

# 7

## Arrays



# OBJETIVOS

Neste capítulo, você aprenderá:

- O que são arrays.
- Como utilizar arrays para armazenar dados e recuperá-los de listas e tabelas de valores.
- Como declarar um array, inicializar um array e referenciar elementos individuais de um array.
- Como utilizar a instrução **for** aprimorada para iterar por arrays.
- Como passar arrays para métodos.
- Como declarar e manipular arrays multidimensionais.
- Como escrever métodos que utilizam listas de argumentos de comprimento variável.
- Como ler argumentos da linha de comando em um programa.



- 7.1**    **Introdução**
- 7.2**    **Arrays**
- 7.3**    **Declarando e criando arrays**
- 7.4**    **Exemplos que utilizam arrays**
- 7.5**    **Estudo de caso: Simulação de embaralhamento e distribuição de cartas**
- 7.6**    **A estrutura for aprimorada**
- 7.7**    **Passando arrays para métodos**
- 7.8**    **Estudo de caso: Classe GradeBook utilizando um array para armazenar notas**
- 7.9**    **Arrays multidimensionais**
- 7.10**   **Estudo de caso: Classe GradeBook utilizando um array bidimensional**
- 7.11**   **Listas de argumentos de comprimento variável**
- 7.12**   **Utilizando argumentos de linha de comando**
- 7.13**   **(Opcional) Estudo de caso de GUIs e imagens gráficas: Desenhando arcos**
- 7.14**   **(Opcional) Estudo de caso de engenharia de software: Colaboração entre objetos**
- 7.15**   **Conclusão**

# 7.1 Introdução

- **Arrays:**

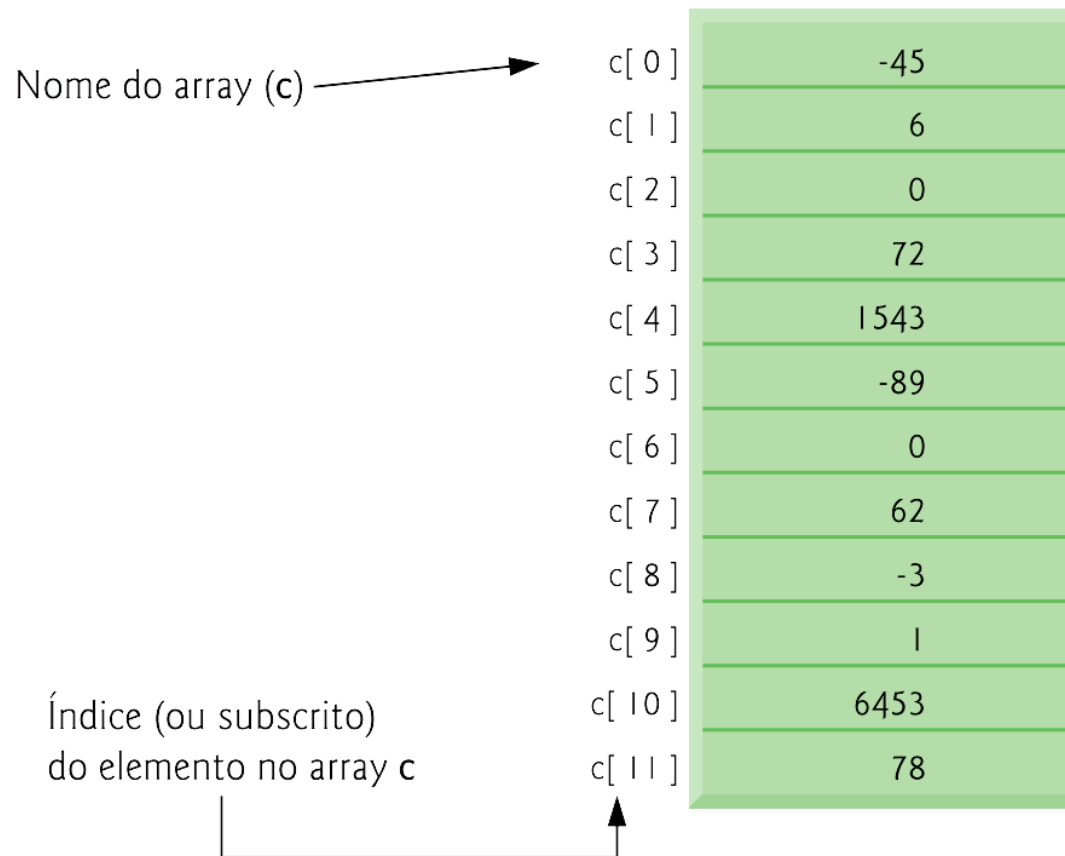
- **Estruturas de dados que consistem em itens de dados do mesmo tipo relacionados.**
- **Permanecem com o mesmo tamanho depois de criados.**
  - **Entradas de largura fixa.**



## 7.2 Arrays

- **Um Array:**
  - É um grupo de variáveis (*elementos* ou *componentes*) que contém valores que são todos do mesmo tipo.
  - É um objeto, daí ser considerado *tipo por referência*.





**Figura 7.1 | Um array de 12 elementos.**

## 7.2 Arrays (*Continuação*)

- **Índice:**

- Também chamado subscrito.
- Número de posição entre colchetes.
- Deve ser inteiro positivo ou expressão de inteiro.
- Primeiro elemento tem o índice zero:

```
a = 5;  
b = 6;  
c[ a + b ] += 2;
```

- Adiciona 2 a c[ 11 ].



## Erro comum de programação 7.1

---

**Utilizar um valor de tipo `long` como um índice de array resulta em um erro de compilação.**

**Um índice deve ser um valor `int` ou um valor de um tipo que possa ser promovido para `int` — a saber, `byte`, `short` ou `char`, mas não `long`.**





## 7.2 Arrays (*Continuação*)

- **Examine o array C:**
  - **C** é o *nome* do array.
  - **c.length** acessa o comprimento do array **C**.
  - **C** tem 12 elementos (**c[0]**, **c[1]**, ... **c[11]**)
    - O *valor* de **c[0]** é – 45.



