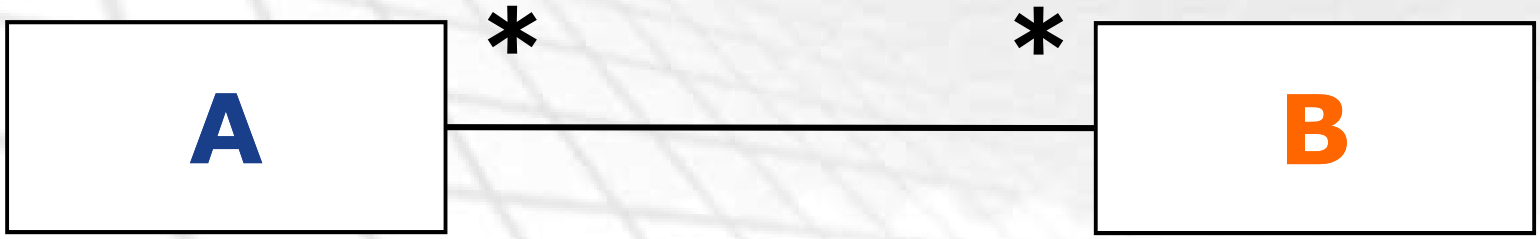
Universidade do Estado do Amazonas
Escola Superior de Tecnologia

Projeto de Programa

Associações e dependências – B

Associação muitos-para-muitos

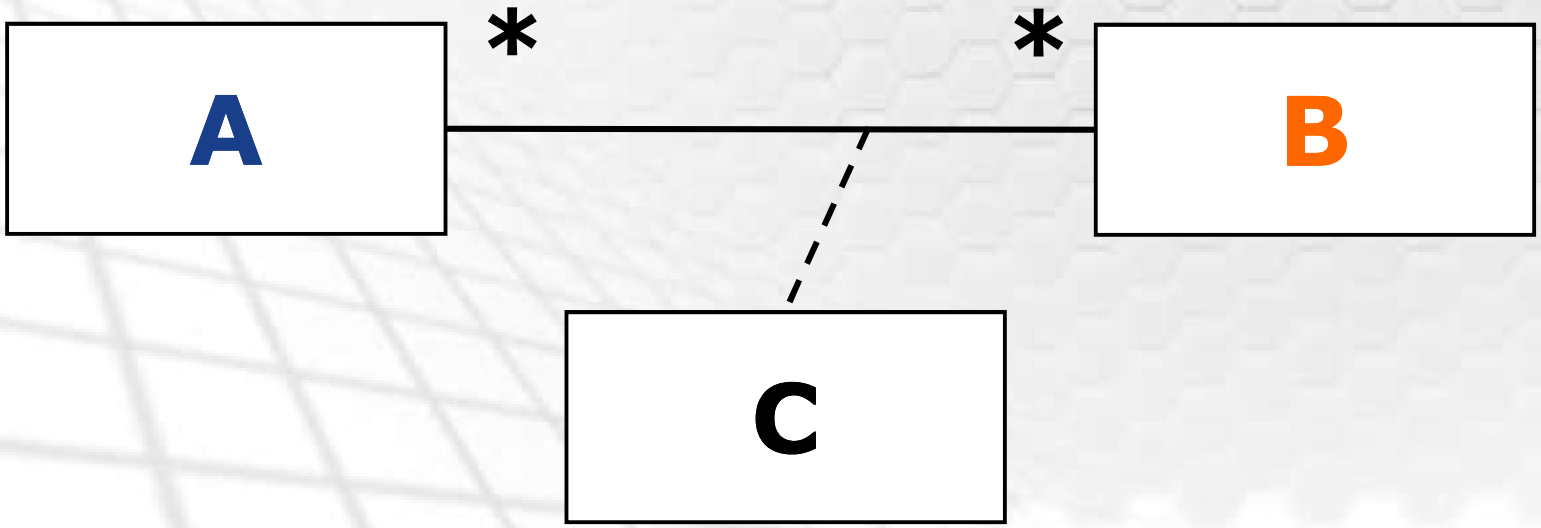
- **A tem uma lista de b's e B tem uma lista de a's.**



Associação muitos-para-muitos

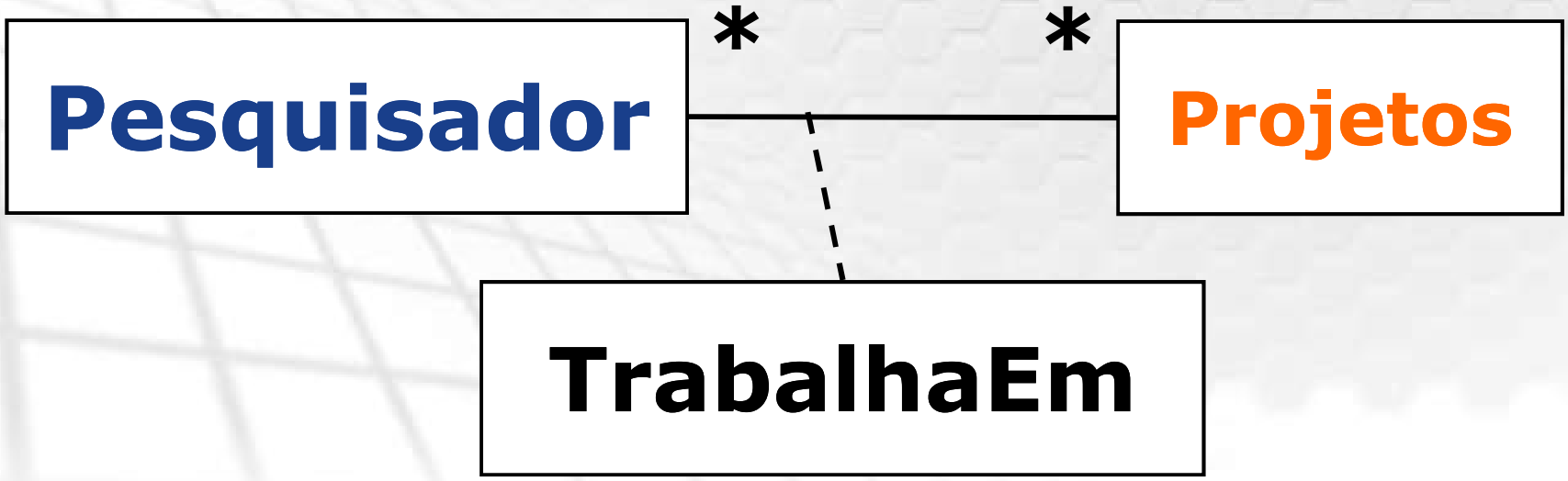
- **Associações muitos-para-muitos podem utilizar uma classe associativa.**

