Membros de dados *static*

Em nossos estudos iniciais sobre classes, vimos que cada objeto de uma classe possui sua cópia dos membros de dados da classe. Porém, existe uma exceção a esta regra. Em certos casos, apenas uma cópia de certo membro de dados é compartilhada por todos os objetos da classe.

Este é denominado membro de dados *static* e representa uma informação da classe compartilhada por todas as instâncias, não uma informação de um objeto específico da classe.

Um exemplo muito interessante, utilizado em [3], é o seguinte: Imagine um jogo de videogame com marcianos e outras criaturas do espaço. Cada marciano tende a ser corajoso e atacar outras criaturas espaciais quando o marciano está ciente de que há pelo menos cinco marcianos presentes.

Se menos de cinco marcianos estiverem presentes, cada marciano individualmente torna-se covarde. Assim, cada marciano precisa saber a quantidadeDeMarcianos.

Seria interessante fornecer a cada marciano uma cópia de quantidadeDeMarcianos?

Não.

Por que?

Dificuldades de atualização e duplicidade de informação.

Em vez disto, declaramos quantidadeDeMarcianos como *static*, permitindo assim que todos objetos instanciados da classe possam acessar quantidadeDeMarcianos e o programa manterá apenas uma cópia deste membro de dados.

Um membro de dados *static* de um tipo fundamental é inicializado por padrão como zero. Para inicializar com um valor diferente, um membro de dados *static* pode ser inicializado uma única vez.

Para satisfazer a esta restrição os membros de dados static devem ser definidos no escopo do arquivo, ou seja, fora do corpo da definição de classe e inicializados somente nessas definições. É interessante salientar que membros de dados static que são de tipos de classe são inicializados pelos seus construtores-padrão.

Os membros static da classe existem até mesmo quando não existe nenhum objeto desta classe. Para acessar um membro public static quando não existe nenhum objeto da classe, prefixe o nome de classe e o operador de resolução de escopo binário com o nome do membro de dados.

Você visualiza necessidade de uma função membro static?

Qual?

Para acessar um membro *private* ou *protected static* quando não existe nenhum objeto da classe, torna-se necessária uma função membro *public static*.

Em outras palavras, uma função membro *static* é um serviço da classe e não de um objeto específico da classe.

Funções membros *static* podem acessar apenas os membros de dados *static* da classe.

Um exemplo facilitará a compreensão.

```
//conteúdo do arquivo empregado.h
#ifndef EMPREGADO_H
#define EMPREGADO_H
  class Empregado
  {
  public:
    Empregado(char [30], char [30]);
    ~Empregado();
    void getPrimeiroNome(char [30]);
    void getUltimoNome(char [30]);
    static int getContador();
```

```
  private:
    char primeiroNome[30];
    char ultimoNome[30];
    static int contador;
  };
#endif

//conteúdo do arquivo empregado.cpp
#include <iostream>
#include <cstring>
using namespace std;
#include "empregado.h"
int Empregado::contador = 0;
int Empregado::getContador()
{
  return contador;
}
Empregado::Empregado(char primeiro[30], char ultimo[30])
{
  strcpy(primeiroNome, primeiro );
  strcpy(ultimoNome, ultimo );
  contador++;
```

```
  cout << "Construtor executado para " << primeiroNome <<
  ' ' << ultimoNome << '.' << endl;
}
Empregado::~Empregado()
{
  cout << "Destrutor chamado para " << primeiroNome
    << ' ' << ultimoNome << endl;
  contador--;
}
void Empregado::getPrimeiroNome(char primeiro[30])
{
   strcpy(primeiro, primeiroNome);
}
void Empregado::getUltimoNome(char ultimo[30])
{
   strcpy(ultimo, ultimoNome);
}

//conteúdo do arquivo principalEmpregado.cpp
#include <iostream>
using std::cout;
using std::endl;
```

```
#include "empregado.h"
int main() {
   char nome[30];
   cout << "O numero de empregados antes da instanciacao de " <<
  "algum objeto eh " << Empregado::getContador() << endl;
   Empregado *e1Ptr = new Empregado( "Maria", "Aparecida" );
   Empregado *e2Ptr = new Empregado( "Roberto", "Carlos" );
   cout << "O numero de empregados depois da instanciacao " <<
   "de objetos eh " << e1Ptr->getContador();
   cout << endl << endl << "Empregado 1: ";
   e1Ptr->getPrimeiroNome(nome);
   cout << nome << " ";
   e1Ptr->getUltimoNome(nome);
   cout << nome << endl << "Empregado 2: " ;
   e2Ptr->getPrimeiroNome(nome);
   cout << nome << " ";
   e2Ptr->getUltimoNome(nome);
   cout << nome << endl << endl;
   delete e1Ptr;
   e1Ptr = 0;
   delete e2Ptr;
   e2Ptr = 0;
   cout << "O numero de empregados depois dos objetos deletados
eh " << Empregado::getContador() << endl;
   return 0;
}
```

Exercício:

Crie uma classe Poupanca. Utilize um membro de dados static taxaDeJurosAnual para armazenar a taxa de juros anual para os correntistas. Cada objeto da classe deve conter um membro de dados privado saldo para indicar a quantia que os correntistas têm anualmente em depósito. Forneça uma função membro calculaJurosMensais que calcula os juros mensais multiplicando o saldo pela taxaDeJurosAnual e dividindo por 12; esse valor deve ser adicionado a saldo. Forneça uma função membro static modifiqueTaxaDeJurosAnual que configura taxaDeJurosAnual com um novo valor. Escreva um programa driver para testar a classe Poupanca, instanciando dois objetos diferentes da classe Poupanca, o1 e o2, com saldos de R$ 2.000,00 e R$ 3.000,00, respectivamente. Configure a taxaDeJurosAnual como 3%. Em seguida, calcule os juros mensais e imprima os novos saldos de cada um dos correntistas. Então configure a taxaDeJurosAnual como 4%, calcule os juros do próximo mês e imprima os novos saldos para cada um dos poupadores.