Design Patterns and Anti-Patterns, Love and Hate

3 February 2016 Palo Alto, CA Yegor Bugayenko

Design Patterns are ... Come on, you know what they are. They are something we **love** and **hate**. We love them because they let us write code without thinking. We hate them when we see the code of someone who is used to writing code without thinking. Am I wrong? Now, let me try to go through all of them and show you how much I love or hate each one. Follow me, in alphabetic order.

© The Shining (1980) by Stanley Kubrick

Abstract Factory. It's OK.

Adapter. Good one!

Bridge. Good one!

Builder. Terrible concept, since it encourages us to create and use big, complex objects. If you need a builder, there is already something wrong in your code. Refactor it so any object is easy to create through its constructors.

Chain of Responsibility. Seems fine.

Command. It's OK.

Composite. Good one; check out this too.

Data Transfer Object. It's just a shame.

Decorator. My favorite one. I highly recommend you use it.

Facade. Bad idea. In OOP, we need objects and only objects, not facades for them. This design pattern is very procedural in its spirit, since a facade is nothing more than a collection of procedures.

Factory Method. This one seems OK.

Flyweight. It's a workaround, as I see it, so it's not a good design pattern. I would recommend you not use it unless there is a really critical performance issue. But calling it a design pattern ... no way. A fix for a performance problem in Java? Yes.

Front Controller. Terrible idea, as well as the entire MVC. It's very procedural, that's why.

Interpreter. It's OK, but I don't like the name. "Expression" would be a much better alternative.

Iterator. Bad idea, since it is mutable. It would be much better to have immutable "cursors."

Lazy Initialization. It's OK.

Marker. It's a terrible idea, along with reflection and type casting.

MVC. Bad idea, since it's very procedural. Controllers are the key broken element in this concept. We need real objects, not procedural controllers.

Mediator. I don't like it. Even though it sounds like a technique for decreasing complexity and coupling, it is not really object-oriented. Who is this mediator? Just a "channel" between objects? Why shouldn't objects communicate directly? Because they are too complex? Make them smaller and simpler, rather than inventing these mediators.

Memento. This idea implies that objects are mutable, which I'm against in general.

Module. If Wikipedia is right about this pattern, it's something even more terrible than the Singleton.

Multiton. Really bad idea. Same as Singleton.

Null Object. Good one. By the way, see Why NULL Is Bad

Object Pool. Good one.

Observer. The idea is good, but the name is bad, since it ends with -ER. A much better one would be "Source" and "Target." The Source generates events and the Target listens to them.

ORM. It's terrible and "offensive"; check this out.

Prototype. Good idea, but what does it have to do with OOP?

Proxy. Good one.

RAII. This is a really good one, and I highly recommend you use it.

Servant. A very bad idea, because it's highly procedural.

Singleton. It's the king of all anti-patterns. Stay away from it at all costs.

Specification. It's OK.

State. Although it's not implied, I feel that in most cases the use of this pattern results in mutability, a code characteristic that I'm generally against.

Strategy. A good one.

Template Method. is wrong, since implementation inheritance is procedural.

Visitor. A rather procedural concept that treats objects as data structures, which we can manipulate.

I have nothing against concurrency patterns either; they are all good, since they have almost nothing to do with object-oriented programming.

If you know some other design (anti-)patterns, let me know in the comments below. I'll add them here.