Decorator

Prof. Luiz Merschmann

Padrão de Projeto *Decorator*

Descrição do problema

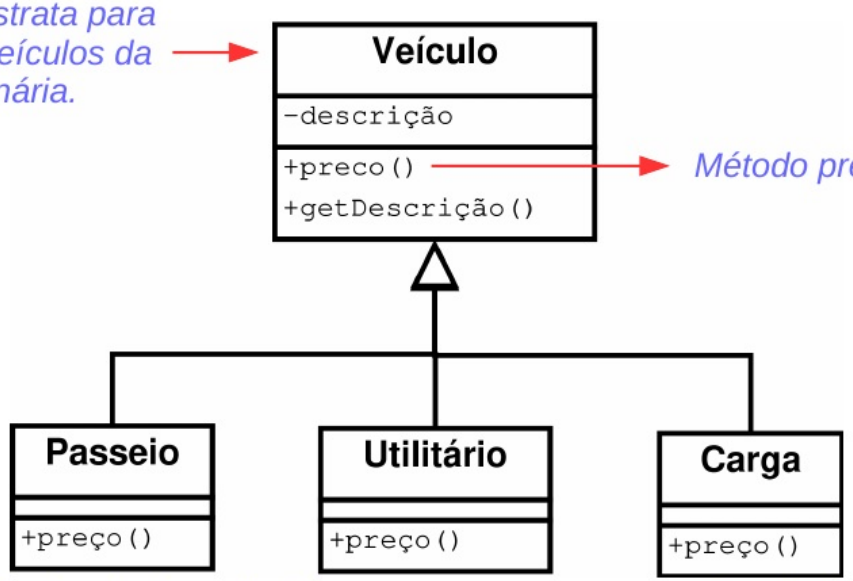
- Ele anexa responsabilidades adicionais a um objeto dinamicamente. Os decoradores fornecem uma alternativa flexível de subclasse para estender a funcionalidade.

Padrão de Projeto *Decorator*

Uma concessionária resolveu alterar o seu negócio de modo a permitir que os clientes possam "montar" seus veículos da forma que bem entenderem.

O sistema que lá funcionava antes dessa alteração foi projetado como apresentado na modelagem a seguir:

Classe abstrata para todos os veículos da concessionária.



Método preço é abstrato.

Cada subclasse implementa e retorna o preço do veículo.

Padrão de Projeto *Decorator*

No novo modelo de negócio cada veículo tem um preço base (inicial) que é incrementado de acordo com os acessórios escolhidos pelo cliente.

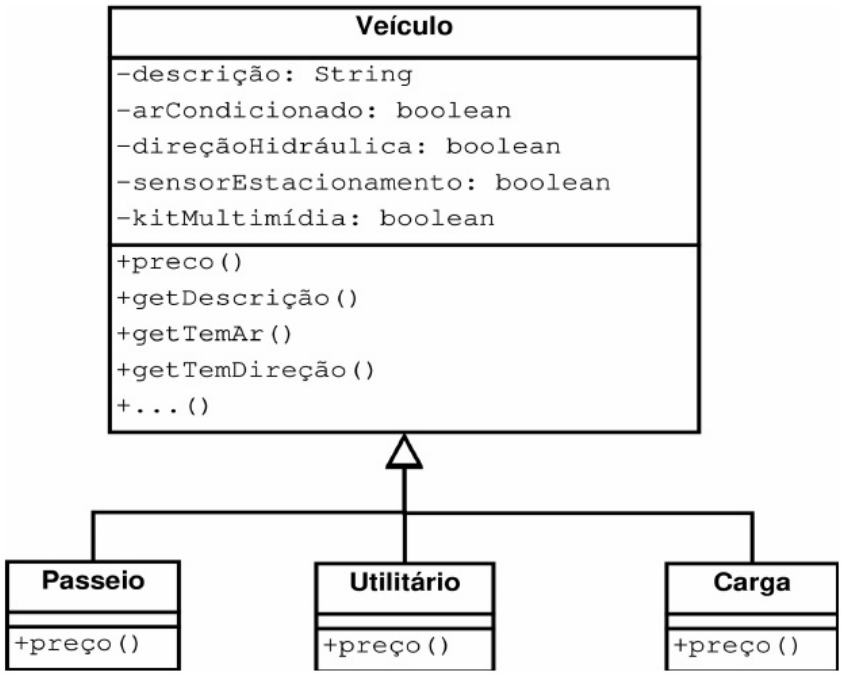
Ou seja, ao escolher **um tipo de veículo** (passeio, utilitário ou carga), o cliente pode **incrementá-lo** adicionando algum(ns) acessório(s):

- Ar condicionado
- Direção hidráulica
- Sensor de estacionamento
- Kit multimídia

E agora? Como ajustar a modelagem anterior para esse novo cenário?

