4

Instruções de controle: Parte 1

OBJETIVOS

- Neste capítulo, você aprenderá
- A utilizar técnicas básicas de solução de problemas.
- A desenvolver algoritmos por meio do processo de refinamento passo a passo de cima para baixo.
- A utilizar instruções de seleção `if` e `if…else` a fim de escolher ações alternativas.
- A utilizar a instrução de repetição `while` para executar instruções em um programa repetidamente.
- A utilizar repetição controlada por contador e repetição controlada por sentinela.
- A utilizar os operadores de atribuição, incremento e decremento.

4.2 Algoritmos

- **Algoritmos:**
  - **As *ações* a executar.**
  - **A *ordem* em que essas ações executam.**
- **Controle do programa:**
  - **Especifica a ordem em que as ações são executadas em um programa.**

4.3 Pseudocódigo

- **Pseudocódigo:**
  - **Um idioma informal semelhante ao inglês.**
  - **Ajuda os programadores a desenvolver algoritmos.**
  - **Não executa em computadores.**
  - **Deve conter ações de entrada, saída e cálculo.**
  - **Não deve conter declarações de variáveis.**

4.4 Estruturas de controle

- **Execução seqüencial:**
  - **Instruções são normalmente executadas uma após a outra na ordem em que são escritas.**
- **Transferência do controle:**
  - **Especifica a próxima instrução a executar que não necessariamente é a próxima na seqüência.**
  - **Pode ser realizada pela instrução `goto`.**
    - **A programação estruturada eliminou instruções `goto`.**

4.4 Estruturas de controle (*Cont.*)

- **Pesquisa de Bohm e Jacopini:**
  - **Demonstrou que programas poderiam ser escritos sem instruções `goto`.**
  - **Demonstrou que todos os programas poderiam ser escritos com três estruturas de controle:**
    - **a estrutura de seqüência;**
    - **a estrutura de seleção; e**
    - **a estrutura de repetição.**

4.4 Estruturas de controle (*Cont.*)

- **Diagrama de atividades UML (www.uml.org):**
  - **Modela o fluxo de trabalho (ou atividade) de uma parte de um sistema de software.**
  - **Símbolos do estado de ação (retângulos com seus lados substituídos por arcos curvados para fora):**
    - **Representam expressões da ação que especificam as ações a realizar.**
  - **Losangos:**
    - **Símbolos de decisão (explicados na Seção 4.5).**
    - **Símbolos de mesclagem (explicados na Seção 4.7).**

4.4 Estruturas de controle (*Cont.*)

- **Círculos pequenos:**
  - **Círculo sólido representa o estado inicial da atividade.**
  - **Círculo sólido cercado por um círculo oco representa o estado final da atividade.**
- **Setas de transição:**
  - **Indicam a ordem em que as ações são realizadas.**
- **Notas (retângulos com o canto superior direito dobrado):**
  - **Explicam os propósitos dos símbolos (como comentários em Java).**
  - **São conectadas aos símbolos que elas descrevem por linhas pontilhadas.**

