Não jogue xadrez com pombos!

Membros Estáticos

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR

Conteúdo Ministrado

Analise a classe `ConteudoMinistrado` disponibilizada com o projeto.

Representa um conteúdo (ex.: aula sobre ponteiros) que foi ministrado em determinada disciplina, e a quantidade de horas que foram gastas nesse conteúdo.

Identificador Único

Adicionar um identificador único para cada objeto do tipo `ConteudoMinistrado` criado de maneira automática.

Dados membro estáticos

Um dado membro estático pertence a classe, e não aos objetos.

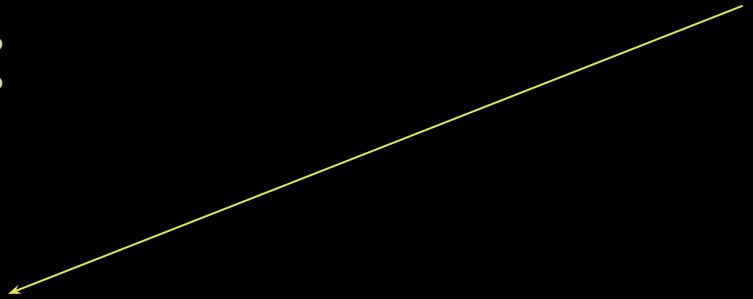
Um dado membro estático é compartilhado entre todos os objetos da classe (entre todas as instâncias).

Para declarar um dado membro como estático, basta adicionar o modificador `static` na sua declaração:

```
static tipoDado nomeDado;
```

Exemplo

Público por enquanto para facilitar os testes



```
#ifndef CONTEUDO_MINISTRADO_HPP
#define CONTEUDO_MINISTRADO_HPP

#include<string>

class ConteudoMinistrado{
   public:
       static unsigned int proxId;

       ConteudoMinistrado(std::string descricao, unsigned short cargaHorariaConteudo);

       std::string& getDescricao();
       unsigned short getCargaHorariaConteudo();
       unsigned int getId();
   private:
       std::string descricao;
       unsigned short cargaHorariaConteudo;
       unsigned int id;
};
#endif
```

Inicialização

Por padrão, dados membro estáticos primitivos são inicializados com zero pelo compilador.

Caso você não indique como inicializar.

Leia na seção 9.15 static Class Members de Deitel, Deitel, P. J. C++: como programar. 10a ed.

Dados estáticos são inicializados no cpp.

Não incluímos o modificador static no inicializador.

```
tipoDado NomeClasse::nomeDado{valorInicial}.
```

Exemplo

```cpp
#include "ConteudoMinistrado.hpp"

unsigned int ConteudoMinistrado::proxId{0};

ConteudoMinistrado::ConteudoMinistrado(std::string descricao, unsigned short cargaHorariaConteudo):
   descricao(descricao), cargaHorariaConteudo(cargaHorariaConteudo), id{0}{
}

std::string& ConteudoMinistrado::getDescricao(){
   return descricao;
}

unsigned short ConteudoMinistrado::getCargaHorariaConteudo(){
   return cargaHorariaConteudo;
}

unsigned int ConteudoMinistrado::getId(){
   return id;
}
```

Acessando

Uma forma é acessar os dados (e funções estáticas) como qualquer dado normal.

