

ORIENTAÇÃO A OBJETOS

Exercício

DESENVOLVA

Pensando na Faculdade Senac, desenvolva as seguintes classes:

Aluno

Atributos: nome, CPF, endereço e estado civil e uma turma (ex.: 2018/1).

Funcionário Administrativo

Atributos: nome, CPF, endereço, estado civil, salário e setor.

Funcionário Professor

Atributos: nome, CPF, endereço, estado civil, salário e titulação

Obs.: Por enquanto os métodos não precisam ser pensados/implementados, somente a arquitetura de classes.

ANÁLISE

```
1 export class Aluno {  
2   private _nome: string;  
3   private _cpf: string;  
4   private _endereco: string;  
5   private _estadoCivil: string;  
6   private _turma: string;  
7 }  
8
```

```
1 export class FuncionarioAdministrativo {  
2   private _nome: string;  
3   private _cpf: string;  
4   private _endereco: string;  
5   private _estadoCivil: string;  
6   private _salario: number;  
7   private _setor: string;  
8 }
```

```
1 export class FuncionarioAdministrativo {  
2   private _nome: string;  
3   private _cpf: string;  
4   private _endereco: string;  
5   private _estadoCivil: string;  
6   private _salario: number;  
7   private _titulacao: string;  
8 }
```

PROBLEMAS

Se quisermos adicionar uma informação básica para todos tipos de "pessoas", como um número de telefone, teremos que adicionar em todas as classes.

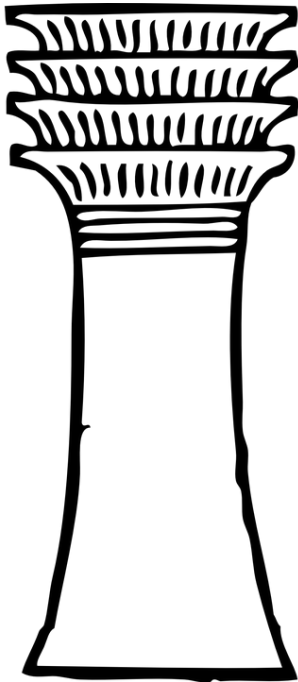
Como resolver?

