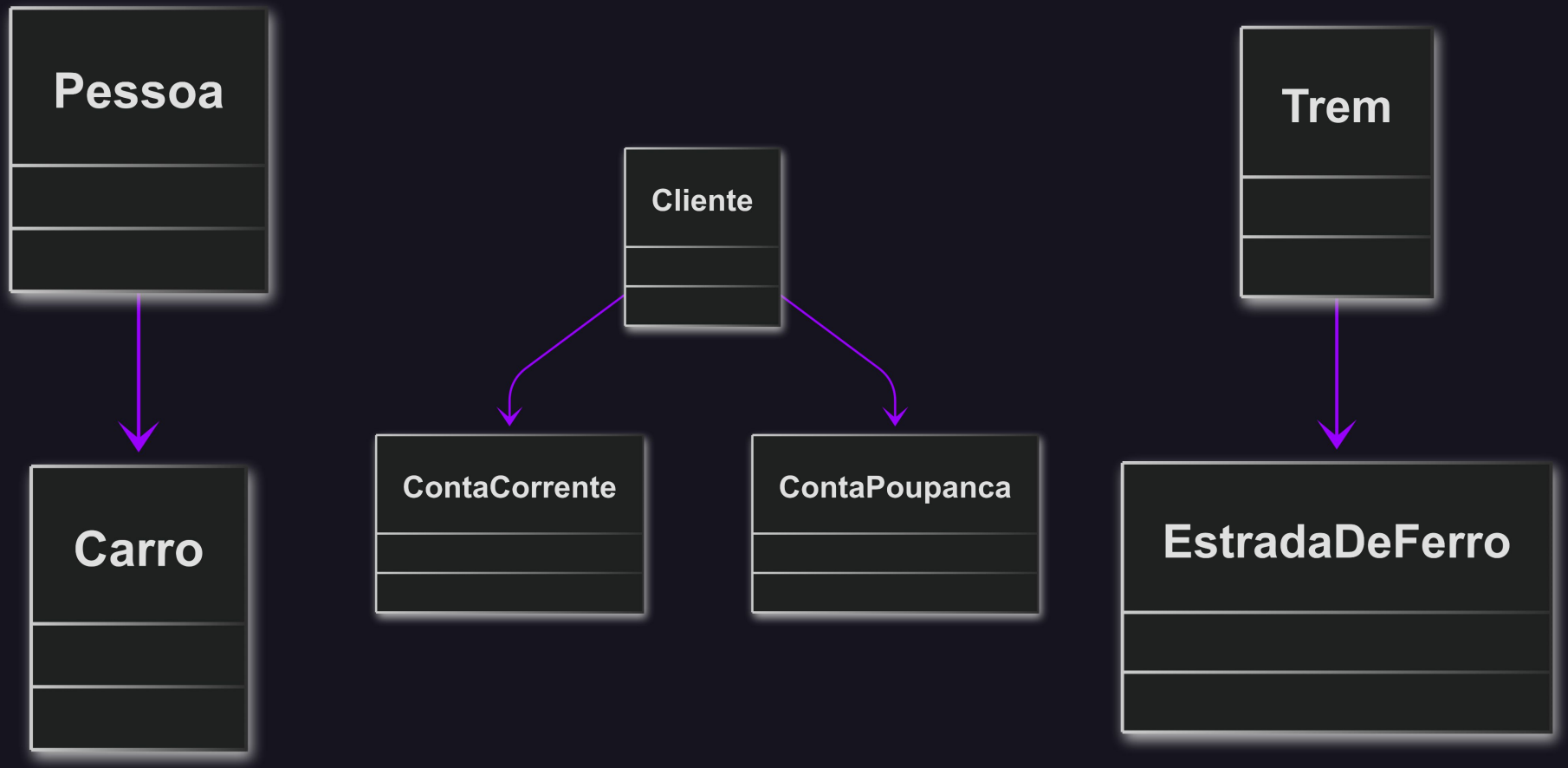
Relacionamento Entre Classes e Herança

Matheus Barbosa
matheus.barbosa@dcx.ufpb.br

Um sistema orientado a objetos é composto por dezenas de classes que se relacionam entre si para solucionar o problema proposto

Uma **associação** indica algum relacionamento significativo e de interesse entre objetos


Pessoa
Carro
Cliente
ContaCorrente
ContaPoupanca
Trem
EstradaDeFerro

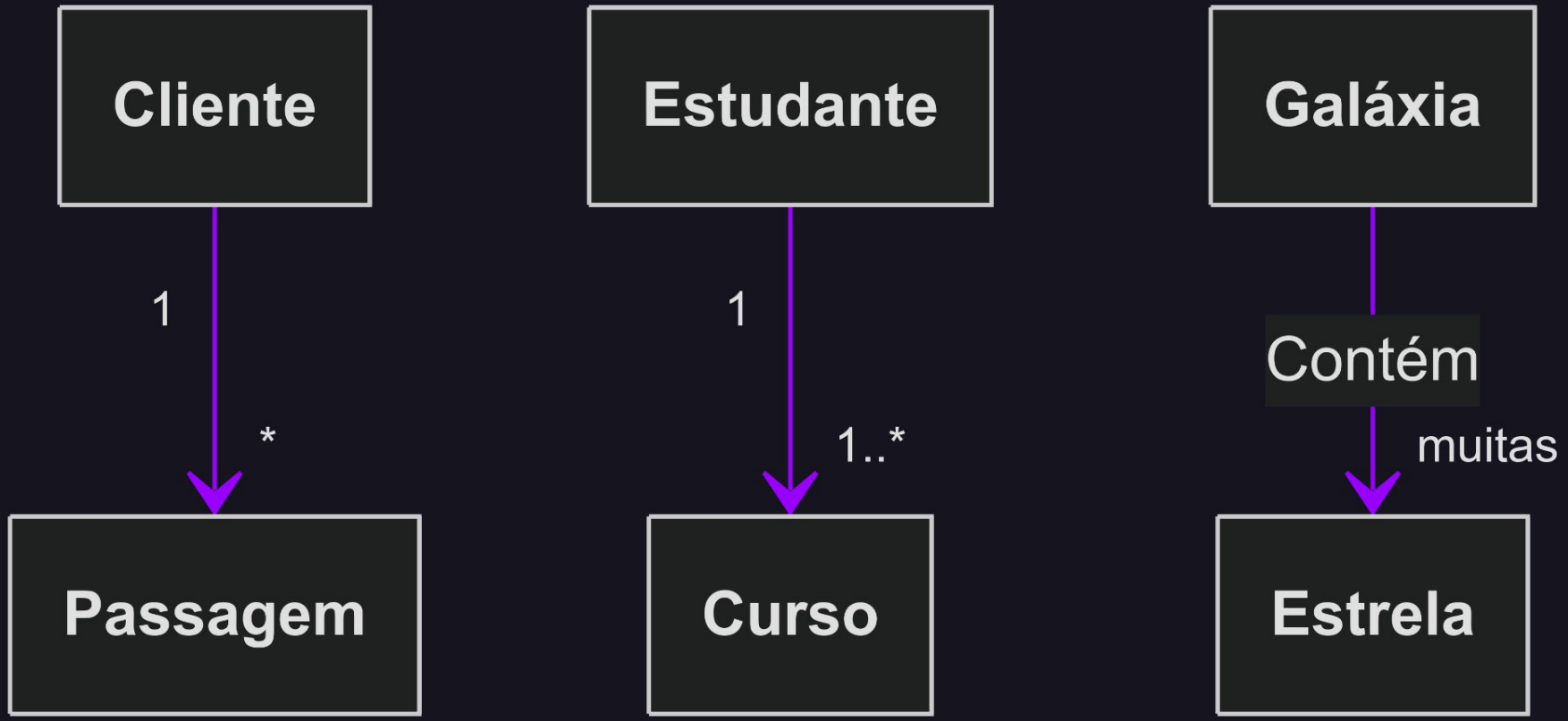
A **multiplicidade** descreve quantas instâncias de uma classe podem estar associadas a uma instância de outra classe.

