Departamento de Engenharia de Software
Centro de Educação Superior do Alto Vale do Itajaí - CEAVI
Universidade do Estado de Santa Catarina - UDESC

Programação orientada a objetos
Agregação e composição

Relacionamentos entre classes

- Classes podem se relacionar entre si, definindo um vínculo entre os objetos dessas classes.

Exemplos

- Um **cliente** possui um **endereço**.
- Uma **empresa** é composta por **funcionários**.
- Uma **moto** é um tipo de **veículo**.
- Um **restaurante** possui **pratos**.
- Uma **correspondência** possui um **remetente** e um **destinatário**.

Relacionamentos entre classes

- **Associação:** conexão entre classes.
- **Agregação e composição:** especialização de uma associação onde um todo é relacionado com suas partes (relacionamento "parte-de").
- **Dependência:** um objeto depende de alguma forma de outro (relacionamento de utilização).
- **Herança (generalização):** um dos princípios da orientação a objetos, permite a reutilização, uma nova classe pode ser definida a partir de outra já existente.
- **Realização:** um contrato que a classe segue (obrigação).

Relacionamentos entre classes

- **Associação:**

- **Agregação**

- **Composição:**

- **Dependência:**

- **Herança (generalização):**

- **Realização:**

Relacionamento todo-parte

- A agregação e a composição definem relacionamentos do tipo **todo-parte,** onde uma das classes representa o todo e a outra representa suas partes.
- Neste sentido, uma instância da classe todo possui uma ou mais instâncias da classe parte. A instância da classe parte complementa as informações da classe todo, de modo que o todo não é completo sem as suas partes.

- **Exemplos**
  - Um veículo (todo) é composto por quatro rodas (parte).
  - Um computador (todo) possui um teclado, um mouse e um monitor (partes).
  - Uma lista de compras (todo) possui uma lista de itens a comprar (parte).
  - Uma empresa (todo) é composta por departamentos (parte).
  - Um livro (todo) é composto por capítulos (parte).
  - Um capítulo do livro (todo) é composto por páginas (parte).

Relacionamento todo-parte

Forma de identificar um relacionamento todo-parte (janela/botão)

- O relacionamento é descrito com uma frase "parte de"?
  - Um botão é "parte de" uma janela.
- Algumas operações no todo são automaticamente aplicadas a suas partes?
  - Mover a janela, (implica em) mover o botão.
- Alguns valores de atributos são propagados do todo para todos ou algumas de suas partes?
  - A fonte da janela é Arial, a fonte do botão Arial.
- Existe uma assimetria inerente no relacionamento onde uma classe é subordinada a outra?
  - Uma botão É parte de uma janela, uma janela NÃO É parte de um botão.

Relacionamento todo-parte

Forma de identificar um relacionamento todo-parte (carro/porta)

- O relacionamento é descrito com uma frase "parte de"?
  - Uma porta é "parte de" um carro.
- Algumas operações no todo são automaticamente aplicadas a suas partes?
  - Mover o carro, (implica em) mover a porta.
- Alguns valores de atributos são propagados do todo para todos ou algumas de suas partes?
  - O carro é azul, a porta é azul.
- Existe uma assimetria inerente no relacionamento onde uma classe é subordinada a outra?
  - Uma porta É parte de um carro, um carro NÃO É parte de uma porta.

Agregação x composição

Agregação

- A agregação é usada para relacionar um todo com as suas partes, de modo que ambas as entidades podem existir de forma independente à outra.
- Uma entidade parte pode estar relacionada com mais de uma entidade todo.

Composição

- A composição é usada para relacionar um todo com as suas partes, de modo que a entidade parte só existe em função do relacionamento que possui com a entidade todo. Caso a entidade todo seja destruída, suas partes também são destruídas.
- Uma entidade parte pode estar relacionada com apenas uma entidade todo.

