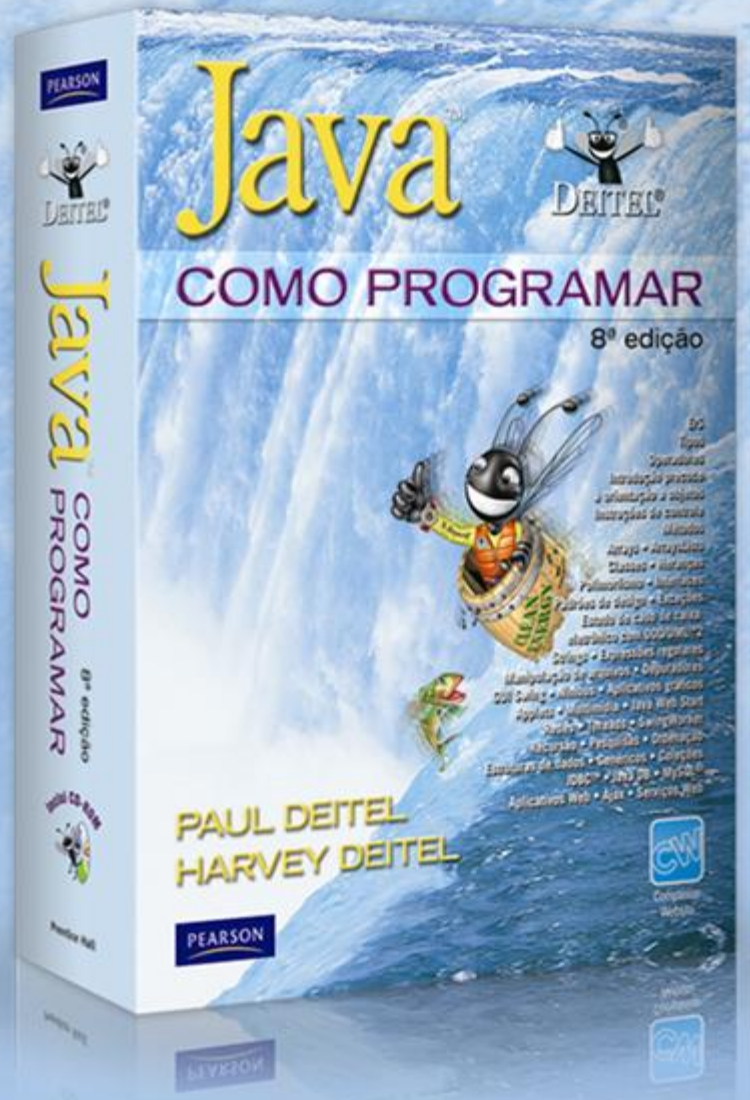
Capítulo 2
Introdução aos aplicativos Java



Java™ Como Programar, 8/E

OBJETIVOS

Neste capítulo, você aprenderá:

- A escrever aplicativos Java simples.
- A utilizar instruções de entrada e saída.
- Os tipos primitivos do Java.
- Os conceitos básicos de memória.
- A utilizar operadores aritméticos.
- A precedência dos operadores aritméticos.
- A escrever instruções de tomada de decisão.
- A utilizar operadores relacionais de igualdade.

2.1 Introdução

- Programação de aplicativo Java.
- Utilize as ferramentas do JDK para compilar e executar programas.
- Vídeos em www.deitel.com/books/jhtp8/

  Ajuda a aprender a usar os ambientes de desenvolvimento integrado Eclipse e NetBeans.

2.2 Nosso primeiro programa Java: imprimindo uma linha de texto

- **Aplicativo** Java.

  Um programa de computador que é executado quando você utiliza o **comando java** para carregar a Java Virtual Machine (JVM).
- O programa de **exemplo** na Figura 2.1 exibe uma linha de texto.


```
1  // Figura 2.1: Welcome1.java
2  // Programa de impressão de texto.
3
4  public class Welcome1
5  {
6     // método principal inicia a execução do aplicativo Java
7     public static void main( String[] args )
8     {
9        System.out.println( "Welcome to Java Programming!" );
10    } // fim do método main
11 } // fim da classe Welcome1
```

```
Welcome to Java Programming!
```

Figura 2.1 | Programa de impressão de texto.