Padrão de Projeto *Decorator*

Então... Que tal acrescentarmos os acessórios na superclasse e tratá-los nas subclasses?



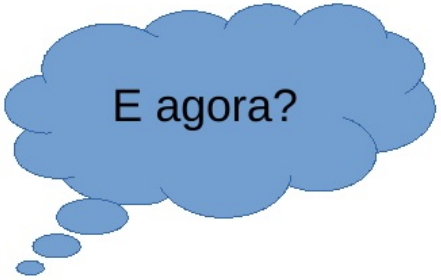
Padrão de Projeto *Decorator*

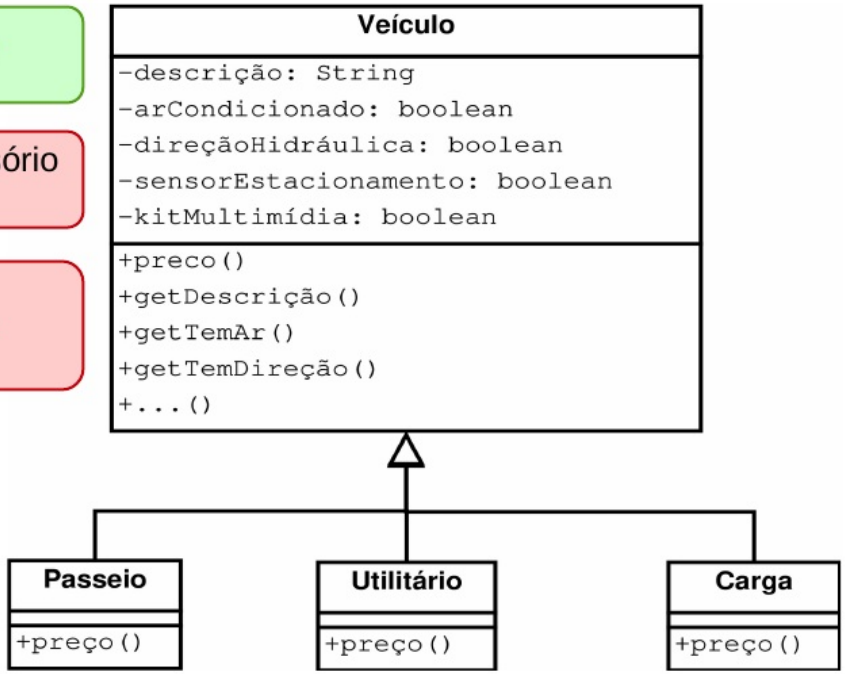
Essa solução é bem melhor do que a primeira! Agora temos novamente apenas 4 classes e conseguimos tratar os acessórios.

Mas essa solução é realmente boa?

Cada vez que inserirmos um novo acessório precisaremos alterar a “classe base”.

Os objetos podem ter atributos desnecessários (para os acessórios que não possuem).





Padrão de Projeto *Decorator*

E se começássemos com um tipo de veículo e fossemos "decorando" o veículo com os acessórios que desejamos em tempo de execução?

Como assim?

Veja um exemplo:

1. Começamos com um objeto da classe Utilitário.

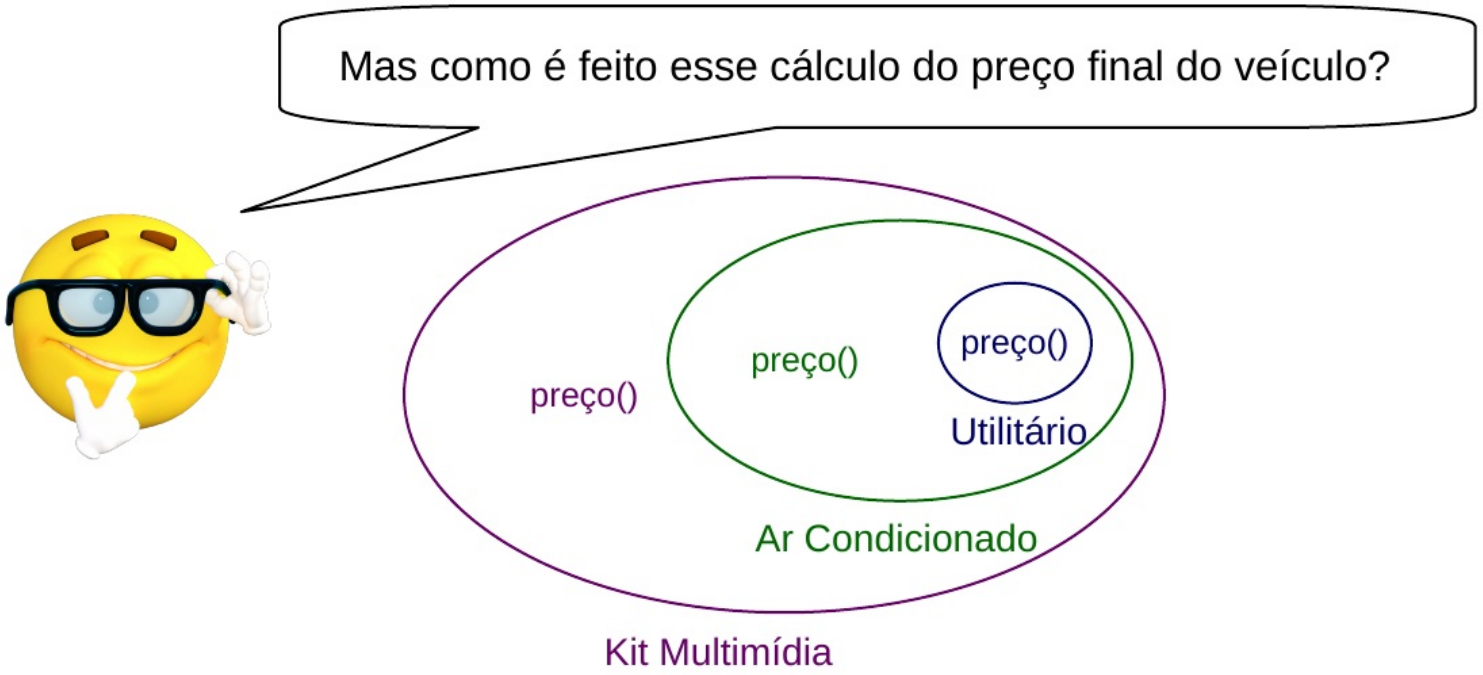
2. Decoramos o objeto anterior com o objeto ArCondicionado.

