

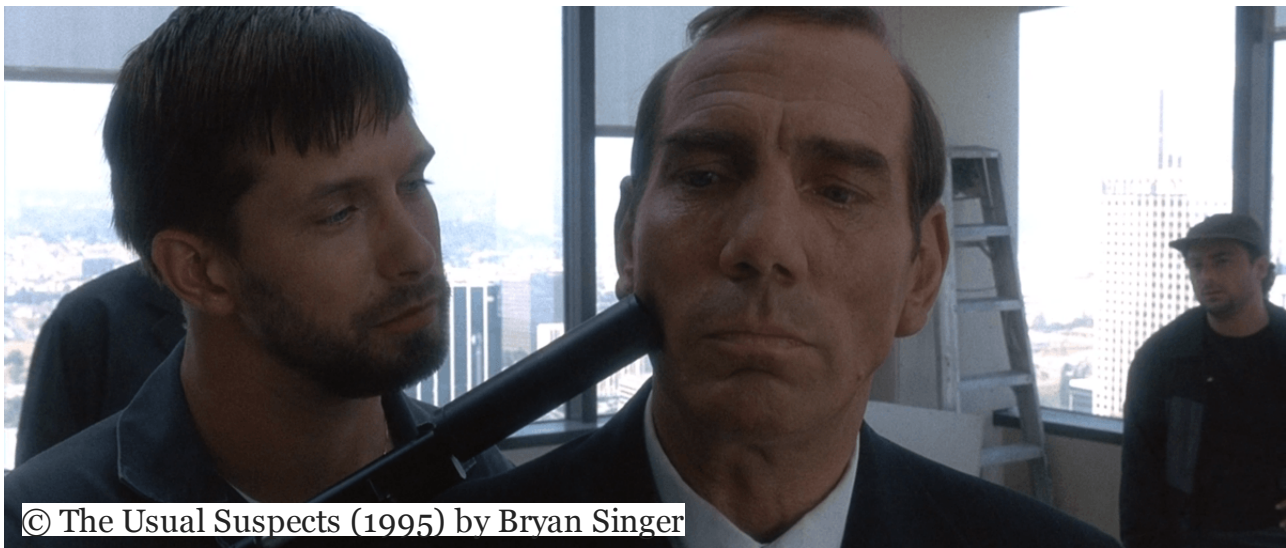
<http://www.yegor256.com/2014/12/22/immutable-objects-not-dumb.html>

Immutable Objects Are Not Dumb

22 December 2014 modified on 6 May 2015 Yegor Bugayenko

After a few recent posts about immutability, including "Objects Should Be Immutable" and "How an Immutable Object Can Have State and Behavior?", I was surprised by the number of comments saying that I badly misunderstood the idea. Most of those comments stated that an immutable object must always behave the same way — that is what immutability is about. What kind of immutability is it, if a method returns different results each time we call it? This is not how well-known immutable classes behave. Take, for example, `String`, `BigInteger`, `Locale`, `URI`, `URL`, `Inet4Address`, `UUID`, or wrapper classes for primitives, like `Double` and `Integer`. Other comments argued against the very definition of an immutable object as a representative of a mutable real-world entity. How could an immutable object represent a mutable entity? Huh?

I'm very surprised. This post is going to clarify the definition of an immutable object. First, here is a quick answer. How can an immutable object represent a mutable entity? Look at an immutable class, `File`[↗], and its methods, for example `length()` and `delete()`. The class is immutable, according to Oracle documentation, and its methods may return different values each time we call them. An object of class `File`, being perfectly immutable, represents a mutable real-world entity, a file on disk.



In [this post](#), I said that "an object is immutable if its state can't be modified after it is created." This definition is not mine; it's taken from [Java Concurrency in Practice by Goetz et al.](#)[↗], Section 3.4 (by the way, I highly recommend you read it). Now look at this class (I'm using [jcabi-http](#)[↗] to read and write over HTTP):

```
@Immutable
class Page {
    private final URI uri;
    Page(URI addr) {
        this.uri = addr;
    }
    public String load() {
        return new JdkRequest(this.uri)
            .fetch().body();
    }
    public void save(String content) {
        new JdkRequest(this.uri)
            .method("PUT")
            .body().set(content).back()
            .fetch();
    }
}
```

What is the "state" in this class? That's right, `this.uri` is the state. It uniquely identifies every object of this class, and it is not modifiable. Thus, the class makes only immutable objects. And each object represents a

mutable entity of the real world, a web page with a URI.

There is no contradiction in this situation. The class is perfectly immutable, while the web page it represents is mutable.

Why do most programmers I have talked to believe that if an underlying entity is mutable, an object is mutable too? I think the answer is simple — they think that objects are data structures with methods. That's why, from this point of view, an immutable object is a data structure that never changes.

This is where the fallacy is coming from — an object is **not a data structure**. It is a living organism representing a real-world entity inside the object's living environment (a computer program). It does encapsulate some data, which helps to locate the entity in the real world. The encapsulated data is the **coordinates** of the entity being represented. In the case of `String` or `URL`, the coordinates are the same as the entity itself, but this is just an isolated incident, not a generic rule.

An immutable object is not a data structure that doesn't change, even though `String`, `BigInteger`, and `URL` look like one. An object is immutable if and only if it doesn't change the coordinates of the real-world entity it represents. In the `Page` class above, this means that an object of the class, once instantiated, will never change `this.uri`. It will always point to the same web page, no matter what.

And the object doesn't guarantee anything about the behavior of that web page. The page is a dynamic creature of a real world, living its own life. Our object can't promise anything about the page. The only thing it promises is that it will always stay loyal to that page — it will never forget or change its coordinates.

Conceptually speaking, immutability means loyalty, that's all.

