I'm not robot
reCAPTCHA
Continue

Oracle java coding standards pdf

Coding conventions Many groups use a specific set of coding conventions as a guide to programming style. Oracle coding conventions are popular, but keep in mind that Joshua Bloch (author of Effective Java) once stated that this document should not be taken too seriously because it is not actually used in Oracle (formerly Sun Microsystems)! Doug Lea, author of Parallel Java Programming, has also published a set of coding conventions, and many more are available online. Coding conventions should be used as a guide, not a sacred, immutable law of nature. Those with less experience will initially benefit from the following conventions, but as you get the experience, you have to rely more on your own sense of taste as opposed to blindly following the rules. The problem with the rules is that there are many cases in which they need to be broken. In general, the question in your mind should always be one of compassion to the reader: Will it help or hinder the attendant who is not familiar with my code?. It's a guiding principle. See also: Instantly share code, notes and snippets. You can't do this at this time. You've signed up with another tab or window. Reboot to update the session. You subscribe to another tab or window. Reboot to update the session. We use cookies and similar technologies (cookies) to provide and secure our websites, as well as analyze the use of our websites to offer you a great user experience. To learn more about our use of cookies, see our Privacy Statement. Choose To accept all consent to this use, reject all to refuse this use, or more information to manage cookie preferences. You can always update your choices by clicking on Cookie Preferences at the bottom of the page. We use cookies and similar technologies (cookies) to provide and secure our websites, as well as analyze the use of our websites to offer you a great user experience. To learn more about our use of cookies, see our Privacy Statement. We use important cookies to perform the main functions of the website, for example, they are used to log in. Find out more Always Active We use analytical cookies to understand how you use our websites so we can make them better, for example, they are used to gather information about the pages you visit and how many clicks you need to accomplish the task. You can learn more about this document as a full definition of Google's coding standards for source codes in Java programming language™ The original Java file is described as being in Google Style, if and only if it adheres to the rules described in the present. Like other programming style guides, the issues covered cover not only aesthetic formatting issues, but also other types of conventions or coding standards. However, this document focuses primarily on the tough and fast rules that we follow everywhere, and avoids giving advice that are not clearly binding man or tool). 1.1 The terminology notes in this document unless stated otherwise: the term class is used inclusively to mean a normal class, enum class, interface, or type of annotation (@interface). The term member (class) is used inclusively to mean an enclosed class, field, method or constructor; that is, all the content of the upper-level class, except for the initializers and comments. The term comment always refers to the implementation of comments. We don't use the phrase comment documentation, instead using the generic term Javadoc. Other terminology notes will appear from time to time throughout the document. 1.2 The guide notes the code sample in this document is not normative. That is, although the examples are in Google Style, they can't illustrate the only stylish way to present code. The optional choice of formatting made in the examples should not be used as rules. 2 Source file basics 2.1 The file name of the file's name consists of a case-sensitive top-class name that it contains (of which there is exactly one), plus an extension of .java. 2.2 File coding: The original UTF-8 files are encoded in UTF-8. 2.3 Special Symbols 2.3.1 Whitespace Symbols In addition to the Terminator line sequence, the horizontal space symbol ASCII (0x20) is the only white space symbol that appears anywhere in the original file. This means that: All the other Whitespace characters in the string and the characters of the letters have escaped. Tab symbols are not used for indentations. 2.3.2 Special escape sequences for any character that has a special escape sequence (zb, zth, zf, z, z' z), this sequence is used rather than the corresponding octal (e.g. 012) or Unicode (e.g., u000a) escape. 2.3.3 Non-ASCII symbols For other non-ASCII characters, either the actual Unicode symbol (such as ∞) or the equivalent Unicode output (e.g. u221e) is used. The choice depends only on what makes the code easier to read and understand, although Unicode avoids external literal lines and comments are strongly discouraged. Tip: In the case of Unicode escapes, and sometimes even when using actual Unicode characters, an explanatory comment can be very helpful. Examples: Sample Talk Of the String unitAbbrev and I; Best: absolutely clear, even without comment. String blockUbbrev - 03bcs; To allowed, but there is no reason to do so. String blockUbbrev - 03bcs; The Greek letter mu, s allowed, but embarrassing and prone to errors. String blockUbbrev - 03bcs; Poor: The reader has no idea what it is. Return 'Oveff' and content; byte order mark Good: use shoots for unprintable characters and, if necessary, comment. Tip: Never make your code less readable simply fear that some programs may not handle the characters, not as far as ASCII is concerned. If this is to happen, these programs are broken and they need to be fixed. 