This structure can give you some advantages:

- **Encapsulation:** Everything associated with a state is in one place.
- **Separation of concern:** Each state has its class, so you have an excellent separation.
- **More simple logic:** There is no need for nasty if-else or switch statements around a string attribute.

However, it can be a **significant overhead** if you have only a few states in your model and it's only used to check some basic behavior.

Moving on, we need to change the state from Pending to Paid at some point. I think we all wrote code similar to this in the past:

```
class OrderController extends Controller
{
  public function pay(PayOrderRequest $request, Order $order)
  {
    // Some logic here ...
    $order->status = 'paid';
    $order->save();
  }
}
```

Domain-Driven Design teaches us the following: we have to treat these transitions as first-class citizens. So let's put them in dedicated classes! First, we can create some kind of abstraction. In this case, we don't need a class, only an interface: