"Abandone toda esperança aquele que por aqui entrar" (Divina Comédia).

Associações e Forward Declarations

Paulo Ricardo Lisboa de Almeida

DSBD

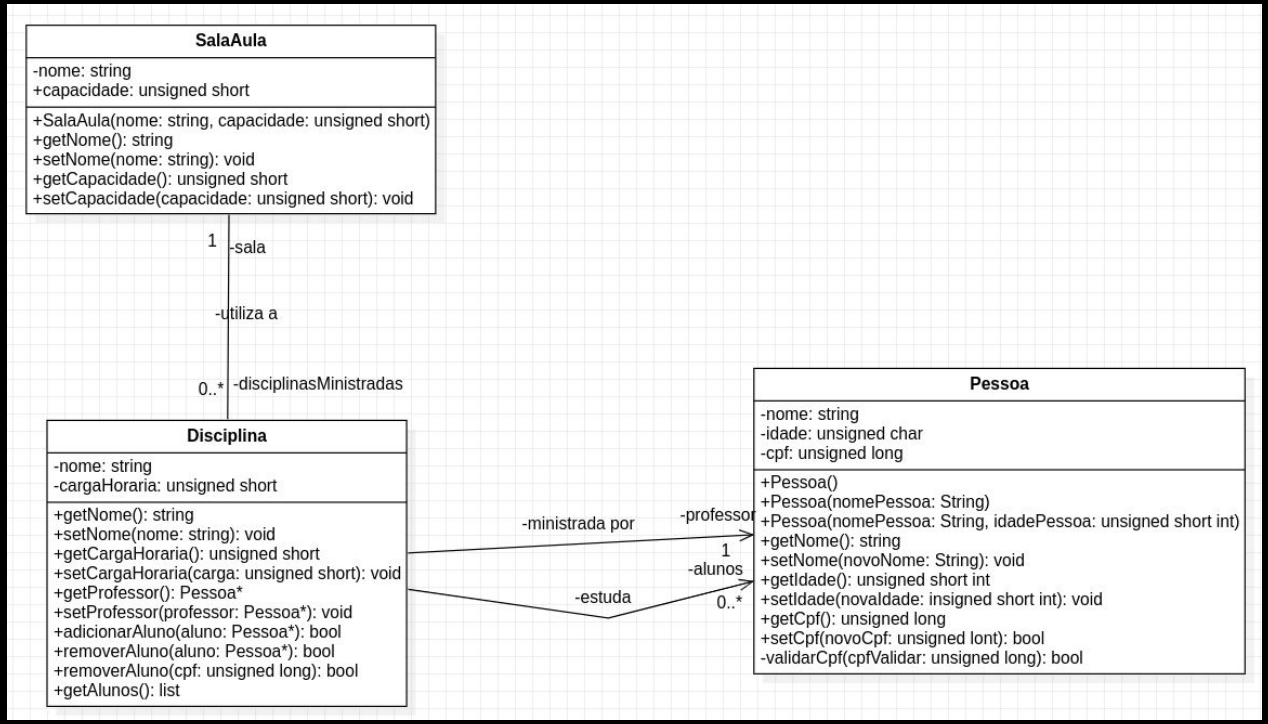
UFPR

Antes de começar

Faça o download do material disponibilizado no site.