Tipos de Multiplicidade

- **1:1**: Um objeto de uma classe está relacionado a exatamente um objeto de outra classe

- **1:N**: Um objeto de uma classe pode estar relacionado a vários objetos de outra classe (um para muitos).

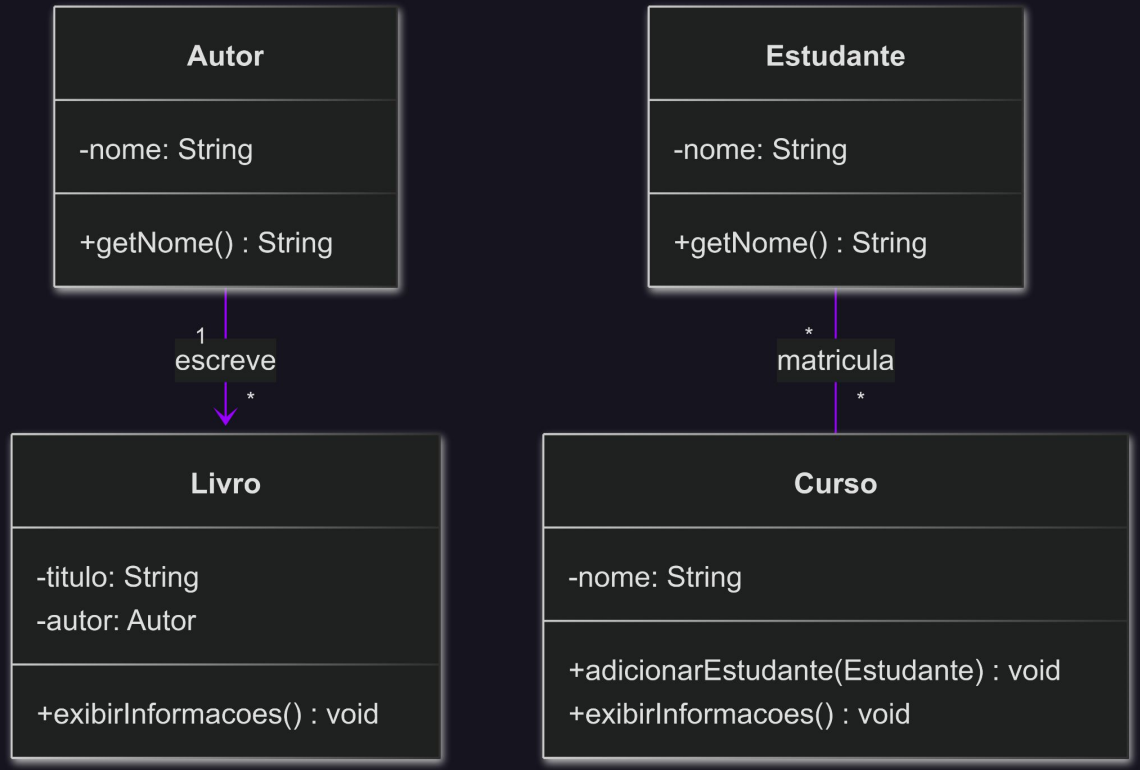
- **N:N**: Vários objetos de uma classe podem estar relacionados a vários objetos de outra classe (muitos para muitos).


Cliente
1
*
Passagem
Estudante
1
1..*
Curso
Galáxia
Contém
muitas
Estrela

🎯 Exemplo prático

Um Autor pode escrever vários Livros, mas cada Livro tem exatamente um Autor

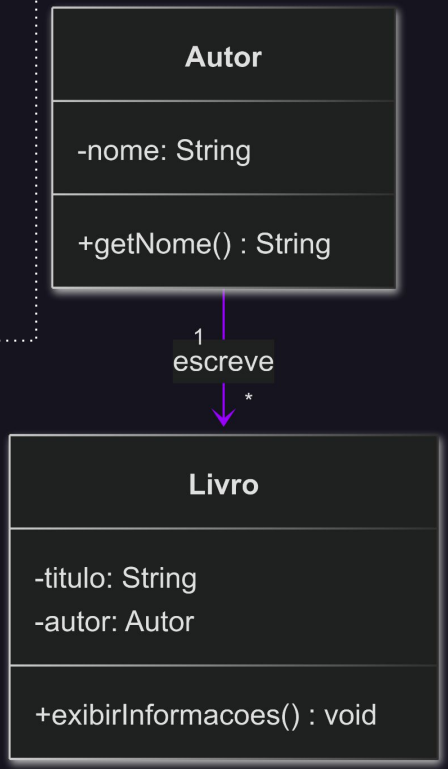
Muitos Estudantes podem estar matriculados em muitos Cursos, e um Curso pode ter muitos Estudantes



🎯 Autor e Livro

```
Classe Autor
   privado nome: texto
   // Métodos especiais…
FimClasse
```

```
Classe Livro
   privado titulo: texto
   privado autor: Autor // Um autor para muitos livros

   publico exibirInformacoes()

   // Métodos especiais…
FimClasse
```

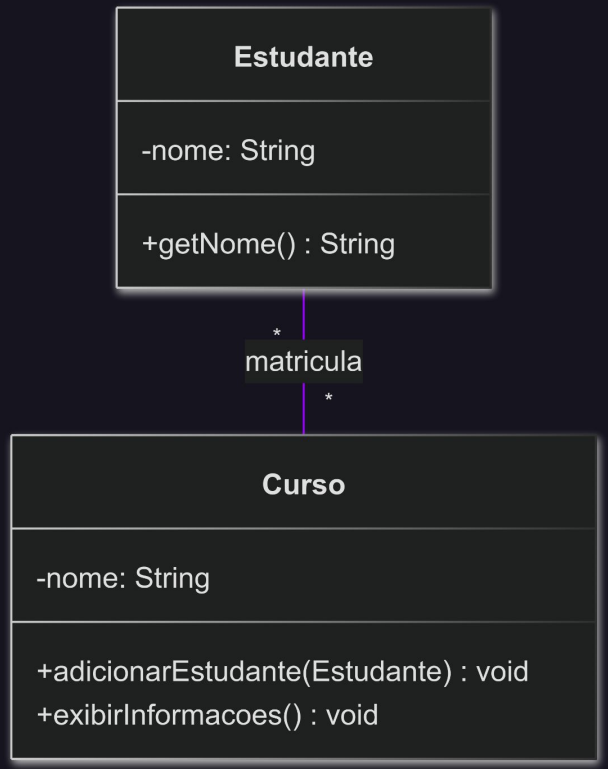



```
autor = novo Autor("J.K. Rowling")
livro1 = novo Livro("Harry Potter e a Pedra Filosofal", autor)
livro2 = novo Livro("Harry Potter e a Câmara Secreta", autor)

livro1.exibirInformacoes()
livro2.exibirInformacoes();
```

