

Linguagem de Programação 1

Associação entre classes

Prof. Fábio José Rodrigues Pinheiro

NOVEMBRO DE 2021



**INSTITUTO
FEDERAL**

Catarinense

Campus
Videira

Diagrama de classes UML

- Linguagem (notação gráfica + semântica) para especificar, construir e documentar os artefatos dos sistemas
- Formas de uso para a UML
 - **Como rascunho** – diagramas incompletos e informais (geralmente escritos em um quadro branco) para explorar partes difíceis do problema
 - **Como projeto de software** – diagramas detalhados que podem ser usados para gerar código (engenharia direta) ou foram gerados para entender um código existente (engenharia reversa)



Dado
valorDaFace

Perspectiva conceitual

Dado
- valorDaFace : int
+ obterValorDaFace() : int

Perspectiva de implementação

- **Perspectiva conceitual** – diagramas são interpretados como descrevendo coisas em uma situação no mundo real
- **Perspectiva de implementação** – diagramas descrevem implementações de software em uma tecnologia particular



■ Diagramas estruturais

- Diagrama de classes
- Diagrama de objetos
- Diagrama de componentes
- Diagrama de implantação

■ Diagramas comportamentais

- Diagrama de caso de uso
- Diagrama de sequência
- Diagrama de colaboração
- Diagrama de atividades



■ Diagramas estruturais

- Diagrama de classes
- Diagrama de objetos
- Diagrama de componentes
- Diagrama de implantação

■ Diagramas comportamentais

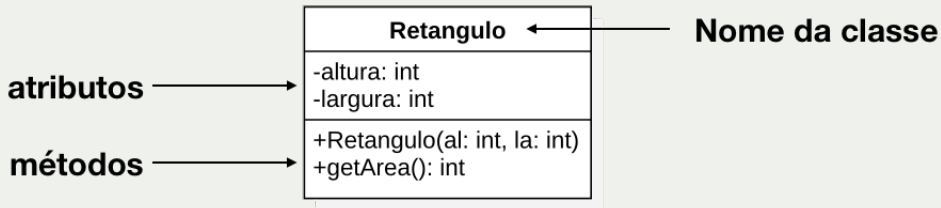
- Diagrama de caso de uso
- Diagrama de sequência
- Diagrama de colaboração
- Diagrama de atividades



- **Classes** podem ser vistas como um **agrupamento de objetos de um mesmo tipo**, porém cada objeto é um exemplo único de um determinado grupo
- **Diagrama de classes** permite visualizar as classes que irão compor o sistema, com seus **atributos** e **métodos**, bem como em demonstrar como essas **classes se relacionam**

Um **diagrama de classes** é composto por suas **classes** e pelas **associações** existentes entre elas





- A representação de atributos e métodos é opcional
- Métodos triviais podem ser omitidos (p. ex: getAltura())
- Modificadores de visibilidade (+, -, #)
- Membros estáticos ficam sublinhados

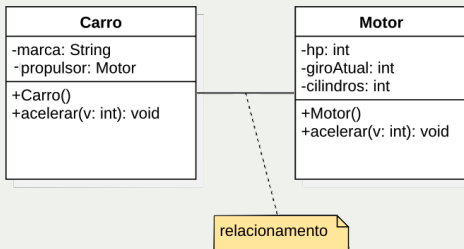


Associação entre classes

Associação entre classes

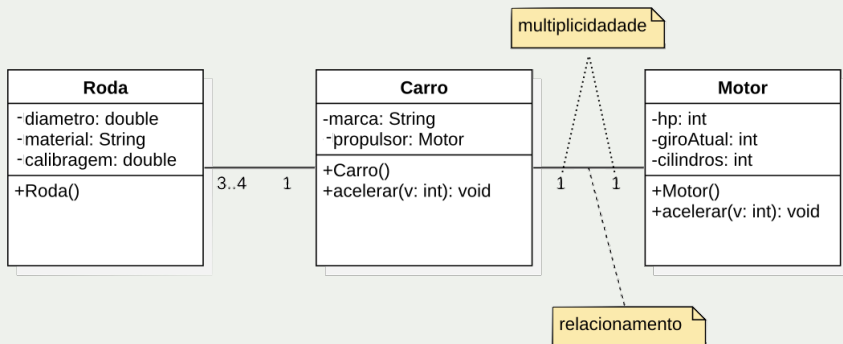
- Relacionamento entre classes que permite o compartilhamento de informação e colaboração para a execução de computação
- Descreve o vínculo entre objetos de uma ou mais classes
- A classe **Carro** possui um relacionamento com a classe **Motor**, pois um objeto da classe Carro contém 1 objeto da classe Motor

```
1 public class Carro{  
2     private String marca;  
3     private Motor propulsor;  
4  
5     public Carro(String m, Motor mo)  
6     {  
7         this.marca = m;  
8         this.propulsor = mo;  
9     }  
}
```



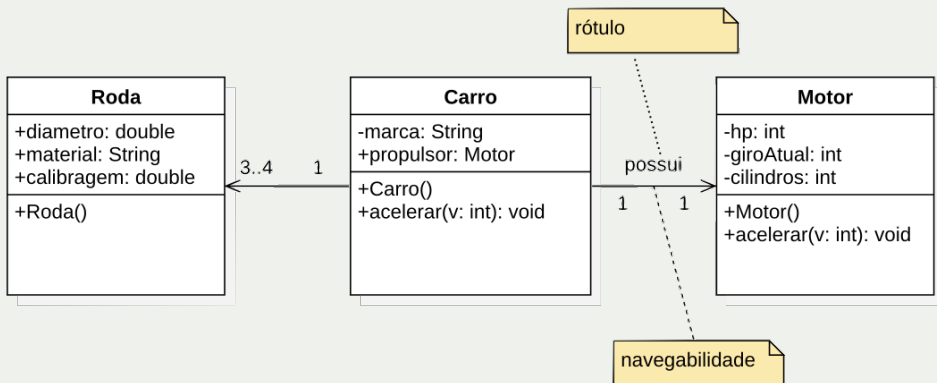
Associação bidirecional

multiplicidade	
0..*	Zero ou mais
1..*	Um ou mais
*	Muitos
1	Exatamente um
3..4	De 3 a 4



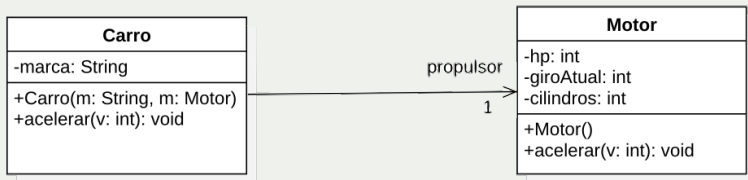
Associação unidirecional

- Indica o sentido em que as informações são transmitidas, isto é, indica o sentido em que os métodos poderão ser disparados
- Para acelerar, objeto da classe Carro invoca método do objeto da classe Motor



Associação: notação textual vs linha de associação

- Uso da notação de associação para indicar que Carro possui uma referência para uma instância de Motor



- Notação textual de atributo



- Para representar a relação entre o *objeto-todo* e *objetos-parte*
- Agregação ocorre quando uma **classe é uma coleção** ou contêiner **de outras classes**, porém o **ciclo de vida da classe contida não depende** fortemente da classe que a contém
- Quando o objeto contido faz sentido existir no sistema mesmo sem ser parte do objeto que o contém
- Serve para **indicar a obrigatoriedade de uma complementação das informações de um objeto-todo por seus objetos-parte**, quando esse for consultado



Associação do tipo Agregação

- Para representar a relação entre o *objeto-todo* e *objetos-parte*
- Agregação ocorre quando uma **classe é uma coleção** ou contêiner **de outras classes**, porém o **ciclo de vida da classe contida não depende** fortemente da classe que a contém
- Quando o objeto contido faz sentido existir no sistema mesmo sem ser parte do objeto que o contém
- Serve para **indicar a obrigatoriedade de uma complementação das informações de um objeto-todo por seus objetos-parte**, quando esse for consultado