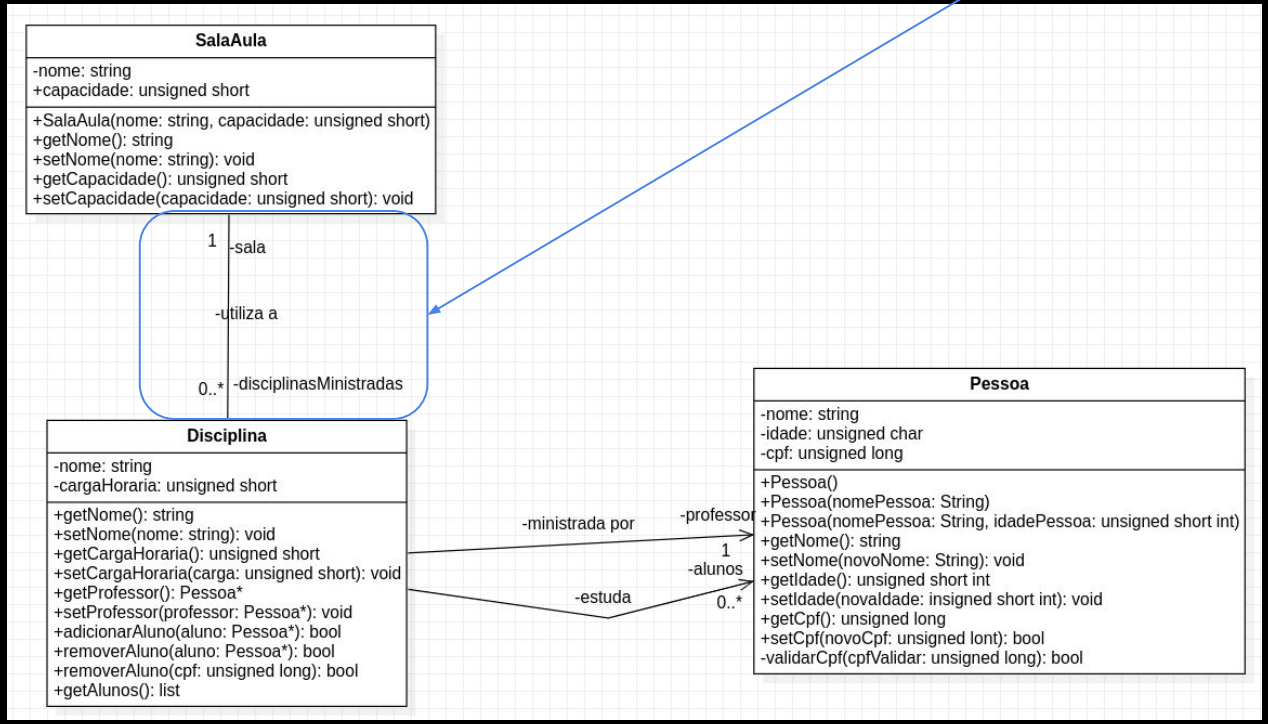
"Programas Antes Aula".
Entenda a classe SalaAula criada.

Diagrama


SalaAula
-nome: string
+capacidade: unsigned short
+SalaAula(nome: string, capacidade: unsigned short)
+getNome(): string
+setNome(nome: string): void
+getCapacidade(): unsigned short
+setCapacidade(capacidade: unsigned short): void
1
-sala
-utiliza a
0..*
-disciplinasMinistradas
Disciplina
-nome: string
-cargaHoraria: unsigned short
+getNome(): string
+setNome(nome: string): void
+getCargaHoraria(): unsigned short
+setCargaHoraria(carga: unsigned short): void
+getProfessor(): Pessoa*
+setProfessor(professor: Pessoa*): void
+adicionarAluno(aluno: Pessoa*): bool
+removerAluno(aluno: Pessoa*): bool
+removerAluno(cpf: unsigned long): bool
+getAlunos(): list
-ministrada por
-professor
1
-alunos
-estuda
0..*
Pessoa
-nome: string
-idade: unsigned char
-cpf: unsigned long
+Pessoa()
+Pessoa(nomePessoa: String)
+Pessoa(nomePessoa: String, idadePessoa: unsigned short int)
+getNome(): string
+setNome(novoNome: String): void
+getIdade(): unsigned short int
+setIdade(novaIdade: insigned short int): void
+getCpf(): unsigned long
+setCpf(novoCpf: unsigned lont): bool
-validarCpf(cpfValidar: unsigned long): bool

