Capítulo 4
Instruções de controle: Parte I

Java™ Como Programar, 8/E

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 utilizando pseudocódigo.
- A utilizar instruções de seleção `if` e `if...else` para escolher entre 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 compostos de atribuição, incremento e decremento.
- A portabilidade dos tipos de dados primitivos.

4.1 Introdução

- Antes de escrever um programa para resolver um problema, tenha um entendimento completo do problema e uma abordagem cuidadosamente planejada para resolvê-lo.
- Entenda os tipos de blocos de construção que estão disponíveis e empregue técnicas comprovadas de construção de programas.
- Este capítulo introduz as instruções `if`, `if...else` e `while`

Operadores de atribuição compostos e operadores de incremento e decremento.

Portabilidade dos tipos primitivos do Java.

4.2 Algoritmos

- Qualquer problema de computação pode ser resolvido executando uma série de ações em uma ordem específica.
- Um **algoritmo** é um procedimento para resolver um problema em termos
  - das **ações** a executar e
  - da **ordem** em que essas ações executam
- Considere o "algoritmo cresça e brilhe" seguido por um executivo para sair da cama e ir trabalhar:

  (1) Levantar da cama; (2) tirar o pijama; (3) tomar banho; (4) vestir-se; (5) tomar café da manhã; (6) dirigir o carro até o trabalho.
- Suponha que os mesmos passos sejam seguidos em uma ordem um pouco diferente:

  (1) Levantar de cama; (2) tirar o pijama; (3) vestir-se; (4) tomar banho; (5) tomar café da manhã; (6) dirigir o carro até o trabalho.
- Especificar a ordem em que as instruções (ações) são executadas em um programa é chamado **controle de programa**.

4.3 Pseudocódigo

- **Pseudocódigo** é uma linguagem informal que ajuda você a desenvolver algoritmos sem se preocupar com os detalhes estritos da sintaxe da linguagem Java.
- Particularmente útil para desenvolver algoritmos que serão convertidos em partes estruturadas de programas Java.
- Similar ao inglês cotidiano.
- Ajuda a "estudar" um programa antes de tentar escrevê-lo em uma linguagem de programação como Java.
- Você pode digitar o pseudocódigo convenientemente, utilizando um programa editor de textos qualquer.
- O pseudocódigo cuidadosamente preparado pode ser facilmente convertido em um programa Java correspondente.
- Normalmente, o pseudocódigo só descreve as instruções que representam as ações que ocorrem depois que você converte um programa no pseudocódigo em Java e depois de o programa ser executado em um computador.
  e.g., entrada, saída ou cálculos.

4.4 Estruturas de controle

- **Execução sequencial:** As instruções em um programa são executadas uma após a outra na ordem em que são escritas.
- **Transferência do controle:** Várias instruções Java permitem especificar que a próxima instrução a executar não seja necessariamente a próxima na sequência.
- Bohm e Jacopini

  Demonstraram que todos os programas poderiam ser escritos *sem nenhuma instrução* `goto`.

  Todos os programas podem ser escritos em termos de apenas três tipos de estruturas de controle: — **estrutura de sequência**, a **estrutura de seleção** e a **estrutura de repetição**.
- Ao introduzirmos as implementações das estruturas de controle do Java, na terminologia da *Java Language Specification, nós as chamamos de "instruções de controle".*

- Estrutura da sequência

  Nativa do Java.

  A menos que instruído de outro modo, o computador executa instruções Java uma após a outra na ordem em que elas são escritas.

  O **diagrama de atividades** na Figura 4.1 ilustra uma estrutura de sequência típica em que dois cálculos são realizados na ordem.

  O Java permite ter o número de ações que você quiser em uma estrutura de sequência.

  Em qualquer lugar que uma ação única pode ser colocada, podemos colocar várias ações em sequência.

Java™ COMO PROGRAMAR 8ª edição

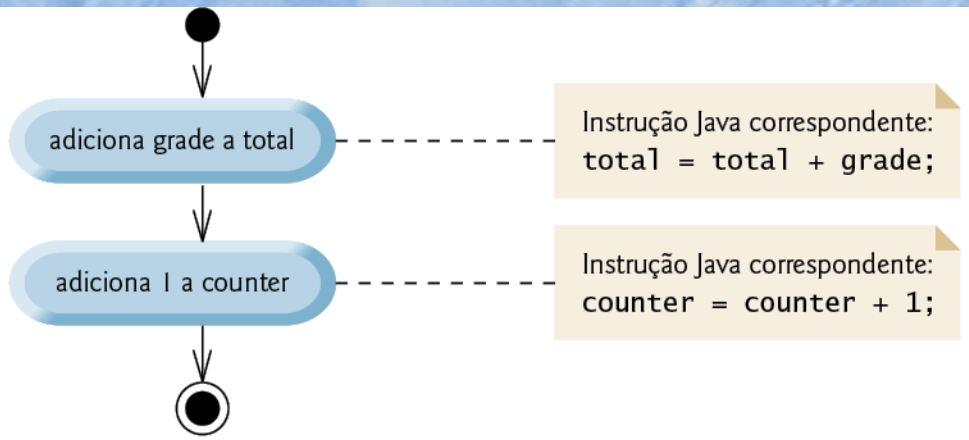

Figura 4.1 | Diagrama de atividades da estrutura de sequência.

- Diagrama de atividades UML.
- Um diagrama de atividades modela o **fluxo de trabalho** (também chamado **atividade**) de uma parte de um sistema de software.
- Podem incluir uma parte de um algoritmo, como a estrutura de sequência na Figura 4.1.
- Compostos de símbolos

  Símbolos do estado de ação (retângulos com seus lados esquerdo e direito substituídos por arcos curvados para fora)

  losangos

  círculos pequenos
- Esses símbolos são conectados por **setas de transição**, que representam o fluxo da atividade — isto é, a ordem em que as ações devem ocorrer.
- Ajudam a desenvolver e representar algoritmos.
- Mostram claramente como as estruturas de controle funcionam.

- Diagrama de atividades da estrutura de sequência na Figura 4.1.
- Ele contém dois **estados de ações** que representam ações a realizar.
- Cada um contém um **expressão de ação** que especifica uma ação particular a executar.
- Setas representam **transições** (a ordem em que ocorrem as ações representadas pelos estados de ação).
- O **círculo sólido** localizado na parte superior representa o **estado inicial** — o início do fluxo de trabalho antes de o programa realizar as ações modeladas.
- O **círculo sólido cercado por um círculo vazio** que aparece na parte inferior o **estado final** — o final do fluxo de trabalho depois que o programa realiza suas ações.

