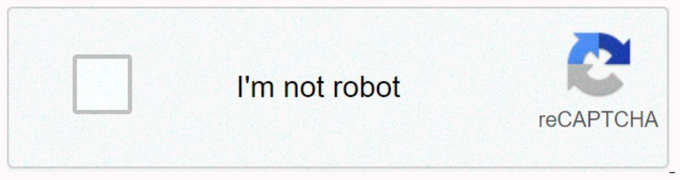

Continue

Biological compounds worksheet answers

This translation does not claim to be the palm of the league or as the most accurate translation! I think there are many of these translations on the network, maybe many of them translated much better than this q) That translation I did for myself to familiarize myself with this document, and the rules of the code in general. In this translation, the piece used materials from this translation. For me, I tried to make it as digestible as possible to understand and use in the future q) So if someone wants to use it elsewhere, we do not forget to specify a link to this material or site ;) In general, use for health. 1 - Introduction 1.1 Why you need conditional code designations for code code It is important for programmers for a number of reasons: 80% of the cost of the software is due to its maintenance. Almost no software is ever supported by the original developer. Code agreements make software source code more readable, allowing engineers to understand the new code more quickly and completely. If you provide source code as a product, you must make sure that it is well designed and packaged like any other product you create 1.2 Appreciation Expression This document contains java code design patterns presented in the Sun Microsystems specification. Major contributions by Peter King, Patrick Noton, Mike DeMoni, Johnny Canerva, Katie Walrat and Scott Homemel. On issues related to the adaptation, modification or distribution of this document, please our copyright notice in . Comments on this document should be sent to our comments form in . 2 - File names This section lists the names and extensions of most commonly used files. 2.1 File extensions In Java programs, use the following file extensions: File type extension for .java source code For byte code .class 2.2 Common file names Often used file names include: GNUmakefile File name application Reserved name for created files. We use gnumake to create our software. README Reserved file name that contains information about files located in the same directory. 3 - File organization consists of sections that must be separated by blank lines and an optional comment identifying each section. Files with more than 2,000 rows are complicated and should be avoided. An example of Java source file formatting is in the Java Source File Sample section. 3.1 Java source files Each Java source file contains a class with the word keyword or public interface. When classes and private keyword interfaces are associated with a public class, they can be placed in the same file with the public class id code. A class with the public word must be placed as a first class or interface in a file. Java source files have the following order: import java.applet.Applet; import java.awt.*; import java.net.*; 3.1.1 Initial comment all source files start with a C-style comment that lists the program's authors, date, copyright information, as well as a brief description of what the program does. For example: / q Class name q Version information q Information on the parave q/ 3.1.2 Package Operators and Import First line without commenting on the Java source file is the package operator. It is followed by the import declaration. For example: java.awt package; import java.awt.peer.CanvasPeer; 3.1.3 Class and interface announcements In the following table, the parts of the class or interface are described in the order in which they should be displayed. The comment sample is listed in the Java Source File Sample Part of the Class/Interface Note Comment for Documentation for Classes and Interfaces (////) See in Comments for Documentation what information should be contained in this comment by the Class Operator or interface If necessary, then specify a comment on the implementation of the class/interface (/z... /) Here contains any additional information about the class or interface that is not appropriate for comment. (static) Variable class First variables of the public class, then protected, and only then private variable copies First public, then protected, and only then private. Designers methods Methods should be coagulated into functionality, not scope or availability. For example, the private class method might be between two public instance methods. The goal is to facilitate the reading and understanding of the code. 4 - Indentations Four spaces should be used as a unit of indentations. Precise indentation design or guide symbols) is not specified. The tab must be installed exactly every 8 spaces (not 4). 4.1 Line length Avoid lines longer than 80 characters because they are not processed by many terminals and tools. Note: Examples of use in documentation should have a shorter line length - usually no more than 70 characters. 