

Diagrama de classes UML e Associação entre classes

POO29004 – Engenharia de Telecomunicações

Prof. Emerson Ribeiro de Mello

<http://docente.ifsc.edu.br/mello/poo>

20 DE MARÇO DE 2020



**INSTITUTO
FEDERAL**
Santa Catarina

Câmpus
São José

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

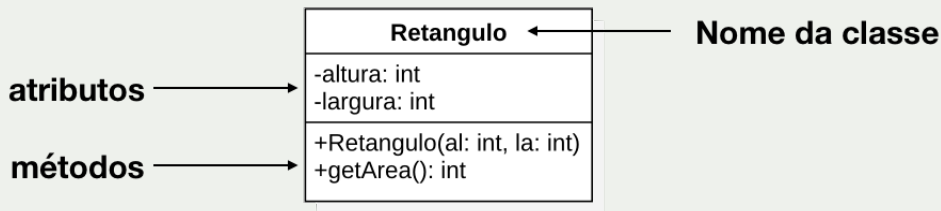
- Diagrama de caso de uso
- Diagrama de sequência
- Diagrama de colaboração
- Diagrama de atividades



- **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



- **DrawIO** – <https://www.draw.io/>
- **Lucid Chart** – <https://www.lucidchart.com/>
- **Dia** – <http://dia-installer.de/>
- **Umbrello** – <https://umbrello.kde.org/>
- **StarUML** – <http://staruml.io>

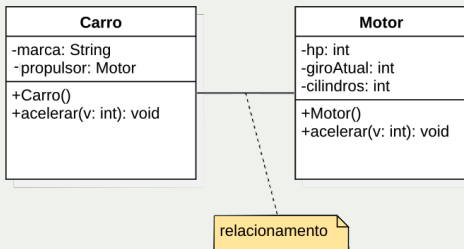


Associação entre classes

Associação entre classes

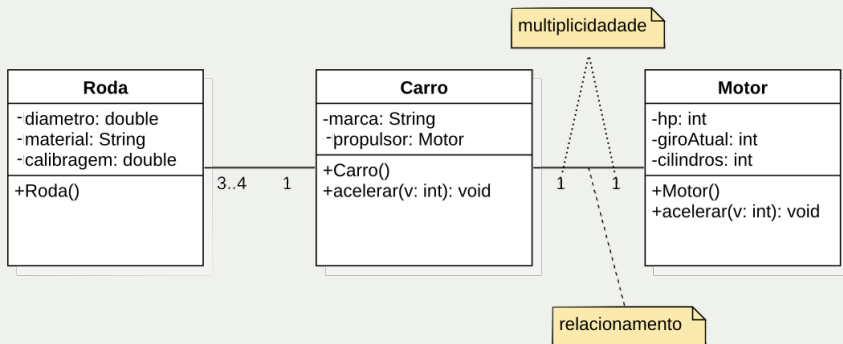
- Relacionamento entre classes que permite o compartilhamento de informação e colaboração para a execução de computação
- Descreve o vínculo entre objetos de uma ou mais classes
- A classe **Carro** possui um relacionamento com a classe **Motor**, pois um objeto da classe Carro contém 1 objeto da classe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