## 7.3 Declarando e criando arrays

- **Declarando e criando arrays:**

- Arrays são objetos que ocupam memória.
- São criados dinamicamente com a palavra-chave **new**:

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

Equivalente a:

```
int c[]; // declara a variável  
array
```

```
c = new int[ 12 ]; // cria o array
```

- Também podemos criar arrays de objetos:

```
String b[] = new String[ 100 ];
```



## Erro comum de programação 7.2

---

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



# 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.**



## 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 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 array, e `b` e `c` variáveis `int` individuais, então essa declaração é incorreta — a declaração `int a[], b, c;` alcançaria o resultado desejado.**

---



## 7.4 Exemplos que utilizam arrays

- **Declarando arrays.**
- **Criando arrays.**
- **Inicializando arrays.**
- **Manipulando elementos do array.**



## 7.4 Exemplos que utilizam arrays (*Continuação*)

- **Criando e inicializando um array:**
  - **Declare o array.**
  - **Crie o array.**
  - **Inicialize os elementos do array.**



# Resumo

## InitArray.java

Linha 8

Declara **array** como um array de **ints**

Linha 10

Cria 10 **ints** para um array; cada **int** é inicializado como 0 por padrão

Linha 15

**array.length** retorna o comprimento do array

Linha 16

**array[counter]** retorna o **int** associado com o índice em um array

Saída do programa

```

1 // Fig. 7.2: InitArray.java
2 // Criando um array.
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 espaço
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

```

Declara **array** como um array de **ints**

Cria 10 **ints** para um array; cada **int** é inicializada como 0 por padrão

**array.length** retorna o comprimento do array

Cada **int** é inicializado como 0 por padrão

**array[counter]** retorna o **int** associado com o índice em um array

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



## 7.4 Exemplos que utilizam arrays (*Continuação*)

- **Utilizando um inicializador de array:**
  - **Utiliza a *lista inicializadora*:**
    - Itens entre chaves (}).
    - Itens em uma lista separada por vírgulas.  
`int n[] = 1 i, 20, 30, 40, 50`
    - Cria um array de cinco elementos.
    - Indexa valores de 0, 1, 2, 3, 4.
  - **Não precisa da palavra-chave `new`.**

## Resumo

### InitArray.java

```

1 // Fig. 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 inicializadores especifica o valor de
9         int array[] = { 32, 27, 64, 18, 95, 14, 90, 70, 60 };
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

```

Declara array como um array de ints

Compilador utiliza a lista inicializadora para alocar um array

Linha 9  
Declara array como um array de ints

Linha 9  
Compilador utiliza a lista inicializadora para alocar um array

Saída do programa

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



## 7.4 Exemplos que utilizam arrays (*Continuação*)

- **Calculando um valor para armazenar em cada elemento de array:**
  - **Inicializa elementos do array de 10 elementos como inteiros pares.**



# Resumo

## InitArray.java

```

1 // Fig. 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 a constante
9         int array[] = new int[ ARRAY_LENGTH ]; // cria o array
10
11         // calcula o valor para 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 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

```

Declara a variável constante `ARRAY_LENGTH` utilizando o modificador `final`

Declara e cria um array que contém 10 ints

Utiliza o índice array para atribuir um valor de array

a 8  
para a  
variável  
constante

Linha 9  
Declara e cria um  
array que contém  
10 ints

Linha 13  
Utiliza o índice  
do array para  
atribuir o array

Saída do programa

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



## Boa prática de programação 7.2

---

**Variáveis constantes também são chamadas constantes identificadas ou variáveis de leitura (*read-only*). Frequentemente, essas variáveis tornam os programas mais legíveis do 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 no contexto em que ele é utilizado.**

---



## Erro comum de programação 7.4

---

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

# Erro comum de programação 7.5

---

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



## 7.4 Exemplos que utilizam arrays (*Continuação*)

- **Somando elementos de um array:**
  - **Elementos do array podem representar uma série de valores.**
    - **Podemos somar esses valores.**



```

1 // Fig. 7.5: SumArray.java
2 // Calculando a soma dos elementos de um
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

```

Declara array com uma lista inicializadora

Soma todos os valores do array

## Resumo

### SumArray.java

Linha 8

Declara array com uma lista inicializadora

Linhas 12-13

Soma todos os valores de array

Total of array elements: 849

Saída do programa



## 7.4 Exemplos que utilizam arrays (*Continuação*)

- **Utilizando gráficos de barras para exibir dados de array graficamente:**
  - Apresenta os dados graficamente.
    - Por exemplo: gráfico de barras.
  - Examina a distribuição das notas.



```

1 // Fig. 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            // output bar label ( "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
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
27        } // fim do for externo
28    } // fim de main
29 } // fim da classe BarChart

```

Declara array com uma lista inicializadora

Utiliza o flag 0 para exibir nota de um algarismo com um 0 inicial

Para cada elemento do array, imprime o número associado de asteriscos

## Resumo

### BarChart.java

(1 de 2)

Linha 8

Declara array com uma lista inicializadora

Linha 19

Utiliza o flag 0 para exibir nota de um

Utiliza o flag 0 para exibir nota de um algarismo com um 0 inicial

Para cada elemento do array, imprime o número associado de asteriscos



## Resumo

### BarChart.java

(2 de 2)

Saída do programa

Grade distribution:

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



## 7.4 Exemplos que utilizam arrays (*Continuação*)

- **Utilizando os elementos de um array como contadores:**
  - Utiliza uma série de variáveis contadoras para resumir os dados.

## Resumo

```

1 // Fig. 7.7: RollDie.java
2 // Rola um dado de seis lados 6000 vezes.
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
13        for ( int roll = 1; roll <= 6000; roll++ )
14            ++frequency[ 1 + randomNumbers.nextInt( 6 ) ];
15
16        System.out.printf( "%s%10s\n",
17
18        // gera saída do valor de cada face
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

```

Declara frequency  
como um array de 7 ints

Linha 10  
Declara frequency  
como um array de 7

Gera 6000 inteiros  
aleatórios no intervalo 1-6

Linhas 13-14  
Geram 6000 inteiros  
aleatórios no  
intervalo 1-6

Incrementa os valores de  
frequency no índice associado com  
um número aleatório

Linha 14  
Incrementa valores de  
frequency no índice  
associado com o  
número aleatório

Saída do programa

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



## 7.4 Exemplos que utilizam arrays (*Continuação*)

- **Utilizando arrays para analisar resultados de pesquisas:**
  - **40 alunos avaliam a qualidade da comida:**
    - Escala de avaliação de 1-10 — 1 significa horrível, 10 significa excelente.
  - **Coloque 40 respostas no array de inteiros.**
  - **Resuma os resultados da enquete.**



## Resumo

```

1 // Fig. 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 pesquisa
9         int responses[] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
10             10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6, 5,
11             4, 8, 6, 8, 10 };
12         int frequency[] = new int[ 11 ]; // array de contagem
13
14         // para cada resposta, seleciona elemento de respostas e usa esse valor
15         // como índice de frequência para determinar 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

```

Declare responses as array to store 40 responses

Declara frequency como um array de 11 ints e ignora o primeiro elemento

Para cada resposta, incrementa os valores de frequency no índice associados com essa resposta

armazenar 40 respostas  
Linha 12  
Declara frequency como array de 11  
ignora o elemento  
5-17  
a resposta,  
com os  
de  
frequency no índice  
associado com essa  
resposta





## Resumo

# StudentPoll. java

(2 de 2)

