

6

Métodos: Um exame mais profundo



OBJETIVOS

- Neste capítulo, você aprenderá:
- Como métodos e campos `static` são associados a uma classe inteira em vez de instâncias específicas da classe.
- Como utilizar métodos `Math` comuns disponíveis no API do Java.
- Os mecanismos para passar informações entre métodos.
- Como o mecanismo de retorno/chamada de método é suportado pela pilha de chamadas de métodos e registros de ativação.
- Como pacotes agrupam classes relacionadas.
- O que é a sobrecarga de método e como criar métodos sobrecarregados.



6.1 Introdução

- **Técnica do ‘dividir para conquistar’:**
 - Construa um programa grande a partir de partes menores (ou módulos).
 - Pode ser realizado utilizando métodos.
- **Métodos `Static` podem ser chamados sem a necessidade de um objeto da classe.**



6.2 Módulos de programa em Java

- **Java Application Programming Interface (API):**
 - Também conhecida como *API do Java* ou *biblioteca de classes Java*.
 - Contém os métodos e classes predefinidas:
 - Classes relacionadas são organizadas em pacotes.
 - Inclui os métodos para matemática, manipulações de strings/caracteres, entrada/saída, bancos de dados, rede, processamento de arquivos, verificação de erros e outros.



Boa prática de programação 6.1

Familiarize-se com a rica coleção de classes e métodos fornecidos pela API do Java
(java.sun.com/j2se/5.0/docs/api/index.html).

Na Seção 6.8, apresentamos uma visão geral dos vários pacotes comuns. No Apêndice G, explicamos como navegar pela documentação da API do Java.



Observação de engenharia de software 6.1

Não tente reinventar a roda. Quando possível, reutilize as classes e métodos na API do Java. Isso reduz o tempo de desenvolvimento de programas e evita a introdução de erros.



6.2 Módulos de programa em Java (*Continuação*)

- Métodos:
 - São denominados *funções* ou *procedimentos* nas outras linguagens.
 - Permitem ao programador modularizar os programas separando suas tarefas em unidades autocontidas.
 - Têm uma abordagem de ‘dividir para conquistar’.
 - São reutilizáveis em futuros programas.
 - Evitam repetição de código.



Observação de engenharia de software 6.2

Para promover a capacidade de reutilização de software, todos os métodos devem estar limitados à realização de uma única tarefa bem definida e o nome do método deve expressar essa tarefa efetivamente.

Esses métodos tornam mais fácil escrever, depurar, manter e modificar programas.



Dica de prevenção de erro 6.1

Um pequeno método que realiza uma tarefa é mais fácil de testar e depurar do que um método maior que realiza muitas tarefas.



Observação de engenharia de software 6.3

Se você não puder escolher um nome conciso que expresse a tarefa de um método, seu método talvez tente realizar um número excessivo de tarefas.

Em geral, é melhor dividir esse método em várias declarações de método menores.



6.3 Métodos `static`, campos `static` e a classe `Math`

- **Método `static` (ou método de classe)**
 - Aplica-se à classe como um todo, em vez de a um objeto específico da classe.
 - Chame um método `static` utilizando a chamada de método:
NomeDaClasse.nomeDoMétodo(argumentos)
 - Todos os métodos da classe `Math` são `static`:
 - Exemplo: `Math.sqrt(900.0)`



Observação de engenharia de software 6.4

A classe `Math` faz parte do pacote `java.lang`, que é implicitamente importado pelo compilador; assim, não é necessário importar a classe `Math` para utilizar seus métodos.



6.3 Métodos `static`, campos `static` e a classe `Math` (*Cont.*)

- **Constantes:**
 - Palavra-chave `final`.
 - Não pode ser alterada depois da inicialização.
- **Campos `static` (ou variáveis de classe):**
 - São campos em que uma cópia da variável é compartilhada entre todos os objetos da classe.
- `Math.PI` e `Math.E` são campos `final static` da classe `Math`.