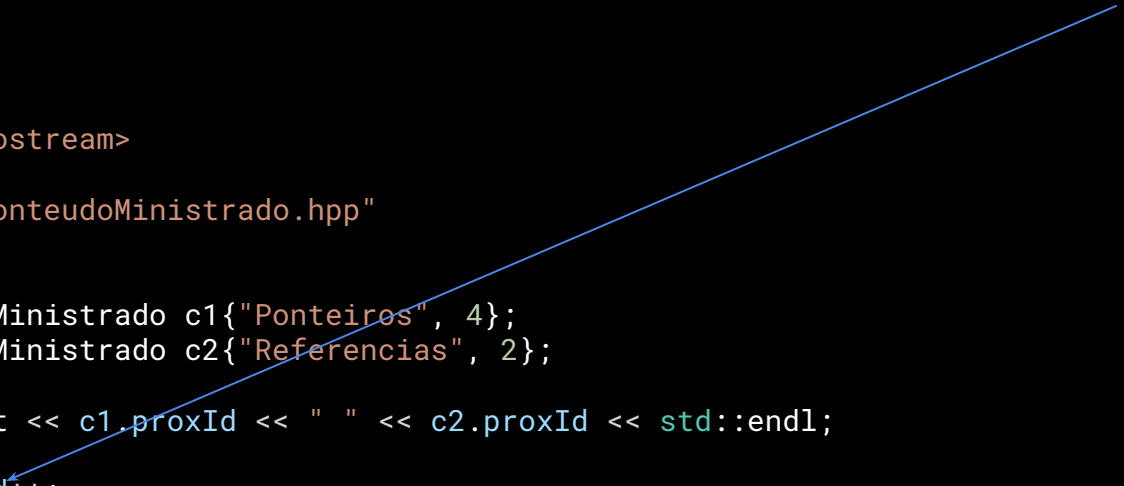
Podemos utilizar, por exemplo, o operador . (ponto).

Exemplo

```cpp
#include <iostream>

#include "ConteudoMinistrado.hpp"

int main(){
   ConteudoMinistrado c1{"Ponteiros", 4};
   ConteudoMinistrado c2{"Referencias", 2};

   std::cout << c1.proxId << " " << c2.proxId << std::endl;

   c1.proxId++;
   std::cout << c1.proxId << " " << c2.proxId << std::endl;

   return 0;
}
```

Exemplo

Alteramos *proxId* através de *c1*, mas o dado acessado por *c2* também é alterado.
É o mesmo dado, pois o dado estático pertence a classe.
Existe apenas uma cópia desse dado na memória.



```cpp
#include <iostream>

#include "ConteudoMinistrado.hpp"

int main(){
   ConteudoMinistrado c1{"Ponteiros", 4};
   ConteudoMinistrado c2{"Referencias", 2};

   std::cout << c1.proxId << " " << c2.proxId << std::endl;

   c1.proxId++;
   std::cout << c1.proxId << " " << c2.proxId << std::endl;

   return 0;
}
```

Acessando via a Classe

Acessamos o dado estático via objetos.

- Esse tipo de acesso é válido, mas é uma **má prática**.
- Passa a impressão que estamos acessando um dado/função membro convencional.

Uma abordagem melhor é realizar acessos através da classe:

```
NomeClasse::nomeDadoEstatico;
```

Deixar claro que estamos acessando algo que pertence a classe.

Exemplo

```cpp
#include <iostream>

#include "ConteudoMinistrado.hpp"

int main(){
   ConteudoMinistrado c1{"Ponteiros", 4};
   ConteudoMinistrado c2{"Referencias", 2};

   std::cout << c1.proxId << " " << c2.proxId << "\n";

   ConteudoMinistrado::proxId++;
   std::cout << ConteudoMinistrado::proxId << "\n";

   return 0;
}
```

Podemos fazer isso?

Podemos fazer isso? Note que não temos objetos do tipo `ConteudoMinistrado` na memória!

```cpp
#include <iostream>

#include "ConteudoMinistrado.hpp"

int main(){
   ConteudoMinistrado::proxId++;
   std::cout << ConteudoMinistrado::proxId << std::endl;

   return 0;
}
```

Podemos fazer isso?

Podemos fazer isso? Note que não temos objetos do tipo `ConteudoMinistrado` na memória!

O dado pertence a classe, e não aos objetos. Podemos acessar os dados estáticos mesmo sem nenhum objeto instanciado.

```
#include <iostream>

#include "ConteudoMinistrado.hpp"

int main(){
   ConteudoMinistrado::proxId++;
   std::cout << ConteudoMinistrado::proxId << std::endl;

   return 0;
}
```

Identificador Único

Como usar a ideia de um dado estático para atribuir um identificador único para cada objeto?