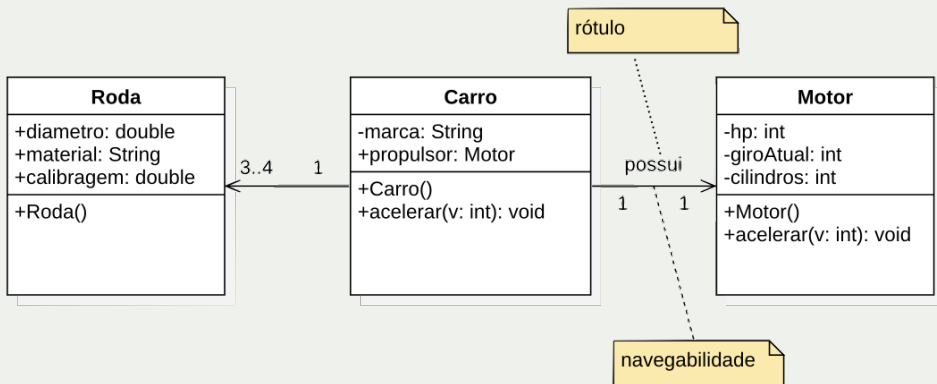
Associação bidirecional

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



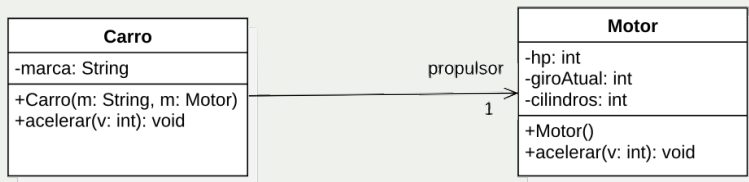
Associação unidirecional

- 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

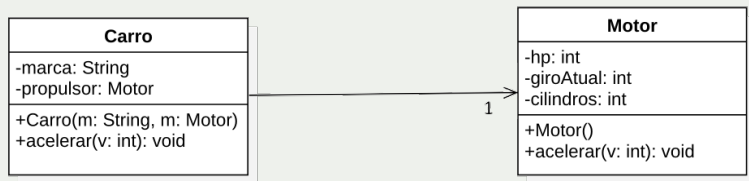


Associação: notação textual vs linha de associação

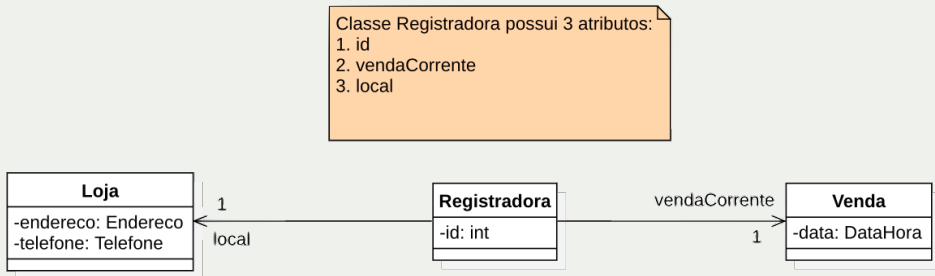
- Uso da notação de associação para indicar que Carro possui uma referência para uma instância de Motor



- Notação textual de atributo



Associação: notação textual vs linha de associação



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



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

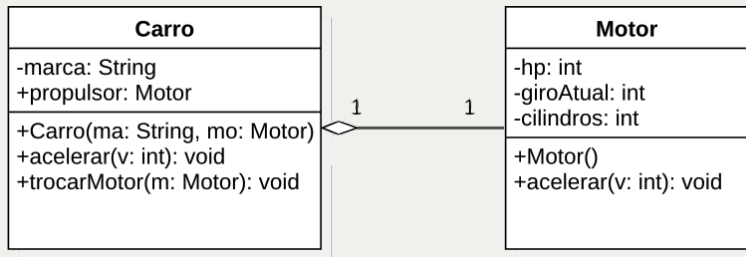
Agregação pode ser substituída por uma associação simples

Um dos criadores da UML até sugere nunca usar Agregação, se preocupe mais em representar a Composição