Método	Descrição	Exemplo
<code>abs(x)</code>	valor absoluto de x	<code>abs(23.7)</code> é 23.7 <code>abs(0.0)</code> é 0.0 <code>abs(-23.7)</code> é 23.7
<code>ceil(x)</code>	arredonda x para o menor inteiro não menor que x	<code>ceil(9.2)</code> é 10.0 <code>ceil(-9.8)</code> é -9.0
<code>cos(x)</code>	co-seno trigonométrico de x (x em radianos)	<code>cos(0.0)</code> é 1.0
<code>exp(x)</code>	método exponencial e^x	<code>exp(1.0)</code> é 2.71828 <code>exp(2.0)</code> é 7.38906
<code>floor(x)</code>	arredonda x para o maior inteiro não maior que x	<code>floor(9.2)</code> é 9.0 <code>floor(-9.8)</code> é -10.0
<code>log(x)</code>	logaritmo natural de x (base e)	<code>log(Math.E)</code> é 1.0 <code>log(Math.E * Math.E)</code> é 2.0
<code>max(x, y)</code>	maior valor de x e y	<code>max(2.3, 12.7)</code> é 12.7 <code>max(-2.3, -12.7)</code> é -2.3
<code>min(x, y)</code>	menor valor de x e y	<code>min(2.3, 12.7)</code> é 2.3 <code>min(-2.3, -12.7)</code> é -12.7
<code>pow(x, y)</code>	x elevado à potência de y (isto é, x^y)	<code>pow(2.0, 7.0)</code> é 128.0 <code>pow(9.0, 0.5)</code> é 3.0
<code>sin(x)</code>	seno trigonométrico de x (x em radianos)	<code>sin(0.0)</code> é 0.0
<code>sqrt(x)</code>	raiz quadrada de x	<code>sqrt(900.0)</code> é 30.0
<code>tan(x)</code>	tangente trigonométrica de x (x em radianos)	<code>tan(0.0)</code> é 0.0

Figura 6.2 | Métodos da classe Math.



6.3 Métodos `static`, campos `static` e a classe `Math` (*Cont.*)

- **Método `main`:**
 - `main` é declarado `static` para poder ser invocado sem criar um objeto da classe que contém `main`.
 - Qualquer classe pode conter um método `main`:
 - A JVM invoca o método `main` que pertence à classe especificada pelo primeiro argumento da linha de comando para o comando `java`.



6.4 Declarando métodos com múltiplos parâmetros

- **Múltiplos parâmetros podem ser declarados especificando uma lista separada por vírgulas.**
 - **Os argumentos passados em uma chamada de método devem ser consistentes com o número, tipos e ordem dos parâmetros.**
 - Às vezes, eles são chamados *parâmetros formais*



Resumo

Maxi mumFi nder.j ava

(1 de 2)

```
1 // Fig. 6.3: Maxi mumFi nder.j ava
2 // Método maxi mum declarado pelo programador.
3 import java.util.Scanner;
4
5 public class Maxi mumFi nder
6 {
7     // obtém três valores de ponto flutuante e localiza o valor máximo
8     public void determi neMaxi mum()
9     {
10         // cria Scanner para entrada a partir da janela de comando
11         Scanner input = new Scanner( System.in );
12
13         // obtém a entrada do usuário
14         System.out.print(
15             "Enter three floating-point values separated by spaces: " );
16         double number1 = input.nextDouble(); // read first double
17         double number2 = input.nextDouble(); // read second double
18         double number3 = input.nextDouble(); // read third double
19
20         // determi na o val or máximo
21         double resul t = maxi mum( number1, number2, number3 );
22
23         // exi be o val or máximo
24         System.out.println( "Maxi mum i s: " + resul t );
25     } // fim do método determi neMaxi mum
26
```

