Herança

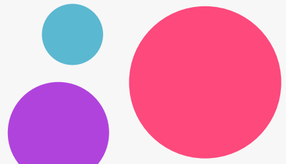
Aula 09 - Programação orientada a objetos

Professor Daniel Marques

INSTITUTO FEDERAL
Paraíba
Campus Campina Grande

Conteúdo da aula

1. Introdução
2. Generalização e especialização
3. Herança em diagramas de classe
4. Sintaxe de herança em C++
5. Herança múltipla

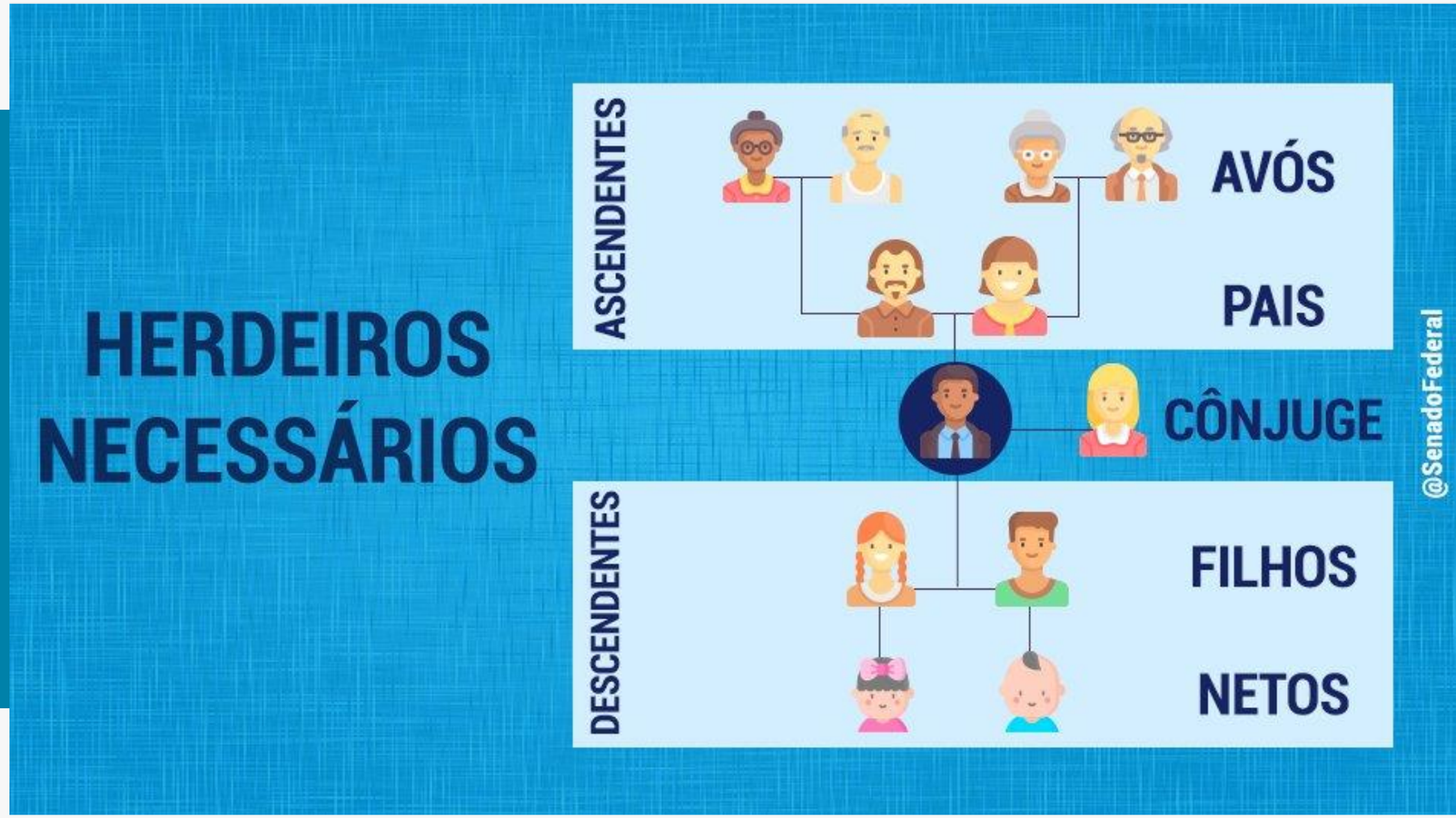


1

Herança

O que é herança?

Imagine sua família, note que você possui características que herdou do seu pai, da sua mãe, do seu avô e avó.



E note que caso um dia você tenha filhas ou filhos biológicos, eles herdaram suas características genéticas, como altura, formato do rosto, similaridades corporais.