- Comentários

```
// Figura 2.1: Welcome1.java
```

// indicam que a linha é um **comentário**.

Utilizados para **documentar programas** e aprimorar sua legibilidade

O compilador ignora comentários.

Um comentário que começa com // é um **comentário de fim de linha** — ele termina no fim da linha em que aparece.

- Um **comentário tradicional** pode se distribuir por várias linhas como em

```
/* Isso é um comentário tradicional. Ele
   pode ser dividido em várias linhas */
```

Esse tipo de comentário começa com /* e termina com */.

Todo o texto entre os delimitadores é ignorado pelo compilador.

- Comentários do **Javadoc**

  delimitados por /** e */.

  Todo o texto entre os delimitadores de comentários do Javadoc é ignorado pelo compilador.

  Permitem incorporar a documentação do programa diretamente nos programas.

  O **programa utilitário** javadoc (Apêndice M) lê comentários no estilo Javadoc e utiliza-os para preparar a documentação do seu programa no formato HTML.

Erro comum de programação 2.1

Esquecer um dos delimitadores de um comentário tradicional no estilo Javadoc causa um erro de sintaxe. A ***sintaxe*** *de uma linguagem de programação especifica as regras para criar programas apropriados nessa linguagem, assim como as regras de gramática de uma língua natural especificam a estrutura da frase. Um* ***erro de sintaxe*** *ocorre quando o compilador encontra o código que viola as regras da linguagem do Java (isto é, sua sintaxe). Nesse caso, o compilador emite uma mensagem de erro e impede o programa de compilar. Erros de sintaxe também são chamados* ***erros de compilador***, ***erros em tempo de compilação*** *ou* ***erros de compilação***, *porque o compilador detecta-os durante a fase de compilação.*

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 2.1

Algumas organizações exigem que todo programa comece com um comentário que informa o objetivo e o autor do programa, a data e a hora em que o programa foi modificado pela última vez.

- Linhas em branco e caracteres de espaço em branco facilitam a leitura dos programas.

  Juntos, linhas em branco, espaços e tabulações são conhecidos como **espaço em branco**.

  Espaços em branco são ignorados pelo compilador.

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 2.2

Utilize linhas e espaços em branco para aprimorar a legibilidade do programa.

- Declaração de classe

  public class Welcome1

  Todo programa Java consiste em pelo menos uma classe que você define.

  A **palavra-chave class** introduz uma declaração de classe e é imediatamente seguida pelo **nome de classe**.

  Palavras-chave são reservadas para uso pelo Java e sempre são escritas em letras minúsculas.

- Nomes de classe

  Por convenção, iniciam com uma letra maiúscula e apresentam a letra inicial de cada palavra que eles incluem em maiúscula (por exemplo, SampleClassName).

  O nome de uma classe Java é um **identificador** — uma série de caracteres que consiste em letras, dígitos, sublinhados (_) e sinais de cifrão ($) que não iniciem com um dígito e não contenham espaços.

  O Java faz **distinção entre maiúsculas e minúsculas** — letras maiúsculas e letras minúsculas são diferentes — portanto, a1 e A1 são identificadores diferentes (mas ambos válidos).

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 2.3

Por convenção, inicie o identificador de cada nome de classe com uma letra maiúscula e inicie cada palavra subsequente do identificador com uma letra maiúscula.

Java™ COMO PROGRAMAR 8ª edição

Erro comum de programação 2.2

O Java diferencia letras maiúsculas de minúsculas. O uso incorreto de letras maiúsculas e minúsculas para um identificador normalmente causa um erro de compilação.