Solicita que o usuário insira e leia
três valores **double**

Chama o método **maxi mum**

Exibe o valor máximo



```
27 // retorna o máximo dos seus três parâmetros de double
28 public double maximo( double x, double y, double z )
29 {
30     double maximoValue = x; // supõe que x é o maior valor inicial
31
32     // determina se y é maior que maximoValue
33     if ( y > maximoValue )
34         maximoValue = y;
35
36     // determina se z é maior que maximoValue
37     if ( z > maximoValue )
38         maximoValue = z;
39
40     return maximoValue;
41 } // fim do método maximo
42 } // fim da classe MaximoFinder
```

Declara o método **maximo**

Compara **y** e **maximoValue**

2)

Compara **z** e **maximoValue**

Retorna o valor máximo

MaximoFinder.java



Resumo

Maxi mumFi nderTest
.j ava

```

1 // Fig. 6.4: Maxi mumFi nderTest.j ava
2 // Aplicativo para testar a classe Maxi mumFi nder.
3
4 public class Maxi mumFi nderTest
5 {
6     // ponto de partida do aplicativo
7     public static void main( String args[] )
8     {
9         Maxi mumFi nder maxi mumFi nder = new Maxi mumFi nder();
10        maxi mumFi nder.determi neMaxi mum();
11    } // fim de mai n
12 } // fim da classe Maxi mumFi nderTest

```

Cria um objeto
Maxi mumFi nder

Chama o método
determi neMaxi mum

Enter three floating-point values separated by spaces: 9.35 2.74 5.1
Maxi mum i s: 9.35

Enter three floating-point values separated by spaces: 5.8 12.45 8.32
Maxi mum i s: 12.45

Enter three floating-point values separated by spaces: 6.46 4.12 10.54
Maxi mum i s: 10.54



Erro comum de programação 6.1

Declarar parâmetros de método do mesmo tipo como `float x, y` em vez de `float x, float y` é um erro de sintaxe — um tipo é necessário para cada parâmetro na lista de parâmetros.



Observação de engenharia de software 6.5

Um método com muitos parâmetros pode estar realizando tarefas demais. Considere dividir o método em métodos menores que realizam tarefas separadas.

Como uma diretriz, se possível, tente ajustar o cabeçalho do método em uma linha.



6.4 Declarando métodos com múltiplos parâmetros (Cont.)

- **Reutilizando o método `Math.max`:**
 - A expressão de `Math.max (x, Math.max (y, z))` determina o máximo de `y` e `z` e, então, determina o máximo de `x` e esse valor.
- **Concatenação de string:**
 - Utilizar o operador `+` com duas `Strings` concatena-as em uma nova `String`.
 - Utilizar o operador `+` com uma `String` e um valor de um outro tipo de dados concatena `String` com uma representação de `String` do outro valor.
 - Se o outro valor for um objeto, seu método `toString` é chamado para gerar sua representação de `String`.



Erro comum de programação 6.2

É um erro de sintaxe dividir uma literal de String em múltiplas linhas dentro de um programa.

Se uma string não couber em uma linha, divida a string em várias strings menores e utilize a concatenação para formar a String desejada.



Erro comum de programação 6.3

Confundir o operador + utilizado para concatenação de string com o operador + utilizado para adição pode levar a resultados estranhos. O Java avalia os operandos de um operador da esquerda para a direita. Por exemplo, suponha que a variável inteira y tenha o valor 5, a expressão "y + 2 = " + y + 2 resulta na string "y + 2 = 52", e não em "y + 2 = 7", pois o primeiro valor de y (5) é concatenado com a string "y + 2 = ". Em seguida, o valor 2 é concatenado com a nova e maior string "y + 2 = 5". A expressão "y + 2 = " + (y + 2) produz o resultado desejado "y + 2 = 7" .



