APRESENTAÇÃO
DE APOIO

PROGRAMAÇÃO ORIENTADA A OBJETOS

Alessandro Valério – Aula 02

PUCRS online uol edtech

Professores

ALESSANDRO VALÉRIO DIAS
Professor Convidado

Mestre em Administração e Negócios pela PUCRS, possui também os títulos de especialista em duas áreas: Gerenciamento de Projetos de Tecnologia da Informação e Informática na Educação. É bacharel em Ciência da Computação e Psicologia pela Pontifícia Universidade Católica do Rio Grande do Sul. Atualmente, é analista de sistemas da Pontifícia Universidade Católica do Rio Grande do Sul e professor dos cursos de Análise e Desenvolvimento de Sistemas e de Ciência da Computação do Centro Universitário UniRitter.

EDSON IFARRAGUIRRE MORENO
Professor PUCRS

Doutor em Ciência da Computação pela Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS). Atualmente, é professor adjunto pela mesma PUCRS, estando vinculado a Escola Politécnica, sendo responsável por disciplinas da área de hardware para os cursos de Ciência da Computação, e Engenharia da Computação. Adicionalmente, trabalha com a orientação de alunos no desenvolvimento de projetos do curso de Engenharia de Software. Desde 2016, coordena o laboratório iSeed Labs, uma parceria entre a academia e a iniciativa privada que tem por objetivo fomentar a inovação e o empreendedorismo. Seus principais temas de pesquisa incluem: sistemas multiprocessados em chip (Multiprocessor System on Chip, MPSoC), projeto em nível de sistema e redes em chip (Network on Chip, NoC).

Ementa da disciplina

Estudo sobre conceitos de Classes (atributos, métodos, propriedades, visibilidade, instancia ou classe). Estudo de conceitos de Herança, Polimorfismo, Interfaces, Genéricos e Arrow functions. Estudo sobre funções de filtragem, mapeamento e redução. Estudo sobre construtores de tipos.

LITERAIS

```
var nomeDoObjeto = {
  nomeMembro1:
valorMembro1,
  nomeMembro2:
valorMembro2,
  nomeMembro3: valorMembro3
};
```



```
var pessoa = {
  nome: [‘Fulano’, ‘de Tal’],
  anoDeNascimento: 1990,
  profissao: ‘Estudante‘,
  calculaIdade: function() {
    return new Date().getFullYear() -
this.anoDeNascimento;
  }
};
```

FÁBRICAS

Funções que criam e retornam objetos

```
function criarPessoa () {
  return {
        nome: [‘Fulano’, ‘de Tal’],
        anoDeNascimento: 1990,
        profissao: ‘Estudante‘,
        calculaIdade: function() {
              return new Date().getFullYear() -
        this.anoDeNascimento;
        }
  };
};

const pessoa = criarPessoa();
console.log(pessoa);
```

FÁBRICAS

```
function criarPessoa (nome, anoDeNascimento, profissao) {
  return {
        nome,
        anoDeNascimento,
        profissao,
        calculaIdade: function() {
              return new Date().getFullYear() -
        this.anoDeNascimento;
        }
  };
};

const pessoa = criarPessoa("Fulano", 1990, "Estudante");
console.log(pessoa);
```

CONSTRUTORES

Funções que criam objetos

```
function Pessoa () {
        this.nome = ['Fulano', 'de Tal'];
        this.anoDeNascimento = 1990;
        this.profissao = "Estudante";
    this.calculaIdade = function() {
        return new Date().getFullYear() - this.anoDeNascimento;
    }
};

const pessoa = new Pessoa();
console.log(pessoa);
```

CONSTRUTORES

Funções que criam objetos

```
function Pessoa (nome, anoDeNascimento, profissao) {
        this.nome = nome;
        this.anoDeNascimento = anoDeNascimento;
        this.profissao = profissao;
    this.calculaIdade = function() {
        return new Date().getFullYear() - this.anoDeNascimento;
    }
};

const pessoa = new Pessoa([‘Fulano’, ‘de Tal’], 1990,
‘Estudante’);
console.log(pessoa);
```

CONSTRUTORES

O que é o tal *this*?