Associação muitos-para-muitos

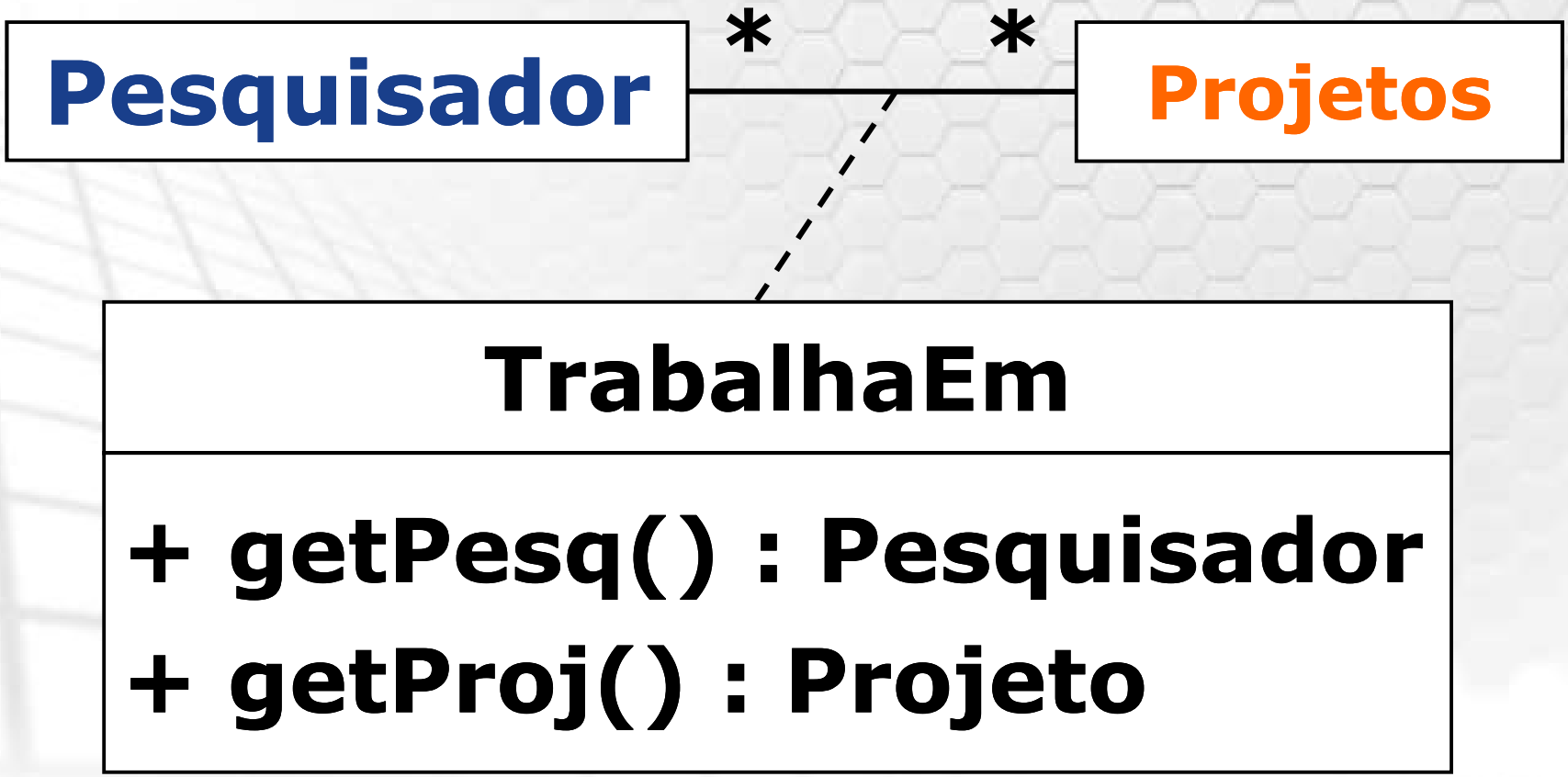


Associação muitos-para-muitos

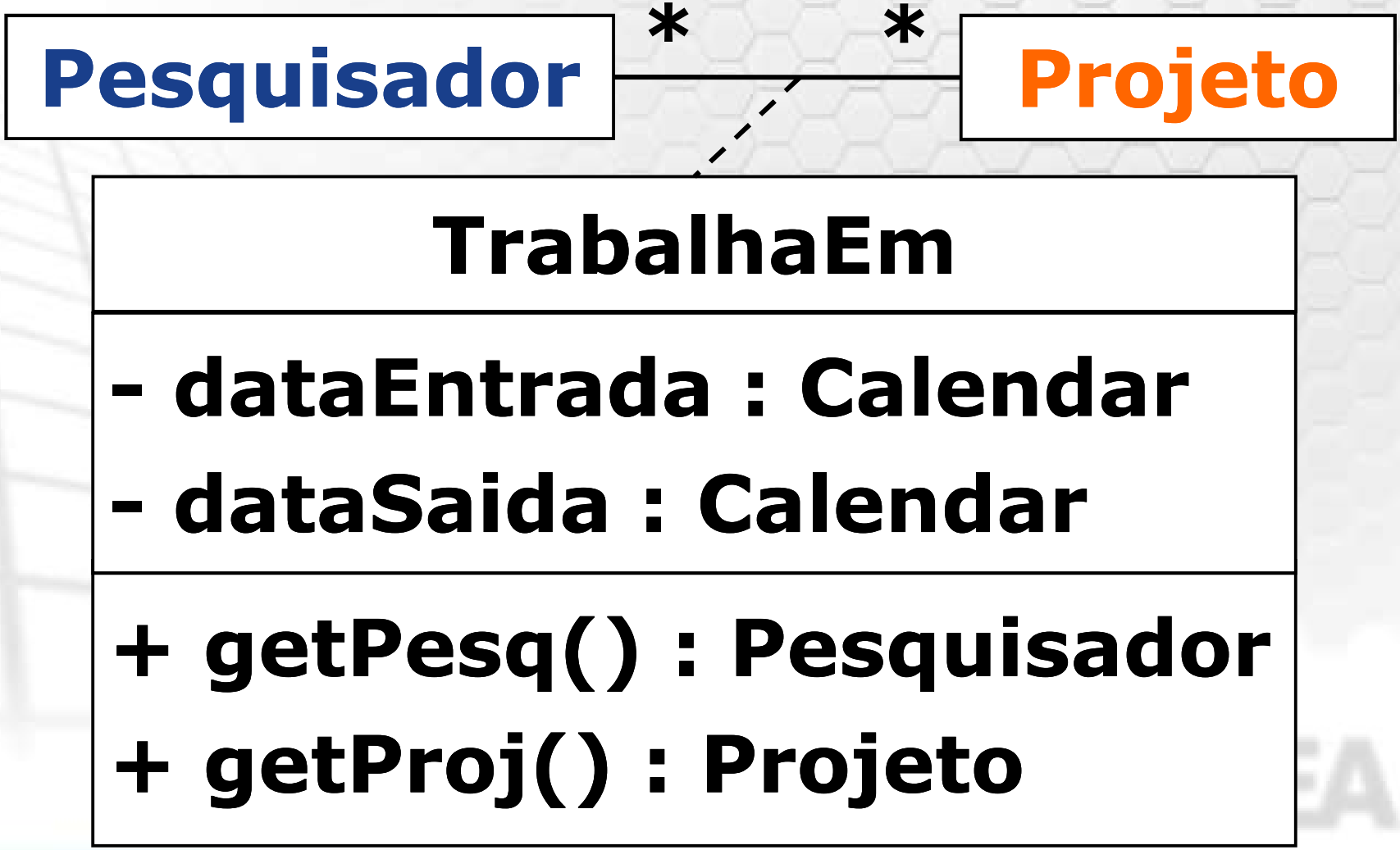
Classe associativa: TrabalhaEm.



