"Brace yourself for what you are about to see!" (Patterson, Henessy; Arquitetura de Computadores, 2014).

Herança Múltipla

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR

ProfessorEngenheiro

Considere a classe `Engenheiro` disponibilizada.

Considere que existem professores na Universidade, que podem atuar tanto como professores, quanto como engenheiros.

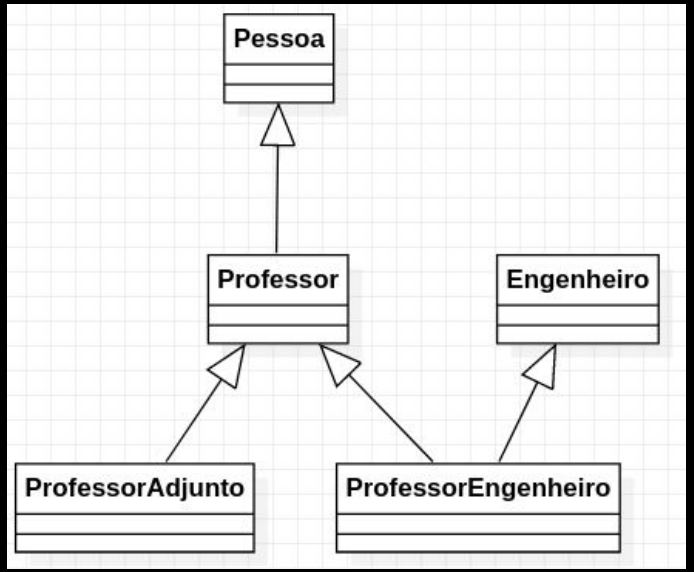
> Exemplo: um professor da Engenharia Civil também tem CREA.

Classe `ProfessorEngenheiro.`

Um `ProfessorEngenheiro` possui os comportamentos da classe Professor, e da classe Engenheiro simultaneamente.

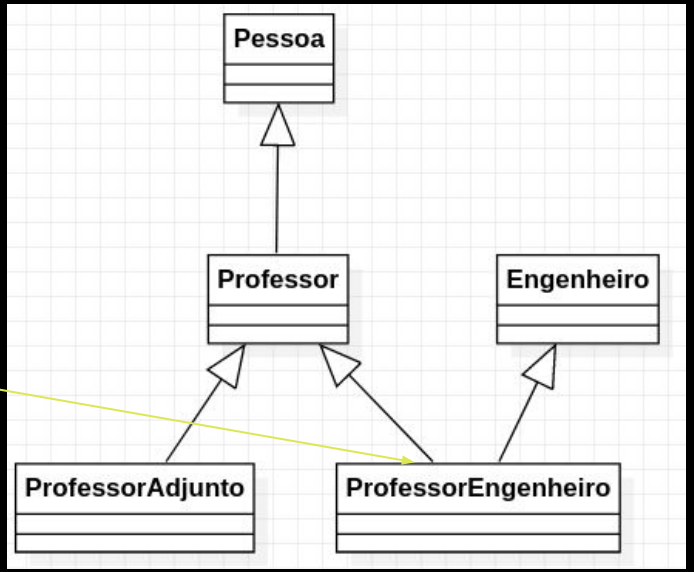
> Herda de Professor e de Engenheiro.

ProfessorEngenheiro



ProfessorEngenheiro

Herda de Professor e de Engenheiro.
Herança múltipla.



