

What's in a Name?

Though a coding element will perform the way it was designed to perform, regardless of its name, it's good practice to choose semantics wisely

by *Stephen Kelvin Friedrich*

October 20, 2004

A project I've been involved with for two years involving major refactorings to our over 4,000 classes project has led me to some dark corners in the sources. Working with code that is unknown to me has made one issue very pressing—names. I'm not talking about syntactical issues, which are covered in almost every coding standard and generally agreed upon. Rather, I'm talking about the relation of a name and the meaning or semantics of the concept it names.

In this project I often encounter names that lead me on the wrong track, and, after spending valuable time on something, I finally feel that somebody has played a good joke on me. In the end, most of the designing, modeling, and code writing capture some concepts that are grounded in real-world counterparts. For these concepts you need good names to easily establish the relationship from virtual to real. Other concepts are pure inventions without a real-life counterpart. These, even more so, need good names to visualize their meanings.

Is it hard to choose good names? I don't think so. Most programmers take a casual approach to this task. After all, laziness has been cited as one of a programmer's biggest virtues. Being focused on getting the work done means that naming is perceived as an obstacle that gets in the way. Code will run just as well if you name a variable "a" or if you name it "averageGasConsumption." Still, a name is meant to communicate something to those who will work with the code after you have finished it, and that may even be you a week later. You'll write that line of code only once, but chances are you and your team will be reading it a dozen times later on. Several factors contribute to poor naming habits.

Poor instruction If your instructors spent more time on lessons of good naming than the occasional lecture to "choose expressive names," you are lucky (and went to a different university than I did). For sure, if you named those variables in your calculator homework assignment "a" and "b," you'd get back some bad remarks, but "result" and "plus" would have been dandy.

Bad examples Most code examples you see are short and specific. That comes naturally, of course, with their intention to explain fundamental algorithms, APIs, or language features. So, examples typically neglect anything else, which may be justified in some areas like error checking and exception handling (those issues at least receive some attention separately). Yet good names would do nothing but help most examples be more easily understood.

No guidelines, no conventions You already have your company-specific set of code style guidelines—if not, start with Sun's developers' site. It probably advocates variable names to start in lowercase, class names to start in uppercase, and continue in camel case. Maybe it also tells you to use nouns for class and variable names and verbs for methods. Chances are that besides those (important) syntactical instructions, further directions on choosing a good name are nonexistent.

Good and Bad

Some of my favorite bad names have become `secretHashMap` and `mef`. I don't want to point any fingers, but bad examples can be helpful to improve your own standards, and I have made my share of bad name selections.

What does a variable named `mef` tell you about its purpose? Nothing. If, however, you already know the application area in some depth, you might guess that it stands for `ModelElementFactory`. But if you already know, who cares about the name?

The name should be helpful to the unknowing eye. A similar argument holds for `secretHashMap`. All that the uninitiated reader might infer from it is: "some hack going on here." Staying for a moment with the importance of *semantic value*, other examples come to my mind, and all of these should almost always make you reconsider the concept you are trying to name: `Manager`, `Handler`, `System`, and

Process. Like a professor of mine once said, "If you don't know what it is, call it a system. If you don't know what it does, call it a process."

If you haven't noticed already, Fortran is dead, so please let go of that infamous "i." It's not the worst thing to use for an integer variable in a small loop, and it traditionally points to that very syntax (int loop variable). But then with modern IDE's code completion, it's not much more work to type "bookIndex" instead, which will be much nicer for future readers.

Think of "banana." It's a fruit—yellow, tasty, peel, grocery—right? It could be, or it could be that large, plastic inflatable that's fun to ride on and pulled by a motorboat through the bay outside my holiday hotel. If a name leads you to think of one type of object, while in fact it refers to some different type of object altogether, you need to get more specific. (Something like "bananaBoat" would have provoked more accurate associations.) This issue is not about semantic value itself, but about using correct semantics.

