INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PARAÍBA
Campus Cajazeiras

Engenharia de Controle e Automação
Programação Orientada a Objetos – 2021.2

Professor: Leandro Luttiane da Silva Linhares
E-mail: leandro.luttiane@ifpb.edu.br

Documento Texto
Membros Estáticos e Construtores

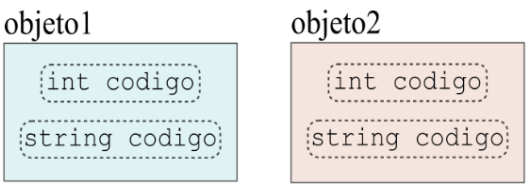
Nesta semana da disciplina de **Programação Orientada a Objetos** (POO) do **Curso de Engenharia de Controle e Automação**, durante o encontro síncrono, foi apresentado um tópico de estudo bastante simples, mas que é fundamental no paradigma de POO: os métodos construtores. Pode-se afirmar que os construtores dão início ao **ciclo de vida dos objetos**, uma vez que quando um objeto é instanciado sempre ocorre a chamada de algum construtor, seja ele o construtor padrão definido pelo próprio compilador ou um construtor definido/implementado pelo desenvolvedor da classe.

Neste documento, um novo conceito será apresentado, o conceito de membros (atributos ou métodos) estáticos. Estes componentes estão mais vinculados à definição da classe em si do que aos objetos de uma classe. Por essa razão, eles não precisam que objetos de uma classe sejam instanciados para que possam ser acessados/utilizados.

Juntamente com este documento estão sendo disponibilizadas as implementações de três classes: `ObjetosInstanciados`, `Moeda` e `Moeda1`. Estas classes são apenas de intuito didático e foram implementadas de acordo com o apresentado no livro **Treinamento em Linguagem C++ Módulo 2** (MIZRAHI, 2005), sendo feitas pequenas modificações.

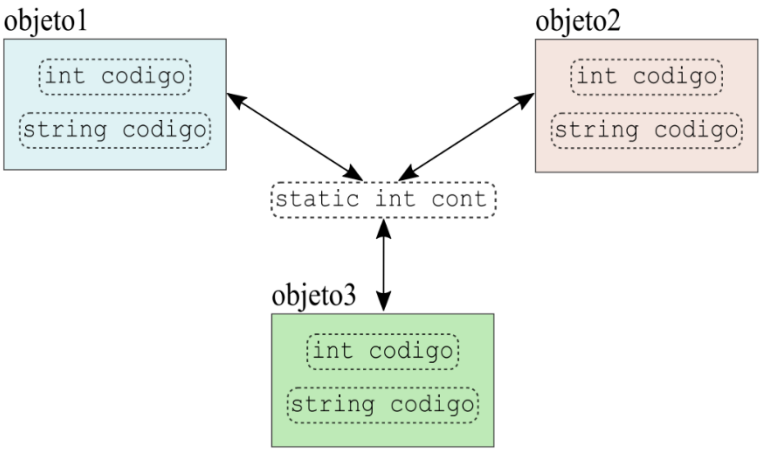
1. Atributos Estáticos

Sabe-se que cada objeto instanciado de uma classe possui os seus próprios atributos, individualizados. Suponha uma classe que defina os atributos (não estáticos, ou seja, automáticos) `codigo` (`int`) e `descricao` (`string`). Se forem instanciados dois objetos desta classe, cada um destes objetos possuirá seus próprios atributos `codigo` e `descricao`, com possíveis valores distintos armazenados em diferentes posições de memória.



Algumas vezes, entretanto, é necessário que todos os objetos instanciados de uma mesma classe tenham acesso a um atributo que é único para a classe. Isso é possível ao declarar um determinado atributo com classe de armazenamento estático (`static`). Neste caso, o objeto

`static` dará origem a um membro de dado único para toda a classe, não importando o número de objetos da classe que sejam instanciados. A informação contida em um atributo `static` é **compartilhada** por todos os objetos da mesma classe. Este atributo estático não é criado para cada um dos objetos instanciados da classe. Ele é criado uma única vez, no início da execução do programa, e permanece disponível para todos os objetos da classe que forem sendo criados durante a execução de um determinado programa.