ORIENTAÇÃO A OBJETOS

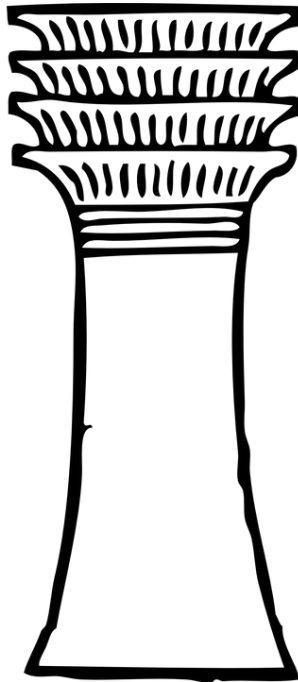
Herança

4 PILARES

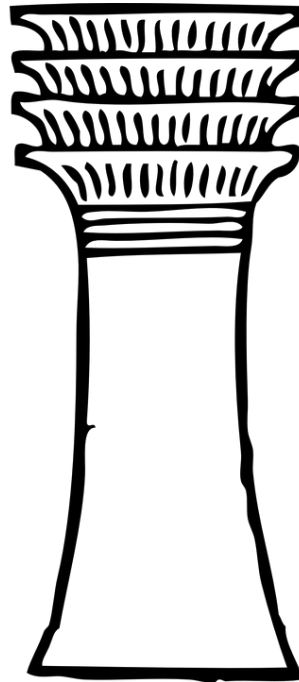
Abstração



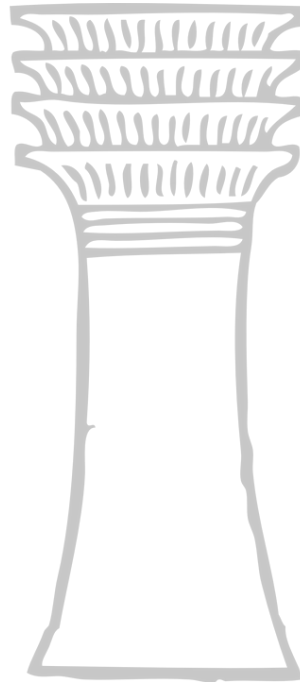
Encapsulamento



Herança



Polimorfismo



PROGRAMAÇÃO ORIENTADA A OBJETOS

SOLUÇÃO

Utilizando herança podemos ter uma classe, chamada **superclasse**, que irá centralizar (**generalizar**) todas as características em comum entre classes, e classes **especializadas**, que terão apenas as especificidades destas **subclasses**.

CODIFICAÇÃO

Para utilizar a herança, deve-se utilizar, após a declaração da classe, a palavra chave **extends**, seguida do nome da classe que será estendida.

```
export class Aluno extends Pessoa{
```

Declarando assim, estamos sinalizando que a classe Aluno herda todos os atributos e comportamentos da classe Pessoa que não são privados.

CODIFICAÇÃO

Para o nosso problema, poderíamos ter algo assim:

```
1 export class Pessoa {  
2   protected _nome: string;  
3   protected _cpf: string;  
4   protected _endereco: string;  
5   protected _estadoCivil: string;  
6 }
```

```
1 import { Pessoa } from "../Pessoa";  
2  
3 export class Aluno extends Pessoa{  
4  
5   private _turma: string;
```

Torna Aluno uma
subclasse de
Pessoa

Assim, declaramos tudo que existe em comum nas classes Aluno, FuncionarioAdministrativo e FuncionarioProfessor em uma superclasse chamada **Pessoa**, e as suas subclasses terão apenas os atributos e métodos que são pertencentes unicamente a elas.

CODIFICAÇÃO

Para o nosso problema, poderíamos ter algo assim:

```
1 export class Pessoa {  
2   protected _nome: string;  
3   protected _cpf: string;  
4   protected _endereco: string;  
5   protected _estadoCivil: string;  
6 }
```

```
1 import { Pessoa } from "./Pessoa";  
2  
3 export class Aluno extends Pessoa{  
4  
5   private _turma: string;
```

```
Principal.ts > 00 aluno  
2  
3 const aluno: Aluno = {  
4   const aluno: Aluno  
5 };  
6  
7  
'aluno' é declarado, mas seu valor nunca é lido. ts(6133)  
  
O tipo '{}' não tem as propriedades a seguir do tipo 'Aluno': _turma, _nome, _cpf, _endereco, _estadoCivil ts(2739)  
Exibir o Problema  Correção Rápida... (N.)
```

HERANÇA X ASSOCIAÇÃO

Para sabermos se devemos utilizar uma herança ou associação entre classes, basta fazer um teste simples, o teste do **é um** ou **tem um**.

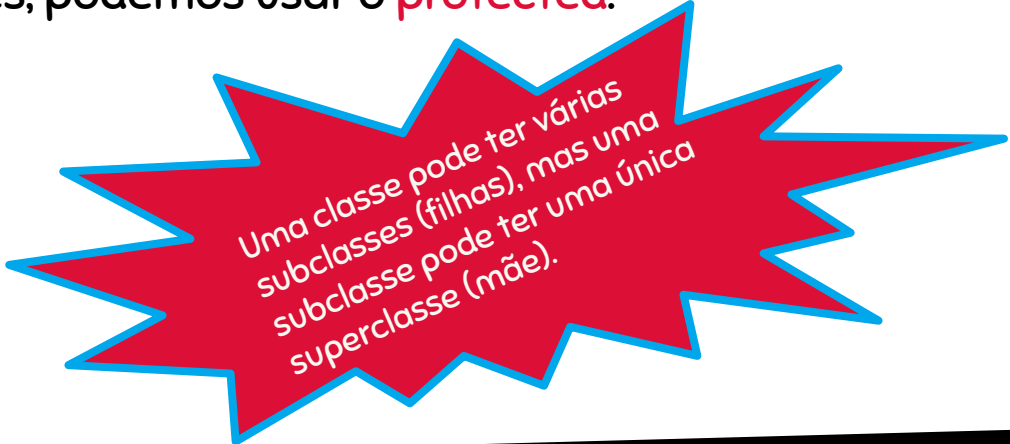
Toda vez que se fizer essa pergunta entre duas classes que se relacionam, quando a resposta for **é um**, significa que uma herança é apropriada, se for **tem um**, temos uma associação.