ProfessorEngenheiro

Separe as múltiplas classes base por vírgula.

```
#ifndef PROFESSOR_ENGENHEIRO_HPP
#define PROFESSOR_ENGENHEIRO_HPP

#include "string"

#include "Professor.hpp"
#include "Engenheiro.hpp"

class ProfessorEngenheiro : public Professor, public Engenheiro{
   public:
       ProfessorEngenheiro(const std::string& nome, const unsigned long cpf,
           const unsigned int valorHora, const unsigned short cargaHoraria,
           const unsigned int numeroCrea);

       virtual ~ProfessorEngenheiro();
};
#endif
```

ProfessorEngenheiro

Chame os construtores das classes base usando o member-initializer-list.
Os construtores das classes base são chamados **na ordem de declaração da herança**, e não na ordem em que aparecem na lista de inicialização de membros.

```
#include "ProfessorEngenheiro.hpp"

ProfessorEngenheiro::ProfessorEngenheiro(const std::string& nome,
           const unsigned long cpf, const unsigned int valorHora,
           const unsigned short cargaHoraria, const unsigned int numeroCrea)
                 : Professor(nome, cpf, valorHora, cargaHoraria), Engenheiro(numeroCrea){
}

ProfessorEngenheiro::~ProfessorEngenheiro(){}
```

Pergunta

Podemos fazer isso?

```cpp
#include <iostream>

#include "ProfessorEngenheiro.hpp"
#include "Professor.hpp"
#include "Engenheiro.hpp"


int main(){
   ProfessorEngenheiro pe{"Maria", 11111111111,85, 40, 1234};

   Engenheiro *e{&pe};
   Professor *p{&pe};

   std::cout << p->getNome() << " " << p->getSalario() << "\n";
   std::cout << e->getNumeroCrea() << "\n";

   return 0;
}
```

Pergunta

As relações de “tipo de” são válidas para herança múltipla.

`ProfessorEngenheiro` é um tipo de:

- `Professor`;
- `Engenheiro`;
- `Pessoa`.

```cpp
#include <iostream>

#include "ProfessorEngenheiro.hpp"
#include "Professor.hpp"
#include "Engenheiro.hpp"


int main(){
   ProfessorEngenheiro pe{"Maria", 11111111111,85, 40, 1234};

   Engenheiro *e{&pe};
   Professor *p{&pe};

   std::cout << p->getNome() << " " << p->getSalario() << "\n";
   std::cout << e->getNumeroCrea() << "\n";

   return 0;
}
```

Herança Múltipla

Herança múltipla parece simples.

Mas é um **conceito complexo que deve ser usado somente quando necessário.**

Gera problemas sutis de difícil detecção e correção se não prestarmos atenção.

Por conta disso, linguagens que tentam simplificar a vida do programador comumente não aceitam herança múltipla.

Exemplos: Java e C#.

Com herança simples, os compiladores, interpretadores e coletores de lixo se tornam mais simples.

E o programador é mais feliz ;)

Ao menos até encontrar um problema que ele não consiga resolver com herança simples.

Engenheiro

Adicione o seguinte:

```
class Engenheiro{
   public:
       Engenheiro();
       Engenheiro(const unsigned int numeroCrea);
       virtual ~Engenheiro();

       unsigned int getNumeroCrea() const;
       void setNumeroCrea(const unsigned int numeroCrea);

       virtual unsigned int getSalario() const;

   private:
       constexpr static unsigned int salarioPadrao{9405};

       unsigned int numeroCrea;
};
```

```
#include "Engenheiro.hpp"

//...

unsigned int Engenheiro::getSalario() const{
   return Engenheiro::salarioPadrao;
}
```

Engenheiro

Onde está o problema?

```cpp
class Engenheiro{
   public:
       Engenheiro();
       Engenheiro(const unsigned int numeroCrea);
       virtual ~Engenheiro();

       unsigned int getNumeroCrea() const;
       void setNumeroCrea(const unsigned int numeroCrea);

       virtual unsigned int getSalario() const;

   private:
       constexpr static unsigned int salarioPadrao{9405};

       unsigned int numeroCrea;
};
```

```cpp
#include "Engenheiro.hpp"

//...

unsigned int Engenheiro::getSalario() const{
   return Engenheiro::salarioPadrao;
}
```

Faça você mesmo

Teste o seguinte no main:

```cpp
#include <iostream>

#include "ProfessorEngenheiro.hpp"
#include "Professor.hpp"
#include "Engenheiro.hpp"


int main(){
   ProfessorEngenheiro pe{"Maria", 11111111111,85, 40, 1234};

   std::cout << pe.getNome() << " " << pe.getSalario() << "\n";

   return 0;
}
```

Faça você mesmo

`GetSalario` é definida na classe `Engenheiro`, e na classe `Professor`.

