What's the best approach to naming classes?

Asked 16 years, 3 months ago Modified 4 years, 9 months ago

Viewed 65k times

106

Coming up with good, precise names for classes is notoriously difficult. Done right, it makes code more self-documenting and provides a vocabulary for reasoning about code at a higher level of abstraction.

Classes which implement a particular design pattern might be given a name based on the well known pattern name (e.g. FooFactory, FooFacade), and classes which directly model domain concepts can take their names from the problem domain, but what about other classes? Is there anything like a programmer's thesaurus that I can turn to when I'm lacking inspiration, and want to avoid using generic class names (like FooHandler, FooProcessor, FooUtils, and FooManager)?

naming

Share

Improve this question

Follow

asked Sep 1, 2008 at 14:55

Henry

1,489 ● 2 ● 11 ● 17

8 this is an excellent question! Closing it is IMHO unjustified, better would be to migrate programmers site. – Timofey Dec 4, 2014 at 12:06

2 I agree, should be reopened or moved – ftl Aug 9, 2017 at 22:28

1 Related posts - What is a better name than Manager, Processor etc?, Naming convention and structure for utility classes and methods, What names do you find yourself prepending/appending to classes regularly?, & Naming Classes - How to avoid calling everything a “<WhatEver>Manager”? – RBT Jul 11, 2018 at 9:30

1 I have seen a lot of these excellent questions "closed by casperOne". Maybe he could serve as deletionist at Wikipedia? – Perlator Sep 21, 2019 at 7:17

6 Answers

Sorted by: Highest score (default)

70

I'll cite some passages from Implementation Patterns by Kent Beck:

Simple Superclass Name

> "[...] The names should be short and punchy. However, to make the names precise sometimes seems to require several words. A way out of this dilemma is picking a strong metaphor for the computation. With a metaphor in mind, even single words bring with them a rich web of associations, connections, and implications. For

> example, in the HotDraw drawing framework, my first name for an object in a drawing was **DrawingObject**. Ward Cunningham came along with the typography metaphor: a drawing is like a printed, laid-out page. Graphical items on a page are figures, so the class became **Figure**. In the context of the metaphor, **Figure** is simultaneously shorter, richer, and more precise than **DrawingObject**."

Qualified Subclass Name

> "The names of subclasses have two jobs. They need to communicate what class they are like and how they are different. [...] Unlike the names at the roots of hierarchies, subclass names aren’t used nearly as often in conversation, so they can be expressive at the cost of being concise. [...]
>
> Give subclasses that serve as the roots of hierarchies their own simple names. For example, *HotDraw* has a class **Handle** which presents figure- editing operations when a figure is selected. It is called, simply, **Handle** in spite of extending **Figure**. There is a whole family of handles and they most appropriately have names like **StretchyHandle** and **TransparencyHandle**. Because **Handle** is the root of its own hierarchy, it deserves a simple

> superclass name more than a qualified subclass name.
>
> Another wrinkle in subclass naming is multiple-level hierarchies. [...] Rather than blindly prepend the modifiers to the immediate superclass, think about the name from the reader's perspective. What class does he need to know this class is like? Use that superclass as the basis for the subclass name."

Interface

> Two styles of naming interfaces depend on how you are thinking of the interfaces. Interfaces as classes without implementations should be named as if they were classes (*Simple Superclass Name*, *Qualified Subclass Name*). One problem with this style of naming is that the good names are used up before you get to naming classes. An interface called **File** needs an implementation class called something like **ActualFile**, **ConcreteFile**, or (yuck!) **FileImpl** (both a suffix and an abbreviation). In general, communicating whether one is dealing with a concrete or abstract object is important, whether the abstract object is implemented as an interface or a superclass is less important. Deferring the distinction between interfaces and

> superclasses is well >supported by this style of naming, leaving you free to change your mind later if that >becomes necessary.
>
> Sometimes, naming concrete classes simply is more important to communication than hiding the use of interfaces. In this case, prefix interface names with “I”. If the interface is called **IFile**, the class can be simply called **File**.

For more detailed discussion, buy the book! It's worth it! :)

Share Improve this answer Follow

edited Jun 20, 2020 at 9:12
Community Bot
1 ● 1

answered Sep 7, 2008 at 1:05

Marcio Aguiar
14.5k ● 6 ● 40 ● 42

---

2 "Sometimes, naming concrete classes simply is more important to communication than hiding the use of interfaces. In this case, prefix interface names with “I”. If the interface is called IFile, the class can be simply called File." It's so funny, cause in the book Clean Code by Robert C. Martin, I found out that we would rather name implementations as FileImpl, rather than naming the interface like IFile, because we do not want the client to know we are serving them an interface. And in the book ^ there are quotes from the book you are referring to :) – Cristian Ciobotea Sep 3, 2017 at 10:28

43

Always go for MyClassA, MyClassB - It allows for a nice alpha sort..