Saída do programa

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



## Dica de prevenção de erro 7.1

---

**Uma exceção indica que ocorreu um erro em um programa. Frequentemente, um programador pode escrever código para recuperar-se de uma exceção e continuar a execução do programa, em vez de terminar o programa de maneira anormal. Quando um programa tenta acessar um elemento fora dos limites do array, ocorre uma `ArrayIndexOutOfBoundsException`. O tratamento de exceções é discutido no Capítulo 13.**

---



## Dica de prevenção de erro 7.2

---

**Ao escrever código para fazer um loop por um array, assegure-se de que o índice do array sempre seja maior ou igual a 0 e menor que o comprimento do array.**

**A condição de continuação do loop deve evitar acesso a elementos fora desse intervalo.**



## 7.5 Estudo de caso: Simulação de embaralhamento e distribuição de cartas

- **O programa simula o ato de embaralhar e distribuir cartas:**
  - Utiliza geração de números aleatórios.
  - Utiliza um array de elementos de tipo por referência para representar as cartas.
  - Três classes:
    - **Card:**
      - Representa uma carta do baralho.
    - **DeckOfCards:**
      - Representa um baralho com 52 cartas.
    - **DeckOfCardsTest:**
      - Demonstra o embaralhamento e a distribuição de cartas.



## Resumo

### Card.java

#### Linhas 17-20

```
1 // Fig. 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; // naipes da carta ("Hearts", "Diamonds", ...)
8
9     // construtor de dois argumentos inicializa face e naipes da carta
10    public Card( String cardFace, String cardSuit )
11    {
12        face = cardFace; // inicializa face da carta
13        suit = cardSuit; // inicializa naipes 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
```

Retorna a representação de  
string de uma carta



## Resumo

```

1 // Fig. 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
9     private final int NUMBER_OF_CARDS = 52; // número constante de cards
10    private Random randomNumbers; // gerador de número aleatório
11
12    // construtor preenche baralho de cards
13    public DeckOfCards()
14    {
15        String faces[] = { "Ace", "Deuce", "Three", "Four", "Five", "Six",
16                           "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King" };
17        String suits[] = { "Hearts", "Diamonds", "Clubs", "Spades" };
18
19        deck = new Card[ NUMBER_OF_CARDS ]; // cria array de objetos Card
20        currentCard = 0; // configura currentCard então o primeiro Card distribuído é deck[ 0 ]
21        randomNumbers = new Random(); // cria gerador de números aleatórios
22
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

```

Declara **deck** como um array para armazenar objetos **Card**

A constante **NUMBER\_OF\_CARDS** indica o número de **Cards** no baralho

(1 de 2)

Declara e inicializa **faces** com **Strings** que representam a face da carta

Declara e inicializa **suits** com **Strings** que representam o naipe da carta

Linhas 15-16

Linhas 17

Linhas 24-26

Preenche o array **deck** com **Cards**



## Resumo

# DeckOfCards. java

(2 de 2)

```

28 // embaralha as cartas com um algoritmo de uma passagem
29 public void shuffle()
30 {
31     // depois de embaralhar, a distribuição deve iniciar em deck[ 0 ] novamente
32     currentCard = 0; // reinicializa currentCard
33
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 de for
46 } // fim do método shuffle
47
48 // distribui um Card
49 public Card dealCard()
50 {
51     // determina se ainda há Cards a ser distribuídos
52     if ( currentCard < deck.length )
53         return deck[ currentCard++ ]; // retorna Card atual no array
54     else
55         return null; // retorna nulo p/ indicar que todos os Cards foram distribuídos
56 } // fim do método dealCard
57 } // fim da classe DeckOfCards

```

Troca a Card atual por  
uma Card aleatoriamente  
selecionada

Linha 52

Determine se deck  
está vazio



## Resumo

# DeckOfCards Test .java

(1 de 2)

```
1 // Fig. 7.11: DeckOfCardsTest.java
2 // Aplicativo de embaralhar e distribuir 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 elas são distribuídas
13        for ( int i = 0; i < 13; i++ )
14        {
15            // distribui e imprime 4 Cards
16            System.out.printf( "%-20s%-20s%-20s%-20s\n",
17                               myDeckOfCards.dealCard(), myDeckOfCards.dealCard(),
18                               myDeckOfCards.dealCard(), myDeckOfCards.dealCard() );
19        } // fim de for
20    } // fim de main
21 } // fim da classe DeckOfCardsTest
```





Resumo**DeckOfCardsTest**  
**.java**

(2 de 2)



## 7.6 A estrutura for aprimorada

- Instrução **for** aprimorada:
  - Novo recurso do J2SE 5.0.
  - Permite iterar por elementos de um array ou por uma coleção sem utilizar um contador.
  - Sintaxe:

**for** ( *parâmetro* : *nomeDoArray* )  
    *instrução*



## Resumo

### EnhancedForTest.java

```
1 // Fig. 7.12: EnhancedForTest.java
2 // Utilizando 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 de main
17 } // fim da classe EnhancedForTest
```

Para cada iteração, atribui o próximo elemento do array à variável `int number`, e então o adiciona a `total`

Total of array elements: 849



## 7.6 A estrutura de repetição for (*Continuação*)

- As linhas 12-13 são equivalentes a:

```
for ( int counter = 0; counter <  
    array.length; counter++ )  
    total += array[ counter ];
```

- **Uso:**
  - Pode acessar elementos do array.
  - Não pode modificar elementos do array.
  - Não pode acessar o contador que indica o índice.



## 7.7 Passando arrays para métodos

- Para passar um argumento de array para um método:
  - Especifique o nome de array sem colchetes:
    - Array `hourlyTemperatures` é declarado como  
`int hourlyTemperatures = new int[24];`
    - A chamada de método:  
`modifyArray( hourlyTemperatures );`
    - Passa o array `hourlyTemperatures` para o método `modifyArray`.



## Resumo

## PassArray.java

```

1 // Fig. 7.13: PassArray.java
2 // Passando arrays e elementos do arrays individuais aos métodos.
3
4 public class PassArray
5 {
6     // main cria array e chama modifyArray
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 referência de 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 );
25
26         System.out.printf(
27             "\n\nEffects of passing array element value:\n" +
28             "array[3] before modifyElement: %d\n", array[ 3 ] );

```

Declara array de 5  
ints com uma lista  
inicializadora

Passa o array inteiro para o  
método modifyArray

de 2)