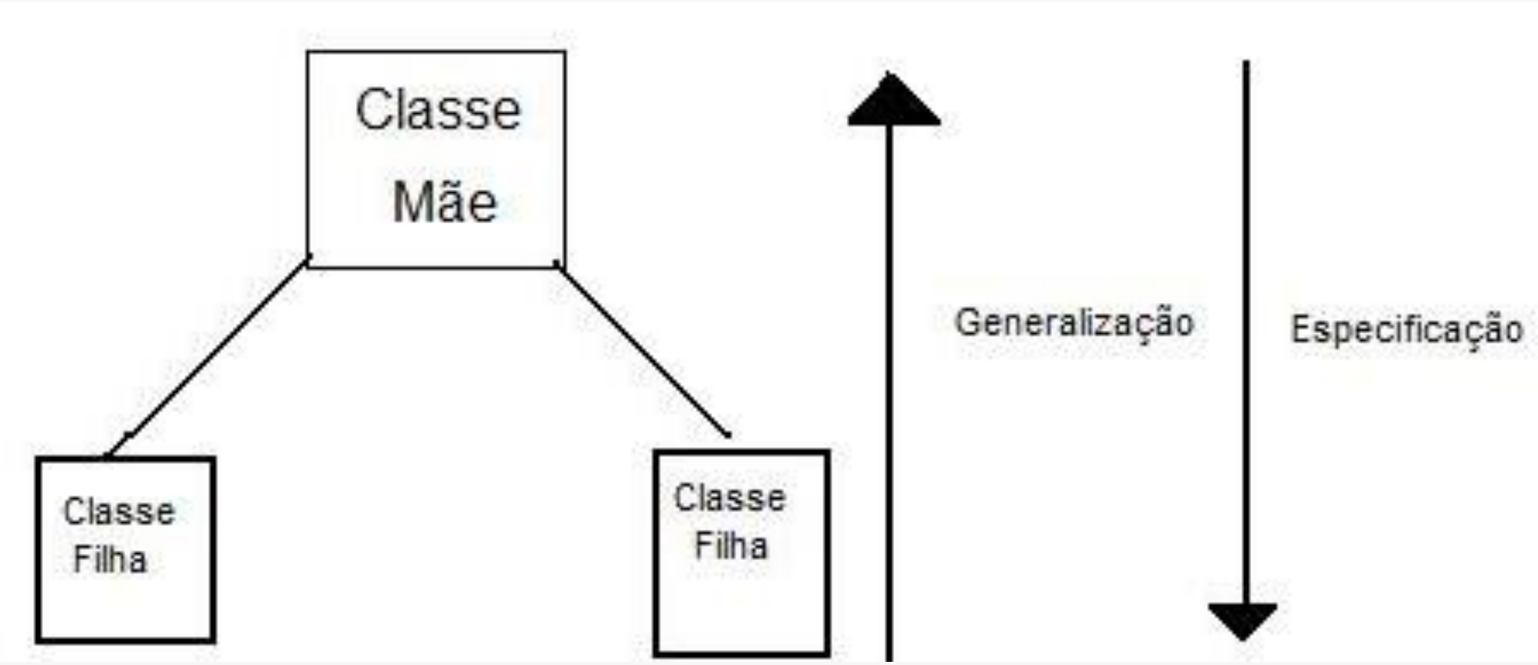
O que é herança?

Possíveis características compartilhadas entre mãe e filha: formato do nariz, dos lábios, tipo sanguíneo

Herança em programação orientada a objetos

A herança é uma outra forma de reuso de software em POO.

Propriedades e métodos são herdados, evitando sua duplicação em várias classes similares



Novas classes são criadas a partir das classes existentes

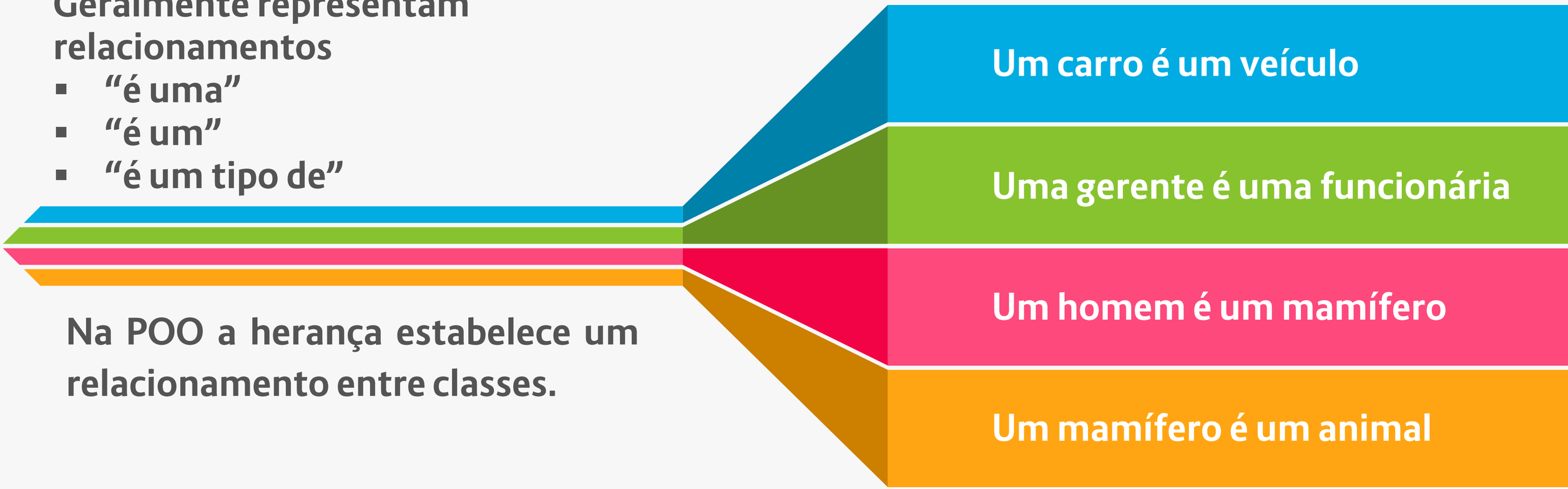
- reutilizando seus atributos e métodos não-privados
- adicionando novos recursos que as novas classes exigem