- **Notas** na UML

  Como comentários Java.

  Retângulos com o canto superior direito dobrado.

  Linha pontilhada conecta cada nota com o elemento que ela descreve.

  Os diagramas de atividades normalmente não mostram o código Java que implementa a atividade. Fazemos isso aqui para ilustrar como o diagrama se relaciona ao código Java.
- Mais informações sobre a UML

  veja nosso estudo de caso opcional (Capítulos 12–13)

  visite `www.uml.org`

- Três tipos de **instruções de seleção.**
- Instrução `if`

  Executa uma ação, se uma condição é verdadeira; pula-a, se falsa.

  Instrução de uma única seleção — seleciona ou ignora uma única ação (ou o grupo de ações).
- Instrução `if...else`

  Realiza uma ação se uma condição for verdadeira e realiza uma ação diferente se a condição for falsa.

  Instrução de seleção dupla — seleciona entre duas ações diferentes (ou grupos de ações).
- Instrução `switch`

  Executa um de várias ações, com base no valor de uma expressão.

  Instrução de seleção múltipla — seleciona entre muitas ações diferentes (ou grupos de ações).

- Três **instruções de repetição** (também chamadas **instruções de loop**) executam instruções repetidamente enquanto **condição de continuação de loop** permanecer verdadeira.
- As instruções `while` e `for` realizam a(s) ação(ões) no seu corpo zero ou mais vezes se a condição de continuação de loop for inicialmente falsa, o corpo não será executado.
- A instrução `do…while` realiza a(s) ação(ões) no seu corpo uma ou mais vezes.
- `if`, `else`, `switch`, `while`, `do` e `for` são palavras-chave.

  Apêndice C: Lista completa das palavras-chave do Java.

- Cada programa é formado combinando o número de instruções de sequência, instruções de seleção (três tipos) e instruções de repetição (três tipos) conforme apropriado para o algoritmo que o programa implementa.
- Podemos modelar cada instrução de controle como um diagrama de atividades.

  O estado inicial e o estado final representam um ponto de entrada e um ponto de saída da instrução de controle, respectivamente.

  Instruções de controle de entrada única/saída única.

  Empilhamento de instruções de controle — conectam o ponto de saída de uma instrução ao ponto de entrada da instrução seguinte.

  Aninhamento de instruções de controle — instrução de controle dentro de outra.

4.5 Instrução de seleção única `if`

- Pseudocódigo

  If (Se) a nota do aluno é maior que ou igual a 60
  Imprime "Aprovado"

- Se a condição for falsa, a instrução Print é ignorada e a próxima instrução de pseudocódigo na sequência é realizada.
- Recuo

  Opcional, mas recomendado.

  Enfatiza a estrutura inerente de programas estruturados.
- O pseudocódigo *If* precedente em Java:

```
if ( studentGrade >= 60 )
   System.out.println( "Passed" );
```

- Corresponde precisamente com o pseudocódigo.

- Figura 4.2 – Diagrama de atividade UML da instrução if.
- O losango, ou **símbolo de decisão**, indica que uma decisão será tomada.
- O fluxo de trabalho continua ao longo de um caminho determinado pelas **condições de guarda** do símbolo associado, que podem ser verdadeiras ou falsas.
- Cada seta de transição que sai de um símbolo de decisão tem uma condição de guarda (entre colchetes ao lado da seta).
- Se uma condição de guarda for verdadeira, o fluxo de trabalho entra no estado de ação para o qual a seta de transição aponta.

Java™ COMO PROGRAMAR 8ª edição

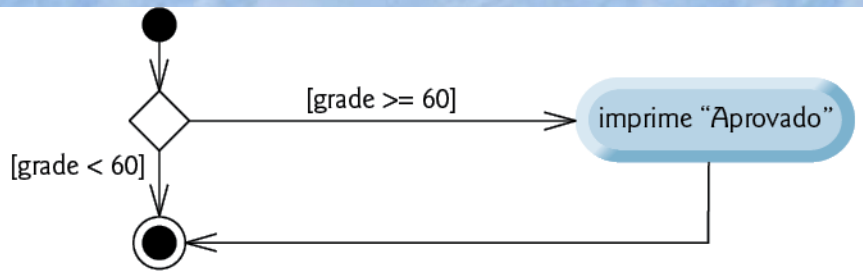

Figura 4.2 | Diagrama de atividades UML de uma instrução de seleção única `if`.

4.6 Instrução de seleção `if…else` dupla

- A **instrução de seleção dupla `if…else`** especifica uma ação a realizar quando a condição é verdadeira e uma ação diferente quando a condição é falsa.
- Pseudocódigo

  Se (if) a nota do aluno for maior que ou igual a 60
  Imprima "Aprovado"
  Caso contrário (else)
  Imprima "Reprovado"

- A instrução *If…Else precedente em pseudocódigo Java:*

```
if ( grade >= 60 )
   System.out.println( "Passed" );
else
   System.out.println( "Passed" );
```

- Observe que o corpo de `else` também é recuado.

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 4.1

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

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 4.2

Se existem vários níveis de recuo, cada nível deve ser recuado pela mesma quantidade adicional de espaço.

- A Figura 4.3 ilustra o fluxo de controle na instrução `if...else`.
- Os símbolos no diagrama de atividades UML (além do estado inicial, setas de transição e estado final) representam os estados e decisões da ação.

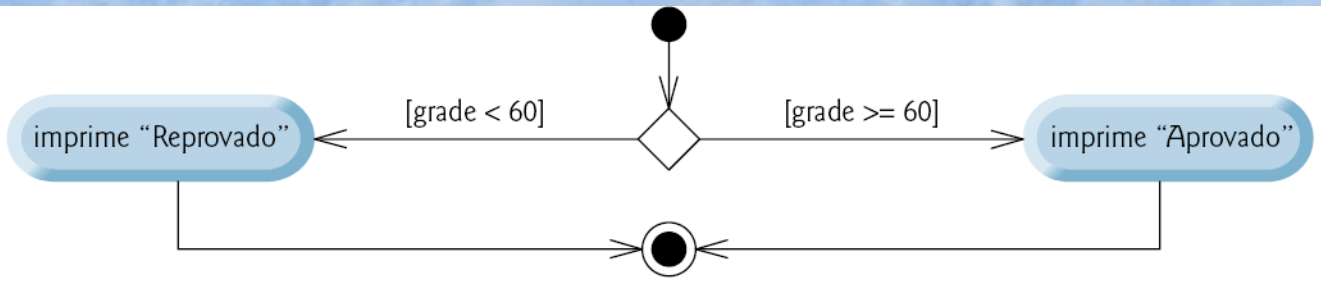

Figura 4.3 | Diagrama de atividades da UML de instrução de seleção dupla `if...else`.