🎯 Estudante e Curso

```
Classe Estudante
   privado nome: texto
   // Métodos especiais…
FimClasse
```

```
Classe Curso
   privado nome: texto
   privado autor: Lista<Estudante> // Muitos estudantes podem estar matriculados em muitos cursos.

   publico exibirInformacoes()
   publico adicionarEstudante(Estudante e)

   // Métodos especiais…
FimClasse
```



🎯 Estudante e Curso

```
Classe Estudante
   privado nome: texto
   // Métodos especiais…
FimClasse
```

```
Classe Curso
   privado nome: texto
   privado autor: Lista<Estudante> // Muitos estudantes podem estar matriculados em muitos cursos.

   publico exibirInformacoes()
   publico adicionarEstudante(Estudante e)

   // Métodos especiais…
FimClasse
```

```
estudante1 = novo Estudante("Ana");
estudante2 = novo Estudante("Carlos");

curso = novo Curso("Computação")

curso.adicionarEstudante(estudante1)
curso.adicionarEstudante(estudante2)

curso.exibirInformacoes()
```

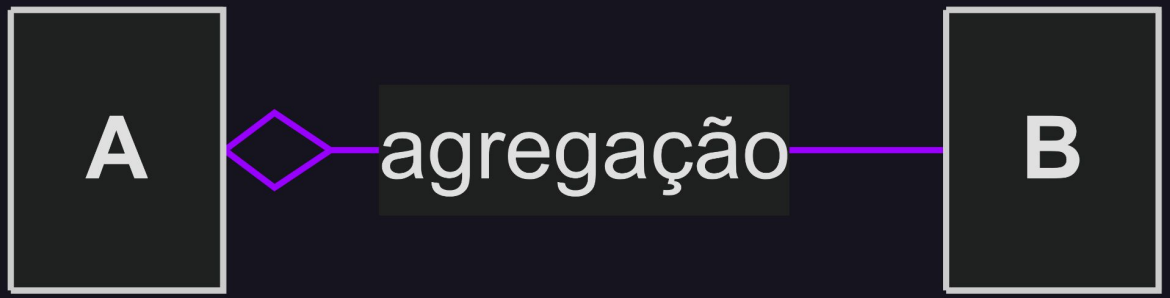
Tipos de relacionamento

- Agregação
- Composição
- Herança

Agregação é uma associação onde o objeto "pai" contém objetos "filhos", mas eles podem existir sozinhos. Se o "pai" for destruído, os "filhos" continuam existindo.

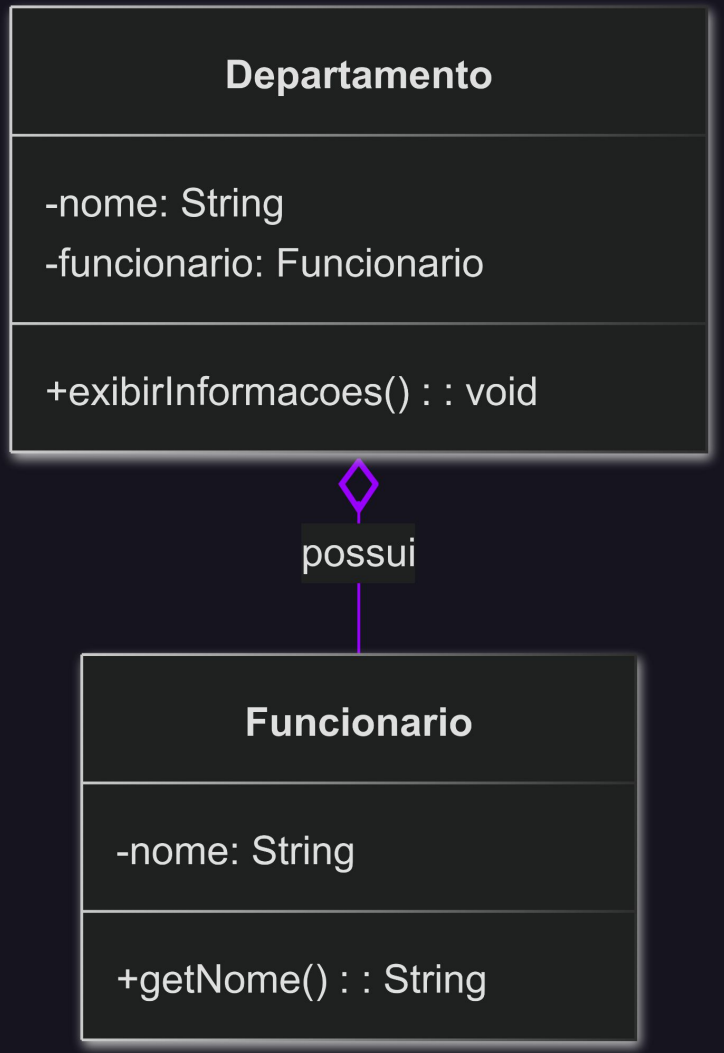
A relação é de: “*é parte de*”;

É representado por um losango vazio junto ao objeto representando o todo.



🎯 Exemplo prático

Um departamento e seus funcionários. O departamento pode existir sem os funcionários, e os funcionários podem existir sem o departamento.



🎯 Departamento e Funcionário

```
Classe Funcionario
   privado nome: texto
   // Métodos especiais…
FimClasse
```

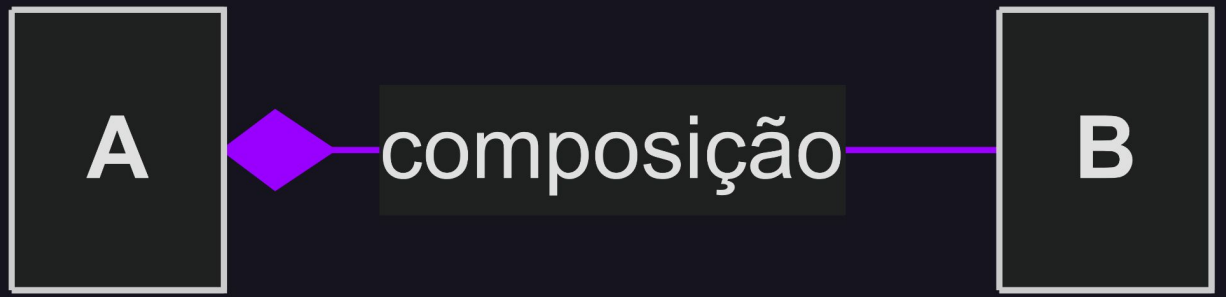
```
Classe Departamento
   privado nome: texto
   privado funcionario: Funcionario

   Departamento(nome, funcionario){
     nome = nome;
     funcionario = funcionario // Mesmo que
o Departamento seja destruído, o Funcionário
pode continuar existindo.
   }
FimClasse
```

```
funcionario = novo Funcionario("Ana");
departamento = novo Departamento("RH", funcionario);

departamento.exibirInformacoes()
```

Composição é uma associação forte em que os objetos "filhos" dependem do "pai".
Se o objeto "pai" for destruído, os "filhos" também são.