Id único automático

```
#ifndef CONTEUDO_MINISTRADO_HPP
#define CONTEUDO_MINISTRADO_HPP

#include<string>

class ConteudoMinistrado{
   public:
       ConteudoMinistrado(std::string descricao, unsigned short cargaHorariaConteudo);

       std::string& getDescricao();
       unsigned short getCargaHorariaConteudo();
       unsigned int getId();
   private:
       static unsigned int proxId;

       std::string descricao;
       unsigned short cargaHorariaConteudo;
       unsigned int id;
};
#endif
```

Id único automático

```cpp
#include "ConteudoMinistrado.hpp"

unsigned int ConteudoMinistrado::proxId{0};

ConteudoMinistrado::ConteudoMinistrado(std::string descricao, unsigned short cargaHorariaConteudo):
   descricao(descricao), cargaHorariaConteudo(cargaHorariaConteudo), id{ConteudoMinistrado::proxId}{
       ConteudoMinistrado::proxId++;
}

std::string& ConteudoMinistrado::getDescricao(){
   return descricao;
}

unsigned short ConteudoMinistrado::getCargaHorariaConteudo(){
   return cargaHorariaConteudo;
}

unsigned int ConteudoMinistrado::getId(){
   return id;
}
```

Data Races

A atualização do dado estático no construtor funciona assumindo que temos apenas um processo ou thread sendo executado.

Programas com múltiplos processos ou threads, que são comuns em arquiteturas web (mesmo que você não esteja consciente disso) vão causar problemas de concorrência.

Como resolver?

Data Races

A atualização do dado estático no construtor funciona assumindo que temos apenas um processo ou thread sendo executado.

Programas com múltiplos processos ou threads, que são comuns em arquiteturas web (mesmo que você não esteja consciente disso) vão causar problemas de concorrência.

Como resolver?

> Exemplo: mutexes ou semáforos.
> Veja detalhes em disciplinas como Sistemas Operacionais.

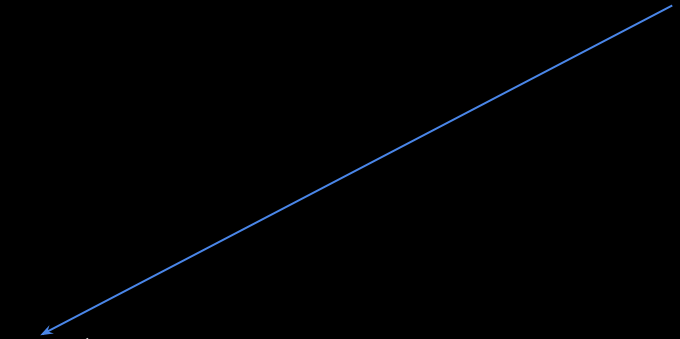
Adicionando em disciplina

Vamos adicionar uma lista de conteúdos ministrados na disciplina.

A classe disciplina terá uma função que criará os conteúdos automaticamente.

Disciplina.hpp

Adicionar componentes dessa forma segue uma abordagem similar a função `emplace` encontrada em muitas das estruturas da STL. Pesquise!



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

       //...

       void adicionarConteudoMinistrado(std::string conteudo, unsigned short cargaHorariaConteudo);
       void imprimirConteudosMinistrados();
       std::list<ConteudoMinistrado*>& getConteudos();
   private:
       std::string nome;
       unsigned short int cargaHoraria;
       Pessoa* professor;
       SalaAula* sala;

       std::list<ConteudoMinistrado*> conteudos;
};
```

Disciplina.cpp

```
//...

void Disciplina::adicionarConteudoMinistrado(std::string conteudo, unsigned short cargaHorariaConteudo){
   this->conteudos.push_back(new ConteudoMinistrado{conteudo, cargaHorariaConteudo});
}

void Disciplina::imprimirConteudosMinistrados(){
   std::list<ConteudoMinistrado*>::iterator it;
   for(it = conteudos.begin(); it!=conteudos.end(); it++){
       std::cout << "Id: " << (*it)->getId() << std::endl
           << "Conteudo: " << (*it)->getDescricao() << std::endl
           << "Carga: " << (*it)->getCargaHorariaConteudo() << std::endl << std::endl;
   }
}

std::list<ConteudoMinistrado*>& Disciplina::getConteudos(){
   return this->conteudos;
}
```

main.cpp