O compilador não sabe qual `getSalario` chamar!

```
Saída do g++:

error: request for member
‘getSalario’ is ambiguous
  std::cout << pe.getNome() << " " <<
pe.getSalario() << std::endl;

...

note: candidates are: virtual unsigned int
Engenheiro::getSalario() const
      virtual unsigned int getSalario()
const;
...
virtual unsigned int Professor::getSalario()
const
   virtual unsigned int getSalario() const;
...
```

```
#include <iostream>

#include "ProfessorEngenheiro.hpp"
#include "Professor.hpp"
#include "Engenheiro.hpp"


int main(){
   ProfessorEngenheiro pe{"Maria", 11111111111,85, 40, 1234};

   std::cout << pe.getNome() << " " << pe.getSalario() << "\n";

   return 0;
}
```

Contornando

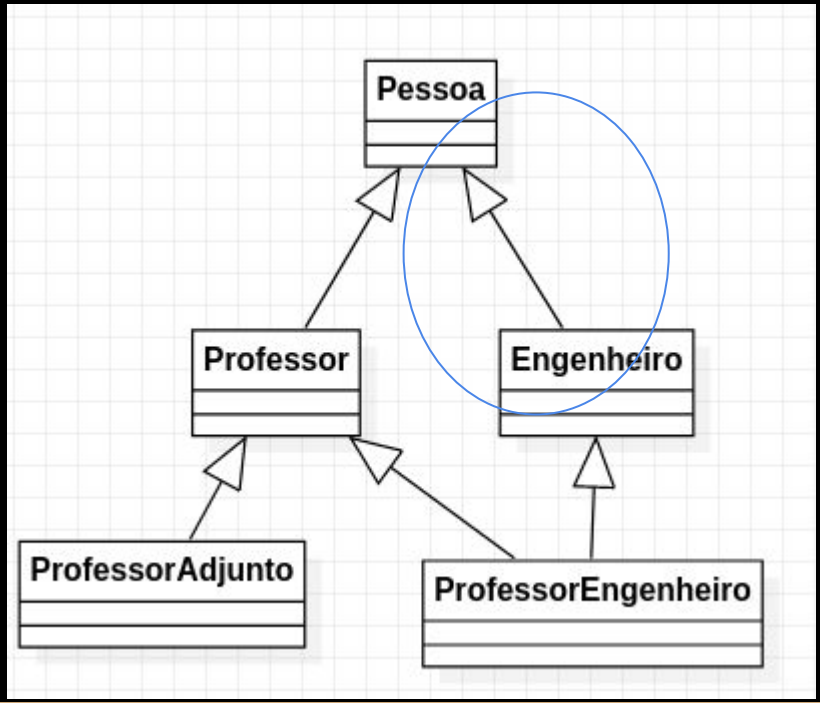
Podemos contornar o problema com o operador de resolução de escopo ::

```cpp
#include <iostream>

#include "ProfessorEngenheiro.hpp"
#include "Professor.hpp"
#include "Engenheiro.hpp"


int main(){
   ProfessorEngenheiro pe{"Maria", 11111111111,85, 40, 1234};

   std::cout << pe.getNome() << " " << pe.Professor::getSalario() << "\n";
   std::cout << pe.getNome() << " " << pe.Engenheiro::getSalario() << "\n";

   return 0;
}
```

Complicando

Todo `Engenheiro` também é uma `Pessoa`.

