheres to this rule.

## CheckStyle Module: SimplifyBooleanReturn

***Check***: Checks for over-complicated boolean return statements.

***Reason***: Complex boolean logic makes code hard to understand and maintain.

***ISC Current Code Status***: Adheres to this rule.

# Checks for class design:

## CheckStyle Module: FinalClass

***Check***: Checks that a class which has only private constructors is declared as final.

***Reason***: A class with only private constructors can’t be subclassed. Thus declaring final will have value in reminding code readers this fact. There could also be some performance gains.

***ISC Current Code Status***: Does not adhere to this rule.

**Magnitude:** Medium

## CheckStyle Module: HideUtilityClassConstructor

***Check***: Makes sure that utility classes (classes that contain only static methods or fields in their API) do not have a public or default constructor.

***Reason***: Instantiating utility classes does not make sense. Hence the constructors should either be private or protected (if subclassing is to be allowed). Following thread might be useful in understanding this requirement better:

<http://stackoverflow.com/questions/14398747/hide-utility-class-constructor-utility-classes-should-not-have-a-public-or-def>

***ISC Current Code Status***: Does not adhere to this rule.

**Magnitude:** Medium

**Note: This is currently one of our main class design flaws in our ISC code base in my opinion.**

## CheckStyle Module: InterfaceIsType

***Check***: Checks that Interfaces are used only to define types.

***Reason***: An interface should describe a *type*. It is therefore inappropriate to define an interface that does not contain any methods but only constants.

***ISC Current Code Status***: Adheres to this rule.

## CheckStyle Module: VisibilityModifier

***Check***: Checks visibility of class members. Only static final, immutable or annotated by specified annotation members may be public; other class members must be private.

***Reason***: Enforce encapsulation.

***ISC Current Code Status***: Does not adhere to this rule.

**Magnitude:** Large

**Proposal: Looking at ISC code, I think it makes sense to tweak this rule so that protected members are also allowed. But the bigger problem is that a lot of POJOs in our ISC code have public members which would be flagged by CheckStyle. So definitely a great effort needs to be invested to fix this and add the necessary getters/setters.**

## CheckStyle Module: DesignForExtension

***Check***: This rule checks that nonprivate, nonstatic methods of classes that can be subclassed must be abstract or be final or have an empty implementation

***Reason***: The rationale here is to protect superclasses against being broken by subclasses.

***ISC Current Code Status***: Does not adhere to this rule.

**Magnitude:** Large

**Drawback and proposal: It is acknowledged in the CheckStyle documentation that this rule will limit flexibility of subclasses. They cannot prevent execution of code in the superclasses. ISC code also is very far from adhering to this rule. Therefore I do not see this rule to be necessary at this point.**

# Checks for naming conventions:

## CheckStyle Module: ConstantName

***Check***: All constants (static, final fields) names should comply with the following regex: ^[A-Z][A-Z0-9]\*(\_[A-Z0-9]+)\*$(In plain English: should be all uppercase with words separated by underscores ("\_").

***Reason***: Oracle code conventions: <http://www.oracle.com/technetwork/java/codeconventions-135099.html>

***ISC Current Code Status***: Does not adhere to this rule.

**Magnitude:** Large

## CheckStyle Modules: MemberName, LocalVariableName, LocalFinalVariableName, MethodName, ParameterName, StaticVariableName, TypeName

***Check***: The above modules check for naming convention based on camelCase notation.

***Reason***: Comply with camelCase notation.

***ISC Current Code Status***: Does not adhere to this rule.

**Magnitude:** Large

# Checks for Javadoc comments:

## CheckStyle Module: JavadocMethod

***Check***: Checks the Javadoc of a method or constructor

***Reason***: Enhance code documentation

***ISC Current Code Status***: Does not adhere to this rule.

**Magnitude:** Extra Large

**Note: Currently ISC code base lacks this to a great extent and in fact may require much effort to completely adhere to this rule. Also it is important to remember that auto generated Javadocs usually do not add much value to the code documentation.**