Java™ COMO PROGRAMAR 8ª edição

Erro comum de programação 2.3

Um classe `public` *deve ser colocada em um arquivo que tenha o mesmo nome da classe (tanto na ortografia como no uso de maiúsculas e minúsculas) mais a extensão* `.java`*; caso contrário, ocorre um erro de compilação. Por exemplo, a classe* `public Welcome` *deve ser colocada em um arquivo chamado* `Welcome.java`.

- Chaves

  Uma **chave esquerda**, **{**, inicia o **corpo** de toda definição de classe.

  Uma **chave direita** correspondente, **}**, deve terminar cada definição de classe.

  O código entre chaves deve ser recuado.

  Esse recuo é uma das convenções de espaçamento mencionadas anteriormente.

Java™ COMO PROGRAMAR 8ª edição

Dica de prevenção de erro 2.1

Quando você digitar uma chave de abertura, ou chave esquerda, {*, imediatamente digite a chave de fechamento, ou chave direita,* }*, e, então reposicione o cursor entre as chaves e dê um recuo para começar a digitação do corpo. Essa prática ajuda a evitar erros devidos à ausência das chaves. Muitos IDEs inserem os colchetes para você.*

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 2.4

Recue o corpo inteiro de cada declaração de classe por um "nível" entre a chave esquerda e a chave direita que delimitam o corpo da classe. Esse formato enfatiza a estrutura da declaração de classe e torna mais fácil sua leitura.

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 2.5

Muitos IDEs inserem recuos ou indentações em todos os lugares certos. A tecla Tab também pode ser utilizada para criar recuos, mas as paradas de tabulação variam entre editores de textos. Recomendamos a utilização de três espaços para formar um nível de recuo.

Java™ COMO PROGRAMAR 8ª edição

Erro comum de programação 2.4

É um erro de sintaxe se chaves não ocorrerem em pares correspondentes.

- Declarando o método `main`

```
public static void main( String[] args )
```

O ponto de partida de todo aplicativo Java.

Parênteses depois do identificador `main` indicam que ele é um bloco de construção do programa; esse bloco é chamado de **método**.

Declarações de classe Java normalmente contêm um ou mais métodos.

`main` deve ser declarado conforme mostrado; caso contrário, a JVM não executará o aplicativo.

Os métodos realizam tarefas e podem retornar informações quando completam suas tarefas.

A palavra-chave `void` indica que esse método não devolverá nenhuma informação.

- **Corpo da declaração de método**

  Colocada entre um par de chave de abertura e fechamento

- Instrução

```
System.out.println("Welcome to Java
  Programming!");
```

  Instrui o computador a realizar uma ação.

  Imprime a **string** de caracteres contida entre as aspas duplas.

  Uma string às vezes é chamada de **string de caracteres** ou **string literal**.

  Os caracteres de espaço em branco em strings não são ignorados pelo compilador.

  As strings não podem distribuir várias linhas de código.

- **Objeto System.out**

  Objeto de saída padrão

  Permite que aplicativos Java exibam strings na **janela de comando** a partir da qual o aplicativo Java executa.
- Método **System.out.println**

  Exibe (ou imprime) uma linha de texto na janela de comando.

  A string entre parênteses é o **argumento** para o método.

  Posiciona o cursor de saída no início da próxima linha na janela de comando.
- A maioria das instruções termina com um ponto-e-vírgula.

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 2.6

Recue o corpo inteiro de cada declaração de método um "nível" entre as chaves que definem o corpo do método. Isso faz com que a estrutura do método se destaque, tornando a declaração do método mais fácil de ler.

Java™ COMO PROGRAMAR 8ª edição

Erro comum de programação 2.5

Quando um ponto-e-vírgula é necessário para acabar uma instrução, omitir o ponto-e-vírgula é um erro de sintaxe.