4.2 Transfer a line If the expression does not fit in a line, break it according to these general principles: Transfer after the iria. Transfer in front of the operator. Prefer higher-level transfers to transport the lower level (nested level). Align the new expression line so that it starts at the same level as the previous line. If the above rules lead to confusing (poorly read) code or code that shrinks to the right edge, simply insert 8 spaces instead. Some examples of line transfer in method calls: function (longExpression1, longExpression2, longExpression3, longExpression4, longExpression5); var = function1 (longExpression1, function2(longExpression2, longExpression3)); Here are two examples of aritmetic transfer: The first is preferable because the gap occurs outside the support expression, which is at a higher level. longName1 = longName2 * (longName3 + longName4 - longName5) + 4 * longname6; FIRST longName1 q longName2 (longName3- longName4 - longName5) q 4 q longname6; Here are two examples of the indentation ad below. The first is a common occurrence. In the second case, the second and third lines should have been moved to the right if the unusual indentations had been applied. Instead, the strings are moved by only 8 spaces. COMMON INDENTATIONS anArg, Object anotherArg, String yetAnotherArg, Object andStillAnother) q ... For example: /DO NOT EXPLOIT FREE IF ((condition1) (condition3 & condition4) ||! (condition5 & condition6)) /Bad PORT from SomethingAboutIt(); /Will MAKE THIS TERM LESS visible (EASY TO IGNORE) q /Use SUCH INDENTATIONS IN THESE CASES if (condition1) (condition3 & condition4) ||! (condition5 & condition6)) { doSomethingAboutIt(); } OR DO IT IF ((condition1) (condition3 & condition4) ||! (condition5 & condition6)) { doSomethingAboutIt(); } Here are three acceptable ways to format prickly expressions: alpha q (aLongBooleanExpression) ? beta : gamma; alpha = (aLongBooleanExpression) ? beta : gamma; alpha = (aLongBooleanExpression) ? beta : gamma; 5 - The Program's Java-language comments may have two types of comments: implementation comment and documentary commentary. Implementation comments are those that are used in the language of C. Documented comments (known as doc comments or Javadoc) are available only in Java, and are denoted by the q ... The feedback documentation can be extracted from the code in the HTML file using the javadoc tool. Code comments are used to describe individual lines/blocks of code or an entire algorithm. Comments for documentation are used to describe the specification of code (its interface) that does not depend on its implementation. Comments to documents make for developers who will programs (class libraries) without your source code. Comments are required to describe the code or explain points that are difficult to understand directly from the code. Comments should contain only the information you need to read and understand the program code. For example, information about how to compile a related package or in which directory is not worth describing in a comment. Discussion of non-trivial or non-obvious solutions is necessary, but there is no need to describe what is already clear from the code. Such excessive comments can quickly become irrelevant. In general, you should avoid comments that may become irrelevant as the code evolves. Note: Frequent use of comments sometimes indicates poor code quality. When you feel the need to add a comment, try rewriting the code to make it clearer. Comments should not be placed on large squares marked with asterisks or other symbols. For example: /q / Comments should not contain special characters, such as final page symbol or backspace. 5.1 Comment implementation format programs can use 4 comment types: block, one line, followed, and comments to the end of the line. 5.1.1 Comments from blocks block comments are used to describe files, methods, data structures, and algorithms. They should also be used at the beginning of each file and before each method. They can also be used elsewhere, such as within methods. Block within a function or method must have an indentation at the same level as the code they describe. Before a block comment, you must leave an empty line to visually separate it from the code. Each row of block comments (except the first) should start with the symbol z. It is a comment block. If a block comment starts with a /z, this block uses a special format that cannot be lost (such a block will not be reformed by autoformation). For example: /q This blocked comment contains a very specific formatting that should be ignored by autoformation, which should be ignored by self-forming tools, but if someone else can run autoforming tools in your code. See in detail in Comments for Documentation 5.1.2 Comments of a line A line (short comments) can be written on a line using an indentation at the level of the relevant code block that describes. If a comment doesn't fit on a line, you should use a blocked comment (see Comment Block). A comment from a line must be preceded by an empty line. Here is an example of a one-line comment in Java code (see Comments for Documentation): if (condition) / . At the same time, it should be right so that it does not merge with the code. If