Aluno **é uma** Pessoa, então temos relação de herança.

Faculdade **tem** alunos, então aqui temos uma associação (atributo do tipo Aluno na classe Faculdade).

UTILIZAR PROTECTED OU PRIVATE?

Em termos de proteção dos campos, é interessante que sempre que possível os campos sejam **privados**, mas por facilidade em hierarquias de classes, podemos usar o **protected**.



Uma classe pode ter várias subclasses (filhas), mas uma subclasse pode ter uma única superclasse (mãe).

SOBRESCRITA DE MÉTODOS

Agora podemos sobrescrever nossos métodos, se tivermos um método na subclasse com **a mesma assinatura** que o método da superclasse, estaremos sobrescrevendo o da classe mãe.

SOBRESCRITA DE MÉTODOS

Se tivermos um simples método "falar" implementado na super e na subclasse, que retorna uma String que representa a fala de um objeto, podemos ver a sobrescrita acontecendo.

```
TS Pessoa.ts poo X
TS Pessoa.ts > Pessoa > _estadoCivil
1 export class Pessoa {
2   protected _nome: string = "";
3   protected _cpf: string = "";
4   protected _endereco: string = "";
5   protected _estadoCivil: string = "";
6
7   public falar(): string {
8     return "Sou uma pessoa";
9   }
10 }
11

TS Aluno.ts poo X
TS Aluno.ts > ...
1 import { Pessoa } from "../Pessoa";
2
3 export class Aluno extends Pessoa {
4   private _turma: string = "";
5
6   public falar(): string {
7     return "São muitas provas. Mimimi";
8   }
9 }
10

TS Principal.ts poo •
TS Principal.ts > ...
1 import { Aluno } from "../Aluno";
2
3 const aluno: Aluno = new Aluno();
4
5 console.log("aluno.falar :>> ",
6   aluno.falar());
7
```


SOBRESCRITA DE MÉTODOS

```
● → poo ts-node Principal.ts  
    aluno.falar :>> São muitas provas. Mimimi  
○ → poo []
```

```
TS Pessoa.ts poo X  
TS Pessoa.ts > Pessoa > _estadoCivil  
1 export class Pessoa {  
2   protected _nome: string = "";  
3   protected _cpf: string = "";  
4   protected _endereco: string = "";  
5   protected _estadoCivil: string = "";  
6  
7   public falar(): string {  
8     return "Sou uma pessoa";  
9   }  
10 }  
11
```

```
TS Aluno.ts poo X  
TS Aluno.ts > ...  
1 import { Pessoa } from "../Pessoa";  
2  
3 export class Aluno extends Pessoa {  
4   private _turma: string = "";  
5  
6   public falar(): string {  
7     return "São muitas provas. Mimimi";  
8   }  
9 }  
10
```

```
TS Principal.ts poo ●  
TS Principal.ts > ...  
1 import { Aluno } from "../Aluno";  
2  
3 const aluno: Aluno = new Aluno();  
4  
5 console.log("aluno.falar :>> ",  
6   aluno.falar());  
7
```