Java™ COMO PROGRAMAR 8ª edição

Dica de prevenção de erro 2.2

Ao aprender a programar, às vezes é útil "quebrar" um programa funcional para você poder familiarizar-se com as mensagens de erro de sintaxe do compilador. Essas mensagens nem sempre declaram o problema exato no código. Quando encontrar essas mensagens de erro de sintaxe, você terá uma ideia do que causou o erro. [Tente remover um ponto-e-vírgula ou chave do programa da Figura 2.1 e, então, recompile o programa para ver as mensagens de erro geradas pela omissão.]

Java™ COMO PROGRAMAR 8ª edição

Dica de prevenção de erro 2.3

Quando o compilador informa um erro de sintaxe, o erro pode não estar na linha indicada pela mensagem de erro. Primeiro, verifique a linha em que o erro foi informado. Se essa linha não contiver erros de sintaxe, verifique as várias linhas anteriores.

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 2.7

Para aprimorar a legibilidade de programa, coloque depois da chave de fechamento do corpo de uma declaração de método ou de classe um comentário que indica a declaração de método ou de classe à qual a chave pertence.

- ***Compilando e executando seu primeiro Aplicativo Java.***

  Abra uma janela de prompt de comando e mude para diretório onde programa está armazenado.

  Muitos sistemas operacionais utilizam o comando `cd` para mudar de diretório.

  Para compilar o programa, digite

  ```
  javac Welcome1.java
  ```

  Se o programa não contiver nenhum erro de sintaxe, o comando anterior cria um novo arquivo chamado `.class` (conhecido como o **arquivo de classe**) contendo os bytecodes Java independentes de plataforma que representam o aplicativo.

  Quando utilizamos o comando `java` para executar o aplicativo em uma dada plataforma, esses bytecodes serão traduzidos pela JVM em instruções que são entendidas pelo sistema operacional subjacente.

Java™ COMO PROGRAMAR 8ª edição

Dica de prevenção de erro 2.4

Ao tentar compilar um programa, se receber uma mensagem como "`bad command or filename`*," "*`javac: command not found`*" or "*`'javac' is not recognized as an internal or external command, operable program or batch file`*", sua instalação do software Java não foi completada corretamente. No JDK, isso é um sinal de que a variável de ambiente* `PATH` *do sistema não foi configurada corretamente. Revise cuidadosamente as instruções de instalação na Seção "*Antes de você começar*" deste livro. Em alguns sistemas, depois de corrigir o* `PATH`*, é necessário reinicializar o computador ou abrir uma nova janela de comando para efetuar as configurações.*

Java™ COMO PROGRAMAR 8ª edição

Dica de prevenção de erro 2.5

Toda mensagem de erro de sintaxe contém o nome do arquivo e número da linha em que o erro ocorreu. Por exemplo, `Welcome1.java:6` *indica que um erro ocorreu no arquivo* `Welcome1.java` *na linha 6. O restante da mensagem de erro fornece as informações sobre o erro de sintaxe.*

Java™ COMO PROGRAMAR 8ª edição

Dica de prevenção de erro 2.6

A mensagem de erro do compilador "`class Welcome1 is public, should be declared in a file named Welcome1.java`*" indica que o nome de arquivo não corresponde exatamente ao nome da classe* `public` *no arquivo ou que você digitou o nome de classe incorretamente ao compilar a classe.*

- Para executar o programa, digite `java Welcome1`.
- Isso carrega a JVM, que carrega o arquivo `.class` para a classe `Welcome1`.
- Observe que a extensão do nome de arquivo `.class` é omitida do comando precedente; caso contrário, a JVM não executará o programa.
- A JVM chama o método `main` para executar o programa.