Associação muitos-para-muitos

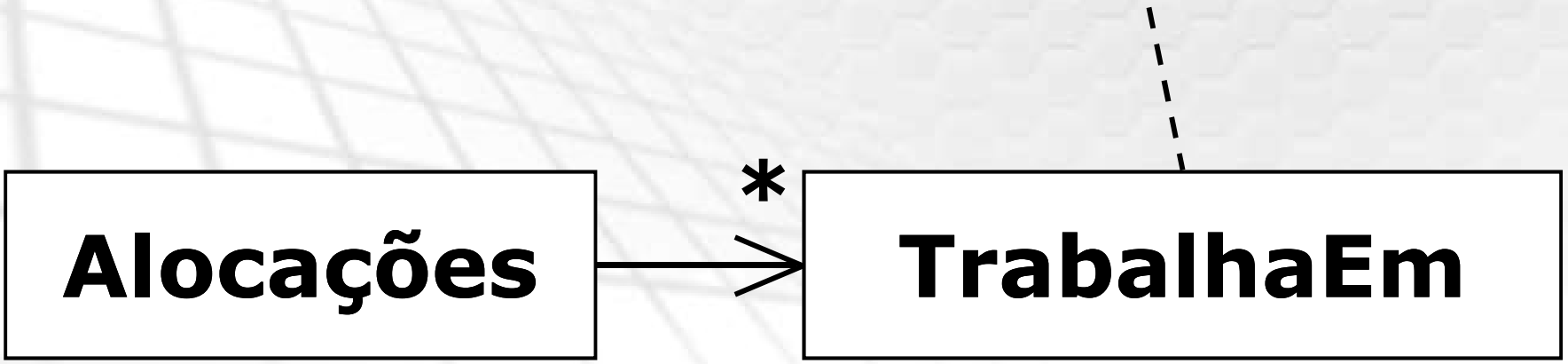


Associação muitos-para-muitos

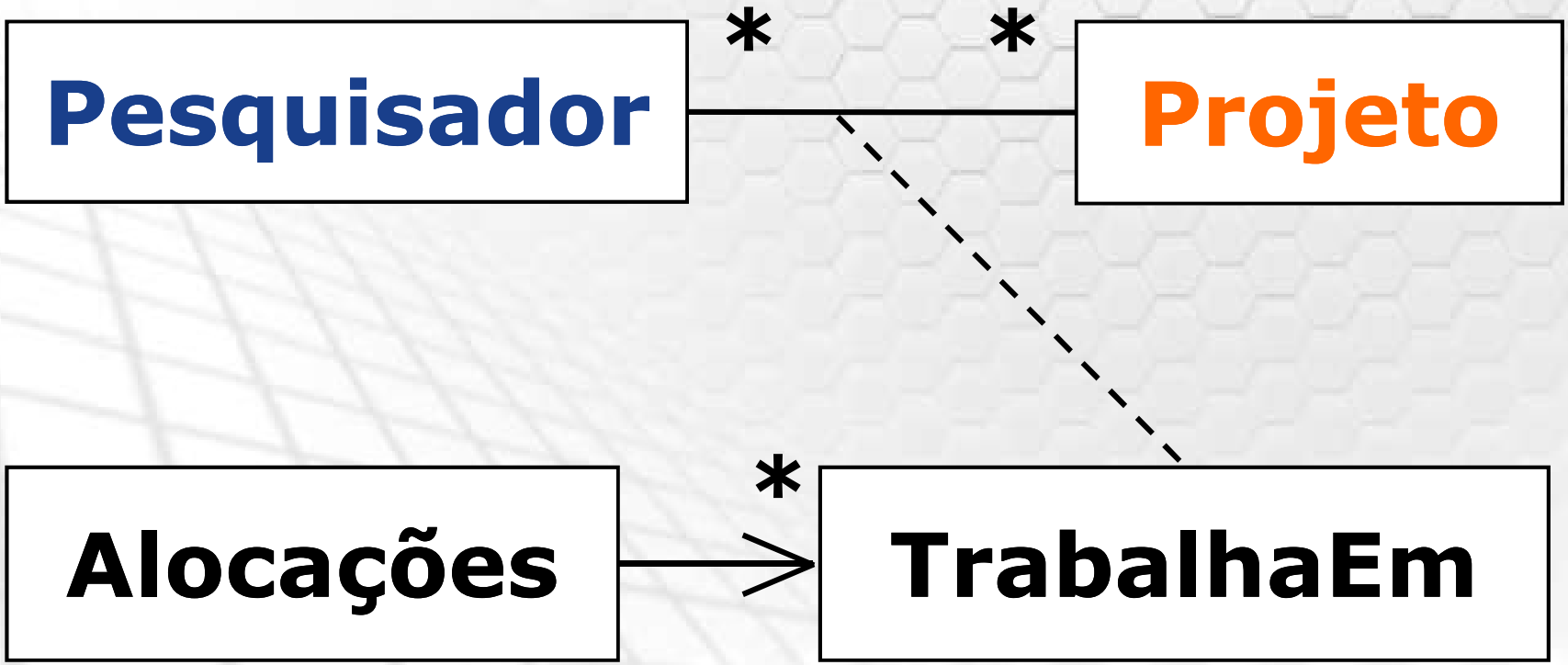


Associação muitos-para-muitos

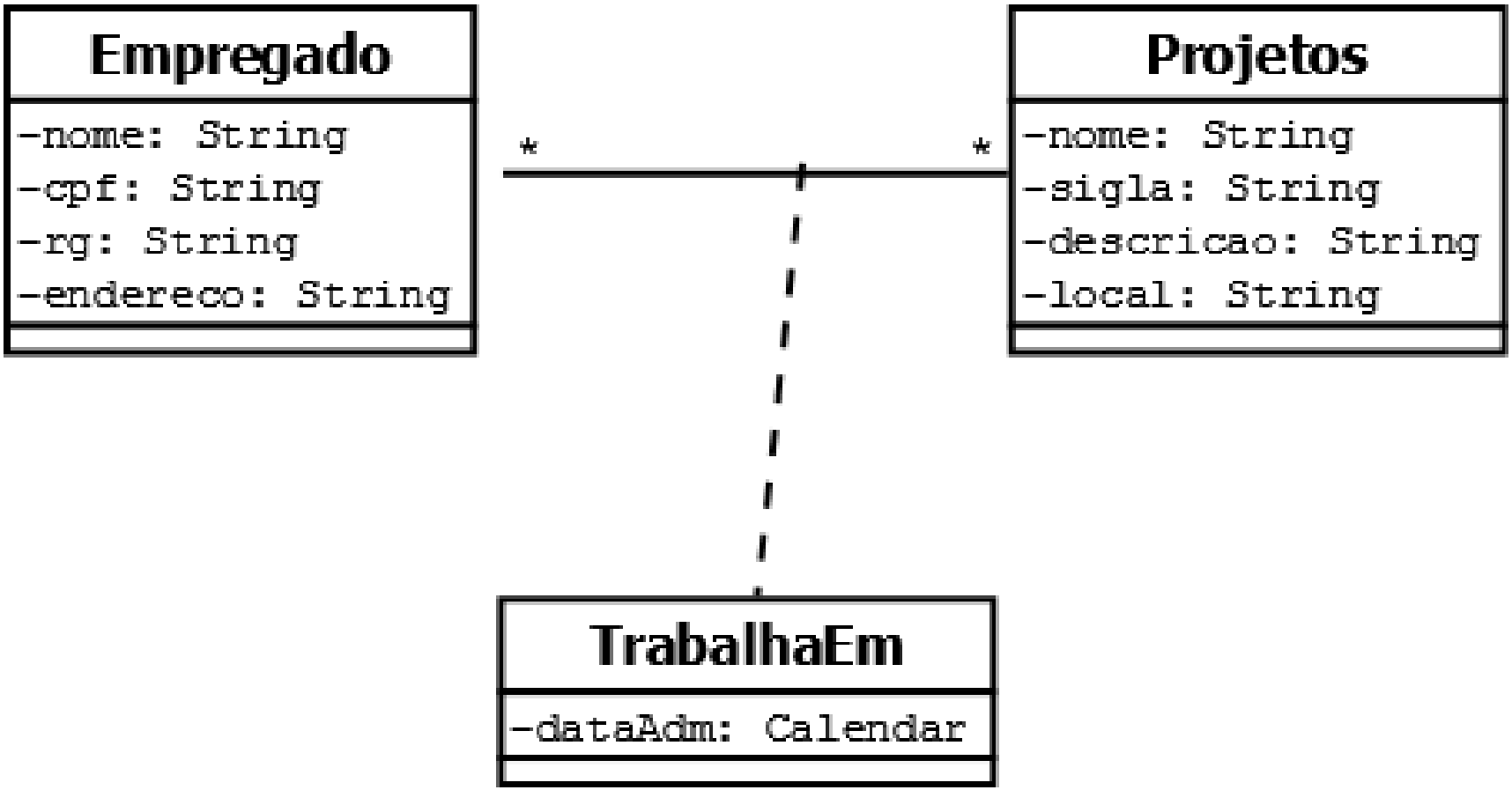
- **É preciso manter uma lista de objetos da classe associativa.**