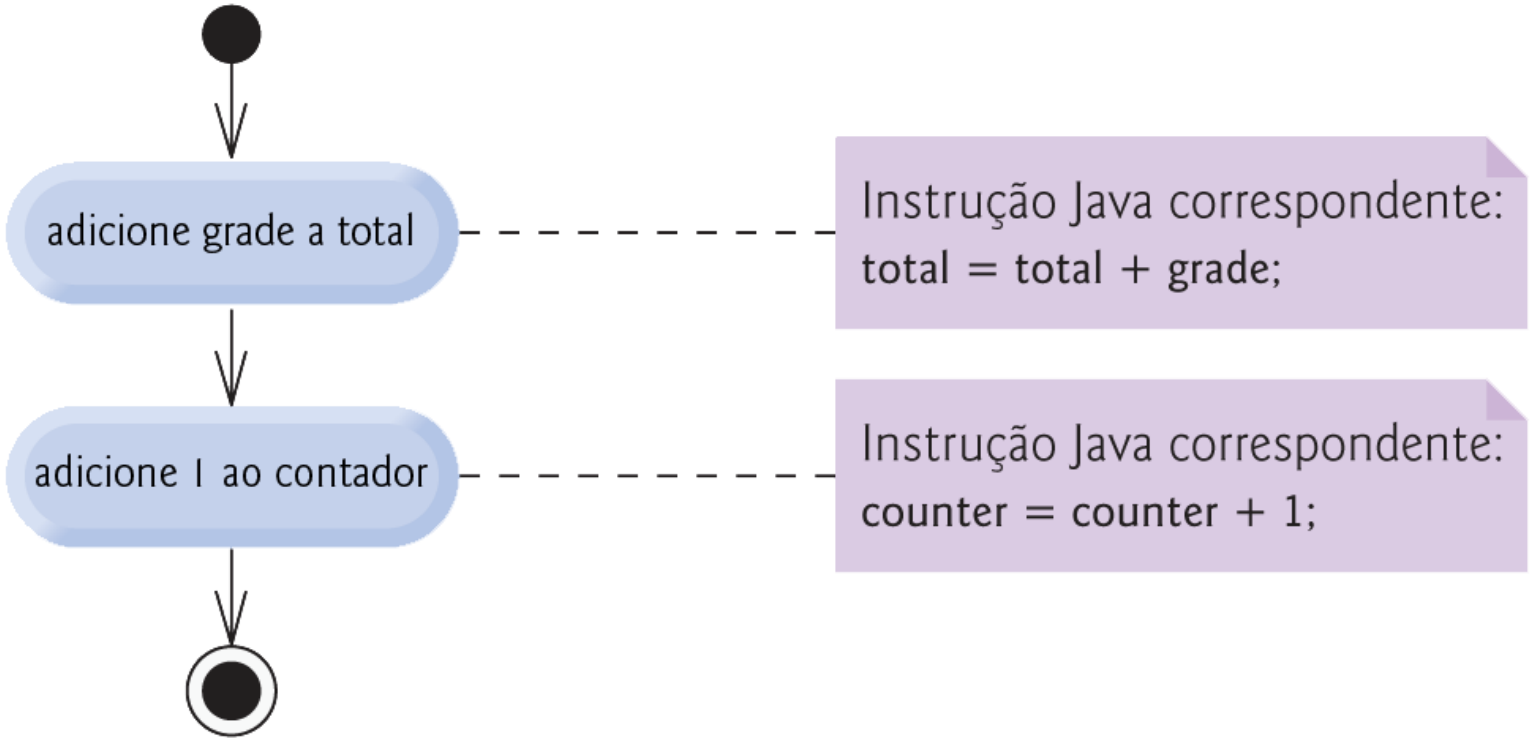

Figura 4.1 | Diagrama de atividades da estrutura de seqüência.

4.4 Estruturas de controle (*Cont.*)

- **Instruções de seleção em Java:**
  - **Instrução `if`:**
    - **Instrução de uma única seleção.**
  - **Instrução `if...else`:**
    - **Instrução de seleção dupla.**
  - **Instrução `switch`:**
    - **Instrução de seleção múltipla.**

4.4 Estruturas de controle (*Cont.*)

- **Instruções de repetição em Java:**
  - **Também conhecidas como *instruções de loop*.**
  - **Realizam repetidamente uma ação enquanto a condição de continuação do loop permanecer verdadeira.**
  - **Instrução `while`:**
    - **Realiza as ações no seu corpo zero ou mais vezes.**
  - **Instrução `do…while`:**
    - **Realiza as ações no seu corpo uma ou mais vezes.**
  - **Instrução `for`:**
    - **Realiza as ações no seu corpo zero ou mais vezes.**

4.4 Estruturas de controle (*Cont.*)

- **O Java tem três tipos de estruturas de controle:**
  - **instrução de seqüência;**
  - **instruções de seleção (três tipos); e**
  - **instruções de repetição (três tipos).**
  - **Todos os programas são compostos por essas instruções de controle.**
    - **Empilhamento de instruções de controle:**
      - **Todas as instruções de controle são de entrada única/saída única.**
    - **Aninhamento de instruções de controle.**

4.5 A instrução de uma única seleção `if`

- **Instruções `if`:**
  - **Executam uma ação se a condição especificada for verdadeira.**
  - **Podem ser representadas por um símbolo de decisão (losango) em um diagrama de atividades UML.**
    - **Setas de transição fora de um símbolo de decisão têm condições de guarda.**
      - **O fluxo de trabalho segue a seta de transição cuja condição de guarda é verdadeira.**