Linha 9

Linha 19



## Resumo

y.j

```

29  modifyElement( array[ 3 ] ); // tenta modificar o array[ 3 ]
30  System.out.printf(
31      "array[3] after modifyElement
32  } // fim de main
33
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 o 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 o elemento de array array[3] para o método modifyElement

O método modifyArray manipula o array diretamente

O método modifyElement manipula a cópia de um primitivo

(2 de 2)

Linha 30

Linhas 36-40

Linhas 43-48

Saída do programa

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



## 7.7 Passando arrays para métodos (*Continuação*)

- **Notas sobre a passagem de argumentos para métodos:**
  - **Duas maneiras de passar argumentos para métodos:**
    - **Passagem *por valor*:**
      - Cópia do valor do argumento é passada para o método chamado.
      - No Java, todo primitivo é passado por valor.
    - **Passagem *por referência*:**
      - O chamador fornece ao método chamado acesso direto aos dados do chamador.
      - Método chamado pode manipular esses dados.
      - Desempenho aprimorado em relação à passagem por valor.
      - No Java, todo objeto é passado por referência.
        - No Java, arrays são objetos.
        - Portanto, arrays são passados para os métodos por referência.





## 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

- **Desenvolve ainda mais a classe GradeBook.**
- **Classe GradeBook:**
  - Representa um boletim de notas que armazena e analisa notas.
  - Não mantém valores de notas individuais.
  - A repetição dos cálculos exige a reinserção das mesmas notas.
    - Isso pode ser resolvido armazenando notas em um array.



## Resumo

### GradeBook.java

```

1 // Fig. 7.14: GradeBook.java
2 // Grade book utilizando 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 array de notas
10    public GradeBook( String name, int gradesArray[] )
11    {
12        courseName = name; // inicializa courseName
13        grades = gradesArray; // armazena notas
14    } // construtor de dois argumentos inicializa courseName e array de notas
15
16    // método para configura 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
22    // método para recuperar o nome do curso
23    public String getCourseName()
24    {
25        return courseName;
26    } // fim do método getCourseName
27

```

Declara o array **grades**  
para armazenar notas  
individuais

de 5)

inha 7

Linha 13

Atribui a referência do array  
à variável de instância  
**grades**



## Resumo

# GradeBook.java

(2 de 5)

```
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 média
43     System.out.printf( "\nClass average is %.2f\n", getAverage() );
44
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 nota
50     outputBarChart();
51 } // fim do método processGrades
52
53 // encontra nota mínima
54 public int getMinimum()
55 {
56     int lowGrade = grades[ 0 ]; // assume que grades[ 0 ] é a menor nota
57
```



## Resumo

# gradeBook.java

(3 de 5)

Linhas 59-64

Linhas 75-80

Faz um loop pelas grades a fim de encontrar a nota mais baixa

Faz um loop pelas grades para encontrar a nota mais alta

```
58 // faz um loop pelo array de notas
59 for ( int grade : grades )
60 {
61     // se nota for mais baixa que lowGrade, atribui-a a lowGrade
62     if ( grade < lowGrade )
63         lowGrade = grade; // nova nota mais baixa
64 } // for final

65
66 return lowGrade; // retorna a menor nota
67 } // fim do método getMinimum

68
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     } // for final

81
82     return highGrade; // retorna a nota mais alta
83 } // fim do método getMaximum
84
```



## Resumo

## GradeBook.java

```
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 a 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 ];
109 }
```

Faz um loop pelas **grades**  
para somar as notas de um  
aluno

4 de 5)

linhas 91-92

Linhas 107-108

Faz um loop pelas **grades**  
para calcular a frequência



## Resumo

# GradeBook.java

(5 de 5)

Linhas 134-136

```

110 // para cada frequência de nota, imprime barrano gráfico
111 for ( int count = 0; count < frequency.length; count++ )
112 {
113     // gera saída do 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
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

```

Faz um loop pelas grades  
para exibir cada nota



# Observação de engenharia de software 7.1

---

**Um arreio de teste (ou 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 14) ou de uma rede (como você verá no Capítulo 24). Depois de passar esses dados para o construtor da classe para instanciar o objeto, o arreio de teste deve chamar o objeto para testar seus métodos e manipular seus dados. Reunir os dados em um arreio de teste como esse permite à classe manipular dados de várias fontes.**





## Resumo

deBook

Test

.java

(1 de 2)

Linha 10

Linha 13

```
1 // Fig. 7.15: GradeBookTest.java
2 // Cria objeto GradeBook utilizando um array de notas.
3
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
```

Declara e inicializa  
gradesArray com 10  
elementos

Passa gradesArray  
para construtor de  
GradeBook



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
```

Grade distribution:

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

## Resumo

## GradeBook Test

## .java

(2 de 2)

Saída do programa



## 7.9 Arrays multidimensionais

- **Arrays multidimensionais:**
  - **Tabelas com linhas e colunas:**
    - **Array bidimensional.**
    - **Array  $m$  por  $n$ .**



	Coluna 0	Coluna 1	Coluna 2	Coluna 3
Linha 0	a[ 0 ][ 0 ]	a[ 0 ][ 1 ]	a[ 0 ][ 2 ]	a[ 0 ][ 3 ]
Linha 1	a[ 1 ][ 0 ]	a[ 1 ][ 1 ]	a[ 1 ][ 2 ]	a[ 1 ][ 3 ]
Linha 2	a[ 2 ][ 0 ]	a[ 2 ][ 1 ]	a[ 2 ][ 2 ]	a[ 2 ][ 3 ]

Índice de coluna  
Índice de linha  
Nome do array

**Figura 7.16 | Array bidimensional com três linhas e quatro colunas.**

## 7.9 Arrays multidimensionais (*Continuação*)

- **Arrays de arrays unidimensionais:**

- **Declarando um array bidimensional**

**b[2][2]**

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

- 1 e 2 inicializam b[0][0] e b[0][1].

- 3 e 4 inicializam b[1][0] e b[1][1].

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

- A linha 0 contém elementos 1 e 2.

- A linha 1 contém elementos 3, 4 e 5.

## 7.9 Arrays multidimensionais (*Continuação*)

- Arrays bidimensionais com linhas de diferentes comprimentos:
  - Os comprimentos das linhas no array não precisam ser idênticos.
  - Por exemplo, `int b[] [] = { { 1, 2 }, { 3, 4, 5 } };`

## 7.9 Arrays multidimensionais (Cont.)

- Criando arrays bidimensionais com expressões de criação de arrays:

- Pode ser criado dinamicamente.

- Array 3 por 4:

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

- Cada linha pode ter um número diferente de colunas:

```
int b[][];  
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
```



## Resumo

## InitArray.j

ava

l de 2)

linha 9

Linha 10

```

1 // Fig. 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

```

Utiliza inicializadores de array aninhados para inicializar array1

Utiliza inicializadores de array aninhados de comprimentos diferentes para inicializar array2





```

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

```

array[row].length retorna o número de colunas associado com o subscrito row

InitArray.java

(2 de 2)

Linha 26

Linha 27

Utiliza a notação de colchete duplo para acessar valores do array bidimensional

a do programa

Values in array1 by row are

```

1 2 3
4 5 6

```

Values in array2 by row are

```

1 2
3
4 5 6

```



## 7.9 Arrays multidimensionais (*Continuação*)

- **Manipulações de arrays multidimensionais comuns realizadas com as instruções `for`:**
  - **Muitas manipulações de array comuns utilizam as instruções `for`.**

**Por exemplo:**

```
for ( int column = 0; column < a[ 2 ].length;  
    column++ )  
    a[ 2 ][ column ] = 0;
```

## 7.10 Estudo de caso: Classe GradeBook utilizando um array bidimensional

- **Classe GradeBook:**
  - **Array unidimensional:**
    - Armazena as notas de um aluno em um exame individual.
  - **Array bidimensional:**
    - Armazena as notas de um aluno individual e da turma como um todo.



