

How do you find a needle in a haystack?

Asked 16 years, 4 months ago Modified 8 months ago

Viewed 14k times



76

When implementing a needle search of a haystack in an object-oriented way, you essentially have three alternatives:



1. `needle.find(haystack)`
2. `haystack.find(needle)`
3. `searcher.find(needle, haystack)`

Which do you prefer, and why?

I know some people prefer the second alternative because it avoids introducing a third object. However, I can't help feeling that the third approach is more conceptually "correct", at least if your goal is to model "the real world".

In which cases do you think it is justified to introduce helper objects, such as the searcher in this example, and when should they be avoided?

oop

class-design

program-structure

Share

edited Aug 28, 2008 at 19:00

Improve this question

Follow

community wiki

4 revs, 4 users 100%

Anders Sandvig

-
- 1 +1 Good title, and good semantics question. Most every OO language hits a problem like this (think Python's `string.join` method). – [new123456](#) Mar 30, 2011 at 0:56
-

30 Answers

Sorted by:

Highest score (default)



Of the three, I prefer option #3.

56



The [Single Responsibility Principle](#) makes me not want to put searching capabilities on my DTOs or models. Their responsibility is to be data, not to find themselves, nor should needles need to know about haystacks, nor haystacks know about needles.



For what it's worth, I think it takes most OO practitioners a LONG time to understand why #3 is the best choice. I did OO for a decade, probably, before I really grokked it.

@wilhelmtell, C++ is one of the very few languages with template specialization that make such a system actually work. For most languages, a general purpose "find" method would be a HORRIBLE idea.

Share Improve this answer

answered Aug 23, 2008 at 1:08

Follow



Brad Wilson

70.4k ● 9 ● 77 ● 85



53

Usually actions should be applied to what you are doing the action on... in this case the haystack, so I think option 2 is the most appropriate.



You also have a fourth alternative that I think would be better than alternative 3:



```
haystack.find(needle, searcher)
```

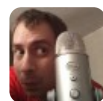


In this case, it allows you to provide the manner in which you want to search as part of the action, and so you can keep the action with the object that is being operated on.

Share Improve this answer

answered Aug 23, 2008 at 0:05

Follow




Mike Stone

44.6k ● 30 ● 114 ● 140

1 This looks a lot like the Strategy Pattern (en.wikipedia.org/wiki/Strategy_pattern) – ARKBAN Nov 13, 2008 at 16:36

This looks a lot like a violation of the Single Responsibility Principle. Option #3 is definitely a better design choice.
– Kyle W. Cartmell Dec 29, 2010 at 18:26

1 Actually this looks more like the Visitor pattern; certainly that's its implied intent. "searcher" could be any search algorithm which a goal of the Visitor. Certainly *something* of "Haystack" must be exposed (the HayCollection I suppose)

for "searcher" to search; but this do not mean SRP is violated in fact or in spirit. en.wikipedia.org/wiki/Visitor_pattern.
– radarbob Mar 8, 2012 at 20:29 



There is another alternative, which is the approach utilized by the STL of C++:

17

```
find(haystack.begin(), haystack.end(), needle)
```



I think it's a great example of C++ shouting "in your face!" to OOP. The idea is that OOP is not a silver bullet of any kind; sometimes things are best described in terms of actions, sometimes in terms of objects, sometimes neither and sometimes both.

Bjarne Stroustrup said in TC++PL that when you design a system you should strive to reflect reality under the constraints of effective and efficient code. For me, this means you should never follow anything blindly. Think about the things at hand (haystack, needle) and the context we're in (searching, that's what the expression is about).

If the emphasis is about the searching, then using an algorithm (action) that emphasizes searching (i.e. is flexibly to fit haystacks, oceans, deserts, linked lists). If the emphasis is about the haystack, encapsulate the find method inside the haystack object, and so on.

That said, sometimes you're in doubt and have hard times making a choice. In this case, be object oriented. If you change your mind later, I think it is easier to extract an action from an object then to split an action to objects and classes.

Follow these guidelines, and your code will be clearer and, well, more beautiful.