This é uma referência ao próprio objeto, nesse caso, à *Pessoa* criada

Quando precisamos acessar um membro dentro do próprio objeto, usamos o **this** para deixar claro que membro tentamos acessar

```
function Pessoa (nome, anoDeNascimento, profissao) {
        this.nome = nome;
        this.anoDeNascimento = anoDeNascimento;
        this.profissao = profissao;
    this.calculaIdade = function() {
        return new Date().getFullYear() - this.anoDeNascimento;
    }
};

const pessoa = new Pessoa([‘Fulano’, ‘de Tal’], 1990,
‘Estudante’);
console.log(pessoa);
```

CONSTRUTORES

O construtor Object()

```
const pessoa = new Object();

console.log(pessoa);

pessoa.nome = [‘Fulano’, ‘de Tal’];
pessoa[“anoDeNascimento”] = 1990;
pessoa.profissao = ‘Estudante’;
pessoa.calculaIdade = function() {
        return new Date().getFullYear() - this.anoDeNascimento;
    };

console.log(pessoa);
```

CONSTRUTORES

Todo o objeto em Javascript possui o atributo *constructor*

```
const pessoa = new Object();

pessoa.nome = [‘Fulano’, ‘de Tal’];
pessoa[“anoDeNascimento”] = 1990;
pessoa.profissao = ‘Estudante’;
pessoa.calculaIdade = function() {
        return new Date().getFullYear() - this.anoDeNascimento;
    };

console.log(pessoa.constructor);
```

PROTÓTIPOS

Mecanismo de herança entre objetos

```
function Pessoa () {
        this.nome = ['Fulano', 'de Tal'];
        this.anoDeNascimento = 1990;
        this.profissao = "Estudante";
    this.calculaIdade = function() {
        return new Date().getFullYear() - this.anoDeNascimento;
    }
};

const pessoa = new Pessoa();
console.log(pessoa.valueOf());
```

PROTÓTIPOS

JavaScript é descrita como uma linguagem baseada em protótipos

Objetos podem ter um objeto de protótipo

Modelo que fornece atributos e métodos

PROTÓTIPOS

Um objeto de protótipo pode ter outro objeto
(e assim por diante)

É a chamada cadeia de protótipos

PROTÓTIPOS

Formas de acessar

Object.getPrototypeOf()

__proto__ (somente na console)

Em funções construtoras existe a propriedade prototype

PROTÓTIPOS

Todo objeto possui uma propriedade chamada ***prototype***

```
function Pessoa () {
        this.nome = ['Fulano', 'de Tal'];
        this.anoDeNascimento = 1990;
        this.profissao = "Estudante";
    this.calculaIdade = function() {
        return new Date().getFullYear() - this.anoDeNascimento;
    }
};

const pessoa = new Pessoa();
console.log(Object.getPrototypeOf(pessoa));
```

PROTÓTIPOS

O método Object.create() usa protótipos

```
const pessoa = new Object();

pessoa.nome = [‘Fulano’, ‘de Tal’];
pessoa[“anoDeNascimento”] = 1990;
pessoa.profissao = ‘Estudante’;
pessoa.calculaIdade = function() {
        return new Date().getFullYear() - this.anoDeNascimento;
    };

console.log(Object.getPrototypeOf(pessoa));

const pessoa2 = Object.create(pessoa);

console.log(Object.getPrototypeOf(pessoa2));
```

