Os métodos da classe `Math`

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.

Método	Descrição	Exemplo

`abs(x)`	valor absoluto de x	`abs(23.7)` é `23.7`
		`abs(0.0)` é `0.0`
		`abs(-23.7)` é `23.7`

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

6.4 Declarando métodos com múltiplos parâmetros

```
1  // Figura 6.3: MaximumFinder.java
2  // Método maximum declarado pelo programador com três parâmetros double.
3  import java.util.Scanner;
4
5  public class MaximumFinder
6  {
7     // obtém três valores de ponto flutuante e localiza o valor máximo
8     public static void main(String[] args)
9     {
10       // cria Scanner para entrada a partir da janela de comando
11       Scanner input = new Scanner(System.in);
12
13       // solicita e insere três valores de ponto flutuante
14       System.out.print(
15          "Enter three floating-point values separated by spaces: ");
16       double number1 = input.nextDouble(); // lê o primeiro double
17       double number2 = input.nextDouble(); // lê o segundo double
18       double number3 = input.nextDouble(); // lê o terceiro double
19
20       // determina o valor máximo
21       double result = maximum(number1, number2, number3);
22
23       // exibe o valor máximo
24       System.out.println("Maximum is: " + result);
25    }
26
```

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

```
Enter three floating-point values separated by spaces: 9.35 2.74 5.1
Maximum is: 9.35
```

```
Enter three floating-point values separated by spaces: 5.8 12.45 8.32
Maximum is: 12.45
```

```
Enter three floating-point values separated by spaces: 6.46 4.12 10.54
Maximum is: 10.54
```

Implementando o método `maximum` reutilizando o método `Math.max`

```
return Math.max(x, Math.max(y, z));
```

6.7 Promoção e coerção de argumentos

```
square((int) doubleValue)
```

Essa chamada de método faz uma coerção explícita (converte) do valor de `doubleValue` em um inteiro temporário para uso no método `square`. Assim, se o valor do `doubleValue` for `4.5`, o método receberá o valor `4` e retornará `16`, não `20.25`.

Tipo	Promoções válidas
`double`	None
`float`	`Double`
`long`	`float` ou `double`
`int`	`long`, `float` ou `double`
`char`	`int`, `long`, `float` ou `double`
`short`	`int`, `long`, `float` ou `double` (mas não `char`)
`byte`	`short`, `int`, `long`, `float` ou `double` (mas não `char`)
`boolean`	Nenhuma (os valores `boolean` não são considerados números em Java)

6.8 Pacotes de Java API

Pacote	Descrição
`java.awt.event`	O **Java Abstract Window Toolkit Event Package** contém classes e interfaces que permitem tratamento de evento para componentes GUI em nos pacotes `java.awt` e `javax.swing` (Veja o Capítulo 12, "Componentes GUI: parte 1", e o Capítulo 22, "Componentes GUI: parte 2".)
`java.awt.geom`	O **Java 2D Shapes Package** contém classes e interfaces para trabalhar com as avançadas capacidades gráficas bidimensionais do Java. (Veja o Capítulo 13, "Imagens gráficas e Java 2D".)
`java.io`	O **Java Input/Output Package** contém classes e interfaces que permitem aos programas gerar entrada e saída de dados. (Veja o Capítulo 15, "Arquivos, fluxos e serialização de objetos".)
`java.lang`	O **Java Language Package** contém classes e interfaces (discutidas por todo este texto) que são exigidas por muitos programas Java. Esse pacote é importado pelo compilador em todos os programas.