Share Improve this answer

answered Aug 23, 2008 at 1:05

Follow



wilhelmtell

58.6k ● 20 ● 97 ● 131



15



I would say that option 1 is completely out. The code should read in a way that tells you what it does. Option 1 makes me think that this needle is going to go find me a haystack.

Option 2 looks good if a haystack is meant to contain needles. ListCollections are always going to contain ListItems, so doing `collection.find(item)` is natural and expressive.

I think the introduction of a helper object is appropriate when:

1. You don't control the implementation of the objects in question
IE: `search.find(ObsecureOSObject, file)`
2. There isn't a regular or sensible relationship between the objects

IE: nameMatcher.find(houses,trees.name)

Share Improve this answer

answered Aug 22, 2008 at 23:56

Follow



Tilendor

48.9k ● 17 ● 53 ● 58

You could use `needle.findIn(haystack)`. The variable name isn't the issue here, it's the syntax. – [sep332](#) Nov 13, 2008 at 16:23



11



I am with Brad on this one. The more I work on immensely complex systems, the more I see the need to truly decouple objects. He's right. It's obvious that a needle shouldn't know anything about haystack, so 1 is definitely out. But, a haystack should know nothing about a needle.



If I were modeling a haystack, I might implement it as a collection -- but as a collection of *hay* or *straw* -- not a collection of needles! However, I would take into consideration that stuff does get lost in a haystack, but I know nothing about what exactly that stuff. I think it's better to not make the haystack look for items in itself (how *smart* is a haystack anyway). The right approach to me is to have the haystack present a collection of things that are in it, but are not straw or hay or whatever gives a haystack its essence.

```
class Haystack : ISearchableThingsOnAFarm {  
    ICollection<Hay> myHay;
```

```

ICollection<IStuffSmallEnoughToBeLostInAHaystack>
stuffLostInMe;

    public ICollection<Hay> Hay {
        get {
            return myHay;
        }
    }

    public
ICollection<IStuffSmallEnoughToBeLostInAHayStack>
LostAndFound {
    get {
        return stuffLostInMe;
    }
}

class Needle :
IStuffSmallEnoughToBeLostInAHaystack {
}

class Farmer {
    Search(Haystack haystack,

IStuffSmallEnoughToBeLostInAHaystack itemToFind)
}

```

There's actually more I was going to type and abstract into interfaces and then I realized how crazy I was getting. Felt like I was in a CS class in college... :P

You get the idea. I think going as loosely coupled as possible is a good thing, but maybe I was getting a bit carried away! :)

Share Improve this answer

answered Aug 23, 2008 at 1:52

Follow



Peter Meyer

26.1k ● 1 ● 35 ● 53



If both Needle and Haystack are DAOs, then options 1 and 2 are out of the question.

6



The reason for this is that DAOs should only be responsible for holding properties of the real world objects they are modeling, and only have getter and setter methods (or just direct property access). This makes serializing the DAOs to a file, or creating methods for a generic compare / generic copy easier to write, as the code wouldn't contain a whole bunch of "if" statements to skip these helper methods.



This just leaves option 3, which most would agree to be correct behaviour.

Option 3 has a few advantages, with the biggest advantage being unit testing. This is because both Needle and Haystack objects can be easily mocked up now, whereas if option 1 or 2 were used, the internal state of either Needle or Haystack would have to be modified before a search could be performed.

Secondly, with the searcher now in a separate class, all search code can be held in one place, including common search code. Whereas if the search code was put into the DAO, common search code would either be stored in a

complicated class hierarchy, or with a Searcher Helper class anyway.

Share Improve this answer

answered Sep 8, 2008 at 3:47

Follow



jklp

2,101 ● 2 ● 23 ● 38



5



This entirely depends on what varies and what stays the same.

For example, I am working on a (non-OOP) [framework](#) where the find algorithm is different depending both on the type of the needle and the haystack. Apart from the fact that this would require double-dispatch in an object-oriented environment, it also means that it isn't meaningful to write either `needle.find(haystack)` or to write `haystack.find(needle)`.

On the other hand, your application could happily delegate finding to either of both classes, or stick with one algorithm altogether in which case the decision is arbitrary. In that case, I would prefer the `haystack.find(needle)` way because it seems more logical to apply the finding to the haystack.