Agregação x composição

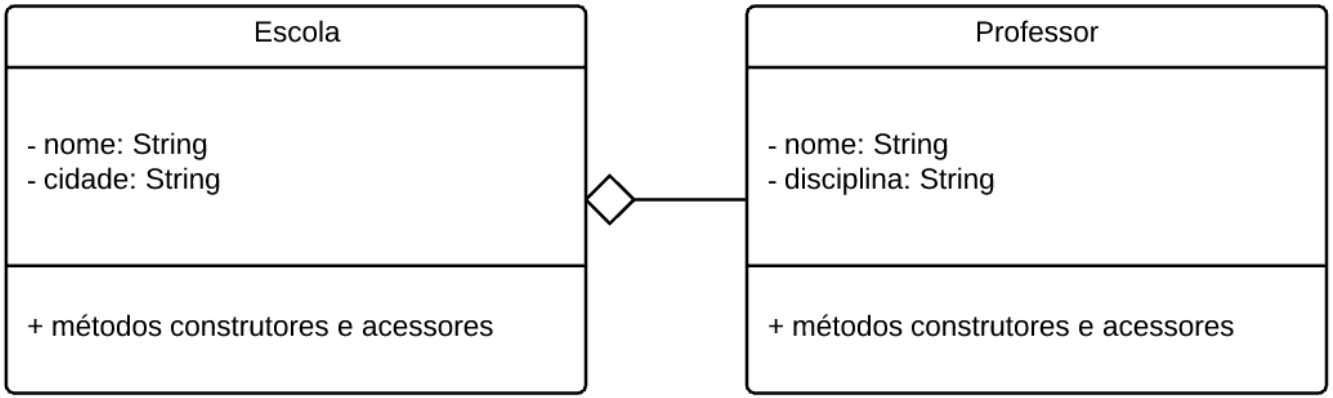
Exemplos

- Uma escola possui vários professores.
  - **Agregação,** pois um professor pode existir fora do relacionamento com a escola.
  - **Agregação,** pois um professor pode trabalhar em duas escolas ao mesmo tempo, sua entidade se relaciona com dois objetos diferentes do todo.

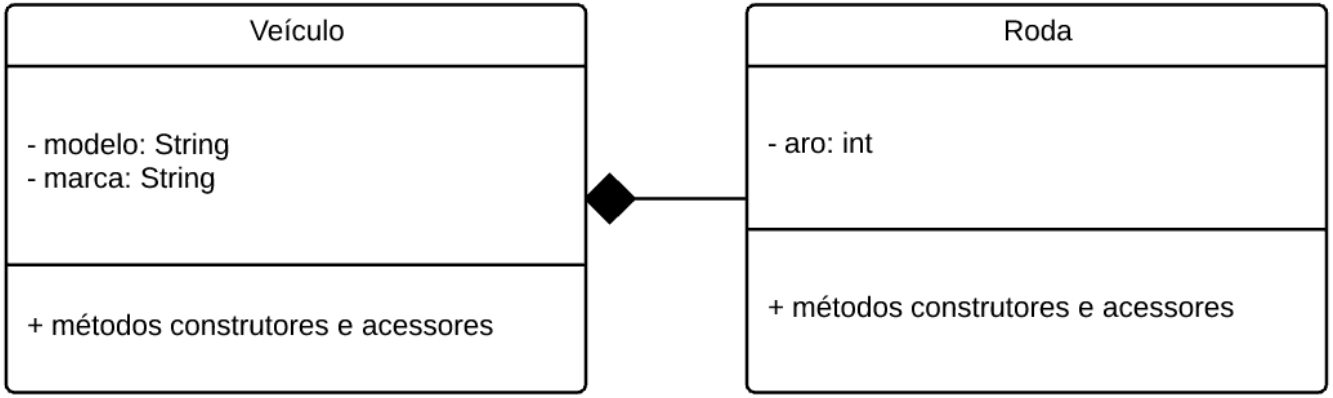
- Um veículo possui quatro rodas.
  - **Composição,** pois uma roda não existe sem estar vinculada ao veículo.
  - **Composição,** pois uma roda não pode estar em dois veículos ao mesmo tempo, sua entidade se relaciona a apenas um objeto do todo.