Java™ COMO PROGRAMAR 8ª edição

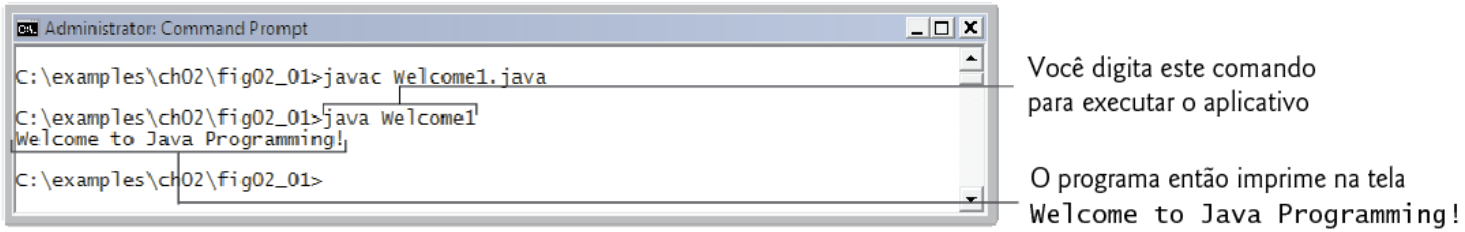


Figura 2.2 | Executando `Welcome1` a partir do **Prompt de Comando**.

Java™ COMO PROGRAMAR 8ª edição

Dica de prevenção de erro 2.7

Ao tentar executar um programa Java, se receber uma mensagem como "`Exception in thread "main" java.lang.NoClassDefFoundError: Welcome1`*", sua variável de ambiente* `CLASSPATH` *não foi configurada corretamente. Revise cuidadosamente as instruções de instalação na Seção "*Antes de você começar*" deste livro. Em alguns sistemas, talvez seja necessário reinicializar seu computador ou abrir uma nova janela de comando depois de configurar a* `CLASSPATH`*.*

2.3 Modificando nosso primeiro programa Java

- A classe `Welcome2`, mostrada na Figura 2.3, utiliza duas instruções para produzir a mesma saída mostrada na Figura 2.1.
- Novos recursos e os principais recursos em cada listagem de código são destacados em amarelo.
- O método `print` de System.out exibe uma string.
- Diferente de `println`, `print` não posiciona o cursor de saída no início da próxima linha na janela de comando.

  O próximo caractere que o programa exibe aparecerá imediatamente depois do último caractere que `print` exibe.


```
1  // Figura 2.3: Welcome2.java
2  // Imprimindo uma linha de texto com múltiplas instruções.
3
4  public class Welcome2
5  {
6     // método principal inicia a execução do aplicativo Java
7     public static void main( String[] args )
8     {
9        System.out.print( "Welcome to " );
10       System.out.println( "Java Programming!" );
11    } // fim do método main
12 } // fim da classe Welcome2
```

Imprime `welcome to` e deixa o cursor na mesma linha

Imprime `Java Programming!` iniciando onde o cursor estava posicionado e, então, um caractere de nova linha

```
Welcome to Java Programming!
```

Figura 2.3 | Imprimindo uma linha de texto com múltiplas instruções.

- **Caracteres de nova linha** indicam para os métodos `print` e `println` de `System.out` quando eles devem posicionar o cursor de saída no começo da próxima linha na janela de comando.
- Caracteres de nova linha são caracteres de espaço em branco.
- A **barra invertida (\)** é chamada **caractere de escape**.
  Indica um "caractere especial"
- A barra invertida é combinada com o próximo caractere para formar uma **sequência de escape**.
- A sequência de escape `\n` representa o caractere de nova linha.
- Lista completa de sequências de escape

  `java.sun.com/docs/books/jls/third_edition/html/lexical.html#3.10.6.`


```
1  // Figura 2.4: Welcome3.java
2  // Imprimindo múltiplas linhas de texto com uma única instrução.
3
4  public class Welcome3
5  {
6     // método principal inicia a execução do aplicativo Java
7     public static void main( String[] args )
8     {
9        System.out.println( "Welcome\n to \n Java\n Programming!" );
10    } // fim do método main
11 } // fim da classe Welcome3
```

Cada n move o cursor de saída para a próxima linha, onde a saída continua

```
Welcome
to
Java
Programming!
```

Figura 2.4 | Imprimindo múltiplas linhas de texto com uma única instrução.