## CheckStyle Module: JavadocType

***Check***: Checks Javadoc comments for class and interface definitions.

***Reason***: Enhance code documentation.

***ISC Current Code Status***: Does not adhere to this rule.

**Magnitude:** Large

## CheckStyle Module: JavadocVariable

***Check***: Checks that variables have Javadoc comments.

***Reason***: Enhance code documentation.

***ISC Current Code Status***: Does not adhere to this rule.

**Magnitude:** Extra Large

**Proposal: This check will make the code base very verbose since Javadocs will become required for all the class variables. My research shows that many developers also do not like this rule. I think it may not be a bad idea to eliminate this rule at this point. Any thoughts on this are more than welcome!**

## CheckStyle Module: JavadocStyle

***Check***: Validates Javadoc comments to help ensure they are well formed (such as proper punctuation, etc.)

***Reason***: Proper code documentation.

***ISC Current Code Status***: Does not adhere to this rule.

**Magnitude:** Medium

# Checks for modifiers:

## CheckStyle Module: ModifierOrder

***Check***: Checks that the order of modifiers conforms to the suggestions in the Java Language Specification (JLS)

***Reason***: Conform to the JLS

***ISC Current Code Status***: Does not adhere to this rule.

**Magnitude:** Medium

## CheckStyle Module: RedundantModifier

***Check***: Checks for redundant modifiers in interface and annotation definitions, the final modifier on methods of final classes, and inner interface declarations that are declared as static.

***Reason***: The JLS strongly discourages the usage of *public* and *abstract* for method declarations in interface definitions as a matter of style. Variables in interfaces and annotations are automatically public, static and final, so these modifiers are redundant as well.

***ISC Current Code Status***: Does not adhere to this rule.

**Magnitude:** Small

# Miscellaneous other checks:

## CheckStyle Module: ArrayTypeStyle

***Check***: Checks the style of array type definitions. Some like Java style: public static void main(String[] args) and some like C style: public static void main(String args[])

***Reason***: Enforce consistency across the code base

***ISC Current Code Status***: Adheres mostly to the Java style.

**Magnitude:** Small

**Proposal: It makes much sense to enforce the Java style. Hence the policy for array declarations should be Java style.**

## CheckStyle Module: TodoComment

***Check***: Reports TODO: comments.

***Reason***: Having them reported by CheckStyle makes it very hard to forget about them!

## CheckStyle Module: FinalParameters

***Check:*** Check that parameters for methods, constructors, and catch blocks are final.

***Reason:*** Making a parameter final guarantees that the value used at any location in the method refers to the value passed.

***ISC Current Code Status***: Does not adhere to this rule.

**Magnitude:** Extra Large

**Note: We are currently definitely not practicing this rule. Adhering to this rule means a lot more verbosity. Also lots of effort will be needed to fix this in the current code base. I would propose to remove this rule from the default ruleset at this point.**

## CheckStyle Module: UpperEll

***Check:*** Checks that long constants are defined with an upper ell. That is ' L' and not 'l'. This is in accordance with the JLS.

***ISC Current Code Status***: Adheres to this rule.

# Proposed additional rules:

The following two rules are very popular among developer community even though are not part of Sun default rulesets.

## CheckStyle Module: MultipleVariableDeclarations

***Check***: Checks that each variable declaration is in its own statement and on its own line.

***Reason***:  [the Java code conventions chapter](http://java.sun.com/docs/codeconv/html/CodeConventions.doc5.html#2991)  recommends that declarations should be one per line/statement.

***ISC Current Code Status:*** Does not adhere to this rule

**Magnitude:** Small

## CheckStyle Module: OneStatementPerLine

***Check***: Checks that there is only one statement per line.

***Reason***:  Enhances code readability

***ISC Current Code Status:*** Adheres to this rule

# What should establish critical violations?

1. File length (more than 2000 lines)
2. Method Length (more than 800 lines)
3. InnerAssignment

# Maven CheckStyle plugin properties:

The following options can be in the pom.xml to instruct maven on certain aspects of this integration:

**configLocation**: The location/filename of the customized CheckStyle rulesets which would be the basis of CheclStyle enforcement criteria

**failOnViolation**: This flag decides if the build should fail upon encountering violations. Please note that CheckStyle configuration XML file allows defining different violation levels for different rules.

There are other properties that can be used depending on the integration requirements

# Additional Resources:

<http://checkstyle.sourceforge.net/>

<http://www.oracle.com/technetwork/java/codeconventions-135099.html>