6.5 Notas sobre a declaração e utilização de métodos

- **Há três maneiras de chamar um método:**
 - Utilize o nome de um método sozinho para chamar um outro método da mesma classe.
 - Utilize uma variável que contém uma referência a um objeto, seguido por um ponto (.) e o nome do método para chamar um método do objeto referenciado.
 - Utilize o nome da classe e um ponto (.) para chamar um método `static` de uma classe.
- **Métodos `static` não podem chamar diretamente métodos não-`static` da mesma classe.**



6.5 Notas sobre a declaração e utilização de métodos (*Continuação*)

- Há três maneiras de retornar o controle à instrução chamadora:
 - Se o método não retornar um resultado:
 - o fluxo do programa alcança a chave direita de fechamento do método; ou
 - o programa executa a instrução `return; .`
 - Se o método retornar um resultado:
 - O programa executa a *expressão* da instrução `return; .`
 - A *expressão* é primeiro avaliada e, então, seu valor é retornado ao chamador.



Erro comum de programação 6.4

Declarar um método fora do corpo de uma declaração de classe ou dentro do corpo de um outro método é um erro de sintaxe.



Erro comum de programação 6.5

Omitir o tipo-do-valor-de-retorno em uma declaração de método é um erro de sintaxe.



Erro comum de programação 6.6

Colocar um ponto-e-vírgula após o parêntese direito que envolve a lista de parâmetros de uma declaração de método é um erro de sintaxe.



Erro comum de programação 6.7

Redeclarar um parâmetro de método como uma variável local no corpo do método é um erro de compilação.



Erro comum de programação 6.8

Esquecer de retornar um valor em um método que deve retornar um valor é um erro de compilação. Se um tipo-do-valor-de-retorno além de void for especificado, o método deverá conter uma instrução `return` que retorne um valor consistente com o tipo-do-valor-de-retorno do método. Retornar um valor de um método cujo tipo de retorno foi declarado como void é um erro de compilação.



6.7 Promoção e coerção de argumentos

- **Promoção de argumentos**
 - O Java promoverá um argumento de chamada de método a fim de coincidir com seu parâmetro de método correspondente de acordo com as regras da promoção.
 - Os valores em uma expressão são promovidos para o tipo ‘mais alto’ na expressão (uma cópia temporária do valor é criada).
 - Converter valores para tipos mais baixos resulta em um erro de compilação, a menos que o programador faça com que a conversão ocorra explicitamente.
 - Coloque o tipo de dados desejado entre parênteses antes do valor (por exemplo: `(int) 4.5`).



Tipo	Promoções válidas
doubl e	Nenhuma
fl oat	doubl e
l ong	fl oat ou doubl e
i nt	l ong, fl oat ou doubl e
char	i nt, l ong, fl oat ou doubl e
shout	i nt, l ong, fl oat ou doubl e (mas não char)
byte	shout, i nt, l ong, fl oat ou doubl e (mas não char)
bool ean	Nenhuma (os valores bool ean não são considerados como números em Java)

Figura 6.5 | Promoções permitidas para tipos primitivos.



Erro comum de programação 6.9

Converter um valor de tipo primitivo em um outro tipo primitivo pode alterar o valor se o novo tipo não for uma promoção válida.

Por exemplo, converter um valor de ponto flutuante em um valor integral pode introduzir erros de truncamento (perda da parte fracionária) no resultado.



6.8 Pacotes da API do Java

- Incluir a declaração
`import java.util.Scanner;`
permite ao programador utilizar `Scanner` em vez de `java.util.Scanner`.
- Documentação da API do Java:
 - java.sun.com/j2se/5.0/docs/api/index.html
- Visão geral dos pacotes no JDK 5.0:
 - java.sun.com/j2se/5.0/docs/api/overview-summary.html