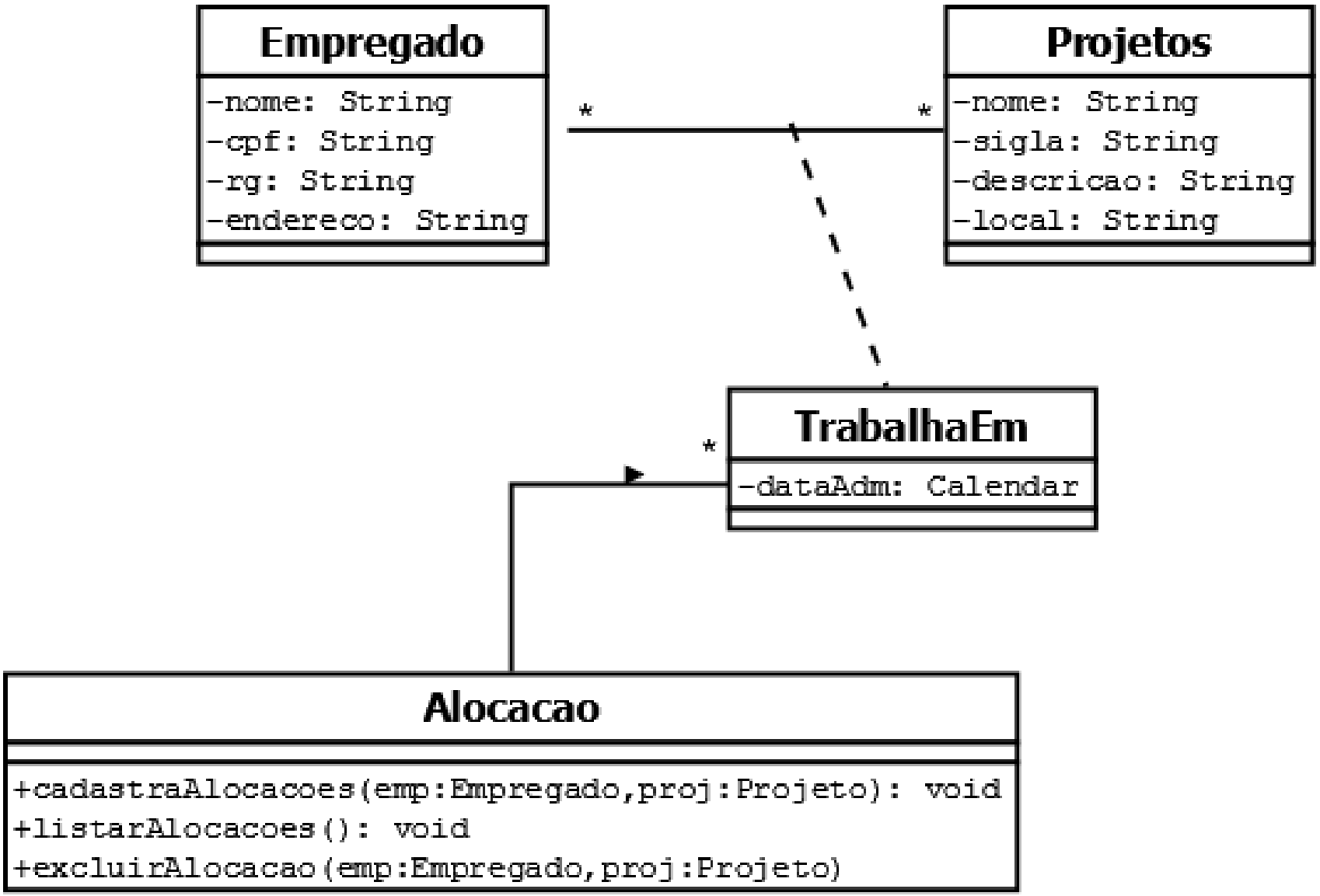


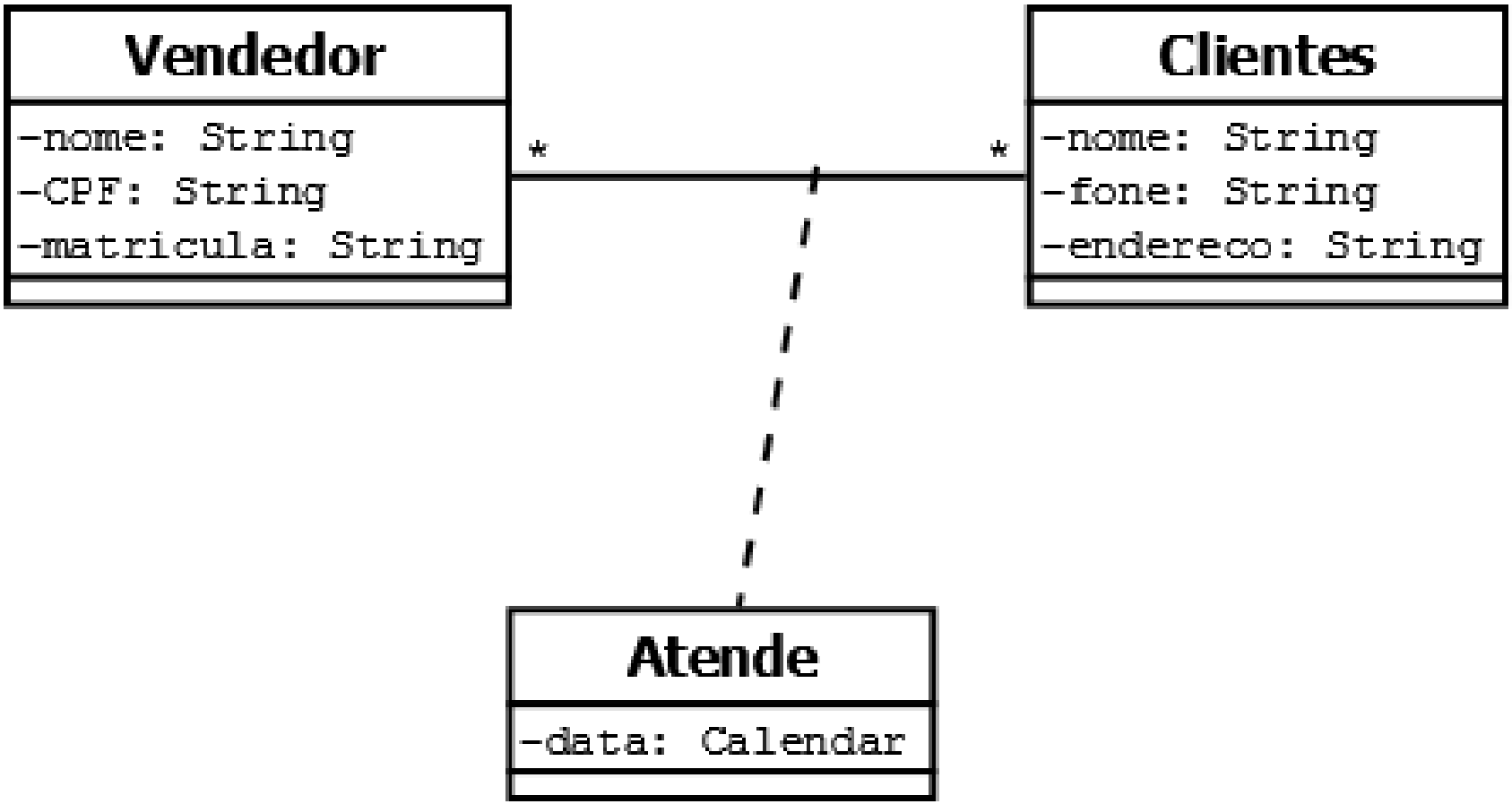
Associação muitos-para-muitos

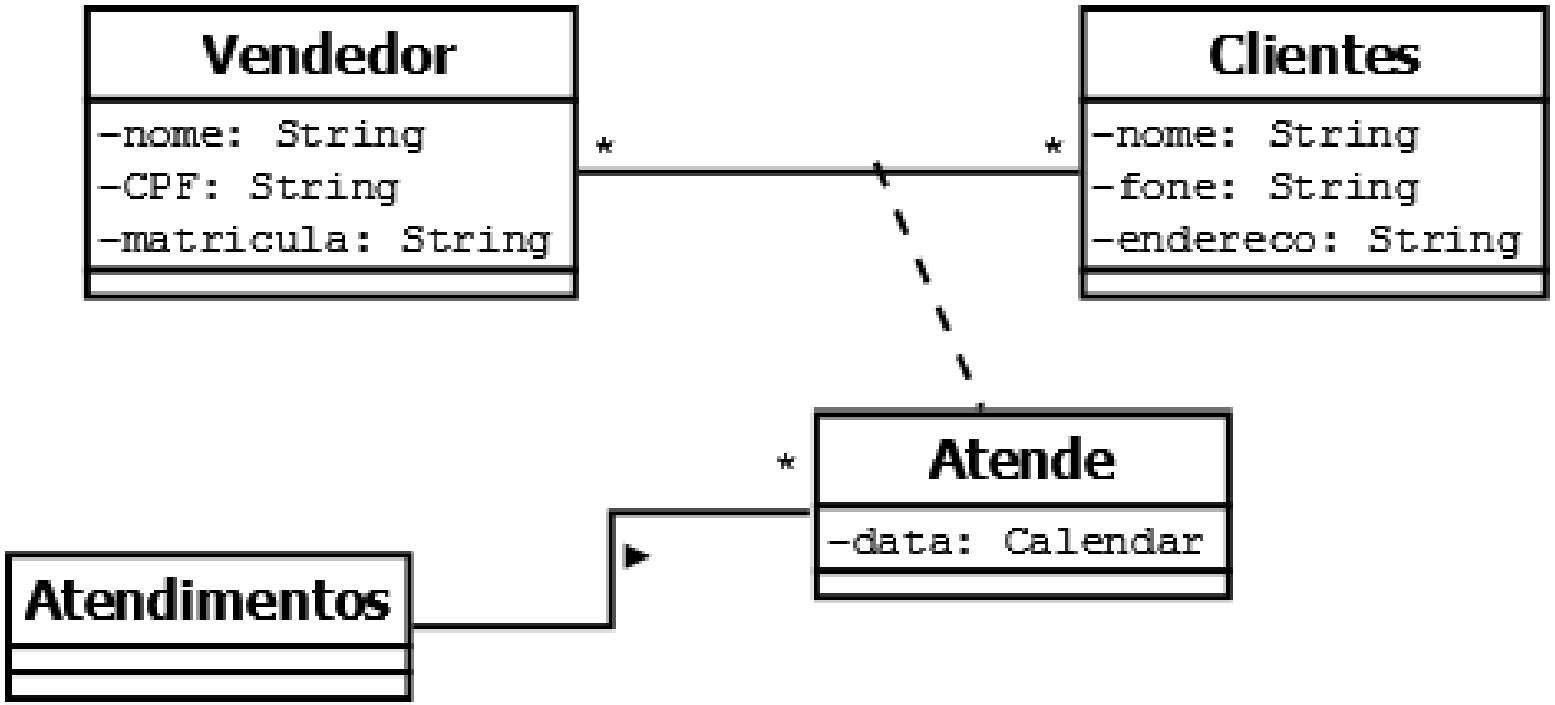


- **Código no eclipse.**


Empregado
-nome: String
-cpf: String
-rg: String
-endereco: String
*
*
Projetos
-nome: String
-sigla: String
-descricao: String
-local: String
TrabalhaEm
-dataAdm: Calendar


Empregado
-nome: String
-cpf: String
-rg: String
-endereco: String
*
*
Projetos
-nome: String
-sigla: String
-descricao: String
-local: String
TrabalhaEm
-dataAdm: Calendar
*
Alocacao
+cadastraAlocacoes(emp:Empregado,proj:Projeto): void
+listarAlocacoes(): void
+excluirAlocacao(emp:Empregado,proj:Projeto)


Vendedor
-nome: String
-CPF: String
-matricula: String
*
*
Clientes
-nome: String
-fone: String
-endereco: String
Atende
-data: Calendar


Vendedor
-nome: String
-CPF: String
-matricula: String
*
*
Clientes
-nome: String
-fone: String
-endereco: String
*
Atende
-data: Calendar
Atendimentos

- **Associação é uma relação estrutural:**
  - **Uma classe faz parte da estrutura de outra**
  - **Exemplo:**
    - **Motor faz parte de Carro**
    - **Empregado associado a Empresa**

- **Relacionamento de Associação ainda pode ser subdivido em:**
  - **Agregação**
  - **Composição**