Um atributo `static` apresenta características similares às variáveis e aos objetos estáticos. Eles são visíveis somente dentro da classe (escopo em que são criados) e apenas deixam de existir quando o programa tem a sua execução concluída. Vamos entender isto melhor a partir da apresentação de um exemplo. Suponha que um certo objeto precise ter o conhecimento de quantos objetos de uma determinada classe encontram-se instanciados em um dado momento. A classe `ObjetosInstanciados` que acompanha este documento trata exatamente deste ponto. Abra os arquivos desta classe e acompanhe as explicações que serão dadas a seguir.

No arquivo de definição, `ObjetosInstanciados.hpp`, pode-se observar a existência de um **construtor** (linha 9) e um **destrutor** (linha 13) para a classe. O conceito de destrutores ainda não foi abordado durante a disciplina. Para este momento, tenha em mente que os destrutores são métodos que são sempre chamados quando um objeto instanciado for deixar de existir, ou seja, quando o seu escopo for finalizado. Os métodos destrutores possuem o mesmo nome da classe, mas com um til "~" como prefixo (ver linha 13 de `ObjetosInstanciados.hpp`).

```
6  class ObjetosInstanciados
7  {
8      public:
9          ObjetosInstanciados();
10
11         int getNObjetos() const;
12
13         ~ObjetosInstanciados();
14
15     private:
16         static int nObjetos;
17 };
```

Em relação aos construtores, foi visto no encontro síncrono da disciplina, que estes métodos devem possuir o mesmo nome da classe e que eles são chamados no momento de instanciação de seus objetos. Desta forma, pode-se afirmar que os métodos construtores são os primeiros métodos executados pelos objetos. Mais de um método construtor pode ser definido em uma classe. Eles devem possuir o mesmo nome, não apresentam tipo de retorno, mas devem possuir um conjunto de parâmetros de entrada diferentes. Desta forma, no momento da instanciação do objeto, o compilador poderá definir qual construtor será executado. Mesmo quando nenhum construtor é expressamente definido/implementado na classe, ela possuirá um construtor considerado padrão sem execução de código.

Na definição da classe `ObjetosInstanciados`, pode-se observar a presença de um método público chamado `getNObjetos()`. Este método retorna um número inteiro que representa a quantidade de objetos da classe atualmente instanciados. O `const` utilizado na definição de `getNObjetos()` apenas indica que este método não modificará o estado do objeto, ou seja, não alterará os valores de seus atributos. Falando em atributos, para esta classe há um único membro de dados chamado `nObjetos` (linha 16), que foi declarado como `private` e `static`. Este atributo é utilizado para armazenar a quantidade de objetos instanciados da classe.

No arquivo de implementação, `ObjetosInstanciados.cpp`, na linha 5, é realizada a associação do atributo estático `nObjetos` com a classe `ObjetosInstanciados`. É necessário realizar este procedimento para que este atributo seja reconhecido no código incluído no arquivo de implementação da classe. Isso é realizado de forma semelhante à associação dos métodos: tipo de dado (`int`), nome da classe (`ObjetosInstanciados`), operador de resolução de escopo "`::`" e o nome do atributo estático (`nObjetos`). Observem que o termo `static` não é utilizado. Ele aparece no início da linha 5 como comentário, apenas para deixar claro que se trata de um atributo estático.

```
5  /* static */ int ObjetosInstanciados::nObjetos;
```

Nas linhas 8-11, tem-se a implementação do construtor da classe `ObjetosInstanciados`. Toda vez que um objeto é criado, o atributo estático `nObjetos` é incrementado em uma unidade. Nas linhas 19-23, pode-se notar a implementação do destrutor da classe. Sempre que um objeto é destruído, o atributo estático `nObjetos` é decrementado em uma unidade. Desta forma, é possível ter o controle de quantos objetos se encontram instanciados em um certo momento. O método público `getNObjetos()`, linhas 14-17, permite que o valor de `nObjetos` seja recuperado durante a execução do programa.

```
7  // construtor da classe
8  ObjetosInstanciados::ObjetosInstanciados()
9  {
10     nObjetos++;     // incrementa o atributo
11 }
```

```cpp
// Destrutor da classe
ObjetosInstanciados::~ObjetosInstanciados()
{
    nObjetos--;     // decrementa o atributo
}
```

```cpp
// Metodo get que retorna o número de objetos
int ObjetosInstanciados::getNObjetos() const
{
    return nObjetos;
}
```