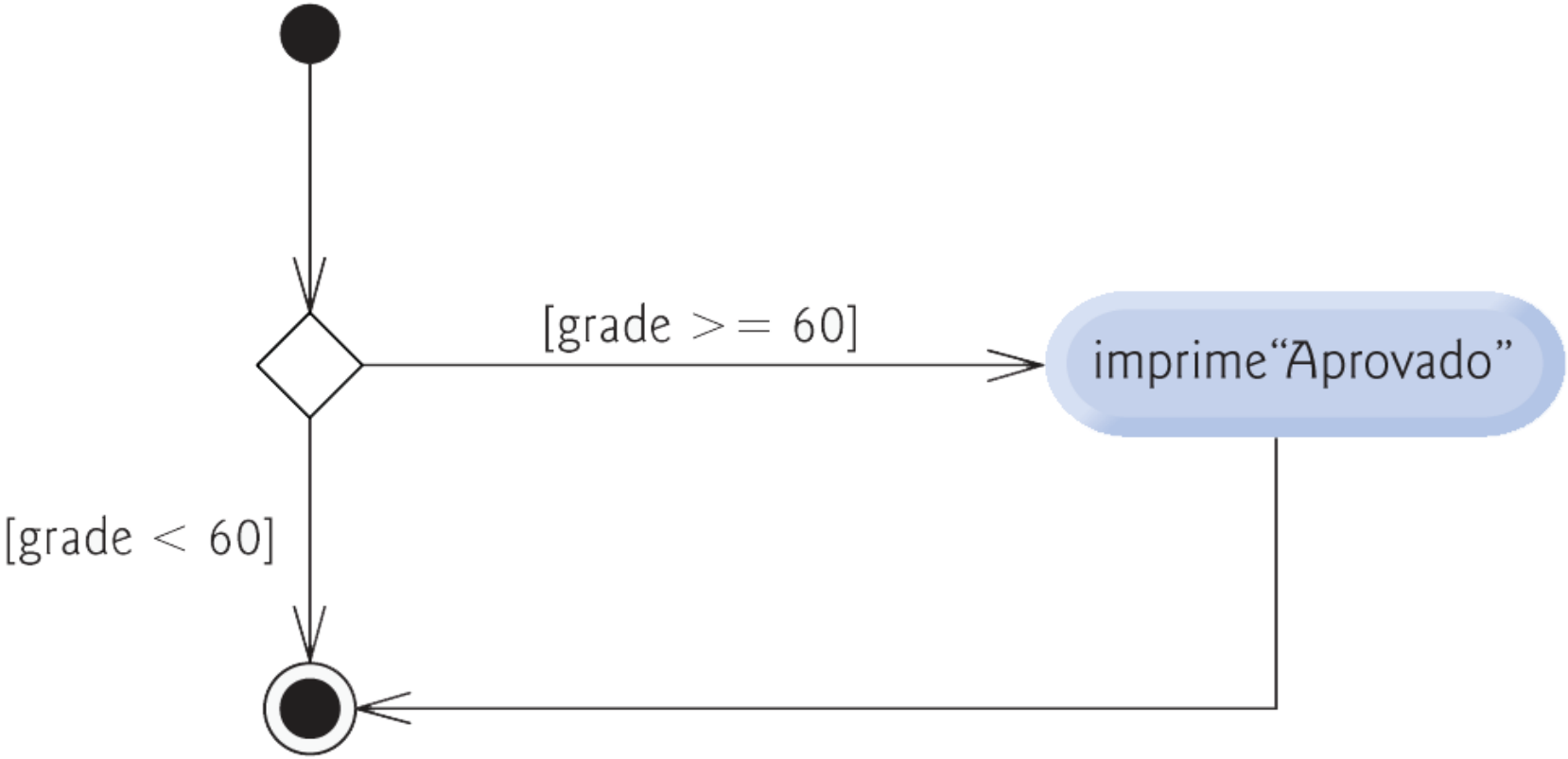

Figura 4.2 | Diagrama de atividades da UML de uma instrução de seleção única `if`.

4.6 A instrução de seleção dupla `if...else`

- **Instrução `if...else`:**
  - **Executa uma ação se a condição especificada for `true` ou uma ação diferente se a condição especificada for `false`.**
- **Operador condicional (`? :`):**
  - **Único *operador ternário* do Java (recebe três operandos).**
  - **`? :` e seus três operandos formam uma *expressão condicional*.**
    - **Toda uma expressão condicional é avaliada para o segundo operando se o primeiro operando for `true`.**
    - **Toda uma expressão condicional é avaliada para o terceiro operando se o primeiro operando for `false`.**

Boa prática de programação 4.1

Recue ambas as instruções do corpo de uma instrução `if...else`.

Boa prática de programação 4.2

Caso existam vários níveis de recuo, cada nível deve ser recuado pela mesma quantidade adicional de espaços.

Boa prática de programação 4.3

As expressões condicionais são mais difíceis de ler do que as instruções `if...else` e devem ser utilizadas para substituir somente instruções `if...else` simples que escolhem entre dois valores.

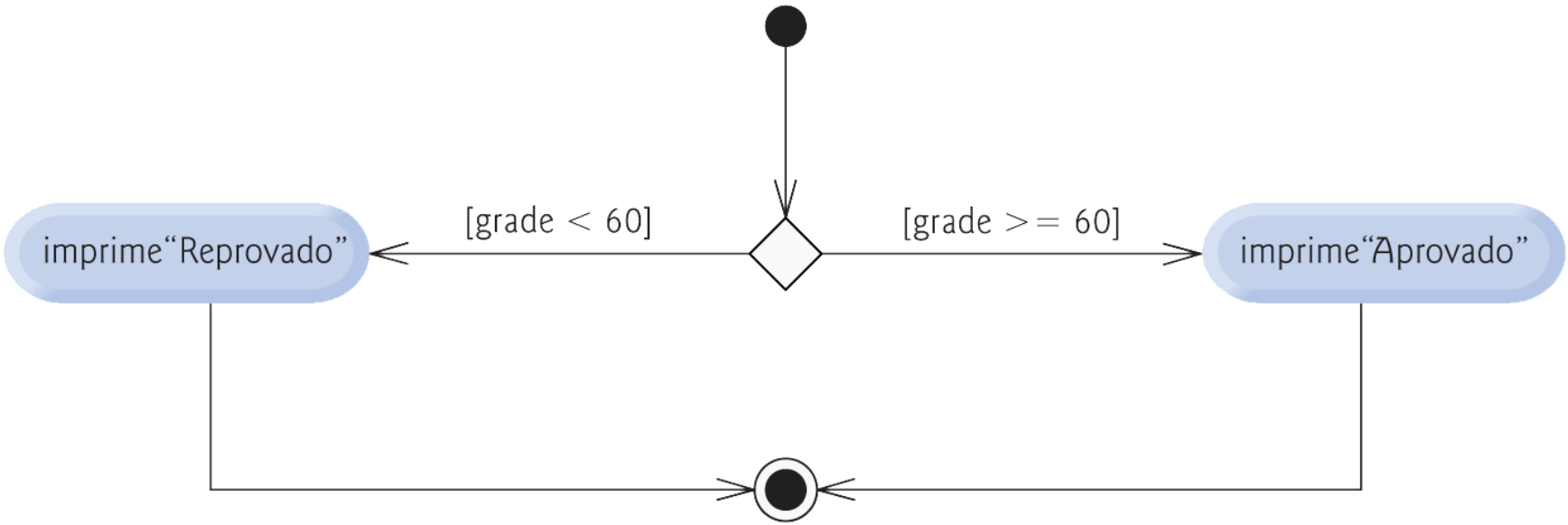

Figura 4.3 | Diagrama de atividades da UML de uma de instrução de seleção dupla `if...else`.

4.6 A instrução de seleção dupla `if...else` (*Continuação*)

- **Instruções `if...else` aninhadas:**
  - **Instruções `if...else` podem ser colocadas dentro de outras instruções `if...else`.**
- **O problema do `else` oscilante:**
  - **Instruções `else` sempre estão associadas com a `if` imediatamente precedente, a menos que seja especificado o contrário pelas chaves `{ }`.**
- **Blocos:**
  - **Chaves `{ }` associam instruções a blocos.**
  - **Blocos podem substituir instruções individuais, como o corpo de uma `if`.**

4.6 A estrutura de seleção dupla `if...else` (*Continuação*)

- **Erros de lógica:**
  - **Um *erro fatal de lógica* faz com que um programa falhe e finalize prematuramente.**
  - **Um *erro de lógica não-fatal* permite que o programa continue a executar, mas o faz produzir resultados incorretos.**
- **Instruções vazias:**
  - **Representadas colocando-se um ponto-e-vírgula (`;`) onde uma instrução normalmente estaria. Podem ser utilizadas como o corpo de uma `if`.**

Erro comum de programação 4.1

Esquecer uma ou ambas as chaves que delimitam um bloco pode levar a erros de sintaxe ou erros de lógica em um programa.

