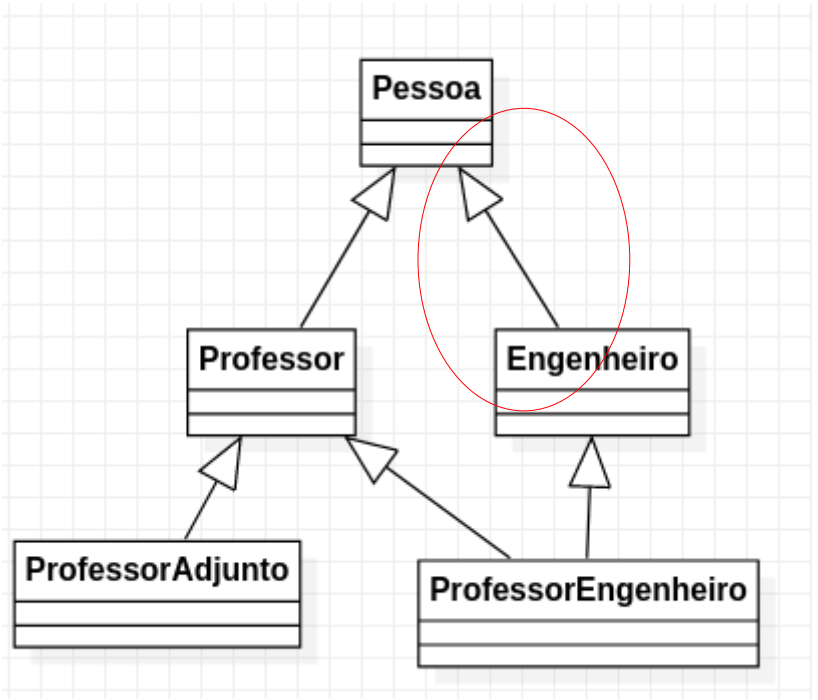
Herança Múltipla – Classes base virtuais