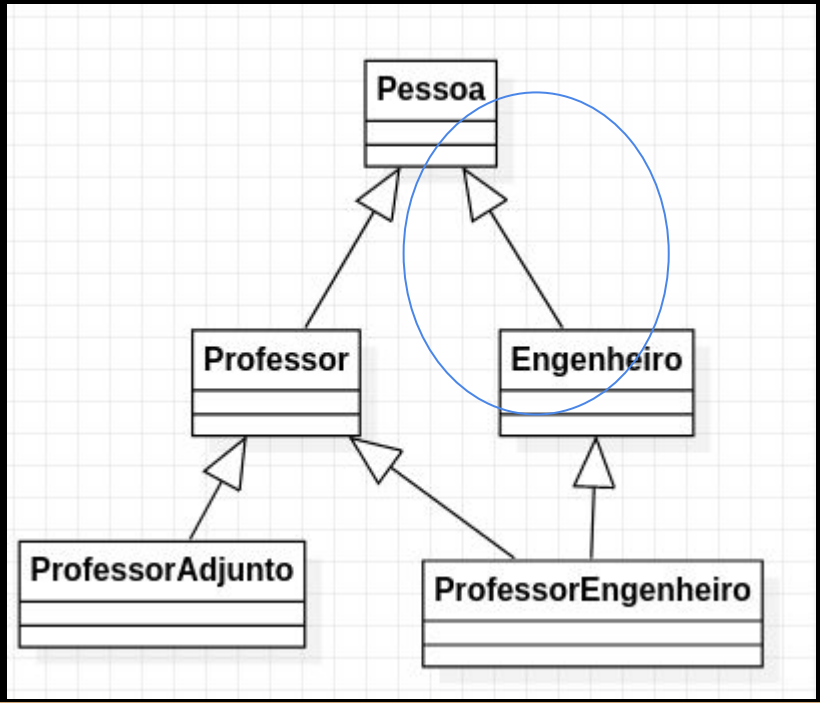
- Quais potenciais novos problemas?



Complicando

Todo `Engenheiro` também é uma `Pessoa`.

- Quais potenciais novos problemas?
  - Duplicatas das funções e dados membro de Pessoa em ProfessorEngenheiro;
  - Problemas de ambiguidade ao chamar funções de Pessoa a partir de ProfessorEngenheiro.



Faça você mesmo

```cpp
#include <iostream>

#include "ProfessorEngenheiro.hpp"
#include "Professor.hpp"
#include "Engenheiro.hpp"


int main(){
   ProfessorEngenheiro pe{"Maria", 11111111111,85, 40, 1234};

   std::cout << pe.getNome() << std::endl;

   return 0;
}
```

Faça você mesmo

```
g++ -c main.cpp -Wall
main.cpp: In function 'int main()':
main.cpp:11:18: error: request for member 'getNome' is ambiguous
   11 |  std::cout << pe.getNome() << std::endl;
...
```

```cpp
#include <iostream>

#include "ProfessorEngenheiro.hpp"
#include "Professor.hpp"
#include "Engenheiro.hpp"


int main(){
   ProfessorEngenheiro pe{"Maria", 11111111111,85, 40, 1234};

   std::cout << pe.getNome() << std::endl;

   return 0;
}
```

Chamada ambígua. Erro de compilação.

Faça você mesmo

Usar um resolvedor de escopo nesse caso faz o programa **compilar**.

Mas ele continua com sérios problemas.

Temos duplicatas na memória.

`ProfessorEngenheiro` é Pessoa duas vezes? Possui dois nomes? As funções são duplicadas?

Entendendo o tamanho do problema

Coloque um `cout` em cada construtor de:

- `Pessoa;`
- `Professor;`
- `Engenheiro;`
- `ProfessorEngenheiro.`

O que acontece de estranho?

Entendendo o tamanho do problema

Coloque um `cout` em cada construtor de:

- `Pessoa;`
- `Professor;`
- `Engenheiro;`
- `ProfessorEngenheiro.`

O construtor de pessoa é chamado 2x.

Entendendo o tamanho do problema

As coisas realmente estão duplicadas na memória.