I'm kidding!

This is a good question, and something I experienced not too long ago. I was reorganising my codebase at work and was having problems of where to put what, and what to call it..

The **real** problem?

I had classes doing too much. If you try to adhere to the single responsibility principle it will make everything all come together much nicer.. Rather than one monolithic *PrintHandler* class, you could break it down into *PageHandler* , *PageFormatter* (and so on) and then have a master *Printer* class which brings it all together.

In my re-org, it took me time, but I ended up binning a lot of duplicate code, got my codebase much more logical and learned a hell of a lot when it comes to thinking before throwing an extra method in a class :D

I would **not** however recommend putting things like pattern names into the class name. The classes interface should make that obvious (like hiding the constructor for a singleton). There is nothing wrong with the generic name, if the class is serving a generic purpose.

Good luck!

Share Improve this answer

Follow

answered Sep 1, 2008 at 15:16

Rob Cooper
28.9k ● 26 ● 105 ● 142

Agree with not putting pattern names into the class name. What if the pattern changes later on when the implementation changes? – thegreendroid May 18, 2014 at 6:12

2 I'd rather put a name of pattern in a name. Why? Because if I have to look at the implementation details (like private constructor) in order to find out that it's a singleton, that's slowing me down as a reader and depending on my skill level, I may not figure out that it's actually a class part of the general pattern. Singleton and private constructor is a questionable example, but for more complicated patterns (Visitor, Adapter etc.) it's getting quite complicated. If the pattern changes, the chaces are that those classes won't exist anymore... – dragan.stepanovic Apr 1, 2015 at 15:03

30

Josh Bloch's excellent talk about good API design has a few good bits of advice:

- Classes should do one thing and do it well.
- If a class is hard to name or explain then it's probably not following the advice in the previous bullet point.
- A class name should instantly communicate what the class is.
- Good names drive good designs.

If your problem is what to name exposed internal classes, maybe you should consolidate them into a larger class.

If your problem is naming a class that is doing a lot of different stuff, you should consider breaking it into multiple classes.

If that's good advice for a public API then it can't hurt for any other class.

Share Improve this answer

Follow

edited Aug 20, 2015 at 20:22

Subhash Dike

1,891 ● 1 ● 23 ● 37

answered Sep 7, 2008 at 1:41

Josh Segall

4,283 ● 4 ● 32 ● 28

1 video link on youtube youtube.com/watch?v=aAb7hSCtvGw – Andrew Harry Feb 25, 2012 at 10:35

15

If you're stuck with a name, sometimes just giving it *any* half-sensible name with commitment to revising it later is a good strategy.

Don't get naming paralysis. Yes, names are very important but they're not important enough to waste huge amounts of time on. If you can't think up a good name in 10 minutes, move on.

Share Improve this answer

Follow

answered Sep 1, 2008 at 15:20

Simon Johnson

7,902 ● 5 ● 38 ● 50

10

If a good name doesn't spring to mind, I would probably question whether there is a deeper problem - is the class serving a good purpose? If it is, naming it should be pretty straightforward.

Share Improve this answer Follow

answered Sep 1, 2008 at 15:12

Luke Halliwell
7,352 ● 6 ● 32 ● 31

4

If your "FooProcessor" really does process foos, then don't be reluctant to give it that name just because you already have a BarProcessor, BazProcessor, etc. When in doubt, obvious is best. The other developers who have to read your code may not be using the same thesaurus you are.

That said, more specificity wouldn't hurt for this particular example. "Process" is a pretty broad word. Is it really a "FooUpdateProcessor" (which might become "FooUpdater"), for example? You don't have to get too "creative" about the naming, but if you wrote the code you probably have a fairly good idea of what it does and doesn't do.

Finally, remember that the bare class name isn't all that you and the readers of your code have to go on - there are usually namespaces in play as well. Those can often give readers enough context to see clearly what your class if really for, even if its bare name is fairly generic.

Share Improve this answer

Follow

answered Sep 1, 2008 at 16:08

McKenzieG1

14.2k ● 7 ● 38 ● 42