`java.net`	O **Java Networking Package** contém classes e interfaces que permitem aos programas comunicar-se via redes de computadores, como a internet. (Veja o Capítulo 28, em inglês, na Sala Virtual do livro.)
`java.security`	O **Java Security Package** contém classes e interfaces para melhorar a segurança do aplicativo.
`java.sql`	O **JDBC Package** contém classes e interfaces para trabalhar com bancos de dados. (Veja o Capítulo 24, "Acesso a bancos de dados com JDBC".)
`java.util`	O **Java Utilities Package** contém classes e interfaces utilitárias que permitem armazenar e processar grandes quantidades de dados. Muitas dessas classes e interfaces foram atualizadas para suportar novos recursos lambda do Java SE 8. (Veja o Capítulo 16, "Coleções genéricas".)
`java.util.concurrent`	O **Java Concurrency Package** contém classes utilitárias e interfaces para implementar programas que podem realizar múltiplas tarefas paralelamente. (Veja o Capítulo 23, "Concorrência".)
`javax.swing`	O **Java Swing GUI Components Package** contém classes e interfaces para componentes GUI Swing do Java que fornecem suporte para GUIs portáveis. Esse pacote ainda usa alguns elementos do pacote `java.awt` mais antigo. (Veja o Capítulo 12, "Componentes GUI: parte 1" e o Capítulo 22, "Componentes GUI: parte 2".)
`javax.swing.event`	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 do pacote `javax.swing`. (Veja o Capítulo 12, "Componentes GUI: parte 1" e o Capítulo 22, "Componentes GUI: parte 2".)
`javax.xml.ws`	O **JAX-WS Package** contém classes e interfaces para trabalhar com serviços da web no Java. (Consulte o Capítulo 32, em inglês, na Sala Virtual.)
pacotes `javafx`	JavaFX é a tecnologia da GUI preferida para o futuro. Discutiremos esses pacotes no Capítulo 25, "GUI do JavaFX: parte 1" e nos capítulos "JavaFX GUI" e "Multimídia", em inglês, na Sala Virtual.
Alguns pacotes Java SE 8 usados neste livro	
`java.time`	O novo **pacote** Java SE 8 **Date/Time API** contém classes e interfaces para trabalhar com datas e horas. Esses recursos são projetados para substituir as capacidades de data e hora mais antigas do pacote `java.util`. (Veja o Capítulo 23, "Concorrência".)
`java.util.function` e `java.util.stream`	Esses pacotes contêm classes e interfaces para trabalhar com as capacidades de programação funcional do Java SE 8. (Veja o Capítulo 17, "Lambdas e fluxos Java SE 8".)

6.9 Estudo de caso: geração segura de números aleatórios

```java
// Figura 6.6: RandomIntegers.java
// Inteiros aleatórios deslocados e escalonados.
import java.security.SecureRandom; // o programa usa a classe SecureRandom

public class RandomIntegers
{
   public static void main(String[] args)
   {
      // o objeto randomNumbers produzirá números aleatórios seguros
      SecureRandom randomNumbers = new SecureRandom();

      // faz o loop 20 vezes
      for (int counter = 1; counter <= 20; counter++)
      {
         // seleciona o inteiro aleatório entre 1 e 6
         int face = 1 + randomNumbers.nextInt(6);

         System.out.printf("%d  ", face); // exibe o valor gerado

         // se o contador for divisível por 5, inicia uma nova linha de saída
         if (counter % 5 == 0)
            System.out.println();
      }
   }
} // fim da classe RandomIntegers
```

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

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

```java
// Figura 6.7: RollDie.java
// Rola um dado de seis lados 6.000.000 vezes.
import java.security.SecureRandom;

public class RollDie
{
   public static void main(String[] args)
   {
      // o objeto randomNumbers produzirá números aleatórios seguros
      SecureRandom randomNumbers = new SecureRandom();

      int frequency1 = 0; // contagem de 1s lançados
      int frequency2 = 0; // contagem de 2s lançados
      int frequency3 = 0; // contagem de 3s lançados
      int frequency4 = 0; // contagem de 4s lançados
      int frequency5 = 0; // contagem de 5s lançados
      int frequency6 = 0; // contagem de 6s lançados

      // soma 6.000.000 lançamentos de um dado
      for (int roll = 1; roll <= 6000000; roll++)
      {
         int face = 1 + randomNumbers.nextInt(6); // número entre 1 e 6

         // usa o valor 1-6 para as faces a fim de determinar qual contador incrementar
         switch (face)
         {
            case 1:
               ++frequency1; // incrementa o contador de 1s
               break;
            case 2:
               ++frequency2; // incrementa o contador de 2s
               break;
            case 3:
               ++frequency3; // incrementa o contador de 3s
               break;
            case 4:
               ++frequency4; // incrementa o contador de 4s
               break;
            case 5:
               ++frequency5; // incrementa o contador de 5s
               break;
            case 6:
               ++frequency6; // incrementa o contador de 6s
               break;
         }
      }

      System.out.println("Face\tFrequency"); // cabeçalhos de saída
      System.out.printf("1\t%d%n2\t%d%n3\t%d%n4\t%d%n5\t%d%n6\t%d%n",
         frequency1, frequency2, frequency3, frequency4,
         frequency5, frequency6);
   }
} // fim da classe RollDie
```

```
Face   Frequency
1      999501
2      1000412
3      998262
4      1000820
5      1002245
6      998760
```

```
Face   Frequency
1      999647
2      999557
3      999571
4      1000376
5      1000701
6      1000148
```

