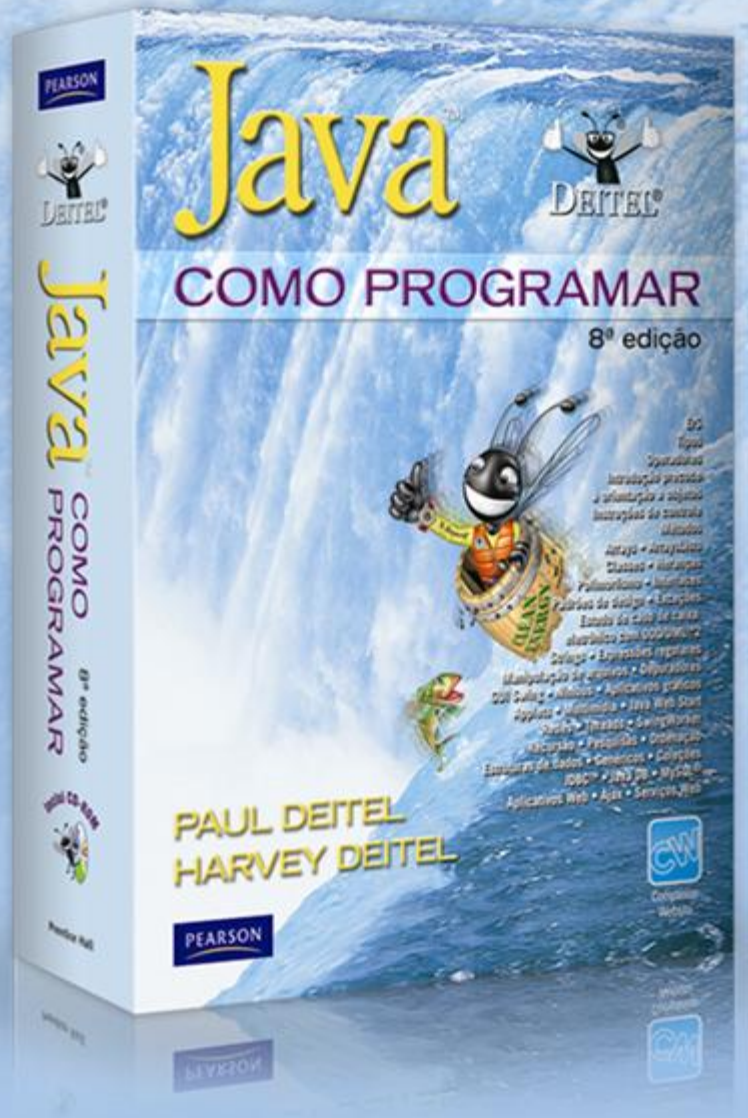
Capítulo 7
Arrays e ArrayLists

Java™ Como Programar, 8/E



OBJETIVOS

Neste capítulo, você aprenderá:

- O que são arrays.
- A utilizar arrays para armazenar dados e recuperá-los de listas e tabelas de valores.
- A declarar arrays, inicializar arrays e referir-se a elementos específicos dos arrays.
- A utilizar a instrução `for` aprimorada para iterar por arrays.
- A passar arrays para métodos.
- A declarar e manipular arrays multidimensionais.
- A escrever métodos que utilizam listas de argumentos de comprimento variável.
- A ler argumentos da linha de comando em um programa.

7.1 Introdução

- **Estruturas de dados**
  - Coleções de itens de dados relacionados.
  - Discutido a fundo nos Capítulos 20–22.
- **Arrays**
  - Estruturas de dados consistindo em itens de dados relacionados do mesmo tipo.
  - Tornam conveniente processar grupos relacionados de valores.
  - Permanecem com o mesmo tamanho depois de criados.
- Declaração `for` aprimorada para iterar por um array ou coleção de itens de dados.
- Listas de argumentos de comprimento variável.
  - Permitem criar métodos com números variáveis de argumentos.
- Processam argumentos de linha de comando no método `main`.

- Manipulações de array comuns com métodos `static` da classe `Arrays` do pacote `java.util`.
- Coleção `ArrayList`.

  Semelhantes a arrays.

  Redimensionamento dinâmico.

  Eles aumentam automático de tamanho em tempo de execução para acomodar elementos adicionais.

7.2 Arrays

- Array
  Grupo de variáveis (chamados **elementos**) contendo valores do mesmo tipo.
  Arrays são objetos, portanto, são tipos por referência.
  Os elementos podem ser tipos primitivos ou tipos por referência.
- Referencia um determinado elemento em um array.
  Use o **índice** dos elementos.
  Expressão de acesso a array — o nome do array seguido pelo índice do elemento particular entre **colchetes, [ ].**
- O primeiro elemento em cada array tem **índice zero.**
- O índice mais alto em array é um a menos que o número de elementos no array.
- Nomes de array seguem as mesmas convenções que outros nomes de variável.

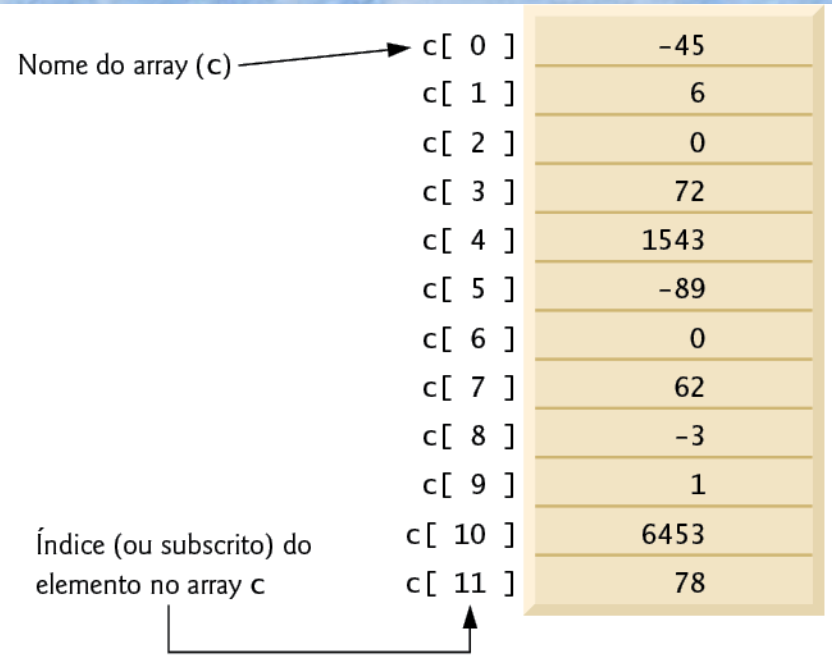


Figura 7.1 | Um array de 12 elementos.

- Um índice deve ser um inteiro não negativo.

  Um programa pode utilizar uma expressão como um índice.
- Um nome de array indexado é uma expressão de acesso ao array.

  Podem ser utilizadas no lado esquerdo de uma atribuição para colocar um novo valor em um elemento de array.
- Cada objeto de array conhece seu próprio tamanho e armazena-o em uma **variável de instância length.**

  `length` não pode ser alterado porque é uma variável `final`.

Java™ COMO PROGRAMAR 8ª edição

Erro comum de programação 7.1

Um índice precisa ser um valor `int` *ou um valor de um tipo que possa ser promovido a* `int` *— a saber,* `byte`, `short` *ou* `char`, *mas não* `long`; *caso contrário, ocorrerá um erro de compilação.*

7.3 Declarando e criando arrays

- Objetos array

  Criados com palavra-chave • `new`.

  Especifica o tipo de elemento e o número de elementos em uma **expressão de criação de array**, que retorna uma referência que pode ser armazenada em uma variável de array.

- Declaração e expressão de criação de arrays de 12 elementos `int`

```
int[] c = new int[ 12 ];
```

- Pode ser realizado em duas etapas como segue:

```
int[] c; // declara a variável de array
c = new int[ 12 ]; // cria o array
```

- Na declaração, os colchetes que se seguem ao tipo indicam que a variável que irá referenciar um array (isto é, a variável armazenará uma referência de array).
- Quando um array é criado, cada elemento do array recebe um valor padrão.

  Zero para os elementos numéricos de tipo primitivo, `false` para elementos `boolean` e `null` para referências • .

Java™ COMO PROGRAMAR 8ª edição

Erro comum de programação 7.2

Em uma declaração de array, especificar o número de elementos entre os colchetes da declaração (por exemplo, `int[12] c;`*) é um erro de sintaxe.*

- Quando o tipo do elemento e os colchetes são combinados no início da declaração, todos os identificadores na declaração são variáveis de array.

  Para legibilidade, declare apenas uma variável por declaração.

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 7.1

Para legibilidade, declare apenas uma variável por declaração. Mantenha cada declaração em uma linha separada e inclua um comentário que descreva a variável sendo declarada.

Java™ COMO PROGRAMAR 8ª edição

Erro comum de programação 7.3

Declarar múltiplas variáveis de array em uma única declaração pode levar a erros sutis. Considere a declaração `int [] a, b, c;`. *Se* `a`, `b` *e* `c` *devem ser declarados como variáveis de array, então essa declaração é correta — colocar os colchetes logo depois do tipo indica que todos os identificadores na declaração são variáveis de array. Entretanto, se apenas* `a` *destina-se a ser uma variável de array, e* `b` *e* `c` *variáveis* `int` *individuais, então essa declaração é incorreta — a declaração* `int a[], b, c;` *alcançaria o resultado desejado.*

- Cada elemento de um array do tipo primitivo contém um valor do tipo de elemento declarado do array.

  Cada elemento de um array `int` é um valor `int`.
- Cada elemento de um array de tipo por referência é uma referência a um objeto do tipo de elemento declarado no array.

  Cada elemento de um array `String` é uma referência a um objeto `String`.