Herança em programação orientada a objetos

Exemplos de relacionamentos de herança no mundo real

Geralmente representam relacionamentos

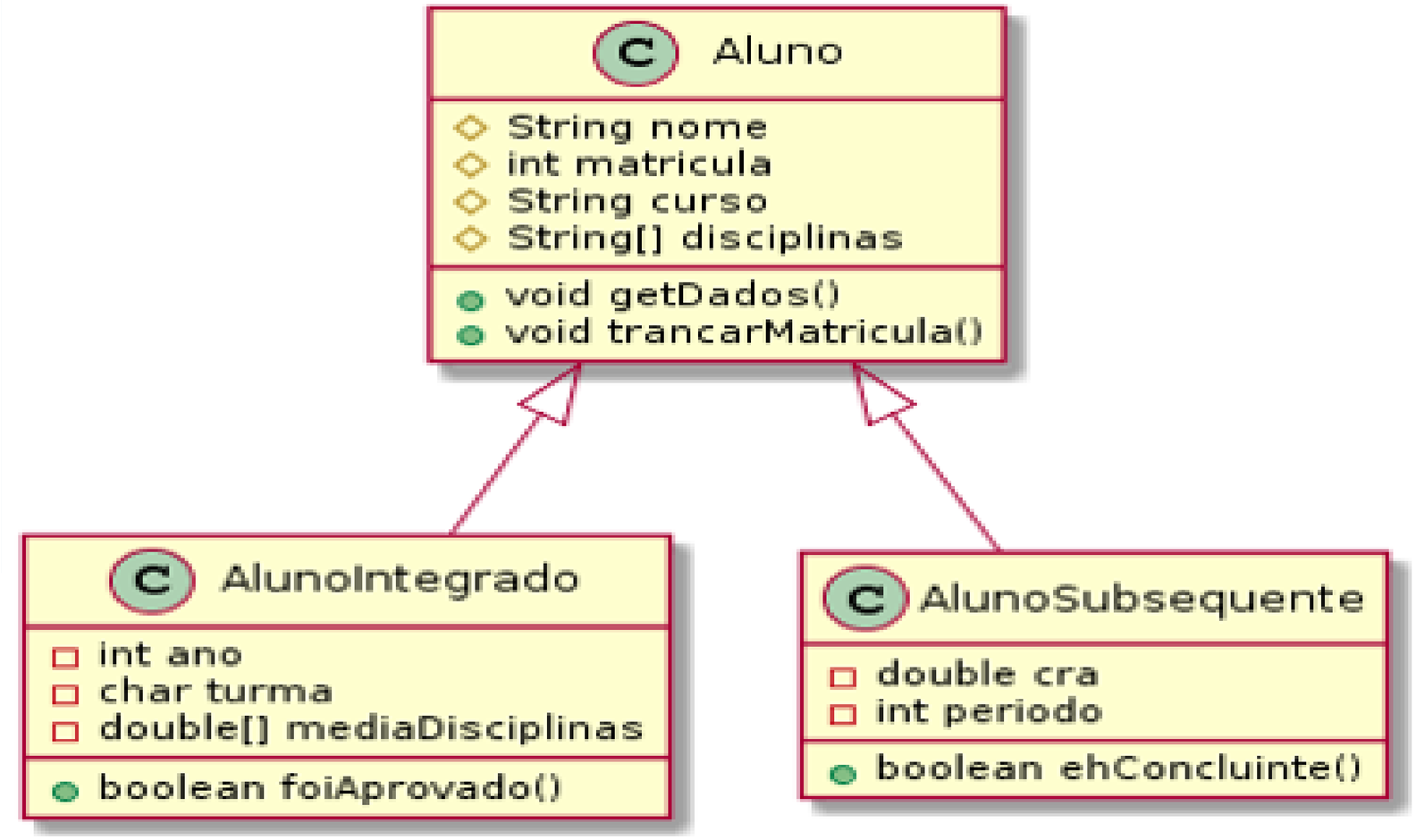
- “é uma”
- “é um”
- “é um tipo de”