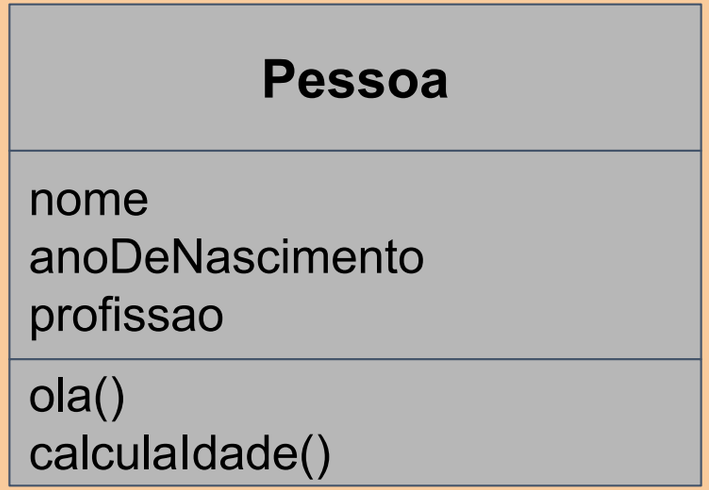
HERANÇA PROTOTIPADA

```
function Pessoa (nome, anoDeNascimento, profissao) {
        this.nome = nome;
        this.anoDeNascimento = anoDeNascimento;
        this.profissao = profissao;
    this.calculaIdade = function() {
        return new Date().getFullYear() - this.anoDeNascimento;
    }
};

const pessoa = new Pessoa([‘Fulano’, ‘de Tal’], 1990,
‘Estudante’);
console.log(pessoa);

Pessoa.prototype.saudar = function() {
        console.log(“Olá”);
};
console.log(pessoa);
```

CLASSES

Modelos para criar objetos

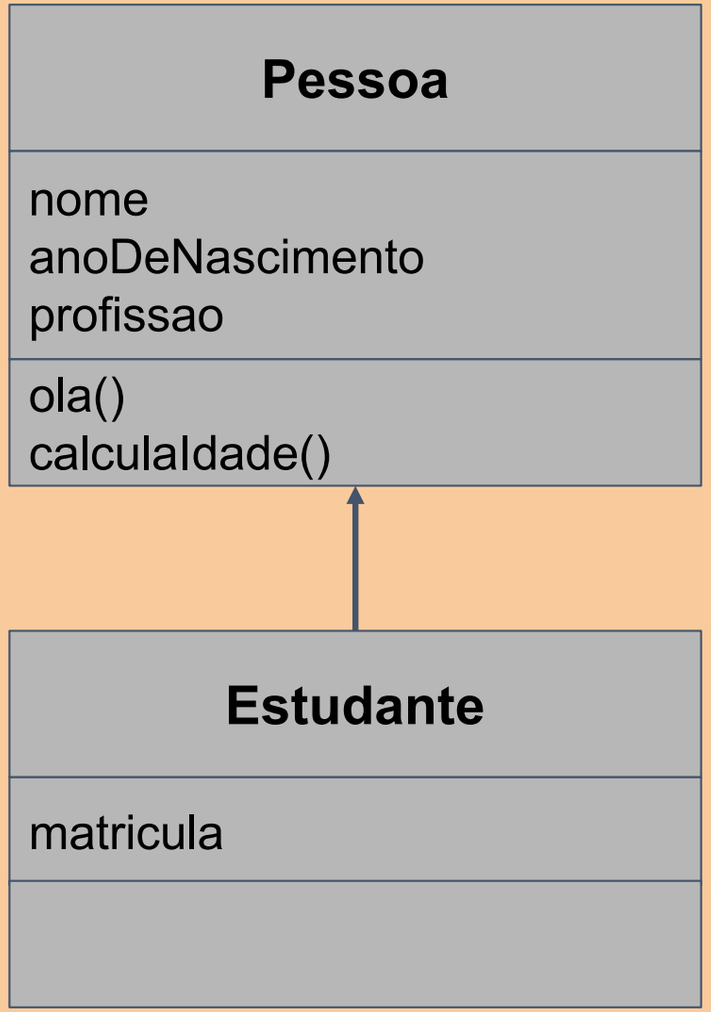
```
class Pessoa {
        constructor(nome, anoDeNascimento, profissao) {
                this.nome = nome;
        this.anoDeNascimento = anoDeNascimento;
        this.profissao = profissao;
    };
    ola() {
        console.log("Olá");
    };
    calculaIdade() {
        return new Date().getFullYear() - this.anoDeNascimento;
    };
};

const pessoa = new Pessoa(['Fulano', 'de Tal'], 1990,
'Estudante');
```