7.4 Exemplos que utilizam arrays

- O aplicativo da Figura 7.2 utiliza a palavra-chave `new` para criar um array de 10 elementos `int`, que são inicialmente zero (o padrão para variáveis `int`).

```
1  // Figura 7.2: InitArray.java
2  // Inicializando os elementos de um array como valores padrão de zero.
3
4  public class InitArray
5  {
6     public static void main( String[] args )
7     {
8        int[] array; // declara o array identificado
9
10       array = new int[ 10 ]; // cria o objeto do array
11
12       System.out.printf( "%s%8s\n", "Index", "Value" ); // títulos de coluna
13
14       // gera saída do valor de cada elemento do array
15       for ( int counter = 0; counter < array.length; counter++ )
16          System.out.printf( "%5d%8d\n", counter, array[ counter ] );
17    } // fim de main
18 } // fim da classe InitArray
```

A variável `array` irá referenciar um array de valores int

Cria um array de 10 elementos `int`, cada um com valor 0 por padrão

A instrução `for` itera enquanto `counter` for menor que o `lenght` do array

A expressão de acesso ao array obtém o valor no índice representado por `counter`

Figura 7.2 | Inicializando os elementos de um array com valores zero padrão. (Parte 1 de 2.)


```
Index   Value
    0       0
    1       0
    2       0
    3       0
    4       0
    5       0
    6       0
    7       0
    8       0
    9       0
```

Figura 7.2 | Inicializando os elementos de um array com valores zero padrão. (Parte 2 de 2.)

- **Inicializador de array.**

  Uma lista de expressões separadas por vírgulas (chamadas **lista de inicializadores**) entre chaves.

  Utilizado para criar um array e inicializar seus elementos.

  O comprimento, ou tamanho, do array é determinado pelo número de elementos na lista inicializadora.

```
int[] n = { 10, 20, 30, 40, 50 };
```

  Cria um array de cinco elementos com valores de índice 0–4.

- O compilador conta o número de inicializadores na lista para determinar o tamanho do array.

  Configura a operação `new` apropriada "nos bastidores".


```
1  // Figura 7.3: InitArray.java
2  // Inicializando os elementos de um array com um inicializador de array.
3
4  public class InitArray
5  {
6     public static void main( String[] args )
7     {
8        // lista de inicializador especifica o valor de cada elemento
9        int[] array = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
10
11       System.out.printf( "%s%8s\n", "Index", "Value" ); // títulos de coluna
12
13       // gera saída do valor de cada elemento do array
14       for ( int counter = 0; counter < array.length; counter++ )
15          System.out.printf( "%5d%8d\n", counter, array[ counter ] );
16    } // fim de main
17 } // fim da classe InitArray
```

Lista inicializadora de array para um array de 10 elementos `int`

Figura 7.3 | Inicializando os elementos de um array com um inicializador de array. (Parte 1 de 2.)


```
Index   Value
    0      32
    1      27
    2      64
    3      18
    4      95
    5      14
    6      90
    7      70
    8      60
    9      37
```

Figura 7.3 | Inicializando os elementos de um array com um inicializador de array. (Parte 2 de 2.)

- O aplicativo na Figura 7.4 cria um array de 10 elementos e atribui a cada elemento um dos inteiros pares de 2 a 20 (2, 4, 6, …, 20).


```
1  // Figura 7.4: InitArray.java
2  // Calculando valores a serem colocados em elementos de um array.
3
4  public class InitArray
5  {
6     public static void main( String[] args )
7     {
8        final int ARRAY_LENGTH = 10; // declara constante
9        int[] array = new int[ ARRAY_LENGTH ]; // cria o array
10
11       // calcula o valor de cada elemento do array
12       for ( int counter = 0; counter < array.length; counter++ )
13          array[ counter ] = 2 + 2 * counter;
14
15       System.out.printf( "%s%8s\n", "Index", "Value" ); // títulos de coluna
16
17       // gera a saída do valor de cada elemento do array
18       for ( int counter = 0; counter < array.length; counter++ )
19          System.out.printf( "%5d%8d\n", counter, array[ counter ] );
20    } // fim de main
21 } // fim da classe InitArray
```

Constante nomeada representando o tamanho do array

Cria um array com ARRAY_LENGHT elementos

Calcula e armazena um valor para cada elemento do array

Figura 7.4 | Calculando os valores a serem colocados nos elementos de um array. (Parte 1 de 2.)


```
Index   Value
    0       2
    1       4
    2       6
    3       8
    4      10
    5      12
    6      14
    7      16
    8      18
    9      20
```

Figura 7.4 | Calculando os valores a serem colocados nos elementos de um array. (Parte 2 de 2.)

- Variáveis `final` devem ser inicializadas antes de serem utilizadas e não podem ser modificadas depois.
- Tentar modificar uma variável `final` depois que é ela inicializada causa um erro de compilação.

  `cannot assign a value to final variable` *nomeDaVariável*
- Tentar acessar o valor de uma variável `final` depois que é ela inicializada causa um erro de compilação.

  `variable` *nomeDaVariável* `might not have been initialized`

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 7.2

Variáveis constantes também são chamadas ***constantes nomeadas****. Frequentemente, elas tornam os programas mais legíveis que os programas que utilizam valores literais (por exemplo,* 10*) — uma constante identificada como* ARRAY_LENGTH *indica claramente seu propósito, enquanto um valor literal poderia ter diferentes significados com base em seu contexto.*

Java™ COMO PROGRAMAR 8ª edição

Erro comum de programação 7.4

Atribuir um valor a uma variável constante depois de ela ter sido inicializada é um erro de compilação.

Java™ COMO PROGRAMAR 8ª edição

Erro comum de programação 7.5

Tentar utilizar uma constante antes de ela ser inicializada é um erro de compilação.

- A Figura 7.5 soma os valores contidos em um array de 10 elementos inteiros.
- Os elementos de um array costumam representar uma série de valores a ser utilizados em um cálculo.

```
1  // Figura 7.5: SumArray.java
2  // Calculando a soma dos elementos de um array.
3
4  public class SumArray
5  {
6     public static void main( String[] args )
7     {
8        int[] array = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
9        int total = 0;
10
11       // adiciona o valor de cada elemento ao total
12       for ( int counter = 0; counter < array.length; counter++ )
13          total += array[ counter ];
14
15       System.out.printf( "Total of array elements: %d\n", total );
16    } // fim de main
17 } // fim da classe SumArray
```

Adiciona cada valor em `array` a `total`, o qual é exibido quando o loop termina

```
Total of array elements: 849
```

Figura 7.5 | Calculando a soma dos elementos de um array.

- Muitos programas apresentam dados graficamente aos usuários.
- Valores numéricos costumam ser exibidos como barras em um gráfico de barras.

  Barras mais longas representam valores numéricos proporcionalmente maiores.
- Uma maneira simples de exibir os dados numéricos graficamente é utilizar um gráfico de barras que mostra cada valor numérico como uma barra de asteriscos (*).
- O especificador de formato `%02d` indica que um valor `int` deve ser formatado como um campo de dois dígitos.

  O **flag 0** exibe um `0` inicial para valores com menos dígitos do que a largura do campo (2).

```
1  // Figura 7.6: BarChart.java
2  // programa de impressão de gráfico de barras.
3
4  public class BarChart
5  {
6     public static void main( String[] args )
7     {
8        int[] array = { 0, 0, 0, 0, 0, 0, 1, 2, 4, 2, 1 };
9
10       System.out.println( "Grade distribution:" );
11
12       // para cada elemento de array, gera saída de uma barra do gráfico
13       for ( int counter = 0; counter < array.length; counter++ )
14       {
15          // imprime rótulo de barra ( "00-09: ", ..., "90-99: ", "100: " )
16          if ( counter == 10 )
17             System.out.printf( "%5d: ", 100 );
18          else
19             System.out.printf( "%02d-%02d: ",
20                counter * 10, counter * 10 + 9  );
21
```

Figura 7.6 | Programa de impressão de gráfico de barras. (Parte 1 de 2.)

```
22             // imprime a barra de asteriscos
23             for ( int stars = 0; stars < array[ counter ]; stars++ )
24                System.out.print( "*" );
25
26             System.out.println(); // inicia uma nova linha de saída
27          } // fim do for externo
28       } // fim de main
29    } // fim da classe BarChart
```

Loop for aninhado utiliza a variável `counter` do loop `for` mais externo para determinar qual elemento do array acessar e, então, exibe o número apropriado de asteriscos

```
Grade distribution:
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
  100: *
```

Figura 7.6 | Programa de impressão de gráfico de barras. (Parte 2 de 2.)

- Às vezes, os programas utilizam variáveis contadoras para resumir dados, como os resultados de uma pesquisa.
- Na Figura 6.8, utilizamos os contadores separados em nosso programa de lançamento de dados para monitorar o número de ocorrências de cada face de um dado de seis faces quando o programa lançou o dado 6.000 vezes.
- Uma versão de array desse aplicativo é mostrada na Figura 7.7.

  A linha 14 deste programa substitui as linhas 23–46 da Figura 6.8.
- O array `frequency` deve ser grande o bastante para armazenar seis contadores.

  Utilizamos um array de sete elementos no qual ignoramos `frequency[0]`.

  É mais lógico fazer o valor nominal 1 incrementar `frequency[1]` do que `frequency[0]`.


```
1  // Figura 7.7: RollDie.java
2  // Programa de jogo de dados utilizando arrays em vez de switch
3  import java.util.Random;
4
5  public class RollDie
6  {
7     public static void main( String[] args )
8     {
9        Random randomNumbers = new Random(); // gerador de número aleatório
10       int[] frequency = new int[ 7 ]; // array de contadores de frequência
11
12       // lança o dados 6000 vezes; usa o valor do dado como índice de frequência
13       for ( int roll = 1; roll <= 6000; roll++ )
14          ++frequency[ 1 + randomNumbers.nextInt( 6 ) ];
15
16       System.out.printf( "%s%10s\n", "Face", "Frequency" );
17
18       // gera saída do valor de cada elemento do array
19       for ( int face = 1; face < frequency.length; face++ )
20          System.out.printf( "%4d%10d\n", face, frequency[ face ] );
21    } // fim de main
22 } // fim da classe RollDie
```

Número aleatório de 1 a 6 é utilizado como índice no array `frequency` para determinar qual elemento incrementar

Figura 7.7 | Programa de jogo de dados utilizando arrays em vez de `switch`. (Parte 1 de 2.)


```
Face  Frequency
   1        988
   2        963
   3       1018
   4       1041
   5        978
   6       1012
```

Figura 7.7 | Programa de jogo de dados utilizando arrays em vez de `switch`. (Parte 2 de 2.)

- A Figura 7.8 utiliza arrays para resumir os resultados dos dados coletados em uma pesquisa:

  Foi pedido a quarenta alunos para avaliar a qualidade da comida na cantina estudantil em uma escala de 1 a 10 (onde 1 significa péssimo e 10, excelente). Coloque as 40 respostas em um array de inteiros e resuma os resultados da enquete.
- O array responses é um array int de 40 elementos das respostas da enquete.
- O array frequency de 11 elementos conta o número de ocorrências de cada resposta (1 a 10).