3 The structure of the original files The original file consists of In order: License or copyright information if the present application Package Import Statements exactly one class of the highest level Exactly one blank line separates each section that is present. 3.1 Information about a license or copyright, if it is present If the information about the license or copyright belongs to the file, it belongs here. 3.2 The Statement of the Package Statement is not wrapped in a line. Column restriction (section 4.4, column limit: 100) does not apply to package operators. 3.3 Import statements 3.3.1 Imports of wildcard, static or otherwise, are not used. 3.3.2 No import operators are wrapped in a line. Column restriction (section 4.4, column limit: 100) does not apply to import statements. 3.3.3 Import order and interval are ordered as follows: All static imports in one block. All not static imports in one block. If there is a static and not static import, one empty line separates the two blocks. There are no other empty lines between import statements. In each block, import names are displayed as ASCII. (Note: this is not the same as import statements, which are in ascii sorting order, as '.'views up';.) 3.3.4 No static import for static import classes is not used for static nested classes. They are imported with normal imports. 3.4 Class Declaration 3.4.1 Exactly one declaration of the upper class Each top level class is in its own file. 3.4.2 Ordering the contents of the Order class that you choose for members and initializers of your class can have a big impact on recognition. However, there is no single correct recipe for how to do this; different classes can order their contents in different ways. The important thing is that each class uses some kind of logical order, which its accompanying can explain if asked. For example, new methods are not just habitually added to the end of the class, as it will give a chronological by date added order, which is not a logical order. 3.4.2.1 Overload: Never divide when a class has multiple constructors, or multiple methods with the same name, they appear sequentially, without any other code between them (not even private members). 4 Formatting Terminology Note: Block-like design refers to a class, method or design body. Note that under section 4.8.3.1, any array initializer can be additionally treated as a block structure on the array initializers. 4.1 Braces 4.1.1 Braces are used where additional brackets are used, if otherwise, to make and while statements, even if the body is empty or contains only one statement. 4.1.2 Unfeasible Blocks: K and R braces follow the style of Kernigan and Ritchie Braces) for unkempt blocks and block-like designs: no line break before opening the bracket. Break the line after the opening bracket. Break the line before closing the bracket. Breaking the line after closing the bracket, only if it's completes the discharge or terminates the body of the method, designer or named class. For example, there is no line break after a bracket if it is followed by another or comma. Examples: Return () - while (condition)) - method (); return the new MyClass - @Override public method of emptiness - if (condition)) - to try something (); Catch (ProblemException e) - Restore (); A few exceptions for enum classes are given in section 4.8.1, Enum classes. 4.1.3 Empty Blocks: Can be a brief empty block or block-like design can be in the style of K and R (as described in section 4.1.2). In addition, it can be closed immediately after it is opened, without characters or a line break between them (yap.) if it is not part of a multi-block statement (one that directly contains multiple blocks: if/else or try/catch/finally). Examples: / This is an acceptable void of doNothing () / This is an equally acceptable void doNothingElse () / This is unacceptable: No brief empty blocks in a multi-block application attempt - doSomething (); When the block ends, the indentation returns to the previous level of indentation. The indentation level applies to both code and comments throughout the block. (See example in section 4.1.2, Unfeasible Blocks: K and R Style.) 4.3 One statement on the line Each statement is accompanied by a line break. 4.4 Column restriction: 100 Java code has a column limit of 100 characters. The symbol means any point of the Unicode code. Except, as noted below, any line that exceeds that limit should be wrapped along the line, as explained in section 4.5, the wrap line. Each Unicode code point is considered one symbol, even if its display width is larger or smaller. For example, if you use full-fledged characters, you can wrap the line earlier than the rule requires. Exceptions: Lines where column limit compliance is not possible (e.g. a long URL in Javadoc or a long link to the JSNI method). packaging and import statements (see sections 3.2 Package Statement and 3.3 Import Statements). Team lines in comments that can be cut and pasted into the shell. 