```cpp
#include <iostream>

#include "Pessoa.hpp"
#include "Disciplina.hpp"
#include "ConteudoMinistrado.hpp"

int main(){
   Pessoa prof1{"João", 11111111111, 40};
   Disciplina dis1{"C++", nullptr};
   dis1.setProfessor(&prof1);

   dis1.adicionarConteudoMinistrado("Ponteiros", 4);
   dis1.adicionarConteudoMinistrado("Referencias", 2);

   dis1.imprimirConteudosMinistrados();

   return 0;
}
```

Mais utilizações

Dados estáticos também são comumente utilizados para representar constantes em nossos programas.

Algo que fazíamos via #define em C.

Vantagens:

- Flexibilidade;
- Compilação mais simples;
- Evita colisões com nomes de macros.

Comumente isso é feito em conjunto com o modificador `const` ou `constexpr` (veremos em detalhes no futuro).

Classe Console

Considere agora a classe Console disponibilizada no projeto.

Edite o makefile para compilar essa classe também.

Função Estática

Da mesma forma que um dado estático, uma **função estática pertence a classe**, e não aos objetos.

Pode ser acessada diretamente através da classe.

- Não precisa de um objeto instanciado para se ter acesso.

Classe Console

A função membro `imprimirDados` da classe Console não acessa nenhum dado membro da classe `Console`.

Essa é uma função utilitária da classe.

Nesse caso, faz sentido transformar essa função membro em estática.

Não precisamos instanciar objetos para usar a função.

Funções Estáticas

Basta adicionar o modificador `static` no protótipo da função no .hpp.

O cpp não é alterado.

Para acessar uma função estática, utilizamos o operador `::`

Exemplo

```cpp
#ifndef CONSOLE_HPP
#define CONSOLE_HPP

#include "Disciplina.hpp"

class Console{
   public:
       static void imprimirDadosDisciplina(Disciplina& disciplina);

};
#endif
```

```cpp
int main(){
   Pessoa prof1{"João", 11111111111, 40};
   Disciplina dis1{"C++", nullptr};
   dis1.setProfessor(&prof1);

   dis1.adicionarConteudoMinistrado("Ponteiros", 4);
   dis1.adicionarConteudoMinistrado("Referencias", 2);

   Console::imprimirDadosDisciplina(dis1);

   return 0;
}
```

Podemos fazer isso?

```hpp
#ifndef CONSOLE_HPP
#define CONSOLE_HPP

#include "Disciplina.hpp"

class Console{
   public:
       static void imprimirDadosDisciplina(Disciplina& disciplina);
   private:
       std::string cabecalho;

};
#endif
```

```cpp
void Console::imprimirDadosDisciplina(Disciplina& disciplina){
   std::cout << cabecalho << std::endl;

   std::cout << "Disicplina: "<< disciplina.getNome() << std::endl;
   //...
}
```

Podemos fazer isso?

```
#ifndef CONSOLE_HPP
#define CONSOLE_HPP

#include "Disciplina.hpp"

class Console{
   public:
       static void imprimirDadosDisciplina(Disciplina& disciplina);
   private:
       std::string cabecalho;

};
#endif
```

Uma função estática não pode acessar um dado membro não estático.
Estamos tentando acessar o dado membro de um objeto.
De qual objeto!?
Funções e dados estáticos pertencem a classe, e não aos objetos.

```
void Console::imprimirDadosDisciplina(Disciplina& disciplina){
   std::cout << cabecalho << std::endl;

   std::cout << "Disicplina: "<< disciplina.getNome() << std::endl;
   //...
}
```

Funções Estáticas

Funções estáticas **podem** acessar dados estáticos da classe.

This

O que você acha da construção a seguir?

Considerando ainda que a função `imprimirDadosDisciplina` é estática.