  Cada elemento é inicializado em zero por padrão.

  Ignoramos frequency[0].


```
1  // Figura 7.8: StudentPoll.java
2  // Programa de análise de enquete.
3
4  public class StudentPoll
5  {
6     public static void main( String[] args )
7     {
8        // array de respostas da enquete
9        int[] responses = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10, 1, 6, 3, 8, 6,
10          10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6, 5, 6, 7, 5, 6,
11          4, 8, 6, 8, 10 };
12       int[] frequency = new int[ 11 ]; // array de contadores de frequência
13
14       // para cada resposta, seleciona elemento da resposta e usa esse valor
15       // como índice de frequency para determinar o elemento a incrementar
16       for ( int answer = 0; answer < responses.length; answer++ )
17          ++frequency[ responses[ answer ] ];
18
19       System.out.printf( "%s%10s", "Rating", "Frequency" );
20
21       // gera saída do valor de cada elemento do array
22       for ( int rating = 1; rating < frequency.length; rating++ )
23          System.out.printf( "%d%10d", rating, frequency[ rating ] );
24    } // fim de main
25 } // fim da classe StudentPoll
```

O programa precisa saber apenas 10 contadores; ignoramos o elemento 0

Incrementa o elemento apropriado de `frequency` com base no valor de `responses[answer]`

Figura 7.8 | Programa de análise de enquete. (Parte 1 de 2.)


```
Rating Frequency
     1         2
     2         2
     3         2
     4         2
     5         5
     6        11
     7         5
     8         7
     9         1
    10         3
```

Figura 7.8 | Programa de análise de enquete. (Parte 2 de 2.)

- Se os dados no array `responses` contivessem valores inválidos, como 13, o programa teria tentado adicionar `1` a `frequency[13]`, que está fora dos limites do array.

  O Java não permite isso.

  Quando um programa Java executa, a JVM verifica índices de array para assegurar que eles são maiores que ou igual a 0 e menor que o comprimento do array — isso é chamado **verificação de limites**.

  Se um programa utilizar um índice inválido, o Java gera uma suposta exceção para indicar a ocorrência de um erro no programa em tempo de execução.

Java™ COMO PROGRAMAR 8ª edição

Dica de prevenção de erro 7.1

Uma exceção indica a ocorrência de um erro em um programa. Você pode escrever um código para recuperar-se de uma exceção e continuar a execução do programa, em vez de terminar o programa anormalmente. Quando um programa tenta acessar um elemento fora dos limites do array, ocorree uma `ArrayIndexOutOfBoundsException`. *O tratamento de exceções é discutido no Capítulo 11.*

Java™ COMO PROGRAMAR 8ª edição

Dica de prevenção de erro 7.2

Ao escrever um código para fazer loop por um array, assegure que o índice de array é sempre maior que ou igual a 0 e menor que o comprimento do array. A condição de continuação do loop deve impedir o acesso a elementos fora desse intervalo.

7.5 Estudo de caso: simulação de embaralhamento e distribuição de cartas

- Os exemplos até aqui utilizaram arrays contendo elementos de tipos primitivos.
- Os elementos de um array podem ser tipos primitivos ou tipos por referência.
- O próximo exemplo utiliza um array de elementos de tipo por referência — objetos que representam cartas de baralho — para desenvolver uma classe que simula o embaralhamento e distribuição das cartas.

- A classe `Card` (Figura 7.9) contém duas variáveis de instância `String` — `face` e `suit` — que são utilizadas para armazenar referências ao nome da face e o nome do naipe de uma carta (`Card`) específica.
- O método `toString` cria uma `String` consistindo na `face` da `carta`, `" de "` e o `naipe` da carta.

  Pode ser invocado explicitamente para obter uma representação uma `Card`.

  Chamado implicitamente quando o objeto é usado onde uma `String` é esperada.


```
1  // Figura 7.9: Card.java
2  // Classe Card representa uma carta de baralho.
3
4  public class Card
5  {
6     private String face; // face da carta ("Ace", "Deuce", ...)
7     private String suit; // naipe da carta ("Hearts", "Diamonds", ...)
8
9     // construtor de dois argumentos inicializa face e naipe da carta
10    public Card( String cardFace, String cardSuit )
11    {
12       face = cardFace; // inicializa face da carta
13       suit = cardSuit; // inicializa naipe da carta
14    } // fim do construtor Card de dois argumentos
15
16    // retorna representação String de Card
17    public String toString()
18    {
19       return face + " of " + suit;
20    } // fim do método toString
21 } // fim da classe Card
```

Deve ser declarado com a primeira linha; é para ser chamado implicitamente a fim de converter objetos `Card` em representações `String`

Figura 7.9 | A classe `Card` representa uma carta de baralho.

- A classe `DeckOfCards` (Figura 7.10) declara como uma variável de instância um array `Card` chamado `deck`.
- Os elementos são `Deck` são `null` por padrão.
  O construtor preenche o array `deck` com objetos `Card`.
- O método `shuffle` embaralha as `Card`s do baralho.
  O método faz um loop por todas as 52 `Card`s (índices de array 0 a 51).
  Cada `Card` é trocada por outra carta aleatoriamente escolhida no baralho.
- O método `dealCard` distribui uma `Card` no array.
  `currentCard` indica o índice da próxima `Card` a ser distribuída.
  Retorna `null` se não houver mais cartas a distribuir.


```
1  // Figura 7.10: DeckOfCards.java
2  // classe DeckOfCards representa um baralho.
3  import java.util.Random;
4
5  public class DeckOfCards
6  {
7     private Card[] deck; // array de objetos Card
8     private int currentCard; // índice do próximo Card a ser distribuído
9     private static final int NUMBER_OF_CARDS = 52; // número constante de Cards
10    // gerador de número aleatório
11    private static final Random randomNumbers = new Random();
12
13    // construtor preenche baralho de cartas
14    public DeckOfCards()
15    {
16       String[] faces = { "Ace", "Deuce", "Three", "Four", "Five", "Six",
17          "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King" };
18       String[] suits = { "Hearts", "Diamonds", "Clubs", "Spades" };
19
20       deck = new Card[ NUMBER_OF_CARDS ]; // cria array de objetos Card
21       currentCard = 0; // configura currentCard, então o 1o. Card distribuído é deck[ 0 ]
22
```

Declara o array de `Cars`; `deck` é `null` até que o array seja criado

Cria o array de variáveis `Card`

Figura 7.10 | A classe `DeckOfCards` representa um baralho de cartas. (Parte 1 de 3.)


```
23          // preenche baralho com objetos Card
24          for ( int count = 0; count < deck.length; count++ )
25             deck[ count ] =
26                new Card( faces[ count % 13 ], suits[ count / 13 ] );
27       } // fim do construtor DeckOfCards
28
29       // embaralha as cartas com um algoritmo de uma passagem
30       public void shuffle()
31       {
32          // depois de embaralhar, a distribuição deve iniciar em deck[ 0 ] novamente
33          currentCard = 0; // reinicializa currentCard
34
35          // para cada Card, seleciona outro Card aleatório e os compara
36          for ( int first = 0; first < deck.length; first++ )
37          {
38             // seleciona um número aleatório entre 0 e 51
39             int second =  randomNumbers.nextInt( NUMBER_OF_CARDS );
40
41             // compara Card atual com Card aleatoriamente selecionado
42             Card temp = deck[ first ];
43             deck[ first ] = deck[ second ];
44             deck[ second ] = temp;
45          } // fim do for
```

Cria um Card para o elemento atual do array

Permuta o elemento atual com o elemento aleatoriamente selecionado

Figura 7.10 | A classe DeckOfCards representa um baralho de cartas. (Parte 2 de 3.)


```
46       } // fim do método shuffle
47
48       // distribui um Card
49       public Card dealCard()
50       {
51          // determina se ainda há Cards a serem distribuídos
52          if (currentCard < deck.length )
53             return deck[ currentCard++ ]; // retorna Card atual no array
54          else
55             return null; // retorna nulo para indicar que todos os Cards foram distribuídos
56       } // fim do método dealCard
57    } // fim da classe DeckOfCards
```

Assegura que `currentCard` é menor que o `lenght` do array; se for, um `Card` é retornado; caso contrário `null` é retornado

Figura 7.10 | A classe `DeckOfCards` representa um baralho de cartas. (Parte 3 de 3.)

- A Figura 7.11 demonstra a classe `DeckOfCards` (Figura 7.10).
- Quando uma `Card` é enviada para a saída como uma `String`, o método `toString` de `Card` é implicitamente invocado.


```
1  // Figura 7.11: DeckOfCardsTest.java
2  // Embaralhando e distribuindo cartas.
3
4  public class DeckOfCardsTest
5  {
6     // executa o aplicativo
7     public static void main( String[] args )
8     {
9        DeckOfCards myDeckOfCards = new DeckOfCards();
10       myDeckOfCards.shuffle(); // coloca Cards em ordem aleatória
11
12       // imprime todas as 52 cartas na ordem em que são distribuídas
13       for ( int i = 1; i <= 52; i++ )
14       {
15          // distribui e exibe uma Card
16          System.out.printf( "%-19s", myDeckOfCards.dealCard() );
17
18          if ( i % 4 == 0 ) // imprime uma nova linha a cada 4 cartas
19             System.out.println();
20       } // fim do for
21    } // fim do main
22 } // fim da classe DeckOfCardsTest
```

Distribui um `Card`; o método `toString` de `Card` é chamado para obter implicitamente a representação `String` que é enviada para a saída

Figura 7.11 | Embaralhando e distribuindo cartas. (Parte 1 de 2.)


```
Six of Spades      Eight of Spades    Six of Clubs       Nine of Hearts
Queen of Hearts    Seven of Clubs     Nine of Spades     King of Hearts
Three of Diamonds  Deuce of Clubs     Ace of Hearts      Ten of Spades
Four of Spades     Ace of Clubs       Seven of Diamonds  Four of Hearts
Three of Clubs     Deuce of Hearts    Five of Spades     Jack of Diamonds
King of Clubs      Ten of Hearts      Three of Hearts    Six of Diamonds
Queen of Clubs     Eight of Diamonds  Deuce of Diamonds  Ten of Diamonds
Three of Spades    King of Diamonds   Nine of Clubs      Six of Hearts
Ace of Spades      Four of Diamonds   Seven of Hearts    Eight of Clubs
Deuce of Spades    Eight of Hearts    Five of Hearts     Queen of Spades
Jack of Hearts     Seven of Spades    Four of Clubs      Nine of Diamonds
Ace of Diamonds    Queen of Diamonds  Five of Clubs      King of Spades
Five of Diamonds   Ten of Clubs       Jack of Spades     Jack of Clubs
```

Figura 7.11 | Embaralhando e distribuindo cartas. (Parte 2 de 2.)

