Implementação - Agregação

Orientação a Objetos – DCC025

Prof. Edmar Welington Oliveira
edmar.oliveira@ufjf.edu.br

Universidade Federal de Juiz de Fora – UFJF
Departamento de Ciência da Computação - DCC

2011.3

Agregação/Composição

- Associação
  - É um relacionamento entre objetos.

- Agregação
  - Uma forma especial de associação entre o todo e suas partes
  - Tipos de agregação
    - Agregação Simples
    - Composição

Implementação – Agregação Simples

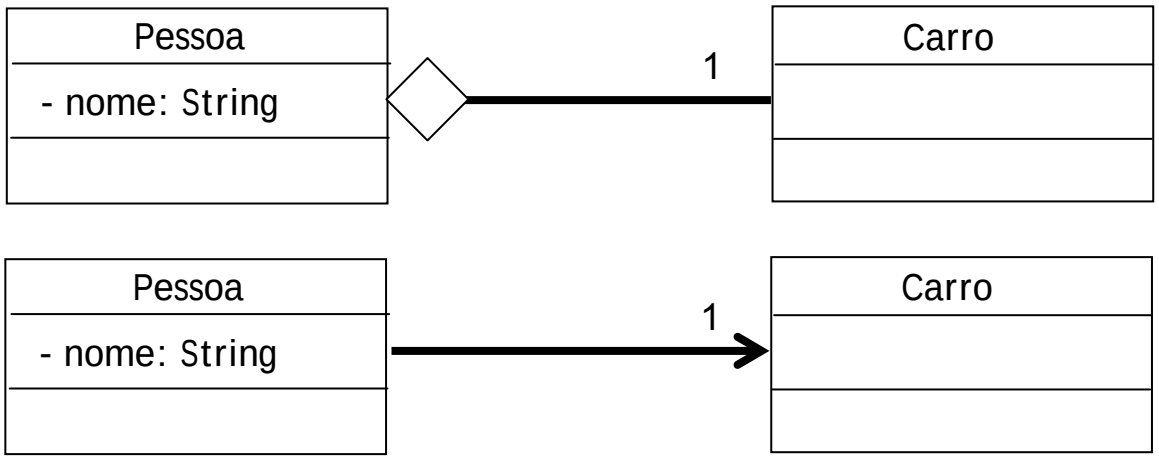
Agregação

- Agregação
  - Uma espécie vaga de associação na UML que sugere, fracamente, um relacionamento todo-parte. Ela não tem semântica significante na UML em comparação a uma associação simples.
  - Então porque ela é definida na UML?
    - Todas as pessoas pensam que é necessária
  - Diretriz
    - Conselho dos criadores da UML: não se preocupe em representar agregação simples em UML. Em vez disso, use composição quando apropriado.

Agregação e Associação

- Agregação e associação
  - Importante
    - Em termos de implementação, não há diferença entre agregação e um relacionamento de associação.

Agregação




```
3  public class Pessoa {
4
5      private String nomePessoa;
6      private Carro carro;
7
8      public Pessoa(String nome, Carro carro){
9          this.nomePessoa = nome;
10         this.carro = carro;
11     }
12 }
```

Agregação

```
3  public class Pessoa {
4
5      private String nomePessoa;
6      private Carro carro;
7
8      public Pessoa(String nome, Carro carro){
9          this.nomePessoa = nome;
10         this.carro = carro;
11     }
12 }
```