3. Decoramos o objeto anterior com o objeto KitMultimídia.

4. Chamamos o método preço e teremos o valor do veículo somado com todos os valores dos acessórios.

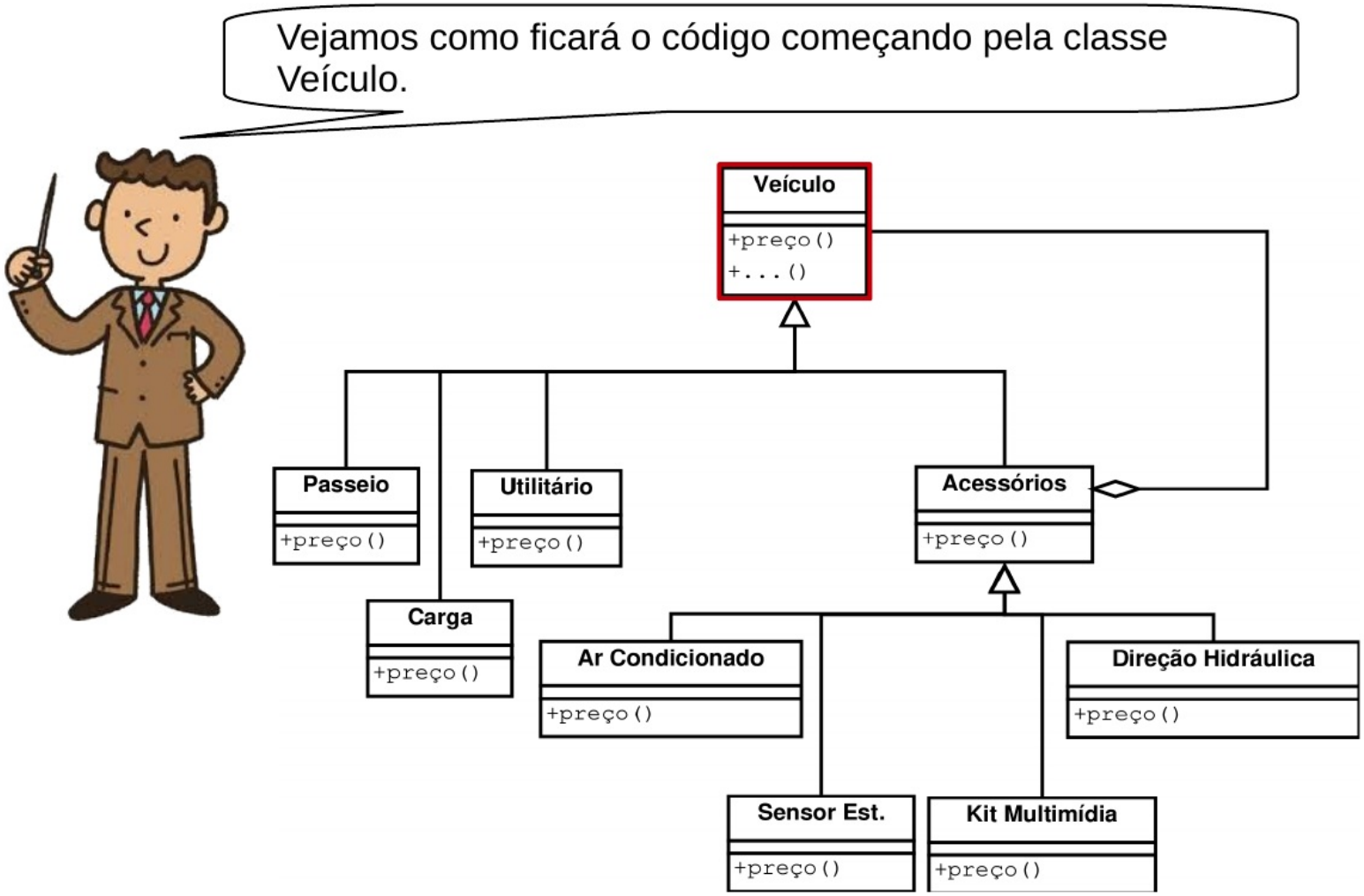
UFLA
UNIVERSIDADE FEDERAL DE LAVRAS

Padrão de Projeto *Decorator*



Cada camada chama o método da camada mais interna e adiciona o retorno com seu próprio preço.