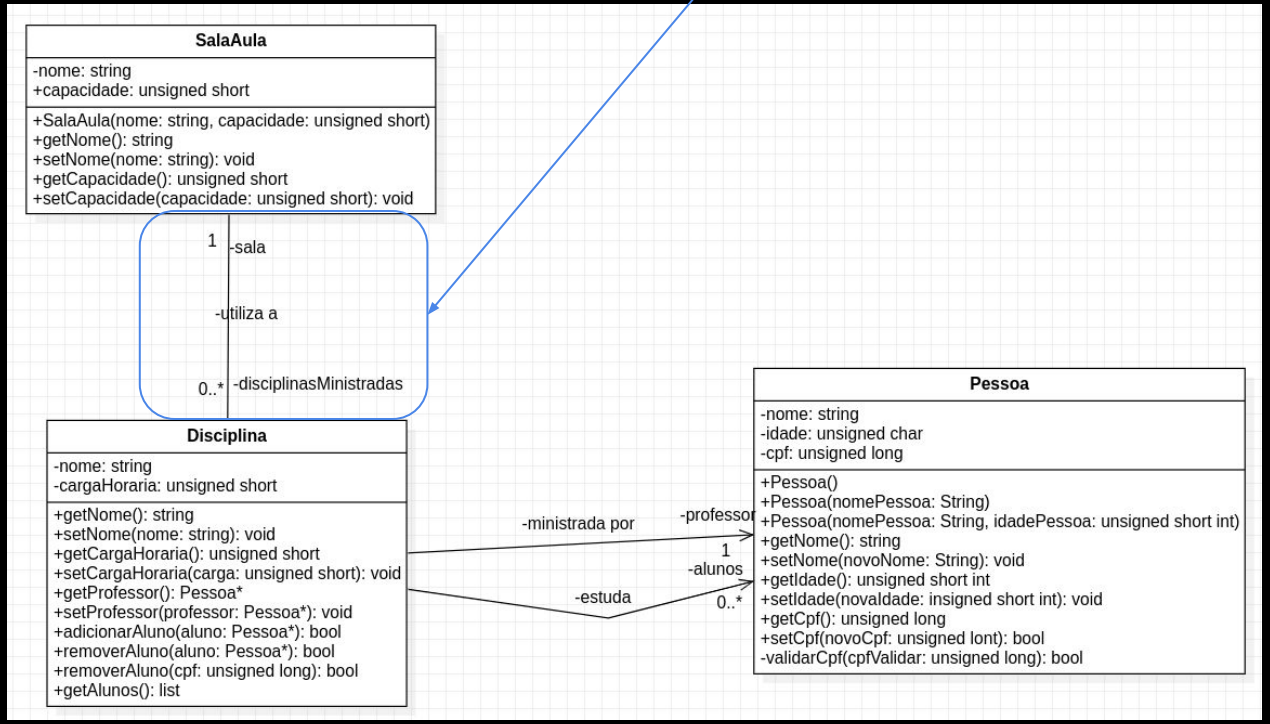
Diagrama

O que isso nos diz exatamente?