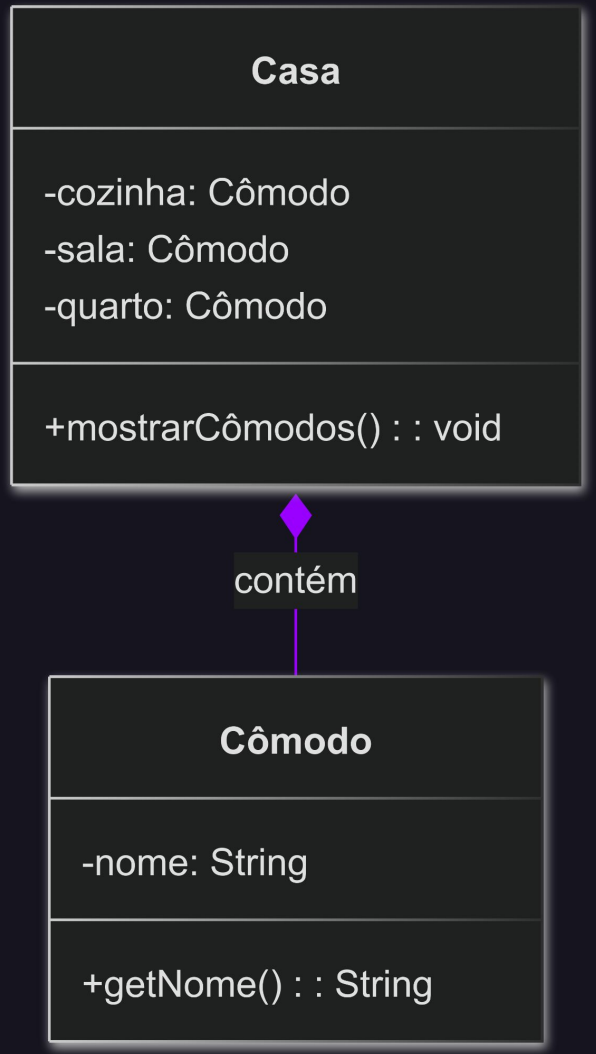
Um objeto “*é parte essencial*” de outro.
O objeto composto não existe sem os seus componentes.

É representado por um losango preenchido em preto



🎯 Exemplo prático

Uma casa e seus cômodos. Se a casa for destruída, os cômodos também serão destruídos, pois não existem independentemente.



🎯 Casa e Cômodo

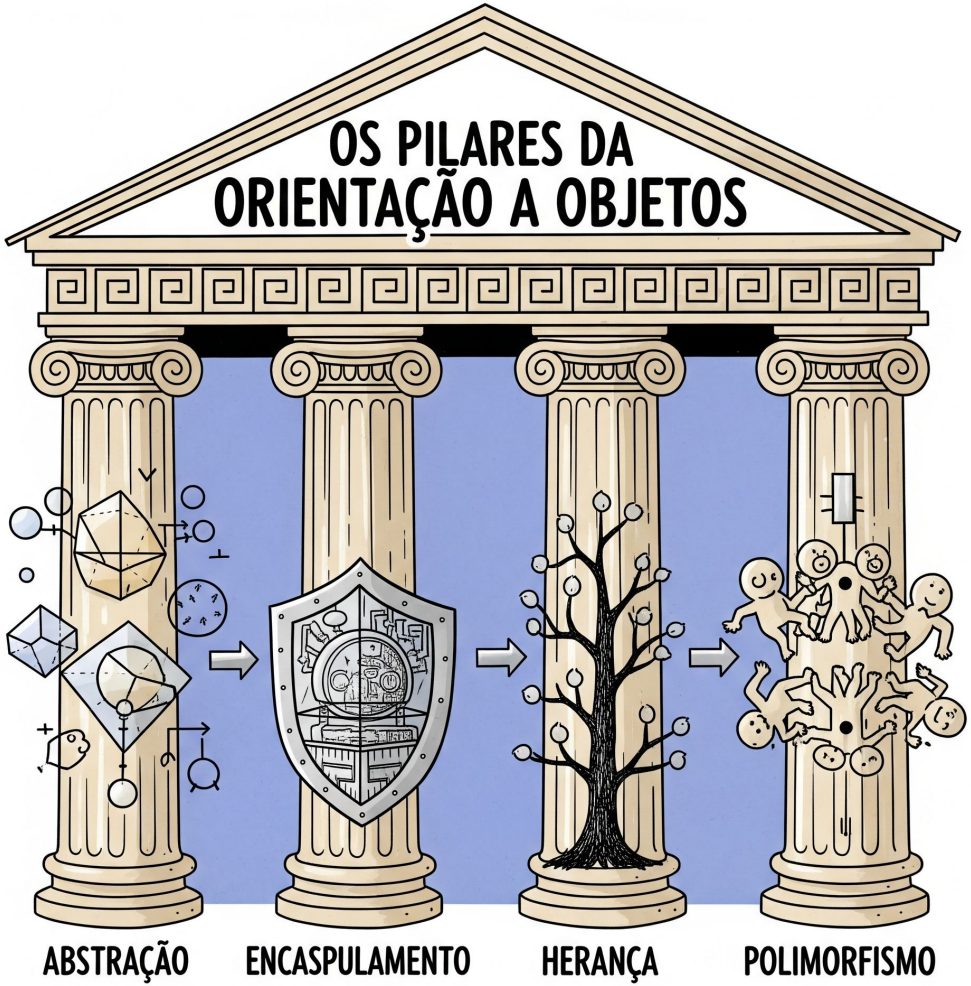
```
Classe Cômodo
   privado nome: texto
   // Métodos especiais…
FimClasse
```

```
Classe Casa
   privado cozinha: Cômodo
   privado sala: Cômodo

   // Construtor que cria os cômodos como parte da casa
   Casa(){
     cozinha = novo Cômodo();
     sala = novo Cômodo();
   }
FimClasse
```

```
casa = nova Casa();
casa.mostrarCômodos();
// Se a casa for destruída, os cômodos também seriam.
```

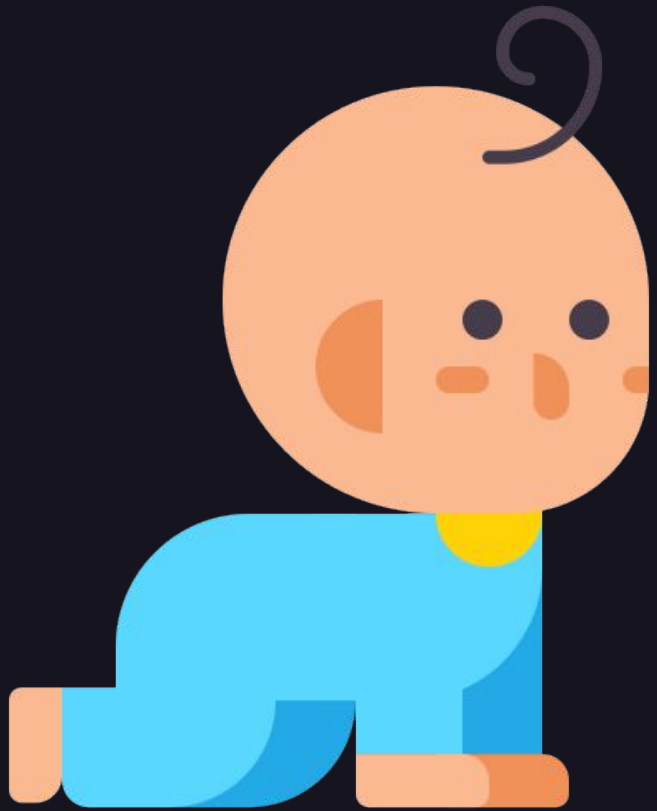
Obs.: Em geral, na prática, é mais comum usar agregação mesmo quando o relacionamento é mais forte

Herança


OS PILARES DA
ORIENTAÇÃO A OBJETOS
ABSTRAÇÃO
ENCASPULAMENTO
HERANÇA
POLIMORFISMO

Herança

Mãe



Filho(a)

Permite basear uma nova classe na definição de outra classe previamente existente

Herança é aplicada tanto para características como para comportamentos.