Boa prática de programação 4.4

Sempre utilizar as chaves em uma instrução `if...else` (ou outra) ajuda a evitar uma omissão acidental, especialmente ao adicionar instruções à parte `if` ou à parte `else` mais tarde.

Para evitar omitir uma ou as duas chaves, alguns programadores digitam as chaves de abertura ou fechamento de blocos antes de digitar as instruções individuais dentro das chaves.

Erro comum de programação 4.2

Colocar um ponto-e-vírgula depois da condição em uma instrução `if` ou `if...else` resulta em um erro de lógica em instruções `if` de uma única seleção e um erro de sintaxe em instruções `if...else` de seleção dupla (quando a parte `if` contém uma instrução completa).

4.7 A instrução de repetição `while`

- **Instrução `while`:**
  - **Repete uma ação enquanto a condição de continuação do loop permanecer verdadeira.**
  - **Utiliza um *símbolo de agregação* no seu diagrama de atividades UML.**
    - **Mescla dois ou mais fluxos de trabalho.**
    - **É representado por um losango (como com os símbolos de decisão), mas com:**
      - **múltiplas setas de transição ‘incoming’;**
      - **somente uma seta de transição ‘outgoing’; e**
      - **nenhuma condição de guarda em uma seta de transição.**

Erro comum de programação 4.3

Não fornecer, no corpo de uma instrução `while`, uma ação que conseqüentemente faz com que a condição na instrução `while` torne-se falsa em geral resulta em um erro de lógica chamado *loop infinito*, no qual o loop nunca termina.

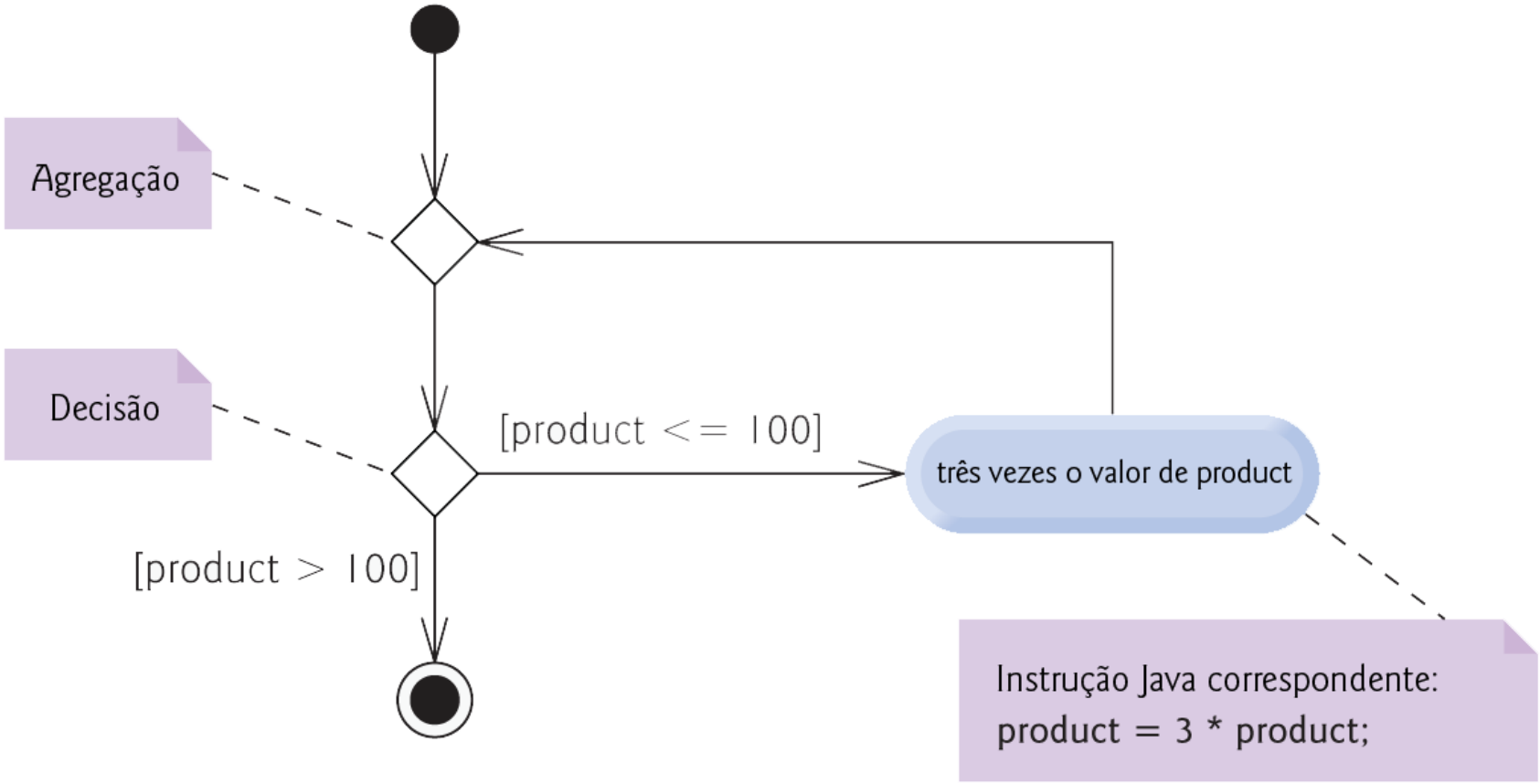

Figura 4.4 | Diagrama de atividades da UML da instrução de repetição `while`.