Agregação e composição

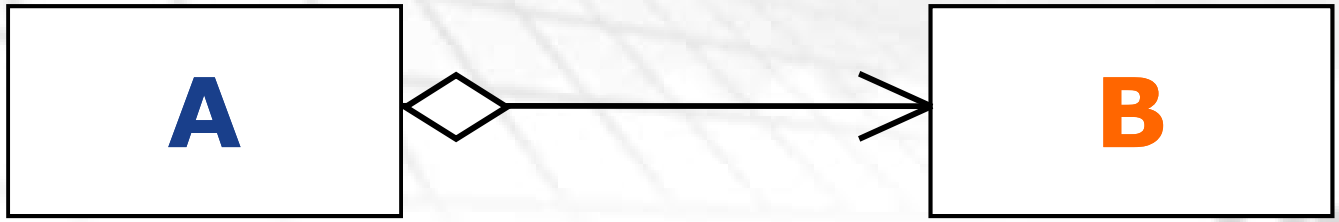
- **É uma interpretação que atribuímos a uma associação onde prevalece a idéia de que alguns objetos são partes de outro (todo-parte, tem-um).**

- **Ela é representada por uma linha ligando as duas classes da relação e a presença de um símbolo: losango.**

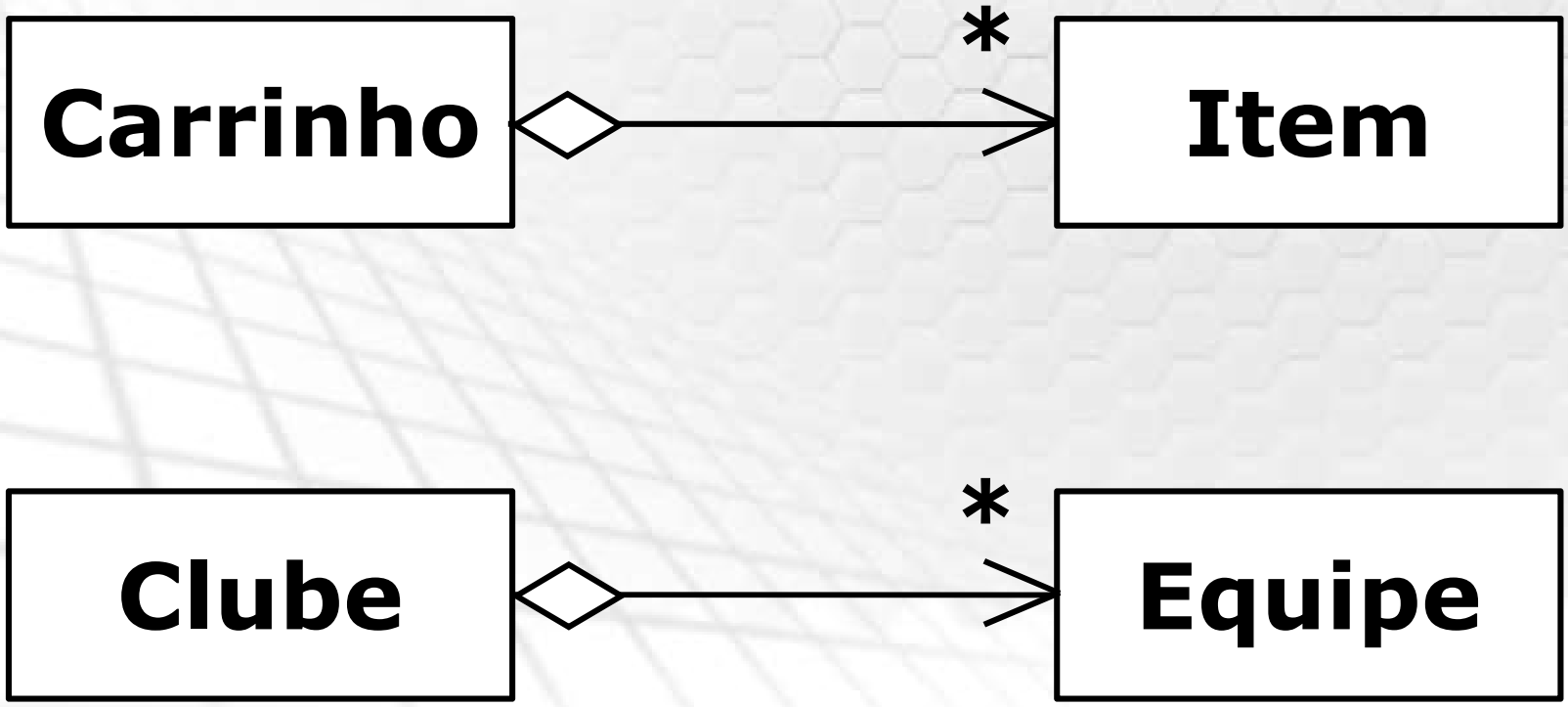
- **O losango é colocado na conexão entre a linha e a classe que é considerada a mais importante da relação.**

Associação agregação

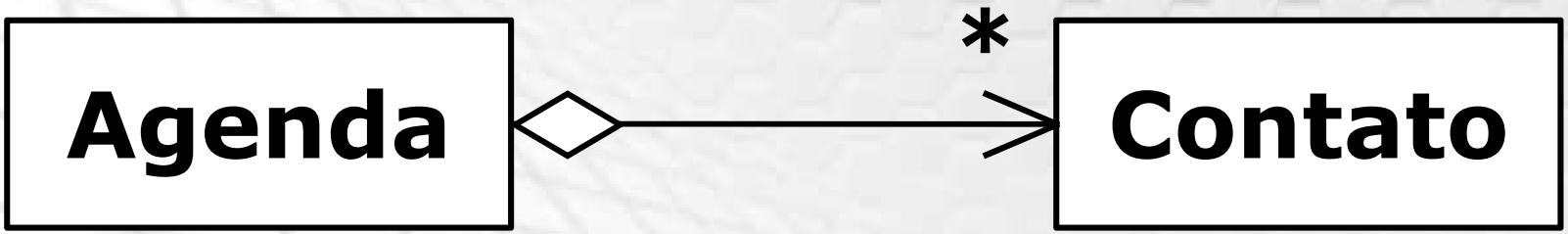
- Em uma agregação um objeto a contém outros objetos {b_1, b_2, ..., b_n} que tem existência independente de a.