Prova:

```cpp
#include <iostream>

#include "ProfessorEngenheiro.hpp"
#include "Professor.hpp"
#include "Engenheiro.hpp"

int main(){
   ProfessorEngenheiro pe{"Maria", 11111111111,85, 40, 1234};

   pe.Professor::setNome("Maria Silva");
   pe.Engenheiro::setNome("Marcia Silveira");

   std::cout << pe.Professor::getNome() << std::endl;
   std::cout << pe.Engenheiro::getNome() << std::endl;

   return 0;
}
```

Maria Silva
Marcia Silveira

Classes base virtuais

Usando um raciocínio similar ao usado nas funções virtuais, vamos definir **classes base virtuais**.

O compilador vai garantir que existe apenas uma cópia das classes base em cada classe que herda.

Existe um pequeno custo computacional extra atrelado.

- Exige ponteiros extras, de maneira similar às funções virtuais.

Classes base virtuais

Não serão necessárias alterações na classe `ProfessorEngenheiro`.

Apenas em `Professor`, e em `Engenheiro`.

Indicador de **herança virtual**.

```
#ifndef PROFESSOR_HPP
#define PROFESSOR_HPP

#include "Pessoa.hpp"
class Professor : virtual public Pessoa{
   public:

        //...
   private:
       unsigned int valorHora;
       unsigned short cargaHoraria;
};
#endif
```

```
#ifndef ENGENHEIRO_HPP
#define ENGENHEIRO_HPP

#include "Pessoa.hpp"

class Engenheiro : virtual public Pessoa{
   public:

       //...
   private:
       constexpr static unsigned int salarioPadrao{9405};

       unsigned int numeroCrea;
};
#endif
```

Faça você mesmo

Agora o trecho funciona como o esperado.

```cpp
#include <iostream>

#include "ProfessorEngenheiro.hpp"
#include "Professor.hpp"
#include "Engenheiro.hpp"

int main(){
   ProfessorEngenheiro pe{"Maria", 11111111111,85, 40, 1234};

   pe.Professor::setNome("Maria Silva");
   pe.Engenheiro::setNome("Marcia Silveira");

   std::cout << pe.Professor::getNome() << std::endl;
   std::cout << pe.Engenheiro::getNome() << std::endl;

   return 0;
}
```

Marcia Silveira
Marcia Silveira

Faça você mesmo

Mas isso não funciona!

Por quê?

```
#include <iostream>

#include "ProfessorEngenheiro.hpp"
#include "Professor.hpp"
#include "Engenheiro.hpp"


int main(){
   ProfessorEngenheiro pe{"Maria", 11111111111,85, 40, 1234};

   std::cout << pe.Engenheiro::getNome() << std::endl;

   return 0;
}
```

Faça você mesmo

A classe `Engenheiro` chama o construtor padrão de `Pessoa`.

Quando criamos hierarquias com herança múltipla e classes base virtuais, é fundamental que toda a hierarquia chame os construtores corretos.

Recomendação: para herança múltipla tente usar construtores default.

É difícil gerenciar as chamadas de construtor em hierarquias complexas com classes base virtuais

```
#include <iostream>

#include "ProfessorEngenheiro.hpp"
#include "Professor.hpp"
#include "Engenheiro.hpp"


int main(){
   ProfessorEngenheiro pe{"Maria", 11111111111,85, 40, 1234};

   std::cout << pe.Engenheiro::getNome() << std::endl;

   return 0;
}
```

Chamando construtores não default

- Comente o construtor default de `Pessoa` (para facilitar);
- Chame o construtor correto de `Pessoa` no member-initializer-list de `Engenheiro` e `Professor`;
- Chame os construtores corretos de `Pessoa`, `Engenheiro` e `Professor` em `ProfessorEngenheiro`;
- Note que é obrigatório que você indique qual construtor de `Pessoa` deve ser usado, apesar de `ProfessorEngenheiro` não herdar diretamente de `Pessoa`.