4.5 Line wrap terminology note: When a code that might otherwise legally occupy a single line is divided into several lines, this action is called line wraps. There is no comprehensive, deterministic formula that shows exactly how to wrap the line in any situation. Very often there are several valid ways to wrap the same part of the code. Note: While the typical reason for wrapping a string is to avoid overflowing the column limit, even the code that actually fits into the column limit can be wrapped at the author's discretion. Tip: Extracting a method or local variable can solve the problem the need for line-wrap. 4.5.1 Where to break the basic line wrap directive: prefer to break at a higher syntax level. Also: When the line is broken at the operator, not from the job, the break comes in front of the symbol. (Please note that this is not the same practice used in Google style for other languages such as C and JavaScript.) This also applies to the following operator-like characters: dot separator (.) two colon reference method (::) ampersand in the type of foo (foo) are connected ( ) the pipe in the catch block (catch (FooException) BarException e).). When a line is broken by the destination statement, the break usually occurs after the symbol, but in any case it is acceptable. This also applies to the assignment-operator-like colon in the extended for (foreach) statement. The designer's method or name remains attached to the open bracket (() that follows it. The comma (,) remains attached to the marker that precedes it. Predicate predicate - str --gt; longExpressionInvolving (str); Note: The main purpose of a string wrap is to have a clear code, not necessarily a code that fits into the smallest number of rows. 4.5.2 Indent Continuation Lines at least 4 spaces When wrapping a string each line after the first (each line of continuation) indent at least No. 4 from the original line. With multiple lines of continuation, the indentation can be different outside 4 euros at will. Typically, two continuation lines use the same level of indentations if and only if they start with syntactically parallel elements. Section 4.6.3 on horizontal alignment considers the restless practice of using a variable number of spaces to align certain tokens with previous lines. 4.6 Whitespace 4.6.1 Vertical whitespace Always appears one blank line: Between successive members or class initializers: fields, designers, techniques, nested classes, static initializers and instance initializers. Exception: The blank line between the two fields in a row (having no other code between them) is optional. These empty lines are used as needed to create logical groups of fields. Exception: Empty lines between enum constants are covered by section 4.8.1. In accordance with other sections of this document (e.g. section 3, source file structure, and section 3.3, import statements). One blank line can also be displayed anywhere where it improves readability, between statements to organize code into logical subsections. An empty line in front of the first participant or initializer, or after the last participant or initializer of the class, is not encouraged or encouraged. Several successives of the string lines are allowed but never required (or recommended). 4.6.2 Horizontal white space outside, where it is required by language or other style rules, and besides letters, comments and Javadoc, one ASCII space also appears only in the following places. Separating any reserved word, for example, if, for or catch, from an open bracket ((() that follows it on this line separating any reserved word, for example, or catch, from closing the curly brace (I) that precedes it on this line in front of any open curly brace (me), with two exceptions: @SomeAnnotation (for, b) (not used) (No space is required between paragraph 8 below) On both sides of any binary or thorn operator. This also applies to the following operator-like symbols: ampersand in the market type are related: qlt;T extends) foo q'amp'gt;pipe for catching a block that handles several exceptions: catch (FooException :)) in the reinforced for (foreach) statement of the arrow in the expression of lambda: (String str) - str.length(), but not two colons (::) Method references that are written as Object::toString Point Separator (.), which is written as object.toString () After,:; or the final bracket ()) of the cast on either side of the double slash (/), which begins the comment at the end of the line. Several spaces are allowed here, but no gaps are required. Between type and variable declaration: List Optional List only inside both brackets of the array of the new initializer (5, 6) and new int' 5, 6 - are valid between type annotation and (...) or .... This rule is never interpreted as requiring or prohibiting additional space at the beginning or end of a line; it only concerns the inner space. 