7.6 A estrutura `for` aprimorada

- **Instrução `for` aprimorada**
  Itera pelos elementos de um array sem utilizar um contador.
  Evita a possibilidade de ultrapassar o limite do array.
  Também funciona com as coleções predefinidas da Java API (veja a Seção 7.14).
- Sintaxe:

```
for ( parâmetro : nomeDoArray )
    instrução
```

  onde *parâmetro* tem um tipo e um identificador e *arrayName* é o array pelo qual iterar.
- O tipo do parâmetro deve ser consistente com o tipo de elemento no array.
- A instrução `for` aprimorada simplifica o código para iterar por um array.

```
1  // Figura 7.12: EnhancedForTest.java
2  // Usando instrução for aprimorada para somar inteiros em um array.
3
4  public class EnhancedForTest
5  {
6     public static void main( String[] args )
7     {
8        int[] array = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
9        int total = 0;
10
11       // adiciona o valor de cada elemento ao total
12       for ( int number : array )
13          total += number;
14
15       System.out.printf( "Total of array elements: %d\n", total );
16    } // fim do main
17 } // fim da classe EnhancedForTest
```

Para cada elemento em `array`, atribui o valor do elemento a `number`; então, adiciona `number` a `total`

```
Total of array elements: 849
```

Figura 7.12 | Utilizando a instrução `for` aprimorada para somar inteiros em um array.

A instrução `for` aprimorada simplifica o código para iterar por um array.

- A instrução `for` aprimorada somente pode ser utilizada para obter elementos do array

  Ela não pode ser utilizada para modificar elementos em um array.

  Para modificar elementos, utilize a instrução `for` tradicional.
- Pode ser utilizada no lugar da instrução `for` loop controlado por contador se você não precisar acessar o índice do elemento.

7.7 Passando arrays para métodos

- Para passar um argumento de array para um método, especifique o nome do array sem nenhum colchete.

  Como cada objeto array "sabe" seu próprio tamanho, não temos de passar o tamanho do array como um argumento adicional.
- Para receber um array, a lista de parâmetros do método deve especificar um parâmetro de array.
- Quando um argumento para um método for um array inteiro ou um elemento de array individual de um tipo por referência, o método chamado recebe uma cópia da referência.
- Quando um argumento para um método é um elemento de array individual de um tipo primitivo, o método chamado recebe uma cópia do valor do elemento.

  Esses valores primitivos são chamados **escalares** ou **quantidades escalares**.


```
1  // Figura 7.13: PassArray.java
2  // Passando arrays e elementos de arrays individuais aos métodos.
3
4  public class PassArray
5  {
6     // main cria array e chama modifyArray e modifyElement
7     public static void main( String[] args )
8     {
9        int[] array = { 1, 2, 3, 4, 5 };
10
11       System.out.println(
12          "Effects of passing reference to entire array:\n" +
13          "The values of the original array are:" );
14
15       // gera saída de elementos do array original
16       for ( int value : array )
17          System.out.printf( "   %d", value );
18
19       modifyArray( array ); // passa a referência do array
20       System.out.println( "\n\nThe values of the modified array are:" );
21
22       // gera saída de elementos do array modificado
23       for ( int value : array )
24          System.out.printf( "   %d", value );
```

Passa a referência a `array` dentro do método `modifyArray`

Figura 7.13 | Passando arrays e elementos de arrays individuais para métodos. (Parte 1 de 3.)

```
25
26          System.out.printf(
27             "\n\nEffects of passing array element value:\n" +
28             "array[3] before modifyElement: %d\n", array[ 3 ] );
29
30          modifyElement( array[ 3 ] ); // tenta modificar o array[ 3 ]
31          System.out.printf(
32             "array[3] after modifyElement: %d\n", array[ 3 ] );
33       } // fim de main
34
35       // multiplica cada elemento de um array por 2
36       public static void modifyArray( int[] array2 )
37       {
38          for ( int counter = 0; counter < array2.length; counter++ )
39             array2[ counter ] *= 2;
40       } // fim do método modifyArray
41
42       // multiplica argumento por 2
43       public static void modifyElement( int element )
44       {
45          element *= 2;
46          System.out.printf(
47             "Value of element in modifyElement: %d\n", element );
48       } // fim do método modifyElement
49    } // fim da classe PassArray
```

Passa uma cópia do valor `int` de `array[3]` para o `modifyElement`

O método recebe uma cópia da referência de um array, que dá ao método acesso direto ao array original na memória

O método recebe uma cópia de um avalor `int`; o método não pode modificar o valor `int` original em `main`

Figura 7.13 | Passando arrays e elementos de arrays individuais para métodos. (Parte 2 de 3.)

```
Effects of passing reference to entire array:
The values of the original array are:
   1   2   3   4   5

The values of the modified array are:
   2   4   6   8   10

Effects of passing array element value:
array[3] before modifyElement: 8
Value of element in modifyElement: 16
array[3] after modifyElement: 8
```

Figura 7.13 | Passando arrays e elementos de arrays individuais para métodos. (Parte 3 de 3.)

- Passagem por valor (também chamada **chamada por valor**).

  Uma cópia do *valor* do argumento é passada para o método chamador.

  O método chamado trabalha exclusivamente com a cópia.

  As alterações na cópia do método chamado não afetam o valor da variável original no chamador.
- Passagem por referência (também chamada **chamada por referência**).

  O método chamado pode acessar o valor do argumento no chamador diretamente e modificar esses dados, se necessário.

  Aprimora o desempenho eliminando a necessidade de copiar quantidades de dados possivelmente grandes.

- Em Java, todos os argumentos são passados por valor.
- Uma chamada de método pode passar dois tipos de valores a um método.
  Cópias de valores primitivos.
  Cópias de referências a objetos.
- Os objetos em si não podem ser passados para os métodos.
- Se um método modificar um parâmetro de tipo por referência para que ele referencie outro objeto, somente o parâmetro referencia o novo objeto.
  A referência armazenada na variável do chamador ainda referencia o objeto original.
- Embora uma referência do objeto seja passada por valor, um método ainda pode interagir com o objeto referenciado chamando seus métodos `public` que utilizam a cópia da referência do objeto.
  O parâmetro no método chamado e o argumento no método chamador referenciam o mesmo objeto na memória.

Java™ COMO PROGRAMAR 8ª edição

Dica de desempenho 7.1

Passar arrays por referência faz sentido por razões de desempenho. Se arrays fossem passados por valor, uma cópia de cada elemento seria passada. Para arrays grandes e frequentemente passados, isso desperdiçaria tempo e consumiria considerável capacidade de armazenamento para as cópias dos arrays.

7.8 Estudo de caso: classe GradeBook utilizando um array para armazenar notas

- Versões anteriores de classe `GradeBook` processavam um conjunto de notas inserido pelo usuário, mas não mantinham os valores das notas individuais em variáveis de instância da classe.

  A repetição dos cálculos exige que o usuário insira as mesmas notas novamente.

  Resolvemos esse problema armazenando as notas escolares em um array.
- O tamanho do array `grades` é determinado pelo tamanho do array que é passado para o construtor. Portanto, um objeto `GradeBook` pode processar um número variável de notas.

```
1  // Figura 7.14: GradeBook.java
2  // classe GradeBook usando um array para armazenar notas de teste.
3
4  public class GradeBook
5  {
6     private String courseName; // nome do curso que essa GradeBook representa
7     private int[] grades; // array de notas de aluno
8
9     // construtor de dois argumentos inicializa courseName e o array de notas
10    public GradeBook( String name, int[] gradesArray )
11    {
12       courseName = name; // inicializa courseName
13       grades = gradesArray; // níveis de armazenamento
14    } // fim do construtor GradeBook de dois argumentos
15
16    // método para configurar o nome do curso
17    public void setCourseName( String name )
18    {
19       courseName = name; // armazena o nome do curso
20    } // fim do método setCourseName
21
```

Irá referenciar um array passado pelo criador de GradeBook

Recebe o array do criador de GradeBook

Inicializa a variável de instância grades

Figura 7.14 | Classe GradeBook utilizando um array para armazenar notas de teste. (Parte 1 de 7.)


```
22       // método para recuperar o nome do curso
23       public String getCourseName()
24       {
25          return courseName;
26       } // fim do método getCourseName
27
28      // exibe uma mensagem de boas-vindas para o usuário GradeBook
29       public void displayMessage()
30       {
31          // getCourseName obtém o nome do curso
32          System.out.printf( "Welcome to the grade book for\n%s!\n\n",
33             getCourseName() );
34       } // fim do método displayMessage
35
36       // realiza várias operações nos dados
37       public void processGrades()
38       {
39          // gera saída de array de notas
40          outputGrades();
41
42          // chama método getAverage para calcular a nota média
43          System.out.printf( "\nClass average is %.2f\n", getAverage() );
44
```

Figura 7.14 | Classe GradeBook utilizando um array para armazenar notas de teste. (Parte 2 de 7.)


```
45          // chama métodos getMinimum e getMaximum
46          System.out.printf( "Lowest grade is %d\nHighest grade is %d\n\n",
47             getMinimum(),getMaximum() );
48
49          // chama outputBarChart para imprimir gráfico de distribuição de notas
50          outputBarChart();
51       } // fim do método processGrades
52
53       // localiza nota mínima
54       public int getMinimum()
55       {
56          int lowGrade = grades[ 0 ]; // assume que grades[ 0 ] é a menor nota
57
58          // faz um loop pelo array de notas
59          for ( int grade : grades )
60          {
61             // se nota for mais baixa que lowGrade, essa nota é atribuída a lowGrade
62             if ( grade < lowGrade )
63                lowGrade = grade; // nova nota mais baixa
64          } // fim do for
65
66          return lowGrade; // retorna nota mais baixa
67       } // fim do método getMinimum
68
```

