

Herança

Paulo Ricardo Lisboa de Almeida

Herança

- Com a **herança** podemos criar classes que herdam (absorvem) as capacidades de classes já existentes
 - Salvamos tempo
 - Não precisamos recriar tudo do zero
 - Criamos software de qualidade
 - Utilizamos classes já existentes e testadas como base

Herança

- Uma **classe B** herda os membros da **classe A**
 - A *classe A* é chamada de **classe base** ou **classe pai**
 - Em Java e C# a classe A é chamada de super classe
 - A *classe B* é chamada de **classe derivada** ou classe **filha**
 - A classe B deriva de A
 - Em Java e C# a classe B é chamada de subclasse
- Uma classe derivada representa uma **especialização** da classe base
- Formamos **hierarquias** de classes

Exemplo

- Desejamos fazer a distinção entre alunos e professores
 - Alunos possuem
 - Número de matrícula
 - O que mais?
 - Professores possuem
 - Carga Horária Semanal
 - Salário
- Mas tanto professores quanto alunos são **tipos de pessoa**
 - Mas copiar e colar a classe Pessoa para criar duas especializações seria uma péssima ideia

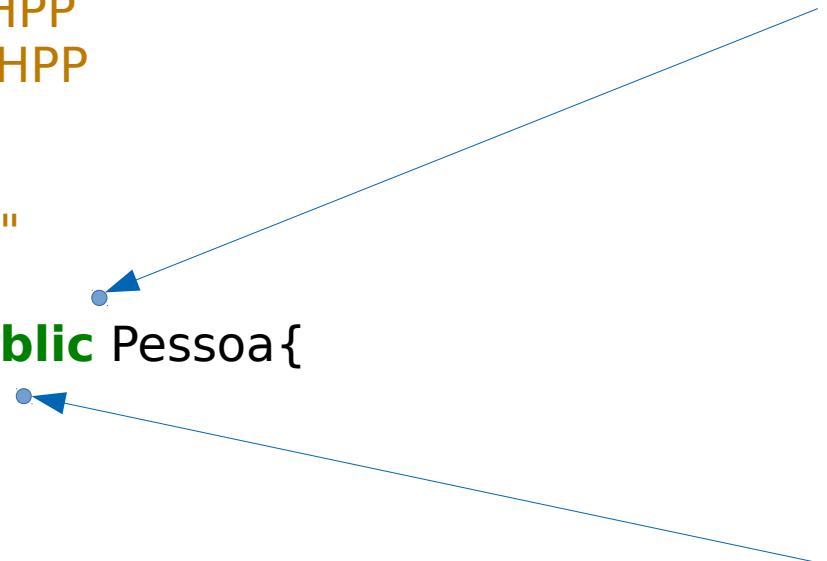
Professor.hpp

Professor **herda** de Pessoa (**deriva** de Pessoa)

```
#ifndef PROFESSOR_HPP  
#define PROFESSOR_HPP
```

```
#include <string>  
#include "Pessoa.hpp"
```

```
class Professor : public Pessoa{  
    //...  
};  
#endif
```



The diagram consists of two blue arrows. The first arrow originates from the text 'Professor herda de Pessoa (deriva de Pessoa)' and points to the 'Pessoa' part of the 'public Pessoa{' line in the code. The second arrow originates from the text 'A herança é pública (tipo mais comum)...' and points to the 'public' access specifier in the same line.

A herança é pública (tipo mais comum). Temos ainda heranças privadas e protegidas que veremos no futuro.

Professor.hpp

```
//...
class Professor : public Pessoa{
public:
    Professor(const std::string& nome, const unsigned long cpf,
              const unsigned int salario, const unsigned short cargaHoraria);
    ~Professor();

    void setSalario(const unsigned int salario);
    unsigned int getSalario() const;

    void setCargaHoraria(const unsigned short cargaHoraria);
    unsigned short getCargaHoraria() const;

private:
    unsigned int salario;
    unsigned short cargaHoraria;
};
```

Professor.hpp

```
//...
class Professor : public Pessoa{
public:
    Professor(const std::string& nome, const unsigned long cpf,
              const unsigned int salario, const unsigned short cargaHoraria);
    ~Professor();

    void setSalario(const unsigned int salario);
    unsigned int getSalario() const;

    void setCargaHoraria(const unsigned short cargaHoraria);
    unsigned short getCargaHoraria() const;

private:
    unsigned int salario;
    unsigned short cargaHoraria;
};
```

Dica do dia: nunca represente valores monetários com floats ou doubles devido aos erros numéricos. Se precisar dos centavos, (ex.: um salário de R\$ 3.000,53) represente o salário em centavos, ou use classes específicas, como as disponíveis na Boost: www.boost.org/doc/libs/1_57_0/libs/multiprecision/doc/html/index.html