Padrão de Projeto *Decorator*

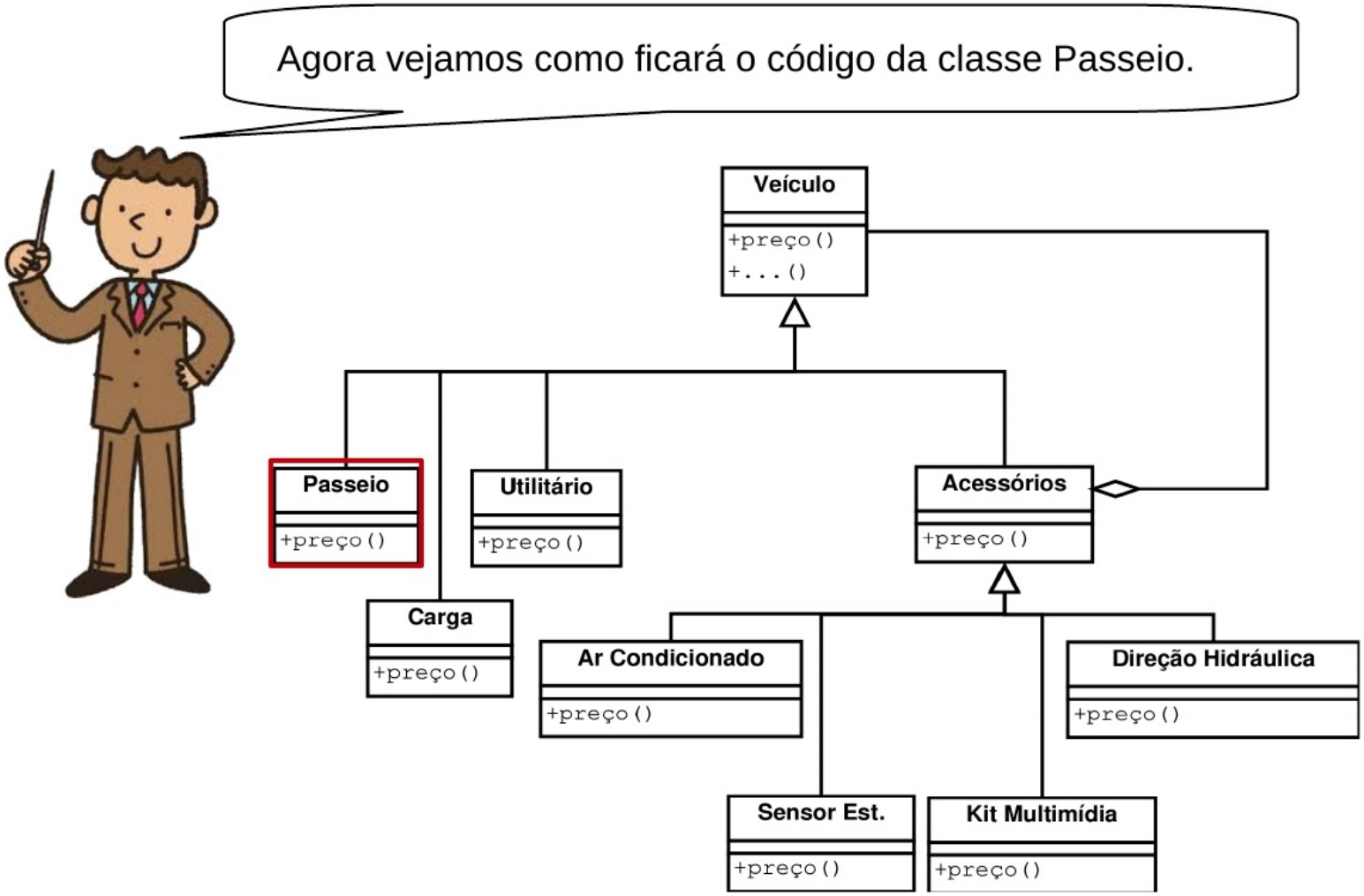


Padrão de Projeto *Decorator*

Vejamos como ficará o código começando pela classe Veículo.

```
public abstract class Veiculo {
    private String descricao;

    public String getDescricao() {
        return descricao;
    }

    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }

    public abstract double preco();

}
```