## Resumo

### GradeBook.java

(1 de 7)

Linha 7

Linha 10

```

1 // Fig. 7.18: GradeBook.java
2 // Livro de nota utilizando um array bidimensional para armazenar notas.
3
4 public class GradeBook {
5     // Declara o array bidimensional grades
6     private String courseName; // nome de 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; // armazena notas
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
22    // método para recuperar o nome do curso
23    public String getCourseName()
24    {
25        return courseName;
26    } // fim do método getCourseName
27

```

Construtor **GradeBook** aceita uma **String** e um array bidimensional



## Resumo

# GradeBook.java

(2 de 7)

```
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 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     // gera saída de gráfico de distribuição de notas de todas as notas em todos os testes
48     outputBarChart();
49 } // fim do método processGrades
50
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
```

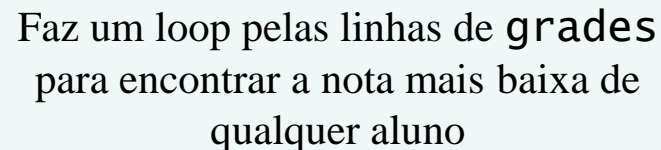


## Resumo

# GradeBook.java

(3 de 7)

Linhas 58-67



Faz um loop pelas linhas de grades para encontrar a nota mais baixa de qualquer aluno

```

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
64         if ( grade < lowGrade )
65             lowGrade = grade;
66     } // fim de for interno
67 } // fim de for externo

68
69 return lowGrade; // retorna nota mais baixa
70 } // fim do método getMinimum

71
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

```



## Resumo

# GradeBook.java

(4 de 7)

Linhas 79-88

Linhas 94-104

```

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
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
93 // determina a média do conjunto de particular de notas
94 public double getAverage( int setOfGrades[] )
95 {
96     int total = 0; // inicializa 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

```

Faz um loop pelas linhas de grades para encontrar a nota mais alta de qualquer aluno

Calcula a média semestral de um aluno em particular



## Resumo

# GradeBook.java

(5 de 7)

Linhas 115-119

```

106 // gera a 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
114     // para cada nota em GradeBook, incrementa a frequência apropriada
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         // gera saída do 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( "*" );

```

Calcula a distribuição de todas as notas dos alunos





## Resumo

# GradeBook.java

(6 de 7)

```
134         System.out.println(); // inicia uma nova linha de saída
135     } // fim do for externo
137 } // fim do método outputBarChart
138
139 // gera 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     }
```



```
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
```

## Resumo

## GradeBook.java

(7 de 7)



## Resumo

# GradeBookTest.java

(1 de 2)

Linhas 10-19

```

1 // Fig. 7.19: GradeBookTest.java
2 // Cria objeto GradeBook utilizando um array bidimensional de notas.
3
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" );
23        myGradeBook.displayMessage();
24        myGradeBook.processGrades();
25    } // fim de main
26 } // fim da classe GradeBookTest
  
```

Declara gradesArray como um array 3 por 10

Cada linha representa um aluno; cada coluna representa uma nota de exame



Resumo

# GradeBook Test

## .java

(2 de 2)

Saída do programa

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

Overall grade distribution:

00-09:

10-19:

20-29:

30-39:

40-49:

50-59:

60-69: \*\*\*

70-79: \*\*\*\*\*

80-89: \*\*\*\*\*

90-99: \*\*\*\*\*

100: \*\*\*



## 7.11 Listas de argumentos de comprimento variável

- **Listas de argumentos de comprimento variável:**
  - **Novo recurso no J2SE 5.0.**
  - **Número não-especificado de argumentos.**
  - **Utilize reticências (...) na lista de parâmetros do método.**
    - **Pode ocorrer somente uma vez na lista de parâmetros.**
    - **Deve ser colocado no final da lista de parâmetros**
  - **O array cujos elementos são todos do mesmo tipo.**



## Resumo

### VarargsTest .java

```
1 // Fig. 7.20: VarargsTest.java
2 // Utilizando listas de argumentos de comprimento variável.
3
4 public class VarargsTest
5 {
6     // calcula média
7     public static double average( double... numbers )
8     {
9         double total = 0.0; // inicializa total
10
11         // calcula total usando a instrução
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     }
```

O método **average** recebe uma sequência  
variável de comprimento de **doubles**

Calcula o total dos **doubles** no array

Acessa **numbers.length** para obter  
o tamanho do array **numbers**

e 2)

Linha 7

Linhas 12-13

Linha 15



## Resumo

### VarargsTest

.java

de 2)

Linha 29

Linha 31

Linha 33

Saída do  
programa

```

25  System.out.printf( "d1 = %.1f\nd2 = %.1f\nd3 = %.1f\nd4 = %.1f\n\n",
26      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

```

Invoca o método **average**  
com dois argumentos

Invoca o método **average**  
com três argumentos

Invoca método **average**  
com quatro argumentos

```

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

```



## Erro comum de programação 7.6

---

**Colocar reticências no meio de uma lista de parâmetros de método 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:**
  - É possível passar argumentos a partir da linha de comando (*argumentos de linha de comando*):
    - `String args[]`
  - Aparece depois do nome da classe no comando `java`:
    - `java MyClass a b`
  - Número de argumentos passados a partir da linha de comando é obtido acessando:
    - `args.length`
  - Primeiro argumento de linha de comando
    - `args[ 0 ]`



## Resumo

### InitArray. java

(1 de 2)

Linha 6

Linha 9

Linha 16

Linhas 20-21

Linhas 24-25

```

1 // Fig. 7.21: InitArray.java
2 // Utilizando argumentos de linha de comando para inicializar um array.
3
4 public class InitArray
5 {
6     public static void main( String args[] )
7     {
8         // verifica número de argumentos de linha de comando
9         if ( args.length != 3 )
10             System.out.println(
11                 "Error: Please re-enter
12                 "an array
13             else
14             {
15                 // obtém o tamanho do array a partir da linha de comando
16                 int arrayLength = Integer.parseInt( args[ 0 ] );
17                 int array[] = new int[ arrayLength ]; // cria array
18
19                 // obtém o valor inicial e o incremento do array
20                 int initialValue = Integer.parseInt( args[ 1 ] );
21                 int increment = Integer.parseInt( args[ 2 ] );
22
23                 // calcula o valor de cada elemento do array
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