Professor.cpp

```
#include "Professor.hpp"
```

```
Professor::Professor(const std::string& nome, const unsigned long cpf,  
                    const unsigned int salario, const unsigned short cargaHoraria)  
    :Pessoa(nome, cpf), salario(salario), cargaHoraria(cargaHoraria) {  
}
```

```
Professor::~~Professor(){  
}
```

```
void Professor::setSalario(const unsigned int salario){  
    this->salario = salario;  
}
```

```
unsigned int Professor::getSalario() const{  
    return this->salario;  
}
```

```
void Professor::setCargaHoraria(const unsigned short cargaHoraria){  
    this->cargaHoraria = cargaHoraria;  
}
```

```
unsigned short Professor::getCargaHoraria() const{  
    return this->cargaHoraria;  
}
```


Professor.cpp

Chamando o construtor da classe base com os parâmetros necessários

Inicializando salário e carga no member initializer list

```
#include "Professor.hpp"
```

```
Professor::Professor(const std::string& nome, const unsigned long cpf,
                    const unsigned int salario, const unsigned short cargaHoraria)
    :Pessoa(nome, cpf), salario(salario), cargaHoraria(cargaHoraria) {
```

```
Professor::~~Professor(){
}
```

```
void Professor::setSalario(const unsigned int salario){
    this->salario = salario;
}
```

```
unsigned int Professor::getSalario() const{
    return this->salario;
}
```

```
void Professor::setCargaHoraria(const unsigned short cargaHoraria){
    this->cargaHoraria = cargaHoraria;
}
```

```
unsigned short Professor::getCargaHoraria() const{
    return this->cargaHoraria;
}
```

Tipo de

- Professor é um **tipo de** Pessoa
 - Possui tudo o que uma pessoa possui (nome, cpf, ...)
- Relações tipo de não são reflexivas!
 - Um professor também é uma pessoa (é um tipo de pessoa, ou uma especialização de pessoa)
 - Mas uma pessoa não é um tipo de professor (nem toda pessoa é professor)

No main

- No main, podemos usar a classe Professor como qualquer outra

```
#include <iostream>
```

```
#include "Disciplina.hpp"
```

```
#include "Professor.hpp"
```

```
int main(){
```

```
    Professor p{"Joao", 1111111111, 30000, 40};
```

```
    std::cout << p.getNome() << std::endl;
```

```
    std::cout << p.getSalario() << std::endl;
```

```
    return 0;
```

```
}
```

Construtores

- O construtor da classe base é chamado antes da classe derivada
 - **Independentemente** se você chamou o construtor da classe base explicitamente ou não
- **Exercício**
 - Coloque um cout no construtor de pessoa, e outro no de professor
 - Compile e execute o programa

Membros

- Os membros privados da classe base **não são acessíveis** nas classes derivadas (filhas)
 - Membros privados são acessíveis **apenas na própria classe**, e isso **não muda** com o conceito de herança
- Membros **públicos** são acessados normalmente
- Exemplo (Professor.cpp)

```
Professor::Professor(const std::string& nome, const unsigned long cpf,  
                    const unsigned int salario, const unsigned short cargaHoraria)  
    :Pessoa(nome, cpf), salario(salario), cargaHoraria(cargaHoraria) {  
//    std::cout << "Construindo professor " << nome << std::endl;  
    this->nome = "Teste";  
}
```

Erro de compilação: nome é privado na classe Pessoa

Protected

- Temos **3** modificadores de acesso possíveis
 - public, private e **protected**
- **Protected**
 - O mesmo que private, mas o acesso é **estendido as classes derivadas** (filhas)
 - Classes amigas também acessam membros protected
 - Da mesma forma que acessam membros private
 - Observação
 - No Java, o *protected* tem uma interpretação diferente
 - Volte na aula “Classes e funções amigas” e reveja a discussão sobre a quebra de encapsulamento do protected no Java

Exemplo

Professor.cpp

```
Professor::Professor(const std::string& nome, const unsigned long cpf,
                    const unsigned int salario, const unsigned short cargaHoraria)
    :Pessoa(nome, cpf), salario(salario), cargaHoraria(cargaHoraria) {
//    std::cout << "Construindo professor " << nome << std::endl;
    this->nome = "Teste";
}
```

Pessoa.hpp

```
class Pessoa{
public:
    //...
protected:
    std::string nome;
    unsigned long cpf;
    unsigned char idade;
};
```

