
# Programação Orientada a Objeto e Qualidade de Código

## 03 - Padrões de Projeto

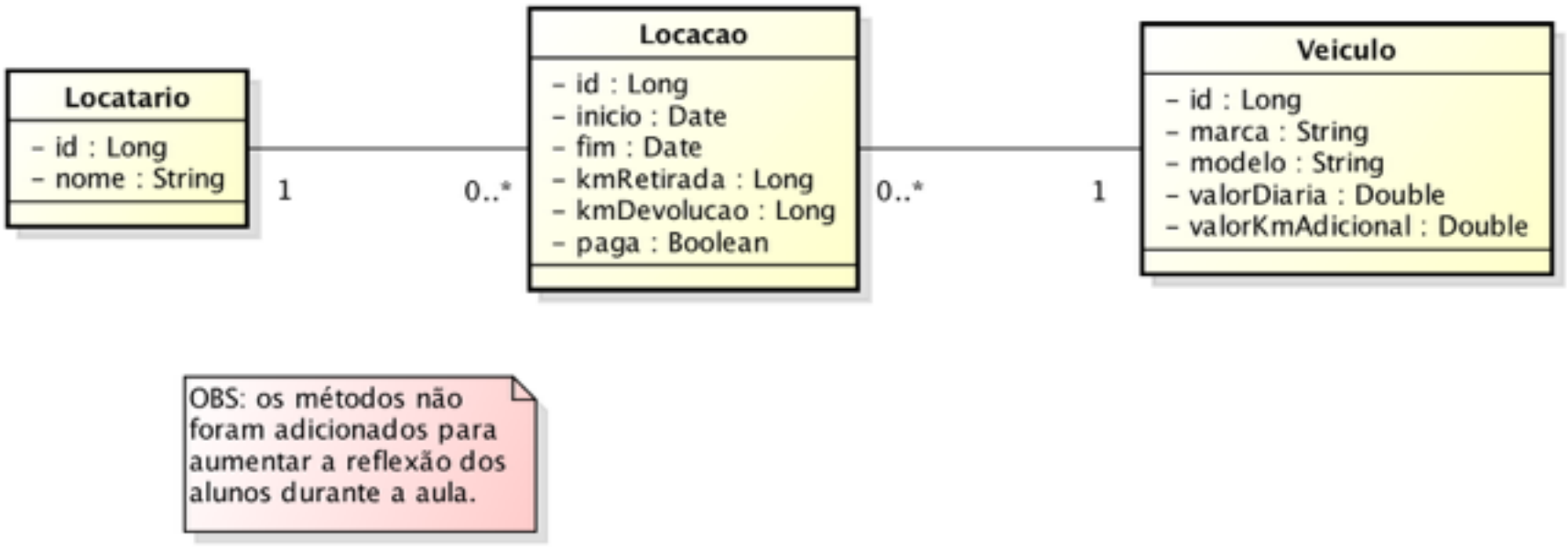
- Leis de Demeter;
- Padrões GRASP.

Dr. Márcia L. Aguena Castro
marcia@39dev.com

Me. Reinaldo de O. Castro
reinaldo@39dev.com

# Leis de Demeter

- Estudo de Caso para as 5 leis:



# Exercício - Leis de Demeter

- Implemente o método `divida()` da classe `Locatario` considerando os seguintes pontos:
  1. O valor padrão de uma locação é quantidade de dias que um veículo foi locado (que será no mínimo de1 dia) multiplicada pelo valor da diária desse veículo;
  2. Caso o locatário tenha percorrido mais de 200 km por dia durante a locação, a quilometragem excedente deve ser multiplicada pelo valor do quilômetro adicional do carro locado;
- O valor total do aluguel é a soma de 1 e 2.