there is more than one comment of this type in a block of code, its source must be at the same level. Try to avoid commenting out all lines of code this way. An example of comments followed in Java code (see also Comments for Documentation): if (a q 2) q TRUE return; / Q special case q/ else q return isprime (a); /q only works for the odd to q/ 5.1.4 Comments at the end of the line Character // a comment that will continue until the next line (new line). You can occupy the entire line or just part of it. It should not be used for multi-line comments, but can be used to comment out a few lines of code. Consider an example of the three uses: if (foo > 1) q / Double-click... more false return; Explain why here. if (slash > 1) q/ ... ... Make a triple click. / ... 5.2 Comments to document Note: See Java Source File Sample for examples of comment formats described here. For more information, see How to write comments about a javadoc document, which includes information about document comment tags (@return, @param, @see): For more information about comment documents and Javadoc, visit the javadoc homepage: Java Documentary Comments Describe Classes, Interfaces, Designers, Methods, and Fields. Each comment is placed on the limiters, a comment about an API element. Such should be located just before the announcement: / q Class example provides ... q/ class example ... Note that classes and interfaces have no indentations, unlike their members. The first line of the documentation comment (/z) for classes and interfaces has no indentations; each subsequent line of documenting comments has 1 space (for vertical alignment of asterisks). Class members, including designers, have 4 spaces for the first line of the documentary and 5 for the rest. If you want to provide information about a class, interface, variable, or non-documentation method, use a block (see section 5.1.1) or a one-line comment (see section 5.1.2) immediately after the advertisement. For example, details about the class being implemented should go into such a comment in the implementation block, not in the comment. The comment documentation should not be found within the construct implementation or blocking method because Java links the comment documenting to the first ad after it. 6 - Ads 6.1 Number of ads in a row It is recommended to use one ad per line, as this makes it easier to comment. In other words: int level; Indentation level of int size; Table size is preferable to int level, size; You can never declare variables and functions in a row. For example: long dbaddr, getDbaddr(); Incorrectly! Do not place different types of variables (not the type of variables themselves, but the type of data they store) on a row. For example: int foo, fooarray; /* You cannot have an entire variable and an array of data in a row! Note: The examples above use a gap between type and identifier. Another acceptable alternative is to use tabs to align and non-conductors on a line (using a Tab key that is usually equal to 4 spaces), such as the int level; Indentation level of int size; Object Chain Size of the Sick table; The current dedicated instance of table 6.2 Place ads only at the beginning of the blocks (the block is any code included in the curly supports z and z). Do not expect variables to be advertised before they are used for the first time; This can confuse an inexperienced programmer and make it difficult to port code within the area. empty MyMethod () q int1; / Start of block if (condition) int2; / The beginning of the statement block if ... q The only exception to this rule are the for cycle indexes, which in Java can be advertised in the for: for: for (int i q 0; i < maxLoops; i q) Avoid local ads that overlap higher-level ads. For example, do not declare the same variable in front of the code block and in the inner block: count int; ... (the condition) is the int count; / DO! ... q 6.3 Startup Try to start local variables where you advertise them. The only reason not to start a variable where it is ad is if its initial value depends on some preliminary calculations. 6.4 Class and interface announcements when writing java classes and interfaces, you must follow the following formatting rules: No between the method name and the support (in the nutri that indicates the list of its parameters The z-aperture bracket is placed at the end of the same line, where the declaration of the class or interface Z-lock support is placed on a separate line with the same indentation of the corresponding aperture operator, except when we have an empty operator, so it should appear immediately after ClassSample. int ivar2; Sample(int i, int j) { ivar1 = i; ivar2 = j; } int emptyMethod() {} ... } Among the methods should be a blank line 7 - Operators 7.1 Simple operators Each line must contain no more than one operator. For example: argv argc--; Avoid! Do not use a cígula for a grouping of multiple operators, even if it is visible to the naked eye. For example: if (err) q Format.print (System.out, error), exit (1); /VERY GOOD! 7.2 Composite Operators - are operators containing lists of operators wrapped in curly supports operators q. Examples are given in the following sections. Nested operators must have a one-level indentation more than a composite operator. The opening bracket must be at the end of the line from which the composite operator begins; the closing bracket should begin with a new line and an indentation corresponding to the beginning of the compound statement. Appliances are used in all operators, even the only ones, when they are part of a control structure, such as whether or for. This is to avoid errors in case new operators are added when you forgot to specify handsets (if there are no appliances, the control design is run only one line after it for the signal ;). 7.3 Return Statement, Return Return, Return value, should not use brackets unless its use makes the return value clearer. For example: return; return myDisk.size(); return (size ? size : defaultSize); 7.4 Operators, if formal Operators, if formal, if they are other Operators (as well as food use) if there is the following view: if (condition) q operators; if (condition) q operators; q other operators; if (condition) q operators; q case (condition) q operators q; Avoid the following: if (condition) /BEFORE! THERE ARE LOST CURLY APPLIANCES! Operator 7.5 Cycle Operator for Cycle Operator To must have the following view: for (initialization; condition; iteration) q q operators; q Empty for loop operator (one where all work is done at startup, since and iteration) must have the following view: for (initialization; condition; iteration); When using a cígule at the initialization or iteration of the loop statement, avoid using more than three variables. If necessary, use separate statements before the loop to (for the boot block case) or at the end of the loop (for the iteration block case). 7.6 The while cycle operator must have the following view: while (condition) q operators; q Empty loop operator, while must have the following view: while (condition); 7.7 Do-while Do-while cycle operator must have the following view: make q operators; q while (condition); 7.8 Switch operator switch must have the following form: switch case (condition) Operators Failure in the DEF case: operators; break; XY' case: Operators; break; Standard: Operators; break; Whenever a choice fails (not including a break declaration), add a comment where the break declaration is usually located. This is shown in the previous code example with the /q comment failure. Each switch operator must include default. The break statement in the default selection is superfluous, but avoids the error if another choice is later added. 7.9 The try-catch try-catch operator must have the following format: try q operators; catch (ExceptionClass and) q operators; 8 - Blank lines 8.1 Empty Strings improve readability, highlighting logically linked sections of code. Two blank lines should always be used in the following cases: Between sections in the source code file between class definitions and interfaces A blank line should always be used in the following cases: Between the methods between the local method variables and their first operator before the block (see section 5.1.1) or a one-line comment (see section 5.1.2) between logical areas of code within the method to improve the readability of 8.2 Closing gaps should be placed under the following circumstances: The keyword followed by a bracket must be separated by a gap. For example: while (true) q ... Note that the space should not be used between the method name and its opening support. This helps distinguish keywords from method calls. The split gap should appear after the cipgula in the argument list. All binary Except. The gap should never be shared by operands and their non-ary operators, such as the unary minus, increment (z) and decoment (--) of their operands. For example: a q c q d; a = (a + b) / (c * d); while (d++ = s++) { n++; } prints (size is + foo +); Expressions in the loop operator must be separated by spaces. For example: for (expr1; expr2; expr3) A gap should follow the type. For example: myMethod (byte) aNum, (Object) x); myFunc((int) (cp + 5), ((int) (i + 3)) + 1); 9 - The nomination agreement makes the nomination agreement more understandable, making them easier to read. They can also provide information about the function of the identifier - for example, whether it is a constant, a package, or a class, makes it easier to understand and read the code. The rules of this section are fundamental. More specific rules are given in the table: The type of id rules for naming class names Class names must be nouns typed into a mixed record (each word with a capital letter with a capital letter). Make sure that your class names are simple and visual. Use whole words - avoid abbreviations and abbreviations (unless the acronym makes the class name more visual and clear than a long form, such as URL or HTML). Raster class; ImageSprite class; Interfaces Interfaces must also