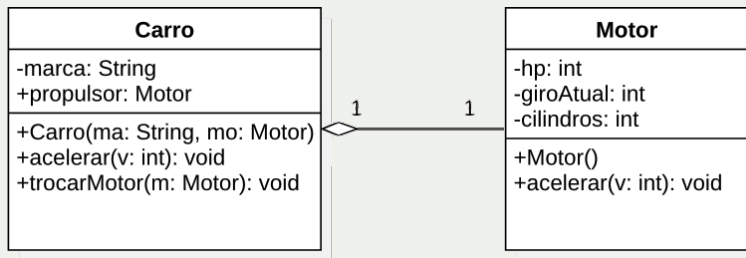
Agregação pode ser substituída por uma associação simples

Um dos criadores da UML até sugere nunca usar Agregação, devendo o analista se preocupar mais em representar a Composição



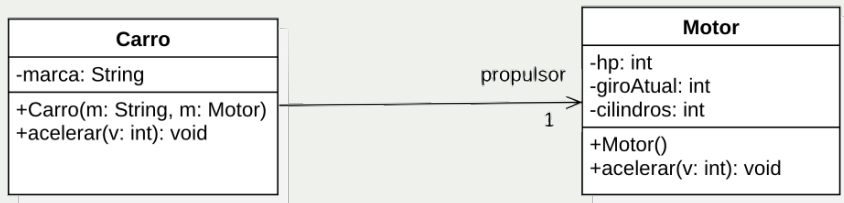
Associação do tipo Agregação

- Quando o objeto contido faz sentido existir no sistema mesmo sem ser parte do objeto que o contém
- Um **carro** possui um **motor**. Se o carro deixar de existir, o motor poderia ser colocado em outro carro



Associação do tipo Agregação

- Quando o objeto contido faz sentido existir no sistema mesmo sem ser parte do objeto que o contém
- Um **carro** possui um **motor**. Se o carro deixar de existir, o motor poderia ser colocado em outro carro



Associação do tipo Agregação

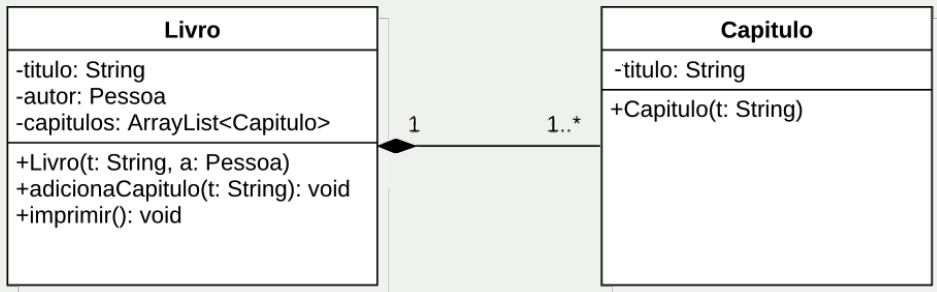
```
1 public class Carro{
2     private String marca;
3     private Motor propulsor;
4
5     public Carro(String m, Motor mo){
6         this.marca = m;
7         this.propulsor = mo;
8     }
9
10    public void acelerar(int valor){
11        this.propulsor.acelerar(valor);
12    }
13
14    public void trocarMotor(Motor mo){
15        this.propulsor = mo;
16    }
17 }
```



- Vínculo mais forte entre o objeto-todo e o objeto-parte, demonstrando que os objeto-partes têm de estar associados a um único objeto-todo
- Objetos-parte não podem ser destruídos por um objeto diferente do objeto-todo ao qual estão relacionados
- Quando o **objeto contido** (objeto-parte) **não faz sentido existir** no sistema **se o objeto que o contém** (objeto-todo) **for excluído**



- Um **livro** é composto por diversos **capítulos**. Se destruirmos um **livro**, não faria mais sentido os capítulos existirem