Padrão de Projeto *Decorator*



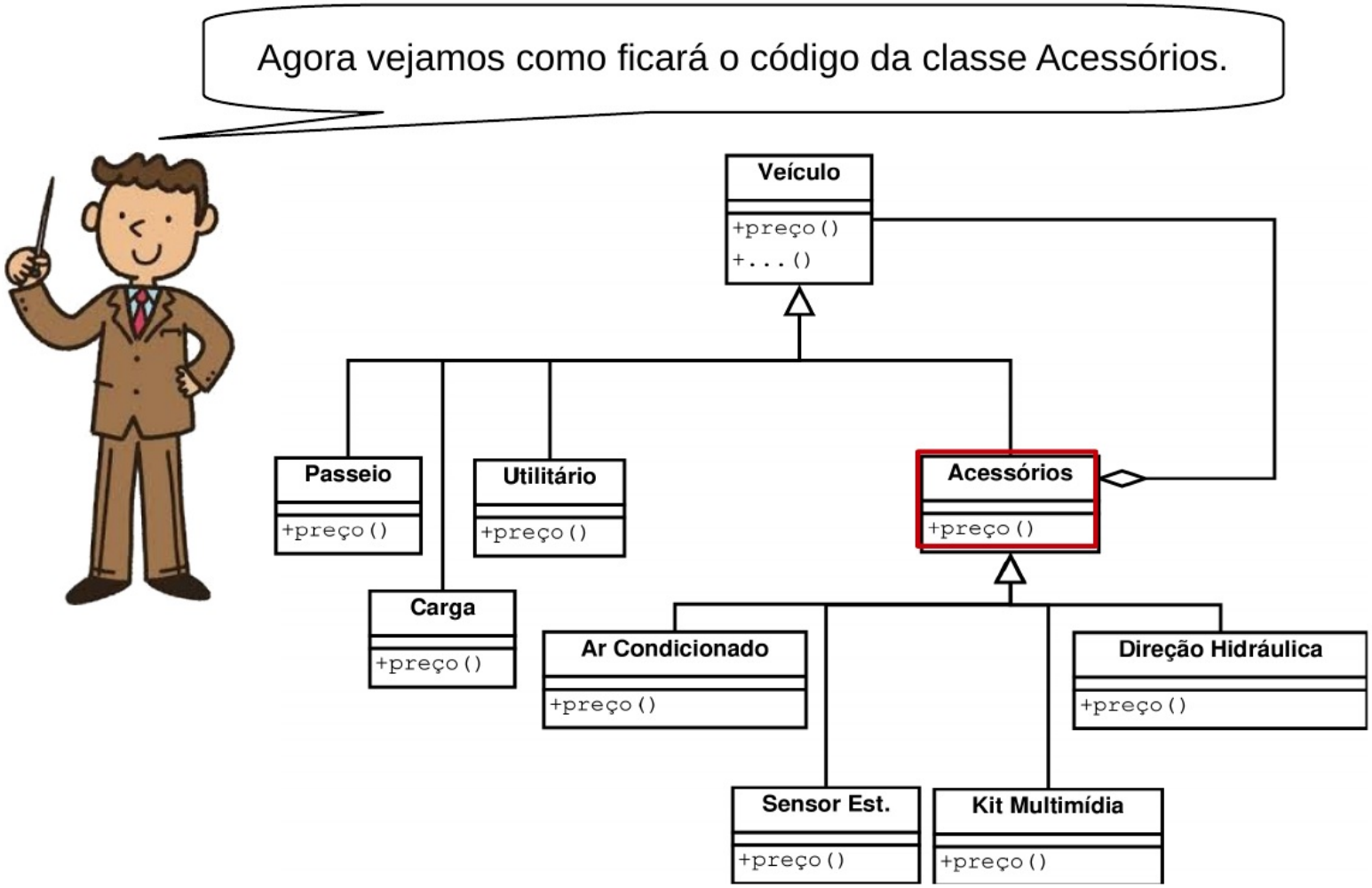
Padrão de Projeto *Decorator*

Agora vejamos como ficará o código da classe Passeio.