Share Improve this answer

answered Aug 22, 2008 at 23:54

Follow



Konrad Rudolph

545k ● 139 ● 956 ● 1.2k



5



When implementing a needle search of a haystack in an object-oriented way, you essentially have three alternatives:

1. `needle.find(haystack)`
2. `haystack.find(needle)`
3. `searcher.find(needle, haystack)`

Which do you prefer, and why?

Correct me if I'm wrong, but in all three examples you already *have* a reference to the needle you're looking for, so isn't this kinda like looking for your glasses when they're sitting on your nose? :p

Pun aside, I think it really depends on what you consider the responsibility of the haystack to be within the given domain. Do we just care about it in the sense of being a thing which contains needles (a collection, essentially)? Then `haystack.find(needlePredicate)` is fine. Otherwise, `farmBoy.find(predicate, haystack)` might be more appropriate.

[Share](#) [Improve this answer](#)

[Follow](#)

answered Aug 24, 2008 at 21:05



[Fredrik Kalseth](#)

14.2k ● 4 ● 26 ● 18

`needle` may be a type. But the ambiguity leads one to believe that this may be a chicken-and-egg situation.

– [new123456](#) Mar 30, 2011 at 0:58



4



To quote the great authors of [SICP](#),

Programs must be written for people to read, and only incidentally for machines to execute

I prefer to have both methods 1 and 2 at hand. Using ruby as an example, it comes with `.include?` which is used like this

```
haystack.include? needle
=> returns true if the haystack includes the
needle
```

Sometimes though, purely for readability reasons, I want to flip it round. Ruby doesn't come with an `in?` method, but it's a one-liner, so I often do this:

```
needle.in? haystack
=> exactly the same as above
```

If it's "more important" to emphasise the haystack, or the operation of searching, I prefer to write `include?`. Often though, neither the haystack or the search is really what you care about, just that the object is present - in this case I find `in?` better conveys the meaning of the program.

Share Improve this answer

answered Aug 23, 2008 at 4:05

Follow



Orion Edwards

123k ● 66 ● 245 ● 339



3



It depends on your requirements.

For instance, if you don't care about the searcher's properties (e.g. searcher strength, vision, etc.), then I would say `haystack.find(needle)` would be the cleanest solution.



But, if you do care about the searcher's properties (or any other properties for that matter), I would inject an `ISearcher` interface into either the haystack constructor or the function to facilitate that. This supports both object oriented design (a haystack has needles) and inversion of control / dependency injection (makes it easier to unit test the "find" function).

Share Improve this answer

answered Aug 23, 2008 at 1:17

Follow



Kevin Pang

41.4k ● 38 ● 122 ● 173



3



I can think of situations where either of the first two flavours makes sense:

1. If the needle needs pre-processing, like in the Knuth-Morris-Pratt algorithm, `needle.findIn(haystack)` (or `pattern.findIn(text)`) makes sense, because the





needle object holds the intermediate tables created for the algorithm to work effectively

2. If the haystack needs pre-processing, like say in a trie, the `haystack.find(needle)` (or `words.hasAWordWithPrefix(prefix)`) works better.

In both the above cases, one of needle or haystack is aware of the search. Also, they both are aware of each other. If you want the needle and haystack not to be aware of each other or of the search, `searcher.find(needle, haystack)` would be appropriate.

Share Improve this answer

Follow

answered Sep 1, 2008 at 6:04



[Binil Thomas](#)

13.8k ● 10 ● 58 ● 70



Easy: Burn the haystack! Afterward, only the needle will remain. Also, you could try magnets.

2



A harder question: How do you find one particular needle in a pool of needles?



Answer: thread each one and attach the other end of each strand of thread to a sorted index (i.e. pointers)



Share Improve this answer

Follow

answered Aug 28, 2008 at 18:35



[John Douthat](#)

41.1k ● 11 ● 70 ● 67

A mix of 2 and 3, really.



2



Some haystacks don't have a specialized search strategy; an example of this is an array. The only way to find something is to start at the beginning and test each item until you find the one you want.

For this kind of thing, a free function is probably best (like C++).