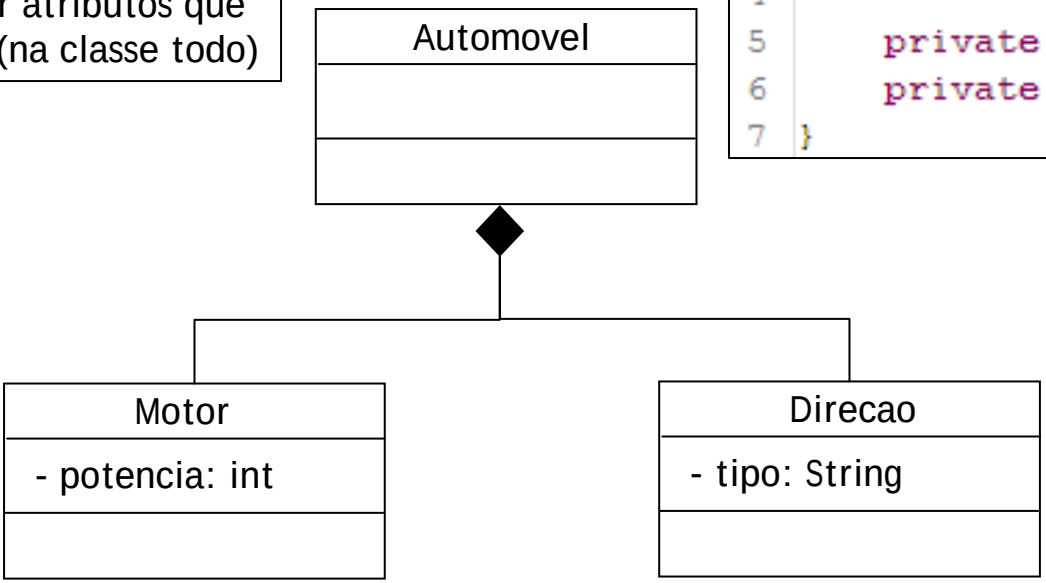
Observe que a classe Pessoa está recebendo um objeto do tipo Carro como parâmetro. Logo, este objeto foi criado/instanciado em um momento qualquer, diferente de Pessoa. Uma vez criado, ele foi passado para pessoa.

Observe, ainda, que se a pessoa "morrer" (for destruída no contexto do sistema), o carro continuará existindo – ou seja, ele foi instanciado e a sua instância ainda existirá, idependente do que acontecer com pessoa. Haverá, apenas, uma quebra da ligação entre pessoa e carro.

Implementação - Composição

Composição

No caso da composição, ainda existe a necessidade de se criar atributos que identifiquem as partes (na classe todo)




```
3  public class Automovel {
4
5      private Motor motor;
6      private Direcao direcao;
7  }
```

```
3  public class Motor {
4
5      private int potencia;
6  }
```

```
3  public class Direcao {
4
5      private String tipo;
6  }
```

