Métodos e Objetos **const**, Composição, Ponteiro *this*, Membros *static*, Classes Proxy

Prof. Valdir Pedrinho de Tomin Junior, Eng.

Universidade Federal de Santa Catarina
Centro Tecnológico de Joinville
EMB5631 – Programação III

e-mail: valdir.pedrinho@ufsc.br

27 de agosto de 2018

Métodos e Objetos **const**

Atributos **const**

Alguns objetos precisam ser modificáveis e alguns não. O programador pode utilizar a palavra-chave **const** para especificar que um objeto não é modificável e que qualquer tentativa de modificá-lo deve resultar em um erro de compilação.

Métodos **const**

Os compiladores C++ não permitem chamadas de métodos para objetos *const* a menos que as próprios métodos também sejam declaradas como *const*. Isso é verdadeiro mesmo para métodos *get* que não modificam o objeto. Além disso, o compilador não permite que métodos declarados como *const* modifiquem o objeto.

Métodos e Objetos **const**

Sintaxe

- Funções / Métodos *const*:

```
tipo nome_da_função(parâmetros) const;
```

- Atributos / objetos *const*:

```
const tipo objeto(parâmetros para o construtor);
```

- Ponteiros constantes:

```
tipo * const ponteiro;
```

Exemplos

10.1-3, 10.4-6 e 10.7-9 do livro *DEITEL. C++, Como programar. 5ª ed.*

Composição

Objetos como membros de classes

Quando um objeto é criado, seu construtor é chamado automaticamente. O construtor de um objeto pode passar argumentos para construtores de objeto-membro, o que é realizado via inicializadores de membro. Os objetos-membro são construídos na ordem em que são declarados na definição de classe (não na ordem em que são listados na lista de inicializadores de membro do construtor) e antes de os objetos da sua classe **contêiner**(*host*) serem construídos.

Exemplo

10.10-14 do livro *DEITEL. C++, Como programar. 5ª ed.*

Ponteiro **this**

O que é o ponteiro **this**?

Vimos que as funções-membro de um objeto podem manipular os dados do objeto. Como as funções-membro sabem quais membros de dados do objeto devem manipular? Cada objeto tem acesso ao seu próprio endereço por um ponteiro chamado **this** (uma palavra-chave do C++). O ponteiro *this* de um objeto não faz parte do objeto em si — isto é, o tamanho da memória ocupado pelo ponteiro *this* não é refletido no resultado de uma operação *sizeof* no objeto. Em vez disso, o ponteiro *this* é passado (pelo compilador) como um argumento implícito para cada uma das funções-membro não-**static** do objeto.

Ponteiro **this**

Tipo do ponteiro **this**

O tipo do ponteiro *this* depende do tipo do objeto e do fato do método em que *this* é usado ser ou não declarada como *const*.

- Em um método não constante, o *this* será:

```
classe * const
```

- Em um método constante, o *this* será:

```
const classe * const
```

Exemplo

10.17 do livro *DEITEL. C++, Como programar. 5ª ed.*

Ponteiro **this**

Chamadas de métodos em cascata

Outra utilização do ponteiro *this* é permitir chamadas de métodos em cascata, nos quais múltiplas funções são invocadas na mesma instrução. Cada método retorna uma referência a um objeto da classe – da qual este faz parte do escopo – para permitir chamadas dos métodos em cascata.

Exemplo

10.18-20 do livro *DEITEL. C++, Como programar. 5ª ed.*

Membros **static**

Para que servem membros **static**?

Há uma exceção importante à regra que diz que cada objeto de uma classe tem sua própria cópia de todos os membros de dados da classe. Em certos casos, apenas uma cópia de uma variável deve ser compartilhada por todos os objetos de uma classe. Um membro de dados **static** é utilizado por essas e outras razões.

Principais Características

- Variáveis *static* representam informações no 'nível da classe' (isto é, uma propriedade da classe compartilhada por todas as instâncias, não uma propriedade de um objeto específico da classe);
- Embora possam parecer variáveis globais, os membros de dados de uma classe *static* têm escopo de classe;
- Os membros *static* podem ser declarados *public*, *private* ou *protected*;

Membros **static**

Principais Características

- Um membro *static* existe mesmo que nenhum objeto da classe tenha sido instanciado;
- Para acessar um membro de classe *public static* quando não existe nenhum objeto da classe, simplesmente prefixe o nome de classe e o operador de resolução de escopo binário (::) com o nome do membro de dados;
- Membros *static* devem ser definidos no escopo do *namespace* global (isto é, fora do corpo da definição de classe) e inicializados somente nessas definições, uma única vez.

Exemplo

10.21-23 do livro *DEITEL. C++, Como programar. 5ª ed.*

Classes *Proxy*

Por que utilizar?

Classes *Proxy* permitem ocultar informações proprietárias aos clientes da classe. O arquivo de implementação da classe é fornecido para o cliente como um arquivo de código-objeto pré-compilado.

Como implementar?

- No arquivo de cabeçalho da classe *proxy* utiliza-se uma **declaração de classe antecipada [forward class declaration]**;
- No arquivo de implementação da classe a ser protegida é feita a inclusão do cabeçalho para a classe *proxy*;
- No arquivo principal o cliente irá incluir apenas o cabeçalho para a classe *proxy*.

Exemplo

10.24-27 do livro *DEITEL. C++, Como programar. 5ª ed.*