- **Operador condicional** (**?**:) — abreviatura `if...else`.
- **Operador de ternário** (aceita três operandos).
- Operandos e `?:` lançar uma **expressão condicional.**
- O operando à esquerda do `?` é uma expressão **boolean** — que é avaliada como um valor `boolean` (**true** ou **false**)
- O segundo operando (entre o `?` e `:`) é o valor se a a expressão `boolean` for `true`
- O terceiro operando (à direita de `:`) é o valor se a a expressão `boolean` for `false`
- Exemplo:

```
System.out.println(
   studentGrade >= 60 ? "Passed" : "Failed" );
```

- Resulta na string `"Passed"` se a expressão `boolean studentGrade >= 60` for verdadeira e na string `"Failed"` se for falsa.

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 4.3

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

- Pode testar múltiplos casos colocando instruções `if...else` dentro de outras instruções `if...else` para criar **instruções *if...else* aninhadas.**
- Pseudocódigo:

If (Se) a nota do aluno é maior que ou igual a 90
Imprima "A"
caso contrário
If (Se) a nota do aluno é maior que ou igual a 80
Imprima "B"
caso contrário
If (Se) a nota do aluno é maior que ou igual a 70
Imprima "C"
caso contrário
If (Se) a nota do aluno é maior que ou igual a 60
Imprima "D"
caso contrário
Imprima "F"

- Esse pseudocódigo pode ser escrito em Java como

```
if ( studentGrade >= 90 )
   System.out.println( "A" );
else
      if ( studentGrade >= 80 )
   System.out.println( "B" );
   else
            if ( studentGrade >= 70 )
   System.out.println( "C" );
      else
            if ( studentGrade >= 60 )
   System.out.println( "D" );
         else
   System.out.println( "F" );
```

- If `studentGrade` >= 90, as primeiras quatro condições serão verdadeiras, mas só a instrução na parte `if` da primeira instrução `if`…`else` será executada. Depois que a parte `else` é executada, a parte `if`…`else` da instrução "mais externa" é pulada.

- A maioria dos programadores Java prefere escrever a instrução aninhada `if...else` anterior assim

```
if ( studentGrade >= 90 )
   System.out.println( "A" );
else if ( studentGrade >= 80 )
   System.out.println( "B" );
else if ( studentGrade >= 70 )
   System.out.println( "C" );
else if ( studentGrade >= 60 )
   System.out.println( "D" );
else
   System.out.println( "F" );
```

- As duas formas são idênticas, exceto quanto ao espaçamento e recuo, que o compilador ignora.

- O compilador Java sempre associa um `else` à instrução `if` imediatamente precedente, a menos que instruído de outro modo pela colocação de chaves (`{` and `}`).
- Consulte a seção sobre o **problema do `else` oscilante**.
- O seguinte código não é o que aparece:

```
if ( x > 5 )
   if ( y > 5 )
      System.out.println( "x and y are > 5" );
else
   System.out.println( "x is <= 5" );
```

- Cuidado! Essa instrução `if`…`else` aninhada não é executada como parece. Na verdade, o compilador interpreta a instrução como

```
if ( x > 5 )
   if ( y > 5 )
      System.out.println( "x and y are > 5" );
   else
   System.out.println( "x is <= 5" );
```

- Para forçar a instrução `if`…`else` aninhada a executar como foi originalmente concebida, devemos escrevê-la da seguinte maneira:

```
if ( x > 5 )
{
   if ( y > 5 )
      System.out.println( "x and y are > 5" );
}
else
   System.out.println( "x is <= 5" );
```

- As chaves indicam que o segundo `if` está no corpo do primeiro e que o `else` está associado com o *primeiro* `if`.
- Os exercícios 4.27–4.28 investigam ainda mais o problema do `else` oscilante.

- A instrução `if` normalmente espera somente uma instrução no seu corpo.
- Para incluir várias instruções no corpo de um `if` (ou no corpo de um `else` de uma instrução i`if`…`else`), inclua as instruções dentro de chaves.
- As instruções contidas em um par de chaves formam um **bloco**.
- Um bloco pode ser colocado em qualquer lugar em que uma instrução individual pode ser colocada.
- Exemplo: Um bloco na parte `else` de uma instrução `if`…`else`

```
if ( grade >= 60 )
   System.out.println( "Passed" );
else
{
   System.out.println( "Passed" );
   System.out.println("You must take this course
again.");
}
```

- Erros de sintaxe (por exemplo, quando não é colocada uma das chaves em um bloco do programa) são capturados pelo compilador.
- Um **erro de lógica** (por exemplo, quando não colocadas as duas chaves em um bloco do programa) são capturadas em tempo de execução.
- Um **erro fatal de lógica** faz com que um programa falhe e finalize prematuramente.
- Um **erro não fatal de lógica** permite que um programa continue a executar, mas faz com que produza resultados incorretos.

Java™ COMO PROGRAMAR 8ª edição

Erro comum de programação 4.1

Esquecer uma ou ambas as chaves que delimitam uma instrução composta pode levar a erros de sintaxe ou de lógica.

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 4.4

Sempre utilizar as chaves em uma instrução de controle `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, digite as chaves de abertura ou fechamento de blocos antes de digitar as instruções individuais dentro das chaves.*

- Assim como um bloco pode ser colocado em qualquer lugar em que uma instrução individual pode ser colocada, também é possível ter uma instrução vazia.
- A instrução vazia é representada colocando um ponto e vírgula (`;`) no qual normalmente estaria uma instrução.

Java™ COMO PROGRAMAR 8ª edição

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 seleção única e um erro de sintaxe em instruções* `if...else` *de seleção dupla (quando a parte* `if` *contém uma instrução de corpo real).*

4.7 Instrução de repetição while

- **Instrução de repetição** — repete uma ação enquanto uma condição permanecer verdadeira.
- Pseudocódigo

  Enquanto houver mais itens em minha lista de compras
  Comprar o próximo item e riscá-lo da minha lista.

- O corpo da declaração de repetição pode ser uma única instrução ou um bloco.
- Por fim, a condição se tornará falsa. Nesse ponto, a repetição termina e a primeira instrução depois da instrução de repetição é executada.

