PROGRAMAÇÃO ORIENTADA A OBJETOS

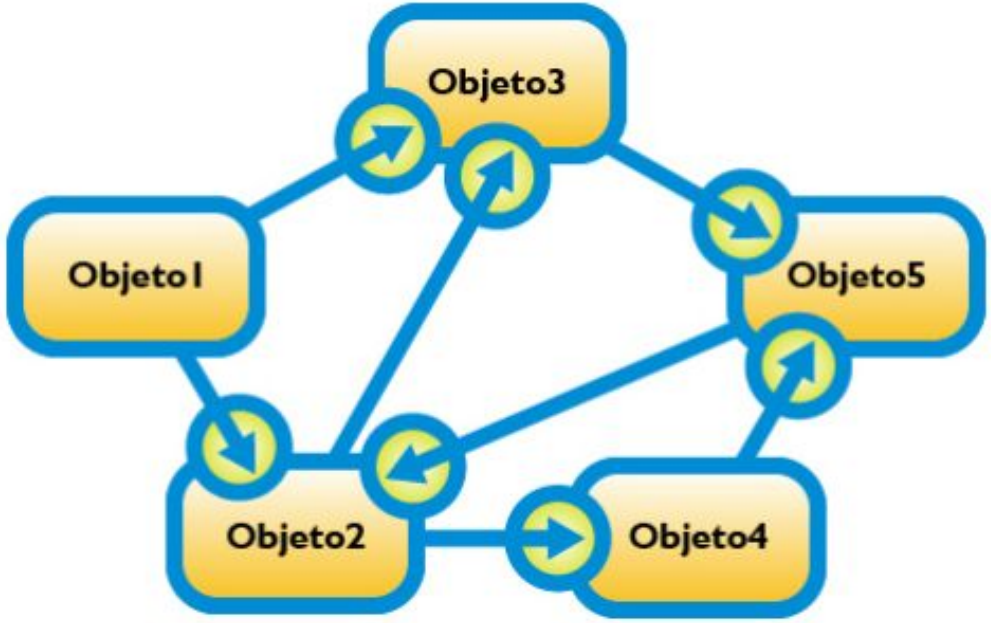
PROF. JOSENALDE OLIVEIRA

josenalde.oliveira@ufrn.br
https://github.com/josenalde/poo

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - UFRN

INTRODUÇÃO AO RELACIONAMENTO ENTRE CLASSES (E OBJETOS) - ASSOCIAÇÕES

- Objetos se relacionam pela troca de **mensagens** entre si, sejam entre objetos da mesma classe ou de classes diferentes



Fonte: https://materialpublic.imd.ufrn.br/curso/disciplina/2/8

INTRODUÇÃO AO RELACIONAMENTO ENTRE CLASSES (E OBJETOS) - ASSOCIAÇÕES

- Quais classes **cooperam** para o entendimento do problema?
- Exemplo: concessionária de carros
  - Carro (modelo, cor, placa); Pessoa (cpf, nome, telefone, endereco)