# Exercício - Leis de Demeter

**Dica 1:** Para obter a quantidade de dias que o veículo foi alugado use o código abaixo:

- Importe a classe `TimeUnit` na classe que você achar apropriada:

```
import java.util.concurrent.TimeUnit;
```

- Para obter a diferença em dias entre duas datas em Java, baseie-se no código a seguir:

```
Long duracaoEmMilisegundos
     = data2.getTime()-data1.getTime();
Long resultado =
 TimeUnit.DAYS.convert(duracaoEmMilisegundos,
                       TimeUnit.MILLISECONDS);
```

# Exercício - Leis de Demeter

**Dica 2**: Siga os passos do seguinte algoritmo (ou crie um seu) do método `divida()` na classe `Locatario` para a programação em Java.

```
public Double divida(){
    // 1. Percorrer as locações do locatário e
    //    filtrar as que não foram pagas.

    // 2. Fazer o cálculo de quanto é o valor
    //    a pagar por cada locação que ainda
    //    não foi paga.

    // 3. Somar um cálculo de cada locação em
    //    um valor total.

    // 4. Retornar esse valor total.
    return 0.0;
}
```

# Exercício - Leis de Demeter

**Dica 3:**

- Parta dos arquivos `Locatario.java`, `Locacao.java` e `Veiculo.java` que está na pasta das apresentações.
- Coloque no mesmo pacote e execute a classe `locadora`.
- Ela irá iniciar os seus objetos do problema tem que imprimir no console do computador uma dívida de 900 reais.

# Exercício - Leis de Demeter

- Aqui apresentamos uma solução que fere a **Lei de Demeter** e geralmente, é como programamos quando começamos a pensar OO.

```
// Versão original, furando a Lei de Demeter.
  // Versão original, furando a Lei de Demeter.
  public Double divida() {

      // 1. Percorrer as locações do locatário e
      //    filtrar as que não foram pagas.
      Collection<Locacao> locacoesNaoPagas = new ArrayList<>();

      for (Locacao locacao : this.getLocacoes()) {

          if (!locacao.isPaga()) {
              locacoesNaoPagas.add(locacao);
          }
      }
```

# Exercício - Leis de Demeter

```
// 2. Fazer o cálculo de quanto é o valor
//    a pagar por cada locação que ainda
//    não foi paga.
Double totalDivida = 0.0;

for (Locacao locacaoNaoPaga : locacoesNaoPagas) {

    Long duracaoEmMilisegundos = locacaoNaoPaga.getFim().getTime()
                               - locacaoNaoPaga.getInicio().getTime();
    Long duracaoEmDias         = TimeUnit.DAYS.convert(duracaoEmMilisegundos, TimeUnit.MILLISECONDS);

    if (duracaoEmDias == 0L) {
        duracaoEmDias = 1L;
    }

    Veiculo veiculo           = locacaoNaoPaga.getVeiculo();
    Double  valorDiarias      = duracaoEmDias * veiculo.getValorDiaria();
    Double  valorKmAdicionais = 0.0;
    Long    kmAdicionais      = (locacaoNaoPaga.getKmDevolucao()
                               - locacaoNaoPaga.getKmRetirada())
                               - (duracaoEmDias * 200);

    if (kmAdicionais > 0) {
        valorKmAdicionais = kmAdicionais * veiculo.getValorKmAdicional();
    }
```

# Exercício - Leis de Demeter

```
87          // 3. Somar o cáculo de cada locação em
88              //    um valor total.
89              totalDivida += valorDiarias + valorKmAdicionais;
90
91          }
92
93          // 4. Retornar esse valor total.
94          return totalDivida;
95      }
```

# Leis de Demeter

- A **Law of Demeter** (em português, *Lei de Demeter*) também é conhecida como **Princípio do Mínimo Conhecimento**.
- Ajuda a diminuir o acoplamento de classes.
- Ela pode ser resumida nos seguintes pontos:
  - Cada classe deve ter **conhecimento limitado** sobre classes que não são "próximas" a ela;
  - Cada classe só se **comunica** (chama métodos) com suas classes amigas diretas, ou seja, ela **não se comunica com estranhos**.

# Leis de Demeter

- De uma maneira um pouco mais formal:
  - (1) um objeto A pode chamar um método em um objeto B,
  - (2) mas o objeto A não pode usar o objeto B (amigo) para alcançar um objeto C (estranho) e chamar um método neste.