4.8 Formulando algoritmos: Repetição controlada por contador

- **Repetição controlada por contador:**
  - **Utiliza uma variável contadora para contar o número de vezes que um loop é iterado.**
- **Divisão de inteiros:**
  - **A parte fracionária de um cálculo de divisão de inteiros é truncada (descartada).**

```
1     Configure o total como zero
2     Configure o contador de notas como um
3
4     Enquanto contador de notas for menor ou igual a dez
5         Solicite para o usuário inserir a próxima nota
6         Insira a próxima nota
7         Adicione a nota ao total
8         Adicione um ao contador de notas
9
10    Configure a média da classe como o total dividido por dez
11    Imprima a média da classe
```

Figura 4.5 | Algoritmo em pseudocódigo que utiliza repetição controlada por contador para resolver o problema de média da classe.

Resumo

GradeBook.java

(1 de 3)

```
1  // Fig. 4.6: GradeBook.java
2  // Classe GradeBook que resolve o problema da média da classe utilizando
3  // repetição controlada por contador.
4  import java.util.Scanner; // programa utiliza a classe Scanner
5
6  public class GradeBook
7  {
8     private String courseName; // nome do curso que essa GradeBook representa
9
10    // construtor inicializa courseName
11    public GradeBook( String name )
12    {
13       courseName = name; // inicializa courseName
14    } // fim do construtor
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
```

Atribui um valor à variável de instância **courseName**

Declara o método **setCourseName**

Declara o método **getCourseName**

Resumo

GradeBook.java

(2 de 3)

```
28    // exibe uma mensagem de boas-vindas para o usuário de 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    // determina a média da classe com base em 10 notas inseridas pelo usuário
37    public void determineClassAverage()
38    {
39       // cria Scanner para obter entrada a partir da janela de comando
40       Scanner input = new Scanner( System.in );
41
42       int total; // soma das notas inseridas pelo usuário
43       int gradeCounter; // número da nota a ser inserida a seguir
44       int grade; // valor da nota inserida pelo usuário
45       int average; // média das notas
46
47       // fase de inicialização
48       total = 0; // inicializa o total
49       gradeCounter = 1; // inicializa o contador de loops
50
```

Declara o método **displayMessage**

Declara o método **determineClassAverage**

Declara e inicializa a variável **Scanner input**

Declara as variáveis **int** locais, **total**, **gradeCounter**, **grade** e **average**

```
51       // fase de processamento
52       while ( gradeCounter <= 10 ) // faz o loop 10 vezes
53       {
54          System.out.print( "Enter grade: " ); // prompt
55          grade = input.nextInt(); // insere a próxima nota
56          total = total + grade; // adiciona grade a total
57          gradeCounter = gradeCounter + 1; // incrementa o contador por 1
58       } // fim do while
59
60       // fase de término
61       average = total / 10; // divisão de inteiros produz um resultado inteiro
62
63       // exibe o total e a média das notas
64       System.out.printf( "\nTotal of all 10 grades is %d\n", total );
65       System.out.printf( "Class average is %d\n", average );
66    } // fim do método determineClassAverage
67
68 } // fim da classe GradeBook
```

O loop **`while`** itera enquanto **`gradeCounter`** < = **10**

Incrementa o contador da variável **`gradeCounter`**

Calcula a nota média

Exibe os resultados

Resumo

GradeBook.java

(3 de 3)

Boa prática de programação 4.5

Separe as declarações de outras instruções nos métodos com uma linha em branco para legibilidade.

Observação de engenharia de software 4.1

A experiência tem mostrado que a parte mais difícil de resolver um problema em um computador é desenvolver o algoritmo para a solução.

Uma vez que um algoritmo correto foi especificado, o processo de produção de um programa Java funcional a partir do algoritmo é, em geral, simples.

Erro comum de programação 4.4

Utilizar o valor de uma variável local antes de ela ser inicializada resulta em um erro de compilação.

Todas as variáveis locais devem ser inicializadas antes de seus valores serem utilizados nas expressões.

Dica de prevenção de erro 4.1

Inicialize cada contador e total em sua declaração ou em uma instrução de atribuição.

Normalmente, os totais são inicializados como 0. Os contadores, em geral, são inicializados como 0 ou 1, dependendo de como eles são utilizados (mostraremos exemplos de quando utilizar 0 e quando utilizar 1).

Resumo

GradeBookTest.java

```
1  // Fig. 4.7: GradeBookTest.java
2  // Cria o objeto da classe GradeBook e invoca seu método determineClassAverage
3
4  public class GradeBookTest
5  {
6     public static void main( String args[] )
7     {
8        // cria o objeto myGradeBookda classe GradeBook e
9        // passa o nome de cursor para o construtor
10       GradeBook myGradeBook = new GradeBook(
11          "CS101 Introduction to Java Programming" );
12
13       myGradeBook.displayMessage();// exibe a mensagem welcome
14       myGradeBook.determineClassAverage();// calcula a média das 10 notas
15    } // fim de main
16
17 } // fim da classe GradeBookTest
```

Cria um novo objeto **GradeBook**

Passa o nome do curso para o construtor **GradeBook** como uma **string**

Chama o método **determineClassAverage** de **GradeBook**

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

Enter grade: 67
Enter grade: 78
Enter grade: 89
Enter grade: 67
Enter grade: 87
Enter grade: 98
Enter grade: 93
Enter grade: 85
Enter grade: 82
Enter grade: 100