```

Array args armazena  
argumentos de linha de  
comando

Verifica o número de argumentos  
passados a partir da linha de  
comando

de linha de comando

Obtém o primeiro argumento na  
linha de comando

Obtém o segundos e o terceiro  
argumentos na linha de comando

Calcula o valor de cada elemento do array  
com base nos argumentos de linha de  
comando

## Resumo

### InitArray.java

(2 de 2)

Saída do programa

```

29         // exibe o valor e o índice de array
30         for ( int counter = 0; counter < array.length; counter++ )
31             system.out.printf( "%5d%8d\n", counter, array[ counter ] );
32     } // fim de else
33 } // fim de main
34 } // fim da classe InitArray

```

```

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

```

```

java InitArray
Index 0
      1
      2
      3
      4

```

Argumentos da linha de comando ausentes

Três argumentos de linha de comando são 5, 0 e 4

```

java InitArray 10 1 2
Index Value
0      1
1      3
2      5
3      7
4      9
5     11
6     13
7     15
8     17
9     19

```

Três argumentos de linha de comando são 10, 1 e 2



## 7.13 (Opcional) Estudo de caso de GUIs e imagens gráficas: Desenhando arcos

- **Desenhe um arco-íris:**
  - Utilize arrays.
  - Utilize a instrução de repetição.
  - Utilize o método `fillArc` de `Graphics`.



## Resumo

### DrawRainbow .java

(1 de 2)

Linha 22

```
1 // Fig. 7.22: 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    final Color VIOLET = new Color( 128, 0, 128 );
11    final 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
25    // desenha um arco-íris u
26    public void paintComponent
27    {
28        super.paintComponent( g );
29
30        int radius = 20; // raio de um arco
```

Configura o segundo plano do  
componente como branco



## Resumo

### DrawRainbow .java

(2 de 2)

Linhas 43-45

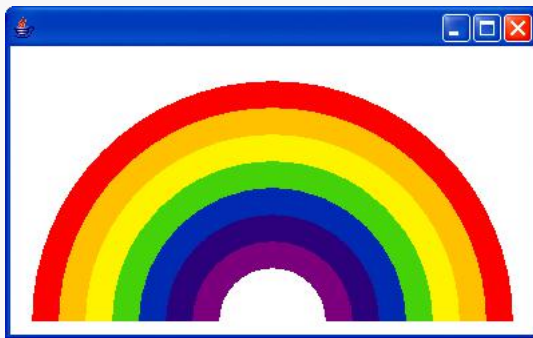
```
31 // desenha o arco-íris perto da parte central inferior
32 // desenha o arco-íris perto da parte central inferior
33 int centerX = getWidth() / 2;
34 int centerY = getHeight() - 10;
35
36 // desenha arcos preenchidos com o mais externo
37 for ( int counter = colors.length; counter > 0; counter-- )
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
```