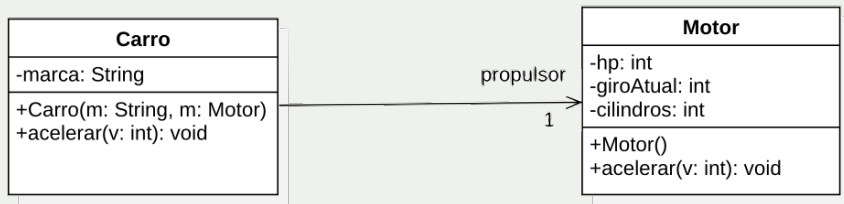
Associação do tipo Agregação

- 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



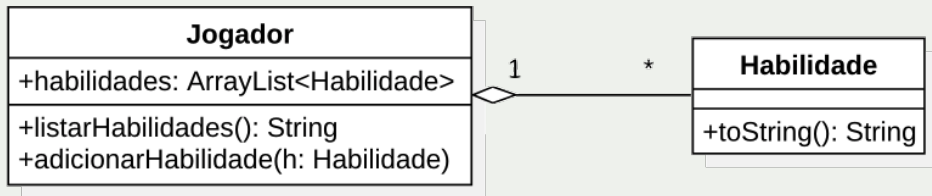
Associação do tipo Agregação

```
10 public class Carro{
11     private String marca;
12     private Motor propulsor;
13
14     public Carro(String m, Motor mo){
15         this.marca = m;
16         this.propulsor = mo;
17     }
18
19     public void acelerar(int valor){
20         this.propulsor.acelerar(valor);
21     }
22
23     public void trocarMotor(Motor mo){
24         this.propulsor = mo;
25     }
26 }
```



Associação do tipo Agregação

- Serve para **indicar a obrigatoriedade de uma complementação das informações de um objeto-todo por seus objetos-parte**, quando esse for consultado



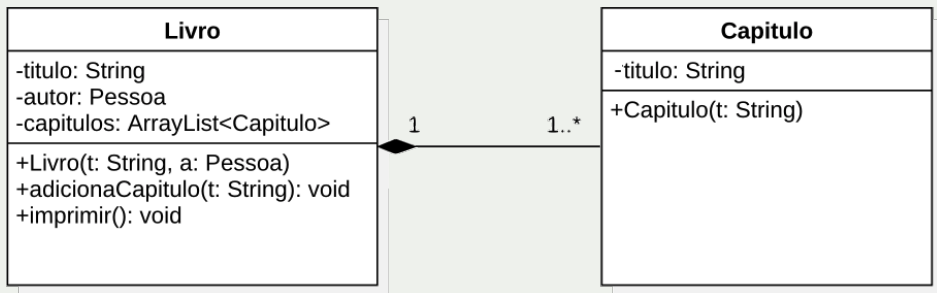
```
27 public String listarHabilidades(){
28     StringBuilder sb = new StringBuilder();
29     for(Habilidade item : habilidades){
30         sb.append(item);
31     }
32     return sb.toString();
33 }
```



- 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 destruímos um **livro**, não faria mais sentido os capítulos existirem



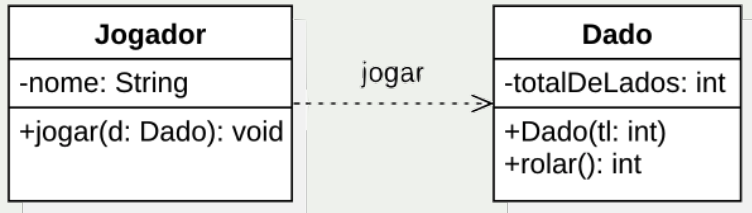
Associação do tipo Composição

```
34 public class Livro{
35     private String titulo;
36     private ArrayList<Capitulo> capitulos;
37     private Pessoa autor;
38
39     public Livro(String t, Pessoa a){
40         this.titulo = t;
41         this.capitulos = new ArrayList<>();
42         this.autor = a;
43     }
44
45     public void adicionaCapitulo(String titulo){
46         this.capitulos.add(new Capitulo(titulo));
47     }
48 }
```



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




```
49 public void jogar(Dado d){  
50     int res = d.rolar();  
51     ...  
52 }
```



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

- Capítulos 16, seções: 16.1 – 16.9, 16.10, 16.13

 GILLEANES T.A. GUEDES
UML2 – UMA ABORDAGEM PRÁTICA
2a. Edição - Editora Novatec, 2011

- Capítulos 4, seções: 4.1–4.2.2, 4.2.4, 4.2.5, 4.2.8



Exemplo

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: +55048998761234 → +55 (48) 9 9876-1234



Exercício

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

- Para a modelagem siga o princípio da **divisão de responsabilidades** (*Separation of Concerns* – SoC) e responsabilidade única (*Single Responsibility Principle* – SRP¹)
- **Cada classe deve ser responsável por uma pequena parte** da funcionalidade de um software
- A **responsabilidade** deve **estar completamente encapsulada** dentro da classe

¹<https://blog.cleancoder.com/uncle-bob/2014/05/08/SingleResponsibilityPrinciple>



Quais classes precisarão ser modeladas? Siga o SoC!



