Unidade 4: Métodos e Funções

Prof. Daniel Caetano

Objetivo: Introduzir o conceito de métodos e funções.

Bibliografia: DEITEL, 2005; CAELUM, 2010; HOFF, 1996.

INTRODUÇÃO

Nas aulas anteriores foram observadas as estruturas fundamentais de seleção e repetição da linguagem Java. Tais estruturas, juntamente com o as variáveis, permitem elaborar algoritmos para praticamente qualquer finalidade.

Entretanto, para que os softwares possam ser desenvolvidos de maneira organizada e seu código fique mais compreensível, é interessante organizar o código em *blocos funcionais*, isto é, blocos que realizam tarefas específicas.

Estes blocos são chamaos de **métodos** ou, mais genericamente, de **funções**. Um método é um bloco de instruções que executam uma tarefa específica. Cada método possui um nome específico, de maneira que sempre que se desejar executar aquela sequência de instruções, bastará usar o nome do método.

Nesta aula serão apresentados os conceitos envolvidos na criação de um método, bem como alguns exemplos.

1. O QUE É UM MÉTODO

Antes de mais nada, é importante ressaltar o **que é** e **para que serve** um método. Um método é um trecho de código que faz alguma tarefa bem definida e que, usualmente, pode ser reutilizada em diversas partes do software.

Os métodos só existem dentro das classes (o objetos, como será visto depois), não sendo possível criar um método sem uma classe. Assim, pode-se dizer que um método é a descrição de algo que uma classe (ou objeto) pode fazer.

E para que serve o método? Bem, o método serve para ser executado quando necssário, o que pode ser feito através de seu nome. Assim, é quase sempre obrigatório que um método tenha um nome.

2. DECLARANDO UM MÉTODO

Como já foi dito, os métodos são sempre definidos **dentro** de uma classe. Assim, uma declaração de método fora dos demarcadores de classe { } causam erro durante a compilação. A declaração de um método é, de maneira geral, feita da seguinte forma:

```
[escopo] <tipoDeRetorno> <nomeDoMetodo> ( [tipoDaVar <nomeDaVar>] [, ...] ) {
  // O código do método vai aqui
  }
```

Por exemplo:

```
public int multiplicaPorDois (int valor) {
  // ... operações executadas pelo método entram aqui
  }
```

O campo **tipoDeRetorno** indica o que esse método "responde" quando ele for executado, ou seja, que tipo de dado ele *retorna* a quem o executou. No caso do exemplo, o método "responde" um número inteiro (int). Pode ser qualquer tipo válido, um nome de classe ou até mesmo o tipo **void**, que indica "nenhum retorno".

O **nomeDoMetodo** deve ser um nome curto e, ao mesmo tempo, descritivo do que o método faz, qual a função que ele executa. O nome do exemplo não é muito bom: algo como multPor2 seria algo melhor, embora de leitura um pouco mais complexa.

O **tipoDaVar** dentro dos parênteses indica o tipo do dado **nomeDaVar**, que é um **parâmetro** do método, ou seja, um valor que o método vai receber para que ele possa fazer seu processamento e dar uma resposta.

Agora, vejamos o campo que ficou sem dicussão: o **escopo**. O escopo define quem pode executar esse método. Alguns dos valores possíveis só ficarão mais claros quando estivermos falando sobre orientação a objetos, mais adiante no curso.

Escopo	Significado
public	método pode ser chamado a partir de qualquer parte do programa
private	método só pode ser chamado por membros da classe
---	
final	método não pode ser modificado em classes especializadas
protected	método só pode ser chamado por membros da classe ou descendentes
static	método é da classe, e não do objeto.

Existem outras definições de escopo, mas essas são as mais comuns. Os usos dos escopos ficam claros pelo seu significado na tabela acima, a não ser pelo escopo **static**. A função desta definição de escopo é melhor detalhada abaixo.

Como já vimos rapidamente, uma "cópia" de uma classe, com características específicas, pode ser criada. Esta "cópia" ou "instância" é chamada **objeto**. Quando criamos uma instância de uma classe, tudo ocorre como se cada objeto passasse a ter o seu próprio conjunto de métodos.