- Exemplo da **instrução de repetição while** do Java: encontrar a primeira potência de 3 maior que 100. Assuma que a variável `int product` é inicializado como `3`.

```
while ( product <= 100 )
   product = 3 * product;
```

- Cada iteração multiplica `product` por 3, portanto `product` assume os valores 9, 27, 81 e 243 successivamente.
- Quando a variável `product` torna-se 243, a condição da instrução `while` — `product <= 100` — torna-se falsa.
- A repetição termina. O valor final de `product` é 243.
- A execução de programa continua com a próxima instrução depois da instrução `while`.

Java™ COMO PROGRAMAR 8ª edição

Erro comum de programação 4.3

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

- O diagrama de atividades da UML na Figura 4.4 ilustra o fluxo de controle na instrução `while` anterior.
- A UML representa o **símbolo de agregação** e o símbolo de decisão como losangos.
- O símbolo de agregação une dois fluxos de atividade a um único.
- Os símbolos de decisão e agregação podem ser separados pelo número de setas de transição "entrantes" e "saintes".

Um símbolo de decisão contém uma seta de transição apontando para o losango e duas ou mais apontando a partir dele para indicar possíveis transições a partir desse ponto. Cada seta de transição apontando de um símbolo de decisão contém uma condição de guarda ao lado dela.

Um símbolo de agregação tem duas ou mais setas de transição apontando para o losango e somente uma seta saindo do losango, para indicar que diversos fluxos de atividades se juntam a fim de dar continuidade à atividade. Nenhuma das setas de transição associadas com um símbolo de agregação contém uma condição de guarda.

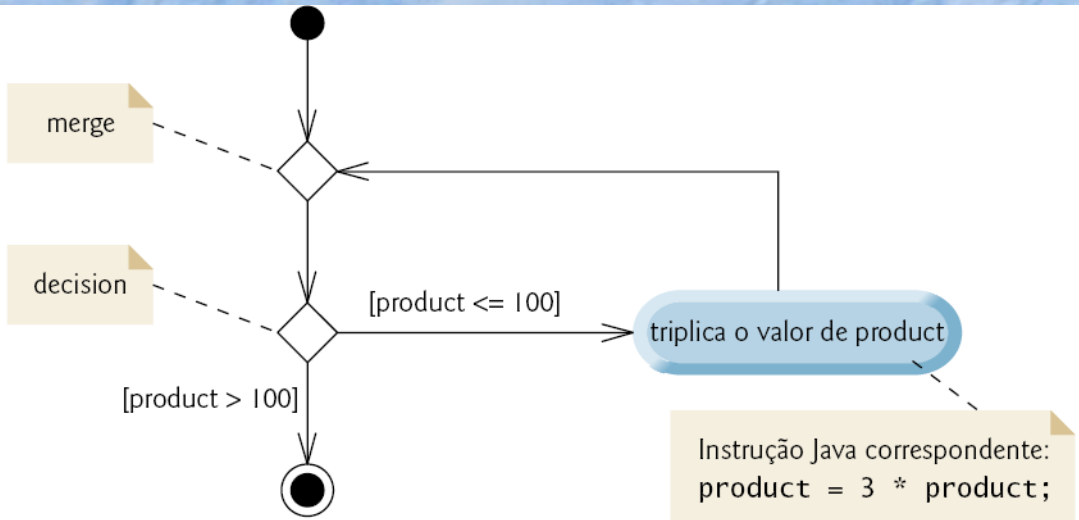

Figura 4.4 | Diagrama de atividades UML da instrução de repetição `while`.

4.8 Formulando algoritmos: Repetição controlada por contador

- *Uma classe de dez alunos se submeteu a um questionário. As notas (inteiros no intervalo 0 a 100) para esse questionário estão disponíveis. Determine a média da classe no questionário.*
- A média de classe é igual à soma das notas divididas pelo número de alunos.
- O algoritmo para resolver esse problema em um computador deve inserir cada nota, armazenar o total de todas as notas inseridas, realizar o cálculo da média e imprimir o resultado.
- Utilizamos **repetição controlada** por contador para inserir as notas uma por vez.
- Uma variável chamada **contador** (ou **variável de controle**) para controlar o número de vezes que um conjunto de instruções será executado.
- Repetição controlada por contador é frequentemente chamada **repetição definida** uma vez que o número de repetições é conhecido antes de o loop começar a executar.

Java™ COMO PROGRAMAR 8ª edição

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, produzir um programa Java que execute o algoritmo é normalmente simples.

- Um **total** é uma variável utilizada para acumular a soma de vários valores.
- Um contador é uma variável utilizada para contar.
- Variáveis utilizadas para armazenar totais normalmente são inicializadas como zero antes de ser utilizadas em um programa.

```
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 | O algoritmo em pseudocódigo com a repetição controlada por contador para resolver o problema da média da classe.


```
1  // Figura 4.6: GradeBook.java
2  // Classe GradeBook que resolve o problema da média da classe
3  // utilizando 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 esse GradeBook representa
9
10    // o 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
```

Figura 4.6 | Repetição controlada por contador: problema da média da classe. (Parte 1 de 3.)


```
22      // método para recuperar o nome do curso
23      public String getCourseName()
24      {
25         return courseName;
26      } // fim do método getCourseName
27
28      // exibe uma mensagem de boas-vindas ao usuário GradeBook
29      public void displayMessage()
30      {
31         // getCourseName obtém o nome do curso
32         System.out.printf( "Welcome to the GradeBook 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
37      public void determineClassAverage()
38      {
39         // cria Scanner para obter entrada 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
```

Declara o método `determineClassAverage`

A variável de controle `gradeCounter` é a variável de controle do loop

Figura 4.6 | Repetição controlada por contador: problema da média da classe. (Parte 2 de 3.)

```
46
47          // fase de inicialização
48          total = 0; // inicializa o total
49          gradeCounter = 1; // inicializa o contador de loops
50
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 ints produz um int
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    } // fim da classe GradeBook
```

Inicializa `gradeCounter` como 1; indica que a primeira a nota será inserida

Incrementa `gradeCounter`

Calcula a média com aritmética de inteiros

Figura 4.6 | Repetição controlada por contador: problema da média da classe. (Parte 3 de 3.)

- Variáveis declaradas no corpo de um método são variáveis locais e podem ser utilizadas apenas da linha de sua declaração até a chave direita de fechamento da declaração de método.
- A declaração de uma variável local deve aparecer antes da variável ser utilizada nesse método.
- Uma variável local não pode ser acessada fora do método em que é declarada.

Java™ COMO PROGRAMAR 8ª edição

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.

Java™ COMO PROGRAMAR 8ª edição

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 normalmente são inicializados como 0 ou 1, dependendo de como eles são utilizados (mostraremos exemplos de quando utilizar 0 e quando utilizar 1).