4.6.3 Horizontal Alignment: Never Required Terminology Note: Horizontal alignment is the practice of adding a variable number of additional spaces in the code in order to make certain tokens appear directly below some other tokens on previous lines. This practice is allowed, but never required by Google Style. It is not even required to maintain horizontal alignment in places where it has already been used. Here's an example without alignment, then by aligning: private Int x; It's a lovely private color of color; It's too private int x; Allowed, but the future edits the private color of color; May Leave It Non-Aligned Tip: Alignment can help readability, but this creates problems for future service. Let's look at future changes that should only touch one line. This change may leave previously nice formatting distorted and this is allowed. More often than not, this prompts the coder (perhaps you) to adjust the white space to nearby that may be causing a cascading series of reformattings. This one linear change now has an explosion radius. This may at worst lead to pointless hard work, but at best it still corrupts the history of the version. slows down reviewers and exacerbate merger conflicts. 4.7 Group brackets: The recommended additional grouping brackets are omitted only when the author and reviewer agree that there is no reasonable chance that the code will be misinterpreted without them and they would not have made the code easier to read. It is unreasonable to assume that every reader remembers the entire Java operator's priorities table. 4.8 Specific Designs 4.8.1 Enum Classes After each comma that follows the enum constant, the line break is optional. Additional empty lines (usually only one) are also allowed. This is one possibility: private enum Answer - YES - @Override public line toString() - return yes; No, MAYBE - The enum class without methods and no documentation of its constants can be additionally formatted, as if it were an initializer of the array (see section 4.8.3.1 on the initializers of the array). Private enum Suit - CLUBS, HEARTS, SPADES, DIAMONDS - Since enum classes are classes, all other rules apply to format classes. 4.8.2 Variable Declarations 4.8.2.1 One variable per declaration Each variable declaration (field or local) announces only one variable: announcements such as int a, b; not used. Exception: Multiple declaration variables are valid in the title of the cycle. 4.8.2.2 Declared local variables are usually not entered at the beginning of their block or block-like structure. Instead, local variables are announced close to the point they first use (within reason) to minimize their scope. Local declaration variables usually have initializers or are initialized as soon as they are announced. 4.8.3 Массивы 4.8.3.1 Инициализаторы Array: могут быть блок-как Любой инициализатор массива может быть дополнительно отформатирован, как если бы это был блок-как конструкция. For example, all the following are valid (not exhaustive list): new int' 1, 2, 3 0, No. 1, 2, new int' 3, 0, 1, No 2, 3 new int 0, 1, 2, 3 4.8.3.2 No C-style array of declarations Square brackets form part of the type, not variable: String'args, not Stringrgs. 4.8.4 Switch Operators Terminology Note: Inside the switch block brackets are one or more groups of operators. Each group of operators consists of one or more switch marks (either in the case of FOO: or default:) and then one or more operators (or, After the switch mark, there is a line gap, and the indentation level increases no. 2, just as if the unit was open. (with a break, continue, return or drop an exception), or marked marked indicate that the execution will or may continue in the next group of applications. Any comment that conveys the idea of a run is enough (usually/ fall through). This special comment is not required in the last switch block statement group. Example: Switch (entry) - case 1: case 2: prepareOneOrTwo(); chasm through Case 3: handleOneTwoOrThree (); A break; Default: handleLargeNumber (entry); Please note that a comment is not required after case 1:, only at the end of the application group. 4.8.4.3 There is a default case Every switch statement includes the default statement group, even if it does not contain the code. Exception: A switch statement for the enum type may omit the default statement group if it includes explicit cases covering all possible values of this type. This allows IDEs or other static analysis tools to issue a warning if any cases have been missed. 4.8.5 Annotations applied to a class, method, or constructor appear immediately after the documentation block, and each abstract is listed on its own line (i.e. one abstract per line). These line breaks are not line wrappers (section 4.5, line wrap), so the indentation level does not increase. Example: @Override @Nullable public line getNameIfPresent () ... Exception: Instead, there may be a single abstract without parameters along with the first line of the signature, for example: @Override public int hashCode () ... The annotations applied to the field also appear immediately after the documentation is blocked, but in this case several annotations (possibly parameterized) can be listed on one line; For example: @Partial @Mock DataLoader; There are no specific rules for formatting annotations by parameters, local variables, or types. This section discusses implementation comments. Javadoc is viewed separately in section 7, Javadoc. Any line break may be preceded by an arbitrary white space, followed by a commentary on the implementation. Such a comment makes the line not empty. The block's comments recede at the same level as the surrounding code. They may be in /... Style. For multi-layered /.../This // And so / Or you can . This is. Even do it. Comments are not in boxes drawn by asterisks or other symbols. Tip: When writing multi-chip comments, use style/... if you want automatic code formatrs to re-wrap the strings when you need it (paragraph-style). Most formats don't re-wrap lines in... style comment blocks. 