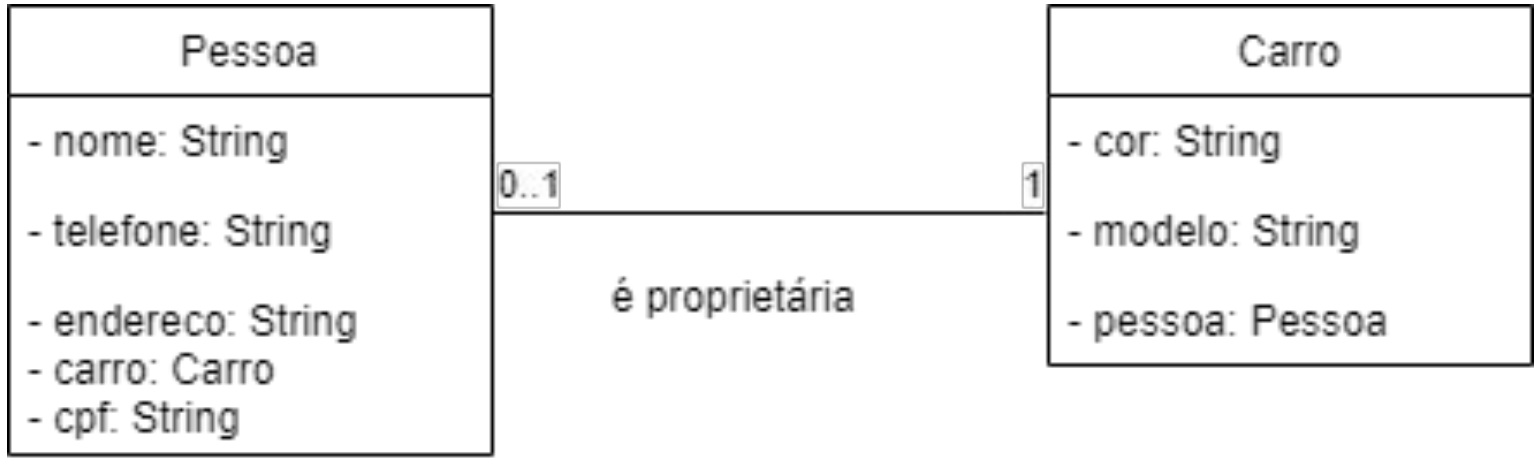


Figura 1: Associação simples com implementação

INTRODUÇÃO AO RELACIONAMENTO ENTRE CLASSES (E OBJETOS) - ASSOCIAÇÕES

- Multiplicidade máxima = 1: único atributo na classe
- Multiplicidade máxima > 1: lista de objetos
- Cada Pessoa tem ao menos 1 Carro e no máximo 1 Carro: **exata**
- Um Carro não necessariamente possui um dono e, se tiver, é exatamente 1 dono

ASSOCIAÇÕES

```
public class Carro {
    private String modelo;
    private String cor;
    private String placa;
    private Pessoa pessoa;

    public Pessoa getPessoa() {
        return pessoa;
    }
    public void setPessoa(Pessoa pessoa) {
        this.pessoa = pessoa;
    }
    // getters & setters...
}
```

```
public class Pessoa {
    private String nome;
    private String telefone;
    private String endereco;
    private String cpf;
    private Carro carro;

    public Carro getCarro() {
        return carro;
    }
    public void setCarro(Carro carro) {
        this.carro = carro;
    }
    // getters & setters...
}
```

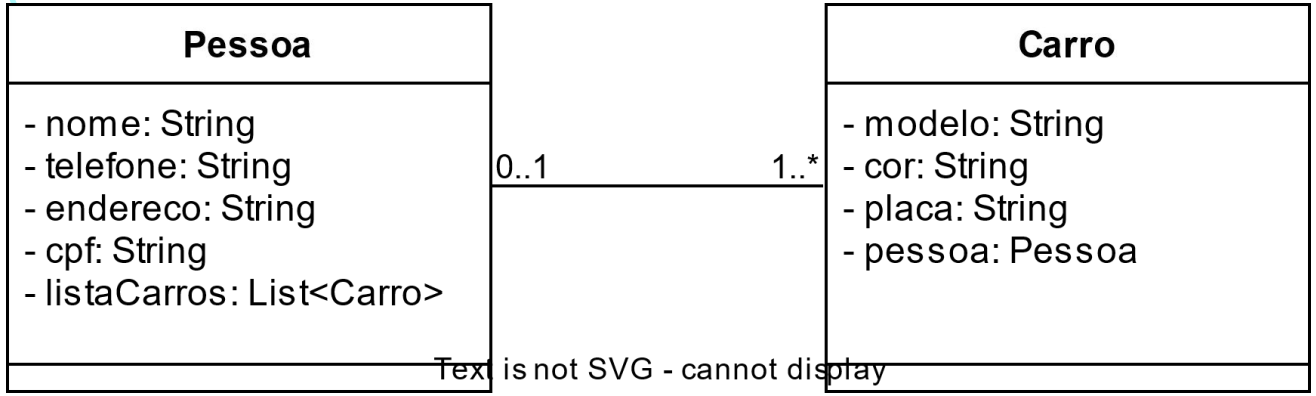
ASSOCIAÇÕES

```
public class Principal {
    //main()
    Pessoa p1 = new Pessoa();
    p1.setNome(“Agnes”);
    p1.setTelefone(“21 99999-9999”);
    p1.setEndereco(“Rua da Silva, 2”);
    Carro c1 = new Carro();
    c1.setModelo(“HB20s”);
    c1.setPlaca(“XYZ 1234”);
    c1.setCor(“Prata”);
    c1.setPessoa(p1);
    p1.setCarro(c1);

}
```

