Unidade 8: Sobrecarga de Funções

e Vetores e Matrizes (Arrays)

Prof. Daniel Caetano

Objetivo: Uso de sobrecarga de funções para criação de código intuitivo e uso de arrays para melhorar a codificação.

INTRODUÇÃO

Agora que a linguagem Java já é conhecida em seus aspectos mais básicos, serão apresentados alguns aspectos um pouco mais avançados da linguagem. Estes aspectos, apesar de serem mais complexos conceitualmente, são de uso bastante simples e podem facilitar em muito a vida do programador.

1. SOBRECARGA DE FUNÇÕES E MÉTODOS

A grande maioria dos métodos que criamos precisam de algum tipo de parâmetro, isto é, dados de entrada para que o método possa fazer seu trabalho. São as notas de um aluno para o qual se quer calcular a média, o salário de um funcionário para o qual se quer calcular os descontos, o nome do cliente para que ele seja cadastrado... enfim, em geral, os métodos precisam de parâmetros para trabalharem. Por exemplo:

```
public static void imprime(String texto) {
   JOptionPane.showMessageDialog(null,texto,"Titulo",JOptionPane.PLAIN_MESSAGE);
   }
```

Neste código, o método imprime precisa do parâmetro "texto", que é uma String qualquer, para realizar seu trabalho.

Na linguagem Java, quando definimos um ou mais parâmetros na assinatura e um método, assinamos um contrato: o método será sempre chamado com aquele determinado número de parâmetros, que devem ser todos do tipo correto. Por exemplo, o método com a assinatura abaixo:

public static void imprime(String texto)

Ele deve ser executado, no programa, da seguinte forma:

imprime("Algum texto qualquer"); <=== Correto!

Ou seja, só podemos ter um parâmetro e ele **tem** que ser uma String. Qualquer tentativa de fazer diferente, como as indicadas abaixo, irão causar erros diversos:

```
imprime("Um texto", "Outro texto");          <=== Errado! Dois parâmetros!
imprime(1);                                  <=== Errado! Parâmetro int!
```

Entretanto, vez ou outra queremos que um mesmo método possa ser chamado com diferentes parâmetros ou diferentes números de parâmetros. Por exemplo, poderíamos querer o método imprime funcionasse em todos os casos exemplificados anteriormente.

Como conseguir isso?

É fácil! Basta definir o método várias vezes, indicando parâmetros diferentes:

```
public static void imprime(String texto) {
   JOptionPane.showMessageDialog(null,texto,"Titulo",JOptionPane.PLAIN_MESSAGE);
   }

public static void imprime(String texto, String titulo) {
   JOptionPane.showMessageDialog(null,texto,titulo,JOptionPane.PLAIN_MESSAGE);
   }

public static void imprime(int numero) {
   String texto = "Número " + numero;
   JOptionPane.showMessageDialog(null,texto,"Titulo",JOptionPane.PLAIN_MESSAGE);
   }
```

Ou seja: em Java, nós **podemos** ter mais de um método com o mesmo nome, na mesma parte do programa, desde que:

a) O número de parâmetros seja diferente

E/OU

b) O tipo dos parâmetros seja diferente

O ato de criar múltiplos métodos com o mesmo nome e parâmetros diferentes recebe o nome de **sobrecarregar funções** (ou métodos).

Esse recurso é **muito** usado e é **muito útil**.

1.1. Otimizando um Pouco o Código

É claro que o código apresentado anteriormente é a versão mais simples. Entretanto, existe um grande número de repetições do código do método (JOptionPane....). Para evitar isso, selecionamos o método mais genérico deles (aquele com dois parâmetros) para ser o principal, como indicado no exemplo a seguir.

```
public static void imprime(String texto, String titulo) {
   JOptionPane.showMessageDialog(null,texto,titulo,JOptionPane.PLAIN_MESSAGE);
   }

public static void imprime(String texto) {
   // ... Código Aqui
   }


public static void imprime(int numero) {
   // ... Código Aqui
   }
```

Agora, reescreveremos as outras duas versões do método para usar a versão principal, eliminando a repetição de chamadas a JOptionPane...

```
public static void imprime(String texto, String titulo) {
   JOptionPane.showMessageDialog(null,texto,titulo,JOptionPane.PLAIN_MESSAGE);
   }

public static void imprime(String texto) {
   imprime(texto,"Título");
   }

public static void imprime(int numero) {
   String texto = "Número: " + numero;
   imprime(texto);
   }
```

Observe como essa representação é mais compacta e, de quebra, ela facilita a manutenção do código, já que mudanças na forma de exibição precisam ser feitas **apenas** no método principal, que é o que chama o JOptionPane.showMessageDialog.

2. EXERCÍCIOS

Para cada exercício, crie o método solicitado e modifique o método **main** para demonstrar o uso.

A) (1,0 pontos) Crie um método para imprimir uma mensagem em uma janela, com a seguinte assinatura:

public static void imprime(String texto, String titulo)

B) (1,0 pontos) Crie outro método que imprima um número do tipo **double** em uma janela, sempre com o título de "Resultado", usando o método do item A, com a seguinte assinatura:

public static void imprime(double numero)

C) (2,0 pontos) Crie um método que tire a média de 2 números double, com a assinatura:

public static double media(double n1, double n2)

D) (2,0 pontos) Crie um método que tire a média de 3 números double, com a assinatura:

public static double media(double n1, double n2, double n3)

E) (2,0 pontos) Crie um método que tire a média de dois números inteiros, com a assinatura:

public static double media(int n1, int n2)

F) (2,0 pontos) Crie um método que receba duas Strings, cada uma contendo um número, e tire a média dos dois, com a assinatura:

public static double media(String n1, String n2)

3. ARRAYS (MATRIZES E VETORES)

Além dos tipos de dados básicos já definidos (boolean, byte, char, int, long, float e double) existe ainda dois outros tipos de dados que o Java compreende: classes e arrays. As classes serão estudadas num momento futuro, mas você já conhece bem uma delas: a classe String.