Pacote	Descrição
<code>java.applet</code>	O Java Applet Package contém uma classe e várias interfaces exigidas para criar applets Java — programas que executam nos navegadores da Web. (Os applets serão discutidos no Capítulo 20, Introdução a applets Java; e as interfaces no Capítulo 10, Programação orientada a objetos: Polimorfismo.)
<code>java.awt</code>	O Java Abstract Window Toolkit Package contém as classes e interfaces exigidas para criar e manipular GUIs no Java 1.0 e 1.1. Nas versões atuais do Java, os componentes GUI Swing dos pacotes <code>javax.swing</code> são frequentemente utilizados em seu lugar. (Alguns elementos do pacote <code>java.awt</code> serão discutidos no Capítulo 11, Componentes GUI: Parte 1, Capítulo 12, Imagens gráficas e Java2D e Capítulo 22, Componentes GUI: Parte 2.)
<code>java.awt.event</code>	O Java Abstract Window Toolkit Event Package contém classes e interfaces que permitem o tratamento de eventos para componentes GUI tanto nos pacotes <code>java.awt</code> como <code>javax.swing</code> . (Você aprenderá mais sobre esse pacote no Capítulo 11: Componentes GUI: Parte 1 e no Capítulo 22: Componentes GUI: Parte 2.)
<code>java.io</code>	O Java Input/Output Package contém classes e interfaces que permitem aos programas gerar entrada e saída de dados. (Você aprenderá mais sobre esse pacote no Capítulo 14, Arquivos e fluxos.)
<code>java.lang</code>	O Java Language Package contém classes e interfaces (discutidas por todo esse texto) que são exigidas por muitos programas Java. Esse pacote é importado pelo compilador para todos os programas, assim o programador não precisa fazer isso.

Figura 6.6 | Pacotes da API do Java (um subconjunto). (Parte 1 de 2)



Slide 36

EFz2

OK

Edson Furmankiewicz; 21/10/2005

Pacote	Descrição
<code>java.net</code>	O Java Networking Package contém classes e interfaces que permitem aos programas comunicar-se via redes de computadores, como a Internet. (Você verá mais detalhes sobre isso no Capítulo 24, Redes.)
<code>java.text</code>	O Java Text Package contém classes e interfaces que permitem aos programas manipular números, datas, caracteres e strings. O pacote fornece recursos de internacionalização que permitem a um programa ser personalizado para um local específico (por exemplo, um programa pode exibir strings em diferentes idiomas com base no país do usuário).
<code>java.util</code>	O Java Utilities Package contém classes utilitárias e interfaces que permitem ações como manipulações de data e hora, processamento de números aleatórios (classe <code>Random</code>), armazenamento e processamento de grandes volumes de dados e a divisão de strings em partes menores chamadas tokens (classe <code>StringTokenizer</code>). (Você aprenderá mais sobre os recursos desse pacote no Capítulo 19, Coleções)
<code>javax.swing</code>	O Java Swing GUI Components Package contém classes e interfaces para componentes GUI Swing do Java que fornecem suporte para GUIs portáteis. (Você aprenderá mais sobre esse pacote no Capítulo 11: Componentes GUI: Parte 1 e no Capítulo 22: Componentes GUI: Parte 2.)
<code>javax.swing.event</code>	O Java Swing Event Package contém classes e interfaces que permitem o tratamento de eventos (por exemplo, responder a cliques de botão) para componentes GUI no pacote <code>javax.swing</code> . (Você aprenderá mais sobre esse pacote no Capítulo 11: Componentes GUI: Parte 1 e no Capítulo 22: Componentes GUI: Parte 2.)

Figura 6.6 | Pacotes da API do Java (um subconjunto). (Parte 2 de 2.)



EFz3

OK

Edson Furmankiewicz; 21/10/2005

Boa prática de programação 6.2

A documentação on-line da API do Java é fácil de pesquisar e fornece vários detalhes sobre cada classe. À medida que você aprende uma classe nesse livro, deve criar o hábito de examinar a classe na documentação on-line para informações adicionais.



6.9 Estudo de caso: Geração de números aleatórios

- **Geração de número aleatório:**
 - **Método `static random` da classe `Math`.**
 - Retorna `double`s no intervalo $0.0 \leq x < 1.0$
 - **Classe `Random` do pacote `java.util`:**
 - Pode produzir números *pseudo-aleatórios* `boolean`, `byte`, `float`, `double`, `int`, `long` e gaussianos.
 - É semeado com a hora do dia atual a fim de gerar diferentes seqüências dos números toda vez que o programa é executado.