O arquivo `mainObjetosInstanciados.cpp` representa um exemplo de utilização da classe `ObjetosInstanciados`. Nas linhas 15 e 18 são criados os objetos 1 e 2. Na linha 18, em específico, `objeto2` é um ponteiro que aponta para um endereço de memória em que um objeto instanciado da classe `ObjetosInstanciados` se encontra alocado. Durante a disciplina de POO, os temas ponteiros e alocação dinâmica de memória foram anteriormente abordados.

```cpp
    ObjetosInstanciados objeto1;    // instanciação de objeto

    // instanciação de objeto: declaração de ponteiro para objeto
    ObjetosInstanciados * objeto2 = new ObjetosInstanciados();
```

O `objeto3` é criado em um bloco de comandos (repare nas chaves das linhas 25 e 32), portanto, ele somente existirá no interior deste bloco, o que faz com que na linha 35 este objeto já tenha sido destruído. É por essa razão que a execução da linha 35 do código exibe que existem apenas dois objetos da classe `ObjetosInstanciados` ativos na execução do programa. Na linha 37, o objeto apontado pelo ponteiro `objeto2` é destruído a partir do comando `delete`. Desta forma, a linha 40 imprime em tela a informação de que apenas um objeto da classe `ObjetosInstanciados` encontra-se instanciado.

```cpp
    {
        cout << "\n ENTRADA DE BLOCO DE COMANDOS  " << endl;

        ObjetosInstanciados objeto3;    // instanciação de objeto local a

        cout << " Numero de objetos: " << objeto2->getNObjetos() << endl;
        cout << " Numero de objetos: " << objeto3.getNObjetos() << endl;
    }

    cout << "\n SAIDA DE BLOCO DE COMANDOS  " << endl;
    cout << " Numero de objetos: " << objeto1.getNObjetos() << endl;

    delete objeto2;     // removendo alocaco de ponteiro (destruicao de o

    cout << "\n DELETE OBJETO 2  " << endl;
    cout << " Numero de objetos: " << objeto1.getNObjetos() << endl;
```

Pense um pouco: o que aconteceria se o atributo `nObjetos` da classe `ObjetosInstanciados` fosse declarado como automático (sem a especificação `static`, todos os atributos sem esta especificação são ditos automáticos)? Neste caso, no momento de criação de cada um dos objetos, o construtor incrementaria o atributo `nObjetos` de cada uma das

instâncias (cada objeto possuiria o seu próprio atributo `nObjetos`) e a saída em tela seria igual a "1" para cada uma das impressões do retorno de chamada do método `getNObjetos()`. Aproveite que o código foi disponibilizado e efetue este teste!! Para isso, remova o termo `static` da frente do atributo `nObjetos` no arquivo de definição da classe `ObjetosInstanciados.hpp` e comente a linha de vinculação do atributo estático à classe no arquivo de implementação `ObjetosInstanciados.cpp` (linha 5).

1.1 **Atributos Estáticos Públicos**

No exemplo anterior, o atributo estático `nObjetos` foi declarado como `private`. Assim, ele não podia ser acessado diretamente a partir do objeto, encontrando-se encapsulado. Entretanto, também é possível definir um atributo `static` como `public`. A classe `Moeda`, disponibilizada juntamente com este documento, apresenta um exemplo de definição e utilização de atributo público estático. Esta classe converte um determinado valor em real para o equivalente em dólar. Sugere-se que os arquivos desta classe sejam abertos e acompanhados para um melhor entendimento das explicações a seguir.