Sequência de escape	Descrição
`\n`	Nova linha. Posiciona o cursor de tela no início da próxima linha.
`\t`	Tabulação horizontal. Move o cursor de tela para a próxima parada de tabulação.
`\r`	Retorno de carro. Posiciona o cursor da tela no início da linha atual — não avança para a próxima linha. Qualquer saída de caracteres depois do retorno de carro sobrescreve a saída de caracteres anteriormente gerados na linha atual.
`\\`	Barras invertidas. Utilizada para imprimir um caractere de barra invertida.

`\"`	Aspas duplas. Utilizada para imprimir um caractere de aspas duplas. Por exemplo,
	`System.out.println( "\"in quotes\"" );`
	exibe
	`"in quotes"`

Figura 2.5 | Algumas sequências de escape comuns.

2.4 Exibindo texto com printf

- Método **System.out.printf**
  `f` significa "formatado"
  exibe dados formatados
- Múltiplos argumentos de método são colocados em uma **lista separada por vírgulas.**
- O Java permite que instruções grandes sejam divididas em muitas linhas.
  Não é permitido dividir uma instrução no meio de um identificador ou no meio de uma string.
- O primeiro argumento do método `printf` é uma **string de formato**
  Pode consistir de **texto fixo** e **especificadores de formato**.
  A saída de texto fixo é gerada por `print` ou `println`.
  Cada especificador de formato é um marcador de lugar para um valor e especifica o tipo da saída de dados.
- Especificadores de formato iniciam com um sinal de porcentagem (%) e são seguidos por um caractere que representa o tipo de dados.
- O especificador de formato **%s** é um espaço reservado para uma string.


```
1  // Figura 2.6: Welcome4.java
2  // Exibindo múltiplas linhas com o método System.out.printf.
3
4  public class Welcome4
5  {
6     // método principal inicia a execução do aplicativo Java
7     public static void main( String[] args )
8     {
9        System.out.printf( "%s\n%s\n",
10          "Welcome to", "Java Programming!" );
11    } // fim do método main
12 } // fim da classe Welcome4
```

Cada %s é um espaço reservado para uma `String` que vem mais adiante no argumento

Instruções podem ser distribuídas por várias linhas

```
Welcome to
Java Programming!
```

Figura 2.6 | Exibindo múltiplas linhas com o método `System.out.printf`.

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 2.8

Coloque um espaço depois de cada vírgula (,) em uma lista de argumentos para tornar os programas mais legíveis.

Java™ COMO PROGRAMAR 8ª edição

Erro comum de programação 2.6

Dividir uma instrução no meio de um identificador ou de uma string é um erro de sintaxe.

2.5 Outra aplicação: somando inteiros

- **Inteiros**

  Números inteiros, como –22, 7, 0 e 1024.
- Os programas lembram-se dos números e outros dados na memória do computador e acessam esses dados por meio de elementos de programa chamados **variáveis**.
- O programa da Figura 2.7 demonstra esses conceitos.

```
1  // Figura 2.7: Addition.java
2  // Programa de adição que exibe a soma de dois números.
3  import java.util.Scanner;  // programa utiliza a classe Scanner
4
5  public class Addition
6  {
7     // método principal inicia a execução do aplicativo Java
8     public static void main( String[] args )
9     {
10       // cria um Scanner para obter entrada da janela de comando
11       Scanner input = new Scanner( System.in );
12
13       int number1; // primeiro número a adicionar
14       int number2; // segundo número a adicionar
15       int sum; // soma de number1 e number2
16
17       System.out.print( "Enter first integer: " ); // prompt
18       number1 = input.nextInt(); // lê primeiro o número fornecido pelo usuário
19
20       System.out.print( "Enter second integer: " ); // prompt
21       number2 = input.nextInt(); // lê o segundo número fornecido pelo usuário
22
23       sum = number1 + number2; // soma os números, depois armazena o total em sum
24
```

Importa a classe Scanner para uso neste programa

Cria Scanner para ler dados fornecidos pelo usuário

Variáveis que são declaradas mas não inicializadas

Lê um valor `int` fornecido pelo usuário

Lê outro valor `int` fornecido pelo usuário

Soma os valores `number1` e `number2`

Figura 2.7 | O programa de adição que exibe a soma de dois números. (Parte 1 de 2)


```
25          System.out.printf( "Sum is %d\n", sum ); // exibe a soma
26       } // fim do método main
27    } // fim da classe Addition
```

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Figura 2.7 | O programa de adição que exibe a soma de dois números. (Parte 2 de 2.)