Estas instruções definem a associação!

ASSOCIAÇÕES

- Ao definir um dos lados como MUITOS, deve ser implementado como uma lista (dinâmica)



Para ir além e saber mais sobre mapeamento de relacionamentos no Jakarta Persistence API: https://jakarta.ee/specifications/persistence/2.2/apidocs/javax/persistence/onetomany

- Navegabilidade padrão bidirecional ⟷
  - Desejo saber quais carros determinado proprietário possui
  - Qual a pessoa, a partir de um objeto Carro, é proprietária
  - Se houver restrição de lista de carros a partir de uma Pessoa, não precisa do atributo listaCarros: List<Carro> em Pessoa. 0..1 ← X → 1..*

ASSOCIAÇÕES

```
public class Pessoa {

    private String nome;

    private String telefone;

    private String endereco;

    private List<Carro> listaCarros;

    public Pessoa() {

        listaCarros = new ArrayList();

    }

    public List<Carro> getCarro() {

        return listaCarros;

    }
```

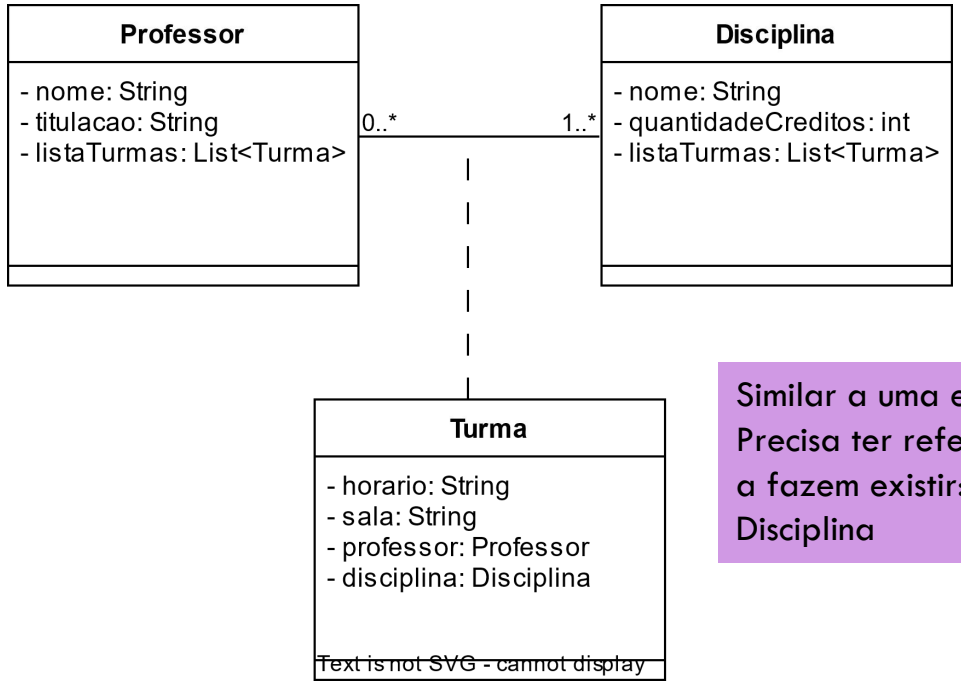
```
public void setCarro(List<Carro> listaCarros) {

    this.listaCarros = listaCarros;

}

public void adicionarCarro(Carro carro) {

        listaCarros.add(carro);

  }

public void removerCarro(Carro carro) {

        listaCarros.remove(carro);

}

//getters setters

}
```