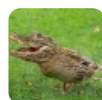
Some haystacks can have a specialized search strategy imposed on them. An array can be sorted, allowing you to use binary searching, for example. A free function (or pair of free functions, e.g. `sort` and `binary_search`) is probably the best option.

Some haystacks have an integrated search strategy as part of their implementation; associative containers (hashed or ordered sets and maps) all do, for instance. In this case, finding is probably an essential lookup method, so it should probably be a method, i.e. `haystack.find(needle)`.

[Share](#) [Improve this answer](#)

answered Aug 28, 2008 at 20:34

[Follow](#)



[DrPizza](#)

18.3k ● 7 ● 42 ● 53



2



The answer to this question should actually depend on the domain the solution is implemented for.

If you happen to simulate a physical search in a physical haystack, you might have the classes

- `Space`



- Straw
- Needle
- Seeker

Space

knows which objects are located at which coordinates
implements the laws of nature (converts energy, detects collisions, etc.)

Needle, Straw

are located in Space
react to forces

Seeker

interacts with space:

moves hand, applies magnetic field, burns hay, applies x-rays, looks for needle...

Thus `seeker.find(needle, space)` or
`seeker.find(needle, space, strategy)`

The haystack just happens to be in the space where you are looking for the needle. When you abstract away space as a kind of virtual machine (think of: the matrix) you could get the above with haystack instead of space (*solution 3/3b*):

`seeker.find(needle, haystack)` or
`seeker.find(needle, haystack, strategy)`

But the matrix was the Domain, which should only be replaced by haystack, if your needle couldn't be

anywhere else.

And then again, it was just an analogy. Interestingly this opens the mind for totally new directions:

1. Why did you loose the needle in the first place? Can you change the process, so you wouldn't loose it?
2. Do you have to find the lost needle or can you simply get another one and forget about the first? (Then it would be nice, if the needle dissolved after a while)
3. If you loose your needles regularly and you need to find them again then you might want to

- make needles that are able to find themselves, e.g. they regularly ask themselves: Am I lost? If the answer is yes, they send their GPS-calculated position to somebody or start beeping or whatever:

`needle.find(space)` Or `needle.find(haystack)`

(solution 1)

- install a haystack with a camera on each straw, afterwards you can ask the haystack hive mind if it saw the needle lately:
`haystack.find(needle)` *(solution 2)*
- attach RFID tags to your needles, so you can easily triangulate them

That all just to say that in your implementation *you* made the needle and the haystack and most of the time the matrix on some kind of level.

So decide according to your domain:

- Is it the purpose of the haystack to contain needles?
Then go for solution 2.
- Is it natural that the needle gets lost just anywhere?
Then go for solution 1.
- Does the needle get lost in the haystack by accident? Then go for solution 3. (or consider another recovering strategy)

Share Improve this answer

answered Sep 13, 2008 at 19:56

Follow



user6246

21 ● 1



haystack.find(needle), but there should be a searcher field.

2



I know that dependency injection is all the rage, so it doesn't surprise me that @Mike Stone's `haystack.find(needle, searcher)` has been accepted. But I disagree: the choice of what searcher is used seems to me a decision for the haystack.



Consider two `ISearchers`: `MagneticSearcher` iterates over the volume, moving and stepping the magnet in a manner consistent with the magnet's strength. `QuickSortSearcher` divides the stack in two until the needle is evident in one of the subpiles. The proper choice of searcher may depend upon how large the haystack is (relative to the magnetic field, for instance), how the needle got into the haystack (i.e., is the needle's position truly random or it it biased?), etc.

If you have `haystack.find(needle, searcher)`, you're saying "the choice of which is the best search strategy is best done outside the context of the haystack." I don't think that's likely to be correct. I think it's more likely that "haystacks know how best to search themselves." Add a setter and you can still manually inject the searcher if you need to override or for testing.

Share Improve this answer
Follow

answered Sep 16, 2008 at 3:45



[Larry OBrien](#)

8,606 ● 1 ● 43 ● 75



1

Personally, I like the second method. My reasoning is because the major APIs I have worked with use this approach, and I find it makes the most sense.



If you have a list of things (haystack) you would search for (`find()`) the needle.