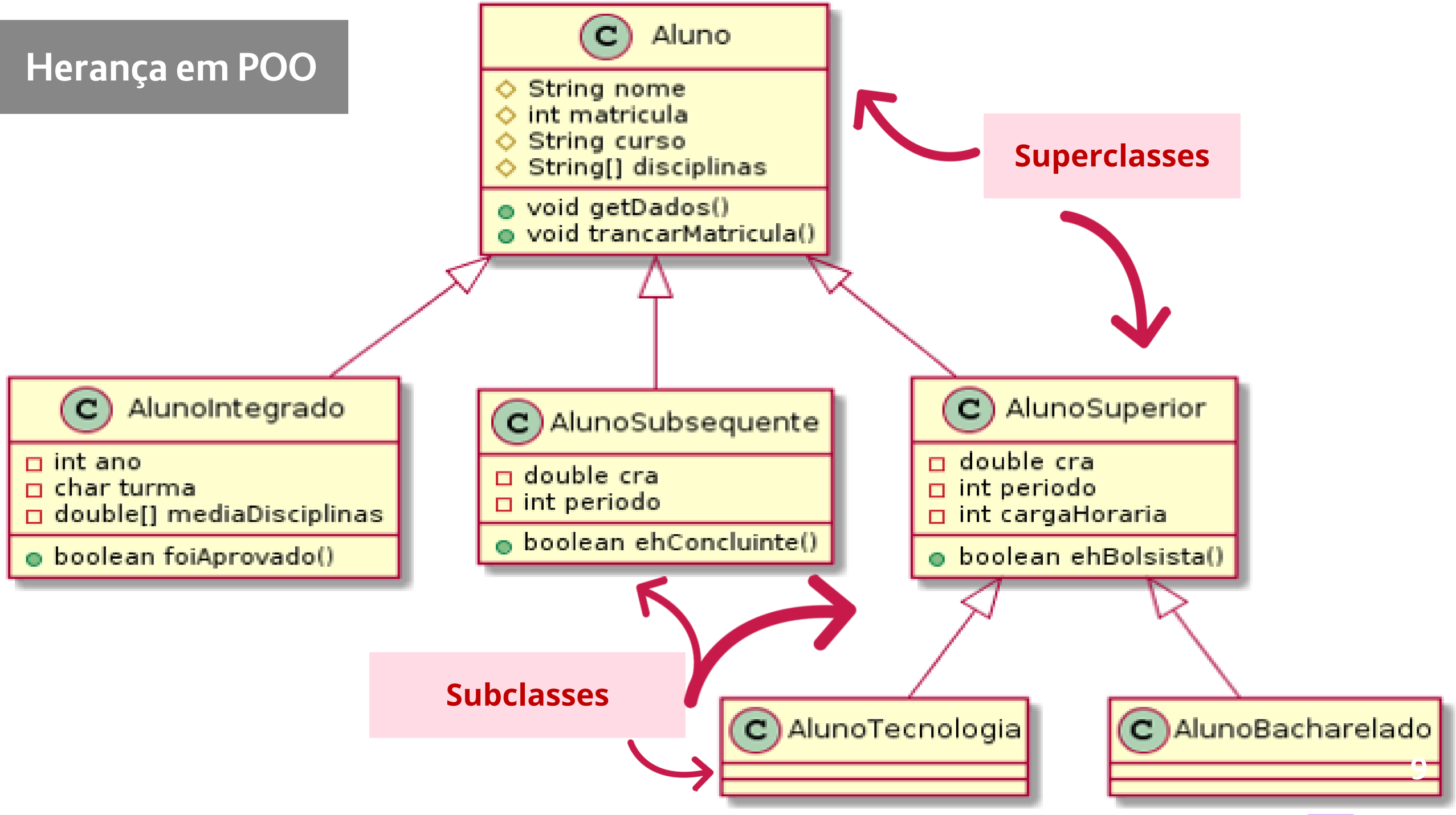
Na POO a herança estabelece um relacionamento entre classes.

Herança em programação orientada a objetos

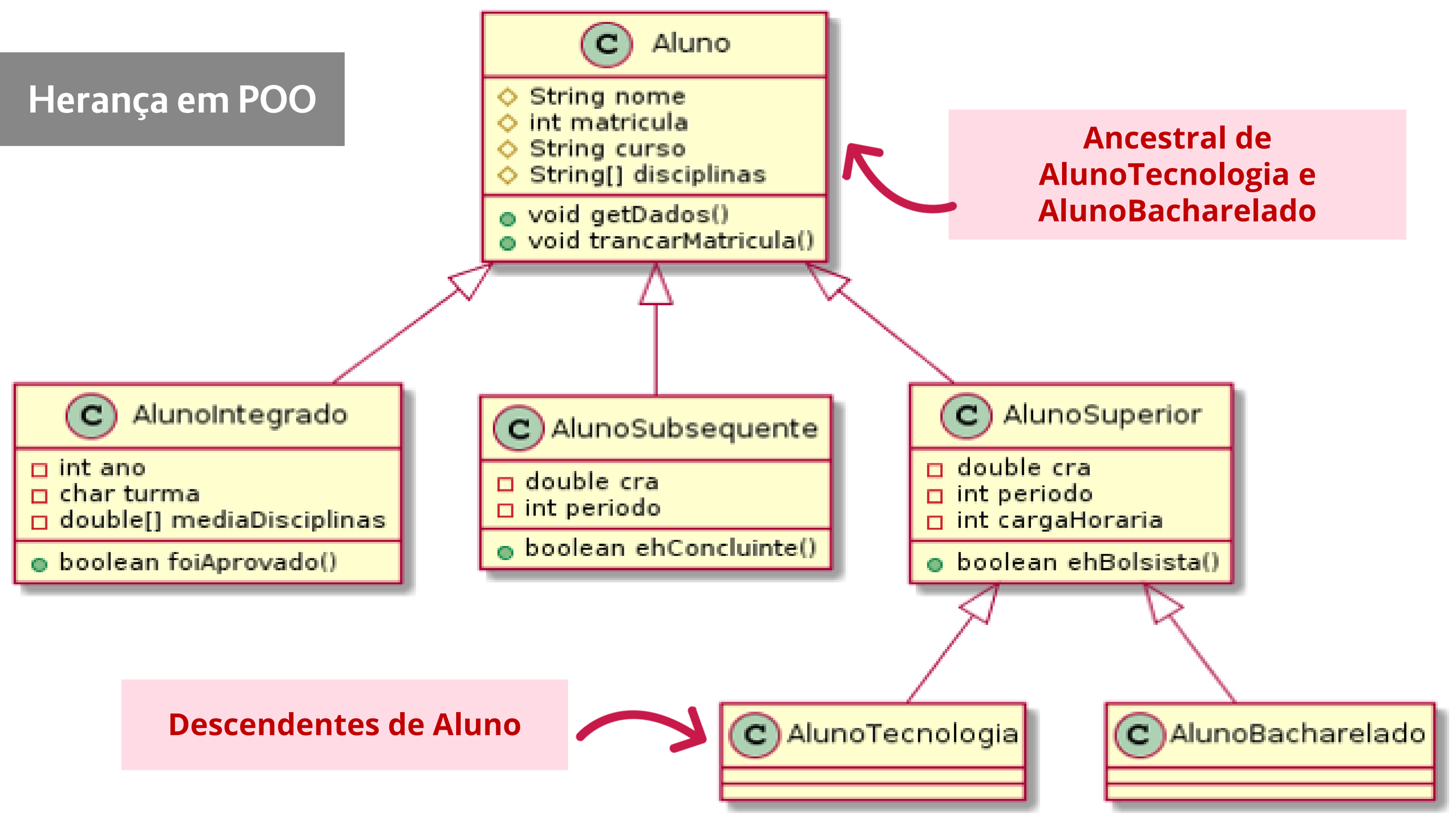
Permite reusar elementos de uma classe já criada como base para a criação de uma nova classe