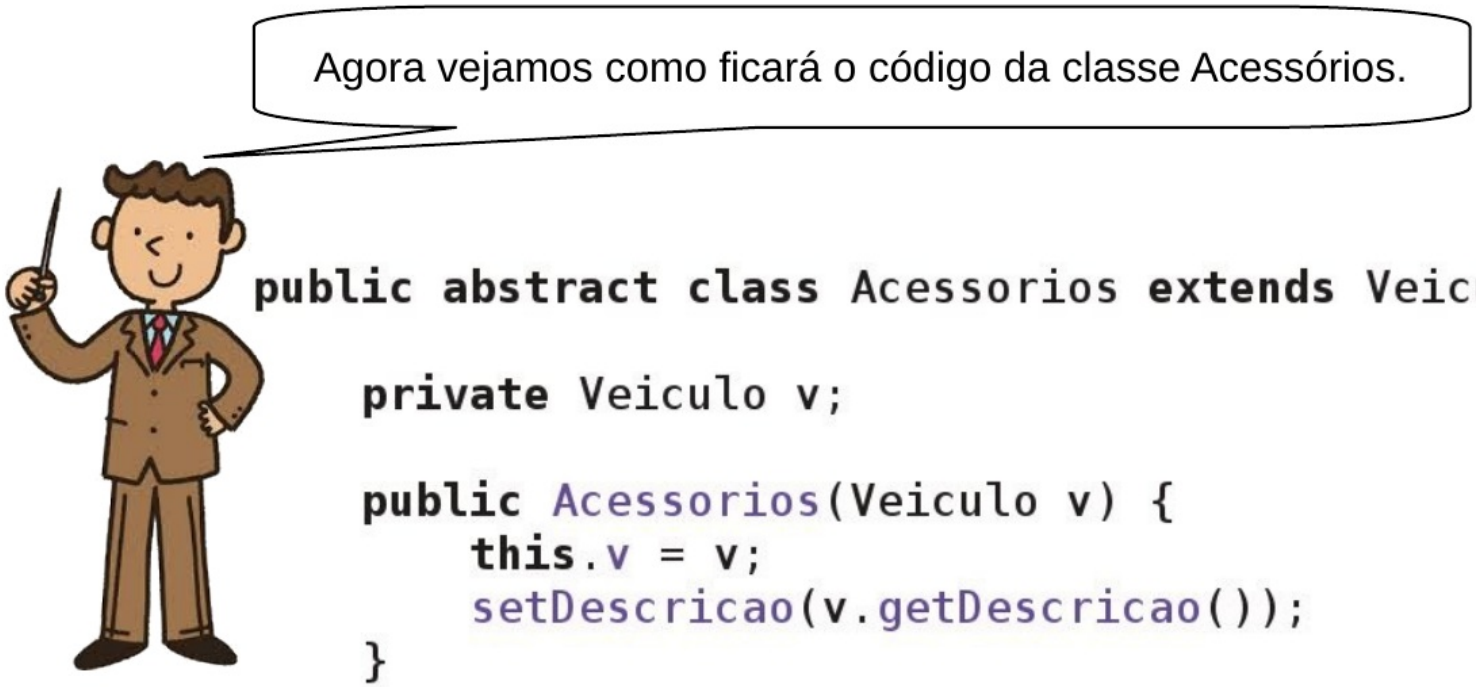
```
public class Passeio extends Veiculo{

    public Passeio(String descricao) {
        setDescricao(descricao);
    }

    @Override
    public double preco() {
        return 30000.0;//Preço de um veículo "base" do tipo passeio
    }

}
```

Os códigos dos demais tipos de veículos (utilitário e carga) são muito semelhantes a este.