Share Improve this answer
Follow

answered Aug 22, 2008 at 23:55



[Dan Herbert](#)

103k ● 51 ● 192 ● 221



1

@Peter Meyer



You get the idea. I think going as loosely coupled as possible is a good thing, but maybe I was getting a bit carried away! :)



Errr... yeah... I think the



`ISstuffSmallEnoughToBeLostInAHaystack` kind of is a red flag :-)

Share Improve this answer

Follow

edited Jun 20, 2020 at 9:12



Community Bot

1 • 1

answered Aug 23, 2008 at 4:09



Orion Edwards

123k • 66 • 245 • 339



1



You also have a fourth alternative that I think would be better than alternative 3:

```
haystack.find(needle, searcher)
```



I see your point, but what if `searcher` implements a searching interface that allows for searching other types of objects than haystacks, and finding other things than needles in them?

The interface could also be implemented with different algorithms, for example:

```
binary_searcher.find(needle, haystack)
vision_searcher.find(pitchfork, haystack)
brute_force_searcher.find(money, wallet)
```

But, as others have already pointed out, I also think this is only helpful if you actually have multiple search algorithms or multiple searchable or findable classes. If not, I agree `haystack.find(needle)` is better because of its simplicity, so I am willing to sacrifice some "correctness" for it.

Share Improve this answer

edited Aug 24, 2008 at 21:47

Follow

answered Aug 24, 2008 at 19:33



Anders Sandvig

21k ● 16 ● 61 ● 74



1



```
class Haystack { //whatever
};
class Needle { //whatever
}:
class Searcher {
    virtual void find() = 0;
};

class HaystackSearcher::public Searcher {
public:
    HaystackSearcher(Haystack, object)
    virtual void find();
};

Haystack H;
Needle N;
HaystackSearcher HS(H, N);
HS.find();
```

Share Improve this answer

answered Aug 28, 2008 at 19:14

Follow



Baltimark

9,272 ● 12 ● 38 ● 35



1



haystack can contain stuffs one type of stuff is needle
finder is something that is responsible for searching of
stuff finder can accept a pile of stuffs as the source of
where to find thing finder can also accept a stuff
description of thing that it need to find

so, preferably, for a flexible solution you would do
something like: IStuff interface

```
Haystack = IList<IStuff> Needle : IStuff
```

```
Finder .Find(IStuff stuffToLookFor, IList<IStuff>  
stuffsToLookIn)
```

In this case, your solution will not get tied to just needle
and haystack but it is usable for any type that implement
the interface

so if you want to find a Fish in the Ocean, you can.

```
var results = Finder.Find(fish, ocean)
```

Share Improve this answer

answered Sep 9, 2008 at 16:56

Follow



Jimmy Chandra

6,580 ● 4 ● 28 ● 38



If you have a reference to needle object why do you
search for it? :) The problem domain and use-cases tell

1



you that you do not need exact position of needle in a haystack (like what you could get from `list.indexOf(element)`), you just need a needle. And you do not have it yet. So my answer is something like this

```
Needle needle =  
(Needle)haystack.searchByName("needle");
```

or

```
Needle needle =  
(Needle)haystack.searchWithFilter(new Filter(){  
    public boolean isWhatYouNeed(Object obj)  
    {  
        return obj instanceof Needle;  
    }  
});
```

or

```
Needle needle =  
(Needle)haystack.searchByPattern(Size.SMALL,  
  
Sharpness.SHARP,  
  
Material.METAL);
```

I agree that there are more possible solutions which are based on different search strategies, so they introduce searcher. There were enough comments on this, so I do not pay attention to it here. My point is solutions above forget about use-cases - what is the point to search for something if you already have reference to it? In the most

natural use-case you do not have a needle yet, so you do not use variable needle.