Figura 7.14 | Classe GradeBook utilizando um array para armazenar notas de teste. (Parte 3 de 7.)


```
69      // localiza nota máxima
70      public int getMaximum()
71      {
72         int highGrade = grades[ 0 ]; // assume que grades[ 0 ] é a maior nota
73
74         // faz um loop pelo array de notas
75         for ( int grade : grades )
76         {
77            // se a nota for maior que highGrade, atribui essa nota a highGrade
78            if ( grade > highGrade )
79               highGrade = grade; // nova nota mais alta
80         } // fim de for
81
82         return highGrade; // retorna nota mais alta
83      } // fim do método getMaximum
84
```

Figura 7.14 | Classe GradeBook utilizando um array para armazenar notas de teste. (Parte 4 de 7.)

```
85      // determina média para o teste
86      public double getAverage()
87      {
88         int total = 0; // inicializa o total
89
90         // soma notas de um aluno
91         for ( int grade : grades )
92            total += grade;
93
94         // retorna média de notas
95         return (double) total / grades.length;
96      } // fim do método getAverage
97
98      // gera a saída do gráfico de barras exibindo distribuição de notas
99      public void outputBarChart()
100     {
101        System.out.println( "Grade distribution:" );
102
103        // armazena frequência de notas em cada intervalo de 10 notas
104        int[] frequency = new int[ 11 ];
105
106        // para cada nota, incrementa a frequência apropriada
107        for ( int grade : grades )
108           ++frequency[ grade / 10 ];
```

Cálculo é baseado no tamanho do array usado para inicializar o GradeBook

Figura 7.14 | Classe GradeBook utilizando um array para armazenar notas de teste. (Parte 5 de 7.)

```
109
110         // para cada frequência de nota, imprime uma barra no gráfico
111         for ( int count = 0; count < frequency.length; count++ )
112         {
113            // imprime rótulo de barra ( "00-09: ", …, "90-99: ", "100: " )
114            if ( count == 10 )
115               System.out.printf( "%5d: ", 100 );
116            else
117               System.out.printf( "%02d-%02d: ",
118                  count * 10, count * 10 + 9  );
119
120            // imprime a barra de asteriscos
121            for ( int stars = 0; stars < frequency[ count ]; stars++ )
122               System.out.print( "*" );
123
124            System.out.println(); // inicia uma nova linha de saída
125         } // fim do for externo
126      } // fim do método outputBarChart
127
```

Figura 7.14 | Classe GradeBook utilizando um array para armazenar notas de teste. (Parte 6 de 7.)


```
128       // gera a saída do conteúdo do array de notas
129       public void outputGrades()
130       {
131          System.out.println( "The grades are:\n" );
132
133          // gera a saída da nota de cada aluno
134          for ( int student = 0; student < grades.length; student++ )
135             System.out.printf( "Student %2d: %3d\n",
136                student + 1, grades[ student ] );
137       } // fim do método outputGrades
138    } // fim da classe GradeBook
```

Figura 7.14 | Classe GradeBook utilizando um array para armazenar notas de teste. (Parte 7 de 7.)

- O aplicativo da Figura 7.15 cria um objeto da classe `GradeBook` (Figura 7.14) usando o array `int grades-Array`.
- As linhas 12–13 passam um nome de curso e o `gradesArray` para o construtor `GradeBook`.

Observação de engenharia de software 7.1

Um aplicativo de teste é responsável por criar um objeto da classe sendo testado e fornecer-lhe dados. Esses dados poderiam vir de qualquer uma das várias fontes. Os dados de teste podem ser colocados diretamente em um array com um inicializador de array, podem vir do teclado, de um arquivo (como você verá no Capítulo 17) ou de uma rede (como você verá no Capítulo 27). Depois de passar esses dados para o construtor da classe para instanciar o objeto, o aplicativo de teste deve chamar o objeto para testar seus métodos e manipular seus dados. Reunir os dados em um aplicativo de teste como esse permite à classe manipular dados de várias fontes.


```
1  // Figura 7.15: GradeBookTest.java
2  // GradeBookTest cria um objeto GradeBook utilizando um array de notas,
3  // então invoca o método processGrades para analisá-las.
4  public class GradeBookTest
5  {
6     // método main inicia a execução de programa
7     public static void main( String[] args )
8     {
9        // array de notas de aluno
10       int[] gradesArray = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
11
12       GradeBook myGradeBook = new GradeBook(
13          "CS101 Introduction to Java Programming", gradesArray );
14       myGradeBook.displayMessage();
15       myGradeBook.processGrades();
16    } // fim de main
17 } // fim da classe GradeBookTest
```

Inicializa GradeBook com o array gradesArray

Figura 7.15 | GradeBookTest cria um objeto GradeBook usando um array de notas, então invoca o método processGrades para analisá-los. (Parte 1 de 3.)


```
Welcome to the grade book for
CS101 Introduction to Java Programming!

The grades are:

Student  1:  87
Student  2:  68
Student  3:  94
Student  4: 100
Student  5:  83
Student  6:  78
Student  7:  85
Student  8:  91
Student  9:  76
Student 10:  87

Class average is 84.90
Lowest grade is 68
Highest grade is 100
```

Figura 7.15 | `GradeBookTest` cria um objeto `GradeBook` usando um array de notas, então invoca o método `processGrades` para analisá-los. (Parte 2 de 3.)

```
Grade distribution:
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
  100: *
```

Figura 7.15 | `GradeBookTest` cria um objeto `GradeBook` usando um array de notas, então invoca o método `processGrades` para analisá-los. (Parte 3 de 3.)

7.9 Arrays multidimensionais

- **Arrays bidimensionais** costumam ser utilizados para representar tabelas de valores consistindo em informações organizadas em linhas e colunas.
- Identificam um elemento particular com dois índices.

  Por convenção, o primeiro identifica a linha do elemento e o segundo sua coluna.
- Arrays multidimensionais podem ter mais de duas dimensões.
- O Java não suporta arrays multidimensionais diretamente.

  Permite especificar arrays unidimensionais cujos elementos também são arrays unidimensionais, alcançando assim o mesmo efeito.
- Em geral, um array com *m* linhas e *n* colunas é chamado de **array *m* por *n*.**

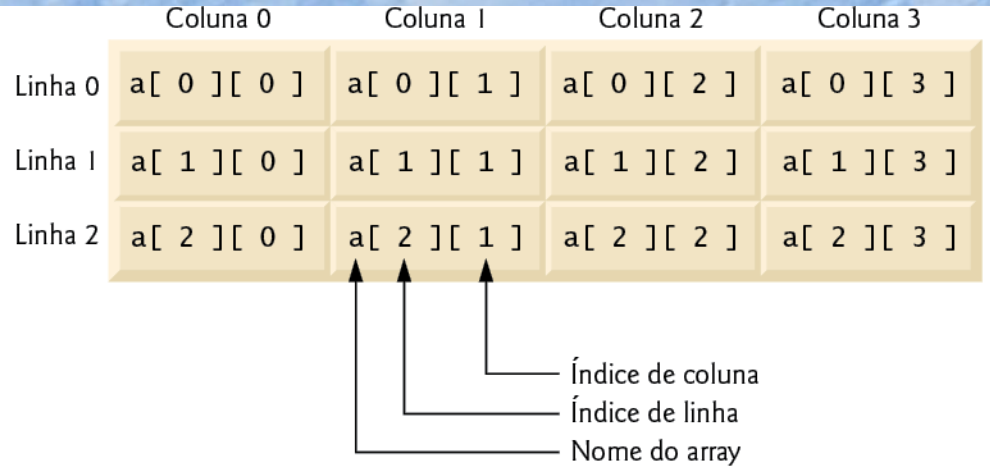


Figura 7.16 | O array bidimensional com três linhas e quatro colunas.

- Arrays multidimensionais podem ser inicializados com inicializadores de array em declarações.
- Um array bidimensional `b` com duas linhas e duas colunas poderia ser declarado e inicializado com **inicializadores de array aninhados** como a seguir:

```
int[][] b = { { 1, 2 }, { 3, 4 } };
```

Os valores iniciais são agrupados por linha entre chaves.

O número de inicializadores de array aninhados (representado por conjuntos de chaves dentro das chaves externas) determina o número de linhas.

O número de valores inicializadores no inicializador de array aninhado para uma linha determina o número de colunas nessa linha.

Linhas podem ter tamanhos diferentes.

- Não é necessário que os comprimentos das linhas em um array bidimensional sejam os mesmos:

```
int[][] b = { { 1, 2 }, { 3, 4, 5 } };
```

Cada elemento de `b` é uma referência a um array unidimensional de variáveis `int`.

O `int` array para a linha `0` é um array unidimensional com dois elementos (`1` and `2`).

O array `int` para a linha `1` é um array unidimensional com três elementos (`3`, `4` e `5`).

- Um array multidimensional com o mesmo número de colunas em cada linha pode ser criado com uma expressão de criação de array.

```
int[][] b = new int[ 3 ][ 4 ];
```

3 linhas e 4 colunas.

- Os elementos de um array multidimensional são inicializados quando o objeto array é criado.
- Pode-se criar um array multidimensional em que cada linha tem um número diferente de colunas como mostrado a seguir:

```
int[][] b = new int[ 2 ][ ];   // cria 2 linhas
b[ 0 ] = new int[ 5 ]; // cria 5 colunas para a
linha 0
b[ 1 ] = new int[ 3 ]; // cria 3 colunas para a
linha 1
```

Cria um array bidimensional com duas linhas.

A linha 0 tem cinco colunas e linha 1 tem três colunas.