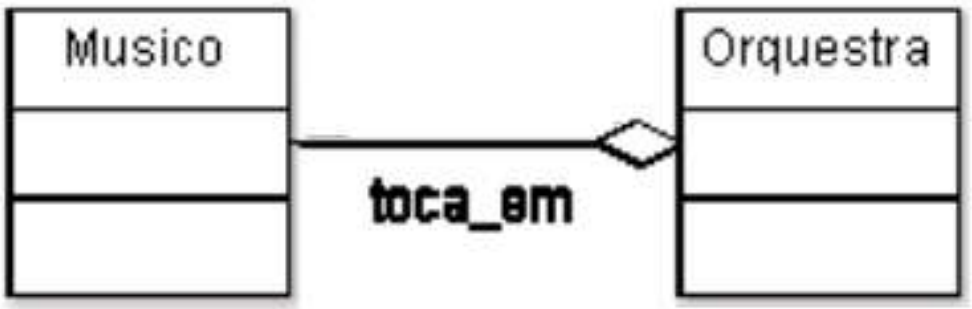
Agregação: exemplos

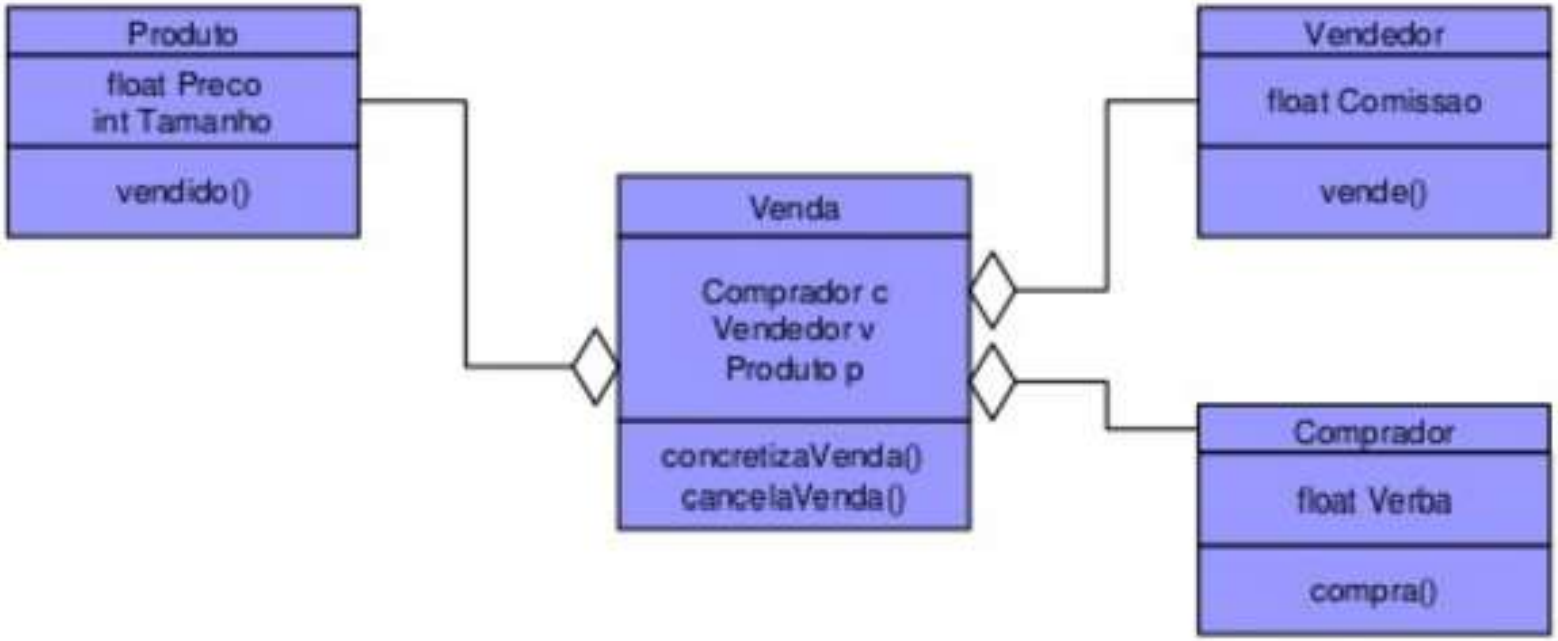


Agregação: exemplos



- **A Figura abaixo representa que "Musico *toca_em* Orquestra".**
- **Nesse contexto, deseja-se destacar que a classe Orquestra é "mais importante ou maior" que a classe Músico.**
  - **Todo-parte**





•Neste caso Venda é o objeto definido como sendo o todo. E este objeto somente pode existir caso os demais objetos que o compõem também existam.

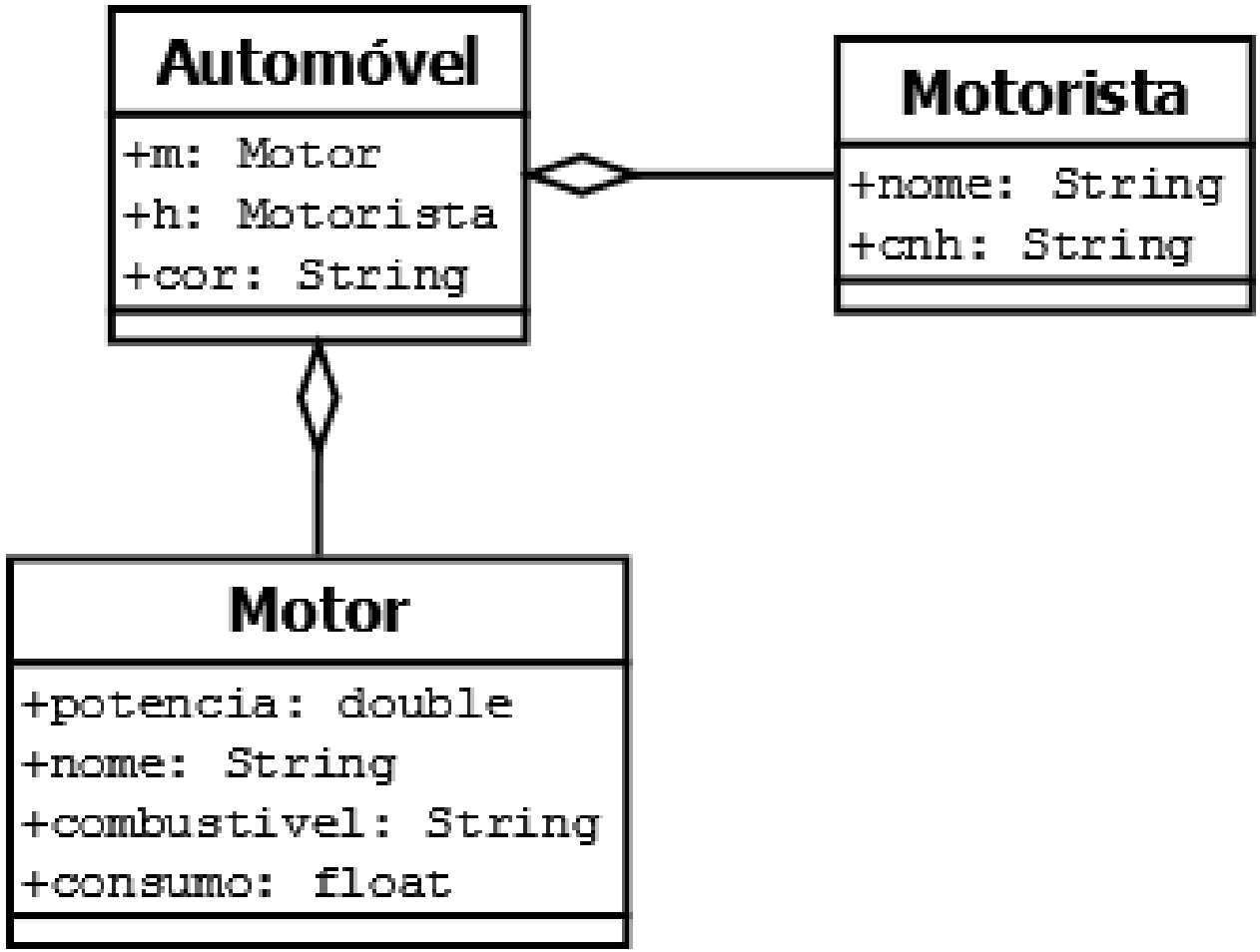
```
public class Vendedor {
  float Comissao;
  public void vende() {
    System.out.println(“Vendido”);
  }
}
```

```
public class Produto {
    float Preco;
    int Tamanho
    public void vendido() {
      System.out.println(“Vendido”);
    }
}
```

```
public class Comprador {
	float Verba;
	public void compra() {
		System.out.println(“Comprado”);
	}
}
```

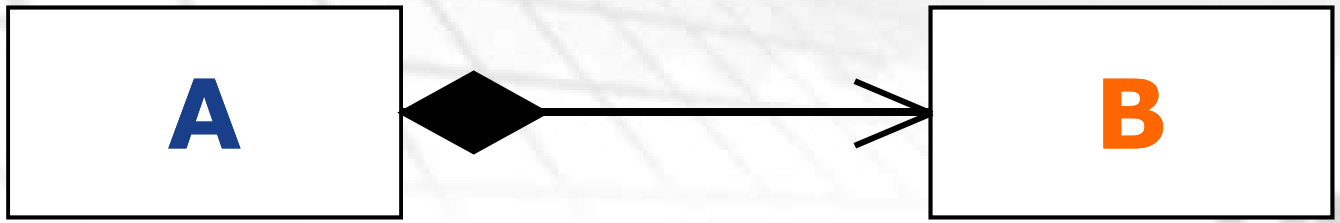
```
public class Venda {
   Comprador c;
   Vendedor v;
   Produto p;
   public void concretizaVenda() {
      System.out.println("Venda efetuada");
       c.Verba -= p.Preco;
       v.Comissao += p.Preco * 0.1f; p.vendido();
    }
   public void cancelaVenda() {
      System.out.println("Venda cancelada");
}
}
```

- **Agregação** é um tipo de relacionamento caracterizado como PARTE-TODO.
- Diferente da composição!!!
- Faz sentido que sobreviva a PARTE mesmo que o TODO não exista.


Automóvel
+m: Motor
+h: Motorista
+cor: String
Motorista
+nome: String
+cnh: String
Motor
+potencia: double
+nome: String
+combustivel: String
+consumo: float

Composição

- Se os objetos {b_1, b_2, ..., b_n} formam o objeto a, a associação passa a ser chamada de composição.

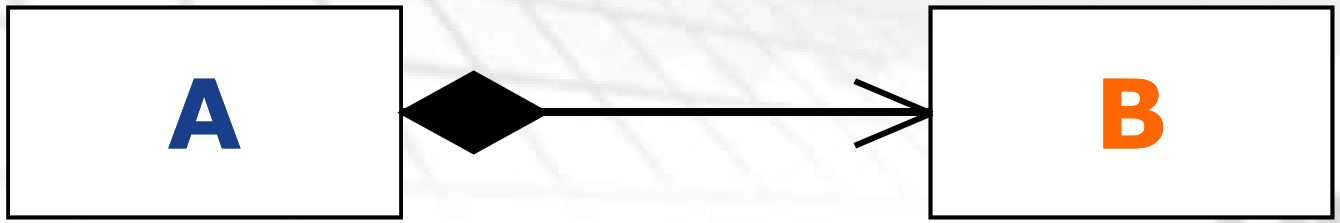


- **Adiciona um grau de importância semântica à relação.**
- **Define a relação de posse.**
- **Significa que um objeto da classe pertence apenas e exclusivamente ao objeto da outra classe.**
- **A classe mais importante gerencia a criação e destruição das partes que a compõe.**