```
1  // Figura 4.7: GradeBookTest.java
2  // Cria o objeto da classe GradeBook e invoca seu método
3  // determineClassAverage
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 } // fim da classe GradeBookTest
```

Figura 4.7 | A classe `GradeBookTest` cria um objeto da classe `GradeBook` (Figura 4.6) e invoca seu método `determineClassAverage`. (Parte 1 de 2.)


```
Welcome to the GradeBook 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
```

Figura 4.7 | A classe `GradeBookTest` cria um objeto da classe `GradeBook` (Figura 4.6) e invoca seu método `determineClassAverage`. (Parte 2 de 2.)

- A saída do programa indica que a soma dos valores das notas na execução de exemplo é 846, que, quando dividido por 10, deve produzir o número de ponto flutuante 84,6.
- O resultado do cálculo `total / 10` (linha 61 da Figura 4.6) é o inteiro 84, porque `total` e `10` são ambos inteiros.
- Dividir dois inteiros resulta em **divisão de inteiros** — qualquer parte fracionária do cálculo é perdida (isto é, **truncada**).

Java™ COMO PROGRAMAR 8ª edição

Erro comum de programação 4.5

Assumir que a 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

- *Desenvolva um programa para tirar a média da classe que processe as notas de acordo com um número arbitrário de alunos toda vez que é executado.*
- A **repetição controlada por sentinela** é frequentemente chamada **repetição indefinida**, uma vez que o número de repetições não é conhecido antes de o loop iniciar a execução.
- Um valor especial chamado **valor de sentinela** (também chamado **valor de sinal**, **valor fictício** ou **valor de flag**) é utilizado para indicar o "fim da entrada de dados".
- Deve-se escolher um valor de sentinela que não possa ser confundido com um valor aceitável de entrada.

Java™ COMO PROGRAMAR 8ª edição

Erro comum de programação 4.6

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

- **Refinamento passo a passo de cima para baixo**
- Iniciamos com uma representação em pseudocódigo do topo — uma única instrução que fornece a função geral do programa:

  Determine a média de classe para o questionário
- O topo é uma *representação completa de um programa.* Raramente o topo fornece detalhes suficientes para escrever um programa em Java.
- Divida o topo em uma série de tarefas menores e liste-as na ordem em que elas serão realizadas.
- Primeiro refinamento:

  Inicializar variáveis
  Inserir, somar e contar as notas do teste
  Calcule e imprima a média da classe
- Esse refinamento utiliza somente a estrutura de sequência — os passos listados devem ser executados na ordem, um depois do outro.

Java™ COMO PROGRAMAR 8ª edição

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.

Java™ COMO PROGRAMAR 8ª edição

Observação de engenharia de software 4.3

Muitos programas podem ser divididos logicamente em três fases: uma fase de inicialização que inicializa as variáveis do programa; uma fase de processamento que insere os valores dos dados e ajusta as variáveis do programa de maneira correspondente; e uma fase de término que calcula e insere os resultados finais.

- **Segundo refinamento:** acrescente variáveis específicas.
- A instrução de pseudocódigo

  Inicializar variáveis

- pode ser refinada desta maneira:

  Inicialize total como zero
  Inicialize counter como zero

- A instrução de pseudocódigo:

  Inserir, somar e contar as notas do teste

- requer uma instrução de repetição que insere sucessivamente cada nota.
- Não conhecemos antecipadamente quantas notas devem ser processadas, assim utilizaremos a repetição controlada por sentinela.
- O segundo refinamento da instrução em pseudocódigo precedente é então:

  Solicita que o usuário insira a primeira nota
  Insira a primeira nota (possivelmente o sentinela)

  Enquanto o usuário não inserir o sentinela
  Adicione essa nota à soma total
  Adicione um ao contador de notas
  Solicite para o usuário inserir a próxima nota
  Insira a próxima nota (possivelmente a sentinela)

- A instrução de pseudocódigo:

  Calcule e imprima a média da classe

  pode ser refinada desta maneira:

  Se o contador não for igual a zero
  Configure a média como o total dividido pelo contador
  Imprima a média
  caso contrário
  Imprima "Nenhuma nota foi inserida"

- Teste a possibilidade de divisão por zero — um erro de lógica que, se passar não detectado, resultaria em falha do programa ou produziria saída inválida.

Java™ COMO PROGRAMAR 8ª edição

Dica de prevenção de erro 4.2

Ao realizar a divisão por uma expressão cujo valor pode ser zero, teste-a e trate-a (p. ex., imprimir uma mensagem de erro) em vez de permitir a ocorrência do erro.


```
1   Initialize total to zero
2   Initialize counter to zero
3
4   Prompt the user to enter the first grade
5   Input the first grade (possibly the sentinel)
6
7   While the user has not yet entered the sentinel
8       Add this grade into the running total
9       Add one to the grade counter
10      Prompt the user to enter the next grade
11      Input the next grade (possibly the sentinel)
12
13  If the counter is not equal to zero
14      Set the average to the total divided by the counter
15      Print the average
16  else
17      Print "No grades were entered"
```

Fig. 4.8 | Class-average problem pseudocode algorithm with sentinel-controlled repetition.

Java™ COMO PROGRAMAR 8ª edição

Observação de engenharia de software 4.4

Termine o processo de refinamento passo a passo de cima para baixo quando tiver especificado o algoritmo de pseudocódigo em detalhes suficientes para você converter o pseudocódigo em Java. Normalmente, implementar o programa Java é então simples e direto.

Java™ COMO PROGRAMAR 8ª edição

Observação de engenharia de software 4.5

Alguns programadores não utilizam 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 isso talvez funcione para problemas simples e conhecidos, pode levar a erros sérios e atrasos em projetos grandes e complexos.

```
1  // Figura 4.9: GradeBook.java
2  // Classe GradeBook que resolve o programa da média da classe
3  // utilizando repetição controlada 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 esse GradeBook representa
9
10    // o 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
```

Figura 4.9 | Repetição controlada por sentinela: o problema da média da classe. (Parte 1 de 4.)