ASSOCIAÇÕES

```
public class Principal {

    public static void main(String[] args) {

        Pessoa p1 = new Pessoa();

        p1.setNome("Agnes");

        p1.setTelefone("21 99999-9999");

        p1.setEndereco("Rua dos capitães, 2");

        Carro c1 = new Carro();

        c1.setModelo("Clio");

        c1.setPlaca("ESB 1234");

        c1.setCor("Prata");

        c1.setPessoa(p1);

        Carro c2 = new Carro();

        c2.setModelo("Pálio");

        c2.setPlaca("TRS 4321");

        c2.setCor("Branco");

        c2.setPessoa(p1);

        p1.adicionarCarro(c1);

        p1.adicionarCarro(c2);

    }

}
```

ASSOCIAÇÕES

- Classes que só fazem sentido na ocorrência de uma certa associação
- Exemplo: uma Turma corresponde a uma Disciplina e um Professor alocado

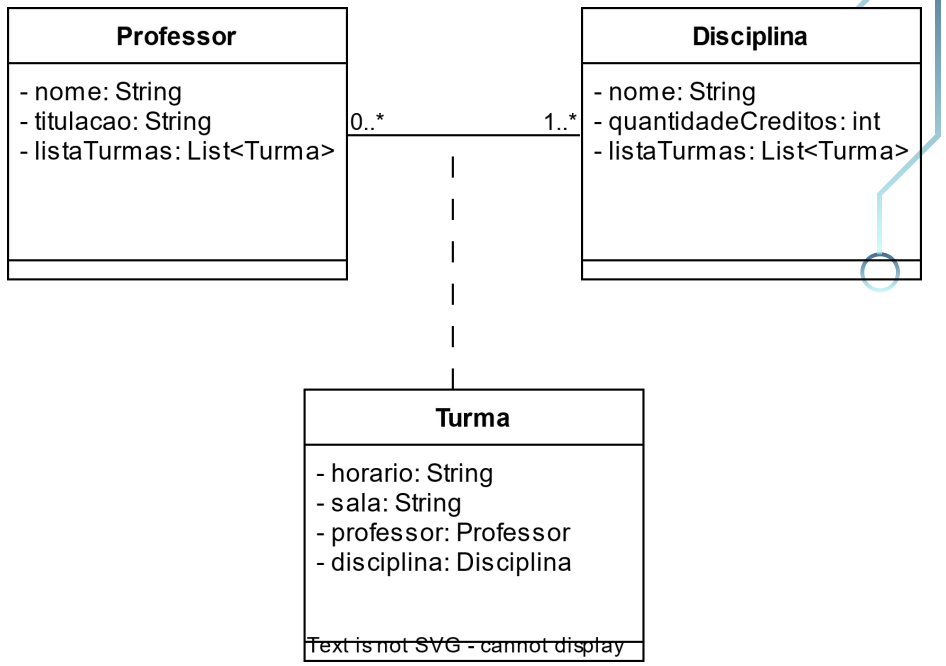


Similar a uma entidade fraca. Precisa ter referências às classes que a fazem existir: Professor e Disciplina

- A classe Turma é associativa, conceitualmente ligada a existência de objetos da classe Professor e Disciplina
- Um professor leciona uma ou mais disciplinas e uma disciplina é lecionada por um ou mais professores (pode não ter ainda professor alocado). Uma Turma teria atributos como horário e sala