6.10 Estudo de caso: um jogo de azar; apresentando tipos enum

Você lança dois dados. Cada dado tem seis faces que contêm um, dois, três, quatro, cinco e seis pontos, respectivamente. Depois que os dados param de rolar, a soma dos pontos nas faces viradas para cima é calculada. Se a soma for 7 ou 11 no primeiro lance, você ganha. Se a soma for 2, 3 ou 12 no primeiro lance (chamado "craps"), você perde (isto é, a "casa" ganha). Se a soma for 4, 5, 6, 8, 9 ou 10 no primeiro lance, essa soma torna-se sua "pontuação". Para ganhar, você deve continuar a rolar os dados até "fazer sua pontuação" (isto é, obter um valor igual à sua pontuação). Você perde se obtiver um 7 antes de fazer sua pontuação.

```
1  // Figura 6.8: Craps.java
2  // A classe Craps simula o jogo de dados craps.
3  import java.security.SecureRandom;
4
5  public class Craps
6  {
7     // cria um gerador seguro de números aleatórios para uso no método rollDice
8     private static final SecureRandom randomNumbers = new SecureRandom();
9
10    // tipo enum com constantes que representam o estado do jogo
11    private enum Status { CONTINUE, WON, LOST };
12
13    // constantes que representam lançamentos comuns dos dados
14    private static final int SNAKE_EYES = 2;
15    private static final int TREY = 3;
16    private static final int SEVEN = 7;
17    private static final int YO_LEVEN = 11;
18    private static final int BOX_CARS = 12 ;
19
20    // joga uma partida de craps
21    public static void main(String[] args)
22    {
23       int myPoint = 0; // pontos se não ganhar ou perder na 1ª rolagem
24       Status gameStatus; // pode conter CONTINUE, WON ou LOST
25
26       int sumOfDice = rollDice(); // primeira rolagem dos dados
27
28       // determina o estado do jogo e a pontuação com base no primeiro lançamento
29       switch (sumOfDice)
30       {
31          case SEVEN: // ganha com 7 no primeiro lançamento
32          case YO_LEVEN: // ganha com 11 no primeiro lançamento
33             gameStatus = Status.WON;
34             break;
35          case SNAKE_EYES: // perde com 2 no primeiro lançamento
36          case TREY: // perde com 3 no primeiro lançamento
37          case BOX_CARS: // perde com 12 no primeiro lançamento
38             gameStatus = Status.LOST;
39             break;
40          default: // não ganhou nem perdeu, portanto registra a pontuação
41             gameStatus = Status.CONTINUE; // jogo não terminou
42             myPoint = sumOfDice; // informa a pontuação
43             System.out.printf("Point is %d%n", myPoint);
44             break;
45       }
46
47       // enquanto o jogo não estiver completo
48       while (gameStatus == Status.CONTINUE) // nem WON nem LOST
49       {
50          sumOfDice = rollDice(); // lança os dados novamente
51
52          // determina o estado do jogo
53          if (sumOfDice == myPoint) // vitória por pontuação
54             gameStatus = Status.WON;
55          else
56             if (sumOfDice == SEVEN) // perde obtendo 7 antes de atingir a pontuação
57                gameStatus = Status.LOST;
58       }
59
60       // exibe uma mensagem ganhou ou perdeu
61       if (gameStatus == Status.WON)
62          System.out.println("Player wins");
63       else
64          System.out.println("Player loses");
65    }
```

```
66
67       // lança os dados, calcula a soma e exibe os resultados
68       public static int rollDice()
69       {
70          // seleciona valores aleatórios do dado
71          int die1 = 1 + randomNumbers.nextInt(6); // primeiro lançamento do dado
72          int die2 = 1 + randomNumbers.nextInt(6); // segundo lançamento do dado
73
74          int sum = die1 + die2; // soma dos valores dos dados
75
76          // exibe os resultados desse lançamento
77          System.out.printf("Player rolled %d + %d = %d%n",
78             die1, die2, sum);
79
80          return sum;
81       }
82    } // fim da classe Craps
```

```
Player rolled 5 + 6 = 11
Player wins
```

```
Player rolled 5 + 4 = 9
Point is 9
Player rolled 4 + 2 = 6
Player rolled 3 + 6 = 9
Player wins
```

```
Player rolled 1 + 2 = 3
Player loses
```

```
Player rolled 2 + 6 = 8
Point is 8
Player rolled 5 + 1 = 6
Player rolled 2 + 1 = 3
Player rolled 1 + 6 = 7
Player loses
```