Share Improve this answer

answered Nov 5, 2008 at 19:21

Follow

community wiki
[Pavel Feldman](#)



1



Brad Wilson points out that objects should have a single responsibility. In the extreme case, an object has one responsibility and no state. Then it can become... a function.

```
needle = findNeedleIn(haystack);
```

Or you could write it like this:

```
SynchronizedHaystackSearcherProxyFactory
proxyFactory =

SynchronizedHaystackSearcherProxyFactory.getInstance
StrategyBasedHaystackSearcher searcher =
    new BasicStrategyBasedHaystackSearcher(
        NeedleSeekingStrategies.getMethodicalInstance());
SynchronizedHaystackSearcherProxy proxy =

proxyFactory.createSynchronizedHaystackSearcherProxy
SearchableHaystackAdapter searchableHaystack =
    new SearchableHaystackAdapter(haystack);
FindableSearchResultObject foundObject = null;
while
    (!HaystackSearcherUtil.isNeedleObject(foundObject))
```

```

{
    try {
        foundObject =
proxy.find(searchableHaystack);
    } catch (GruesomeInjuryException exc) {
        returnPitchforkToShed(); // sigh, i hate
it when this happens
        HaystackSearcherUtil.cleanUp(hay); // XXX
fixme not really thread-safe,
                                                    // but
we can't just leave this mess

        HaystackSearcherUtil.cleanup(exc.getGruesomeMess());
// bug 510000884
        throw exc; // caller will catch this and
get us to a hospital,
                        // if it's not already too late
    }
}
return (Needle)
BarnvardObjectProtocolUtil.createSynchronizedFindab1

```

Share Improve this answer

edited Jan 4, 2010 at 16:46

Follow

community wiki

2 revs

Jason Orendorff

lambda, the ultimate Facade pattern – [Jason Orendorff](#) Nov 20, 2009 at 12:22



1

The haystack shouldn't know about the needle, and the needle shouldn't know about the haystack. The searcher needs to know about both, but whether or not the



haystack should know how to search itself is the real point in contention.



So I'd go with a mix of 2 and 3; the haystack should be able to tell someone else how to search it, and the searcher should be able to use that information to search the haystack.

Share Improve this answer

Follow

edited Jul 3, 2012 at 14:55



user142162

answered Aug 28, 2008 at 18:44



[Mat Noguchi](#)

1,080 ● 6 ● 7



```
haystack.magnet().filter(needle);
```

1

Share Improve this answer

Follow

edited Jul 3, 2012 at 14:55



user142162

answered Aug 24, 2008 at 22:58



[Artur Carvalho](#)

7,149 ● 11 ● 78 ● 109



Is the code trying to find a specific needle or just any needle? It sounds like a stupid question, but it changes the problem.

1



Looking for a specific needle the code in the question makes sense. Looking for any needle it would be more like



```
needle = haystack.findNeedle()
```

or

```
needle = searcher.findNeedle(haystack)
```

Either way, I prefer having a searcher that class. A haystack doesn't know how to search. From a CS perspective it is just a data store with LOTS of crap that you don't want.

Share Improve this answer

edited Jul 3, 2012 at 14:55

Follow



user142162

answered Sep 8, 2008 at 2:56



[John Meagher](#)

24.6k ● 14 ● 56 ● 57

That's a good question. I assume we want to find the specific needle, i.e. one you have lost in a haystack. If you just wanted to find any needle, the magnet approach suggested earlier would also work... ;) – [Anders Sandvig](#) Sep 8, 2008 at 11:29



Definitely the third, IMHO.

0



The question of a needle in a haystack is an example of an attempt to find one object in a large collection of others, which indicates it will need a complex search algorithm (possibly involving magnets or (more likely) child processes) and it doesn't make much sense for a haystack to be expected to do thread management or implement complex searches.

A searching object, however, is dedicated to searching and can be expected to know how to manage child threads for a fast binary search, or use properties of the searched-for element to narrow the area (ie: a magnet to find ferrous items).

Share Improve this answer

answered Nov 13, 2008 at 16:05

Follow

community wiki
[Jeff](#)



0



Another possible way can be to create two interfaces for Searchable object e.g. haystack and ToBeSearched object e.g. needle. So, it can be done in this way

```
public Interface IToBeSearched
{

public Interface ISearchable
{

    public void Find(IToBeSearched a);

}
```

```

Class Needle Implements IToBeSearched
{}

Class Haystack Implements ISearchable
{

    public void Find(IToBeSearched needle)

    {

        //Here goes the original coding of find
        function

    }

}

```

Share Improve this answer

answered [Nov 13, 2008 at 16:19](#)

Follow

community wiki
[Varun Mahajan](#)



0



```

haystack.iterator.findFirst(/* pass here a
predicate returning
                                true if its
argument is a needle that we want */)

```

`iterator` can be interface to whatever immutable collection, with collections having common

`findFirst(fun: T => Boolean)` method doing the job. As long as the haystack is immutable, no need to hide any useful data from "outside". And, of course, it's not good to tie together implementation of a custom non-trivial

collection and some other stuff that does have `haystack`.
Divide and conquer, okay?