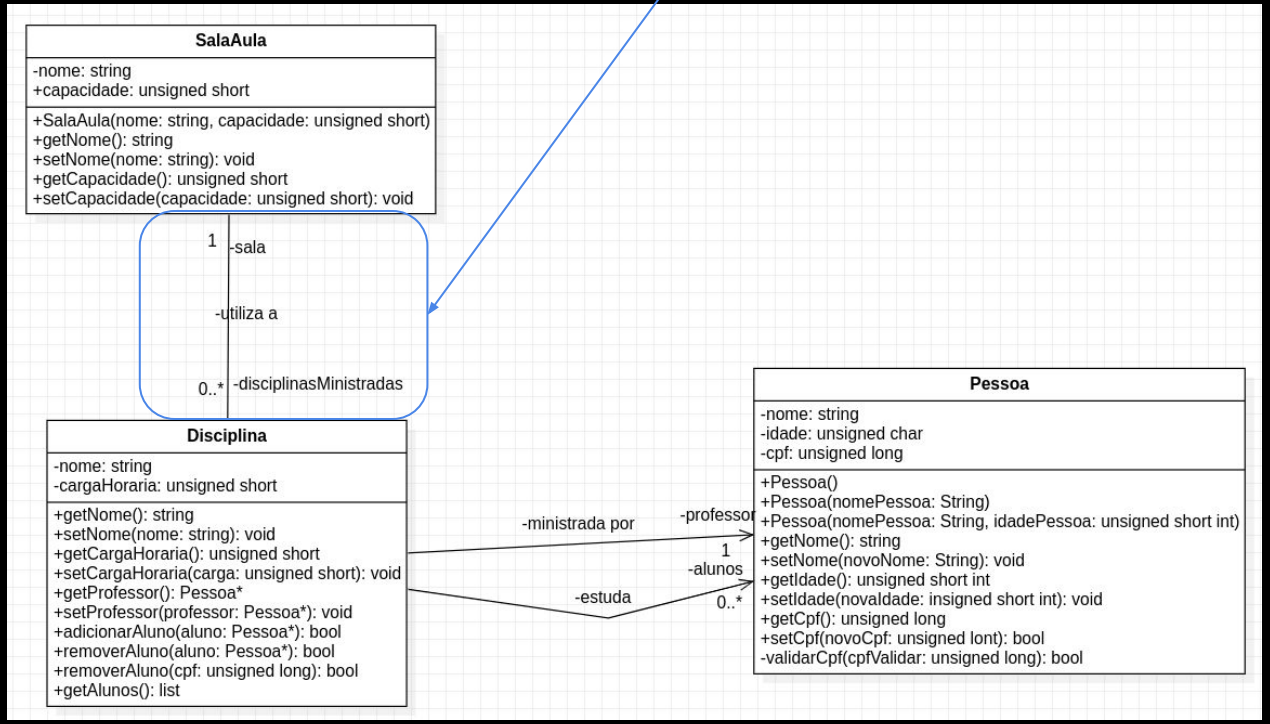
Diagrama

Uma disciplina é ministrada em exatamente uma sala de aula, e várias disciplinas podem ser ministradas em uma sala de aula. Uma associação onde os “dois lados se conhecem”. Note que não há seta indicando a direção da relação.



