05

Lei de Demeter, ou o Princípio do menor conhecimento

Vamos analisar um pouco a classe `Leilao`:

```
class Leilao:

    def __init__(self, descricao):
        self.descricao = descricao
        self.__lances = []
        self.maior_lance = sys.float_info.min
        self.menor_lance = sys.float_info.max

       # restante do código
```

Dentro dessa classe, existe uma lista de lances. Ou seja, temos uma composição com uma lista. O que esperamos que ocorra com essa lista?

Que seja adicionado os lances pertencente ao leilão. Adicionar diretamente a lista, algo como: `leilao.lances.append(lance)` é uma má prática, já que estamos muito acoplados com a implementação da classe `Leilao`.

Além de ferir o princípio do Diga, não pergunte (o *Tell don't ask*), estamos ferindo outro princípio chamado **Lei de Demeter** ou o **Princípio do menor conhecimento** (*Principle of least knowledge*).

Esse princípio diz que devemos ter o menor conhecimento sobre a implementação da classe. Dessa forma, evitamos o acoplamento entre as classes do sistema.

Claro, sempre existirá o acoplamento. Quando estamos utilizando uma lista, string, dicionário, estamos acoplados com essa classe. Porém, esse acoplamento é chamado de acoplamento bom. A chance de uma dessas classes mudarem e afetarem o código são muito pequenas.

> Acoplar com classes estáveis é muito melhor do que se acoplar com classes instáveis

Então só devemos nos acoplar com classes do sistema?

Não! O código que escrevemos existem classes mais estáveis do que outras. São essas classes estáveis que devemos optar por acoplar.

Seguindo esses princípios como a Lei de Demeter e o Diga, não pergunte, escrevemos um código que é mais simples alterar e escalar.

Quem quiser ver um pouco mais sobre a Lei de Demeter, deixo esse testo, em inglês, da Wikipédia: https://en.wikipedia.org/wiki/Law_of_Demeter (https://en.wikipedia.org/wiki/Law_of_Demeter)