Padrão de Projeto *Decorator*

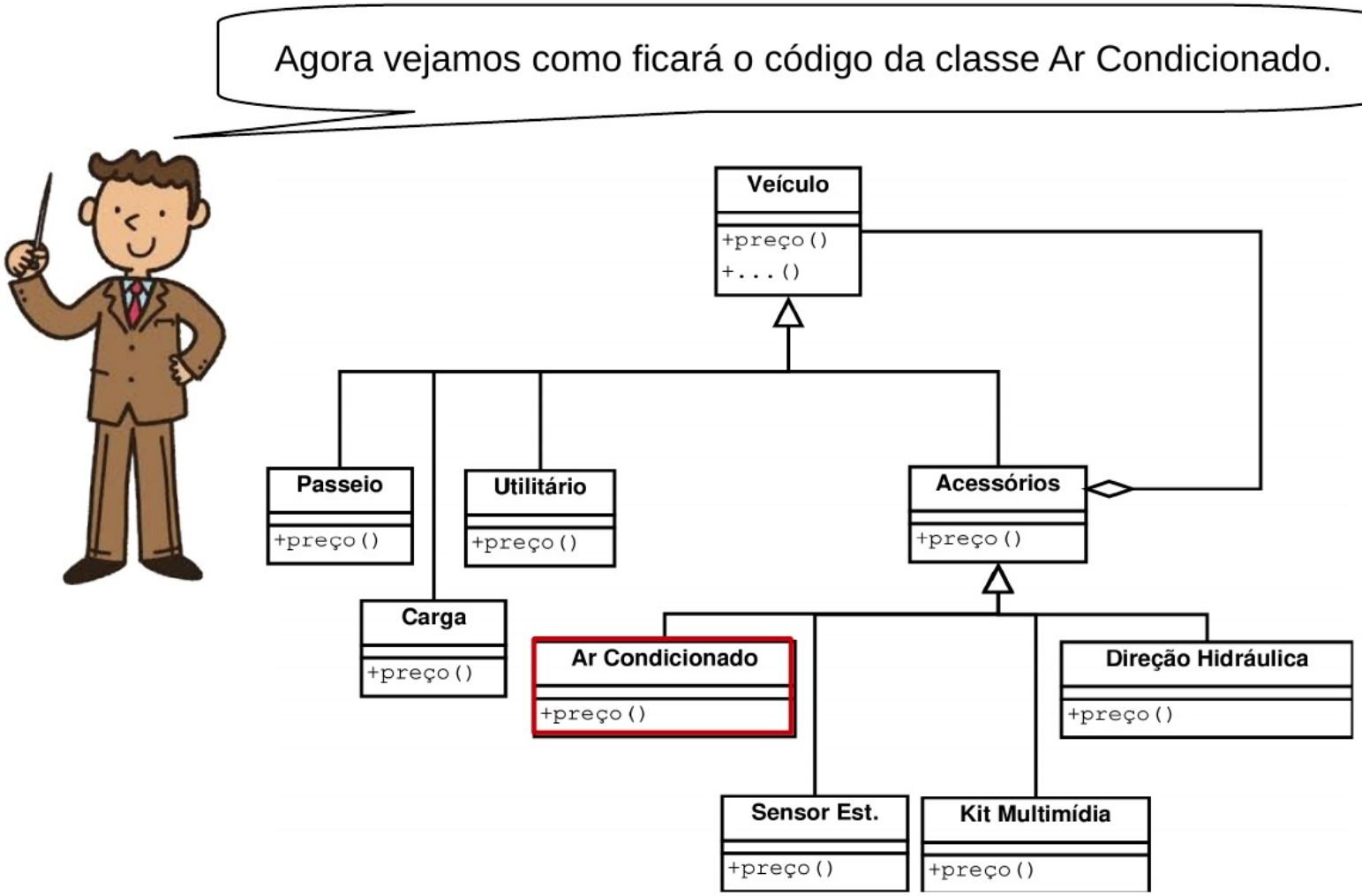


Padrão de Projeto *Decorator*



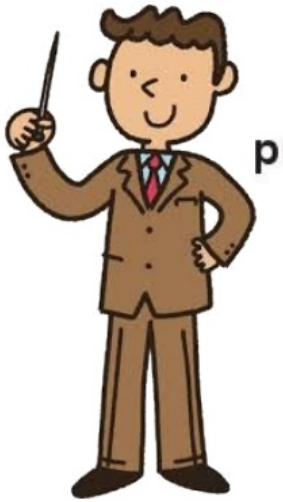

```
public abstract class Acessorios extends Veiculo{

    private Veiculo v;

    public Acessorios(Veiculo v) {
        this.v = v;
        setDescricao(v.getDescricao());
    }

    public Veiculo getVeiculo(){
        return v;
    }
}
```

Padrão de Projeto *Decorator*



Padrão de Projeto *Decorator*

Agora vejamos como ficará o código da classe Ar Condicionado.



```
public class ArCondicionado extends Acessorios{

    public ArCondicionado(Veiculo v) {
        super(v);
    }

    @Override
    public double preco(){
        return 3500 + getVeiculo().preco();
    }
}
```

Os códigos das demais subclasses de **Acessórios** (Direção Hidráulica, Sensor de Estacionamento e Kit Multimídia) são muito semelhantes a este, potencialmente alterando-se apenas o preço do acessório em questão no **método preco**.

Padrão de Projeto *Decorator*

E por fim... Vejamos uma classe contendo o método main a partir da qual iremos decorar um objeto Veículo.

```
public class VeiculoDecorator {

    public static void main(String[] args) {
        Veiculo pas = new Passeio("Carro de passeio");
        gerarRelatorio(pas);
        //Decorando...
        pas = new ArCondicionado(pas);
        gerarRelatorio(pas);
        pas = new DirecaoHidraulica(pas);
        gerarRelatorio(pas);
    }

    public static void gerarRelatorio(Veiculo v){
        System.out.println("Tipo do Veículo: " + v.getDescricao() +
                "\tPreço: " + v.preco());
    }
}
```

SAÍDA:

```
Tipo do Veículo: Carro de passeio       Preço: 30000.0
Tipo do Veículo: Carro de passeio       Preço: 33500.0
Tipo do Veículo: Carro de passeio       Preço: 35500.0
```

Padrão de Projeto *Decorator*

```
Veiculo pas = new Passeio("Carro de passeio");
gerarRelatorio(pas);
//Decorando...
pas = new ArCondicionado(pas);
gerarRelatorio(pas);
pas = new DirecaoHidraulica(pas);
gerarRelatorio(pas);
```