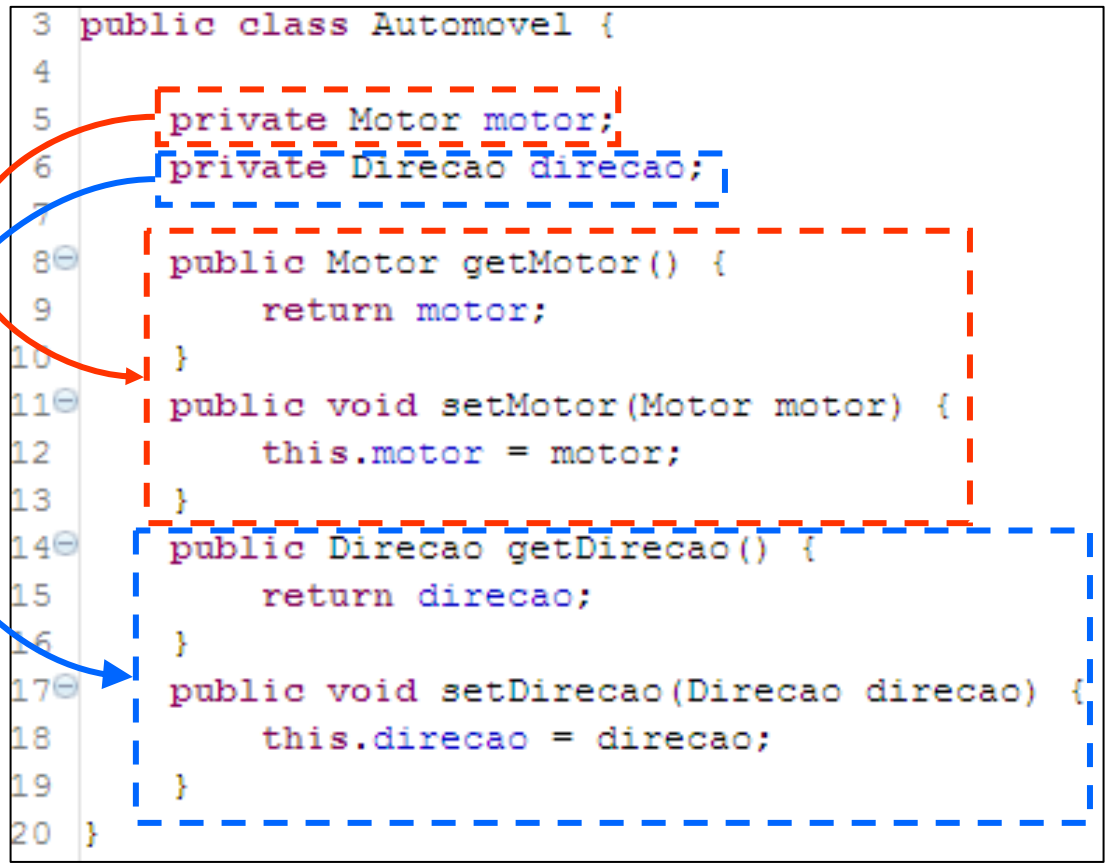
Composição

```
3  public class Motor {
4
5      private int potencia;
6
7      public int getPotencia() {
8          return potencia;
9      }
10
11     public void setPotencia(int potencia) {
12         this.potencia = potencia;
13     }
14 }
```

```
3  public class Direcao {
4
5      private String tipo;
6
7      public String getTipo() {
8          return tipo;
9      }
10
11     public void setTipo(String tipo) {
12         this.tipo = tipo;
13     }
14 }
```

Métodos set e get para os atributos das classes que formam as partes do todo

Composição



```
3  public class Automovel {
4
5      private Motor motor;
6      private Direcao direcao;
7
8      public Motor getMotor() {
9          return motor;
10     }
11     public void setMotor(Motor motor) {
12         this.motor = motor;
13     }
14     public Direcao getDirecao() {
15         return direcao;
16     }
17     public void setDirecao(Direcao direcao) {
18         this.direcao = direcao;
19     }
20 }
```

Até o momento, nada diferente em relação às associações vistas em momentos anteriores. Uma classe está associada a outras, possui atributos que as representam e métodos set e get (para alterá-los e recuperá-los)

Composição

Este método construtor da classe Automóvel chama o construtor de Motor sem parâmetro.

Este método construtor da classe Automóvel chama o construtor de Motor com parâmetro.

```
class Automovel {
        private Motor motor;

public Automovel(){
        motor = new Motor();
    }

public Automovel(int potencia){
        motor = new Motor(potencia);

    }
}
```

É no construtor que representamos a idéia de composição mais claramente

Composição

```
3  public class Principal {
4
5      public static void main(String args[]){
6
7          Automovel a1 = new Automovel(500);
8          Automovel a2 = new Automovel();
9
10         System.out.println(a1.getMotor().getPotencia());
11         System.out.println(a2.getMotor().getPotencia());
12     }
13 }
```

Observe que este construtor recebe um parâmetro que inicializa um outro construtor – ou seja, Automóvel recebe 500, que será utilizado no construtor de Motor. Já discutimos situações como esta, onde um construtor recebe argumentos que serão repassados para outro construtor (aula de implementação de associações).