DESAFIO/CURIOSIDADE

Note que, como um Aluno é uma Pessoa, podemos instanciar uma Pessoa com um aluno (sabia?). Porém, ao fazer isso, o objeto **não "enxerga"** o atributo da subclasse.

```

Ts Pessoa.ts
3 protected _cpf: string = "";
4 protected _endereco: string = "";
5 protected _estadoCivil: string = "";
6
7 public falar(): string {
8     return "Sou uma pessoa";
9 }
10
11 public set nome(nome: string) {
12     this._nome = nome;
13 }
14 public set cpf(cpf: string) {
15     this._cpf = cpf;
16 }
17 public set endereco(endereco: string) {
18     this._endereco = endereco;
19 }
20 public set estadoCivil(estadoCivil: string) {
21     this._estadoCivil = estadoCivil;
22 }
23 }
24

Ts Aluno.ts
1 import { Pessoa } from "./Pessoa";
2
3 export class Aluno extends Pessoa {
4     private _turma: string = "";
5
6     public falar(): string {
7         return "São muitas provas. Mimimi";
8     }
9
10    public set turma(turma: string) {
11        this._turma = turma;
12    }
13 }
14

Ts Principal.ts
1 import { Aluno } from "./Aluno";
2 import { Pessoa } from "./Pessoa";
3
4 const aluno: Pessoa = new Aluno();
5
6 aluno.
```

Tooltip for `aluno` (property) Pessoa: { _cpf: string; _endereco: string; _estadoCivil: string; falar: () string; nome: string; set: { nome: string; cpf: string; endereco: string; estadoCivil: string; } }

DESAFIO/CURIOSIDADE

Qual a saída agora? A fala da Pessoa ou do Aluno, neste caso?



```
1 import { Aluno } from "./Aluno";  
2 import { Pessoa } from "./Pessoa";  
3  
4 const aluno: Pessoa = new Aluno();  
5  
6 console.log("Aluno diz :>> ", aluno.falar());  
7
```

DESAFIO/CURIOSIDADE

Qual a saída agora? A fala da Pessoa ou do Aluno, neste caso?

```
1 import { Aluno } from "./Aluno";  
2 import { Pessoa } from "./Pessoa";  
3  
4 const aluno: Pessoa = new Aluno();  
5  
6 console.log("Aluno diz :>> ", aluno.falar());  
7
```

```
→ poo ts-node Principal.ts  
Aluno diz :>> São muitas provas. Mimimi  
→ poo
```

DESAFIO/CURIOSIDADE

Qual a saída agora? A fala da Pessoa ou do Aluno, neste caso?

```
1 import { Aluno } from "./Aluno";  
2 import { Pessoa } from "./Pessoa";  
3  
4 const aluno: Pessoa = new Aluno();  
5  
6 console.log("Aluno diz :>> ", aluno.falar());  
7
```

```
→ poo ts-node Principal.ts  
Aluno diz :>> São muitas provas. Mimimi  
→ poo
```

ACESSO À SUPERCLASSE

No caso do método sobrescrito, para acessar o método da superclasse, a única forma é utilizando a palavra chave `super`. No caso do "falar", ficaria da seguinte forma:

```
1 import { Pessoa } from "./Pessoa";
2
3 export class Aluno extends Pessoa {
4     private _turma: string = "";
5
6     public falar(): string {
7         return super.falar() + "\nSão muitas provas. Mimimi";
8     }
9
10    public set turma(turma: string) {
11        this._turma = turma;
12    }
13 }
```

A palavra-chave **super** provê acesso à superclasse.

ACESSO À SUPERCLASSE

No caso do método sobrescrito, para acessar o método da superclasse, a única forma é utilizando a palavra chave `super`. No caso do "falar", ficaria da seguinte forma:

```
• → poo ts-node Principal.ts  
Aluno diz :>> Sou uma pessoa  
São muitas provas. Mimimi  
○ → poo
```

```
1 import { Pessoa } from "../Pessoa";  
2  
3 export class Aluno extends Pessoa {  
4   private _turma: string = "";  
5  
6   public falar(): string {  
7     return super.falar() + "\nSão muitas provas. Mimimi";  
8   }  
9  
10  public set turma(turma: string) {  
11    this._turma = turma;  
12  }  
13 }
```

A palavra-chave **super** provê acesso à superclasse.