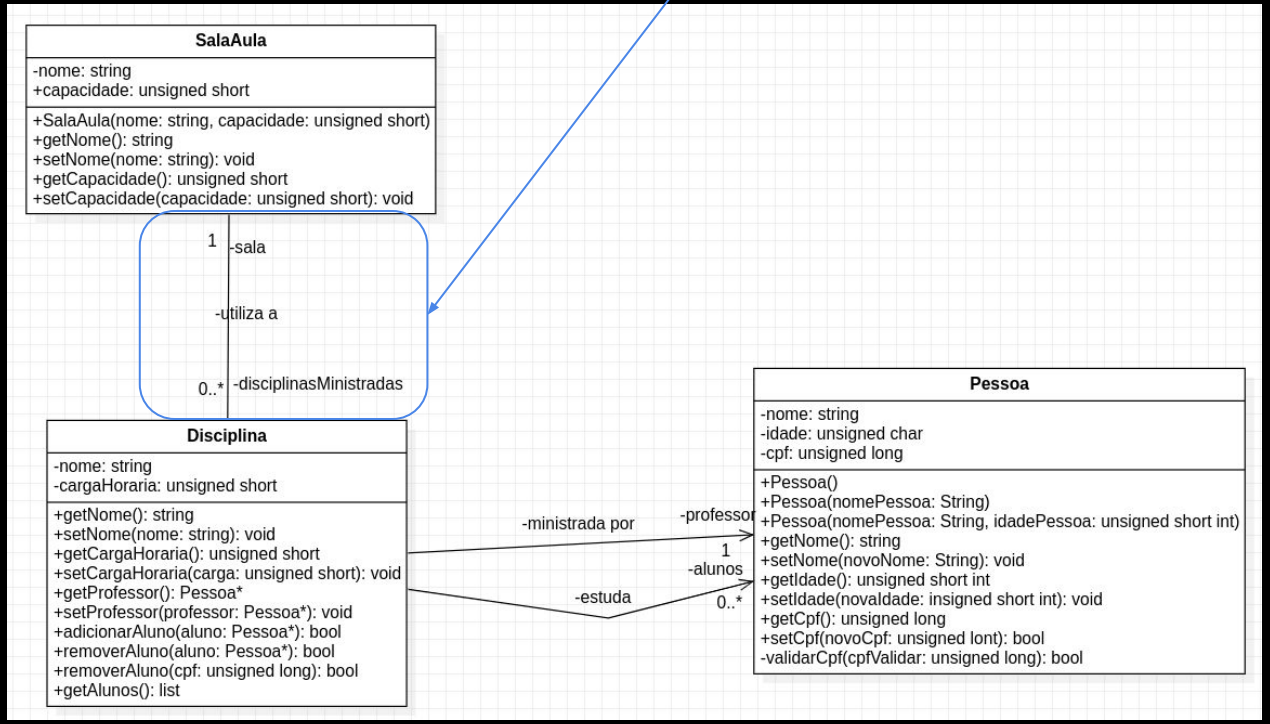
Diagrama

Como implementar isso nas classes *SalaAula* e *Disciplina*?



Diagrama

Disciplina pode conter um ponteiro para SalaAula, e SalaAula pode conter uma lista (ex.: list ou vector) de ponteiros para disciplinas.



Disciplina

```
#include "SalaAula.hpp"

class Disciplina{
   public:

      //...

      void setSalaAula(SalaAula* sala);
      SalaAula* getSalaAula();
   private:
      std::string nome;
      unsigned short int cargaHoraria;
      Pessoa* professor;
      SalaAula* sala;
};
```

```
//...

void Disciplina::setSalaAula(SalaAula* sala){
   //precisariamos ainda verificar se sala não é nullptr
   //faremos isso no tópico exceções no futuro
   this->sala = sala;
}
SalaAula* Disciplina::getSalaAula(){
   return this->sala;
}
```

SalaAula

```cpp
#include "Disciplina.hpp"

class SalaAula{
   public:

      //...

      void adicionarDisciplina(Disciplina* disciplina);
      void removerDisciplina(Disciplina* disciplina);
      std::list<Disciplina*>& getDisciplinas();
   private:
      std::string nome;
      unsigned int capacidade;
      std::list<Disciplina*> disciplinasMinistradas;
};
```

```cpp
//...

void SalaAula::adicionarDisciplina(Disciplina* disciplina){
   disciplinasMinistradas.push_back(disciplina);
}

void SalaAula::removerDisciplina(Disciplina* disciplina){
   disciplinasMinistradas.remove(disciplina);
}

std::list<Disciplina*>& SalaAula::getDisciplinas(){
   return disciplinasMinistradas;
}
```

Faça você mesmo

```
make clean
```

```
make
```

O que acontece?

Faça você mesmo

```
make clean
```

```
make
```

O que acontece?

Desastre!

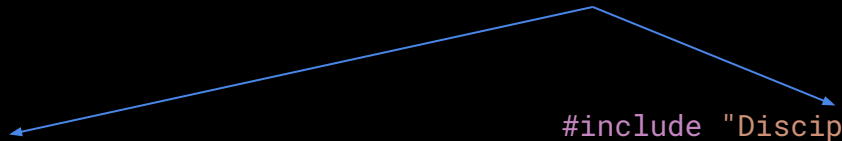
Vários erros estranhos surgem no log de compilação.

Pior, muitos dos erros que aparecem no log nada tem a ver com a raiz do problema.

O que cargas d'água pode estar errado?

Dependência cíclica

Para compilar Disciplina, o compilador precisa compilar SalaAula, e para compilar SalaAula, o compilador precisa compilar Disciplina!



```hpp
#include "SalaAula.hpp"

class Disciplina{
   public:

       //...

       void setSalaAula(SalaAula* sala);
       SalaAula* getSalaAula();
   private:
       std::string nome;
       unsigned short int cargaHoraria;
       Pessoa* professor;
       SalaAula* sala;
};
```

```hpp
#include "Disciplina.hpp"

class SalaAula{
   public:

       //...

       void adicionarDisciplina(Disciplina* disciplina);
       void removerDisciplina(Disciplina* disciplina);
       std::list<Disciplina*>& getDisciplinas();
   private:
       std::string nome;
       unsigned int capacidade;
       std::list<Disciplina*> disciplinasMinistradas;
};
```

Forward Declaration

O **Forward Declaration** de uma classe tem o formato:

```
class NomeClasse;
```

Indica para o compilador que a classe existe, mas que ainda não foi compilada.

