Notas sobre o capitulo 6 de Clean Code – by Lucas Trevizan

Objetos e estrutura de dados

Por que adicionamos tantos getters e setters se a maioria das nossas variáveis/atribtuos são privados?

Abstração de dados

No livro temos um exemplo de uma Interface comparada a uma implementação concreta e como uma Interface da mais sentido, ela nos dá regras de manipulação que um objeto concreto **PODE OU NÃO USAR**. Enquanto a implementação concreta nos obriga a manipular o objeto inteiro.

Ocultar a implementação nos permite expor interfaces que permitem ao usuário alterar a essência dos dados sem precisar conhecer a implementação concreta. Outra forma interessante de se pensar nesse conceito é imaginar a classe concreta como **OS DADOS QUE REPRESENTAM O OBJETO** e as interfaces como **O QUE EU PRECISO EXTRAIR DOS DADOS DO OBJETO,** usar getters e setters não é a melhor opção.

Anti-simetria data/objeto

Aqui rola um bom exemplo sobre como código procedural é diferente de código Orientado a objeto, eles são assimétricos. Existem situações onde um é preferível a outro, mas geralmente o que é difícil pra OO é fácil para procedural e o inverso também.

Aqui tem algo interessante que é ressaltado, o fato de que programação 100% orientada a objetos é um mito, por experiência própria eu sei que isso é bem verdade.

A lei de Demeter

A lei de Demeter basicamente fala para que os módulos que operam um objeto não conheçam seu interior. Da pra usar um exemplo prático com isso usando métodos "linkados".

Por exemplo: digamos que eu tenho um **objeto PESSOA**, dessa pessoa eu tenho um atributo de **CASA (outro objeto)** dessa casa eu tenho um atributo **ENDEREÇO (um objeto de Casa)** e nesse Endereço eu tenho um atributo cep**.**

Se eu quisesse obter o cep, mas tenho apenas A **PESSOA** em meu módulo, eu faria algo como: ***Pessoa.getCasa().getEndereco().getCep();*** isso quebra a lei de Demeter porque eu estou acessando muitos subníveis do meu objeto em questão (**Pessoa**) para obter o CEP, eu estou chamando métodos em objetos retornados consecutivamente.

Carrinhos de trem

No livro ele diz que esse tipo de código é chamado de carrinho de trem justamente porque cada chamada de método parece um vagão acoplado em outro.

Temos um exemplo de instanciar cada um desses objeto e fazer suas devidas chamadas, mas isso entra em outro questionamento ***"Estamos violando a lei de Demeter?",*** e isso entra na dependência de essas informações serem estruturas de dados ou objetos. No nosso caso são objetos e sim estamos quebrando a lei.

Híbridos

Híbridos são aberrações da natureza de código, aqui você faz o pior de dois mundos, sem conseguir se aproveitar dos benefícios do OO nem do procedural.

Quando misturamos códigos procedurais que precisam acessar atributos privados de objetos estamos fazendo uma má prática e isso demonstra que não existia conhecimento sólido o suficiente para saber se os atributos de X objeto precisavam ou não ser privados.

Estruturas ocultas

Aqui é enfatizado aquele lance de "não fale com estranhos", se eu estou usando um objeto então ele **DEVE FAZER ALGO** e não saber como funciona sua estrutura interna.

Nesse ponto tem um exemplo bom de refatoração, onde iriamos até a raiz dessa classe e descobriríamos pra que serve o ***"ligamento" do objeto X com o objeto Y.***

Sabendo pra que serve esse "ligamento" deveríamos então refatorar pra que o **objeto X FAÇA AQUILO** e não me exponha esses dados do objeto **Y.**

Usando esse pensamento em relação ao exemplo de Pessoa e cep, eu deveria poder fazer no meu módulo que conhece somente Pessoa algo como isso: **Pessoa.buscarCep()** ***(toda a estrutura para se chegar nisso ficaria responsável pelos objetos que se conhecem Pessoa -> Casa -> Endereco.cep e escondidas do meu módulo)***

Objetos de transferência de dados

Objetos de transferência de dados são classes com atributos públicos e nenhum método (***DTO, TO siglas comuns aí pra falar nesse padrão).*** A pira desses DTO's é que eles servem para se comunicar com alguma coisa (***banco de dados, mensagens, etc...).***

Geralmente eles são os primeiros caras em uma série de comunicações que convertem dados brutos do banco de dados em objetos de código da aplicação.

O Active Record

Active Records são formas especiais de DTO, eles possuem métodos como ***salvar()*** ou ***buscar()*** , dá pra ver que a ideia aqui é tratar eles como uma ponte entre o banco de dados e a aplicação com mais liberdade do que apenas trafegar dados.

Algo muito importante a se salientar nesse padrão é que **AS REGRAS DE NEGÓCIO NÃO DEVEM FICAR NO DTO**, isso cria híbridos. Se você for usar Active Record, tenha certeza de que ao salvar o DTO no banco você está realmente APENAS O SALVANDO, **sem aplicar regras a esse processo**, as regras devem estar em objetos separados.

Conclusão

Objetos expõem ações e ocultam dados, estruturas de dados expõem dados e não tem ações significativas. Cada uma tem sua vantagem e desvantagem e um bom programador escolhe sem preconceito a que melhor se encaixa no momento.