O outro construtor não é diferente do que já foi visto

Composição

```
 3 public class Automovel {
 4
 5     private Motor motor;
 6     private Direcao direcao;
 7
 8     public Automovel(){
 9         motor = new Motor();
10     }
11
12     public Automovel(int potencia){
13         motor = new Motor(potencia);
14     }
```

Composição

```
3  public class Motor {
4
5      private int potencia;
6
7      public Motor(){
8          this.setPotencia(1000);
9      }
10
11     public Motor(int potencia){
12         this.setPotencia(potencia);
13     }
14
15     public int getPotencia() {
16         return potencia;
17     }
18
19     private void setPotencia(int potencia) {
20         this.potencia = potencia;
21     }
22 }
```

Composição

- Chamadas a métodos contrutores
  - Os métodos construtores das classes componentes (Motor,Direção) que fazem parte da classe composta (Automóvel), podem ser chamados de três maneiras diferentes:
    - Caso 1: chamadas nos construtores da classe que é composta;
    - Caso 2: chamadas em qualquer método da classe que é composta;
    - Caso 3: chamadas fora da classe que é composta.

Composição

Os construtores da classe motor são chamados dentro dos construtores da classe Automóvel.

```
 3 public class Automovel {
 4
 5     private Motor motor;
 6     private Direcao direcao;
 7
 8     public Automovel(){
 9         motor = new Motor();
10     }
11
12     public Automovel(int potencia){
13         motor = new Motor(potencia);
14     }
```

Composição

```
3  public class Automovel {
4
5      private Motor motor;
6      private Direcao direcao;
7
8      public Automovel(){
9          motor = new Motor();
10     }
11
12     public Automovel(int potencia){
13         motor = new Motor(potencia);
14     }
15
16     public void ligarAutomovel(){
17         motor = new Motor();
18     }
19
20     public void ligarAutomovel(int potencia){
21         motor = new Motor(potencia);
22     }
```

Os construtores são chamados em qualquer método da classe que é composta

Composição

```
3  public class Principal {
4
5      public static void main(String args[]){
6
7          Automovel auto = new Automovel();
8          Motor motor = new Motor();
9
10         auto.setMotor(motor);
```

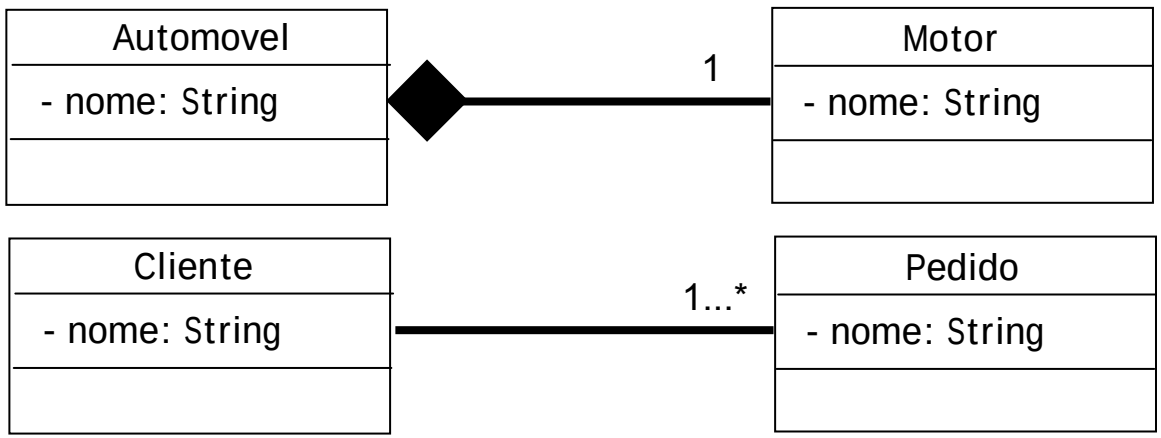
Observe que o construtor da classe Motor foi chamado de fora da classe Automóvel, dentro do método main()da classe. Ou seja, antes de usar o métodosetMotor() o objeto motor não tem nenhum vínculo com o objeto automóvel.