Resumo

```

1 // Fig. 6.7: RandomIntegers.java
2 // Inteiros aleatórios deslocados e escalonados.
3 import java.util.Random; // program uses class Random
4
5 public class RandomIntegers
6 {
7     public static void main( String args[] )
8     {
9         Random randomNumbers = new Random(); // gerador de número aleatório
10        int face; // armazena cada inteiro aleatório gerado
11
12        // itera 20 vezes pelo loop
13        for ( int counter = 1; counter <= 20; counter++ )
14        {
15            // seleciona o inteiro aleatório entre 1 a
16            face = 1 + randomNumbers.nextInt( 6 );
17
18            System.out.printf( "%d ", face ); // exibe o valor gerado
19
20            // se o contador for divisível por 5, inicia uma nova linha de saída
21            if ( counter % 5 == 0 )
22                System.out.println();
23        } // fim do for
24    } // fim de main
25 } // fim da classe RandomIntegers

```

Importa a classe **Random** do pacote **java.util**

Cria um objeto **Random**

Gera uma rolagem
aleatória de um dado

RandomIntegers
.java
(1 de 2)



Resumo

RandomIntegers

.java

(2 de 2)

1	5	3	6	2
5	2	6	5	2
4	4	4	2	6
3	1	6	2	2

Dois conjuntos diferentes de resultados
contendo inteiros no intervalo 1-6

6	5	4	2	6
1	2	5	1	3
6	3	2	2	1
6	4	2	6	4



Resumo

RolIDie.java

(1 de 3)

```
1 // Fig. 6.8: RolIDie.java
2 // Rola um dado de seis lados 6000 vezes.
3 import java.util.Random;
4
5 public class RolIDie
6 {
7     public static void main( String args[] )
8     {
9         Random randomNumbers = new Random(); // gerador de número aleatório
10
11         int frequency1 = 0; // mantém a contagem de 1s lançados
12         int frequency2 = 0; // contagem de 2s lançados
13         int frequency3 = 0; // contagem de 3s lançados
14         int frequency4 = 0; // contagem de 4s lançados
15         int frequency5 = 0; // contagem de 5s lançados
16         int frequency6 = 0; // contagem de 6s lançados
17     }
```

Importa a classe **Random** do pacote **java.util**

Cria um objeto **Random**

Declara contadores de
frequência



Resumo

Rol I Di e. j ava

(2 de 3)

```

18  int face; // armazena o valor lançado mais recentemente
19
20  // resume os resultados de 6000 lançamentos de um dado
21  for ( int roll = 1; roll <= 6000; roll++ )
22  {
23      face = 1 + randomNumbers.nextInt( 6 ); // número entre 1 a 6
24
25      // determina valor de lançamento de 1 a 6 e incrementa o contador apropriado
26      switch ( face )
27      {
28          case 1:
29              ++frequency1; // incrementa o contador de 1s
30              break;
31          case 2:
32              ++frequency2; // incrementa o contador de 2s
33              break;
34          case 3:
35              ++frequency3; // incrementa o contador de 3s
36              break;
37          case 4:
38              ++frequency4; // incrementa o contador de 4s
39              break;
40          case 5:
41              ++frequency5; // incrementa o contador de 5s
42              break;
43          case 6:
44              ++frequency6; // incrementa o contador de 6s
45              break; // opcional no final do switch
46      } // fim do switch
47  } // fim do for
48

```

Itera 6000 vezes

Gera uma rolagem aleatória de dados

switch baseado na rolagem dos dados



Resumo

Roll Die.java

(3 de 3)

```

49      System.out.println( "Face\tFrequency" ); // gera saída dos cabeçalhos
50      System.out.printf( "1\t%d\n2\t%d\n3\t%d\n4\t%d\n5\t%d\n6\t%d\n",
51          frequency1, frequency2, frequency3, frequency4,
52          frequency5, frequency6 );
53  } // fim do main
54 } // fim da classe RollDie

```

Exibe as frequências da
rolagem dos dados

Face	Frequency
1	982
2	1001
3	1015
4	1005
5	1009
6	988

Face	Frequency
1	1029
2	994
3	1017
4	1007
5	972
6	981



Erro comum de programação 6.10

Um erro de compilação ocorre quando uma variável local é declarada mais de uma vez em um método.



