Universidade Federal de Juiz de Fora

PET Elétrica

Introdução à orientação a objetos

Tutor: Francisco José Gomes

Aluno: João Tito Almeida Vianna

18/05/2013

1 Programação Estruturada x Orientação a objetos

1.1 Programação Estruturada

Paradigma de programação que busca a obtenção de uma estrutura lógica clara; boa tanto para compreensão do código, como para manutenção do mesmo. Normalmente essa estrutura é interpretada linha por linha, havendo três tipos de estruturas básicas para navegar pelo código: sequência, seleção e repetição; como ilustrado na Figura 1. Os matemáticos Bohm, C. e Jacopini, G. provaram que qualquer programa pode ser escrito utilizando-se essas três estruturas básicas.

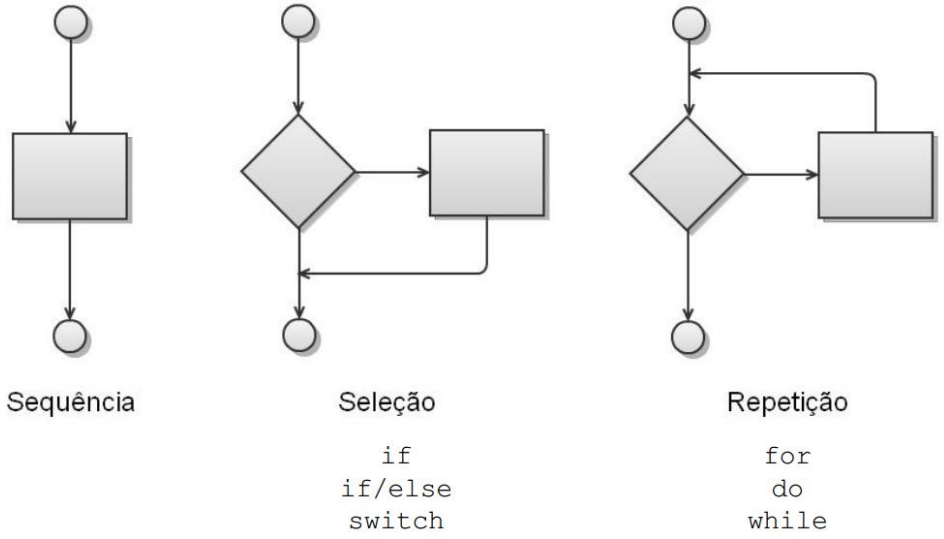

Figura 1 - Programação estruturada [1]

Além disso, este tipo de programação usa como unidade básica a "função". Define-se como função uma sequência de comandos que é executada ao longo do programa várias vezes. Tal sequência é definida em um modulo separado e chamada de dentro da estrutura principal do programa, ou até de outros programas. Além de tornar o programa mais organizado, a utilização de funções otimiza a utilização de memória pelo código.

A programação não-estruturada, com uso do comando "goto", foi caindo em desuso com o advento da estruturada e muito impulsionada pelo artigo publicado por Dijkstra [3]. A estrutura do programa fica comprometida em sua compreensão e manutenção. Basicamente falando, ao "debugar-se" um código, a indicação de erro na linha X não acarreta necessariamente que as linhas acima estão corretas e que o erro encontra-se desta linha em diante. Com a estrutura de "Goto" a busca do erro torna-se desnecessariamente complexa.

1.2 Orientação a objetos

Paradigma de programação que introduz a noção de "classe" e "objeto". Classes são estruturas de dados que contêm tanto campos de dados (variáveis próprias) como seus próprios métodos (funções). Um objeto é uma instância de uma classe, declarada dentro do programa (como uma váriável do tipo "*int*" é declarada, um objeto de qualquer classe criada pelo programador também o deve ser). O surgimento desta estrutura está associado à facilitação da escrita de programas de computador, tornando o processo mais intuitivo.

Um programa orientado a objetos deve ser visto como um conjunto de objetos capazes de interagir entre si. Cada objeto é capaz de receber processar e enviar dados, podendo ser visto como uma "máquina independente".

Essa forma de programação induz o programador a manter o acesso aos dados somente quando a ação é associada a tal objeto. No caso da programação estruturada, na medida em que o programa cresce de tamanho a tendência é o surgimento de variáveis globais, as quais podem ser acessadas e modificadas de qualquer parte do código, o que tende a disseminar *bugs*.

Uma classe pode ser vista como um conjunto de funções (operações) que encapsulam um conjunto de variáveis (atributos), visando garantir o uso correto das mesmas; como ilustrado na Figura 2.

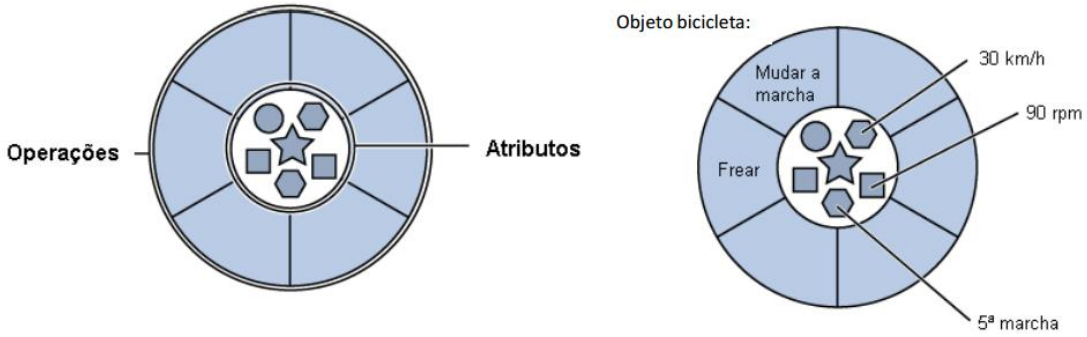

Figura 2 - Representação de objeto [1].