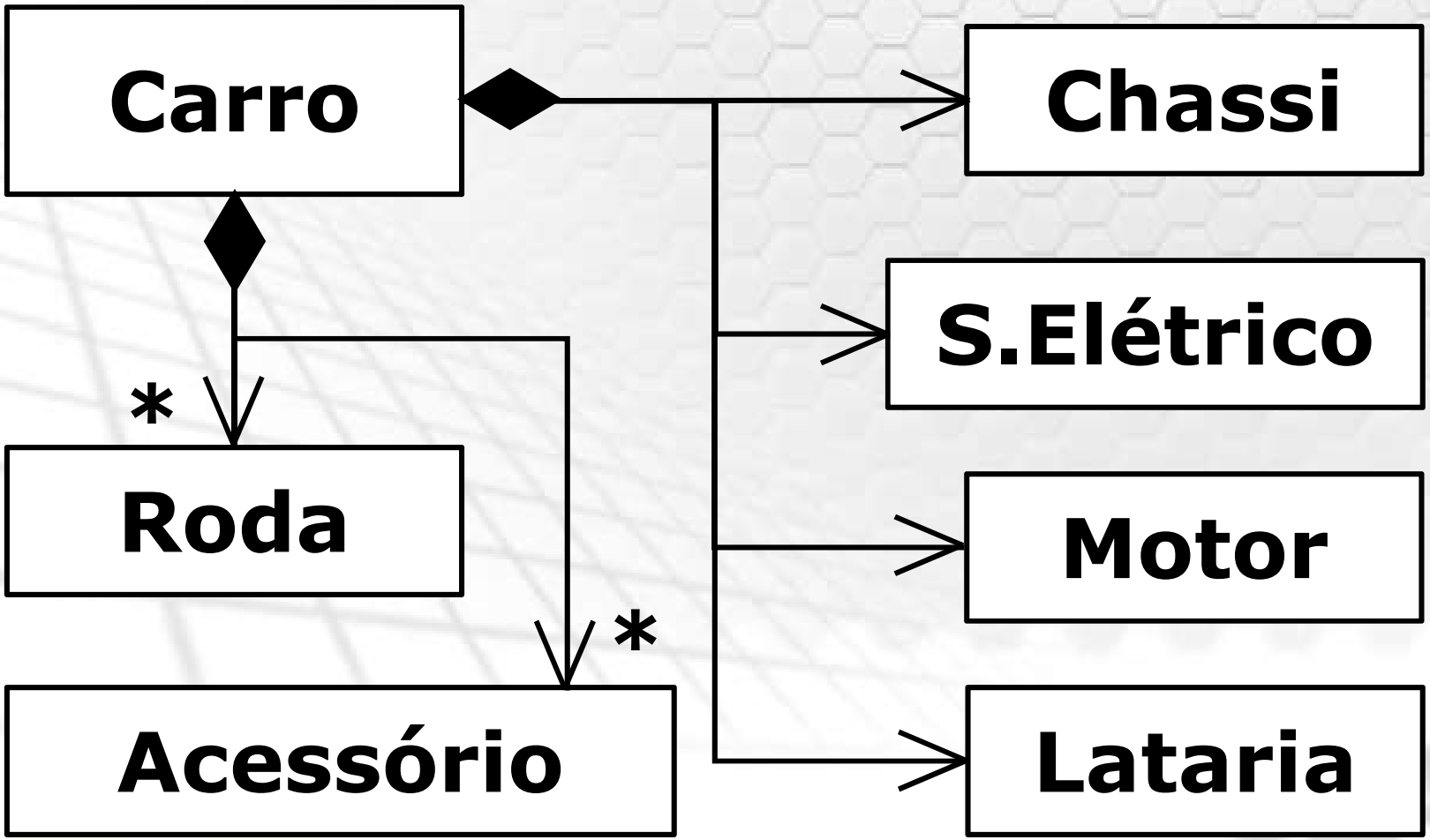
- **Composição:**
  - Tipo de associação mais forte que a agregação.
  - Caracterizado como PARTE-TODO.
  - O TODO ele é responsável pelo ciclo de vida da PARTE.
  - A composição é aplicada quando a PARTE não faz sentido existir sem o TODO e quando o objeto que representa o TODO for destruído a PARTE também deverá ser destruída.

Composição

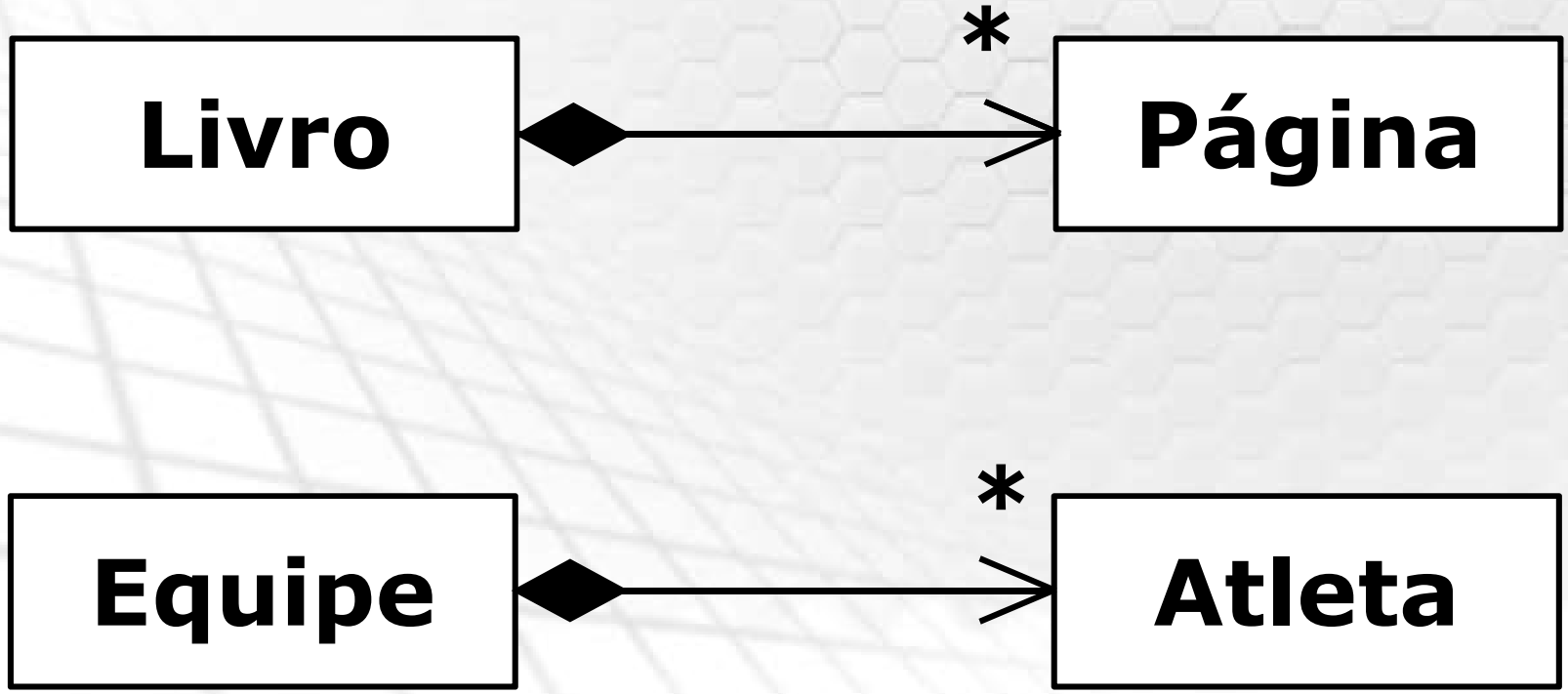
- Se {b_1, b_2, ..., b_n} deixam de existir, a também deixa de existir.

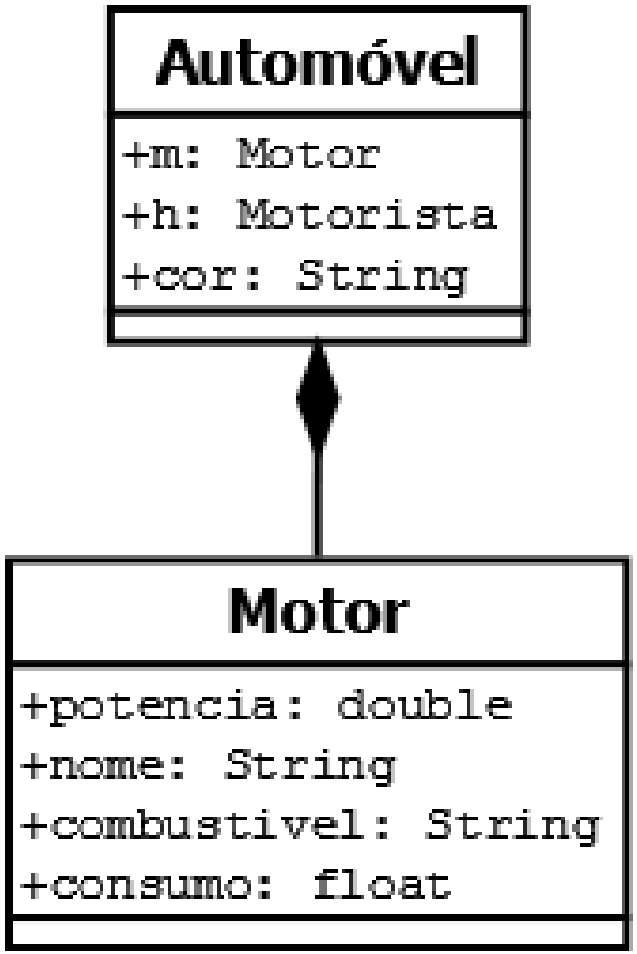


Composição: exemplos



Composição: exemplos




Automóvel
+m: Motor
+h: Motorista
+cor: String
Motor
+potencia: double
+nome: String
+combustivel: String
+consumo: float

```
public class Motor {
    private int potencia;

    public Motor(int potencia)
    { this.potencia = potencia;
    }
   public int getPotencia() {
        return this.potencia;
    }
    public void setPotencia(int potencia) {
        this.potencia = potencia;
    }
}
```

```
public class Automovel {
    private String modelo;
    private Motor motor;
    public Automovel(String modelo, int potenciaMotor) {
        this.modelo = modelo;
/* Um objeto do tipo Motor sendo instânciado dentro do construtor da classe Automovel sendo
assim será gerada uma associação do tipo composição, pois o objeto Motor existirá somente
enquanto o objeto Automovel existir. */
        this.motor = new Motor(potenciaMotor);
    }
    public String getModelo() {}
    public void setModelo(String modelo) {}
    public Motor getMotor() {}
     .....
}
```

```
public class TesteComposicao {

    public static void main(String[] args) {
        Automovel gol = new Automovel("Gol 1.0", 1000);
        System.out.println(gol.toString());
    }
}
```