No arquivo de definição, `Moeda.h`, nota-se a existência de dois construtores (um sem parâmetros na linha 12 e outro com um parâmetro do tipo `float` na linha 13), um método público `toDolar()`, que irá converter o valor do atributo privado `real` para um valor em dólar, e um atributo estático público denominado `dolar`. Este atributo é utilizado para armazenar na classe a cotação do dólar comercial.

```
 7  class Moeda
 8  {
 9      public:
10          static float dolar;
11
12          Moeda();
13          Moeda(float r);
14
15          float toDolar();
16
17      private:
18          float real;
19  };
```

Este é um bom momento para tecer novamente comentários sobre os métodos construtores. Estes métodos podem ser considerados especiais. Eles são chamados quando um objeto é instanciado. Na definição de uma classe, os métodos construtores possuem o mesmo nome da classe, podendo ter ou não parâmetros de entrada. Observem que as linhas 12 e 13 são responsáveis por definir dois construtores para a classe `Moeda`. Um fato importante é que os construtores não possuem tipo de retorno, nem mesmo o tipo `void` deve ser utilizado para indicar este fato.

Mais de um método construtor pode ser definido para uma classe. Todos eles possuem o mesmo nome, o nome da classe. Se eles possuem o mesmo nome, o que irá diferenciá-los é a assinatura dos métodos construtores, ou seja, os seus parâmetros: o construtor da linha 12 não possui parâmetros; o construtor da linha 12 possui um parâmetro do tipo `float`.

No arquivo de implementação da classe `Moeda`, `Moeda.cpp`, o construtor sem parâmetros (linhas 12-16) solicita que o usuário digite um valor monetário em real que será atribuído ao atributo privado `real`. O construtor das linhas 19-22 recebe como parâmetro um valor em real e o atribui para o membro de dados `real`. O método `toDolar()` retorna para o ponto de sua chamada o valor convertido do atributo privado `real` para o seu equivalente em dólar.

```
11  // construtor sem parâmetros
12  Moeda::Moeda()
13  {
14      cout << "\n -> Digite valor em real (R$): ";
15      cin >> real;
16  }
```

```
18  // construtor: recebe valor em real
19  Moeda::Moeda(float r)
20  {
21      real = r;
22  }
```

```
24  // converte valor de atributo para dolar
25  float Moeda::toDolar()
26  {
27      return (real / dolar);
28  }
```

No arquivo `mainMoeda.cpp`, a classe `Moeda` é utilizada em um exemplo de programa. Na linha 12 são instanciados três objetos: `valorA`, `valorB` e `valorC`. Na criação do objeto `valorA`, o construtor sem parâmetros é chamado. Em contrapartida, na criação dos objetos `valorB` e `valorC`, é o construtor com parâmetro do tipo `float` que é chamado.

```
12      Moeda valorA, valorB(450), valorC(100);   // declaração de objetos
```

Na linha 14, o valor comercial do dólar em reais é atribuído ao membro de dado estático público `dolar`. Isto é efetuado a partir do objeto `valorA`, mas é importante comentar que qualquer um dos outros objetos poderia ser utilizado para efetuar esta operação e o resultado para o programa seria o mesmo. Isto torna-se claro quando lembramos que o atributo estático `dolar` é único e acessível a todos os objetos instanciados da classe `Moeda`.

```
14      valorA.dolar = 5.64;
```

Nas linhas 20-22, cada um dos objetos instanciados realiza a sua chamada para o método `toDolar()`. Estas ações fornecem a impressão em tela do valor do atributo real `private` de cada um dos objetos para um valor correspondente em dólar. O valor considerado para a cotação do dólar é o mesmo para todos os objetos e encontra-se armazenado no atributo estático público `dolar` da classe `Moeda`.

```
20      cout << "\n * Valor A em dolar: US$ " << fixed << valorA.toDolar()
21           << "\n * Valor B em dolar: US$ " << valorB.toDolar()
22           << "\n * Valor C em dolar: US$ " << valorC.toDolar() << "\n\n";
```

Observe que na linha 14 do arquivo `mainMoeda.cpp`, o acesso ao atributo estático público `dolar` é realizado a partir do objeto, por meio do ".". Neste caso, o uso do "." não é considerada uma boa prática, pois dá a entender que apenas o atributo `dolar` do `objetoA` está sendo alterado, quando na realidade o que está sendo alterado é o valor do atributo estático `dolar` que é único na classe e pode ser acessado por todos os objetos instanciados.