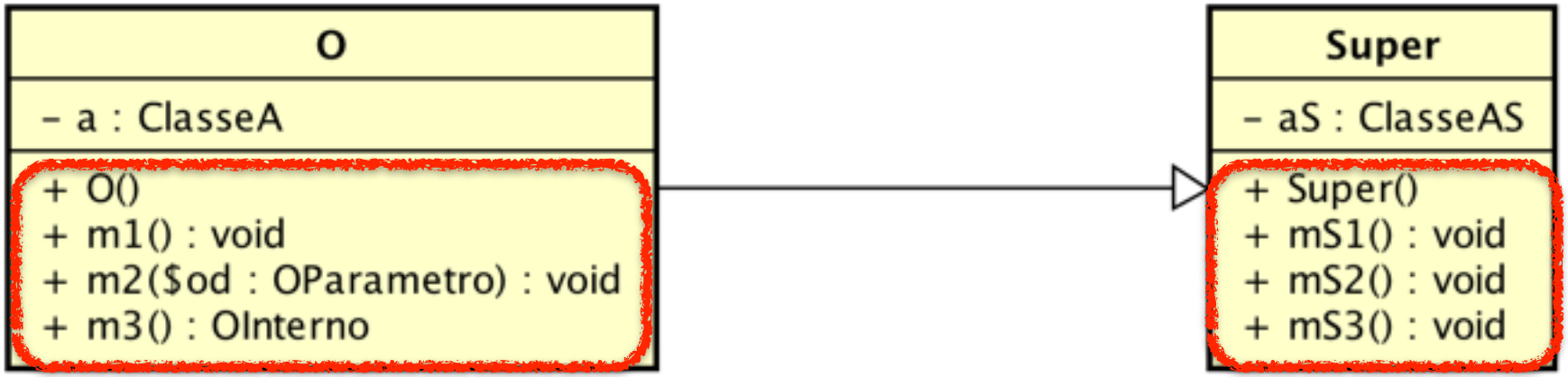
```
3  public class ClasseA{
4      private ClasseB objetoB;
5
6      public void metodoDoObjetoAOk(){
7  (1)     objetoB.metodoDoObjetoB();
8      }
9      public void metodoDoObjetoARuim(){
10 (2)     ClasseC objetoC = objetoB.obterObjetoC();
11         objetoC.metodoDoObjetoC();
12     }
13 }
```

# Leis de Demeter - Os 5 princípios

- A questão aqui é: quem são os objetos amigos?
- Considerando um método `m` em um objeto `o`, os amigos em que `m` pode invocar métodos são:
  1. O **próprio** objeto `o` (ou seja, qualquer `this.metodo()` e `super.metodo()` é permitido);
  2. Qualquer objeto passado como **parâmetro** de `m`;
  3. Qualquer objeto **criado** dentro de `m`;
  4. Qualquer objeto que seja **atributo** do objeto `o`;
  5. Qualquer objeto **global**, acessível por `o`, no escopo de `m`.

# Leis de Demeter - 1o Princípio

- "O próprio objeto `o` (ou seja, qualquer `this.metodo()` e `super.metodo()` é permitido)."




- Qualquer método de 0 =>
  - Pode utilizar qualquer **método** de 0;
  - Pode utilizar qualquer **métodos** de `Super`;

# Leis de Demeter - 1o Princípio

"O próprio objeto `o` (ou seja, qualquer `this.metodo()` e `super.metodo()` é permitido)."

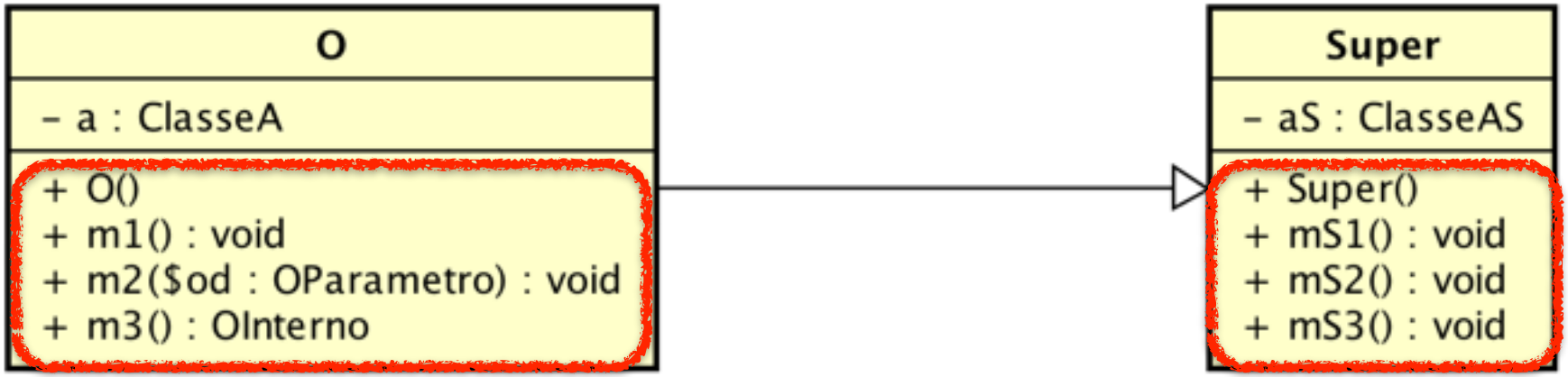
- Suponha a existência de um método `divida()` na classe `Locatario`, que retorna um `Double` representando o valor da dívida de um locatário qualquer, podemos implementar um método `ehBomPagador()`, que retorna `true` quando a dívida do locatário é de até R$ 100, como exemplo da diretiva 1 da Lei de Demeter.