Quais classes precisarão ser modeladas? Siga o SoC!

- Principal e Pessoa?



Quais classes precisarão ser modeladas? Siga o SoC!

- 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? Siga o SoC!

- 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? Siga o SoC!

■ 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? Siga o SoC!

■ 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? Siga o SoC!

■ 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?

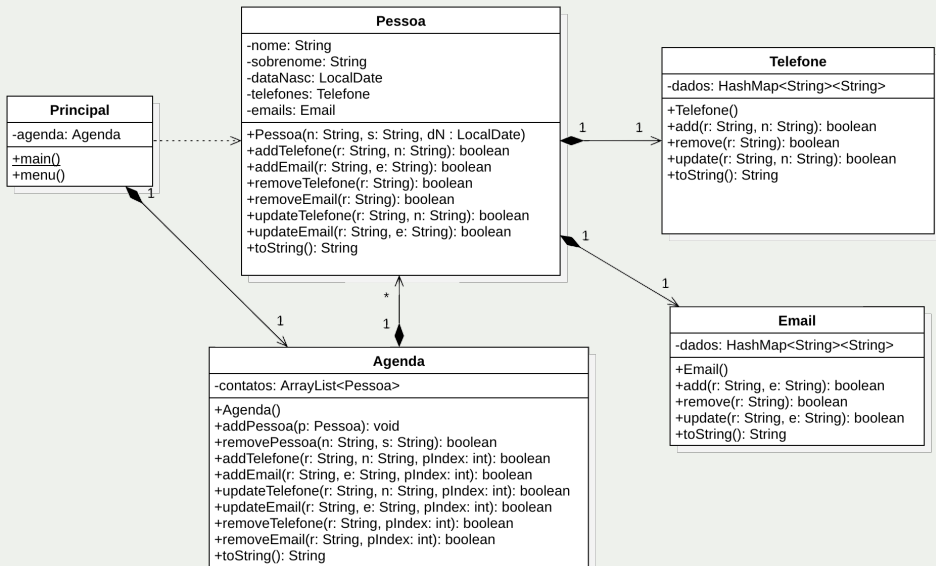


Quais classes precisarão ser modeladas? Siga o SoC!

- 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 ⇐



Sistema para gestão de Agenda telefônica



Material de apoio

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

■ `java.time.LocalDate`

```
53 import java.time.Month;
54 //.....
55 // Dia de hoje
56 LocalDate hoje = LocalDate.now();
57 // Criando objeto com uma data específica
58 LocalDate dataEspecificas = LocalDate.of(2020, Month.JANUARY, 10);
```

■ `java.time.LocalDateTime`

```
59 import java.time.format.DateTimeFormatter;
60 //.....
61 String dhF;
62
63 LocalDateTime dataHora = LocalDateTime.of(2019, 04, 22, 13, 30, 00);
64 dhF = dataHora.format(DateTimeFormatter.ofPattern("yyyyMMdd'T'HHmmss"));
65 System.out.printf(dhF); // Será impresso: 20190422T133000
66
67 // para obter a data e hora no momento que essa linha foi executada
68 LocalDateTime agora = LocalDateTime.now();
```



Material de apoio: Expressão regular

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



```
69 String emailER =  
70     "^[\\w-\\+]+(\\. [\\w]+)*@[\\w-]+(\\. [\\w]+)*(\\. [a-z]{2,})$";  
71  
72 String email = "meu.email@dominio.com.br";  
73 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>



```
76 ArrayList<Pessoa> lista = new ArrayList<>();
77 Pessoa a = new Pessoa();
78 Pessoa b = new Pessoa();
79
80 lista.add(a);
81 lista.add(b);
82
83 for(Pessoa p : lista){ // percorrendo um ArrayList
84     System.out.println(p.toString());
85 }
86
87 int pessoaIndex = lista.indexOf(b); // deverá retornar 1
88
89 // Não teria como buscar o índice para esse caso
90 // pois não existe qualquer referência para o objeto Pessoa
91 // criado dentro da chamada do método add
92 lista.add(new Pessoa());
```



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

 GILLEANES T.A. GUEDES
UML2 – UMA ABORDAGEM PRÁTICA
2a. Edição - Editora Novatec, 2011