```
22      // método para recuperar o nome do curso
23      public String getCourseName()
24      {
25         return courseName;
26      } // fim do método getCourseName
27
28      // exibe uma mensagem de boas-vindas para o usuário GradeBook
29      public void displayMessage()
30      {
31         // getCourseName obtém o nome do curso
32         System.out.printf( "Welcome to the GradeBook 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 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 média
```

Declara o método determineClassAverage

Irá calcular e armazenar a média como um número de ponto flutuante

Figura 4.9 | Repetição controlada por sentinela: o problema da média da classe. (Parte 2 de 4.)

```
46
47          // fase de inicialização
48          total = 0; // inicializa o total
49          gradeCounter = 0; // inicializa o contador de loops
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
56          // faz um loop até ler o valor de sentinela inserido pelo usuário
57          while ( grade != -1 )
58          {
59             total = total + grade; // adiciona grade a total
60             gradeCounter = gradeCounter + 1; // incrementa 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
```

Inicializa `gradeCounter` como 0; nenhuma nota foi inserida ainda e a primeira nota pode ser o valor de sentinela

Obtém a primeira nota antes do loop na repetição controlada por sentinela

Obtém as notas seguintes ao final do loop na repetição controlada por sentinela

Figura 4.9 | Repetição controlada por sentinela: o problema da média da classe. (Parte 3 de 4.)


```
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 2 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, então gera mensagem apropriada
80             System.out.println( "No grades were entered" );
81       } // fim do método determineClassAverage
82    } // fim da classe GradeBook
```

Verifica a possibilidade de divisão por zero

Utiliza operador de coerção para forçar o cálculo da média com ponto flutuante

Figura 4.9 | Repetição controlada por sentinela: o problema da média da classe. (Parte 4 de 4.)

- Lógica do programa de repetição controlada por sentinela

  Lê o primeiro valor antes de alcançar o `while`.

  Esse valor determina se o fluxo do programa de controle deve entrar no corpo do `while`. Se a condição do `while` for falsa, o usuário inseriu o valor de sentinela, portanto o corpo do `while` não é executado (isto é, nenhuma nota foi inserida).

  Se a condição for verdadeira, o corpo começa execução e processa a entrada.

  Então, o corpo de loop insere o próximo valor fornecido pelo usuário antes do fim do loop.

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 4.5

Em um loop controlado por sentinela, os prompts solicitando dados devem lembrar o usuário sobre o valor de sentinela.

Java™ COMO PROGRAMAR 8ª edição

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 entre chaves mesmo que seja uma única instrução.

- A divisão de inteiros produz um resultado inteiro.
- Para realizar um cálculo de ponto flutuante com inteiros, devemos temporariamente tratar esses valores como números de ponto flutuante para utilização no cálculo.
- O **operador unário de coerção (double)** cria uma cópia de ponto flutuante temporária de seu operando.
- Operador de coerção realiza **conversão explícita** (ou **coerção de tipo**).
- O valor armazenado no operando permanece inalterado.
- O Java avalia somente expressões aritméticas em que os tipos dos operandos são idênticos.
- **Promoção** (ou **conversão implícita**) realizada nos operandos.
- Em uma expressão que contém valores dos tipos `int` e `double`, os valores `int` são promovidos para valores `double` para uso na expressão.

Java™ COMO PROGRAMAR 8ª edição

Erro comum de programação 4.8

Um 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.*

- Os operadores de coerção estão disponíveis para qualquer tipo.
- O operador de coerção é formado colocando parênteses em torno do nome de um tipo.
- O operador é um **operador unário** (isto é, um operador que recebe somente um operando).
- O Java também suporta versões dos operadores mais(+) e menos (−).
- Os operadores de coerção são associados da direita para a esquerda e têm a mesma precedência que outros operadores unários como + unário e − unário.
- Essa precedência é um nível mais alto do que aquela dos **operadores multiplicativos** *, / e %.
- Apêndice A: Tabela de precedência de operadores.


```
1  // Figura 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 myGradeBook da 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
15    } // fim de main
16 } // fim da classe GradeBookTest
```

Figura 4.10 | A classe `GradeBookTest` cria um objeto de classe `GradeBook` (Figura 4.9) e invoca seu método `determineClassAverage`. (Parte 1 de 2.)


```
Welcome to the GradeBook 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
```

Figura 4.10 | A classe `GradeBookTest` cria um objeto de classe `GradeBook` (Figura 4.9) e invoca seu método `determineClassAverage`. (Parte 2 de 2.)

4.10 Formulando algoritmos: Instruções de controle aninhadas

- Este estudo de caso examina o **aninhamento** de uma instrução de controle dentro de outra.
- Uma faculdade oferece um curso que prepara os candidatos a obter licença estadual para corretores de imóveis. No ano passado, dez alunos que concluíram esse curso prestaram o exame. A universidade quer saber como foi o desempenho dos seus alunos nesse exame. Você foi contratado para escrever um programa que resuma os resultados. Para tanto, recebeu uma lista desses 10 alunos. Ao lado de cada nome é escrito 1 se o aluno passou no exame ou 2 se o aluno foi reprovado.

- Este estudo de caso examina o **aninhamento** de uma instrução de controle dentro de outra.
- Seu programa deve analisar os resultados do exame assim:

  Insira o resultado de cada teste (isto é, um 1 ou um 2). Exiba a mensagem "Inserir resultado" na tela toda vez que o programa solicitar o resultado de outro teste.

  Conte o número de cada tipo de resultado.

  Exiba um resumo dos resultados do teste indicando o número de alunos aprovados e reprovados.

  Se mais de oito estudantes forem aprovados no exame, imprima a mensagem *"Bonus to instructor!"*


```
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
7           Insira o próximo resultado do exame
8
9           Se o aluno foi aprovado
10                  Adicione um a aprovações
11          Caso contrário (Else)
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 "Bonus to instructor!"
```

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

Java™ COMO PROGRAMAR 8ª edição

Dica de prevenção de erro 4.3

Inicializar variáveis locais quando são declaradas ajuda a evitar quaisquer erros de compilação que poderiam surgir como resultado de se tentar utilizar variáveis não inicializadas. 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.


```
1  // Figura 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 static void main( String[] args )
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 (fornecido pelo usuário)
17
18       // processa 10 alunos com o loop controlado por contador
19       while ( studentCounter <= 10 )
20       {
21          // solicita uma entrada e obtém o valor fornecido pelo usuário
22          System.out.print( "Enter result (1 = pass, 2 = fail): " );
23          result = input.nextInt();
```

Declara e inicializa contadores para aprovações, reprovações e alunos

A instrução de repetição `while` itera 10 vezes

Figura 4.12 | Estruturas de controle aninhadas: problema dos resultados do exame. (Parte 1 de 4.)