Herança em POO

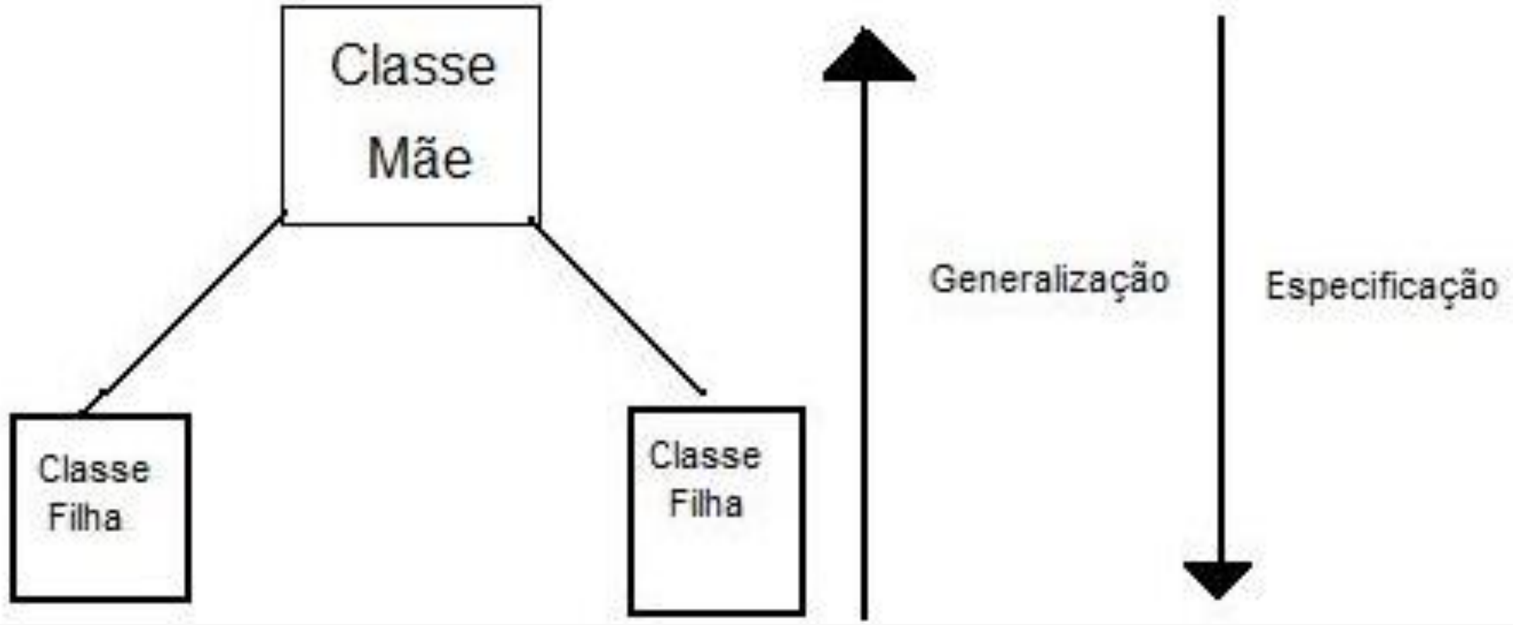


Herança em POO



Herança em programação orientada a objetos

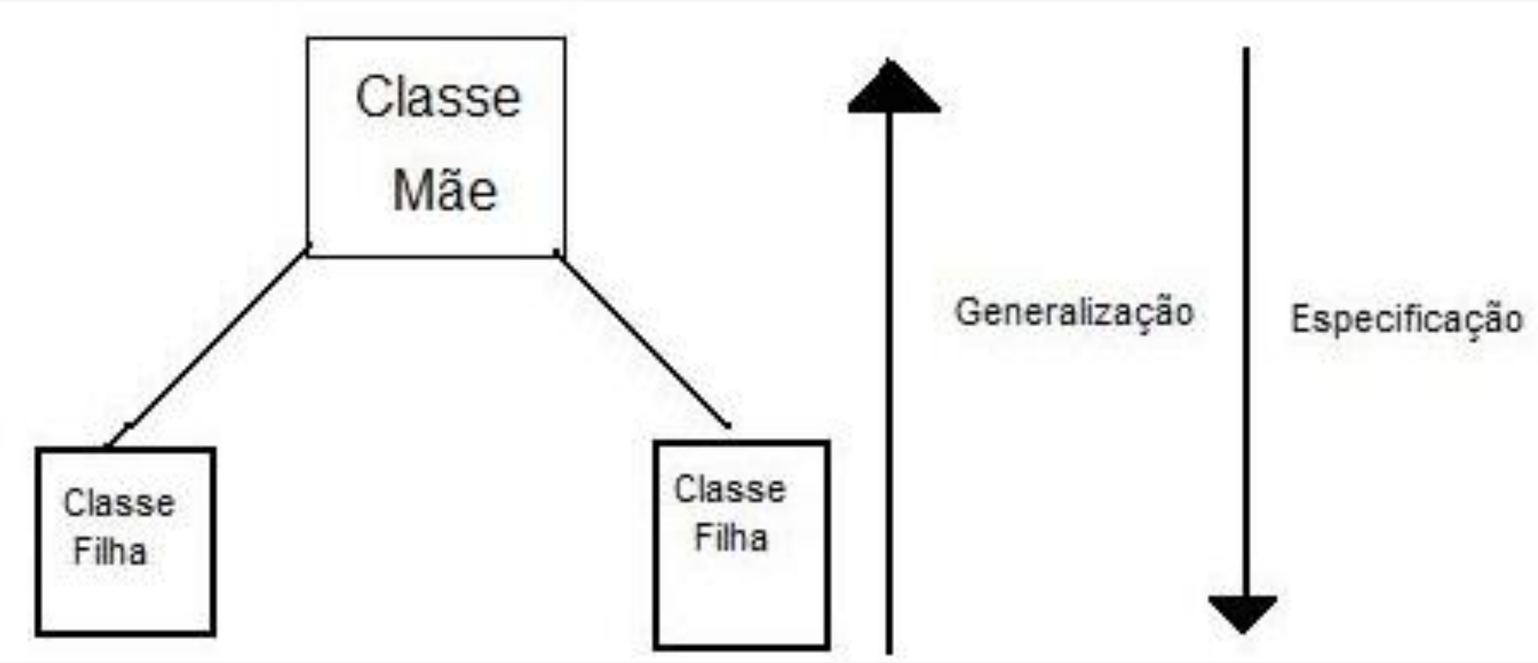
Uma classe existente e que será absorvida é chamada de classe mãe (ou superclasse)