CLASSES

Herança

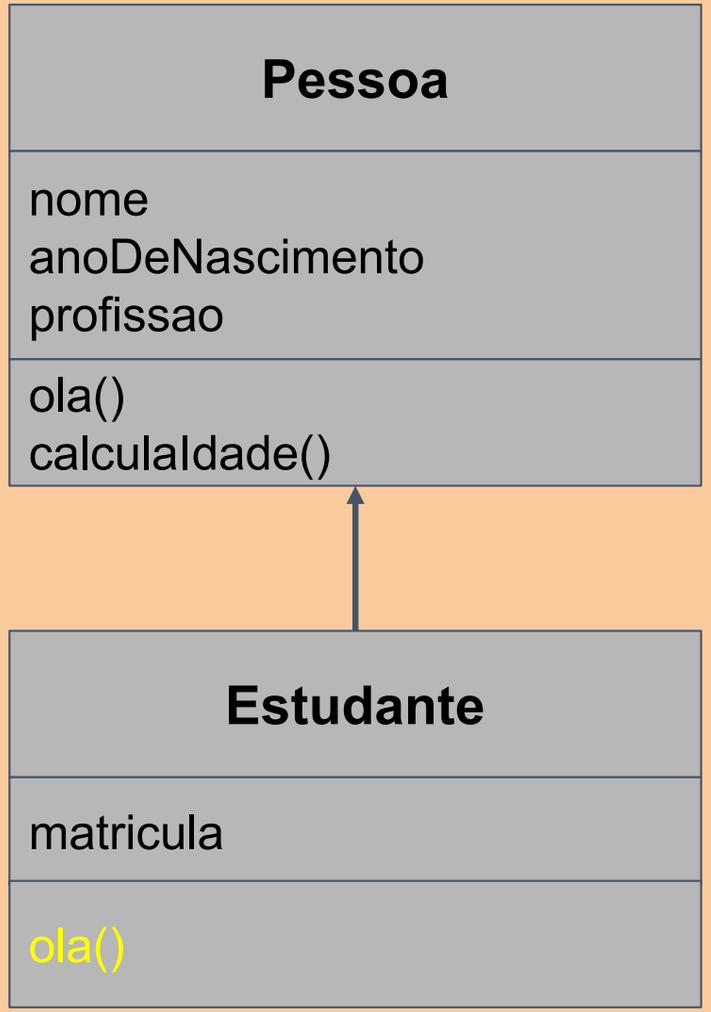
```
class Estudante extends Pessoa {
        constructor(nome, anoDeNascimento, profissao, matricula) {
                super(nome, anoDeNascimento, profissao);
                this.matricula = matricula;
    };
};

const aluno = new Estudante([‘Fulano’, ‘de Tal’], 1990, ‘Estudante’,
120901);
console.log(aluno);
```