This kind of misuse is common. For example, have a look at Swing's JTable implementation: `TableCellRenderer getCellRenderer(int row, int column);` Come again? A "column" is an "int"? This complaint might seem nitpicky, as we all understand the method's purpose, but this bad practice is so widespread that the overall quality of code will improve a lot if you choose better names instead (which would be `rowIndex` and `columnIndex` in this case, of course). This is even more so, because you often see a variable without seeing its declaration also. For example, look at this statement:

```
bookTable.removeColumn(  
    authorColumn)
```

Author column probably is the index of the column showing the author information. But wait, in fact it isn't. Look at the method's signature:

```
void removeColumn(  
    TableColumn aColumn);
```

Don't get me wrong. I just loathe that naming style you see in C/C++ programs with variable names like "IpszFile" with type encoded in the name (Hungarian notation). That style has nothing to do with modern, object-oriented software. Still, the name should give one an idea about what kinds of objects to which it applies.

Simple as it is, *consistency* has always been the hardest for me. First, it's `numberOfBooks` and a couple of lines below `pageCount`. I write `weightX` and `weightY` and a minute afterwards `xButton` and `yButton`. The canonical form I propose has become a habit with me, and it is much easier now to stay consistent.

Better Names

I propose some practices that can improve code a lot and help achieve better quality. Naming is not a side issue. It is the most fundamental mechanism of your work. It helps you to think about what you want to do and to communicate that clearly. It speeds up development by making your code more solid and maintainable. Take your time to find a good name. Consequences of naming differ in importance, depending on the type of name.

Package names – Discussing these with your team can establish a common understanding of what is supposed to be in that package and how it relates to the system as a whole.

Class and interface names – Classes are the fundamental building blocks of object-oriented software, and names are what give you a handle on classes. So a bad name can make somebody try to build a wall with a roof tile, which might work but will damage the overall architecture of your house. Put some effort into choosing a name that captures the concept that the class represents. Doing so will give you a better idea of that concept itself.

Public member names – (You don't have any members public but methods and constants, right?) These names represent the interface of a class, so bad names are an invitation for your coworkers to

misuse the class and spend ages on debugging and rereading your code. A single important convention is not to name a method `get...()` if it changes the (externally visible) state of a class.

Other member names – These names will be read whenever somebody works on your code. If you are sloppy with naming, you are claiming that this is the perfect, bug-free, fast implementation that won't ever have to be touched again.

Local variables – A good name has value even here, yet the meaning can often be inferred quickly from the context. Don't lead readers down the wrong track: naming an index into a list of actors "i" doesn't tell the reader much, but naming it `modelElement` will obscure the code.

It is helpful to think like a reader, not like the author. When you want to introduce a name, you probably already have an idea of the concept. Better yet, try to think the other way around: Imagine that you have no idea of the concept, but you have a name only. Does it make you think of the right concept?

Ask your colleagues. Even if you try to take the reader's viewpoint, you still might fail, typically, because your knowledge in the area you are working on is deeper than anyone else's knowledge. Ask, "What do you think of when you hear `ProjectHandler`?" Check it against your concept of `ProjectHandler`. We came up with `ProjectReaderWriter` instead, which is a bit clumsy but at least it says what it is used for.

Say what it is. Have you ever met an `int` in real life? A tiny 3 hopping across your way without telling you whether it's three pieces of cake or three little dwarfs? Ask yourself what it is. Choose the appropriate suffix for your name: `index`, `weight`, `milliSeconds`, `count`, and so on. The same goes for variables of type `string`; don't name it `actor` if it is an `actorName`.

Stringin' Along

Some typical things that are strings are: name, title, message (in log files or message dialogs), and so on. Most of the string variables are probably names; still, it's important to distinguish between the name of an object and that object itself. If you find a lot of string variables that are neither of these, you probably fell for the mis-stringification bug pattern. Once you cling to these suffixes, the nice consequence is that your naming gets much more coherent.

This canonical form is most valuable for me in achieving consistent naming of classes and fields: Always order subwords from most specific to most general. You should always be able to strip the next subword from the left and still have a name that makes sense but is somewhat broader. For instance, the canonical form—`remainingFileCount`—and stripped forms—`fileCount` and `Count`—all make sense. The nonstandard form `numberOfFilesRemaining` results in the silly stripped forms: `ofFilesRemaining`, `filesRemaining` (it's an `int`, not a file collection), or `Remaining`. Following this simple rule will automatically get you to a pretty good level of consistency. Note that I don't say the name `remainingFileCount` is bad in itself, it just doesn't follow any single naming scheme that can also be applied to other names; therefore, it makes consistency hard to achieve.