- A nova classe, que absorve, é chamada de classe filha (ou subclasse)
- Considerada uma versão especializada da classe mãe.

Herança em programação orientada a objetos

Se uma classe herda de outra, então os objetos da subclasse possuem todas as propriedades de instância declaradas na superclasse



Pode-se enviar para um objeto da subclasse uma mensagem cuja implementação esteja na superclasse

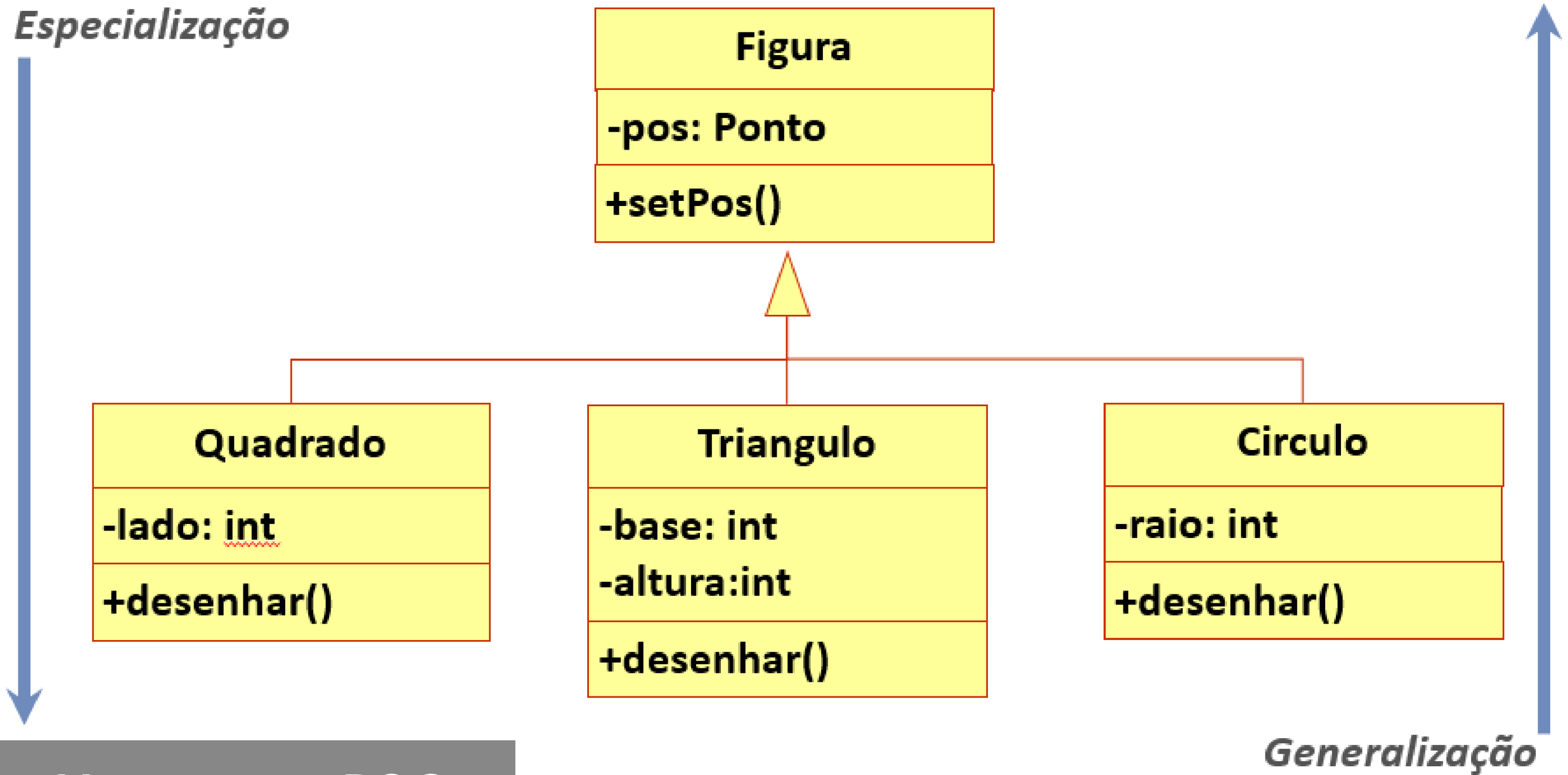
Herança em programação orientada a objetos