ASSOCIAÇÕES

- Suponha dois professores p1 (Maria), p2 (Claudio)
- Suponha duas disciplinas d1 (Programação em Java) e d2 (Programação em C++). Tanto p1 quanto p2 lecionam d1 e d2, em dois horários distintos



Turma	Professor	Disciplina
t1	p1	d1
t2	p1	d2
t3	p2	d1
t4	p2	d2

O que diferencia é o horário e a sala

Turma	Professor	Disciplina	Horário	Sala
t1	p1	d1	19:00	10A
t2	p1	d2	15:00	11A
t3	p2	d1	21:00	20B
t4	p2	d2	17:00	21B
t5	p2	d2	08:00	22B

https://github.com/josenalde/apds/tree/main/scripts/capitulo5/exemplos/exemplo4

AGREGAÇÃO E COMPOSIÇÃO

- Usar outras classes para COMPOR uma classe, reutilizar classes
- Vamos imaginar uma classe “componente/parte” Motor e uma classe “composta” Automóvel.

Métodos construtores

```
class Motor {
    private int potencia;

    public Motor(){
        potencia = 1000;
    }

    public Motor(int potencia){
        this.potencia = potencia;
    }

    public int getPotencia(){
        return this.potencia;
    }

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

}
```

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);
    }
}
```

Fonte: https://materialpublic.imd.ufrn.br/curso/disciplina/2/8/6/2

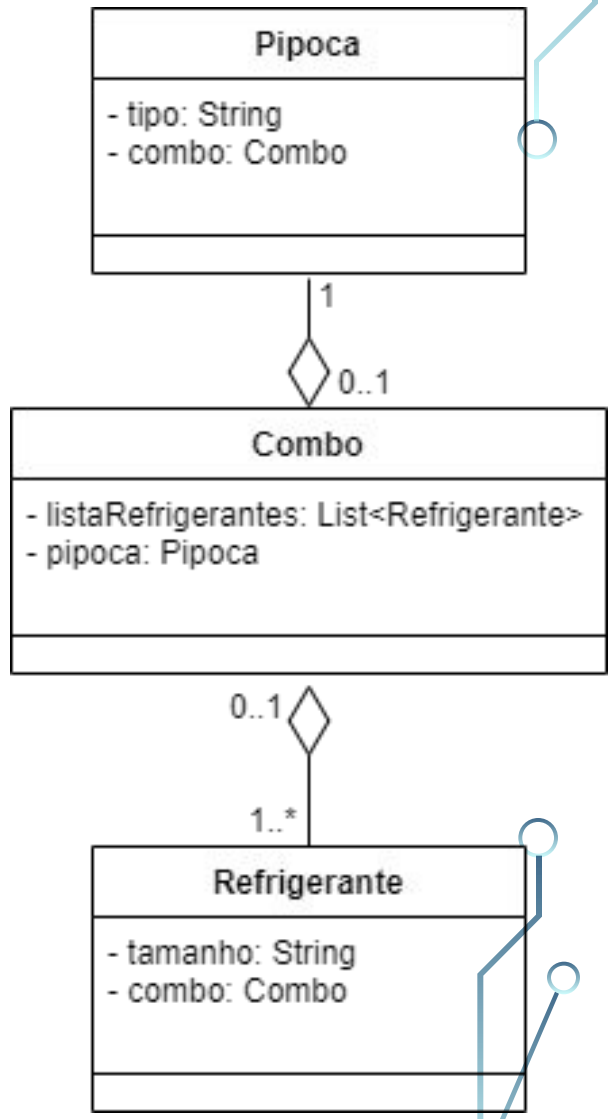
AGREGAÇÃO E COMPOSIÇÃO

- Usar outras classes para COMPOR uma classe, reutilizar classes