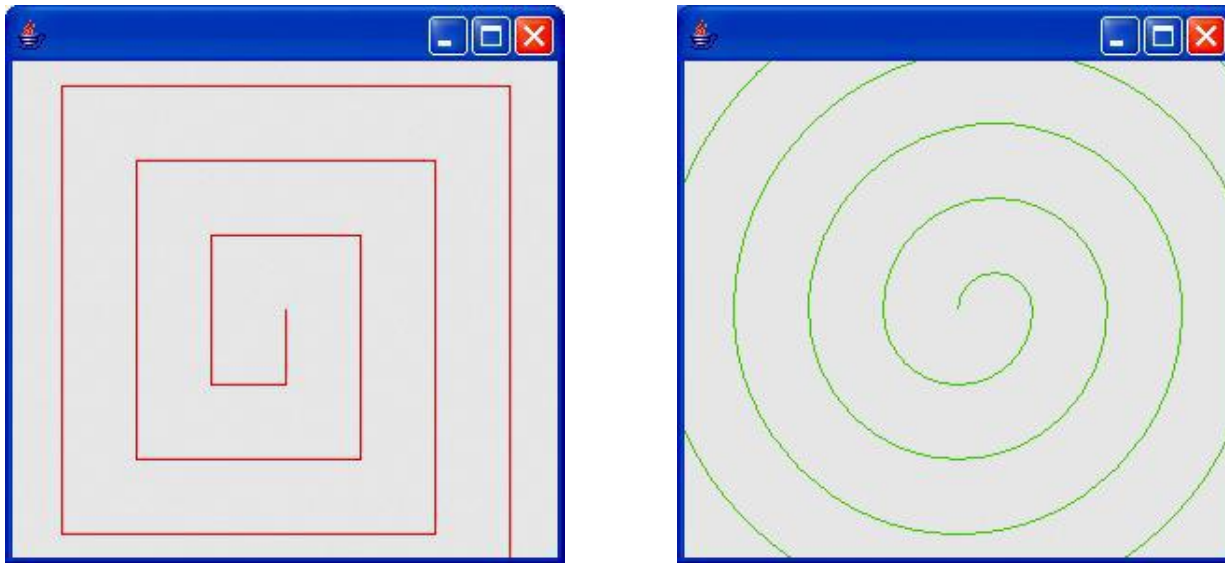
Desenha um semicírculo preenchido



## DrawRainbowTest.java

```
1 // Fig. 7.23: 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 7.24** | Desenhando uma espiral com `drawLine` (esquerda) e `drawArc` (direita).



## 7.14 (Opcional) Estudo de caso de engenharia de software: Colaboração entre objetos

- **Colaborações (interações):**
  - Quando objetos se comunicam para realizar uma tarefa, diz-se que eles são *colaboradores*.
    - Objetos se comunicam invocando operações (métodos) uns dos outros.
  - Consistem em um objeto de uma classe enviar uma *mensagem* para um objeto de outra classe.



## 7.14 (Opcional) Estudo de caso de engenharia de software (*Continuação*)

- **Identificando as colaborações em um sistema:**
  - **Leia o documento de requisitos para localizar:**
    - o que ATM deve fazer para autenticar um uso; e
    - o que ATM deve fazer para realizar transações.
  - **Para cada ação, decida:**
    - **Quais objetos devem interagir:**
      - Objeto emissor.
      - Objeto receptor.



Um objeto da classe...	envia a mensagem...	para um objeto da classe...
ATM	displayMessage getInput authenticateUser execute execute Execute	Screen Keypad BankDatabase BalanceInquiry withdrawal Deposit
BalanceInquiry	getAvailableBalance getTotalBalance displayMessage	BankDatabase BankDatabase Screen
withdrawal	displayMessage getInput getAvailableBalance isSufficientCashAvailable debit dispenseCash	Screen Keypad BankDatabase CashDispenser BankDatabase CashDispenser
Deposit	displayMessage getInput isEnvelopeReceived Credit	Screen Keypad DepositSlot BankDatabase
BankDatabase	validatePIN getAvailableBalance getTotalBalance debit Credit	Account Account Account Account Account

**Figura 7.25 | Colaborações no sistema ATM.**



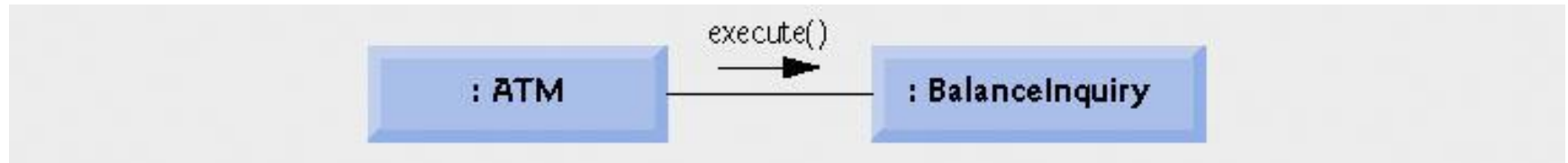