Representação na UML

Agregação

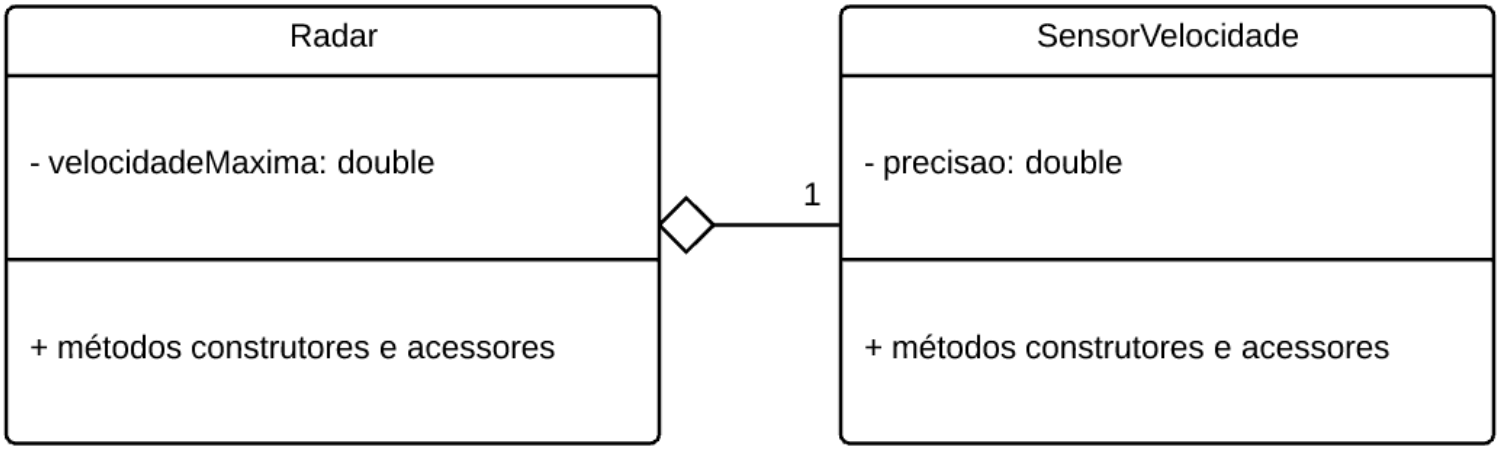


Composição



Implementação – agregação

- Quando a agregação possui multiplicidade 1, ela é implementada como uma associação simples de multiplicidade 1.
- Em geral, o atributo (ou lista) que implementa a agregação fica no objeto todo.



Implementação – agregação

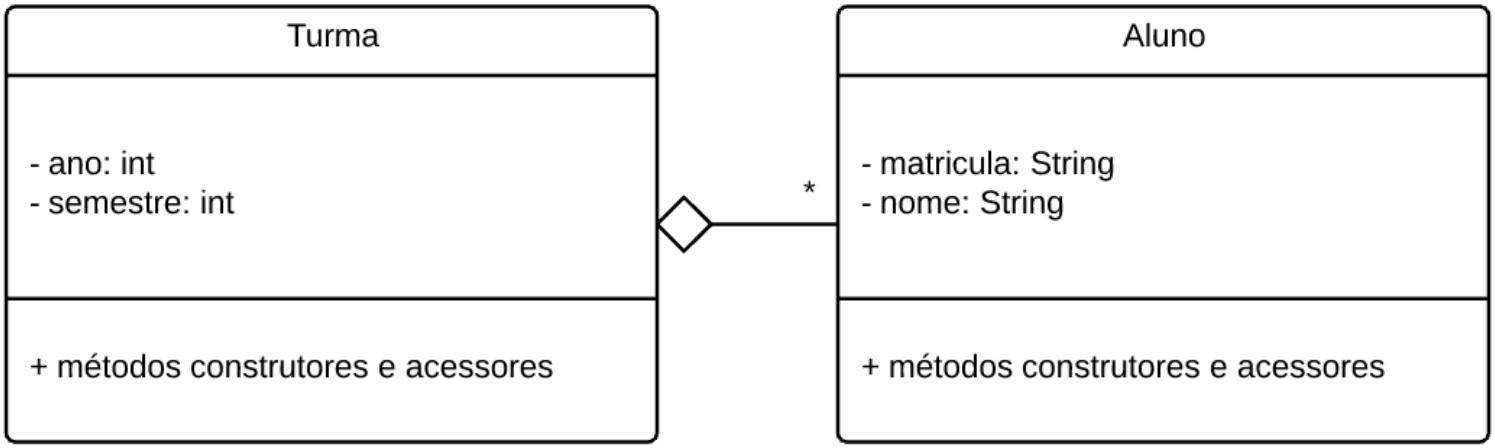
- Quando a agregação possui multiplicidade 1, ela é implementada como uma associação simples de multiplicidade 1.
- Em geral, o atributo (ou lista) que implementa a agregação fica no objeto todo.

```
public class Radar {
  private double velocidadeMaxima;
  private SensorVelocidade sensor;

  public SensorVelocidade getSensor() {
      return sensor;
  }

  public void setSensor(SensorVelocidade s){
      this.sensor = s;
  }
}
```

```
public class SensorVelocidade {
  private double precisao;

  public double getPrecisao() {
      return precisao;
  }

  public void setPrecisao(double precisao) {
      this.precisao = precisao;
  }

}
```

Implementação – agregação

- Quando a agregação possui multiplicidade *, ela é implementada como uma associação simples de multiplicidade *.
- Em geral, o atributo (ou lista) que implementa a agregação fica no objeto todo.