Total of all 10 grades is 846
Class average is 84
```

Erro comum de programação 4.5

Assumir que divisão de inteiros arredonda (em vez de truncar) pode levar a resultados incorretos.

Por exemplo, 7 ÷ 4, que produz 1,75 na aritmética convencional, é truncado para 1 na aritmética de inteiros, em vez de arredondado para 2.

4.9 Formulando algoritmos: Repetição controlada por sentinela

- **Repetição controlada por sentinela:**
  - **Também conhecida como repetição indefinida.**
  - **Utiliza um *valor de sentinela* (também conhecido como *valor de sinal*, *valor fictício* ou *valor de flag*).**
    - **Um valor de sentinela não pode ser também um valor válido de entrada.**

4.9 Formulando algoritmos: Repetição controlada por sentinela (*Continuação*)

- **Refinamento passo a passo de cima para baixo:**
  - **Passo superior: a instrução individual contendo a função geral do programa.**
  - **Primeiro refinamento: múltiplas instruções que utilizam somente a estrutura de seqüência.**
  - **Segundo refinamento: aplicado a variáveis específicas, utiliza estruturas de controle específicas.**

Erro comum de programação 4.6

Escolher um valor de sentinela que também seja um valor legítimo de dados é um erro de lógica.

Observação de engenharia de software 4.2

Cada refinamento, bem como a própria parte superior, é uma especificação completa do algoritmo — somente o nível de detalhe varia.

Observação de engenharia de software 4.3

Muitos programas podem ser divididos logicamente em três fases: uma fase de inicialização que inicializa as variáveis; uma fase de processamento que insere os valores dos dados e ajusta as variáveis do programa (por exemplo, contadores e totais) de maneira correspondente; e uma fase de término que calcula e gera a saída dos resultados finais.

Dica de prevenção de erro 4.2

Ao realizar a divisão de uma expressão cujo valor poderia ser zero, teste essa possibilidade explicitamente e trate-a de maneira adequada no seu programa — por exemplo, imprimindo uma mensagem de erro —, em vez de permitir que o erro ocorra.

```
1   Inicialize total como zero
2   Inicialize o contador como zero
3
4   Solicite que o usuário insira a primeira nota
5   Insira a primeira nota (possivelmente a sentinela)
6
7   Enquanto (While) o usuário não inserir a sentinela
8       Adicione essa nota à soma total
9       Adicione um ao contador de notas
10      Solicite para o usuário inserir a próxima nota
11      Insira a próxima nota (possivelmente a sentinela)
12
13  If (Se) o contador não for igual a zero
14      Configure a média como o total dividido pelo contador
15      Imprima a média
16  else (caso contrário)
17      Imprima "Nenhuma nota foi inserida"
```

Figura 4.8 | Algoritmo em pseudocódigo do problema de média da classe com repetição controlada por sentinela.

Observação de engenharia de software 4.4

Termine o processo de refinamento passo a passo de cima para baixo depois de ter especificado o algoritmo em pseudocódigo com detalhes suficientes para que você possa converter o pseudocódigo em Java.

Normalmente, implementar o programa Java é simples e direto.

Observação de engenharia de software 4.5

Alguns programadores experientes escrevem programas sem jamais utilizar ferramentas de desenvolvimento de programa como pseudocódigo. Eles acreditam que seu objetivo final é resolver o problema em um computador e que escrever pseudocódigo só retarda a produção das saídas finais.

Embora esse método talvez funcione para problemas simples e conhecidos, ele pode levar a erros sérios e atrasos em projetos grandes e complexos.

Resumo

GradeBook.java

(1 de 3)

```
1  // Fig. 4.9: GradeBook.java
2  // Classe GradeBook que resolve o programa da média da classe utilizando
3  // repetição controlado por sentinela.
4  import java.util.Scanner; // programa utiliza a classe Scanner
5
6  public class GradeBook
7  {
8     private String courseName; // nome do curso que essa GradeBook representa
9
10    // construtor inicializa courseName
11    public GradeBook( String name )
12    {
13       courseName = name; // inicializa courseName
14    } // fim do construtor
15
16    // método para configurar o nome do curso
17    public void setCourseName( String name )
18    {
19       courseName = name; // store the course name
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
```

Atribui um valor à variável de instância **courseName**

Declara o método **setCourseName**

Declara o método **getCourseName**

```
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    // determina a média de um número arbitrário de notas
37    public void determineClassAverage()
38    {
39       // cria Scanner para obter entrada a partir da janela de comando
40       Scanner input = new Scanner( System.in );
41
42       int total; // soma das notas
43       int gradeCounter; // número de notas inseridas
44       int grade; // valor da nota
45       double average; // número com ponto de fração decimal para a m
46
47       // fase de inicialização
48       total = 0; // inicializa o total
49       gradeCounter = 0; // inicializa o contador de l
50
51       // fase de processamento
52       // solicita entrada e lê a nota do usuário
53       System.out.print( "Enter grade or -1 to quit: " );
54       grade = input.nextInt();
55
```

Declara o método **displayMessage**

Resumo

GradeBook.java

(2 de 3)

Declara o método **determineClassAverage**

Declara e inicializa a variável **Scanner input**

Declara as variáveis **int** locais, **total**, **gradeCounter** e **grade**, e a variável **double average**

Resumo

GradeBook.java

(3 de 3)