Comments get out of date easily, but code is always current. In ancient times when structural programming was hip, you were told that procedures are the way to reuse some code bits. (But maybe you are lucky—and a lot younger than me—and your first language was Java, Eiffel, or Smalltalk rather than Modula, Pascal, or even Basic). Code reuse is just a nice side effect. More importantly, a procedure (or method, the modern equivalent) is the best way to isolate a concept and give it a name. Because most coders still write comments that do nothing but describe code, here is an example that might convince you to break that habit. I found this code fragment:

```
try {
int rotLength =
    Integer.parseInt(
        rotationLengthField.getText()
        .trim()); // get rot len
RotAscii cipher = new
    RotAscii(rotLength);
    // new cipher
textArea.setText(
```

```

        cipher.transform(
            textArea.getText());
    // transform
}
catch(Exception ex) {
    /* Show exception */
    ExceptionDialog.show(this,
        "Invalid rotation length: ",
        ex);
}

```

The original text goes on and proposes this form as better style:

```

introtLength = 0;
    // rotation length
RotAscii cipher = null;
    // ASCII rotation cipher

try {
    // Get rotation length field and
    // convert to integer.
    rotLength = Integer.parseInt(
        rotationLengthField.getText()
        .trim());

    // Create ASCII rotation cipher
    // and transform the user's text
    // with it.
    cipher = new RotAscii(
        rotLength);
    textArea.setText(
        cipher.transform(
            textArea.getText()));
}
catch(Exception ex) {
    // Report the exception to the
    // user.
    ExceptionDialog.show(this,
        "Invalid rotation length: ",
        ex);
}

```

I'd say that every comment in the preceding snippet is superfluous. Why try to fix bad naming by attaching comments? Why use comments to identify a concept that may be isolated and named by extracting it to a method? This approach is much better:

```

try {
    int rotationLength =
        getRotationLength();
    AsciiRotationCipher cipher = new
        AsciiRotationCipher(
            rotationLength);
    String plainText =
        getPlainText();
    String cryptedException =
        cipher.transform(plainText);
    showCryptedException(cryptedException);
}
catch(Exception e) {
    reportExceptionToUser(
        "Invalid rotation length: ",
        e);
}

```

(I won't comment anymore about the bad style of generically catching Exception.) A modern IDE with refactoring capabilities can help a lot in extracting code to a method.

You want to refactor carefully. When you rename a class, also rename dependant variables, which can be tedious but it prevents names from getting out of sync. Consider this declaration:

```
private ProjectHandler handler
```

We renamed that class so that the declaration would now read:

```
private ProjectReaderWriter  
    handler
```

which is, of course, bad. We use JetBrains' Idea IDE, which searches for variable names when a class is renamed and proposes new names for variables as well (which you can accept or decline). This approach goes further than a simple text search in that it also searches and handles substrings; that is, it proposes readerWriter for the variable's name in this case—kind of like magic.

Avoid abbreviations, except when they are very common. Remember code completion. It saves you only a few keystrokes, but a proper name saves somebody else from guesswork. Using abbreviations also makes it harder to rename variables accordingly when you rename a class. Not even Idea can propose a new name for a variable prh of class ProjectHandler when you rename that class to ProjectReaderWriter.

Of course, you do want to use very well-known abbreviations such as tutorialUrl. If you decide to use an abbreviation, make only the first letter uppercase to show word separations clearly. Compare HTTPURL and httpUrl. (As a funny side note, Sun engineers managed to mix abbreviation case style in a single name: java.net.HttpURLConnection.)

About the Author

Stephen Kelvin Friedrich is chief software architect at Gentleware AG. His previous work included developing large-scale distributed systems in the telecommunications business and consulting services in the financial services area. Stephen currently enjoys working among a team of skilled developers to further improve the Poseidon for UML modeling tool. Contact Stephen at stephen.kelvin@gentleware.com.