# Leis de Demeter - 1o Princípio

```
3  import java.util.Collection;
4
5  public class Locatario {
6      private long id;
7      private String nome;
8      private Collection<Locacao> locacoes;
9
10     public Locatario(){}
11     public double divida(){
12         return 0.0;
13     }
14     public boolean ehBomPagador(){
15         return this.divida() <= 100.00;
16     }
17 }
```

# Leis de Demeter - 2o Princípio

"Qualquer objeto passado como **parâmetro** de `m`"




- O método `m2()` recebe como parâmetro um objeto `$od` da classe `OParametro` =>
- O método `m2()` pode utilizar **qualquer método** do objeto `$od`.

## Leis de Demeter - 2o Princípio

"Qualquer objeto passado como **parâmetro** de `m`."

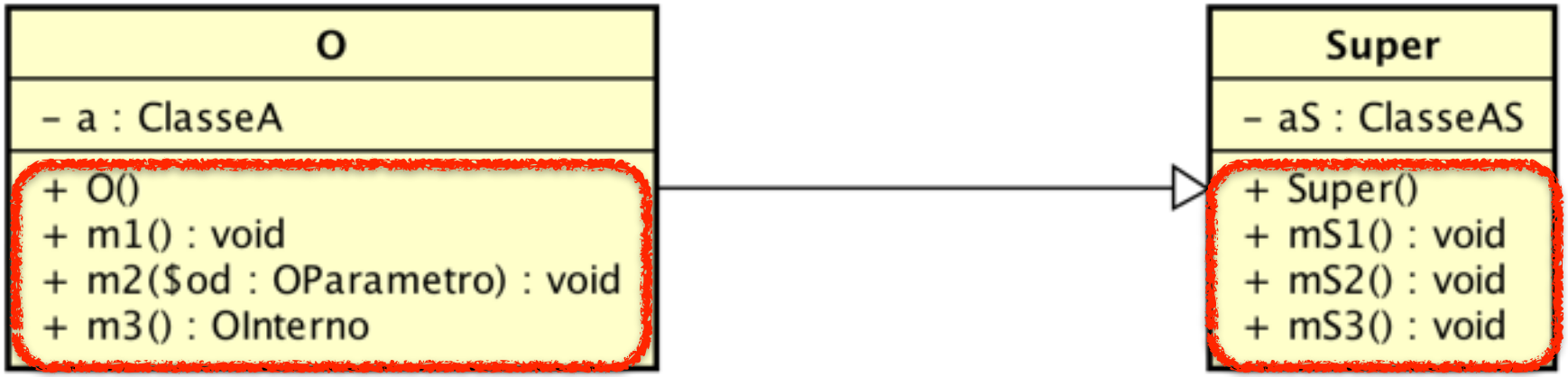
- Suponha a existência de um método `pagar(Locacao $locacao)` na classe `Locatario`, que seta o atributo `paga` da classe `Locacao` para `true` caso o aluguel em questão realmente pertença ao locatário e realmente ainda não foi paga. O método tem tipo de retorno `void`.

# Leis de Demeter - 2o Princípio

```
3  import java.util.Collection;
4
5  public class Locatario {
6      private Long id;
7      private String nome;
8      private Collection<Locacao> locacoes;
9
10     public Locatario(){}
11     public double divida(){..
14     public boolean ehBomPagador(){
17     public void pagar(Locacao $locacao){
18         if (!this.getId().equals($locacao.getLocatario().getId())){
19             System.out.println("O aluguel não pertence ao locatário.");
20         }else if ($locacao.isPaga()){
21             System.out.println("O alguel já está pago.");
22         }else{
23             $locacao.setPaga(true);
24         }
25     }
26     private Long getId() {..
29 }
```

# Leis de Demeter - 3o Princípio

"Qualquer objeto **criado dentro** de `m`"




- O método `m3()` **cria** um objeto `oI` da classe `OInterno` =>
- O método `m3()` pode utilizar qualquer método do objeto `oI`.