Exemplo

```cpp
#include "ProfessorEngenheiro.hpp"

ProfessorEngenheiro::ProfessorEngenheiro(const std::string& nome,
          const unsigned long cpf, const unsigned int valorHora,
          const unsigned short cargaHoraria, const unsigned int numeroCrea)
                :Pessoa(nome, cpf),
                Professor(nome, cpf, valorHora, cargaHoraria),
                Engenheiro(nome, cpf, numeroCrea){
}

ProfessorEngenheiro::~ProfessorEngenheiro(){}
```

```cpp
class ProfessorEngenheiro : public Professor, public Engenheiro{
   public:
       ProfessorEngenheiro(const std::string& nome,
               const unsigned long cpf,
               const unsigned int valorHora,
               const unsigned short cargaHoraria,
               const unsigned int numeroCrea);

       virtual ~ProfessorEngenheiro();
};
#endif
```

Agora sim

Agora as coisas estão corretas na memória, e essas construções não geram problemas.

```cpp
#include <iostream>

#include "ProfessorEngenheiro.hpp"
#include "Professor.hpp"
#include "Engenheiro.hpp"


int main(){
   ProfessorEngenheiro pe{"Maria", 11111111111,85, 40, 1234};

   std::cout << pe.Engenheiro::getNome() << std::endl;

   return 0;
}
```

Faça você mesmo

Teste o trecho com e sem classes base virtuais. Note que sem a declaração virtual o tamanho do objeto na memória é maior, já que temos duplicatas na memória.

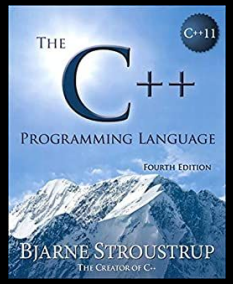
```
#include <iostream>

#include "Pessoa.hpp"
#include "ProfessorEngenheiro.hpp"

int main(){
   ProfessorEngenheiro pe{"Maria", 11111111111,85, 40, 1234};
   std::cout << "Tamanho na memória: " << sizeof pe << "\n";

   return 0;
}
```

Exercícios

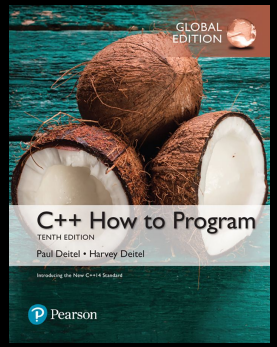
1. Na classe ProfessorEngenheiro sobrecarregue a função getSalario. O salário de um ProfessorEngenheiro é igual ao seu salário de professor, somado ao salário de Engenheiro.
    a. Note que agora você não precisa mais da resolução de escopo para fazer um getSalario()
2. Em linguagens como o Java e o C#, foi introduzido o conceito de interface para remediar a falta de herança múltipla.
    a. Estude o que é uma interface em Java/C#.
    b. O que é equivalente a uma Interface em C++? Isso é possível criar interfaces?
    c. Que tipos de problemas uma interface não resolve, e que podemos resolver apenas com herança múltipla?
3. A classe std::basic_iostream que provê funcionalidades para classes de entrada e saída do C++ precisa de uma herança virtual devido aos problemas discutidos na aula. Pesquise sobre a hierarquia de classes de std::basic_iostream.

Referências

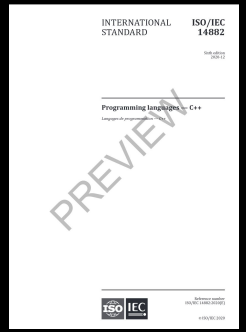
Bjarne Stroustrup. The C++ Programming Language. Addison-Wesley, 2013.



Deitel, H. M., Deitel, P. J. C++: como programar. 10a ed. Pearson Prentice Hall. 2017.



ISO/IEC 14882:2020 Programming languages - C++: www.iso.org/obp/ui/#iso:std:iso-iec:14882:ed-6:v1:en



Licença