Vamos focar, no momento, nos **arrays**. Um array é uma matriz (ou uma tabela, se preferir) que guarda apenas valores de um mesmo tipo. Assim, ter um array de inteiros é como ter uma tabela que só guarda números inteiros. Observe a tabela abaixo:

1	2	3	4	5	6	7

Se eu quiser criar a tabela de inteiros acima, em Java, eu indico da seguinte forma:

```
int umaTabela[] = { 1, 2, 3, 4, 5, 6, 7 };
```

Observe a indicação de [e] em frente ao nome dado à variável. É exatamente essa indicação que faz com que o Java entenda que se trata de um **array** de **inteiros** sendo criado, e não apenas um número inteiro.

Os valores associados inicialmente são indicados separados por vírgula, dentro das chaves, do lado direito do igual.

Para recuperar os valores armazenados, usamos o número da posição. Observe na tabela abaixo a indicação do número de cada posição:

Posição	0	1	2	3	4	5	6
Valor	1	2	3	4	5	6	7

IMPORTANTE: Observe que a numeração das posições começa SEMPRE em zero!

Assim, se eu quiser ler o valor da posição 4, basta usar a seguinte expressão:

umaTabela[4]

Por exemplo:

```
x = umaTabela[4];
```

Isso colocará o valor 5 em X (observe, na tabela, que o valor guardado na posição 4 é o número 5):

Posição	0	1	2	3	4	5	6
Valor	1	2	3	4	5	6	7

O uso de tabelas facilita bastante a vida do programador. Suponhamos que precisamos guardar as notas de 30 alunos e, depois, imprimi-las separadamente. O código abaixo faz o serviço (considerando que os métodos **entrada** e **imprime** sejam definidos como anteriormente):

```
int i;
// Se não vamos inicializar um array, precisamos pedir pro Java alocar espaço para ele,
// Com a instrução new.
double notas[] = new double[30];

for (i = 0; i<30; i=i+1) {
      // Guarda na posição i a nota do aluno i
      notas[i] = entrada("Digite nota do aluno " + i);
      }
for (i = 0; i<30; i=i+1) {
      imprime("A nota do aluno " + i + " é: " + notas[i] + "\n");
      }
```

No exemplo acima, como o vetor notas[] não foi inicializado no código, precisamos pedir para o Java reservar um espaço na memória para todos os 30 números (afinal, só com "notas[]" o Java não tem como adivinhar o espaço necessário!). Para fazer isso, foi usada a instrução **new**, como indicado abaixo:

double notas[] = **new double[30];**

A parte em negrito pede a reserva do espaço de 30 números do tipo double.

3.1. Arrays Multidimensionais (OPCIONAL)

É claro que a tabela não precisa ser unidimensional. Ela pode ter duas ou mais dimensões. Por exemplo, considere a tabela abaixo (as linhas e colunas em cinza marcam a numeração de linhas e colunas, não fazem parte da tabela em si):

	0	**1**	**2**
0	30	50	70
1	40	60	80

Esta tabela poderia ser declarada, em Java, da seguinte forma:

```
int outraTabela[][] = { {30, 50, 70} , {40, 60, 80} };
```

Observe que a declaração tem um par de colchetes a mais e que os números de cada linha da tabela são declarados agrupados.

4. EXERCÍCIOS

A) (1,0 ponto) Crie outro método que imprima um número do tipo **double** em uma janela, com a seguinte assinatura:

public static void imprime(double numero)

B) (1,0 ponto) Crie um método que leia um número digitado pelo usuário, através de uma janela, com a seguinte assinatura:

public static double entrada(String mensagem)

C) (2,0 pontos) Crie um método **main** que leia as médias de 10 alunos, armazenando-os em um **array** do tipo double.

DICA:

- Use um **for** para repetir a pergunta.
- Use o método desenvolvido no item B para ler as notas.

D) (2,0 pontos) Ajuste o programa desenvolvido no item C para que ele calcule a média das notas digitadas e guarde em uma variável chamda **media**.

DICA:

- Use um **for** para somar as notas.
- Para calcular a média, divida a soma total das notas pelo número de alunos.

E) (1,0 ponto) Imprima o resultado da média usando o método do item A.

F) (1,5 pontos) Ajuste o programa para que ele arredonde a nota média final para 2 casas decimais antes de imprimir.

DICA:

- Use a fórmula: valor = (Math.rint(valor*100))/100;

G) (0,5 ponto) Ajuste o método **main** para que ele também imprima a **DIFERENÇA** entre a nota média antes de arredondar e a nota média depois do arredondamento.

H) (0,5 ponto) Ajuste o método **main** para que, no final de tudo, ele imprima a menor nota da turma.

DICA:

- Use um **for** para testar todas as notas e sempre armazene a menor encontrada.

I) (0,5 ponto) Ajuste o método **main** para que, no final de tudo, ele imprima a maior nota da turma.

DICA:

- Use um **for** para testar todas as notas e sempre armazene a maior encontrada.