Associação e Composição

```
3  public class Automovel {
4
5      private Motor motor;
6      private Direcao direcao;
7
```

```
3  public class Cliente {
4
5      private Pedido pedido;
6
7  }
```

Associação e Composição



Associação e Composição

```
3  public class Automovel {
4
5      private Motor motor;
6      private Direcao direcao;
7
8      public Automovel(){
9          motor = new Motor();
10     }
11
12     public Automovel(int potencia){
13         motor = new Motor(potencia);
14     }
```

Composição

Associação

```
3  public class Cliente {
4
5      private Pedido pedido;
6
7      public Cliente(Pedido pedido){
8          this.pedido = pedido;
9      }
```

Associação e Composição

```
3  public class Automovel {
4
5      private Motor motor;
6      private Direcao direcao;
7
8      public Automovel(){
9          motor = new Motor();
10     }
11
12     public Automovel(int potencia){
13         motor = new Motor(potencia);
14     }
```

Composição

```
3  public class Empregado {
4
5      private Projeto projeto;
6
7      public Empregado(){
8          projeto = new Projeto();
9      }
10 }
```

Associação

Observe que, para este caso, não há diferenças de implementação entre composição e associação (ilustrada na igura por 1→1)

Agregação e Composição

Agregação e Composição

```
3  public class Pessoa {
4
5      private String nomePessoa;
6      private Carro carro;
7
8      public Pessoa(String nome, Carro carro){
9          this.nomePessoa = nome;
10         this.carro = carro;
11     }
12 }
```

```
3  public class Automovel {
4
5      private Motor motor;
6      private Direcao direcao;
7
8      public Automovel(){
9          motor = new Motor();
10     }
11
12     public Automovel(int potencia){
13         motor = new Motor(potencia);
14     }
```

Agregação e Composição

- Agregação
  - Um objeto B existe fora de outro objeto A. B é criado e, então, passado como um argumento para o contrutor de A.
    - Ex: Pessoa e Carro. O carro é criado em um contexto diferente e torna-se uma propriedade de pessoa
- Composição
  - Um objeto somente existe, ou somente faz sentido, dentro de outra, como uma parte. Ex: Pessoa e Cérebro. Não se cria um cérebro para, então, passá-lo para uma pessoa.

Resumo

Resumo Relacionamentos

- Dependência (seta tracejada):
  - É o relacionamento mais fraco da UML, simplesmente diz que um elemento client (que atira a seta) é semanticamente ou estruturalmente dependente do supplier (que recebe a seta). Exemplo: Produto e AliquotaImposto.

- Associação (linha contínua):
  - As duas classes são independentes e podem trabalhar juntas. A visibilidade destaca qual lado consegue enxergar as extremidades da associação, mas ambas as classes podem estabelecer o relacionamento. Exemplo: Cliente e Fornecedor.

Resumo Relacionamentos

- Agregação Simples (linha contínua com diamante branco):
  - A classe que possui o diamante branco controla a associação. Exemplo: Equipe e Pessoa. A equipe é quem pode estabelecer o relacionamento e pessoas são adicionadas à equipe. Uma pessoa por sí só não pode "entrar na equipe". É a equipe que manda no relacionamento. Contudo, as classes são independentes. Pessoas existem fora de equipes.

Resumo Relacionamentos

- Composição (linha contínua com diamante negro):
  - A composição é uma agregação mais forte. A classe que possui o diamante controla a associação, e além disso, a outra classe só pode existir associada à classe que tem o diamante e não pode estar associada a outras instâncias. Exemplo: Nota e ItemNota, Pedido e ItemPedido.

Implementação - Agregação

Orientação a Objetos – DCC025

Prof. Edmar Welington Oliveira
edmar.oliveira@ufjf.edu.br

Universidade Federal de Juiz de Fora – UFJF
Departamento de Ciência da Computação - DCC

2011.3