Peixe

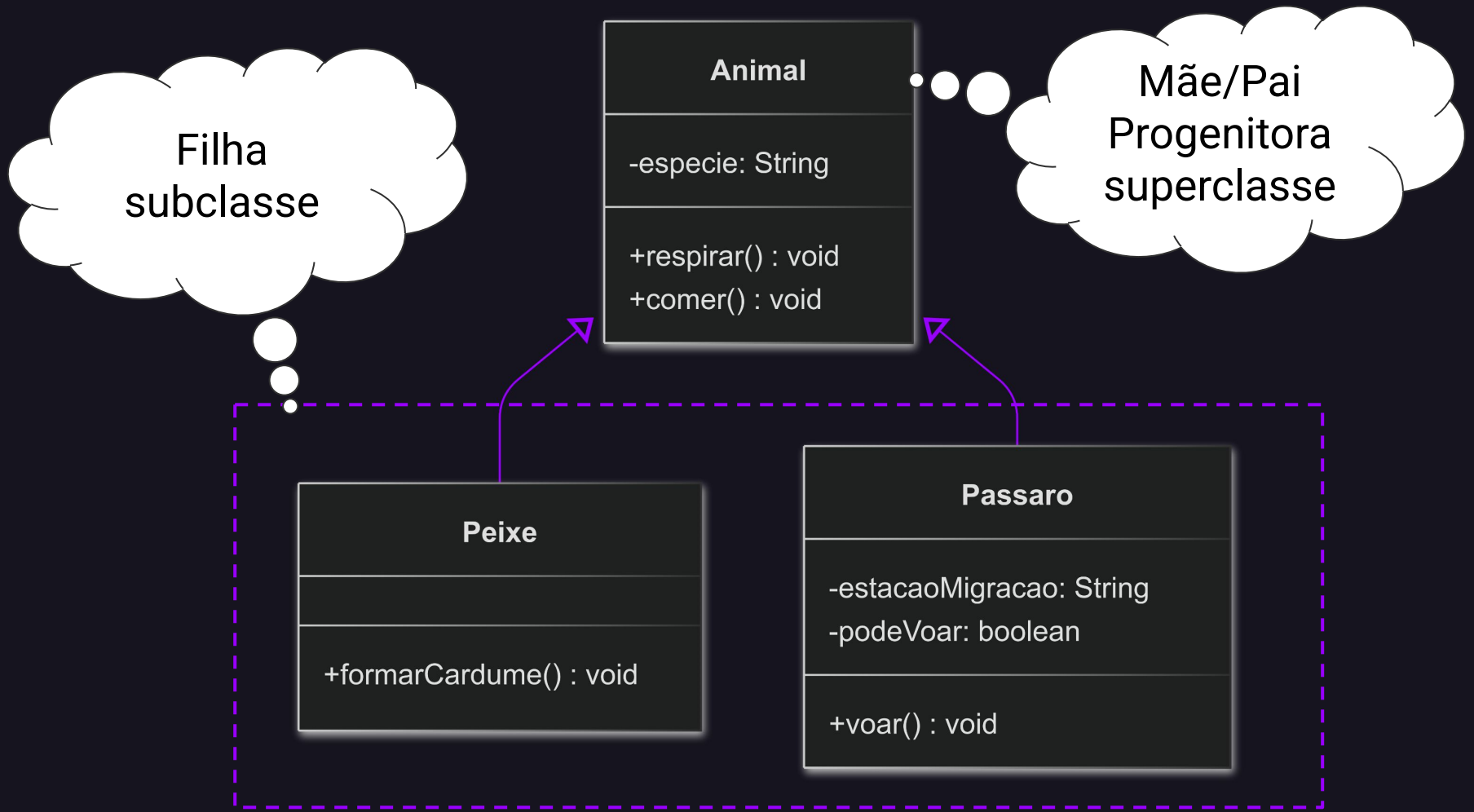
-especie: String

+respirar() : void
+comer() : void
+formarCardume() : void

Passaro

-estacaoMigracao: String
-podeVoar: boolean
-especie: String

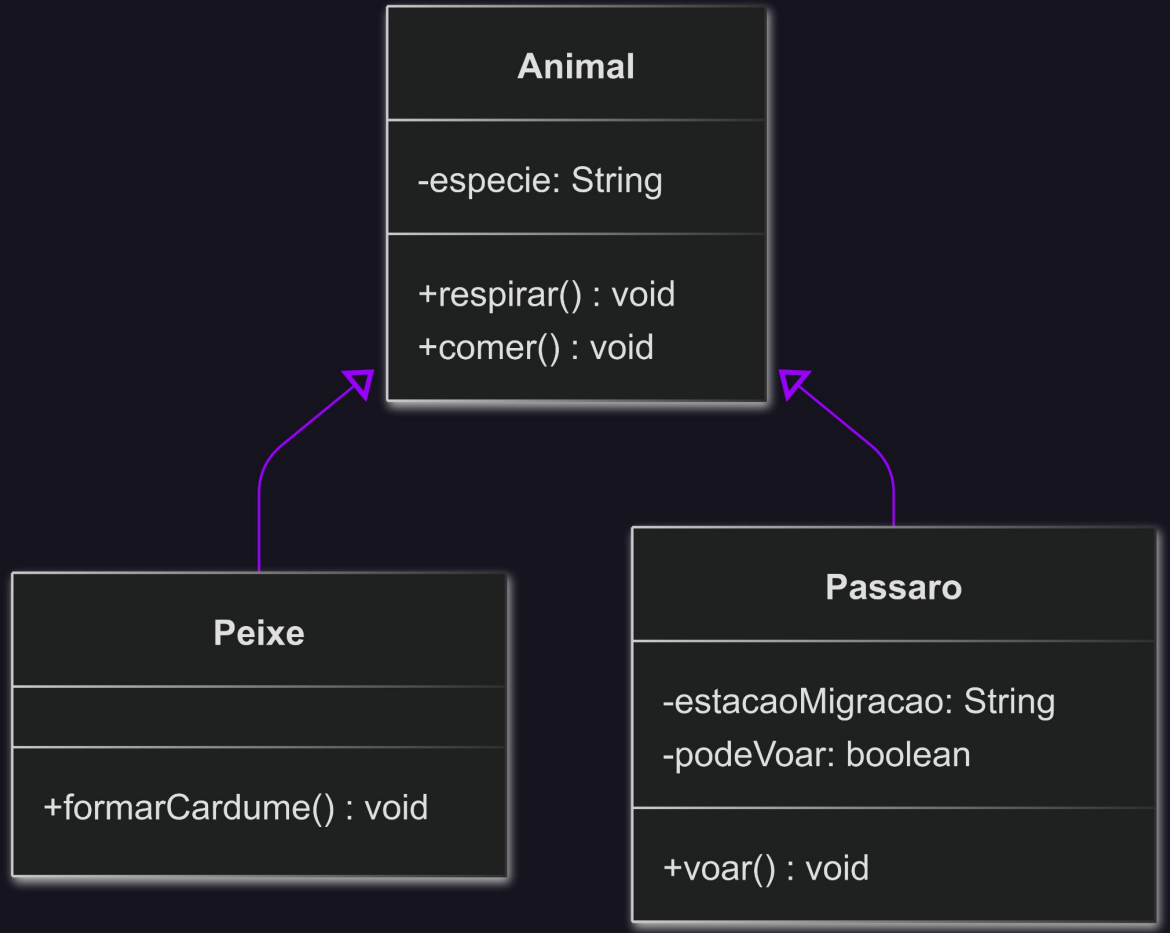
+respirar() : void
+comer() : void
+voar() : void