Dica de prevenção de erro 6.3

Utilize nomes diferentes para campos e variáveis locais para ajudar a evitar erros de lógica sutis que ocorrem quando um método é chamado e uma variável local do método sombreia um campo com o mesmo nome na classe.



6.12 Sobrecarga de método

- **Sobrecarga de método**
 - **Múltiplos métodos com o mesmo nome, mas diferentes tipos, número ou ordem dos parâmetros nas suas listas de parâmetros.**
 - **O compilador decide qual método está sendo chamado comparando uma lista dos argumentos da chamada de método com uma das listas de parâmetros dos métodos sobrecarregados.**
 - **A ordem, nome, número, tipo dos parâmetros de um método formam sua assinatura.**
 - **As diferenças no tipo de retorno são irrelevantes na sobrecarga de método.**
 - **Métodos sobrecarregados podem ter diferentes tipos de retorno.**
 - **Métodos com diferentes tipos de retorno, mas com a mesma assinatura, resultam em um erro de compilação.**



Resumo

MethodOverload.
java

```

1 // Fig. 6.13: MethodOverload.java
2 // Declarações de métodos sobrecarregados.
3
4 public class MethodOverload
5 {
6     // testa métodos square sobrecarregados
7     public void testOverloadedMethods()
8     {
9         System.out.printf( "Square of integer 7 is %d\n", square( 7 ) );
10        System.out.printf( "Square of double 7.5 is %f\n", square( 7.5 ) );
11    } // end method testOverloadedMethods
12
13    // método square com argumento int
14    public int square( int intValue )
15    {
16        System.out.printf( "\nCalled square with int\n",
17                           intValue );
18        return intValue * intValue;
19    } // fim do método square com argumento int
20
21    // método square com argumento double
22    public double square( double doubleValue )
23    {
24        System.out.printf( "\nCalled square with double argument: %f\n",
25                           doubleValue );
26        return doubleValue * doubleValue;
27    } // fim do método square com argumento double
28 } // fim da classe MethodOverload

```

Chama corretamente o método
"square de int"

Chama corretamente o método
"square de double"

Declarando o método
"square de int"

Declarando o método
"square de double"



Resumo

MethodOverloadTest.java

```
1 // Fig. 6.14: MethodOverloadTest.java
2 // Aplicativo para testar a classe MethodOverload.
3
4 public class MethodOverloadTest
5 {
6     public static void main( String args[] )
7     {
8         MethodOverload methodOverload = new MethodOverload();
9         methodOverload.testOverloadedMethods();
10    } // fim de main
11 } // fim da classe MethodOverloadTest
```

Called square with int argument: 7
Square of integer 7 is 49

Called square with double argument: 7.500000
Square of double 7.5 is 56.250000



Resumo

MethodOverloadError.java

```
1 // Fig. 6.15: MethodOverloadError.java
2 // Métodos sobrecarregados com assinaturas idênticas
3 // resulta em erros de compilação, mesmo se os tipos de retorno forem diferentes.
4
5 public class MethodOverloadError
6 {
7     // declaration of method square with int argument
8     public int square( int x )
9     {
10         return x * x;
11     }
12
13     // segunda declaração do método square com argumento int
14     // resulta em erros de compilação mesmo que os tipos de retorno sejam diferentes
15     public double square( int y )
16     {
17         return y * y;
18     }
19 } // fim da classe MethodOverloadError
```

Mesma assinatura de método

```
MethodOverloadError.java:15: square(int) is already defined in
MethodOverloadError
    public double square( int y )
                        ^
```

1 error

Erro de compilação



Erro comum de programação 6.11

Declarar métodos sobrecarregados com listas de parâmetros idênticas é um erro de compilação independentemente de os tipos de retorno serem diferentes.