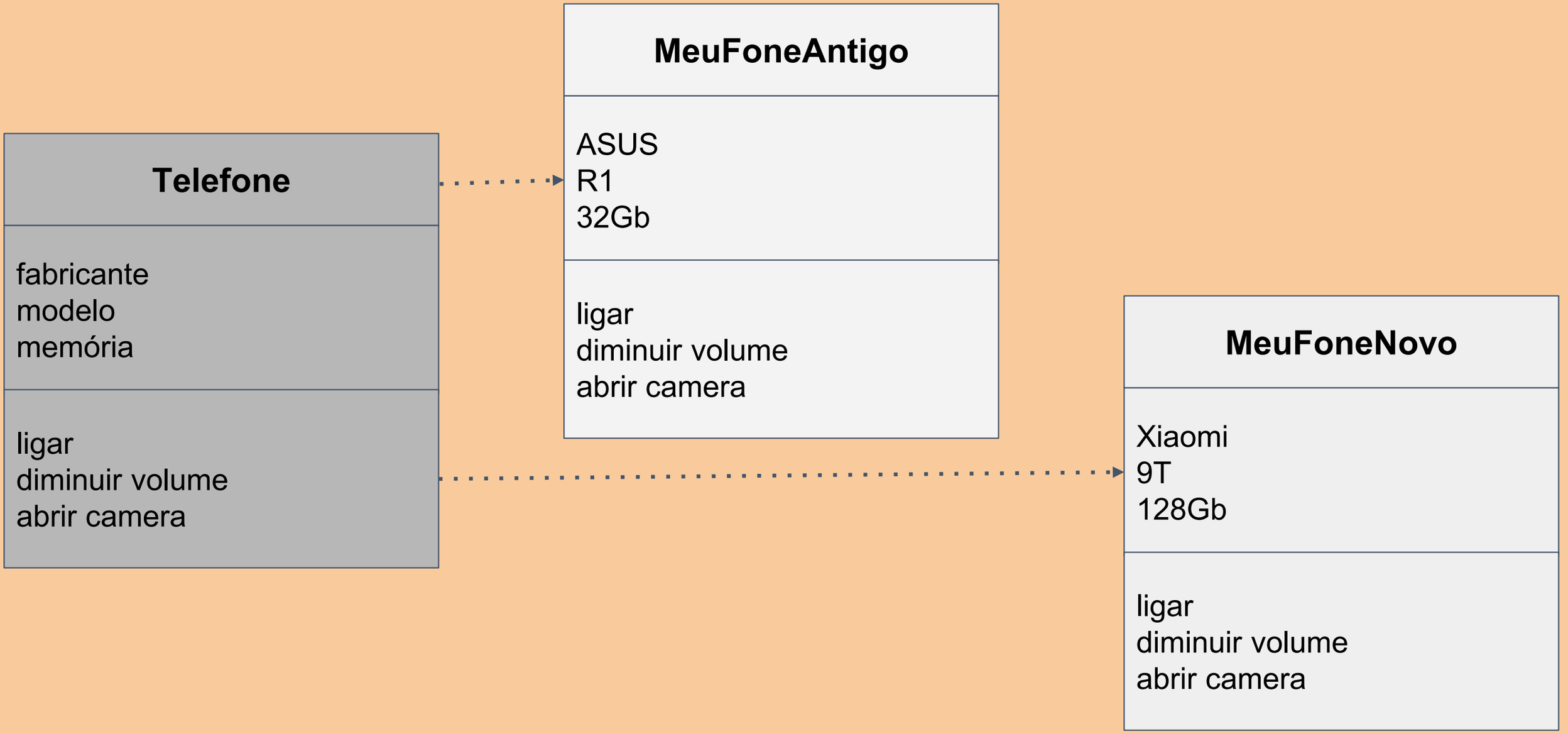
CLASSES

Polimorfismo

```
class Estudante extends Pessoa {
        constructor(nome, anoDeNascimento, profissao, matricula) {
                super(nome, anoDeNascimento, profissao);
                this.matricula = matricula;
    };
    ola() {
        super.ola();
        console.log(“ colega!”);
    };
};

const aluno = new Estudante([‘Fulano’, ‘de Tal’], 1990, ‘Estudante’,
120901);
console.log(aluno);
console.log(aluno.ola());
```