Fazemos uma “promessa” ao compilador, dizendo que essa classe existe e será compilada em algum momento.

Forward Declaration

Em Disciplina.hpp Substitua.

#include ”Disciplina.hpp”

por

class SalaAula;

```
class SalaAula;//Forward Declaration

class Disciplina{
   public:

       //...

       void setSalaAula(SalaAula* sala);
       SalaAula* getSalaAula();
   private:
       std::string nome;
       unsigned short int cargaHoraria;
       Pessoa* professor;
       SalaAula* sala;
};
```

Forward Declaration

Informamos ao compilador que existe uma classe `SalaAula`.

Mas não a incluímos em `Disciplina`.

Com o *forward declaration*, o include da classe `SalaAula` pode ficar no .cpp de `Disciplina`.

```cpp
#include "Disciplina.hpp"

#include <iostream>
#include "SalaAula.hpp"

Disciplina::Disciplina(std::string nome)
    :nome{nome} {
}

std::string Disciplina::getNome(){
    return nome;
}

// ...
```

Cuidado

Use com parcimônia.

Forward declarations dificultam o trabalho do compilador e do linkeditor (e o seu também).

- Em caso de erro o compilador pode te dar um erro maluco!

A compilação se torna mais complexa e lenta.

- Em casos extremos, foward declarations incorretos podem gerar linkedições incorretas!
- O linkeditor pode utilizar uma função membro de uma classe que não era a que você tinha em mente!

Resumo: Use somente quando estritamente necessário!

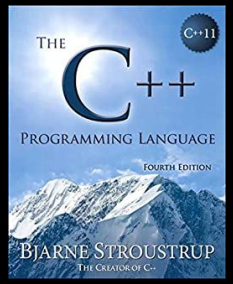
Exercícios

1. Modifique o setSalaAula de Disciplina, adicionando a disciplina na SalaAula via adicionarDisciplina automaticamente. Basicamente ao se alterar a sala de aula da disciplina, essa disciplina vai se remover automaticamente da sala antiga, e se adicionar automaticamente na nova sala. Note que os objetos ainda poderão ficar inconsistentes se o programador utilizar a função adicionarDisciplina de disciplina.

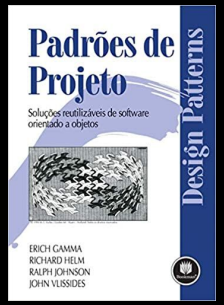
2. Opcional: Tente fazer com que ao adicionar a Disciplina em SalaAula via adicionarDisciplina, a SalaAula de Disciplina também seja atualizada automaticamente. Nesse caso não importa se vamos atualizar a relação a partir de SalaAula ou a partir de Disciplina, tudo se mantém consistente.

Referências

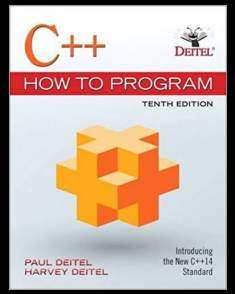
Bjarne Stroustrup. The C++ Programming Language. Addison-Wesley, 2013.



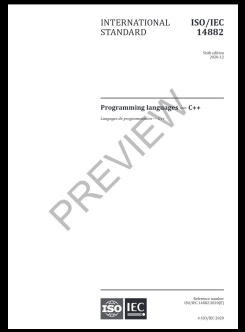
Gamma, E. et Al. Padrões de Projetos: Soluções Reutilizáveis. Bookman. 2009.



Deitel, H. M., Deitel, P. J. C++: como programar. 5a ed. Pearson Prentice Hall. 2006.



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



Licença