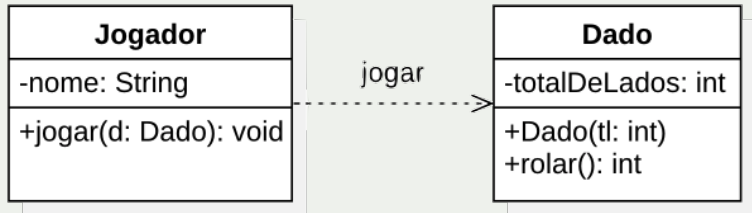
Associação do tipo Composição

```
1 public class Livro{
2     private String titulo;
3     private ArrayList<Capitulo> capitulos;
4     private Pessoa autor;
5
6     public Livro(String t, Pessoa a){
7         this.titulo = t;
8         this.capitulos = new ArrayList<>();
9         this.autor = a;
10    }
11
12    public void adicionaCapitulo(String titulo){
13        this.capitulos.add(new Capitulo(titulo));
14    }
15 }
```



Associação do tipo Dependência

- Para indicar a dependência de uma classe em relação à outra
- Qualquer alteração na classe que é invocada pode resultar em uma quebra da classe que faz a invocação
- Uso temporário



```
1 public void jogar(Dado d){  
2     int res = d.rolar();  
3     ...  
4 }
```



Exercício

Sistema para gestão de Agenda telefônica

- Aplicação que permita ao usuário gerir sua agenda de contatos
 - Adicionar, Remover, Atualizar
 - Listar dados de um contato, Listar todos contatos
- Para cada contato deve-se guardar o nome, sobrenome, data de nascimento, telefone(s) e e-mail(s)
- Para cada telefone ou email é necessário ter um rótulo. Ex:
 - **rótulo:** comercial
 - **valor:** [email@empresa.com](#)
- Ao cadastrar um email, deve-se garantir que é um endereço válido
- Ao listar um telefone, deve-se aplicar uma máscara para facilitar a leitura
 - Ex: +55049998761234 → +55 (49) 9 9876-1234



Exercício

Faça um **diagrama de classes** que contenha somente os nomes das **classes** que serão necessárias, **bem como seus relacionamentos**

- Lembre-se:
 - **Cada classe deve ser responsável por uma pequena parte** da funcionalidade de um software
 - A **responsabilidade** deve **estar completamente encapsulada** dentro da classe



Quais classes precisarão ser modeladas?



Quais classes precisarão ser modeladas?

- Principal e Pessoa?



Quais classes precisarão ser modeladas?

- Principal e Pessoa?

- Onde ficará a lista de pessoas?

- A classe `Principal` é só a interface com o usuário e ela será substituída quando fizermos uma versão com interface gráfica



Quais classes precisarão ser modeladas?

- Principal e Pessoa?
 - Onde ficará a lista de pessoas?
 - A classe `Principal` é só a interface com o usuário e ela será substituída quando fizermos uma versão com interface gráfica
- Principal, Agenda e Pessoa?



Quais classes precisarão ser modeladas?

■ Principal e Pessoa?

■ Onde ficará a lista de pessoas?

- A classe `Principal` é só a interface com o usuário e ela será substituída quando fizermos uma versão com interface gráfica

■ Principal, Agenda e Pessoa?

■ Como garantirá que é um email válido?

- A classe `Pessoa` deve ser responsável somente por dados de pessoa



Quais classes precisarão ser modeladas?

- Principal e Pessoa?

- Onde ficará a lista de pessoas?

- A classe `Principal` é só a interface com o usuário e ela será substituída quando fizermos uma versão com interface gráfica

- Principal, Agenda e Pessoa?

- Como garantirá que é um email válido?

- A classe `Pessoa` deve ser responsável somente por dados de pessoa

- Principal, Agenda, Pessoa e Email?



Quais classes precisarão ser modeladas?

■ Principal e Pessoa?

■ Onde ficará a lista de pessoas?