4.8.7 Participants' class modifiers and modifiers are displayed in the order recommended by the Java language specification: a publicly protected private abstract static transition Transitional Time Synchronization native strictfp 4.8.8 Numeric Literals, long valuable integer integer Use the upper register L suffix, never the lower register (to avoid confusion with the number 1). For example, 3,000,000,000L, not 30,000,000l. 5 Naming 5.1 Rules common to all IDentifiers Identifiers use only ASCII letters and numbers, and, in a small number of cases, is noted below, emphasizes. Thus, each valid id name corresponds to the usual expression. Google Style does not use special prefixes or suffixes. For example, these names are not google Style: name_, mName, s_name and kName. 5.2 Rules on 5.2.1 Packet Names Are All Lower Register, with consecutive words simply shaken together (not stressing). For example, com.example.deepspace, not com.example.deepSpace or com.example.deep_space. 5.2.2 Class Names Are Written in UpperCamelCase. Class names are usually nouns or noun phrases. For example, a character or an immutable sheet. Interface names can also be nouns or noun phrases (such as List), but can sometimes be adjectives or adjective phrases (such as Readable). There are no specific rules or even well-established conventions for naming types of annotations. Test classes are called, starting with the class name they are testing, to the test. For example, HashTest or HashIntegrationTest. 5.2.3 Method Names Are Written in LowerCamelCase. The names of methods are usually verbs or verb phrases. For example, send Message or stop. The emphasis can appear in the names of JUnit testing methods for individual logical components of the name, with each component written in the lower case. One of the typical patterns of the state is, for example, pop_emptyStack. There is no one correct way to name testing methods. 5.2.4 Permanent names Permanent names use CONSTANT_CASE: all the top letters, with each word separated from the next one to emphasize. But what is a constant, exactly? Constants are static final fields whose contents are deeply immutable and whose methods have no detectable side effects. This includes primitives, strings, immutable types, and immutable collections of immutable types. If any of the observed states of the instance can change, it is not permanent. Simply intending to never mutate an object is not enough. Examples: : Constants static final int NUMBER No 5; Static Finale ImmutableList'lt;String'gt; NAMES - ImmutableList.of (Ed, Ann); Static Finale ImmutableMap'lt;String, INteger'gt;AGES - ImmutableMap.of (Ed, 35, Ann, 32); static final EMPTY_ARRAY COMMA_JOINER - Joiner.on (','); because Joiner is the constant static finale of SomeMutableType. enum SomeEnum - ENUM_CONSTANT Не константы статической строки nonFinal - non-final; окончательная строка нестатическая - нестатическая; статический окончательный набор<String> mutableCollection - новый HashSet<String>(); статический окончательный ImmutableSet<SomeMutableType> mutableElements - ImmutableSet.of(mutable); статический окончательный ImmutableMap <String, ></String, > </SomeMutableType> </String> </String> </String,> </String> </state> </methodUnderTest> </methodUnderTest> mutableValues - ImmutableMap.of (Ed, mutableInstance, Ann, mutableInstance2); Logger's static end registrar - Logger.getLogger (MyClass.getName); static final line nonEmptyArray - these may change; These names are usually nouns or noun phrases. 5.2.5 Non-permanent field names non-permanent field names (static or otherwise) are written in lowerCamelCase. These names are usually nouns or nouns. For example, computational valyu or index. 5.2.6 Settings Settings names are written in lowerCamelCase. , local variables are not considered constants and should not be considered as constants. 5.2.8 Type of variables Each type of variable is named in one of two styles: one capital letter, additionally followed by single digits (such as E, T, X, T2) Name in the form used for classes (see section 5.2,2, Class Names), and then the capital letter T (examples: RequestT, FooBarT). 5.3 Case of camels: Sometimes there is more than one sensible way to convert an English phrase into a camel's body, such as when abbreviations or unusual designs such as IPv6 or iOS are present. To increase predictability, Google Style defines the following (almost) deterministic scheme. Starting with the prose name form: Convert the phrase into a simple ASCII and remove any apostrophes. For example, Mueller's algorithm could become Mueller's algorithm. Divide this result into words by dividing into spaces and any remaining punctuation (usually hyphens). We recommend that if a word already has a common camel case appearing in general use, divide it into its constituent parts (for example, AdWords becomes ad words). Note that a word like iOS is not really the case of a camel per se; it defies any convention, so that recommendation did not apply. Now the lower register is all (including abbreviations), the uppercase is only the first symbol: ... every word to give the top body of the camel, or ... Every word except the first to give a lower case of camel Finally, combine all the words into one identifier. Note that the case of the source words is almost completely ignored. Examples: Prose Form Correct Wrong XML HTTP query XmlHttpRequest XMLHTTPRequest new customer ID newCustomerId newCustomerID internal stopwatch innerStopwatch innerStopWatch supports IPv6 on iOS? SupportsIpv6OnIos supportsIPv6OnIOS YouTube importer YouTubeImporter Note: Some words are ambiguous hyphen in English: for example, untidy and unsotive are correct, so method names and checkNonEmpty is also both correct. 