Quando queremos que apenas uma "cópia" do método exista (podendo ser compartilhada por todos os objetos, inclusive), dizemos que o método é **da classe**. Para indicar que um método é **da classe** e não dos objetos, usa-se o escopo do tipo **static**.

Métodos do tipo **static** podem ser executados **sem** a criação e um objeto de uma dada classe. Métodos que não tenham sido declarados como static não podem ser executados sem a criação de pelo menos um objeto da classe.

2.1. Retornando um Valor

Quando o método precisar devolver algum valor para quem solicitou sua execução, deve ser usada a instrução **return** para isso, conforme o exempo abaixo:

```
public int multiplicaPorDois (int valor) {
   int resultado;
   // ... operações executadas entram aqui, calculando o valor em resutlado
   return resultado;
   }
```

2.2. Escopo de Variáveis

Até o momento tínhamos apenas uma classe com um método e, como sempre declarávamos nossas variáveis no nosso único método, não havia nenhum tipo de complicação.

A partir de agora, entretanto, temos mais de um método e é possível perceber um comportamento bastante peculiar: as variáveis de um método só são válidas dentro daquele método. Assim, não é possível que um método acesse diretamente variáveis de outro método. Tenha isso em mente quando estiver programando.

3. CHAMANDO UM MÉTODO

Os métodos normais, não static, são executados da seguinte forma:

nomeDoObjeto.nomeDoMetodo();

Os métodos static são executados da seguinte forma:

NomeDaClasse.nomeDoMetodo();

É importante ressaltar que apenas métodos públicos (**public**) de uma classe ficam disponíveis para outras classes. Em nossos programas estamos ainda trabalhando com apenas uma classe. Futuramente, quando trabalharmos com mais classes, este fato será importante.

A razão para tornar um método privativo (**private**) também será discutida no futuro.

4. IMPLEMENTANDO CÓDIGO

Vamos realizar um programa que imprima os fatoriais de 1 a 10, de maneira que exista um método para calcular o fatorial. Como estamos trabalhando apenas com a classe, este método deve ter escopo do tipo **public static**.

O código está apresentado a seguir:

MetodoFatorial.java

```
class MetodoFatorial {

   public static long fatorial(int numero) {
      long resultado;
      int i;
      resultado = 1;
      for (i = numero; i > 0; i = i - 1) {
         resultado = resultado * i;
         }
      return resultado;
      }

   public static void main(String[] args) {
      long resultado;
      int i;

      for (i = 1; i <= 10; i = i + 1) {
         resultado = fatorial(i);
         System.out.println( i + "! = " + resultado);
         }
      }
   }
```

5. EXERCÍCIOS

A) Crie um método que calcule a potência de um número. O método deve ter a seguinte apresentação:

public static int potencia(short base, short exp)

E a operação a ser realizada é:

$base^{exp}$

B) Use esse método para calcular as potências de 2^1 a 2^{16}

C) Crie um método que receba as notas AV1, AV2, AV3 e FREQ e responda 0 se o aluno está reprovado e 1 se está aprovado.

D) Use o método acima para testar a aprovação dos alunos abaixo:

	AV1	AV2	AV3	FREQ
Aluno 1:	6,0	6,5	0,0	80
Aluno 2:	5,0	5,0	7,0	75
Aluno 3:	3,0	10,0	3,5	90
Aluno 4:	10,0	3,0	10,0	95
Aluno 5:	10,0	10,0	10,0	70

O resultado apresentado pelo software deve ser:

Aluno 1: Aprovado
Aluno 2: Aprovado
Aluno 3: Reprovado
Aluno 4: Reprovado
Aluno 5: Reprovado

6. BIBLIOGRAFIA

CAELUM, FJ-11: Java e Orientação a Objetos. Acessado em: 10/01/2010. Disponível em: < http://www.caelum.com.br/curso/fj-11-java-orientacao-objetos/ >

DEITEL, H.M; DEITEL, P.J. **Java: como programar** - Sexta edição. São Paulo: Pearson-Prentice Hall, 2005.

HOFF, A; SHAIO, S; STARBUCK, O. **Ligado em Java**. São Paulo: Makron Books, 1996.