```
24
25          // if...else aninhado em 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
38       if ( passes >  8 )
40          System.out.println( "Bonus to instructor!" );
41    } // fim de main
42 } // fim da classe Analysis
```

Incrementa `passes` ou `failures` com base na entrada do usuário

Figura 4.12 | Estruturas de controle aninhadas: problema dos resultados do exame. (Parte 2 de 4.)

```
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
Bonus to instructor!
```

Figura 4.12 | Estruturas de controle aninhadas: problema dos resultados do exame. (Parte 3 de 4.)


```
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
```

Figura 4.12 | Estruturas de controle aninhadas: problema dos resultados do exame. (Parte 4 de 4.)

4.11 Operadores de atribuição compostos

- Os **operadores de atribuição compostos** abreviam expressões de atribuição.
- Instruções como

  variável = variável operador expressão;

  onde operador é um dos operadores binários `+`, `-`, `*`, `/` ou `%` podem ser escrito na forma

  variável operador= expressão;
- Exemplo:

```
c = c + 3;
```

pode ser escrito com o **operador de atribuição composto de adição**, **`+=`**, como

```
c += 3;
```

- O operador `+=` adiciona o valor da expressão na sua direita ao valor da variável na sua esquerda e armazena o resultado na variável no lado esquerdo do operador.

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`	10 a c
`-=`	`d -= 4`	`d = d - 4`	1 a d
`*=`	`e *= 5`	`e = e * 5`	20 a e
`/=`	`f /= 3`	`f = f / 3`	2 a f
`%=`	`g %= 9`	`g = g % 9`	3 a g

Figura 4.13 | Operadores aritméticos de atribuição composta.

4.12 Operadores de incremento e decremento

- O operador de **incremento unário**, (**++**), adiciona um ao seu operando.
- O operador de **decremento unário**, (**--**), subtrai um do seu operando.
- Um operador de incremento ou de decremento que é colocado antes de uma variável é chamado de **operador de pré-incremento** ou **operador de pré-decremento**, respectivamente.
- Um operador de incremento ou de decremento que é colocado depois de uma variável é chamado de **operador de pós-incremento** ou **operador de pós-decremento**, respectivamente.

Operador	Nome do operador	Expressão de exemplo	Explicação
++	pré-incremento	++a	Incrementa a por 1, então utiliza o novo valor de a na expressão em que a reside.
++	pós-decremento	a++	Utiliza o valor atual de a na expressão em que a reside, então incrementa a por 1.
--	pré-incremento	--b	Decrementa b por 1, então utiliza o novo valor de b na expressão em que b reside.
--	pós-decremento	b--	Utiliza o valor atual de b na expressão em que b reside, então decrementa b por 1.

Figura 4.14 | Operadores de incremento e decremento.

- Utilizar o operador de pré-incremento (ou de pré-decremento) pré-fixado para adicionar (ou subtrair) 1 de uma variável é conhecido como **pré-incrementar** (ou **pré-decrementar**) a variável.
- Pré-incrementar (ou pré-decrementar) uma variável faz com que a variável seja incrementada (decrementada) por 1; portanto, o novo valor é utilizado na expressão em que ele aparece.
- Utilizar o operador de pré-incremento (ou de pré-decremento) pós-fixado para adicionar (ou subtrair) 1 de uma variável é conhecido como **pré-incrementar** (ou **pré-decrementar**) a variável.
- Isso faz com que o valor atual da variável seja utilizado na expressão em que ele aparece, então o valor da variável é incrementado (decrementado) por 1.

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 4.6

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


```
1  // Figura 4.15: Increment.java
2  // operadores de pré-incremento e pós-incremento.
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; // atribui 5 à variável c
12       System.out.println( c );   // imprime 5
13       System.out.println( c++ ); // imprime 5 e pós-incrementa
14       System.out.println( c );   // imprime 6
15
16       System.out.println(); // pula uma linha
17
18       // demonstra o operador de pré-incremento
19       c = 5; // atribui 5 à variável c
20       System.out.println( c );   // imprime 5
21       System.out.println( ++c ); // pré-incrementa e imprime 6
22       System.out.println( c );   // imprime 6
23    } // fim de main
24 } // fim da classe Increment
```

Lê o valor atual e, então, incrementa c

Incrementa c e, então, utiliza o valor atual

Figura 4.15 | Pré-incrementando e pós-incrementando. (Parte 1 de 2.)


```
5
5
6

5
6
6
```

Figura 4.15 | Pré-incrementando e pós-incrementando. (Parte 2 de 2.)

Java™ COMO PROGRAMAR 8ª edição

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 porque* `(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 4.16 | Precedência e associatividade dos operadores discutidos até agora.

4.13 Tipos primitivos

- O Apêndice D lista os oito tipos primitivos em Java.
- O Java requer que todas as variáveis tenham um tipo.
- O Java é uma **linguagem fortemente tipada**.
- Tipos primitivos em Java são portáveis entre todas as plataformas.
- Variáveis de instância dos tipos `char`, `byte`, `short`, `int`, `long`, `float` e `double` recebem o valor de `0`, por padrão. Atribui-se às variáveis de instância do tipo `boolean` o valor `false` por padrão.
- Variáveis de instância de tipos por referência são inicializadas por padrão para o valor `null`.

Java™ COMO PROGRAMAR 8ª edição

Dica de portabilidade 4.1

Os tipos primitivos em Java são portáveis entre todas as plataformas de computador que suportam Java.

4.14 (Opcional) Estudo de caso de GUI e imagens gráficas: criando desenhos simples

- O **sistema de coordenadas** do Java é um esquema para identificar cada ponto na tela.
- O canto superior esquerdo de um componente GUI tem as coordenadas (0, 0).
- Um par de coordenadas é composto de uma **coordenada x** (a **coordenada horizontal**) e uma **coordenada y** (a **coordenada vertical**).
- A coordenada *x* é a localização horizontal (esquerda para a direita).
- A coordenada *y* é a localização vertical que se estende de cima para baixo.
- O **eixo x** descreve cada coordenada horizontal; e o **eixo y**, cada coordenada vertical.
- Unidades coordenadas são medidas em **pixels**. (O termo pixel significa "picture element" [elemento de imagem]). Um pixel é a menor unidade de exibição de resolução do monitor.

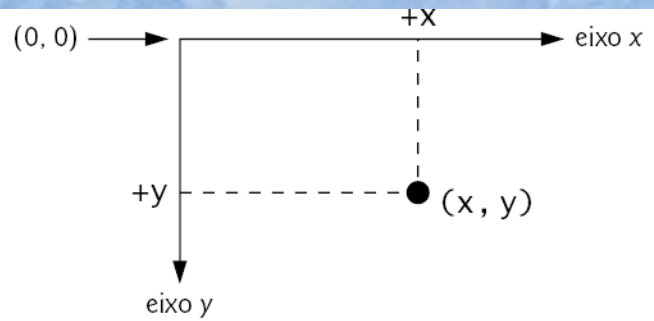

Figura 4.17 | Sistema de coordenadas Java. As unidades são medidas em pixels.

- A classe **Graphics** (do pacote `java.awt`) fornece vários métodos para desenhar texto e formas na tela.
- A classe **JPanel** (from package `javax.swing`) que fornece uma área em que podemos desenhar.