CRIANDO OBJETOS



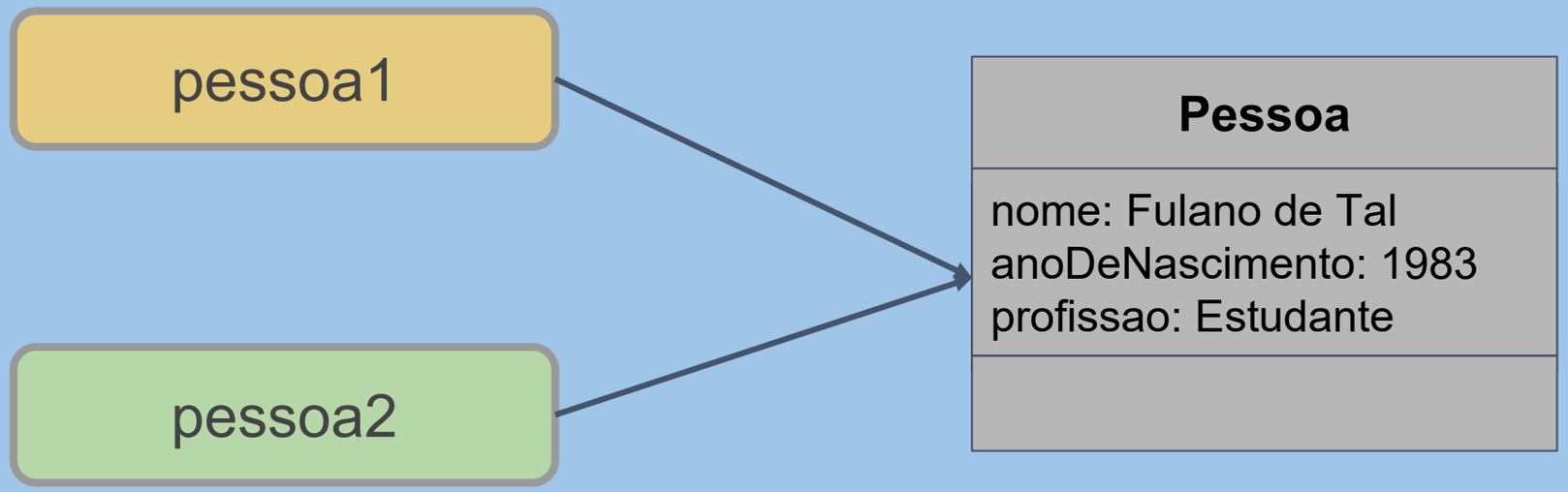
COPIANDO OU REFERENCIANDO OBJETOS?

REFERÊNCIA

```
function criarPessoa () {
  return {
        nome: ‘Fulano de Tal’,
        anoDeNascimento: 1990,
        profissao: ‘Estudante‘
  };
};

const pessoa1 = criarPessoa();
const pessoa2 = pessoa1;
pessoa2.anoDeNascimento = 1983;
console.log(pessoa1);
console.log(pessoa2);
```

REFERÊNCIA

```
function criarPessoa () {
  return {
        nome: 'Fulano de Tal',
        anoDeNascimento: 1990,
        profissao: 'Estudante'
  };
};

const pessoa1 = criarPessoa();
const pessoa2 = pessoa1;
pessoa2.anoDeNascimento = 1983;
console.log(pessoa);
console.log(pessoa2);
```



ALTERANDO OBJETOS

ADICIONAR ATRIBUTOS OU MÉTODOS

OBJETO.NOVO_MEMBRO = ALGO;

ALTERAR ATRIBUTOS OU MÉTODOS

OBJETO.MEMBRO = ALGO;

REMOVER ATRIBUTOS OU MÉTODOS

DELETE OBJETO.NOVO_MEMBRO;

ALTERAR ATRIBUTOS OU MÉTODOS DE UM PROTÓTIPO

HERANÇA PROTOTIPADA

Altera todos os objetos criados pelo protótipo

MEMBROS E SUAS PROPRIEDADES

ATRIBUTOS E MÉTODOS PODEM SER DE OBJETOS OU ESTÁTICOS

DE OBJETOS = CADA OBJETO POSSUI O SEU

ESTÁTICOS = SÃO COMPARTILHADOS

ATRIBUTOS ESTÁTICOS

DADOS QUE VALEM PARA QUALQUER OBJETO DE UM TIPO OU CLASSE

PARA USAR, NÃO PRECISA CRIAR UM OBJETO

ATRIBUTOS ESTÁTICOS

```
class Pessoa {
        static NOME_CLASSE = “Pessoa”;
        constructor(nome, anoDeNascimento, profissao) {
                this.nome = nome;
        this.anoDeNascimento = anoDeNascimento;
        this.profissao = profissao;
    };
    ola() {
        console.log(“Olá”);
    };
    calculaIdade() {
        return new Date().getFullYear() - this.anoDeNascimento;
    };
};

console.log(Pessoa.NOME_CLASSE);
```