- A Figura 7.17 demonstra a inicialização de arrays bidimensionais com inicializadores de array e a utilização de loops `for` para **percorrer** os arrays.

```
1  // Figura 7.17: InitArray.java
2  // Inicializando arrays bidimensionais.
3
4  public class InitArray
5  {
6     // cria e gera saída de arrays bidimensionais
7     public static void main( String[] args )
8     {
9        int[][] array1 = { { 1, 2, 3 }, { 4, 5, 6 } };
10       int[][] array2 = { { 1, 2 }, { 3 }, { 4, 5, 6 } };
11
12       System.out.println( "Values in array1 by row are" );
13       outputArray( array1 ); // exibe array1 por linha
14
15       System.out.println( "\nValues in array2 by row are" );
16       outputArray( array2 ); // exibe array2 por linha
17    } // fim de main
18
```

array1 tem duas linhas, cada uma com três colunas

array2 tem três linhas, com duas, uma e três colunas, respectivamente

Figura 7.17 | Inicializando arrays bidimensionais. (Parte 1 de 2.)

```
19      // gera saída de linhas e colunas de um array bidimensional
20      public static void outputArray(int[][] array )
21      {
22         // faz um loop pelas linhas do array
23         for ( int row = 0; row < array.length; row++ )
24         {
25            // faz um loop pelas colunas da linha atual
26            for ( int column = 0; column < array[ row ].length; column++ )
27               System.out.printf( "%d  ", array[ row ][ column ] );
28
29            System.out.println(); // inicia nova linha de saída
30         } // fim do for externo
31      } // fim do método outputArray
32   } // fim da classe InitArray
```

O método pode receber qualquer array bidimensional de valores `int`

O loop externo define o número de linhas no array

O loop interno determina o número de colunas na *linha atual*

```
Values in array1 by row are
1  2  3
4  5  6

Values in array2 by row are
1  2
3
4  5  6
```

Figura 7.17 | Inicializando arrays bidimensionais. (Parte 2 de 2.)

7.10 Estudo de caso: classe `GradeBook` utilizando um array bidimensional

- Na maioria dos semestres, os alunos fazem vários exames.
- A Figura 7.18 contém uma versão da classe `GradeBook` que utiliza um array bidimensional `grades` para armazenar as notas de vários alunos em múltiplos exames.

  Cada linha representa as notas de um aluno ao longo de todo o curso.

  Cada coluna representa as notas de todos os alunos que fizeram uma prova específica.
- Neste exemplo, utilizamos um array dez por três contendo as notas de três exames de dez alunos.

```
1  // Figura 7.18: GradeBook.java
2  // classe GradeBook com um array bidimensional para armazenar notas
3
4  public class GradeBook
5  {
6     private String courseName; // nome do curso que este livro de nota representa
7     private int[][] grades; // array bidimensional de notas de aluno
8
9     // construtor de dois argumentos inicializa courseName e array de notas
10    public GradeBook( String name, int[][] gradesArray )
11    {
12       courseName = name; // inicializa courseName
13       grades = gradesArray; // níveis de armazenamento
14    } // fim do construtor GradeBook de dois argumentos
15
16    // método para configurar o nome do curso
17    public void setCourseName( String name )
18    {
19       courseName = name; // armazena o nome do curso
20    } // fim do método setCourseName
21
```

Figura 7.18 | Classe GradeBook utilizando um array bidimensional para armazenar notas. (Parte 1 de 9.)


```
22      // método para recuperar o nome do curso
23      public String getCourseName()
24      {
25         return courseName;
26      } // fim do método getCourseName
27
28      // exibe uma mensagem de boas-vindas para o usuário GradeBook
29      public void displayMessage()
30      {
31         // getCourseName obtém o nome do curso
32         System.out.printf( "Welcome to the grade book for\n%s!\n\n",
33            getCourseName() );
34      } // fim do método displayMessage
35
```

Figura 7.18 | Classe GradeBook utilizando um array bidimensional para armazenar notas. (Parte 2 de 9.)


```
36      // realiza várias operações nos dados
37      public void processGrades()
38      {
39         // gera saída de array de notas
40         outputGrades();
41
42         // chama métodos getMinimum e getMaximum
43         System.out.printf( "\n%s %d\n%s %d\n\n",
44            "Lowest grade in the grade book is", getMinimum(),
45            "Highest grade in the grade book is", getMaximum() );
46
47         // saída do gráfico de distribuição de todas as notas em todos os testes
48         outputBarChart();
49      } // fim do método processGrades
50
```

Figura 7.18 | Classe GradeBook utilizando um array bidimensional para armazenar notas. (Parte 3 de 9.)

```
51      // localiza nota mínima
52      public int getMinimum()
53      {
54         // assume que o primeiro elemento de array de notas é o menor
55         int lowGrade = grades[ 0 ][ 0 ];
56
57         // faz um loop pelas linhas do array de notas
58         for ( int[] studentGrades : grades )
59         {
60            // faz um loop pelas colunas da linha atual
61            for ( int grade : studentGrades )
62            {
63               // se a nota for menor que lowGrade, atribui a nota a lowGrade
64               if ( grade < lowGrade )
65                  lowGrade = grade;
66            } // fim do for interno
67         } // fim do for externo
68
69         return lowGrade; // retorna nota mais baixa
70      } // fim do método getMinimum
71
```

Loops aninhados buscam a menor nota entre todos os exames. Observe que a variável de controle do loop é um array unidimensional: o Java representa arrays bidimensionais como um array de arrays unidimensionais

Figura 7.18 | Classe GradeBook utilizando um array bidimensional para armazenar notas. (Parte 4 de 9.)


```
72      // localiza nota máxima
73      public int getMaximum()
74      {
75         // assume que o primeiro elemento de array de notas é o maior
76         int highGrade = grades[ 0 ][ 0 ];
77
78         // faz um loop pelas linhas do array de notas
79         for ( int[] studentGrades : grades )
80         {
81            // faz um loop pelas colunas da linha atual
82            for ( int grade : studentGrades )
83            {
84               // se a nota for maior que highGrade, atribui essa nota a highGrade
85               if ( grade > highGrade )
86                  highGrade = grade;
87            } // fim do for interno
88         } // fim do for externo
89
90         return highGrade; // retorna nota mais alta
91      } // fim do método getMaximum
92
```

O loop aninhado busca a nota mais alta entre todos os exames

Figura 7.18 | Classe `GradeBook` utilizando um array bidimensional para armazenar notas. (Parte 5 de 9.)


```
 93       // determina a média do conjunto de particular de notas
 94       public double getAverage( int[] setOfGrades )
 95       {
 96          int total = 0; // inicializa o total
 97
 98          // soma notas de um aluno
 99          for ( int grade : setOfGrades )
100             total += grade;
101
102          // retorna média de notas
103          return (double) total / setOfGrades.length;
104       } // fim do método getAverage
105
106       // gera saída do gráfico de barras para exibir distribuição total de notas
107       public void outputBarChart()
108       {
109          System.out.println( "Overall grade distribution:" );
110
111          // armazena frequência de notas em cada intervalo de 10 notas
112          int[] frequency = new int[ 11 ];
113
```

O método recebe um array unidimensional de valores `int` e retorna a média das notas de um aluno

Figura 7.18 | Classe `GradeBook` utilizando um array bidimensional para armazenar notas. (Parte 6 de 9.)

```
114         // para cada nota em GradeBook, incrementa a frequência certa
115         for ( int[] studentGrades : grades )
116         {
117            for ( int grade : studentGrades )
118               ++frequency[ grade / 10 ];
119         } // fim do for externo
120
121         // para cada frequência de nota, imprime barra no gráfico
122         for ( int count = 0; count < frequency.length; count++ )
123         {
124            // imprime rótulo de barra ( "00-09: ", …, "90-99: ", "100: " )
125            if ( count == 10 )
126               System.out.printf( "%5d: ", 100 );
127            else
128               System.out.printf( "%02d-%02d: ",
129                  count * 10, count * 10 + 9  );
130
131            // imprime a barra de asteriscos
132            for ( int stars = 0; stars < frequency[ count ]; stars++ )
133               System.out.print( "*" );
134
135            System.out.println(); // inicia uma nova linha de saída
136         } // fim do for externo
137      } // fim do método outputBarChart
```

Figura 7.18 | Classe GradeBook utilizando um array bidimensional para armazenar notas. (Parte 7 de 9.)

```
138
139       // gera a saída do conteúdo do array de notas
140       public void outputGrades()
141       {
142          System.out.println( "The grades are:\n" );
143          System.out.print( "            " ); // alinha títulos de coluna
144
145          // cria um título de coluna para cada um dos testes
146          for ( int test = 0; test < grades[ 0 ].length; test++ )
147             System.out.printf( "Test %d  ", test + 1 );
148
149          System.out.println( "Average" ); // título da coluna de média do aluno
150
151          // cria linhas/colunas de texto que representam notas de array
152          for ( int student = 0; student < grades.length; student++ )
153          {
154             System.out.printf( "Student %2d", student + 1 );
155
156             for ( int test : grades[ student ] ) // gera saída de notas do aluno
157                System.out.printf( "%8d", test );
158
```

Figura 7.18 | Classe GradeBook utilizando um array bidimensional para armazenar notas. (Parte 8 de 9.)


```
159             // chama método getAverage para calcular a média do aluno;
160             // passa linha de notas como o argumento para getAverage
161             double average = getAverage( grades[ student ] );
162             System.out.printf( "%9.2f\n", average );
163          } // fim do for externo
164       } // fim do método outputGrades
165    } // fim da classe GradeBook
```

Passa uma linha de notas do array para o método getAverage

Figura 7.18 | Classe GradeBook utilizando um array bidimensional para armazenar notas. (Parte 9 de 9.)

```
1  // Figura 7.19: GradeBookTest.java
2  // GradeBookTest cria o objeto GradeBook usando um array bidimensional
3  // das notas, então invoca o método proctessGrades para analisá-las.
4  public class GradeBookTest
5  {
6     // método main inicia a execução de programa
7     public static void main( String[] args )
8     {
9        // array bidimensional de notas de aluno
10       int[][] gradesArray = { { 87, 96, 70 },
11                               { 68, 87, 90 },
12                               { 94, 100, 90 },
13                               { 100, 81, 82 },
14                               { 83, 65, 85 },
15                               { 78, 87, 65 },
16                               { 85, 75, 83 },
17                               { 91, 94, 100 },
18                               { 76, 72, 84 },
19                               { 87, 93, 73 } };
20
21       GradeBook myGradeBook = new GradeBook(
22          "CS101 Introduction to Java Programming", gradesArray );
```

Figura 7.19 | GradeBookTest cria o objeto GradeBook utilizando um array bidimensional de notas e invoca o método processGrades para analisá-las. (Parte 1 de 3.)


```
23          myGradeBook.displayMessage();
24          myGradeBook.processGrades();
25       } // fim de main
26    } // fim da classe GradeBookTest
```

```
Welcome to the grade book for
CS101 Introduction to Java Programming!

The grades are:

            Test 1  Test 2  Test 3  Average
Student  1      87      96      70    84.33
Student  2      68      87      90    81.67
Student  3      94     100      90    94.67
Student  4     100      81      82    87.67
Student  5      83      65      85    77.67
Student  6      78      87      65    76.67
Student  7      85      75      83    81.00
Student  8      91      94     100    95.00
Student  9      76      72      84    77.33
Student 10      87      93      73    84.33

Lowest grade in the grade book is 65
Highest grade in the grade book is 100
```

Figura 7.19 | `GradeBookTest` cria o objeto `GradeBook` utilizando um array bidimensional de notas e invoca o método `processGrades` para analisá-las. (Parte 2 de 3.)

```
Overall grade distribution:
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: ***
70-79: ******
80-89: ***********
90-99: *******
  100: ***
```

Figura 7.19 | `GradeBookTest` cria o objeto `GradeBook` utilizando um array bidimensional de notas e invoca o método `processGrades` para analisá-las. (Parte 3 de 3.)