Filha
subclasse
Animal
-especie: String
+respirar() : void
+comer() : void
Mãe/Pai
Progenitora
superclasse
Peixe
+formarCardume() : void
Passaro
-estacaoMigracao: String
-podeVoar: boolean
+voar() : void

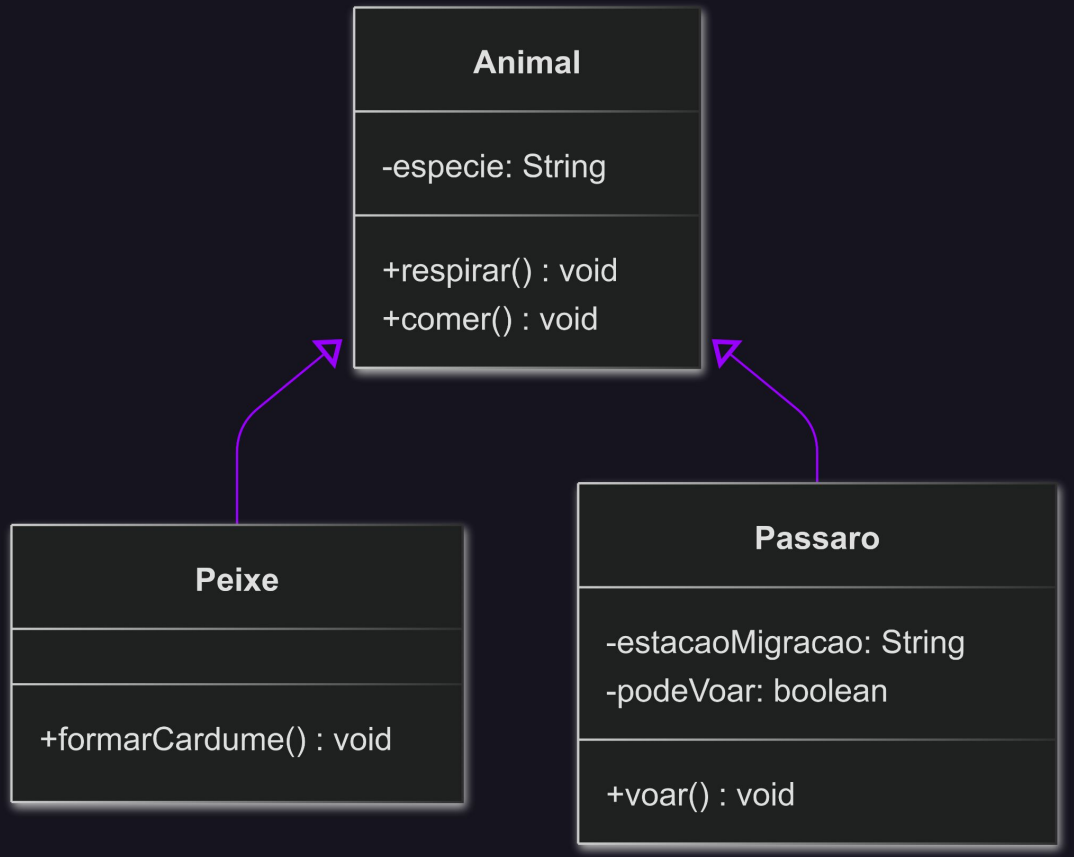
```
Classe Animal
   privado especie: texto

   publico rabiscar()
   publico comer()

   // Métodos especiais…
FimClasse
```

```
Classe Peixe estende Animal
   publico formarCardume()

   // Métodos especiais…
FimClasse
```

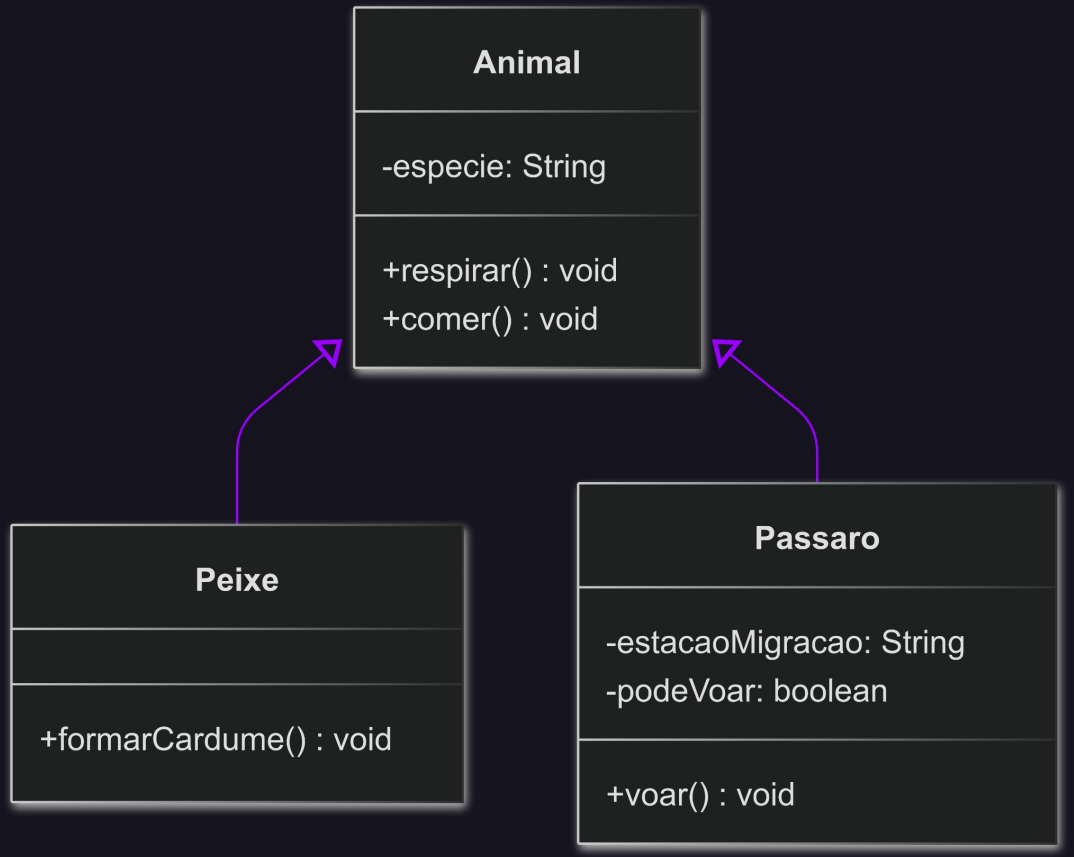
```
Classe Passaro estende Animal
   privado estacaoMigracao: texto
   privado podeVoar: logico

   publico voar()

   // Métodos especiais…
FimClasse
```