```
56       // faz um loop até ler o valor de sentinela inserido pelo usuário
57       while ( grade != -1 )
58       {
59          total = total + grade; // add grade to total
60          gradeCounter = gradeCounter + 1; // increment counter
61
62          // solicita entrada e lê a próxima nota fornecida pelo usuário
63          System.out.print( "Enter grade or -1 to quit: " );
64          grade = input.nextInt();
65       } // fim do while
66
67       // fase de término
68       // se usuário inseriu pelo menos uma nota...
69       if ( gradeCounter != 0 )
70       {
71          // calcula a média de todas as notas inseridas
72          average = (double) total / gradeCounter;
73
74          // exibe o total e a média (com dois dígitos de precisão)
75          System.out.printf( "\nTotal of the %d grades entered is %d\n",
76             gradeCounter, total );
77          System.out.printf( "Class average is %.2f\n", average );
78       } // fim do if
79       else // nenhuma nota foi inserida, assim gera a saída da mensagem apropriada
80          System.out.println( "No grades were entered" );
81    } // fim do método determineClassAverage
82
83 } // fim da classe GradeBook
```

O loop **`while`** itera enquanto **`grade !=`** o valor de sentinela, **`-1`**

Calcula a nota média utilizando **`(double)`** para realizar uma conversão explícita

Exibe a nota média

Exibe uma mensagem **“No grades were entered”**

Boa prática de programação 4.6

Em um loop controlado por sentinela, os prompts solicitando entrada de dados devem lembrar explicitamente o usuário do valor da sentinela.

Erro comum de programação 4.7

Omitir as chaves que delimitam um bloco pode levar a erros de lógica, como loops infinitos.

Para evitar esse problema, alguns programadores incluem o corpo de cada instrução de controle dentro de chaves — mesmo se o corpo contiver somente uma única instrução.

4.9 Formulando algoritmos: Repetição controlada por sentinela (*Continuação*)

- **Operador unário de coerção:**
  - **Cria uma cópia temporária do seu operando com um tipo de dados diferente.**
    - **Exemplo: `(double)` criará uma cópia *temporária* de ponto flutuante do seu operando `total`.**
  - ***Conversão explícita*.**
- ***Promoção* (ou *conversão implícita*):**
  - **Converter um valor (por exemplo, `int`) em um outro tipo de dados (por exemplo, `double`) para realizar um cálculo.**

Erro comum de programação 4.8

O operador de coerção pode ser utilizado para converter entre tipos numéricos e primitivos, como `int` e `double` e entre tipos por referência relacionados (como discutiremos no Capítulo 10, Programação orientada a objetos: Polimorfismo).

Aplicar uma coerção ao tipo errado pode causar erros de compilação ou erros de tempo de execução.

Resumo

GradeBookTest.java

```
1  // Fig. 4.10: GradeBookTest.java
2  // Cria o objeto da classe GradeBook e invoca seu método determineClassAverage
3
4  public class GradeBookTest
5  {
6     public static void main( String args[] )
7     {
8        // cria o objeto myGradeBookda classe GradeBook e
9        // passa o nome de cursor para o construtor
10       GradeBook myGradeBook = new GradeBook(
11          "CS101 Introduction to Java Programming" );
12
13       myGradeBook.displayMessage();exibe a mensagem de boas-vindas
14       myGradeBook.determineClassAverage(); // calcula a média das notas
15    } // fim de main
16
17 } // fim da classe GradeBookTest
```

Cria um novo objeto **GradeBook**

Passa o nome do curso para o construtor **GradeBook** como uma **string**

Chama o método **determineClassAverage** de **GradeBook**

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

Enter grade or -1 to quit: 97
Enter grade or -1 to quit: 88
Enter grade or -1 to quit: 72
Enter grade or -1 to quit: -1

Total of the 3 grades entered is 257
Class average is 85.67
```

4.10 Formulando algoritmos: Instruções de controle aninhadas

- **Instruções de controle podem ser aninhadas uma dentro da outra:**
  - **Colocar uma instrução de controle dentro do corpo de outra.**

```
1  Inicialize as aprovações como zero
2  Inicialize as reprovações como zero
3  Inicialize o contador de alunos como um
4
5  Enquanto o contador de alunos for menor ou igual a 10
6     Solicite que o usuário insira o próximo resultado de exame
7     Insira o próximo resultado de exame
8
9  If (Se) o aluno foi aprovado
10     Adicione um a aprovações
11 Else (caso contrário)
12    Adicione um a reprovações
13
14    Adicione um ao contador de aluno
15
16 Imprima o número de aprovações
17 Imprima o número de reprovações
18
19 Se mais de oito alunos forem aprovados
20    Imprima "Elevar a taxa de matrícula"
```

Figura 4.11 | Pseudocódigo para o problema dos resultados do exame.

```
1  // Fig. 4.12: Analysis.java
2  // Análise dos resultados dos exames.
3  import java.util.Scanner; // classe utiliza a classe Scanner
4
5  public class Analysis
6  {
7     public void processExamResults
8     {
9        // cria Scanner para obter entrada a partir da janela de comando
10       Scanner input = new Scanner( System.in );
11
12       // inicializando variáveis nas declarações
13       int passes = 0; // número de aprovações
14       int failures = 0; // número de reprovações
15       int studentCounter = 1; // contador de alunos
16       int result; // um resultado do exame (obtém o valor a partir do usuário)
17
18       // processa 10 alunos utilizando o loop controlado por contador
19       while ( studentCounter <= 10 )
20       {
21          // solicita ao usuário uma entrada e obtém valor fornecido pelo usuário
22          System.out.print( "Enter result (1 = pass, 2 = fail): " );
23          result = input.nextInt();
24
```

Declara as variáveis locais de **`processExamResults`**

Resumo

Analysis.java

(1 de 2)

O loop **`while`** itera enquanto **`studentCounter < = 10`**

```
25          // if...else aninhado em um while
26          if ( result == 1 )            // se resultar 1,
27             passes = passes + 1;       // incrementa aprovações;
28          else                          // caso contrário, resultado não é 1, então
29             failures = failures + 1; // incrementa reprovações
30
31          // incrementa studentCounter até o loop terminar
32          studentCounter = studentCounter + 1;
33       } // fim do while
34
35       // fase de término; prepara e exibe os resultados
36       System.out.printf( "Passed: %d\nFailed: %d\n", passes, failures );
37
38       // determina se mais de 8 alunos foram aprovados
39       if ( passes > 8 )
40          System.out.println( "Raise Tuition" );
41    } // fim do método processExamResults
42
43 } // fim da classe Analysis
```

Determina se esse aluno passou ou não e incrementa a variável apropriada

Determina se mais de oito alunos passaram no exame

Resumo

Analysis.java

(2 de 2)

Dica de prevenção de erro 4.3

Inicializar variáveis locais quando são declaradas ajuda o programador a evitar quaisquer erros de compilação que poderiam surgir de tentativas para utilizar dados não inicializados. Embora o Java não exija que as inicializações das variáveis locais sejam incorporadas a declarações, ele exige que variáveis locais sejam inicializadas antes de seus valores serem utilizados em uma expressão.