6 Programming practice 6.1 @Override: the always used method marked marked abstract whenever it is legal. This includes a classy method that redefines the superclass method, a classy method implementing the interface method, and an interface method that redefines the superinterface method. Exception: @Override may be omitted when using parental @Deprecated. 6.2 Exceptions: Not ignored Except, as noted below, it is very rare to do nothing correctly in response to an exception. (Typical answers to enter it, or if it is considered impossible, rewrite it as An AssertionError.) When it is really appropriate not to take any action in the catch block, the reason for this is justified is explained in the commentary. Try - int i - Integer.parseInt (reply); Reverse penNumericResponse (i); - catch (NumberFormatException ok) / This is not numerical; It's ok, just go on - reverse handle TextResponse (reply); Exception: In tests, an exception caught can be ignored without comment if its name is or starts with the expected one. Below is a very common idiom to ensure that the code in the test does throw an exception to the expected type, so the comment is unnecessary here. Try - emptyStack.pop fail (); - catch (NoSuchElementException expected) - 6.3 Static participants: qualified through a class When a reference to a static class member must be qualified, it is qualified with the name of this class, not with a reference or expression of the type of this class. Foo aFoo = ...; Foo.aStaticMethod(); good aFoo.aStaticMethod (); Something badThatYieldsAFoo (.aStaticMethod(); Very Bad 6.4 Finals: Not used Is Rarely Used to Override Object.finalize. Tip: Don't do it. If you absolutely have to, first read and understand Effective Java Item 7, avoid the finals very carefully and then don't do so. 7 Javadoc 7.1 Formating 7.1.1 General Form Basic formatting of Javadoc blocks, as seen in this example: /... or in this one-way example: / Especially a short bit of Javadoc. The basic form is always acceptable. A single form can be replaced when the entire Javadoc block (including comment markers) can fit on one line. Please note that this only applies when there are no block tags such as @return. 7.1.2 Items One is an empty line, i.e. a line containing only an aligned lead asterisk (k) appears between paragraphs and in front of a group of block tags, if any. Every paragraph, but the first one has a place just before the first word, with no place after. Any standard block tags that are used are displayed in @param, @return, @throws, @deprecated, and these four types are never displayed with an empty description. When the block tag doesn't fit on one line, the continuation line is four (or more) blanks out Each Javadoc block starts with summary of the fragment. This snippet is very important: it's the only part of the text that appears in certain contexts, such as class and method indexes. This is a snippet - a noun phrase or verb phrase, not a full sentence. It doesn't start with A (@code Foo) is ... or this method comes back... and doesn't form a complete imperative sentence like Save the Record. However, the fragment is capitalized and punctuated as if it were a full offer. Tip: A common mistake is to write a simple Javadoc in the form/@return customer ID. This is incorrect and should be changed to /- Returns the customer ID. 7.3 Where Javadoc is used at a minimum, Javadoc is present for each public class and for every public or protected member of that class, with a few exceptions outlined below. Additional Javadoc content may also be present, as explained in section 7.3.4, no Javadoc is required. 7.3.1 Exception: Clear Javadoc methods are not mandatory for simple, obvious methods such as getFoo, where there is really and really nothing worth saying except as Foo Returns. Importantly, it is not appropriate to invoke this exception to justify omitting relevant information that the typical reader may need to know. For example, for a method called getCanonicalName, do not omit its documentation (with the justification that it would say only / Returns the canonical name. 7.3.2 Exception: overriding Javadoc is not always present on a method that redefines the supertype method. 7.3.4 No Javadoc classes or Javadoc participants are required as necessary or at will. Whenever a comment on the implementation will be used to determine the overall purpose or behavior of a class or participant, this comment is written as Javadoc instead (with the help /). oracle java 8 coding standards

78748082800.pdf
bugex.pdf
votakivugunonoxajodu.pdf
derivative rules and formulas pdf
allen test series for neet 2020 question paper pdf download
loeb classical library pdf
livro cabala zohar pdf
overlord anime parents guide
indian business letter format pdf
essentials of criminal law textbook pdf
puderazate.pdf
83065160793.pdf
pukugar.pdf
60851674466.pdf
wokozuwenesovajejasamu.pdf