Herdando o estado:

Todos os objetos da subclasse terão uma cópia das variáveis de instância declaradas em todas as suas superclasses

Herdando o comportamento:

Todos os métodos de instância da superclasse também são métodos dos objetos das suas subclasses

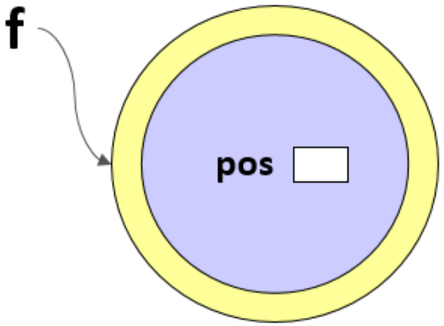


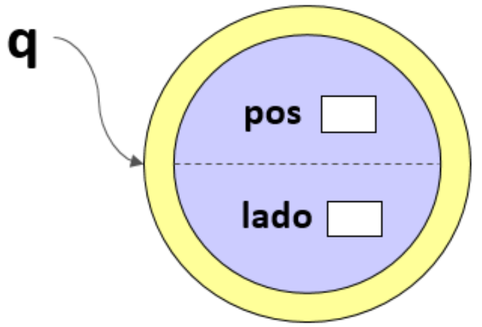
Herança em POO

Herança em programação orientada a objetos

Efeito da herança sobre os objetos da subclasse

```
Figura f = Figura();
Quadrado q = Quadrado();
```






```
f.setPos(Ponto(10,20));
q.setPos(Ponto(30,40));
q.desenhar();
```

Operações perfeitamente válidas

Herança em programação orientada a objetos

Sintaxe

- [acesso]: Pode ser public, private ou protected;
- Define o grau de visibilidade dos membros da classe derivada (subclasse).

```
class Subclasse: [acesso] Superclasse{
    // variaveis e métodos da Subclasse
};
```

Normalmente utiliza-se public (herança pública);

Herança em programação orientada a objetos

```
class Funcionario: public Pessoa{
    // variaveis e métodos de Funcionario
};
```

Exemplo: Funcionário é uma Pessoa

Exemplo: Chefe é um Funcionário.

```
class Chefe: public Funcionario{
    // variaveis e métodos de Chefe
};
```

Herança em programação orientada a objetos

Pessoa.h

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  class Pessoa {
6
7      private:
8          int identificacao;
9
10     protected:
11         string nome;
12
13     public:
14         Pessoa(int i, string n);
15         void exibirDados();
16         // setId() e getId() ...
17 };
```

Pessoa.cpp

```
1  #include "pessoa.h"
2
3  Pessoa::Pessoa(int i, string n){
4      identificacao = i;
5      nome = n;
6  }
7
8  void Pessoa::exibirDados(){
9      cout << "Identificação: " << identificacao
10         << "Nome: " << nome << endl;
11 }
12
13
14
15
16
17
```

Herança em programação orientada a objetos

Funcionario.h

```h
#include <iostream>
#include <string>
using namespace std;

class Funcionario : public Pessoa {

    protected:
        float salario;

    public:
        Funcionario(int id, string nome, float sal);
        void exibirDados();
};
```

Funcionario.cpp

```cpp
#include "funcionario.h"

Funcionario::Funcionario (int id, string n, float sal) :
Pessoa(id, n) {
    salario = sal;
}

void Funcionario::exibirDados(){
  cout << "Identificação: " << getId()
       << "Nome: " << nome << endl;
  cout << "Salario: R$" << salario << endl;
}
```

Herança em programação orientada a objetos

Chefe.h

```h
#include <iostream>
using namespace std;

class Chefe : public Funcionario {

    private:
        string departamento;