## 7.14 (Opcional) Estudo de caso de engenharia de software (*Continuação*)

- **Diagramas de interação:**
  - Interações modelo utilizam UML.
  - Diagramas de comunicação.
    - Também chamados de diagramas de colaboração.
    - Enfatizam quais objetos participam das colaborações.
  - Diagramas de sequência:
    - Enfatizam quando mensagens são enviadas entre objetos.

## 7.14 (Opcional) Estudo de caso de engenharia de software (*Continuação*)

- **Diagramas de comunicação:**
  - **Objetos:**
    - Modelados como retângulos.
    - Contêm nomes na forma **nomeDoObjeto : nomeDaClasse**
  - **Objetos são conectados por linhas sólidas.**
  - **Mensagens são passadas ao longo dessas linhas na direção mostrada pelas setas.**
  - **O nome da mensagem aparece ao lado da seta.**



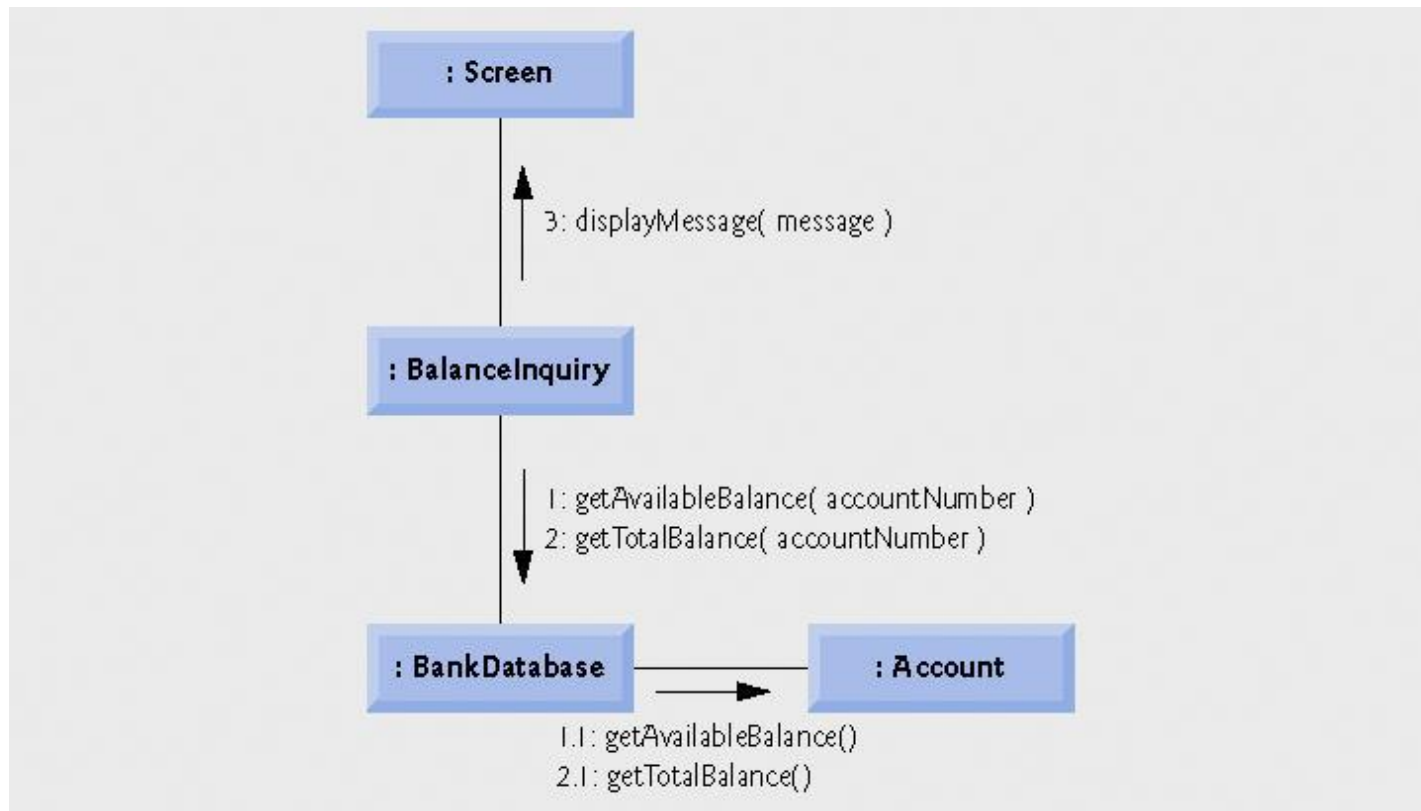


**Figura 7.26** | Diagrama de comunicação do ATM executando uma consulta de saldo.

## 7.14 (Opcional) Estudo de caso de engenharia de software (*Continuação*)

- **Seqüência de mensagens em um diagrama de comunicação:**
  - Aparece à esquerda do nome de uma mensagem.
  - Indica a ordem em que a mensagem é passada.
  - Processa na ordem numérica descendente.





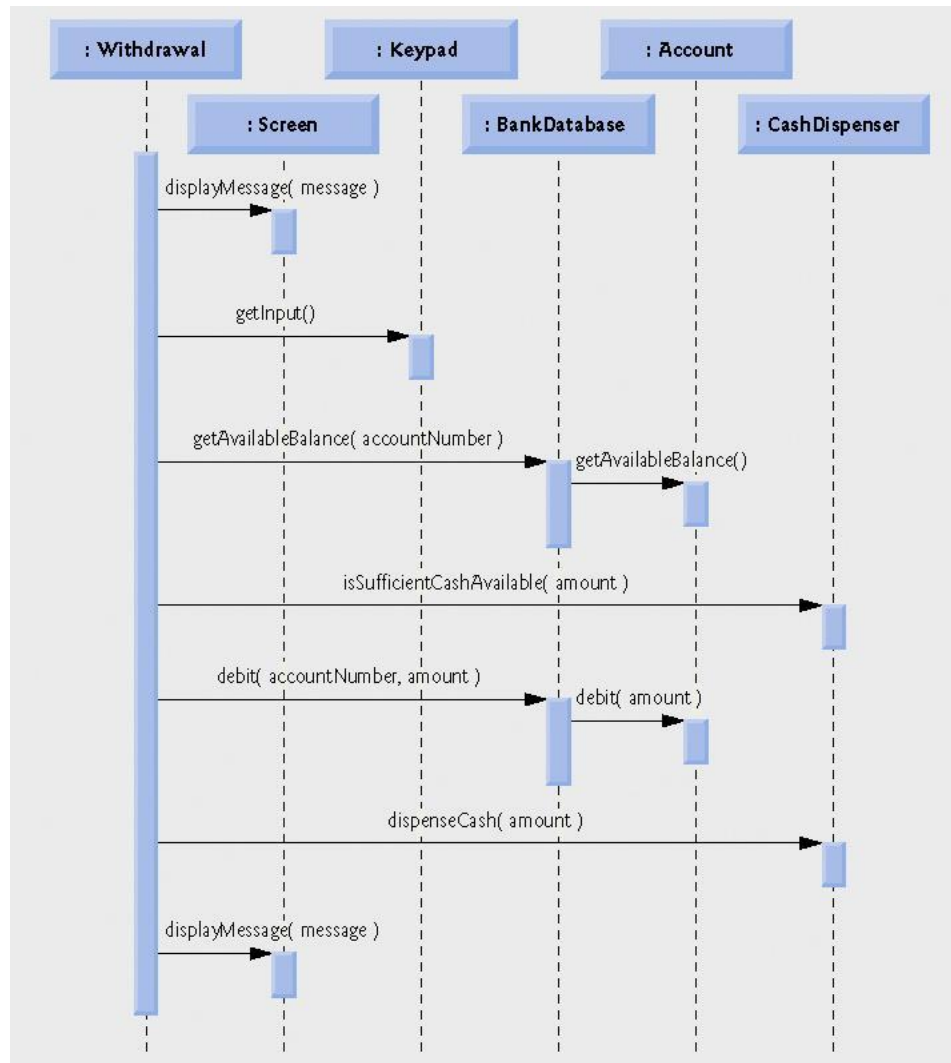
**Figura 7.27 | Diagrama de comunicação para executar uma consulta de saldo.**



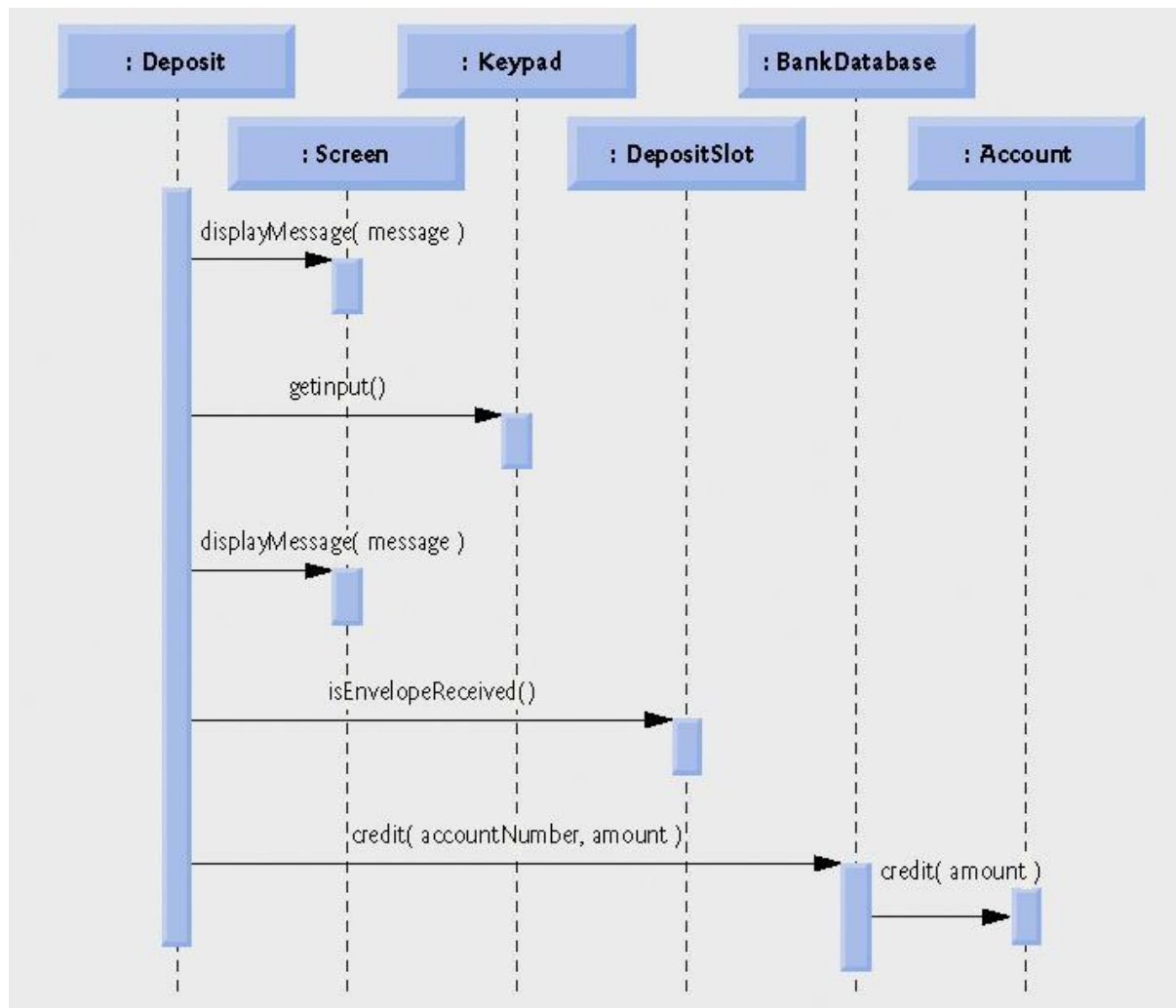
## 7.14 (Opcional) Estudo de caso de engenharia de software (*Continuação*)

- **Diagramas de seqüência:**
  - Ajudam a modelar a sincronização das colaborações.
  - **Linha da vida:**
    - Linha pontilhada que se estende para baixo a partir do retângulo de um objeto:
      - Representa a progressão do tempo.
  - **Ativação:**
    - Pequeno retângulo vertical:
      - Indica que um objeto está em execução.





**Figura 7.28** | Diagrama de seqüência que modela um withdrawal em execução.



**Figura 7.29 |** Diagrama de seqüência que modela um Depo<sup>s</sup>it em execução.