## Leis de Demeter - 3o Princípio

"Qualquer objeto **criado dentro** de `m`."

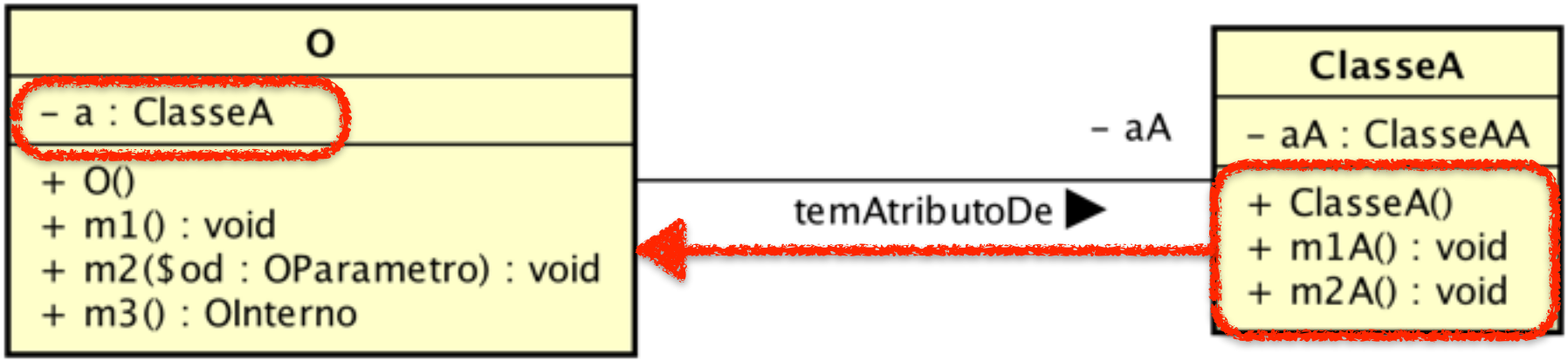
- Suponha a existência de um método `locacoesEmAberto()` na classe `Locatario`, que retorna um `ArrayList` com todos as locações que ainda não foram pagas de um locatário qualquer.

# Leis de Demeter - 3o Princípio

```
77    public Collection<Locacao> locacoesEmAberto() {
78
79        Collection<Locacao> resultado = new ArrayList<>();
80
81        for (Locacao locacao : this.getLocacoes()) {
82
83            if (!locacao.isPaga()) {
84                resultado.add(locacao);
85            }
86        }
87
88        return resultado;
89    }
```

# Leis de Demeter - 4o Princípio

"Qualquer objeto que seja **atributo** do objeto `o`".




- A classe `O` tem um atributo `a` da classe `ClasseA` =>
- Todos os métodos da classe `O` podem utilizar o atributo `a` e os **métodos** da `ClasseA` relacionados a ele.

# Leis de Demeter - 4o Princípio

"Qualquer objeto que seja **atributo** do objeto `o`"

Suponha a existência de um método `categoria()` na classe `Locatario` que retorne uma `String`, de acordo com o volume de locações que um locatário possui:

- **OURO**: caso o locatário possua 100 ou mais locações;
- **PRATA**: caso o locatário possua entre 50 e 99 locações
- **BRONZE**: abaixo de 50 locações

# Leis de Demeter - 4o Princípio

```
 7  public class Locatario {
 8
 9      private Long   id;
10      private String nome;
11      private Collection<Locacao> locacoes;
```

```
72      public String categoria() {
73
74          if (this.locacoes.size() > 100) {
75              return "OURO";
76          }
77
78          if (this.locacoes.size() >= 50) {
79              return "PRATA";
80          }
81
82          return "BRONZE";
83      }
84      public Collection<Locacao> locacoesEmAberto() {
97  }
```

# Leis de Demeter - 5o Princípio

"Qualquer **objeto global**, acessível por `o`, no escopo de `m`"

RECOMENDAÇÃO IMPORTANTE 1: **evite o máximo possível** usar objetos globais, acessíveis por toda a aplicação!