ACESSO À SUPERCLASSE

A palavra-chave **super** também pode ser utilizada para acessar o construtor da superclasse.

Altere sua superclasse **Pessoa** para ter um único construtor com nome e CPF. Analise as consequências e tente resolver o problema na subclasse.

SOLUÇÃO

Note que o construtor da subclasse precisa ter pelo menos o mesmo número de parâmetros do que a superclasse.

```
1 export class Pessoa {
2   protected _nome: string = "";
3   protected _cpf: string = "";
4   protected _endereco: string = "";
5   protected _estadoCivil: string = "";
6
7   constructor(nome: string, cpf: string) {
8     this._nome = nome;
9     this._cpf = cpf;
10  }
```

```
1 import { Pessoa } from "../Pessoa";
2
3 export class Aluno extends Pessoa {
4   private _turma: string = "";
5
6   constructor(nome: string, cpf: string) {
7     super(nome, cpf);
8   }
9 }
```

EXERCÍCIO 0

Você foi contratado para desenvolver um software para a empresa Mussa S/A. Na primeira reunião foram identificados alguns requisitos:

Gerenciar cliente - É necessário armazenar usuário, senha, nome, cpf, status, data do último acesso. O cliente que ficar sem acesso por 1 hora deve ficar inativo.

Gerenciar funcionário - É necessário armazenar usuário, senha, nome, cpf, status, data de admissão, data do último acesso. O funcionário que for desligado do sistema deve ficar inativo.

Gerenciar fornecedor - É necessário armazenar usuário, senha, nome, cnpj, status. O fornecedor pode ser inativo a qualquer momento pelo funcionário da empresa.

EXERCÍCIO 1

Crie um projeto para um banco. O banco possui dois tipos de contas: conta poupança e conta especial. Ambas possuem nome do cliente, número da conta e saldo. Também possui os métodos sacar (que não pode deixar o saldo negativo) e depositar.

O que as diferenciam é que a conta especial possui um atributo **limite**, e também sobrescreve o método de saque possibilitando que o saque deixe a conta negativa no total do limite. A conta poupança tem um **dia de rendimento** e possui um método que recebe a taxa de rendimento e atualiza o saldo da conta.

EXERCÍCIO 2

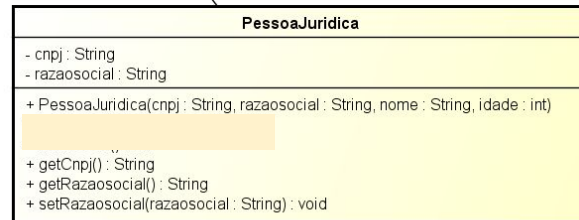
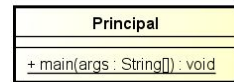
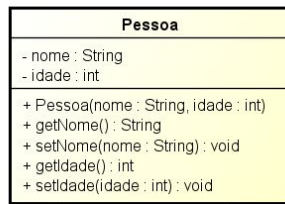
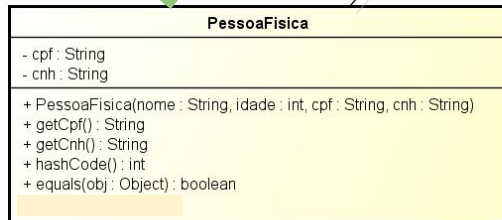
Desenvolva as classes Peixe, Cachorro, Gato e Gambá descritas a seguir. Nenhum atributo deve ser duplicado, então, utilize a quantidade de herança que for necessário, mas sempre mantendo a coerência do que está sendo desenvolvido.