A sintaxe mais recomendada para ter acesso a um atributo estático público é a utilizada nas linhas de código 10 e 24. Observem que antes do nome do atributo estático público, é utilizado o nome da classe seguido do operador " : : ". É interessante comentar que, a partir desta sintaxe, o atributo pode ser acessado mesmo que nenhum objeto da classe `Moeda` tenha sido ainda instanciado (linha 10). Esta sintaxe deixa claro que o atributo está associado à classe e não aos seus objetos.

```
10      Moeda::dolar =  5.00;
```

```
24      Moeda::dolar = 5.50;
```

Um atributo estático é criado na definição da classe e existirá independente da existência de objetos. Pode-se afirmar, portanto, que um atributo estático corresponde a um dado da classe, que é acessível aos seus objetos. Outro ponto relevante, é que se deve ter **cuidado** ao **inicializar** atributos estáticos em um construtor, uma vez que os construtores são chamados toda vez que um objeto da classe é instanciado. Portanto, inicializar um atributo estático em construtores fará com que este atributo seja inicializado sempre que um objeto for instanciado, "reiniciando" o seu valor.

2. Métodos Estáticos

Os métodos `static` possuem comportamentos semelhantes aos apresentados pelos atributos `static`. Eles são utilizados para implementar recursos que são comuns a todos os objetos instanciados de uma classe. Estes métodos poderão ser utilizados independentemente da declaração de qualquer objeto.

Em conjunto com este documento foi disponibilizada a classe `Moeda1`. Esta classe é uma modificação da classe anterior, `Moeda`. No arquivo de definição, `Moeda1.h`, pode-se notar que o

atributo estático `dolar` passou a ser do tipo `private` e foi definido um método estático público, denominado `valorDolar()`, que solicita do usuário a cotação atual do dólar comercial.

```
6  // Definicao da classe moeda
7  class Moeda1
8  {
9      public:
10
11         Moeda1();
12         Moeda1(float r);
13
14         float toDolar();
15
16         static void valorDolar();
17
18     private:
19         float real;
20
21         static float dolar;
22 };
```

A implementação deste método é bastante simples e pode ser verificada no arquivo de implementação da classe, `Moeda1.cpp`, nas linhas 30 a 31. Ele solicita ao usuário o valor do dólar em reais e o atribui ao membro estático e `private` de dados `dolar`. Como este atributo é estático, não há nenhum problema em modificar o seu valor, pois este atributo também existe independentemente da instanciação de algum objeto da classe.

```
30 /*static*/ void Moeda1::valorDolar()
31 {
32     cout << "\n -> Digite o valor do dolar em reais (R$): ";
33     cin >> dolar;
34 }
```

O arquivo `mainMoeda1.cpp` faz uso da classe `Moeda1.cpp` em um programa. Em sua linha 10 é feita a chamada do método estático público `valorDolar()`. Observe o nome da classe seguido do operador "`::`" para fazer acesso a este método estático público. Não é necessário que um objeto tenha sido instanciado para que o método estático possa ser chamado.

```
10      Moeda1::valorDolar();
```

Pense um pouco: se o método `valorDolar()` chamado na linha 10 tentasse modificar o valor de um atributo automático (não estático), o que aconteceria? Lembre-se que um atributo automático é individualizado para cada objeto. Portanto, ele só existe se um objeto for instanciado. Até a linha 10, ainda não existem objetos instanciados da classe `Moeda1`. Por essa razão, os **métodos estáticos públicos só devem acessar atributos `static` da classe**.

Na linha 24, é realizada a chamada do método `valorDolar()` a partir de um objeto e do uso de "`.`". Já foi comentado que esta sintaxe não é recomendada para atributos estáticos, o mesmo ocorre para a chamada de métodos estáticos. Entretanto, ela foi aqui utilizada apenas para

demonstrar que a chamada por meio do objeto `valorB` ocasiona alteração no resultado da chamada realizada do método `toDolar()` que é efetuada a partir do objeto `valorA`. Isso demonstra que o atributo estático privado `dolar` que é modificado pela chamada do método `valorDolar()` é único para todos os objetos.

```
24      valorB.valorDolar();
```

REFERÊNCIAS

DEITEL, H. M.; DEITEL, P. J. **C++: como programar**. 5ª ed. São Paulo: Pearson Prentice Hll, 2006.

STROUSTRUP, Bjarne. The C++ programming language. Pearsing Education India, 2000.

MIZRAHI, Victorine Viviane. **Treinamento em Linguagem C++ Modulo 2**. 2a. Edição. 2005.