- Vamos imaginar uma classe “componente/parte” Motor e uma classe “composta” Automóvel.

- Estes construtores podem ser chamados na classe composta por qualquer outro método ou mesmo fora
  - Automovel automovel = new Automovel(); // neste caso a classe Automovel só tem o get e set para o motor, sem construtores
  - Motor motor = new Motor(); automovel.setMotor(motor);

AGREGAÇÃO E COMPOSIÇÃO

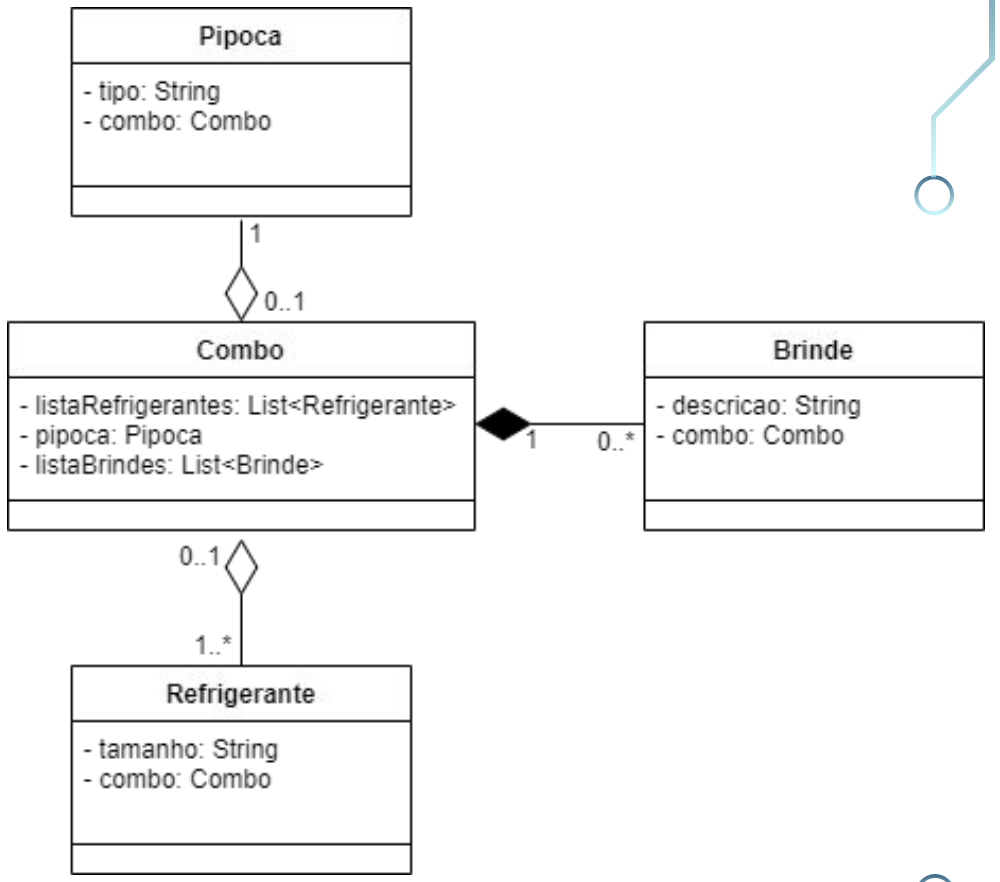
- Relacionamentos do tipo “faz parte de” ou “tem um”, a ideia de todo-parte. A representação é feita com um losango na extremidade do todo (classe composta)
- As associações todo-parte cuja parte é independente do todo são chamadas de agregação. A existência das partes não está condicionada à existência do objeto todo. No caso o losango é aberto.
- Exemplo: uma bombonière de um cinema. Um combo é composto de ao menos um refrigerante e exatamente uma pipoca. Uma pipoca ou refrigerante podem ser comercializados em no máximo um combo.



AGREGAÇÃO E COMPOSIÇÃO

- Composição, mais forte que agregação. A existência do objeto parte está condicionada à existência do objeto todo. Neste caso, o losango é fechado.

