http://www.yegor256.com/2015/03/09/objects-end-with-er.html

Don't Create Objects That End With -ER

9 March 2015 modified on 23 May 2015 Yegor Bugayenko

Manager. Controller. Helper. Handler. Writer. Reader. Converter. Validator. Router. Dispatcher. Observer. Listener. Sorter. Encoder. Decoder. This is the class names **hall of shame**. Have you seen them in your code? In open source libraries you're using? In pattern books? They are all wrong. What do they have in common? They all end in "-er". And what's wrong with that? They are not classes, and the objects they instantiate are not objects. Instead, they are collections of procedures pretending to be classes.

© Fight Club (1999) by David Fincher

Peter Coad used to say: Challenge any class name that ends in "-er". There are a few good articles about this subject, including Your Coding Conventions Are Hurting You by Carlo Pescio, One of the Best Bits of Programming Advice I Ever Got by Travis Griggs, and Naming Objects – Don't Use ER in Your Object Names by Ben Hall. The main argument against this "-er" suffix is that "when you need a manager, it's often a sign that the managed are just plain old data structures and that the manager is the smart procedure doing the real work".

I totally agree but would like to add a few words to this.

I mentioned already in Seven Virtues of a Good Object that a good object name is not a job title, but I didn't explain why I think so. Besides that, in Utility Classes Have Nothing to Do With Functional Programming, I tried to explain the difference between declarative and imperative programming paradigms. Now it's time to put these two pieces together.

Let's say I'm an object and you're my client. You give me a bucket of apples and ask me to sort them by size. If I'm living in the world of imperative programming, you will get them sorted immediately, and we will never interact again. I will do my job just as requested, without even thinking *why* you need them sorted. I would be a sorter who doesn't really care about your real intention:

```
List<Apple> sorted = new Sorter().sort(apples);
Apple biggest = sorted.get(0);
```

As you see here, the real intention is to find the biggest apple in the bucket.

This is not what you would expect from a good business partner who can help you work with a bucket of apples.

Instead, if I lived in the world of declarative programming, I would tell you: "*Consider them sorted; what do you want to do next?*". You, in turn, would tell me that you need the biggest apple now. And I would say, "*No problem; here it is*". In order to return the biggest one, I would not sort them all. I would just go through them all one by one and select the biggest. This operation is much faster than sorting first and then selecting the first in the list.

In other words, I would silently **not** follow your instructions but would try to do my business my way. I would be a much smarter partner of yours than that imperative sorter. And I would become a real object that behaves like a sorted list of apples instead of a procedure that sorts:

```
List<Apple> sorted = new Sorted(apples);
Apple biggest = sorted.get(0);
```

See the difference?

Pay special attention to the difference between the `sorter` and `sorted` names.

Let's get back to class names. When you add the "-er" suffix to your class name, you're immediately turning it into a dumb imperative executor of *your* will. You do not allow it to think and improvise. You expect it to do exactly what you want — sort, manage, control, print, write, combine, concatenate, etc.

An object is a living organism that doesn't want to be told what to do. It wants to be an equal partner with other objects, exposing behavior according to its contract(s), a.k.a. interfaces in Java and C# or protocols in Swift.

Philosophically speaking, the "-er" suffix is a sign of disrespect toward the poor object.