Dependência

- **Dependência é caracterizado quando uma operação precisa de uma outra classe para ser executada.**

- **O seu significado semântico é "precisa de".**

Dependência

```
public class A {
  ...
  public m1(B b, ...) {

    // usa b aqui
  }
```

Dependência

```
public m2(...) {

  B b;

  // usa b aqui

}
```

Dependência

```
public m3(B b1, ...) {
  B b2;
  // usa b1 e b2 aqui
 }
 ...
} // fim classe
```

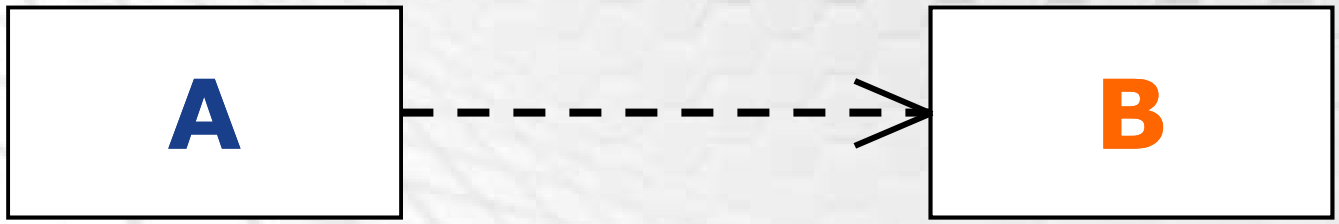
Dependência

- Se uma classe **A**:

1. Possui métodos que declaram um objeto **b** da classe **B**; ou
2. Recebem **b** como parâmetro.

Dependência

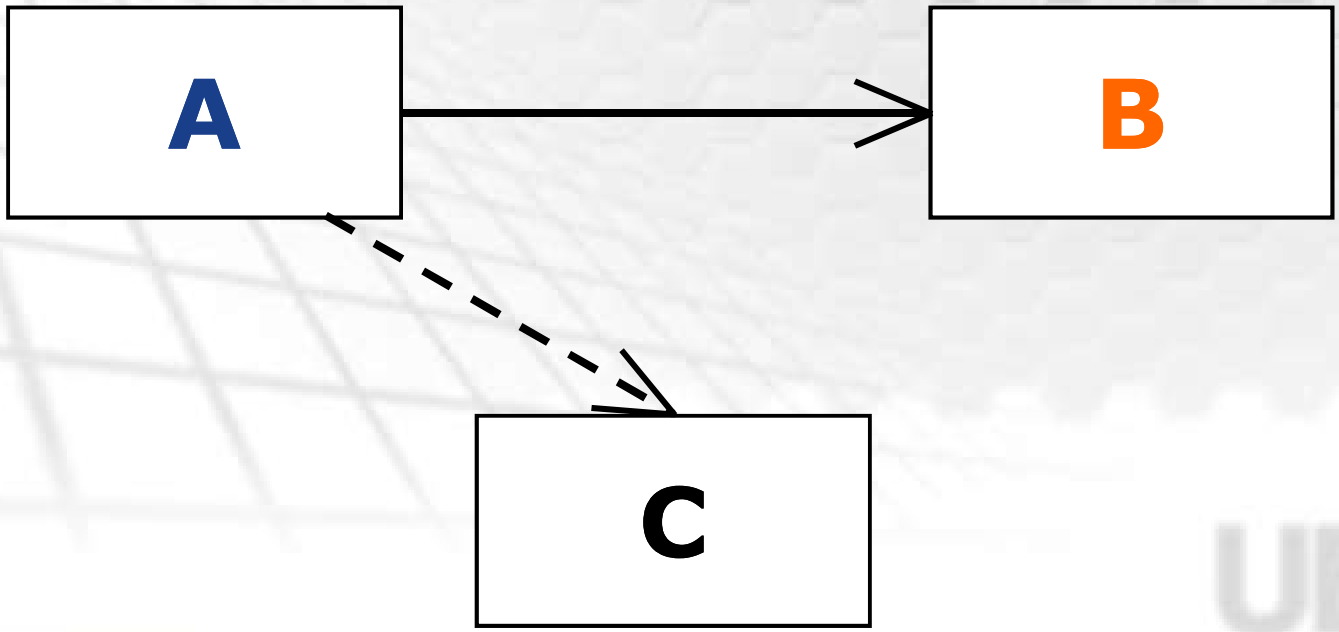
- Dizemos que A depende de B.



- Uma dependência não é uma associação!

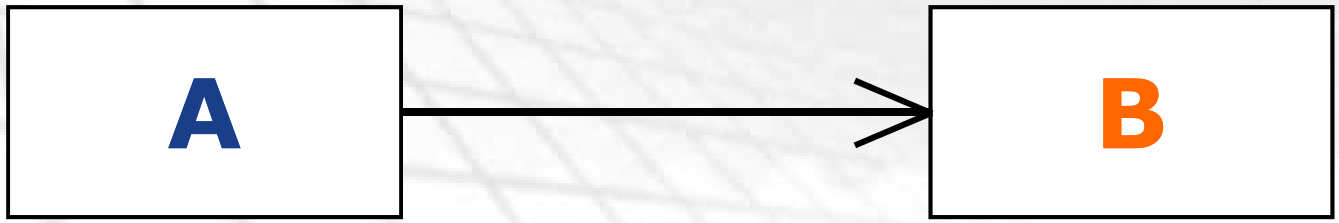
Dependência

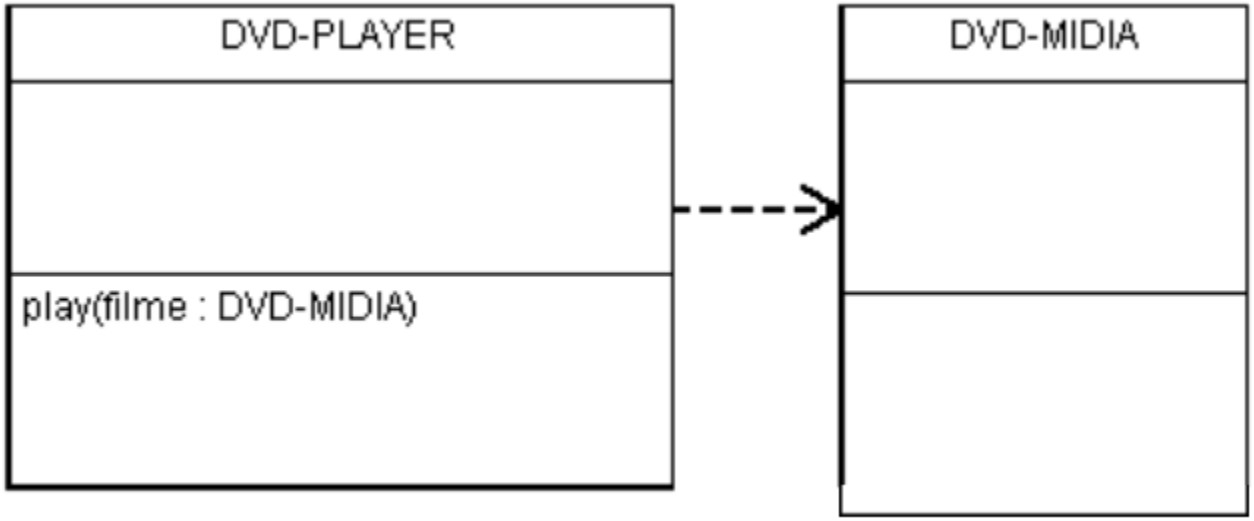
- **Uma classe pode estar associada e ter dependência.**



Dependência

- **Se uma classe está associada e depende de outra podemos mostrar apenas a associação.**






```
public class DVD-PLAYER
{
   public play(DVD-MIDIA filme)
   { }
}
```

```
public class DVD-MIDIA
{
}
```