```
1  // Figura 4.18: DrawPanel.java
2  // Utilizando DrawLine para conectar os cantos de um painel.
3  import java.awt.Graphics;
4  import javax.swing.JPanel;
5
6  public class DrawPanel extends JPanel
7  {
8     // desenha um X a partir dos cantos do painel
9     public void paintComponent( Graphics g )
10    {
11       // chama paintComponent para assegurar que o painel seja exibido corretamente
12       super.paintComponent( g );
13
14       int width = getWidth(); // largura total
15       int height = getHeight(); // altura total
16
17       // desenha uma linha a partir do canto superior esquerdo até o inferior direito
18       g.drawLine( 0, 0, width, height );
19
20       // desenha uma linha a partir do canto inferior esquerdo até o superior direito
21       g.drawLine( 0, height, width, 0 );
22    } // fim do método paintComponent
23 } // fim da classe DrawPanel
```

Importa as classes `Graphics` e `JPanel` para usar nesse arquivo de código-fonte.

`DrawPanel` herda as capacidades existentes da classe `JPanel`

`paintComponent` deve ser exibido como mostrado aqui

Isso deve ser a primeira instrução no método `paintComponent`

Determina a largura e a altura do `DrawPanel` com métodos herdados

Desenha uma linha a partir do canto superior esquerdo até o inferior direito do `DrawPanel`

Desenha uma linha a partir do canto inferior esquerdo até o superior direito do `DrawPanel`

Figura 4.18 | Utilizando `drawLine` para conectar os cantos de um painel.


```
1  // Figura 4.19: DrawPanelTest.java
2  // Aplicativo para exibir uma DrawPanel.
3  import javax.swing.JFrame;
4
5  public class DrawPanelTest
6  {
7     public static void main( String[] args )
8     {
9        // cria um painel que contém nosso desenho
10       DrawPanel panel = new DrawPanel();
11
12       // cria um novo quadro para armazenar o painel
13       JFrame application = new JFrame();
14
15       // configura o frame para ser encerrado quando ele é fechado
16       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
17
18       application.add( panel ); // adiciona o painel ao frame
19       application.setSize( 250, 250 ); // configura o tamanho do frame
20       application.setVisible( true ); // torna o frame visível
21    } // fim de main
22 } // fim da classe DrawPanelTest
```

Importa a classe `JFrame` para usar neste código-fonte

Cria um `JFrame` em que o `DrawPanel` será exibido

Termina o aplicativo quando a janela é fechada

Anexa o `DrawPanel` ao `JFrame`

Configura o tamanho do `JFrame`

Exibe o `JFrame` na tela

Figura 4.19 | Criando `JFrame` para exibir `DrawPanel`. (Parte 1 de 2.)

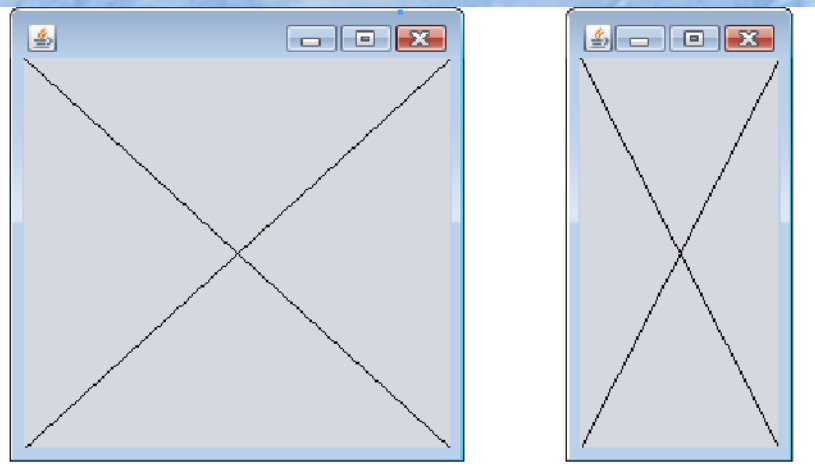

Figura 4.19 | Criando `JFrame` para exibir `DrawPanel`.

- A palavra-chave **extends** cria um relacionamento chamado herança.
- A classe que `DrawPanel` **estende**, ou da qual **herda**, `JPanel`, aparece à direita da palavra-chave `extends`.
- Nessa relação de herança, `JPanel` é chamado de **superclasse** e `DrawPanel` é chamado de **subclasse**.

- `JPanel` tem um método **paintComponent**, que o sistema chama sempre que precisa para exibir `JPanel`.
- A primeira instrução em cada método `paintComponent` que você cria sempre deve ser

```
super.paintComponent( g );
```

- Os métodos **getWidth** and **getHeight** de JPanel retornam a largura e a altura do `JPanel`, respectivamente.
- `O método` **drawLine** de Graphics desenha uma linha entre dois pontos representados pelos seus quatro argumentos. Os dois primeiros argumentos são as coordenadas *x* e *y* para uma extremidade, e os dois últimos argumentos são as coordenadas para a outra extremidade.

- Para exibir o `DrawPanel` na tela, devemos colocá-lo em uma janela.
- Crie uma janela com um objeto da classe **JFrame**.
- `JFrame setDefaultCloseOperation com o argumento JFrame.EXIT_ON_CLOSE indica que o aplicativo deve terminar quando o usuário fecha a janela.`
- O método **add** de JFrame anexa o DrawPanel (ou qualquer outro componente GUI) a um JFrame.
- O método **setSize** de JFrame recebe dois parâmetros que representam a largura e a altura do JFrame, respectivamente.
- O método **setVisible** de Jframe com o argumento `true` exibe o `JFrame`.
- Quando o `JFrame` é exibido, o método `paintComponent` de `DrawPanel` é implicitamente chamado