6.11 Escopo das declarações

```
1  // Figura 6.9: Scope.java
2  // A classe Scope demonstra os escopos de campo e de variável local.
3
4  public class Scope
5  {
6     // campo acessível para todos os métodos dessa classe
7     private static int x = 1;
8
9     // O método main cria e inicializa a variável local x
10    // e chama os métodos useLocalVariable e useField
11    public static void main(String[] args)
12    {
13       int x = 5; // variável local x do método sombreia o campo x
14
15       System.out.printf("local x in main is %d%n", x);
16
17       useLocalVariable(); // useLocalVariable tem uma variável local x
18       useField(); // useField utiliza o campo x da classe Scope
19       useLocalVariable(); // useLocalVariable reinicializa a variável local x
20       useField(); // campo x da classe Scope retém seu valor
21
22       System.out.printf("%nlocal x in main is %d%n", x);
23    }
24
25    // cria e inicializa a variável local x durante cada chamada
26    public static void useLocalVariable()
27    {
28       int x = 25; // inicializada toda vez que useLocalVariable é chamado
29
```

```
30          System.out.printf(
31             "%nlocal x on entering method useLocalVariable is %d%n", x);
32          ++x; // modifica a variável local x desse método
33          System.out.printf(
34             "local x before exiting method useLocalVariable is %d%n", x);
35       }
36
37       // modifica o campo x da classe Scope durante cada chamada
38       public static void useField()
39       {
40          System.out.printf(
41             "%nfield x on entering method useField is %d%n", x);
42          x *= 10; // modifica o campo x da classe Scope
43          System.out.printf(
44             "field x before exiting method useField is %d%n", x);
45       }
46    } // fim da classe Scope
```

```
local x in main is 5

local x on entering method useLocalVariable is 25
local x before exiting method useLocalVariable is 26

field x on entering method useField is 1
field x before exiting method useField is 10

local x on entering method useLocalVariable is 25
local x before exiting method useLocalVariable is 26

field x on entering method useField is 10
field x before exiting method useField is 100

local x in main is 5
```

6.12 Sobrecarga de método

```
1  // Figura 6.10: MethodOverload.java
2  // Declarações de métodos sobrecarregados.
3
4  public class MethodOverload
5  {
6     // teste de métodos square sobrecarregados
7     public static void main(String[] args)
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    }
12
13    // método square com argumento de int
14    public static int square(int intValue)
15    {
16       System.out.printf("%nCalled square with int argument: %d%n",
17          intValue);
18       return intValue * intValue;
19    }
20
21    // método square com argumento double
22    public static double square(double doubleValue)
23    {
24       System.out.printf("%nCalled square with double argument: %f%n",
25          doubleValue);
26       return doubleValue * doubleValue;
27    }
28 } // fim da classe MethodOverload
```

```
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
```

6.13 (Opcional) Estudo de caso de GUIs e imagens gráficas: cores e formas preenchidas

```java
// Figura 6.11: DrawSmiley.java
// Desenhando um rosto sorridente com cores e formas preenchidas.
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;

public class DrawSmiley extends JPanel
{
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);

      // desenha o rosto
      g.setColor(Color.YELLOW);
      g.fillOval(10, 10, 200, 200);

      // desenha os olhos
      g.setColor(Color.BLACK);
      g.fillOval(55, 65, 30, 30);
      g.fillOval(135, 65, 30, 30);

      // desenha a boca
      g.fillOval(50, 110, 120, 60);

      // "retoca" a boca para criar um sorriso
      g.setColor(Color.YELLOW);
      g.fillRect(50, 110, 120, 30);
      g.fillOval(50, 120, 120, 40);
   }
} // fim da classe DrawSmiley
```

```java
// Figura 6.12: DrawSmileyTest.java
// Aplicativo de teste que exibe um rosto sorridente.
import javax.swing.JFrame;

public class DrawSmileyTest
{
   public static void main(String[] args)
   {
      DrawSmiley panel = new DrawSmiley();
      JFrame application = new JFrame();

      application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      application.add(panel);
      application.setSize(230, 250);
      application.setVisible(true);
   }
} // fim da classe DrawSmileyTest
```