MÉTODOS ESTÁTICOS

LÓGICA QUE É EXECUTADA DA MESMA MANEIRA EM QUALQUER OBJETO DE UM TIPO OU CLASSE

MÉTODOS UTILITÁRIOS, INDEPENDENTES DE OBJETOS

PARA USAR, NÃO PRECISA CRIAR UM OBJETO

MÉTODOS ESTÁTICOS

```
class Pessoa {
        constructor(nome, anoDeNascimento, profissao) {
                this.nome = nome;
        this.anoDeNascimento = anoDeNascimento;
        this.profissao = profissao;
    };
    static ola() {
        console.log("Olá");
    };
    calculaIdade() {
        return new Date().getFullYear() - this.anoDeNascimento;
    };
};

console.log(Pessoa.ola());
```

VISIBILIDADE

VISIBILIDADE

PENSAMOS EM ENCAPSULAMENTO

ATRIBUTOS E MÉTODOS PODEM SER

PÚBLICOS

PRIVADOS

ATRIBUTOS E MÉTODOS PÚBLICOS

SÃO O PADRÃO

NÃO É NECESSÁRIO FAZER NADA

ATRIBUTOS E MÉTODOS PRIVADOS

ALGUMAS FORMAS DE FAZER

VARIÁVEIS LOCAIS

IDENTIFICADORES PRÉ-FIXADOS

VARIÁVEIS LOCAIS

```
function Pessoa () {
        let nome = [‘Fulano’, ‘de Tal’];
        let anoDeNascimento = 1990;
        let profissao = “Estudante”;
    this.calculaIdade = function() {
        return new Date().getFullYear() - anoDeNascimento;
    }
};

const pessoa = new Pessoa();
console.log(pessoa);
```

IDENTIFICADORES PRÉ-FIXADOS

```
class Estudante extends Pessoa {
    #matricula;
        constructor(nome, anoDeNascimento, profissao, matricula) {
            super(nome, anoDeNascimento, profissao);
            this.#matricula = matricula;
    };
    ola() {
        super.ola();
        console.log(“ colega!”);
    };
};

const aluno = new Estudante([‘Fulano’, ‘de Tal’], 1990, ‘Estudante’,
120901);
console.log(aluno);
```

IDENTIFICADORES PRÉ-FIXADOS

```
class Estudante extends Pessoa {
        #matricula;
                constructor(nome, anoDeNascimento, profissao, matricula) {
                                super(nome, anoDeNascimento, profissao);
                                this.#matricula = matricula;
                this.#ola();
        };
        #ola() {
                super.ola();
                console.log(“ colega!”);
        };
};

const aluno = new Estudante([‘Fulano’, ‘de Tal’], 1990, ‘Estudante’,
120901);
console.log(aluno);
```