Paulo Ricardo Lisboa de Almeida

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

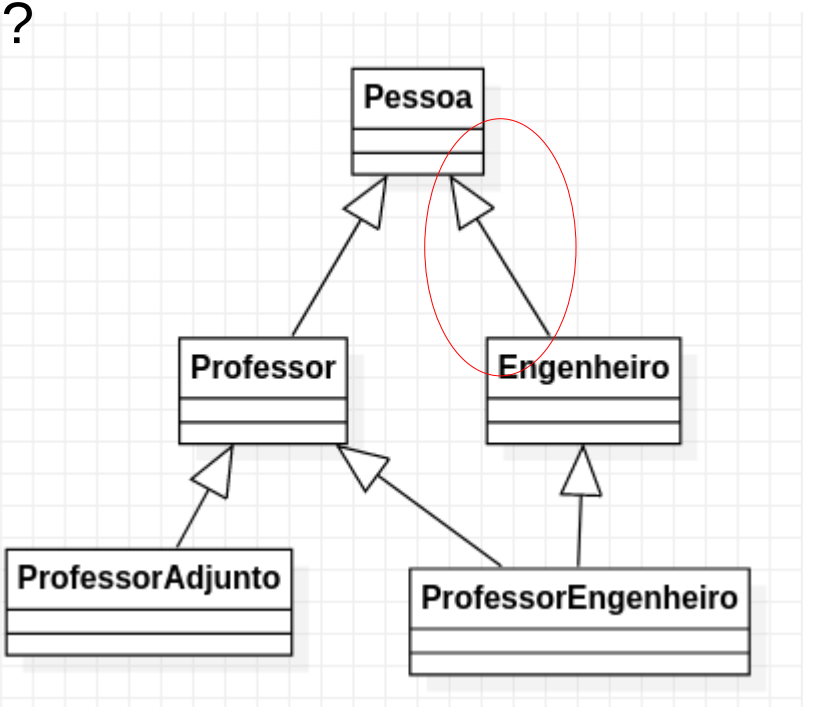
ProfessorEngenheiro

- Vamos adicionar mais uma herança
  - Todo engenheiro também é uma Pessoa



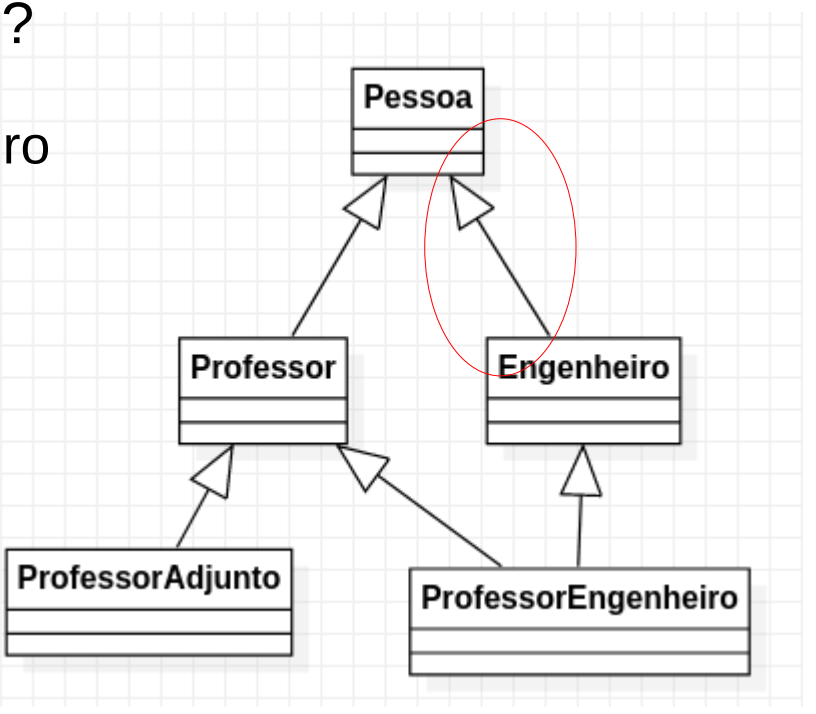
ProfessorEngenheiro

- Vamos adicionar mais uma herança
  - Todo engenheiro também é uma Pessoa
- Quais potenciais novos problemas criamos?



ProfessorEngenheiro

- Vamos adicionar mais uma herança
  - Todo engenheiro também é uma Pessoa
- Quais potenciais novos problemas criamos?
  - Teremos duplicatas das funções e dados membro de Pessoa em ProfessorEngenheiro
  - Teremos problemas de ambiguidade ao chamar funções de Pessoa a partir de ProfessorEngenheiro



Teste você mesmo

```
#include <iostream>

#include "ProfessorEngenheiro.hpp"

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

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

    return 0;
}
```

Teste você mesmo

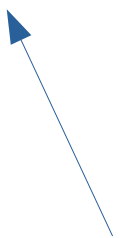
```
#include <iostream>

#include "ProfessorEngenheiro.hpp"

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

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

    return 0;
}
```



Chamada ambígua. Erro de compilação.

UDESC

Teste você mesmo

- Usar um resolvedor de escopo nesse caso faz o programa compilar
  - Mas ele continua com problemas
    - Temos duplicatas na memória
    - Associar um ProfessorEngenheiro a uma Pessoa gera problemas
      - Qual função membro usar? Quais dados? ProfessorEngenheiro é Pessoa duas vezes?

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

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

        Pessoa* e{&pe};
        //...
        return 0;
}
```

Compila

Erro de compilação.

Prova das duplicatas na memória

- Coloque um cout em cada construtor de:
  - Pessoa
  - Professor
  - Engenheiro
  - ProfessorEngenheiro
- Note que o construtor de Pessoa é chamado 2x

Classes base virtuais

- Usando um raciocínio similar ao usado nas funções virtuais, vamos definir classes base como 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 ao que estudamos para funções virtuais

Classes base virtuais

- Não serão necessárias alterações na classe ProfessorEngenheiro
  - Apenas em Professor, e em Engenheiro
    - Indicador de herança virtual

```
class Professor : virtual public Pessoa{
    //...
};
```

```
class Engenheiro : virtual public Pessoa{
    //...
};
```

Teste você mesmo

- Os problemas foram resolvidos e tudo funciona, mas qual o motivo do nome não ser impresso agora?

```
#include <iostream>

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

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

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


	return 0;
}
```

Engenheiro

- Note que 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: tente usar sempre construtores *default*
  - É difícil gerenciar as chamadas de construtor em hierarquias complexas com classes base virtuais

Chamando construtores não default

- Comente o construtor *default* de Pessoa
- 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 em Engenheiro qual construtor de Pessoa deve ser usado, mesmo que ProfessorEngenheiro não herde diretamente de Pessoa

Teste você mesmo

```
#include <iostream>

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

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

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

	Pessoa* p{&pe};
	std::cout << p->getNome() << std::endl;



	return 0;
}
```

Deixamos de ter 2 cópias de Pessoa na memória, e essas construções agora são válidas

UDESC

Teste você mesmo

```
#include <iostream>

#include "ProfessorEngenheiro.hpp"

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

	return 0;
}
```

Execute o trecho usando classes base virtuais, e sem declarar as classes como virtuais. Note que sem a declaração virtual o tamanho do objeto na memória é maior, já que temos duplicatas na memória.

Exercício

1. A classe std::basic_iostream que provê funcionalidades para muitas 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

- DEITEL, P.; DEITEL, H. **C++ how to Program**. [S.l.]: Pearson, 2017. ISBN 9780134448237
- STROUSTRUP, B. **The C++ Programming Language**. Pearson Education, 2013. ISBN 9780133522853.