RECOMENDAÇÃO IMPORTANTE 2: defina um objeto global somente se todos os atributos dele forem declarados como **final**, ou seja, TODOS devem ser constantes!

# Leis de Demeter - Recomendação Geral

Utilize sempre e somente UM operador ponto para invocar métodos das classes que você mesmo programa para seu sistema!

# Exercício - Leis de Demeter

Agora que já conhecemos as Leis de Demeter para minimizar o acoplamento entre as classes:

- Identifique onde o código anterior quebra essas leis.

```
// Versão original, furando a Lei de Demeter.
public Double divida() {

    // 1. Percorrer as locações do locatário e
    //    filtrar as que não foram pagas.
    Collection<Locacao> locacoesNaoPagas = new ArrayList<>();

    for (Locacao locacao : this.getLocacoes()) {

        if (!locacao.isPaga()) {
            locacoesNaoPagas.add(locacao);
        }
    }
```

# Exercício - Leis de Demeter

```
// 2. Fazer o cálculo de quanto é o valor
//    a pagar por cada locação que ainda
//    não foi paga.
Double totalDivida = 0.0;

for (Locacao locacaoNaoPaga : locacoesNaoPagas) {

    Long duracaoEmMilisegundos = locacaoNaoPaga.getFim().getTime()
                               - locacaoNaoPaga.getInicio().getTime();
    Long duracaoEmDias         = TimeUnit.DAYS.convert(duracaoEmMilisegundos, TimeUnit.MILLISECONDS);

    if (duracaoEmDias == 0L) {
        duracaoEmDias = 1L;
    }

    Veiculo veiculo           = locacaoNaoPaga.getVeiculo();
    Double  valorDiarias      = duracaoEmDias * veiculo.getValorDiaria();
    Double  valorKmAdicionais = 0.0;
    Long    kmAdicionais      = (locacaoNaoPaga.getKmDevolucao()
                               - locacaoNaoPaga.getKmRetirada())
                               - (duracaoEmDias * 200);

    if (kmAdicionais > 0) {
        valorKmAdicionais = kmAdicionais * veiculo.getValorKmAdicional();
    }
```

# Exercício - Leis de Demeter

```
87          // 3. Somar o cáculo de cada locação em
88              //    um valor total.
89              totalDivida += valorDiarias + valorKmAdicionais;
90
91          }
92
93          // 4. Retornar esse valor total.
94          return totalDivida;
95      }
```

# Exercício - Leis de Demeter

Refatoração da classe `Locatario` para que o método `divida()` respeite completamente a Lei de Demeter:

```
74  Double valorDiarias      = duracaoEmDias * locacaoNaoPaga.valorDiaria();
75  Double valorKmAdicionais = 0.0;
76  Long   kmAdicionais      = (locacaoNaoPaga.getKmDevolucao()
77                           - locacaoNaoPaga.getKmRetirada())
78                           - (duracaoEmDias * 200);
79
80  if (kmAdicionais > 0) {
81      valorKmAdicionais = kmAdicionais * locacaoNaoPaga.valorKmAdicional();
82  }
```

# Exercício - Leis de Demeter

Métodos incluídos na classe `Locacao`:

```
101    public Double valorDiaria() {
102        return this.getVeiculo().getValorDiaria();
103    }
104
105    public Double valorKmAdicional() {
106        return this.getVeiculo().getValorKmAdicional();
107    }
```

