PRINCIPLES OF SOFTWARE DEVELOPMENT GUIDED EXERCISE #2

REPORT ON CHECKSTYLES

UNIVERSIDAD CARLOS III DE MADRID
WONG YUAN NENG 100435501
RICARDO DIAZ ACOSTA 100405923

Rule 1) Declaration order check

This rule checks for the order in which the parts of a class or interface declaration appears. It ensures the following order: Class (static) variables, instance variables, constructors, methods. Also, this rule ensures that the order of visibility for each of the abovementioned category is public, protected, package, then the private level. Having such specific groupings will ensure readability by other developers, allowing them to work on further developments easily as they would know where to find the variables, constructors and methods that they need reference for.

Rule 2) Overload methods declaration order

This rule checks that overload methods are grouped together. Doing so will allow each developer to easily check if a method has been overloaded, as well as compare the code differences. Without this rule, developers have to check if this overloaded method exists elsewhere in the code, and since some developers have the habit to group the overloaded methods, a new developer could assume that there are only 2 overloaded methods when there existed 3.

Rule 3) Require This

This rule checks that the entire code does not rely on the default "this." and requires developers to explicitly reference to instance variables and methods in the form of "this.varName" and "this.methodName(args)". By preventing the omission of "this." in codes, other developers will have no doubts when identifying if a specific variable or method is static or non-static, and the developers will not have to go through the trouble to find out by reading the entire code.

Rule 4) Empty catch block

This rule checks if a catch block is empty. An empty catch block does not properly handle the exception thrown. Applying this rule ensures that all exceptions thrown will have a set of codes in the catch block that will be executed to handle them.

Rule 5) Missing switch default

This rule checks that every switch statement has a "default" clause. A developer may be sure that all currently possible cases have been covered, but future changes may lead to more possible cases which could result in fatal program errors if the default clause is absent. Furthermore, any developer could possibly miss out a case or two, and the developer would also have to handle invalid cases as well.

Rule 6) Missing Javadoc method

This rule checks that every method and constructor have a Javadoc comment attached to explain the purpose of the method and each of its parameters. A public method or constructor that does not have a Javadoc comment makes an API difficult to understand and maintain by other developers.

Rule 7) Nonempty @-clause description

This rule checks that @-clause tags are followed by a description. This prevents empty Javadoc comments, which provides no information about the method and constructor and is just escaping from rule 6.

Rule 8) Constant names

This rule checks that constant variable names follow the naming convention, which is entirely uppercase. Although it is technically a variable, the unique naming convention sets it apart from a standard variable as a constant variable is one that is static final. Its value will not change throughout the program execution, hence it needs to be manually set and altered, and full uppercase is definitely the way to attract the attention of developers.

Rule 9) Type names

This rule checks that class names follow the naming convention, which is UpperCamelCase with no suffix. This is to allow for differentiation between classes and variables. Although not mandatory, it is recommended to use representative nouns as the class name. This will help developers differentiate between Classes and Interfaces, as Interface names should be an adjective and not a noun.

Rule 10) Local Variable names

This rule checks that local variable names follow the naming convention, which is lowerCamelCase without any suffix. This is to allow for differentiation between variables, classes and interfaces. Classes and Interfaces do not have any brackets suffix as well, hence a variable can be mistaken as a Class or Interface if they share similar naming convention.

Rule 11) Method names

This rule checks that method names follow the naming convention, which is also lowerCamelCase but with a suffix of (). This is to allow for differentiation between methods and constructors, which both are suffixed with brackets that accepts parameters. Although not mandatory, it is recommended to use verbs representative of the function as the method name. This will allow other developers to have a quick understanding of what the method is about without having to delve into the method's code.

Rule 12) Indentation

This rule checks the correct indentation of each statement in the code. Indentation is a crucial element in ensuring readability as it properly segregates the blocks of codes so that the program will not look like 1 huge chunk of codes. Without indentations, it will be difficult for a developer to check if a statement is part of a while loop in a massive chunk of nested loops and if-else blocks. Tabs are not accepted as indentations and only each space is counted as 1 level of indentation.

Rule 13) Inner assignment

This rule checks for assignments in a statement of code, for instance "String s = Integer.toString(i=2);". Such inner assignments ruin code readability because it would be extremely difficult to spot an inner assignment within huge chunks of codes. This will potentially lead to fatal errors like miscalculations or out-of-bound errors because other developers, who did not notice the inner assignment, might have used the variable involved to develop new methods.

Rule 14) Need braces

This rule checks for braces around code blocks, which is extremely helpful in grouping huge chunks of codes into multiple but smaller blocks of codes. This increases readability especially for complex projects involving long code implementations.