- Vamos acrescentar uma classe Brinde, para quem comprar o Combo. O Brinde não pode ser vendido separadamente e está condicionado à existência do Combo. Já a Pipoca e o Refrigerante podem ser vendidos separadamente. Repare as regras de negócio mapeadas nas multiplicidades



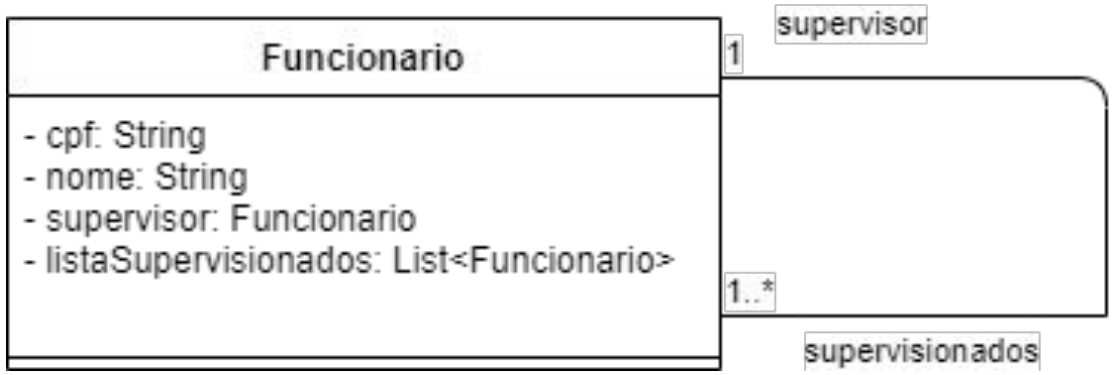
Teste: suponha um combo dito premium composto por 02 refrigerantes de tamanho médio (r1 e r2), uma pipoca salgada (p1) e dois brindes (b1 e b2).

https://github.com/josenalde/apds/tree/main/scripts/capitulo5/exemplos/exemplo5

AUTOASSOCIAÇÃO (REFLEXIVA, UNÁRIA)

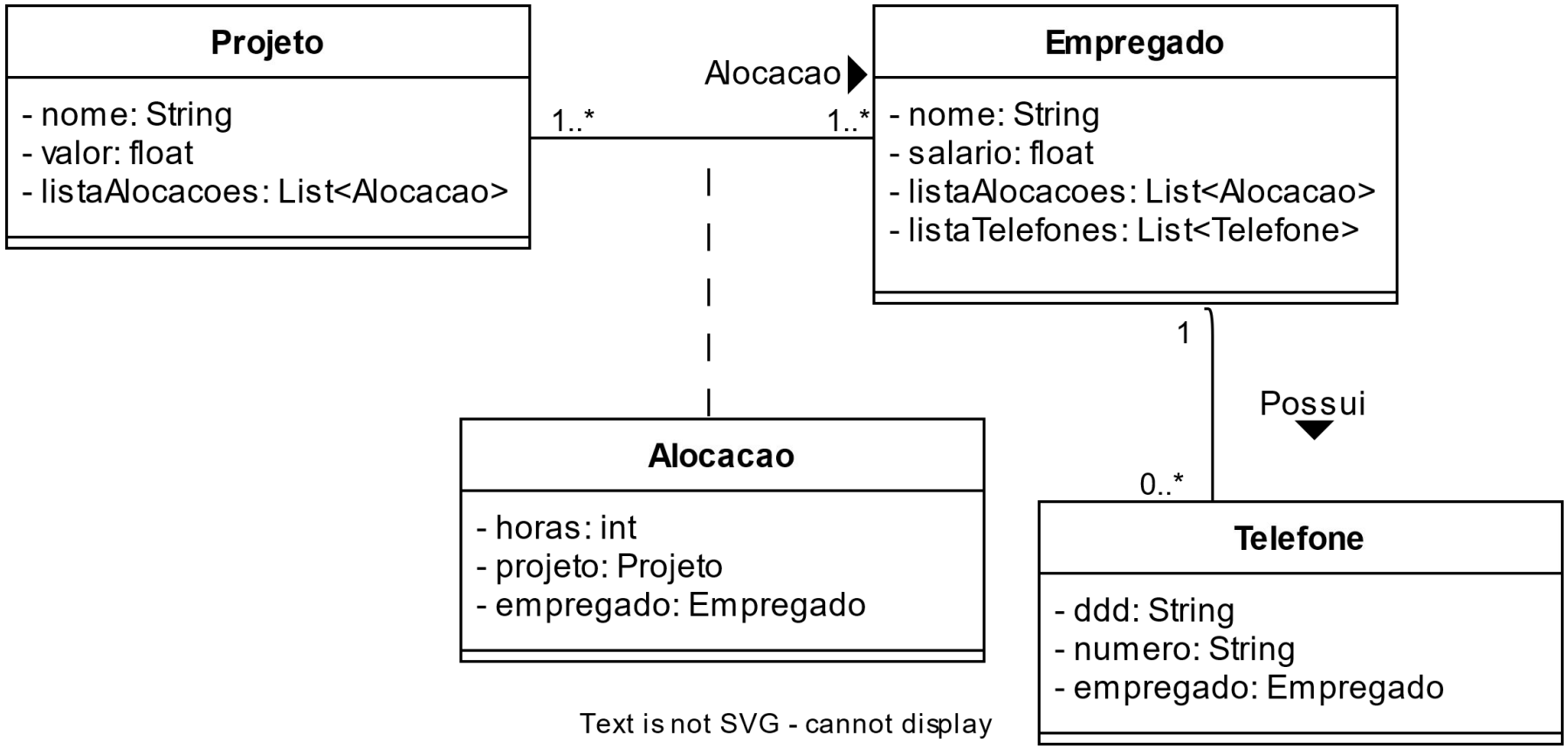
- Objetos de uma mesma classe estão relacionados entre si e precisam comunicar-se uns com os outros – se diferencia pelo papel que exercem
- Exemplo: um funcionário pode ser supervisor e ser supervisionado por outro funcionário
  - Uma empresa com 4 funcionários: Paulo (f1), Marcelo (f2), André (f3), Gabriel (f4). Os funcionários Marcelo e André são subordinados ao supervisor Paulo. André, por outro lado, supervisiona Gabriel.

Funcionário	Supervisionados	Supervisor
f1	f2, f3	-
f2	-	f1
f3	f4	f1
f4	-	f3



https://github.com/josenalde/apds/tree/main/scripts/capitulo5/exemplos/exemplo6

EXERCÍCIO DE FIXAÇÃO



EXERCÍCIO DE FIXAÇÃO

Com base no diagrama UML de classes da página anterior, implemente os itens abaixo:

a) Uma classe Projeto que possua os atributos nome (nome do projeto), valor (valor do orçamento) e listaAlocacoes (guarda a coleção de alocações de empregados no projeto). Faça um construtor que instancie a coleção com um ArrayList

b) Um método para adicionar e um para remover uma alocação na classe Projeto

c) Uma classe Empregado que possua os atributos nome, salário, listaAlocacoes (coleção de alocações de empregados

no projeto) e listaTelefones. Faça um construtor que instancie as coleções com ArrayList

d) Um método para adicionar e um para remover uma alocação na classe Empregado

e) Um método para adicionar e um para remover um telefone na classe Empregado

f) Uma classe Alocacao que possua os atributos horas (quantidade de horas alocadas para o empregado), o atributo

projeto (o projeto em que o empregado está alocado) e empregado (que guarda o empregado alocado).

g) Uma classe Telefone que possua os atributos ddd, numero e empregado (o dono do telefone)*

h) Uma classe Principal contendo o método main() que instancie quatro objetos, um de cada classe, com valores a sua escolha. A seguir, associe os objetos conforme a implementação das multiplicidades.

* Para garantir a primeira forma normal (1FN): https://materialpublic.imd.ufrn.br/curso/disciplina/3/73/6/4