Agora isso é **válido** na classe **Professor**

Pergunta

Considere a classe Disciplina,
onde professor é
representado pela classe
Pessoa

```
//...
```

```
class Disciplina{  
    public:  
        Disciplina(std::string nome);  
        ~Disciplina();  
  
        const Pessoa* getProfessor() const;  
        void setProfessor(Pessoa* professor);  
  
        //...  
    private:  
        //...  
  
        Pessoa* professor;  
        std::list<Pessoa*> alunos;  
        std::list<ConteudoMinistrado*> conteudos;  
};  
#endif
```


Pergunta

- Podemos fazer isso?

```
#include <iostream>
```

```
#include "Disciplina.hpp"
```

```
#include "Professor.hpp"
```

```
int main(){
```

```
    Professor p{"Joao", 11111111111, 30000, 40};
```

```
    Disciplina d{"C++"};
```

```
    d.setProfessor(&p);
```

```
    std::cout << d.getProfessor()->getNome() << std::endl;
```

```
    return 0;
```

```
}
```

```
int main(){
```

```
    Professor p{"Joao", 11111111111, 30000, 40};  
    Disciplina d{"C++"};
```

```
    d.setProfessor(&p);
```

```
    std::cout << d.getProfessor()->getNome() << std::endl;
```

```
    return 0;
```

Pergunta

- Podemos fazer isso?
 - **Sim, podemos!**
- O setProfessor espera }
um ponteiro para **Pessoa**, e estamos passando um professor
 - Mas professor é **um tipo de pessoa**, e isso é aceito

Pergunta

- E isso?

```
#include <iostream>
```

```
#include "Disciplina.hpp"
```

```
#include "Professor.hpp"
```

```
int main(){
```

```
    Professor p{"Joao", 11111111111, 30000, 40};
```

```
    Disciplina d{"C++"};
```

```
    d.setProfessor(&p);
```

```
    std::cout << d.getProfessor()->getNome() << std::endl;
```

```
    std::cout << d.getProfessor()->getSalario() << std::endl;
```

```
    return 0;
```

```
}
```

Pergunta

```
int main(){
```

```
Professor p{"Joao", 1111111111, 30000, 40};
```

```
Disciplina d{"C++"};
```

```
d.setProfessor(&p);
```

```
std::cout << d.getProfessor()->getNome() << std::endl;
```

```
std::cout << d.getProfessor()->getSalario() << std::endl;
```

```
return 0;
```

- Não podemos

- getProfessor retorna um ponteiro para

Pessoa

- Uma pessoa “genérica”
- Na memória essa pessoa é um professor, mas da forma que fizemos, o compilador não tem como saber disso
 - Nem o seu programa em tempo de execução

- Veja o erro:

...

main.cpp:13:33: error: ‘const class Pessoa’ has no member named ‘getSalario’

```
std::cout << d.getProfessor()->getSalario() << std::endl;
```

^~~~~~

...

Disciplina

- Vamos alterar o dado membro de disciplina para Professor, e não Pessoa
 - Não faz sentido aceitar qualquer tipo de pessoa como professor da disciplina

```
//...
class Disciplina{
public:
    Disciplina(std::string nome);
    ~Disciplina();

    const Professor* getProfessor() const;
    void setProfessor(Professor* professor);

    //...
private:
    //...

    Professor* professor;
    std::list<Pessoa*> alunos;
    std::list<ConteudoMinistrado*> conteudos;

};
```

Pergunta

- Com a classe disciplina alterada, podemos fazer isso?

```
int main(){  
    Pessoa p{"Joao"};  
    Disciplina d{"C++"};  
  
    d.setProfessor(&p);  
    std::cout << d.getProfessor()->getNome() << std::endl;  
  
    return 0;  
}
```

Pergunta

- Com a classe disciplina alterada, podemos fazer isso?
 - **Não**
 - Um professor é um tipo de pessoa, mas uma pessoa não é um tipo de professor
 - Veja o erro

```
main.cpp:12:17: error: invalid conversion from 'Pessoa*' to 'Professor*' [-fpermissive]
    d.setProfessor(&p);
                  ^~
```

```
int main(){
    Pessoa p{"Joao"};
    Disciplina d{"C++"};

    d.setProfessor(&p);
    std::cout << d.getProfessor()->getNome() << std::endl;

    return 0;
}
```

Herança

- Geramos ainda alguns outros problemas com nossa hierarquia de classes, mas vamos corrigir nas próximas aulas
 - Essa foi apenas uma introdução ao conceito de herança

Exercícios

1. Crie a classe **Aluno** e faça as atualizações necessárias na classe Disciplina.
2. Pesquise sobre como modelar heranças no diagrama de classes e atualize o diagrama do sistema.

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.
- <https://docs.microsoft.com/en-us/cpp/cpp/protected-cpp?view=vs-2019>