begin with capital letters and be named according to the same rules. RasterDelegate interface; Storage interface; Methods methods must be typed into a mixed record, the first letter of the first word in the lower record, all subsequent words the first letter in Register. run(); runFast(); getBackground(); Variables Except for variables, all class instances and class constants are recruited into a mixed record with the first symbol in the lower record. Subsequent words are collected with a capital letter. The names of variables should be short, but meaningless. The name of the chosen variable should be memorable - that is, a brief description of what it contains. Single-character variable names should be avoided except for point temporary variables. Common names for time variables are i, j, k, m, and n for integers; c, d and e for int i characters; char *cp; float myWidth; Variable name constants are advertised by class constants, and ansi constants must be typed in the upper register (uppercase letters) with a shared sign emphasizing z between words. (The ansi constant should be avoided to facilitate debugging.) int MIN_WIDTH = 4; int MAX_WIDTH = 999; int GET_THE_CPU = 1; 10 - Practical Application 10.1 Ensuring access to instance and class variables Do not make any instance or class variables without any good reason. There are often cases where class fields should not be defined or read directly - their reading and writing should only occur when methods are called. An example of proper use of public fields can be when a class describes only one data structure, without any behavior. In other words, if you could use the structure instead of class (if Java supported the structure), then you could make variables for public class instances. 10.2 Appeal to Variables and Class Methods Avoid to access static fields and class methods. Instead, use the class name. For example: classMethod(); OK AClass.classMethod(); OK anObject.classMethod(); Avoid! 10.3 Numeric Constant constants (literals) should not be encoded directly, except for -1, 0, and 1, which can be used in cycles to control the counter. 10.4 Examples of value assignment to variables and operators, etc. Avoid assigning value to certain variables in a single expression. This makes reading more difficult. For example: fooBar.fChar q barFoo.lchar q 'c'; Avoid! Do not use an assignment statement in locations where it can be easily confused with the comparison operator. For example: if (c) q) q//DOSY! In Java it is forbidden... it is better to write like this: if (c) ! q) ! 0) Q... Do not use the attachments invested in an attempt to speed up the program. This is the compiler's work, and in addition, it actually rarely helps. For example: d q (a b q c) q r; Incorrectly! It should be written as: a q b q c; d = a + r; 10.5 Different programming techniques 10.5.1 Round brackets Typically, the pricctic refrain is the use of round brackets in expressions containing different operators to avoid problems with operators' priority. Even if the operator priority seems obvious to you, it may not be so for others - you shouldn't assume that other programmers know the priority as well as you do. if (a q b q q c) (a q b) (c q d)) a) For example: if (booleanExpression) q return TRUE; q else q return FALSE; q, instead, should write as: booleanExpression return; Similarly: if { return x; } return y; It is better to note this: return (condition ? x : y); 10.5.3 Expressions before '?' in the conditional statement If the expression contains a binary operator in front of the ternar operator ?:, it must be bracketed. For example: (x > q 0) ? x : -x 10.5.4 Special comments Use XXX in comments to show that this code is wrong but works. Use FIXME to show that the code is wrong and does not work. 11 - Code examples 11.1 Java source file example The following example shows how to format a java source file containing a separate class. Interfaces are formatted separately. For more research, read Class Announcement and Interfaces and Documentary Commentary. */ java.blah package; import java.blah.blahdy.BlahBlah; The description of the class is here. * @version 1.10 Oct 04 @author Last Name q/ public class Blah extends SomeClass q /here can be a comment about the implementation of the class. the documenting comment for classVar2, which by chance is more than one line q/ private static object classVar2; /q comment documenting the Var1 field instance/ public object instanceVar1; /q comment documenting the Var2 field instance / int of instanceProtectionVar2; / / comment documenting the field instanceVar3 / private object comment documenting the Blah... Here's the implementation... /** * ... comment documenting the method ofSomething... Here's the implementation... /** * ... comment documenting the SomethingElse method... q @param some description of The SomethingElse (Object someParam) You can find an offline version of the reading in my Github repository If you notice an error, highlight a snippet of text and click Ctrl'Enter Ctrl'Enter