- A classe `Principal` é só a interface com o usuário e ela será substituída quando fizermos uma versão com interface gráfica

■ Principal, Agenda e Pessoa?

■ Como garantirá que é um email válido?

- A classe `Pessoa` deve ser responsável somente por dados de pessoa

■ Principal, Agenda, Pessoa e Email?

■ Como listará o telefone formatado?



Quais classes precisarão ser modeladas?

- Principal e Pessoa?
 - Onde ficará a lista de pessoas?
 - A classe `Principal` é só a interface com o usuário e ela será substituída quando fizermos uma versão com interface gráfica
- Principal, Agenda e Pessoa?
 - Como garantirá que é um email válido?
 - A classe `Pessoa` deve ser responsável somente por dados de pessoa
- Principal, Agenda, Pessoa e Email?
 - Como listará o telefone formatado?
- Principal, Agenda, Pessoa, Email e Telefone ⇐



Agora é Java!

Classes `java.time.LocalDate` e `java.time.LocalDateTime`

■ `java.time.LocalDate`

```
1 import java.time.Month;
2 //.....
3 // Dia de hoje
4 LocalDate hoje = LocalDate.now();
5 // Criando objeto com uma data específica
6 LocalDate dataEspecifica = LocalDate.of(2020, Month.JANUARY, 10);
```

■ `java.time.LocalDateTime`

```
1 import java.time.format.DateTimeFormatter;
2 //.....
3 String dhF;
4
5 LocalDateTime dataHora = LocalDateTime.of(2019, 04, 22, 13, 30, 00);
6 dhF = dataHora.format(DateTimeFormatter.ofPattern("yyyyMMdd'T'HHmmss"));
7 System.out.printf(dhF); // Será impresso: 20190422T133000
8
9 // para obter a data e hora no momento que essa linha foi executada
10 LocalDateTime agora = LocalDateTime.now();
```



Expressões regulares

Metacaractere	Descrição
.	Casa com qualquer caractere
[]	Lista de caracteres. Ex: <code>n[aã]</code> o casa com <i>não</i> e <i>nao</i> . É possível definir intervalos <code>[0-2]</code> casa com 0, 1 ou 2
^	Casa com o começo da cadeia de caracteres
\$	Casa com o final da cadeia de caracteres
*	Casa o elemento precedente zero ou mais vezes. Ex: <code>ab*c</code> casa com <code>ac</code> , <code>abc</code> , <code>abbbbc</code>
+	Casa o elemento precedente um ou mais vezes
?	Torna o elemento precedente opcional
\	Para escapar outros metacaracteres.
{n,m}	De n até m. Ex: <code>{2,}</code> pelo menos 2 caracteres
()	Grupo. Ex: <code>(www\.)?ifc.edu.br</code> casa com <code>www.ifc.edu.br</code> e <code>ifc.edu.br</code>
\w	Corresponde a <code>[a-zA-Z_0-9]</code>




- Uma expressão regular provê uma forma concisa e flexível de identificar cadeias de caracteres, palavras ou padrões de caracteres.


```
1 String emailER =  
2     "^([\\w-\\+]+(\\. [\\w]+)*)@[\\w-]+(\\. [\\w]+)*(\\. [a-z]{2,})$";  
3  
4 String email = "meu.email@dominio.com.br";  
5 System.out.println("Email válido? " + email.matches(emailER));
```

- <https://docs.oracle.com/javase/tutorial/essential/regex/>
- <https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html>



 CRAIG LARMAN
UTILIZANDO UML E PADRÕES
3a. Edição - Editora Bookman, 2007

 MARTIN FOWLER
UML ESSENCIAL : UM BREVE GUIA PARA A LINGUAGEM-PADRÃO DE MODELAGEM DE OBJETOS
3a. Edição - Editora Bookman, 2005

 EDUARDO BEZERRA
PRINCÍPIOS DE ANÁLISE E PROJETO DE SISTEMAS COM UML
2a. Edição - Editora Elsevier, 2007