7.11 Listas de argumentos de comprimento variável

- **Listas de argumentos de comprimento variável**

  Podem ser utilizadas para criar métodos que recebem um número não especificado de argumentos.

  Quando o tipo de parâmetro vem seguido por **reticências (...)**, isso indica que o método recebe um número variável de argumentos desse tipo particular.

  As reticências podem ocorrer somente uma vez no fim de uma lista de parâmetros.

```
1  // Figura 7.20: VarargsTest.java
2  // Utilizando listas de argumentos de comprimento variável.
3
4  public class VarargsTest
5  {
6     // calcula a média
7     public static double average( double... numbers )
8     {
9        double total = 0.0; // inicializa o total
10
11       // calcula o total utilizando a instrução for aprimorada
12       for ( double d : numbers )
13          total += d;
14
15       return total / numbers.length;
16    } // fim do método average
17
18    public static void main( String[] args )
19    {
20       double d1 = 10.0;
21       double d2 = 20.0;
22       double d3 = 30.0;
23       double d4 = 40.0;
24
```

Um número variável de valores `double` pode ser passado para esse método

Argumentos variáveis são automaticamente colocados em um array referenciado pelo parâmetro

Figura 7.20 | Utilizando listas de argumentos de comprimento variável. (Parte 1 de 2.)


```
25          System.out.printf( "d1 = %.1f\nd2 = %.1f\nd3 = %.1f\nd4
26             = %.1f\n\n", d1, d2, d3, d4 );
27
28          System.out.printf( "Average of d1 and d2 is %.1f\n",
29             average( d1, d2 ) );
30          System.out.printf( "Average of d1, d2 and d3 is %.1f\n",
31             average( d1, d2, d3 ) );
32          System.out.printf( "Average of d1, d2, d3 and d4 is %.1f\n",
33             average( d1, d2, d3, d4 ) );
34       } // fim de main
35    } // fim da classe VarargsTest
```

```
d1 = 10.0
d2 = 20.0
d3 = 30.0
d4 = 40.0

Average of d1 and d2 is 15.0
Average of d1, d2 and d3 is 20.0
Average of d1, d2, d3 and d4 is 25.0
```

Figura 7.20 | Utilizando listas de argumentos de comprimento variável. (Parte 2 de 2.)

Java™ COMO PROGRAMAR 8ª edição

Erro comum de programação 7.6

Inserir reticências indicando uma lista de argumentos de tamanho variável no meio de uma lista de parâmetros é um erro de sintaxe. As reticências só podem ser colocadas no fim da lista de parâmetros.

7.12 Utilizando argumentos de linha de comando

- **Argumentos da linha de comando.**

  Pode passar argumentos a partir da linha de comando a um aplicativo.

  Argumentos da linha de comando que aparecem depois do nome de classe no comando `java` são recebidos por `main` no array `String args`.

  O número de argumentos de linha de comando é obtido acessando o atributo `length` do array.

  Os argumentos de linha de comando são separados por um espaço em branco, não vírgulas.

```
1  // Figura 7.21: InitArray.java
2  // Inicializando um array com argumentos de linha de comando.
3
4  public class InitArray
5  {
6     public static void main(String[] args )
7     {
8        // verifica o número de argumentos de linha de comando
9        if (args.length != 3 )
10          System.out.println(
11             "Error: Please re-enter the entire command, including\n" +
12             "an array size, initial value and increment." );
13       else
14       {
15          // obtém o tamanho do array a partir do 1o. argumento da linha de comando
16          int arrayLength = Integer.parseInt( args[ 0 ] );
17          int[] array = new int[ arrayLength ]; // cria o array
18
19          // obtém o valor inicial e o incrementa a partir dos argumentos da linha
            // de comando
20          int initialValue = Integer.parseInt( args[ 1 ] );
21          int increment = Integer.parseInt( args[ 2 ] );
22
```

Verifica se há três argumentos de linha de comando

Utiliza o primeiro argumento de linha de comando como tamanho do array a criar

Utiliza o segundo e o terceiro argumentos de linha de comando como o valor inicial e incrementa para os valores que serão gerados nas linhas 24-25

Figura 7.21 | Inicializando um array com argumentos de linha de comando. (Parte 1 de 3.)

```
23          // calculate value for each array element
24          for ( int counter = 0; counter < array.length; counter++ )
25             array[ counter ] = initialValue + increment * counter;
26
27          System.out.printf( "%s%8s\n", "Index", "Value" );
28
29          // display array index and value
30          for ( int counter = 0; counter < array.length; counter++ )
31             System.out.printf( "%5d%8d\n", counter, array[ counter ] );
32       } // end else
33    } // end main
34 } // end class InitArray
```

```
java InitArray
Error: Please re-enter the entire command, including
an array size, initial value and increment.
```

Fig. 7.21 | Initializing an array using command-line arguments. (Part 2 of 3.)

```
java InitArray 5 0 4
Index   Value
    0       0
    1       4
    2       8
    3      12
    4      16
```

Três argumentos de linha de comando passados para `InitArray`

```
java InitArray 8 1 2
Index   Value
    0       1
    1       3
    2       5
    3       7
    4       9
    5      11
    6      13
    7      15
```

Três argumentos de linha de comando passados para `InitArray`

Figura 7.21 | Inicializando um array com argumentos de linha de comando. (Parte 3 de 3.)

7.13 Classe `Arrays`

- Classe **Arrays**
  Fornece métodos `static` para manipulações de array comuns.
- Métodos incluem:
  sort para classificar um array (ordem ascendente por padrão).
  binarySearch para pesquisar um array classificado.
  Método equals para comparar arrays.
  Método fill para colocar valores em arrays.
- Os métodos são sobrecarregados para arrays de tipo primitivo e arrays de objetos.
- **Método `static` arraycopy** da classe `System`
  Copia o conteúdo de um array para outro.


```
1  // Figura 7.22: ArrayManipulations.java
2  // Métodos da classe Arrays e System.arraycopy.
3  import java.util.Arrays;
4
5  public class ArrayManipulations
6  {
7     public static void main( String[] args )
8     {
9        // classifica doubleArray em ordem crescente
10       double[] doubleArray = { 8.4, 9.3, 0.2, 7.9, 3.4 };
11       Arrays.sort( doubleArray );
12       System.out.printf( "\ndoubleArray: " );
13
14       for ( double value : doubleArray )
15          System.out.printf( "%.1f ", value );
16
17       // preenche o array de 10 elementos com 7s
18       int[] filledIntArray = new int[ 10 ];
19       Arrays.fill( filledIntArray, 7 );
20       displayArray( filledIntArray, "filledIntArray" );
21
```

Classifica o conteúdo de um array pela ordem de classificação padrão

Preenche os elementos de um array com o valor especificado como o segundo argumento

Figura 7.22 | Métodos da classe `Arrays`. (Parte 1 de 4.)

```
22          // copia array intArray em array intArrayCopy
23          int[] intArray = { 1, 2, 3, 4, 5, 6 };
24          int[] intArrayCopy = new int[ intArray.length ];
25          System.arraycopy( intArray, 0, intArrayCopy, 0, intArray.length );
26          displayArray( intArray, "intArray" );
27          displayArray( intArrayCopy, "intArrayCopy" );
28
29          // compara a igualdade de intArray e intArrayCopy
30          boolean b = Arrays.equals( intArray, intArrayCopy );
31          System.out.printf( "\n\nintArray %s intArrayCopy\n",
32             ( b ? "==" : "!=" ) );
33
34          // compara a igualdade de intArray e filledIntArray
35          b = Arrays.equals( intArray, filledIntArray );
36          System.out.printf( "intArray %s filledIntArray\n",
37             ( b ? "==" : "!=" ) );
38
39          // pesquisa em intArray o valor 5
40          int location = Arrays.binarySearch( intArray, 5 );
41
42          if ( location >= 0 )
43             System.out.printf(
44                "Found 5 at element %d in intArray\n", location );
```

Copia elementos do array no primeiro argumento para o array especificado como o terceiro argumento

Verifica se o conteúdo de dois arrays é igual

Verifica se o conteúdo de dois arrays é igual

Figura 7.22 | Métodos da classe `Arrays`. (Parte 2 de 4.)


```
45          else
46             System.out.println( "5 not found in intArray" );
47
48          // pesquisa em intArray o valor 8763
49          location = Arrays.binarySearch( intArray, 8763 );
50
51          if ( location >= 0 )
52             System.out.printf(
53                "Found 8763 at element %d in intArray\n", location );
54          else
55             System.out.println( "8763 not found in intArray" );
56       } // fim de main
57
58       // gera saída de valores em cada array
59       public static void displayArray( int[] array, String description )
60       {
61          System.out.printf( "\n%s: ", description );
62
63          for ( int value : array )
64             System.out.printf( "%d ", value );
65       } // fim do método displayArray
66    } // fim da classe ArrayManipulations
```

Procura o segundo argumento no array especificado como primeiro argumento

Figura 7.22 | Métodos da classe `Arrays`. (Parte 3 de 4.)

```
doubleArray: 0.2 3.4 7.9 8.4 9.3
filledIntArray: 7 7 7 7 7 7 7 7 7 7
intArray: 1 2 3 4 5 6
intArrayCopy: 1 2 3 4 5 6

intArray == intArrayCopy
intArray != filledIntArray
Found 5 at element 4 in intArray
8763 not found in intArray
```

Figura 7.22 | Métodos da classe `Arrays`. (Parte 4 de 4.)

Java™ COMO PROGRAMAR 8ª edição

Erro comum de programação 7.7

Passar um array não classificado para `binarySearch` *é um erro de lógica — o valor retornado é indefinido.*

7.14 Introdução a coleções e classe `ArrayList`

- A Java API fornece várias estruturas de dados predefinidas, chamadas **coleções**, utilizadas para armazenar grupos de objetos relacionados.

