Refatoramento de Código

Exemplo extraído do livro *Refactoring: Improving the Design of Existing Code* (Fowler, 1999)

Parte 5: Seja bem-vindo, Polimorfismo!

I. Criando uma interface de itens "Alugáveis"

Até o momento, temos uma classe **`Automovel`** que possui dois métodos que adotam **um comportamento diferente** de acordo com algum atributo do objeto. Isso é indicativo de aplicação de **polimorfismo**! Observe:

a) **`switch`** do método **`valorDeUmaLocacao()`**: temos cálculos diferentes para cada tipo de **`Automovel`**;
b) O teste de **`getPontosDeAlugadorFrequente()`**: temos cálculos diferentes de bonificação de acordo com o tipo do **`Automovel`**.

De fato, **`Automovel`** diferentes poderiam "responder de forma diferente" às duas perguntas sobre o **`valorDeUmaLocacao()`** e **`getPontosDeAlugadorFrequente()`**. Sendo assim, podemos aplicar polimorfismo em cima de tipos de **`Automovel`** para resolver essa questão. Além disso, vamos acrescentar o fato de que o sistema de locadora de veículos está negociando um lote de automóveis elétricos para aumentar sua frota e esses automóveis terão preços de locação diferenciados, como também, regras específicas para pontos de fidelidade. Sendo assim, o sistema tem que estar preparado para receber os veículos com essa característica (sem causar grandes impactos no que já está rodando).

Na prática, o que vamos fazer é isolar "dois mundos":

a) O mundo das coisas **que podem ser alugadas** (Automovel, Motocicletas, Veículos Elétricos);
b) O mundo **que usa** tais coisas.

A solução é usarmos uma **`interface`** para isolar os "dois mundos". Chamaremos a interface de **`Alugavel`**. Por enquanto, serão apenas objetos **`Automovel`** que são alugáveis, mas depois virão **`AutomovelEletrico`**, etc.

Para definir a **`interface`**, perguntamos: "quais métodos cada coisa que vai ser alugada deve implementar?". Para este problema, respondemos esta pergunta com a seguinte lista de métodos abstratos:

```
public String getDescricao()
public String getAno()
public double getValorDaLocacao(int diasAlugada);
public int getPontosDeAlugadorFrequente(int diasAlugada);
```

II. Tarefa (refactoring4)

Crie a interface **`Alugavel`** e faça **`Automovel`** implementar a interface **`Alugavel`**. Na classe **`Locacao`**, ao invés de termos um objeto específico da classe **`Automovel`**, vamos definir uma propriedade item que é do tipo da interface **`Alugavel`**. Ou seja, pode receber qualquer tipo de objeto que seja "alugável". Faça as atualizações necessárias dentro da classe (e dependentes) após a mudança. Realize os ajustes necessários para poder rodar novamente o código. A saída deve ser a mesma e a classe **`Locadora`** não deve ser afetada pelas mudanças.

NOTA: não resolvemos ainda o problema do **`switch`**. Ainda vamos criar objetos que representem cada tipo de classificação de um **`Automovel`**.