1.3 Visão geral

Para finalização do tópico introdutório, apresenta-se a

Tabela 1, da qual pode-se tirar uma visão básica da diferença entre os dois paradigmas de programação apresentados.

Enquanto na programação estruturada temos como unidade básica uma função, a qual pode ser chamada diversas vezes dentro do código e o enfoque é dado às ações executadas por cada comando; temos na programação estruturada a função básica sendo a classe, ou seja, a definição de um tipo de objeto, e o enfoque da programação gira em torno da ideia de objeto apresentada anteriormente.

Tabela 1: Comparação entre programação estruturada e orientada a objetos [1].

	Programação estruturada	Programação Orientada a Objetos
Enfoque	Objetos	Ações
Unidade básica	Classe	Função

2 Orientação a objetos

2.1 Classe

Para desenvolvimento de um raciocínio é interessante pensar nas *structs* que podem ser criadas em programações estruturadas. A *struct* nada mais é do que a definição de um novo tipo de variável, dentro da qual encontram-se algumas outras variáveis. Abaixo encontra-se um exemplo de estrutura criada na linguagem C.

```
typedef struct aluno
{
        char nome[50];
        int numeroDeMatricula;
        int dataDeNascimento[3];
        char curso[30];
} aluno;
```

A definição de classe pode ser vista como uma *struct* com um nível a mais de complexidade. A classe é uma *struct* que, além de definir suas próprias variáveis, define as funções que lidarão com as mesmas. Abaixo um exemplo de classe criada na linguagem Java.

```
public class Aluno
{
        private String nome;
        private int numeroDeMatricula;
        private int[] dataDeNascimento = new int[3];
        private String curso;

        public Aluno(String N, int Mat, int[] Nasc, String cur)
        {
                nome = N;
                numeroDeMatricula = Mat;
                dataDeNascimento = Nasc;
                curso = cur;
        }
        public void setNumeroMatricula(int novo)
        {
                numeroDeMatricula = novo
        }
        public void setCurso(String cur)
        {
                curso = cur;
        }

        public String getNome()
        {
                return nome;
        }
}
```

Normalmente as variáveis de uma classe são denominadas *atributos* e suas funções, *métodos*. Além dos atributos e métodos, toda classe possui um *construtor*, que é chamado na criação de um objeto e garante a inicialização correta do mesmo. O *construtor* possui obrigatoriamente o mesmo nome da classe. A ordem de definição dos componentes de uma classe é livre.

É possível criar mais de um construtor para a classe, ou método com o mesmo nome. Para tal é necessário que os parâmetros passados para o referido método devem ser diferentes.

Observando o código acima com cuidado, percebe-se a presença de uma outra declaração antes do tipo de variável ou função. Em orientação a objetos, os atributos e métodos de uma determinada classe podem receber três *especificadores de acesso a membro:*

- **public:** acessível de qualquer parte do programa.
- **private:** acessível somente através das funções da própria classe.
- **protected:** acessível à classe e também a suas subclasses (tópico, que será discutido mais a frente).

2.1.1 Encapsulamento

Encapsular é proteger os atributos de uma classe, tornando-os acessíveis somente através de seus métodos. Esta é uma prática bastante difundida entre os programas orientados a objeto, pois a responsabilidade de modificar os atributos da classe é dela mesma, não de quem deseja realizar a ação executada pelo método. A Figura 2 ilustra o encapsulamento.

Conhecendo os especificadores de acesso a membro, já podemos dizer que para realização do encapsulamento em uma classe, os atributos definidos devem ser do tipo private e os métodos do tipo public.

Um padrão muito difundido é a criação de funções "*getVariavel*" e "*setVariavel*", as quais retornam o valor contido na variável e mudam o valor da mesma, respectivamente. Esta fica sendo então a única maneira de acessar os atributos de uma classe.

2.2 Herança

A herança permite definir uma classe como a extensão de uma outra. Esta propriedade permite o aproveitamento de código, evitando duplicação; além de facilitar a manutenção do mesmo.

Um exemplo pode ser dado a partir da análise de contas de banco. Todas as contas de um banco possuem um saldo, podem receber depósitos, realizar saques,... Ainda assim existem diferentes tipos de contas, como conta corrente e a poupança. Na poupança, por exemplo, deve ser incluído um rendimento mensal sobre o saldo presente nela.

Este é um caso claro em que a herança pode e deve ser utilizada. A criação de uma série de métodos comuns às contas na classe "Conta" é sucedida pela criação de diversas classes que herdam toda a sua estrutura (atributos e métodos). Além disso, os métodos criados dentro da *subclasse* (ContaCorrente, Poupança, ...) podem sobreescrever os da *superclasse* (Conta), devendo para isso ter o mesmo nome.

2.3 Polimorfismo

Denomina-se polimorfismo o fato de uma mesma chamada de método poder invocar métodos diferentes. Este conceito acaba sendo uma consequência natural do uso da herança. Como explicado no item anterior, ao sobreescrever-se um determinado método de uma superclasse, consequentemente abre-se a possibilidade do polimorfismo.

3 Bibliografia

[1] Reis, G. R., Material do minicurso: "*C++ e Programação Orientada a Objetos*", ministrado na XXXIII Semana da Engenharia, 2010.

[2] Bohm, Corrado; and Giuseppe Jacopini (May 1966). "*Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules*". *Communications of the ACM* **9** (5): 366–371.

[3] Dijkstra, E. "*Go-to statement considered harmful*", em Commun. ACM 11 (1968), 3: 147–148. (http://www.cs.utexas.edu/~EWD/ewd02xx/EWD215.PDF)