Resumo

AnalysisTest.java

```
1  // Fig. 4.13: AnalysisTest.java
2  // Programa de teste para classe Analysis.
3
4  public class AnalysisTest
5  {
6     public static void main( String args[] )
7     {
8        Analysis application = new Analysis(); // cria o objeto da classe Analysis
9        application.processExamResults(); // chama o método para processar os resultados
10    } // fim de main
11
12 } // fim da classe AnalysisTest
```

Cria um objeto `Analysis`

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 9
Failed: 1
Raise Tuition
```

Mais de 8 alunos passaram no exame

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 6
Failed: 4
```

4.11 Operadores de atribuição compostos

- **Operadores de atribuição compostos:**
  - **Uma instrução de atribuição na forma *variável* = *expressão do operador de variável*; onde *operador* é +, -, *, / ou % pode ser escrita como: *operador de variável*= *expressão*;**
  - **Exemplo: `c = c + 3;` pode ser escrito como**
  - `c += 3;`
    - **Essa instrução adiciona `3` ao valor na variável `c` e armazena o resultado na variável `c`.**

Operador de atribuição	Expressão de exemplo	Explicação	Atribuições
Suponha: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`C = c + 7`	`10ac`
`-=`	`d -= 4`	`d = d - 4`	`1ad`
`*=`	`e *= 5`	`e = e * 5`	`20ae`
`/=`	`f /= 3`	`f = f / 3`	`2af`
`%=`	`g %= 9`	`g = g % 9`	`3ag`

Figura 4.14 | Operadores de atribuição compostos aritméticos.

4.12 Operadores de incremento e decremento

- **Operadores de incremento e decremento unários**
  - **Operador de incremento unário (++) adiciona 1 ao seu operando.**
  - **Operador de decremento unário (--) subtrai 1 do seu operando.**
  - **Operador de pré-incremento (e pré-decremento):**
    - **Altera o valor do seu operando e então utiliza o novo valor do operando na expressão em que a operação aparece**
  - **Operador de pós-incremento (e pós-decremento):**
    - **Utiliza o valor atual do seu operando na expressão em que a operação aparece e então altera o valor do operando**

Boa prática de programação 4.7

Diferentemente dos operadores binários, os operadores de incremento e decremento unários e devem ser colocados ao lado dos seus operandos, sem espaços no meio.

Operador	Chamado	Expressão de exemplo	Explicação
`++`	pré-incremento	`++a`	Incrementa `a` por `1` e então utiliza o novo valor de `a` na expressão em que `a` reside.
`++`	pós-decremento	`a++`	Utilize o valor atual de a na expressão em que a reside, então incremente `a` por `1`.
`--`	pré-incremento	`--b`	Decrementa `b` por `1` e então utiliza o novo valor de `b` na expressão em que `b` reside.
`--`	pós-decremento	`b--`	Utilize o valor atual de b na expressão em que b reside, então decrementa `b` por `1`.

Figura 4.15 | Operadores de incremento e de decremento.

Slide 66

EFz1 OK

Edson Furmankiewicz; 21/10/2005

Resumo

Increment.java

```
1  // Fig. 4.16: Increment.java
2  // Operadores de pré-incremento e pós-decremento.
3
4  public class Increment
5  {
6     public static void main( String args[] )
7     {
8        int c;
9
10       // demonstra o operador de pós-incremento
11       c = 5; // assign 5 to c
12       System.out.println( c );   // imprime 5
13       System.out.println( c++ ); // imprime 5 e então pós-incrementa
14       System.out.println( c );   // imprime 6
15
16       System.out.println(); // skip a line
17
18       // demonstra o operador de pré-incremento
19       c = 5; // assign 5 to c
20       System.out.println( c );   // imprime 5
21       System.out.println( ++c ); // pré-incrementa e então imprime 6
22       System.out.println( c );   // imprime 6
23
24    } // fim de main
25
26 } // fim da classe Increment
```

Pós-incrementando a variável **c**

Pré-incrementando a variável **c**

```
5
5
6

5
6
6
```

Erro comum de programação 4.9

Tentar utilizar o operador de incremento ou decremento em uma expressão diferente daquela a que um valor pode ser atribuído é um erro de sintaxe.

Por exemplo, escrever `++(x + 1)` é um erro de sintaxe, uma vez que `(x + 1)` não é uma variável.

Operadores	Associatividade	Tipo
`++ --`	da direita para a esquerda	unário pós-fixo
`++ -- + - (` *tipo* `)`	da direita para a esquerda	unário pré-fixo
`* / %`	da esquerda para a direita	multiplicativo
`+ -`	da esquerda para a direita	aditivo
`< <= > >=`	da esquerda para a direita	relacional
`== !=`	da esquerda para a direita	igualdade
`?:`	da direita para a esquerda	ternário condicional
`= += -= *= /= %=`	da direita para a esquerda	atribuição

Figura 4.17 | Precedência e associatividade dos operadores discutidos até agora.

Slide 69

EFz2 Na minha tela aparece perfeitamente.
Edson Furmankiewicz; 21/10/2005

4.13 Tipos primitivos

- **O Java é uma *linguagem fortemente tipificada*:**
  - **Todas as variáveis têm um tipo.**
- **Tipos primitivos em Java são portáveis entre todas as plataformas que suportam Java.**

Dica de portabilidade 4.1

Diferente de C e C++, os tipos primitivos em Java são portáveis entre todas as plataformas de computador que suportam Java.

Graças a isso e a muitos outros recursos da portabilidade do Java, um programador pode escrever um programa uma vez e estar certo de que ele executará em qualquer plataforma de computador que suporte o Java.

Essa capacidade é referida às vezes como *WORA* (Write Once, Run Anywhere – Escreva uma vez, execute em qualquer lugar).