Implementação – agregação

- Quando a agregação possui multiplicidade *, ela é implementada como uma associação simples de multiplicidade *.
- Em geral, o atributo (ou lista) que implementa a agregação fica no objeto todo.

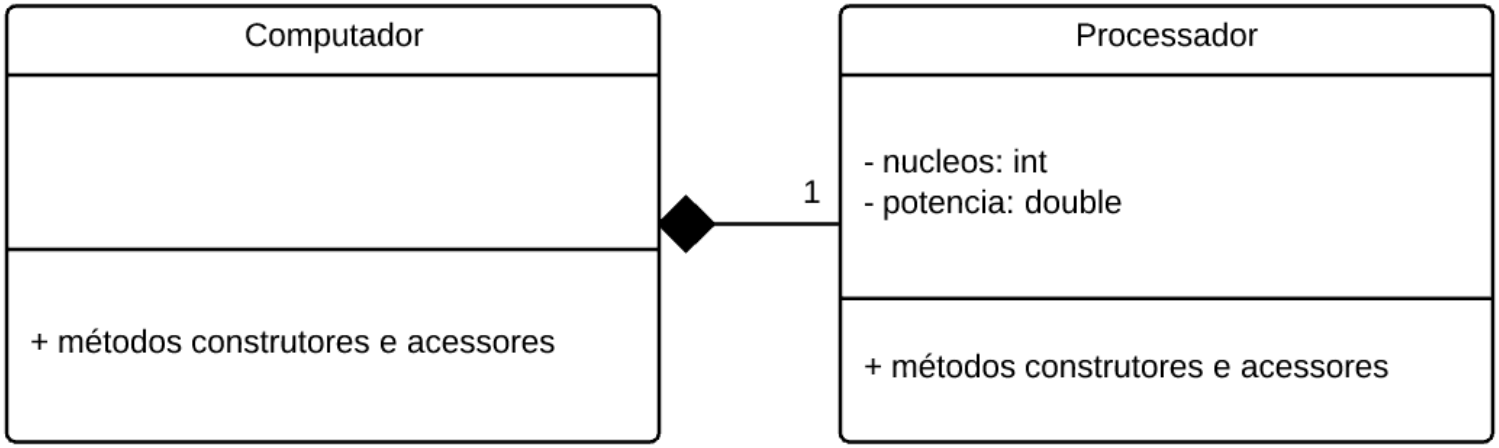
```
public class Turma {
  private int ano;
  private int semestre;
  private List<Aluno> alunos =
                      new ArrayList<Aluno>();

  public List<Aluno> getAlunos() {
      return alunos;
  }

  public void setAlunos(List<Aluno> alunos){
      this.alunos = alunos;
  }
}
```

```
public class Aluno {

  private String matricula;
  private String nome;

  public String getMatricula() {
      return matricula;
  }

  public void setMatricula(String mat) {
      this.matricula = mat;
  }

}
```

Na classe todo podem ser implementados métodos para a manipulação das partes (addAluno, removeAluno, etc.).

Implementação – composição

- Quando a composição possui multiplicidade 1, ela é implementada como uma associação simples de multiplicidade 1.
- Em geral, as operações de criação e destruição do objeto parte ficam no todo.



Implementação – composição

- Quando a composição possui multiplicidade 1, ela é implementada como uma associação simples de multiplicidade 1.
- Em geral, as operações de criação e destruição do objeto parte ficam no todo.

```
public class Computador {

private Processador processador;

  public void addProcessador(
           int nucleos, double potencia) {

      processador = new Processador();

      processador.setNucleos(nucleos);
      processador.setPotencia(potencia);

  }

  public void removeProcessador() {
      this.processador = null;
  }


}
```

```
public class Processador {
  private int nucleos;
  private double potencia;

  public int getNucleos() {
      return nucleos;
  }

  public void setNucleos(int nucleos) {
      this.nucleos = nucleos;
  }

  public double getPotencia() {
      return potencia;
  }

  public void setPotencia(double potencia) {
      this.potencia = potencia;
  }
}
```