EXERCÍCIO 2

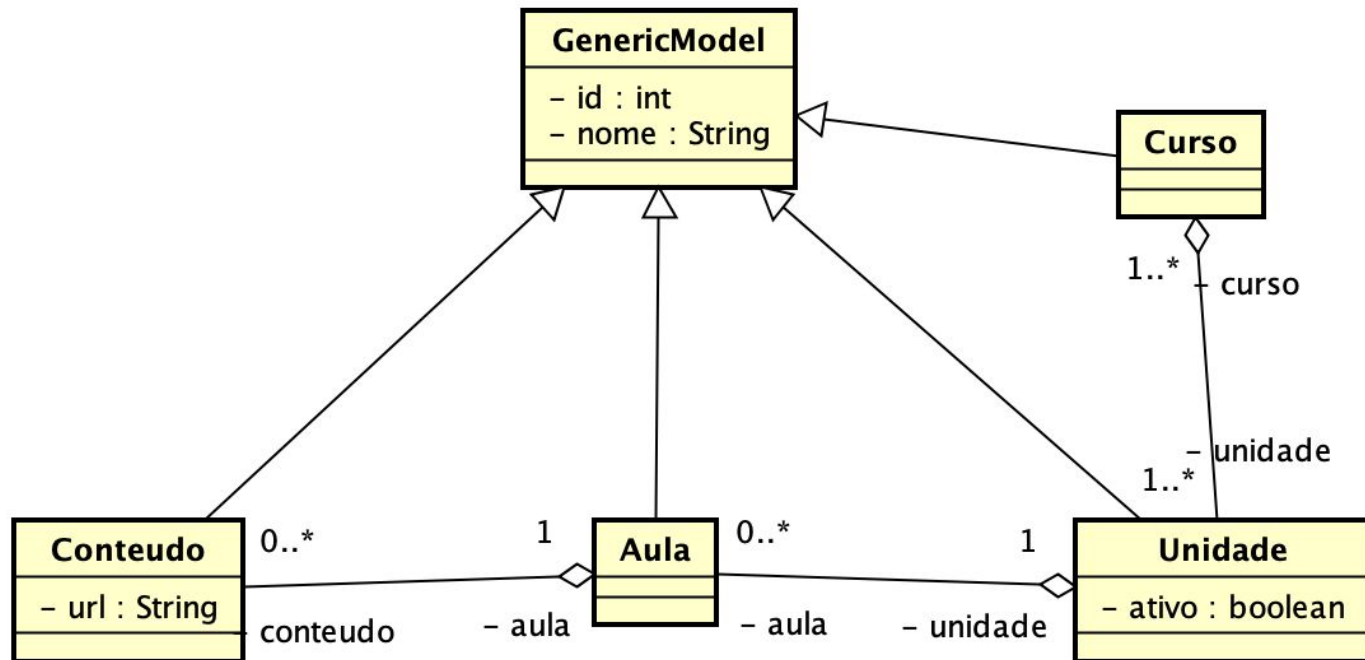
Peixe	Cachorro	Gato	Gambá
<p>Características: peso idade sexo escamas (boolean)</p> <p>Comportamentos: movimentar comer (aumenta o peso em 5%) envelhecer (em um ano)</p>	<p>Características: peso idade sexo nome pedigree (boolean)</p> <p>Comportamentos: movimentar latir comer(aumenta o peso em 10%) envelhecer(um um ano)</p>	<p>Características: peso idade sexo nome pedigree</p> <p>Comportamentos: movimentar miar comer(aumenta o peso em 5%) envelhecer(um um ano)</p>	<p>Características: peso idade sexo nível de aroma (0-10)</p> <p>Comportamentos: movimentar aromatizar comer(aumenta o peso em 5%) envelhecer(um um ano)</p>
<p>Informações adicionais: movimentar deve apenas retornar uma mensagem específica do animal. Ex.: O peixe está nadando.</p>			

EXERCÍCIO 3

Se liga! Os métodos nos diagramas com o mesmo nome da classe, representam os construtores



EXERCÍCIO 4



NÃO ESQUEÇA!

Realize uma análise no seu projeto da UC e aplique a herança neste.

MATERIAL PRÁTICO DE APOIO

