In the end, it all comes down to 0 and 1

Vineet Goel

But a good developer should care about other software developers and not about machines. Since we, software developers, are used to be very lazy it is a good idea to get acquainted with design patterns.

Wait, what?! New information again (Figure 1)?

Figure 1: Whaaaaaat?

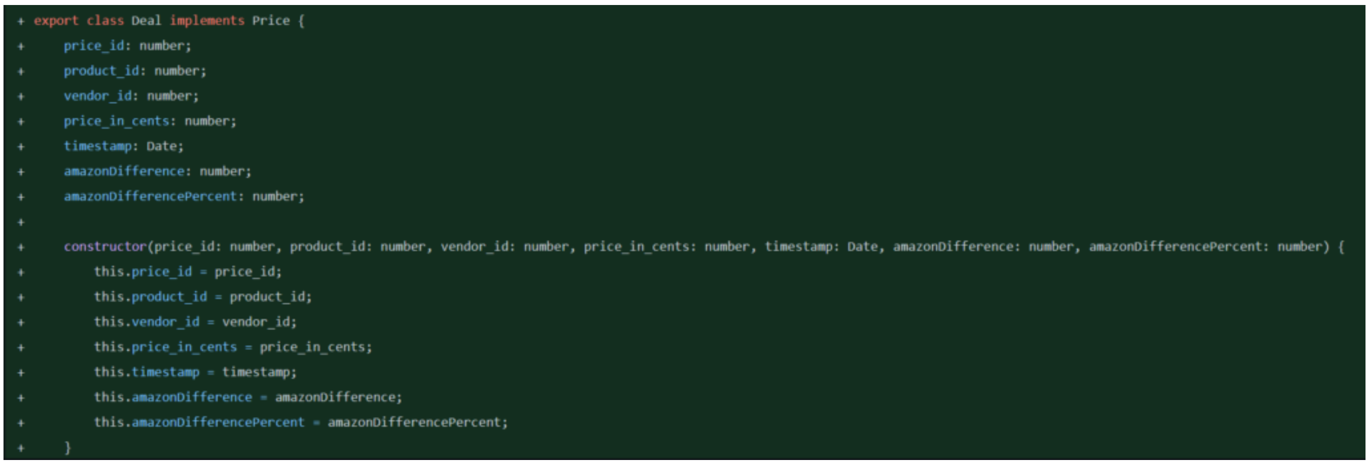
Of course! Because stagnation is self-abdication. In software engineering, a **design pattern** is a general repeatable solution to a commonly occurring problem in software design. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations [1].

But what are these design patterns good for?

Well, here is what you've got after investing a little bit of your time in researching. Design patterns can speed up the development process by providing tested, proven development paradigms. Effective software design requires considering issues that may not become visible until later in the implementation. Reusing design patterns helps to prevent subtle issues that can cause major problems and improves code readability for coders and architects familiar with the patterns [1].

So what guys did you do?

This time we decided to implement the so called *Proxy Pattern*. It is a structural design pattern that lets you provide a substitute or placeholder for another object. A proxy controls access to the original object, allowing you to perform something either before or after the request gets through to the original object. If you would like to implement this pattern in your own project you can check this blog post. Here is also a link to the commit in which we made our changes. There you can check out how we implemented this pattern. The biggest change is that we “extend” the Price interface to get a Deal interface, which has two additional fields that we use on deals only (Have a look at it in the screenshot below). We use this design pattern as it makes the structure of the code more clear, so it’s easier for new developers to understand the existing code (or for us in 3 months :D)



```
+ export class Deal implements Price {
+     price_id: number;
+     product_id: number;
+     vendor_id: number;
+     price_in_cents: number;
+     timestamp: Date;
+     amazonDifference: number;
+     amazonDifferencePercent: number;
+
+     constructor(price_id: number, product_id: number, vendor_id: number, price_in_cents: number, timestamp: Date, amazonDifference: number, amazonDifferencePercent: number) {
+         this.price_id = price_id;
+         this.product_id = product_id;
+         this.vendor_id = vendor_id;
+         this.price_in_cents = price_in_cents;
+         this.timestamp = timestamp;
+         this.amazonDifference = amazonDifference;
+         this.amazonDifferencePercent = amazonDifferencePercent;
+     }
```

The newly created Deal class

As we already use frameworks like Angular and Express that already contain design patterns like e.g. observables, we didn’t find too many other design patterns that we can implement in a useful manner. And because we use TypeScript in both the front- and backend, showing you a class diagram of our code here wouldn’t make too much sense as well, as it would show you 0 classes before and 1 class after the change