Implementação – composição

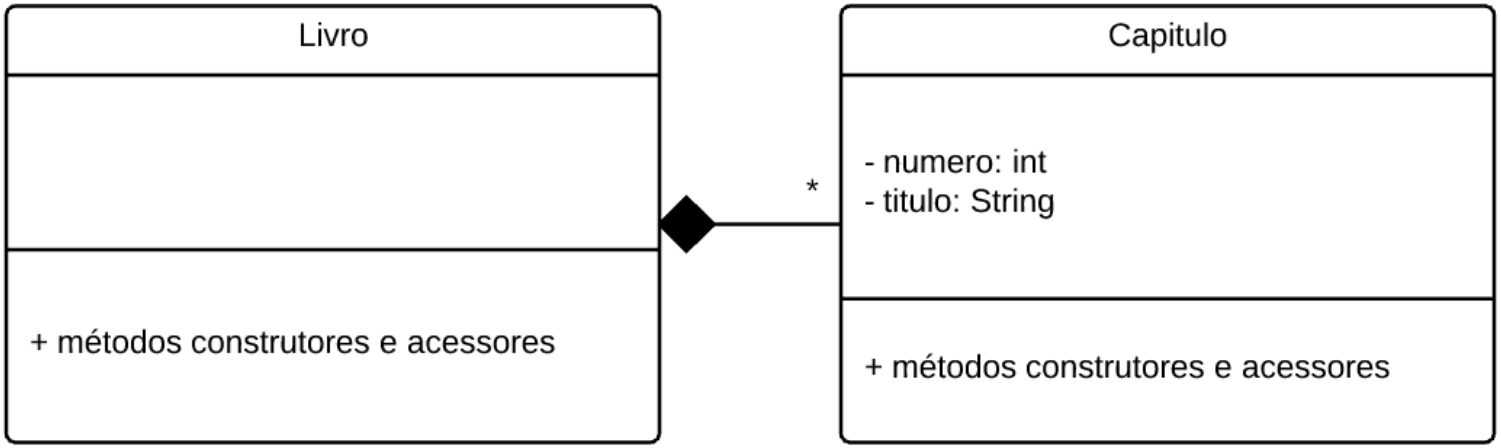
O processador é criado dentro da classe Computador pois só existe dentro da mesma e só pode ser vinculado a um computador (conceito de composição). Repare que a classe computador não possui o método **setProcessador**. Esta é uma boa prática, pois impede que o processador de um computador seja associado a outro computador (conceito de composição).

```
public class Computador {

private Processador processador;

  public void addProcessador(
           int nucleos, double potencia) {

      processador = new Processador();

      processador.setNucleos(nucleos);
      processador.setPotencia(potencia);

  }

  public void removeProcessador() {
      this.processador = null;
  }


}
```

```
public class Processador {
  private int nucleos;
  private double potencia;

  public int getNucleos() {
      return nucleos;
  }

  public void setNucleos(int nucleos) {
      this.nucleos = nucleos;
  }

  public double getPotencia() {
      return potencia;
  }

  public void setPotencia(double potencia) {
      this.potencia = potencia;
  }
}
```

Implementação – composição

- Quando a composição possui multiplicidade *, ela é implementada como uma associação simples de multiplicidade *.
- Em geral, as operações de criação e destruição do objeto parte ficam no todo.



Implementação – composição

- Quando a composição possui multiplicidade *, ela é implementada como uma associação simples de multiplicidade *.
- Em geral, as operações de criação e destruição do objeto parte ficam no todo.

```
public class Livro {

    private List<Capitulo> caps =
                new ArrayList<Capitulo>();

    public void addCapitulo(
                int numero, String titulo) {

        Capitulo c = new Capitulo();

        c.setNumero(numero);
        c.setTitulo(titulo);
        caps.add(c);

    }

}
```

```
public class Capitulo {

    private int numero;
    private String titulo;

    public int getNumero() {
        return numero;
    }


    public void setNumero(int numero) {
        this.numero = numero;
    }

    //Set e get para o título...

}
```

Implementação – composição

O capítulo é criado dentro da classe Livro. Com isso se garante que o capítulo esteja vinculado a apenas um livro e não exista fora do mesmo, ele é destruído junto com o livro (composição). Repare que a classe Livro não possui o método **setCapitulos** e **getCapitulos**. Caso necessário eles podem ser criados, permitindo ao programador quebrar o conceito da composição.

```
public class Livro {

    private List<Capitulo> caps =
                new ArrayList<Capitulo>();

    public void addCapitulo(
                int numero, String titulo) {

        Capitulo c = new Capitulo();

        c.setNumero(numero);
        c.setTitulo(titulo);
        caps.add(c);

    }

}
```

```
public class Capitulo {

    private int numero;
    private String titulo;

    public int getNumero() {
        return numero;
    }


    public void setNumero(int numero) {
        this.numero = numero;
    }

    //Set e get para o título...

}
```

Referências

DEITEL, H. M. **Java: como programar**. H. M Deitel e P. J. Deitel - 8a ed. Porto Alegre: Prentice-Hall, 2010.

Leitura complementar

TutorialsPoint Java (http://www.tutorialspoint.com/java).