Share Improve this answer

answered Aug 13, 2012 at 0:42

Follow

community wiki
[Display Name](#)



0



In most cases I prefer to be able to perform simple helper operations like this on the core object, but depending on the language, the object in question may not have a sufficient or sensible method available.



Even in languages like [JavaScript](#)) that allow you to augment/extend built-in objects, I find it can be both convenient and problematic (e.g. if a future version of the language introduces a more efficient method that gets overridden by a custom one).

[This article](#) does a good job of outlining such scenarios.

Share Improve this answer

answered Jan 4, 2018 at 21:18

Follow

community wiki
[blizzrdof77](#)



0



So many answers, but many of them seem to assume a lot about the circumstances in which the search happens (or entirely ignore them, which is the same as assuming they were always the same)

This is not (much) based on "pure" theory and more on practical experience:

It depends on what the searching/finding algorithm is specific to

A class should "know" about the things specific to itself. This means that if searching a haystack requires a method that is specific to haystacks, *and especially if that method can be adapted to suit a particular instance of a haystack better*, then the search method should be an attribute of the haystack:

```
haystack.searchthishaystack(needle)
# Gets a description of what to look for and knows best
# iterate through this particular haystack
```

If your haystacks are completely straightforward to search for any algorithm who knows nothing about haystacks, but your needles are hard to detect, then of course it's the needle which should know how to figure out where it is:

```
needle.findyourself_in(haystack)
# unnecessary if it's really just a large number or
# to be tested for identity with the `needle` object
# But maybe needles affect some of the haystack attributes
# is some metaphorical magnet that can help traverse
```

```
# quicker, then the needle (or its `metal_objects`  
# where it's defined.
```

The previous two could also happen in combined form, in which case the search algorithm would have to be decomposed:

```
haystack.search(needle, detector=needle.detector)  
# The haystack contains a haystack-specific search  
# needle provides a needle-specific "detector" whi  
# about whether the search is moving closer or fur  
# which distance it is going to snap to the magnet  
# should help speed things up.
```

Then there could also be a case where you have multiple objects of the same class but with different attributes, and you are trying to find other needles which are "matching" the reference needle in a particular way. Then the `needle` object should contain the method to check whether some other needle is a match or not.

```
haystack.search(tester=needle.match_test)  
# this is very similar to the `needle.detector` ca  
# the task of checking whether a particular object  
# criteria or not
```

So when is it useful to have a `searcher` object? ... obviously, if there is *something specific* about how I want to search things in my code! Like, maybe we need to limit searching activities to avoid

```
specific_searcher = searchlib.define_searcher(how_  
specific_searcher.search(where=haystack, what=need
```

```
...
specific_searcher.search(where=szechwan, what=good)
# now the searcher object contains specific information
# something that is relevant in my code, and that
```

If there is nothing specific or special about searching haystacks or detecting/identifying needles, then it's not really necessary to create an instance-specific method for anything. In such situations, I'm not a fan of turning verbs into nouns for the sole purpose of telling the noun to verb:

```
searcher = searchlib.define_searcher()
searcher.search(where=haystack, what=needle)
# Don't think this makes anything easier to read/t
# worse: If I see `searcher.search()`, I need to f
# difference between this specific `searcher` obje
# and any old generic search before I know what it
# (Spoiler: there is no difference, but you can't
```

...instead, the straightforward way is to simply state a command and arguments:

```
searchlib.search(where=haystack, what=needle)
# plain, generic search for a generic object with
# assumes that `haystack` is some generic iterable
# that are compared to `needle`
# might be slower than a magnet but if all I have
# this is the straightforward way.
```


Share Improve this answer

answered [Apr 12 at 8:36](#)

Follow

community wiki

[Zak](#)