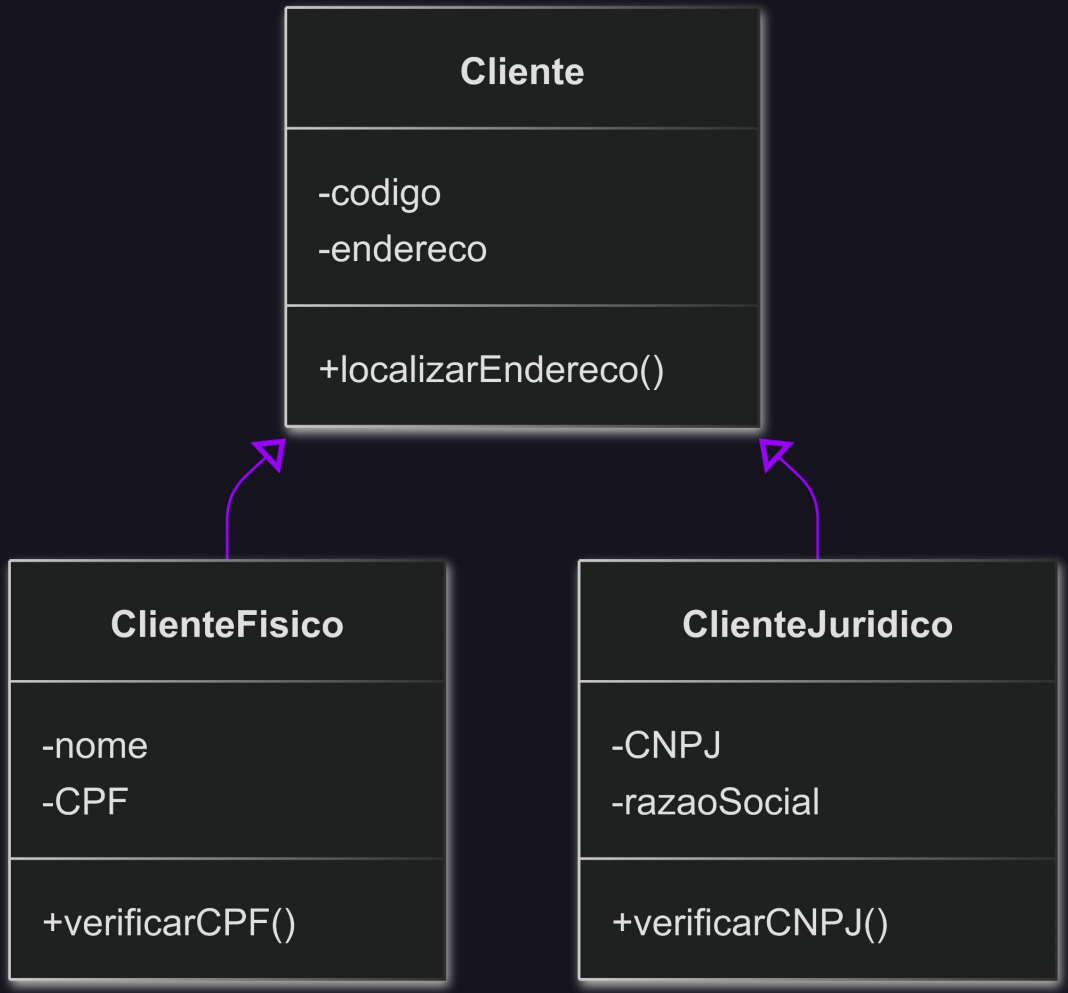


```
animal = novo Animal()
peixe = novo Peixe()
passaro = novo Passaro()

animal.setEspecie(“Labrador”)
peixe.setEspecie(“Peixe-palhaço”)
passaro.setEspecie(“Bem-te-vi”)

animal.comer()
peixe.comer()
passaro.comer()

animal.respirar()
peixe.formarCardume()
passaro.voar()

animal.voar()
peixe.voar()
passaro.formarCardume()
```



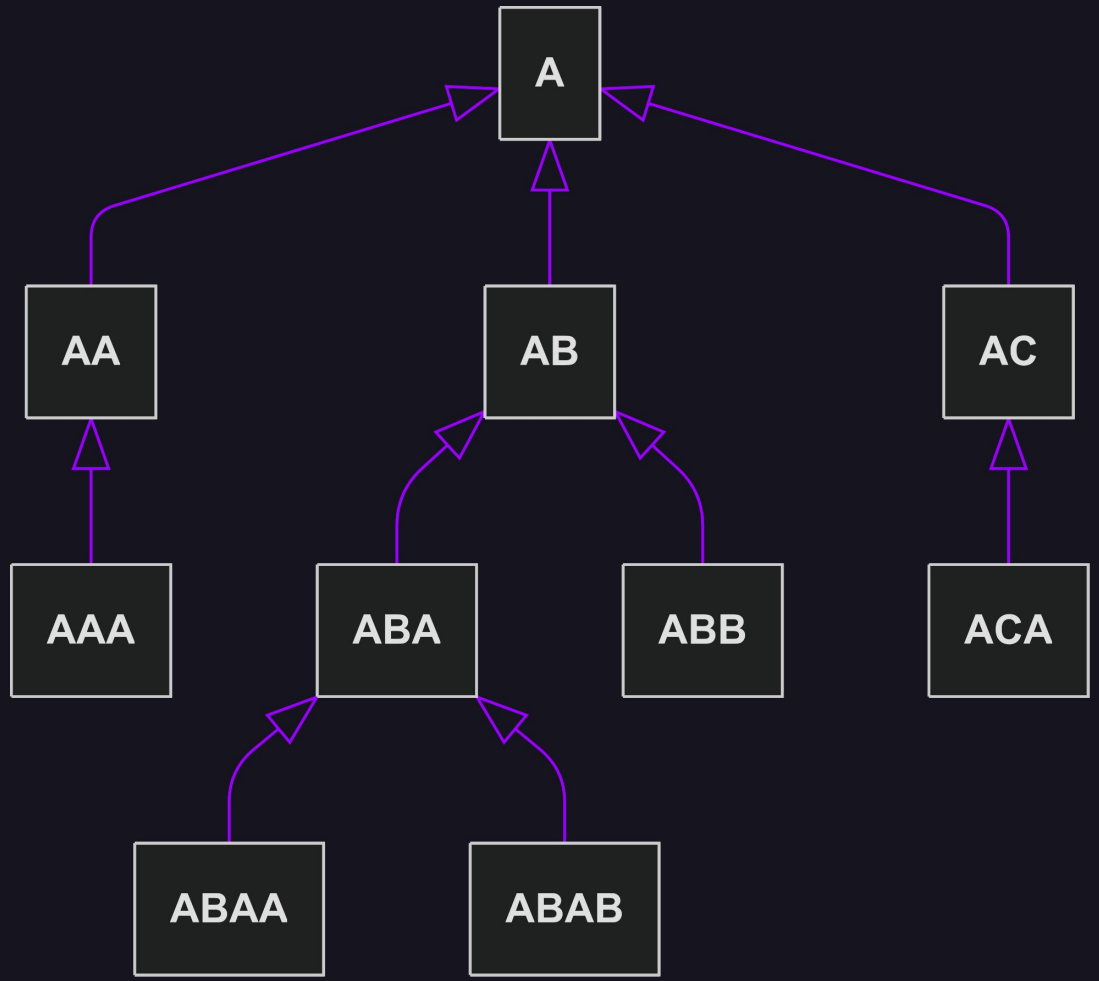
```
animal = novo Animal()
peixe = novo Peixe()
passaro = novo Passaro()

animal.setEspecie(“Labrador”)
peixe.setEspecie(“Peixe-palhaço”)
passaro.setEspecie(“Bem-te-vi”)

animal.comer()
peixe.comer()
passaro.comer()

animal.respirar()
peixe.formarCardume()
passaro.voar()

~~animal.voar()~~
~~peixe.voar()~~
~~passaro.formarCardume()~~
```