```
#include "Console.hpp"

#include<iostream>

void Console::imprimirDadosDisciplina(Disciplina& disciplina){
      std::cout << Meu endereco:  << this << std::endl;
}
```

This

O ponteiro `this` não existe para uma função estática.

`this` aponta para o objeto atual, mas a função estática não pertence a nenhum objeto.

```
#include "Console.hpp"

#include<iostream>

void Console::imprimirDadosDisciplina(Disciplina& disciplina){
      std::cout << Meu endereco:  << this << std::endl;
}
```

Dependência

A classe `Console` não possui nenhum dado membro `Disciplina`.

Mas a classe Disciplina é necessária para se executar a sua função `imprimirDadosDisciplina`.

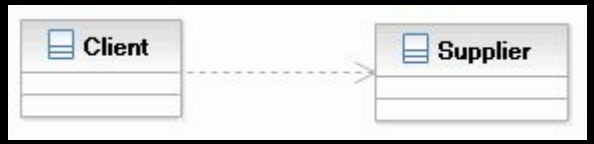
Isso se trata de uma **Dependência**.

Em uma dependência, **uma classe depende da outra para executar uma tarefa.**

Mas não possui nenhuma referência interna (no formato de um dado membro) ligando-a a essa classe.

Dependência

No diagrama de Classes UML, a dependência é anotada como uma seta tracejada.

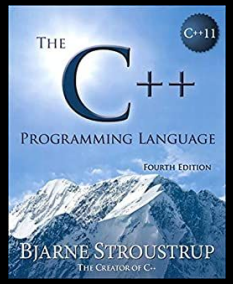


Exercícios

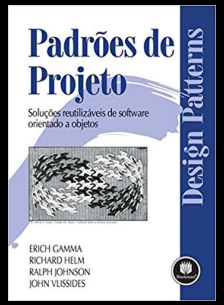
1. Pesquise mais sobre o relacionamento de dependência.
    a. Alguns links interessantes:
        i. https://www.ibm.com/support/knowledgecenter/pt-br/SSCLKU_7.5.5/com.ibm.xtools.modeler.doc/topics/cdepend.html
        ii. Seção 10.5 do livro "Pressman, R.;Maxim, B. Engenharia de Software: uma abordagem Profissional. McGraw Hill Brasil, 2016. 8 ed. ISBN 9788580555349."
            A seção está disponível gratuitamente em books.google.com
2. Complete a classe Disciplina.
    a. Adicione uma função removerConteudoMinistrado(unsigned long id)
    b. Adicione uma função que retorna a lista de conteúdos ministrados.
    c. Apague da memória os conteúdos ministrados (que foram alocados dinamicamente) antes de terminar o programa
        i. Você pode, por exemplo, criar uma função em Disciplina chamada "limparConteudos" que libera a memória, e chamar essa função antes de terminar o programa.
3. Atualize o diagrama de classes para conter todas as classes do sistema até agora.
    a. Dica: Um dado estático é representado com um sublinhado no diagrama de classes. Selecione o dado no StarUML, e marque a opção "isStatic".
    b. Na classe retângulo (solicitada em aulas passadas) crie uma função membro que retorna quantos retângulos já foram criados no programa.

Referências

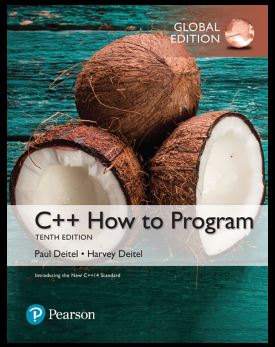
Bjarne Stroustrup. The C++ Programming Language. Addison-Wesley, 2013.



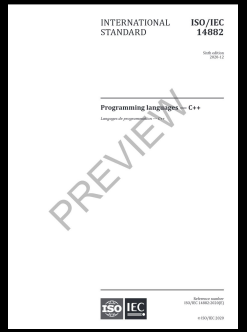
Gamma, E. Padrões de Projetos: Soluções Reutilizáveis. Bookman. 2009.



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