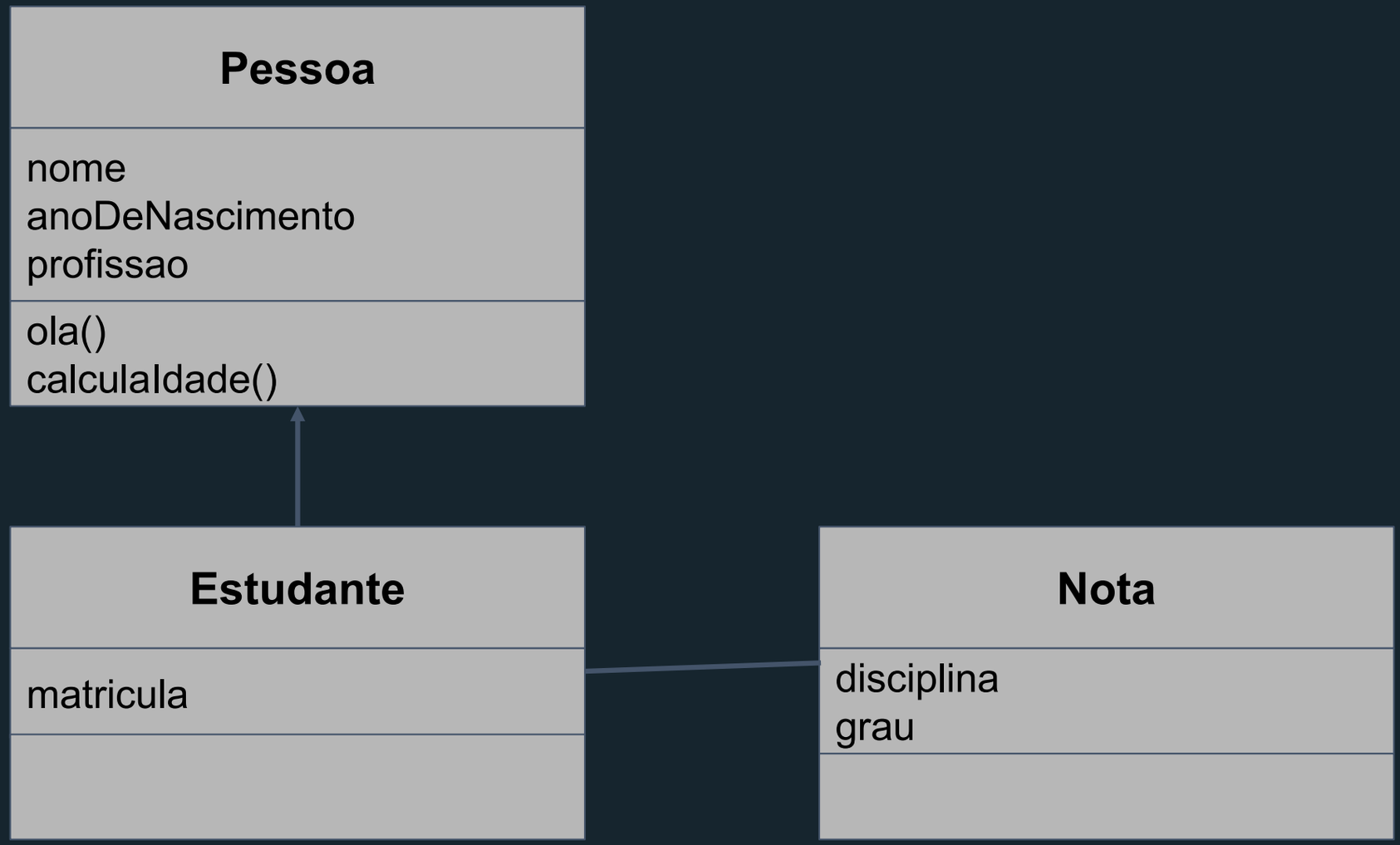
MÉTODOS DE ACESSO

```
class Estudante extends Pessoa {
        #matricula;
                constructor(nome, anoDeNascimento, profissao, matricula) {
                        super(nome, anoDeNascimento, profissao);
                        this.#matricula = matricula;
        };
        getMatricula() {
           return this.#matricula;
        };
};

const aluno = new Estudante([‘Fulano’, ‘de Tal’], 1990, ‘Estudante’,
120901);
console.log(aluno.getMatricula());
```

MÉTODOS DE ATRIBUIÇÃO

```
class Estudante extends Pessoa {
    #matricula;
        constructor(nome, anoDeNascimento, profissao, matricula) {
            super(nome, anoDeNascimento, profissao);
            this.#matricula = matricula;
    };
    getMatricula() {
       return this.#matricula;
    };
    setMatricula(valor) {
       this.#matricula = valor;
    };
};

const aluno = new Estudante([‘Fulano’, ‘de Tal’], 1990, ‘Estudante’,
120901);
console.log(aluno.getMatricula());
aluno.setMatricula(158590);
console.log(aluno.getMatricula());
```

ASSOCIAÇÃO ENTRE OBJETOS



ASSOCIAÇÃO

```
class Nota {
   constructor(disciplina, grau) {
      this.disciplina = disciplina;
      this.grau = grau;
   }
};
```

```
class Estudante extends Pessoa {
        #matricula;
    notas = [];
        constructor(nome, anoDeNascimento, profissao,
matricula) {
                super(nome, anoDeNascimento, profissao);
                this.#matricula = matricula;
    };
        getMatricula() {
        return this.#matricula;
        };
        setMatricula(valor){
        this.#matricula = valor;
    };
        addNota(nota){
        this.notas.push(nota);
        };
};
```

A SEGUIR

INTERFACES

GENÉRICOS

ARROW FUNCTIONS

FILTROS, MAPEAMENTOS E REDUÇÕES

PUCRS online uol edtech