## Leis de Demeter - Recomendação Geral

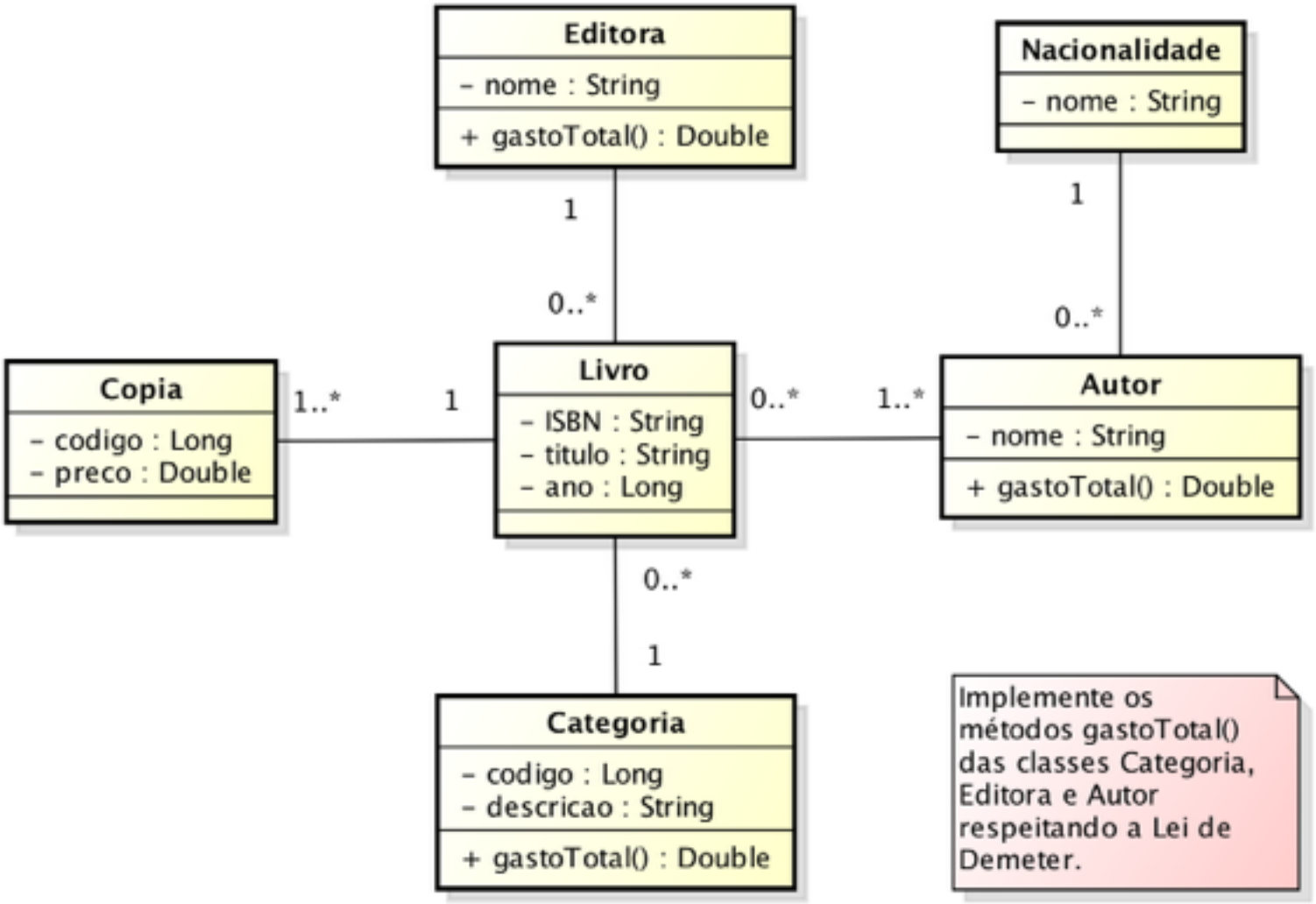
- As Leis de Demeter só diminuem o acoplamento das classes.
- A aplicação delas é só um preparo para os padrões de projeto.

# Exercício - Leis de Demeter

**Exercício Biblioteca:**

- Estendendo o problema da biblioteca visto antes, para ter um controle maior de como está gastando o seu dinheiro, a biblioteca gostaria de saber o gasto total por: editora, categoria e autor (veja o método na `gastoTotal()` nas respectivas classes).

# Exercício - Leis de Demeter



# Exercício - Leis de Demeter

**Exercício Hospital:**

- Estendendo o problema do Hospital:
- Um item muito importante para o sistema é o cálculo do faturamento geral do hospital (veja o método na `faturamento()` na classe **Hospital**), levando em conta:
  - todas as suas unidades;
  - o valor hora de cada médico e
  - o quanto tempo levou cada parto que o médico fez.
  - Note também que um parto, quando considerado complicado, deve ter seu valor aumentado em 20%.
- Além do faturamento total do hospital, deseja-se também saber o faturamento por especialidade (veja o método na `faturamento()` na classe **Especialidade**)
- Esse sistema só tem informações sobre os partos do mês atual, assim, não é necessária nenhuma comparação entre datas para obter o faturamento total do mês corrente.

# Exercício - Leis de Demeter