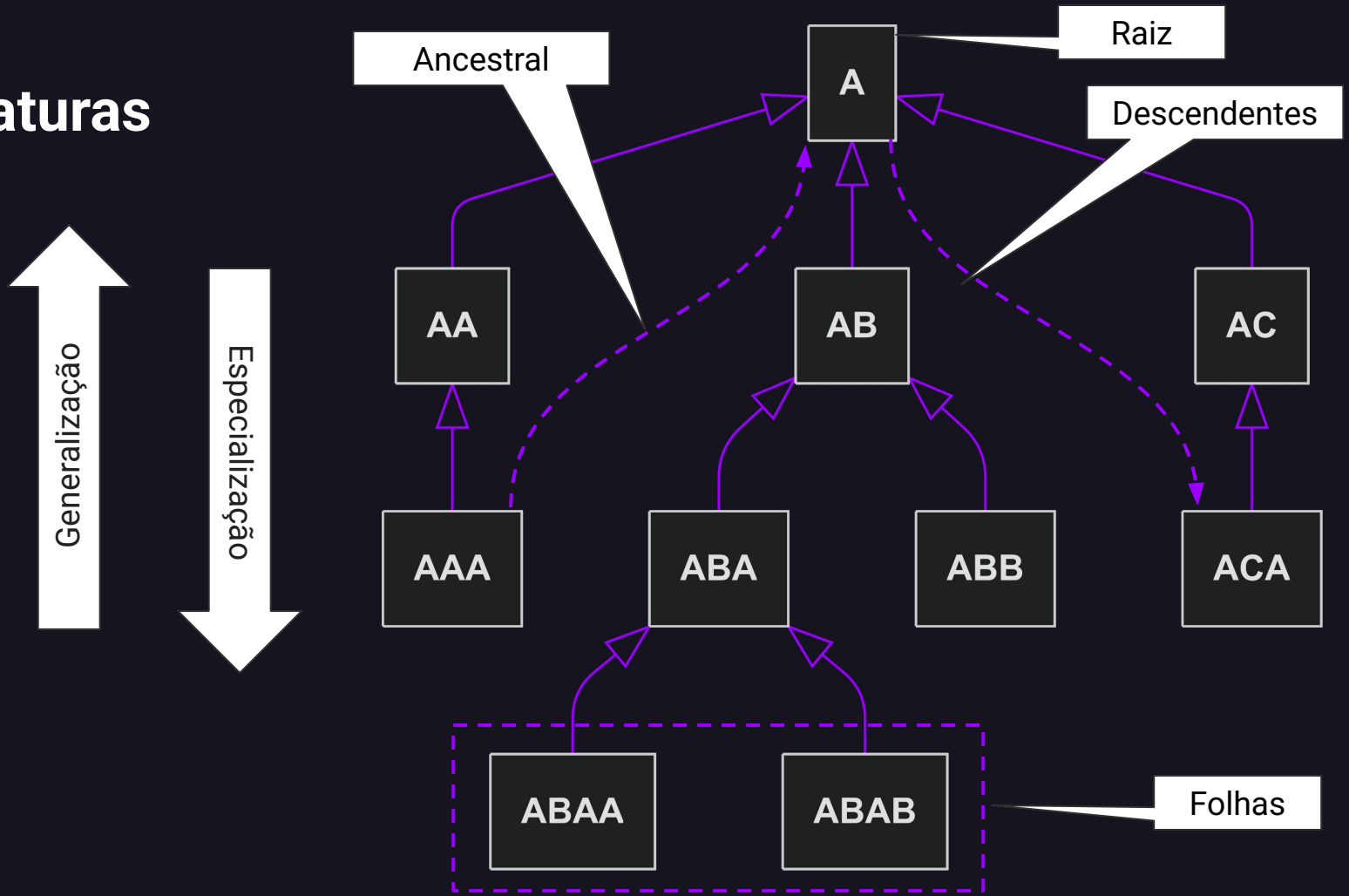

Cliente
-codigo
-endereco
+localizarEndereco()
ClienteFisico
-nome
-CPF
+verificarCPF()
ClienteJuridico
-CNPJ
-razaoSocial
+verificarCNPJ()

A subclasse pode reutilizar, estender ou sobrescrever o comportamento herdado.

Navegação pela herança



Nomenclaturas



Toda subclasse herda todos os conteúdos dos seus ancestrais, mesmo que não esteja totalmente disponível.

Ou seja, os campos privados são herdados, mas como só podem ser acessados e modificados pelas classes que os declararam diretamente, não podem ser acessados diretamente pela classe herdeira.

Abstrato e final

Classe Abstrata	Método Abstrato	Classe Final	Método Final
Não pode ser instanciada. Só deve servir como superclasse.	Declarado, mas não implementado na superclasse.	Não pode ser herdada por outra classe. Obrigatoriamente folha.	Não pode ser sobrescrito pelas suas sub-classes. Obrigatoriamente herdado.

🎯 Contexto: Clínica Veterinária

Na clínica veterinária, todos os animais atendidos devem passar por um exame padrão, que inclui observações básicas como a respiração e o comportamento sonoro do animal.
Para organizar o sistema da clínica, a equipe de TI modelou o seguinte:

- Existe uma classe **abstrata** chamada `Animal`, que define o que todos os animais têm em comum (como o nome e o ato de respirar).
- Como cada animal emite um som diferente, o método `emitirSom()` é **abstrato**, forçando cada tipo específico de animal a implementar o som que faz.
- Um exemplo de animal atendido é o `Cachorro`, que emite o som "Au Au".
- O atendimento dos animais é feito por um `Veterinário` **(classe final)**, que não pode ser herdado ou alterado (por segurança do sistema), pois encapsula regras fixas de atendimento.
- Durante o exame, o veterinário chama o **método final** `respirar()` (que não pode ser alterado em subclasses) e o método emitirSom() (que depende do tipo de animal).

🎯 Contexto: Clínica Veterinária

```
Classe abstrata Animal
   privado nome: texto

   publico final respirar()
   publico abstrato emitirSom()

   // Métodos especiais…
FimClasse
```

```
Classe final Veterinario
   publico examinar(Animal a){
      a.respirar()
      a.emitirSom()
   }
FimClasse
```

```
Classe Cachorro estende Animal
   @Sobrepor
   publico emitirSom(){
      print("AUAU")
   }
FimClasse
```

```
~~Animal animal = novo Animal()~~
Animal cachorro = novo Cachorro()
Veterinario v = novo Veterinario()
v.examinar(cachorro)
```