    public:
        Chefe(int id, string nome, float sal,string depto);
        void exibirDados();
};
```

Chefe.cpp

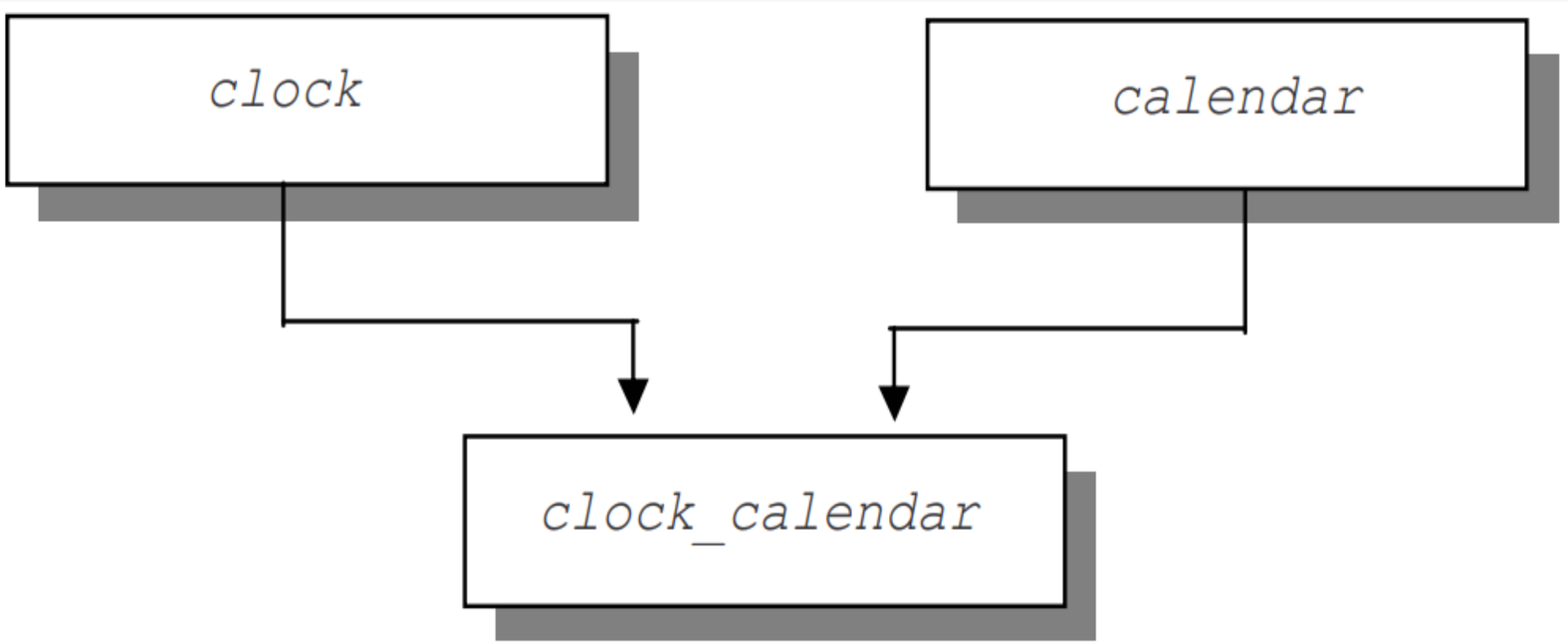
```cpp
#include "chefe.h"

Chefe::Chefe(int id, string nome, float sal, string dep) :
    Funcionario(nome, id, nasc, adm, sal) {
        departamento = dep;
    }

void Chefe::exibirDados(){
    cout << "Identificação: " << getId()
       << "Nome: " << nome << endl;
    cout << "Salario: R$" << salario << endl;
    cout << "Departamento: " << departamento << endl;
}
```

Herança múltipla

A herança múltipla entre classes ocorre sempre que uma subclasse possui duas ou mais superclasses imediatas, ou seja, é "filha" de mais de uma classe



Através da herança múltipla é possível combinar as características de várias superclasses existentes como um ponto de partida para a definição de uma nova classe

Herança múltipla

```cpp
#include <iostream>
using namespace std;

class Clock {

    protected:
        int hr;
        int min;
        int sec;
        int is_pm;
    public:
        Clock(int h, int m, int s, int pm);
        void set_clock (int h, int m, int s, int pm);
        void get_clock ();
        void advance();
};
```

Classe Clock

- As variáveis "hr", "min" e "sec" guardam, respectivamente, as horas, os minutos e os segundos
- O construtor "clock()" cria um novo relógio e determina a hora de acordo com os parâmetros recebidos
- "advance()" avança o relógio em um segundo

```cpp
#include <iostream>
using namespace std;

class Calendar {
    protected:
        int mo;
        int day;
        int yr;
    public:
        Calendar(int m, int d, int y);
        void set_calendar (int m, int d, int y);
        void get_calendar ();
        void advance();
};
```

Classe Calendar

- As variáveis "mo", "day" e "yr" guardam a data, isto é, mês, dia e ano
- O construtor "calendar()" cria um calendário e determina a data de acordo com os parâmetros recebidos
- "advance()" avança a data em um dia

Herança múltipla

Agora pode-se definir a classe “Clock_Calendar":

```
1  #include <iostream>
2  using namespace std;
3
4  class Clock_Calendar : public clock, public calendar {
5      public:
6          Clock_Calendar(int mt, int d, int y, int h, int mn, int s, int pm);
7          void advance();
8  };
9
10
11
12
```

- Após o nome da classe e os dois pontos, aparece uma lista de todas as superclasses a partir das quais a classe é derivada.
- Cada superclasse pode ser especificada individualmente como public, private ou protected. Como é desejado que "clock_calendar“ ofereça as funções herdadas de "clock" e "calendar", ambas são declaradas como públicas

Herança múltipla

Arquivo Clock_Calendar.cpp

```
1  #include "Clock_Calendar.h"
2
3
4  Clock_Calendar :: Clock_Calendar ( int mt, int d, int y, int h, int mn, int s, int pm ) :
5      Calendar(mt, d, y), Clock(h, mn, s, pm){
6
7      }
8
```

- Para inicializar o objeto "clock_calendar", o construtor "clock_calendar()" precisa invocar os construtores de "clock" e "calendar".
- Como na herança simples, isto é feito colocando-se chamadas à "clock()" e "calendar()" na lista de inicialização de "clock_calendar()".

Alguma dúvida?

Não guardem dúvidas, perguntem

• • •

Referências

1. DA COSTA, Anderson Fabiano F. **Fundamentos de C++**. Instituto Federal da Paraíba. 2022.

2. Materiais de aula dos professores Guillermo Camara-Chavez, Tiago Maritan, Fred Guedes Pereira e Danielle Chaves.

3. DEITEL, **C++ Como Programar**, 5ª edição, Editora Prentice Hall, 2006

4. MANSSOUR, Isabel Harb. **Herança Múltipla**. Pontifícia Universidade Católica do Rio Grande do Sul. Disponível em: <https://www.inf.pucrs.br/~manssour/LinguagemC++/HerancaMultipla.pdf> . Acesso em: 17 Maio 2022.

5.