  Fornecem métodos eficientes que organizam, armazenam e recuperam seus dados sem que seja necessário conhecer como os dados são armazenados.

  Reduz o tempo de desenvolvimento de aplicativos.
- Arrays não mudam automaticamente de tamanho em tempo de execução para acomodar elementos adicionais.
- **ArrayList<T>** (pacote `java.util`) pode alterar dinamicamente seu tamanho para acomodar mais elementos.

  `T` é um espaço reservador para o tipo de elemento armazenado na coleção.

  Isso é semelhante a especificar o tipo ao declarar um array, exceto que apenas tipos não primitivos podem ser utilizados com essas classes de coleção.
- Classes com essa espécie de marcador de lugar que podem ser utilizadas com qualquer tipo são chamadas **classes genéricas**.

Método	Descrição
`add`	Adiciona um elemento ao fim de `ArrayList`.
`clear`	Remove todos os elementos de `ArrayList`.
`contains`	Retorna `true` se `ArrayList` contiver o elemento especificado; do contrário, retorna `false`.
`get`	Retorna o elemento no índice especificado.
`indexOf`	Retorna o índice da primeira ocorrência do elemento especificado em `ArrayList`.
`remove`	Remove a primeira ocorrência do valor especificado.
`remove`	Remove o elemento no índice especificado.
`size`	Retorna o número de elementos armazenados no `ArrayList`.
`trimToSize`	Reduz a capacidade de `ArrayList` de acordo com o número de elementos atual.

Figura 7.23 | Alguns métodos e propriedades da classe `ArrayList<T>`.

- A Figura 7.24 demonstra algumas capacidades `ArrayList` comuns.
- Uma capacidade do `ArrayList` indica quantos itens ele pode armazenar sem crescer.
- Quando o tamanho de `ArrayList` aumenta, ele precisa criar um array interno maior e copiar cada elemento para o novo array.

  Essa é uma operação demorada. Não seria eficiente se o tamanho de `ArrayList` crescesse toda vez que um elemento é adicionado.

  Um `ArrayList` cresce somente quando um elemento é adicionado e o número de elementos é igual à capacidade — isto é, não há nenhum espaço para o novo elemento.

- O método **add** adiciona elementos ao `ArrayList`.
  A versão de um argumento acrescenta seu argumento ao final do `ArrayList`.
  A versão de dois argumentos insere um novo elemento na posição especificada.
  Índices de coleção iniciam em zero.
- O método **size** retorna o número de elementos no `ArrayList`.
- O método **get** obtém o elemento em um índice especificado.
- O método **remove** exclui um elemento com um valor específico.
  Uma versão sobrecarregada do método remove o elemento no índice especificado.
- O método **contains** determina se um item está no `ArrayList`.

```
1  // Figura 7.24: ArrayListCollection.java
2  // Demonstração da coleção ArrayList genérica.
3  import java.util.ArrayList;
4
5  public class ArrayListCollection
6  {
7     public static void main( String[] args )
8     {
9        // cria um novo ArrayList de strings
10       ArrayList< String > items = new ArrayList< String >();
11
12       items.add( "red" ); // acrescenta um item à lista
13       items.add( 0, "yellow" ); // insere o valor no índice 0
14
15       // cabeçalho
16       System.out.print(
17          "Display list contents with counter-controlled loop:" );
18
19       // exiba as cores na lista
20       for ( int i = 0; i < items.size(); i++ )
21          System.out.printf( " %s", items.get( i ) );
22
```

Cria um `ArrayList` que armazena elementos `String`

Adiciona elementos ao `ArrayList`

O método `size` retorna o número de elementos na coleção; o método `get` retorna o elemento no índice especificado

Figura 7.24 | Demonstração da coleção `ArrayList<T>` genérica. (Parte 2 de 3.)

```
23          // exibe as cores utilizando foreach no método display
24          display( items,
25             "\nDisplay list contents with enhanced for statement:" );
26
27          items.add( "green" ); // adiciona "green" ao final da lista
28          items.add( "yellow" ); // adiciona "yellow" ao final da lista
29          display( items, "List with two new elements:" );
30
31          items.remove( "yellow" ); // remove o primeiro "yellow"
32          display( items, "Remove first instance of yellow:" );
33
34          items.remove( 1 ); // remove o item no índice 1
35          display( items, "Remove second list element (green):" );
36
37          // verifica se um valor está em List
38          System.out.printf( "\"red\" is %sin the list\n",
39             items.contains( "red" ) ? "": "not " );
40
41          // exibe o número de elementos em List
42          System.out.printf( "Size: %s\n", items.size() );
43       } // fim de main
44
```

O métdo remove exclui a primeira ocorrência do valor especificado

Esta versão de remove exclui o elemento no índice especificado

O método contains determina se o valor especificado está na coleção

Figura 7.24 | Demonstração da coleção `ArrayList<T>` genérica. (Parte 2 de 3.)

```
45      // exibe os elementos do ArrayList no console
46      public static void display( ArrayList< String > items, String header )
47      {
48         System.out.print( header ); // exibe o cabeçalho
49
50         // exibe cada elemento nos itens
51         for ( String item : items )
52            System.out.printf( " %s", item );
53
54         System.out.println(); // exibe o fim de linha
55      } // fim do método display
56   } // fim da classe ArrayListCollection
```

É possível utilizar a instrução for aprimorada com coleções

```
Display list contents with counter-controlled loop: yellow red
Display list contents with enhanced for statement: yellow red
List with two new elements: yellow red green yellow
Remove first instance of yellow: red green yellow
Remove second list element (green): red yellow
"red" is in the list
Size: 2
```

Figura 7.24 | Demonstração da coleção `ArrayList<T>` genérica. (Parte 3 de 3.)

7.15 (Opcional) Estudo de caso de GUI e imagens gráficas: desenhando arcos

- Desenhar arcos no Java é semelhante a desenhar ovais — um arco é simplesmente uma fatia de uma oval.
- O método `Graphics` **fillArc** desenha um arco preenchido.
- O método `fillArc` requer seis parâmetros.

  Os quatro primeiros parâmetros representam o retângulo delimitador em que o arco será desenhado.

  O quinto parâmetro é o ângulo inicial na oval e o sexto especifica a **varredura**, ou a quantidade do arco a cobrir.

  O ângulo inicial e a varredura são medidos em graus, com zero grau apontando para a direita.

  Uma varredura positiva desenha o arco no sentido anti-horário.
- O método **drawArc** requer os mesmos parâmetros que `fillArc`, mas desenha a borda do arco em vez de preenchê-lo.
- O método **setBackground** muda a cor de fundo de um componente.


```
1  // Figura 7.25: DrawRainbow.java
2  // Demonstra a utilização de cores em um array.
3  import java.awt.Color;
4  import java.awt.Graphics;
5  import javax.swing.JPanel;
6
7  public class DrawRainbow extends JPanel
8  {
9     // define as cores índigo e violeta
10    private final static Color VIOLET = new Color( 128, 0, 128 );
11    private final static Color INDIGO = new Color( 75, 0, 130 );
12
13    // a utilizar no arco-íris, iniciando da parte mais interna
14    // As duas entradas em branco resultam em um arco vazio no centro
15    private Color[] colors =
16       { Color.WHITE, Color.WHITE, VIOLET, INDIGO, Color.BLUE,
17         Color.GREEN, Color.YELLOW, Color.ORANGE, Color.RED };
18
19    // construtor
20    public DrawRainbow()
21    {
22       setBackground( Color.WHITE ); // configura o fundo como branco
23    } // fim do construtor DrawRainbow
24
```

Figura 7.25 | Desenhando um arco-íris com arcos e um array de cores. (Parte 1 de 2.)

```
25       // desenha um arco-íris utilizando arcos concêntricos
26       public void paintComponent( Graphics g )
27       {
28          super.paintComponent( g );
29
30          int radius = 20; // raio de um arco
31
32          // desenha o arco-íris perto da parte central inferior
33          int centerX = getWidth() / 2;
34          int centerY = getHeight() - 10;
35
36          // desenha arcos preenchidos com o mais externo
37          for ( int counter = colors.length; counter > 0; counter-- )
38          {
39             // configura a cor para o arco atual
40             g.setColor( colors[ counter - 1 ] );
41
42             // preenche o arco de 0 a 180 graus
43             g.fillArc( centerX - counter * radius,
44                centerY - counter * radius,
45                counter * radius * 2, counter * radius * 2, 0, 180 );
46          } // fim de for
47       } // fim do método paintComponent
48    } // fim da classe DrawRainbow
```

Figura 7.25 | Desenhando um arco-íris com arcos e um array de cores. (Parte 2 de 2.)


```
1  // Figura 7.26: DrawRainbowTest.java
2  // Aplicativo de teste para exibir um arco-íris.
3  import javax.swing.JFrame;
4
5  public class DrawRainbowTest
6  {
7     public static void main( String[] args )
8     {
9        DrawRainbow panel = new DrawRainbow();
10       JFrame application = new JFrame();
11
12       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13       application.add( panel );
14       application.setSize( 400, 250 );
15       application.setVisible( true );
16    } // fim de main
17 } // fim da classe DrawRainbowTest
```

Figura 7.26 | Criando `JFrame` para exibir um arco-íris. (Parte 1 de 2.)

Java™ COMO PROGRAMAR 8ª edição

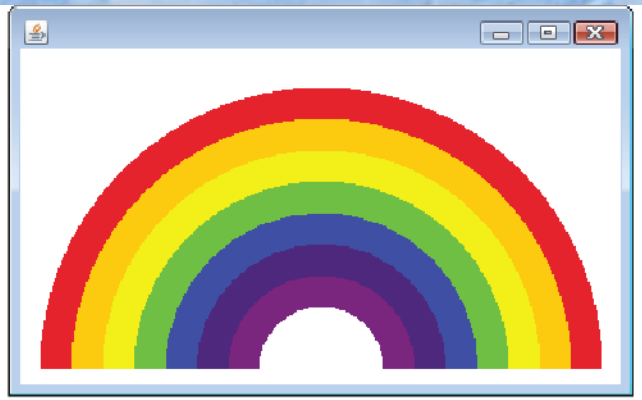


Figura 7.26 | Criando `JFrame` para exibir um arco-íris. (Parte 2 de 2.)