- **Declaração import**

  Ajuda o compilador a localizar uma classe utilizada nesse programa.

  Rico conjunto de classes predefinidas que você pode reutilizar em vez de "reinventar a roda".

  Essas classes são agrupadas em **pacotes** — grupos nomeados de classes relacionadas — e são coletivamente referidos como **biblioteca de classes Java**, ou **Java Application Programming Interface (Java API).**

  Utilize declarações `import` para identificar as classes predefinidas utilizadas em um programa Java.

Java™ COMO PROGRAMAR 8ª edição

Erro comum de programação 2.7

Todas as declarações `import` *devem aparecer antes da primeira declaração da classe no arquivo. Colocar uma declaração* `import` *dentro do corpo de uma declaração de classe ou depois de uma declaração de classe é um erro de sintaxe.*

Java™ COMO PROGRAMAR 8ª edição

Dica de prevenção de erro 2.8

Em geral, esquecer-se de incluir uma declaração `import` *para uma classe utilizada no seu programa resulta em um erro de compilação contendo uma mensagem como "*`cannot find symbol`*". Quando isso ocorre, verifique se você forneceu as declarações* `import` *adequadas e que os nomes nas declarações* `import` *estão escritos corretamente, incluindo a utilização adequada de letras maiúsculas e minúsculas.*

- **Instrução de declaração de variável**

```
Scanner input = new Scanner( System.in );
```

Especifica o nome (`input`) e o tipo (`Scanner`) de uma variável que utilizada no programa.

- Variável

Uma posição na memória do computador na qual um valor pode ser armazenado para utilização posterior em um programa.

Devem ser declaradas com um **nome** e um **tipo** antes de poderem ser utilizadas.

O nome de uma variável permite que o programa acesse o valor da variável na memória.

O nome de uma variável pode ser qualquer identificador válido.

O tipo de uma variável especifica o tipo de informações armazenado nessa posição na memória.

- **Scanner**

  Permite que um programa leia dados para uso.

  Os dados podem ser provenientes de várias origens, como os digitados pelo usuário ou um arquivo do disco.

  Antes de utilizar um `Scanner`, você deve criá-lo e especificar a origem dos dados.
- O sinal de igual (=) em uma declaração indica que a variável deve ser **inicializada** (isto é, preparada para uso no programa) com o resultado da expressão à direita do sinal de igual.
- A palavra-chave **new** cria um objeto.
- O **objeto de entrada padrão, System.in**, permite que aplicativos leiam bytes de informações digitadas pelo usuário.
- O objeto `Scanner` traduz esses bytes em tipos (como ints) que podem ser utilizados em um programa.

- Instruções de declaração de variável

```
int number1, // primeiro número a somar
int number2, // segundo número a somar
int sum;   // soma de number1 e number2
```

  declaram as variáveis `number1`, `number2` e `sum` para armazenar dados do tipo **int**

  Podem armazenar inteiros.

  O intervalo de valores para um `int` é –2.147.483.648 a +2.147.483.647.

  Valores `int` reais podem não conter vírgulas.

- Diversas variáveis do mesmo tipo podem ser declaradas em uma declaração com os nomes de variável separados por vírgulas.

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 2.9

Declare cada variável em uma linha separada. Esse formato permite que um comentário descritivo seja facilmente inserido ao lado de cada declaração.

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 2.10

Escolher nomes de variáveis significativos ajuda um programa ser ***autodocumentado*** *(isto é, pode-se entender o programa simplesmente lendo-o em vez de ler manuais ou visualizar um número excessivo de comentários).*

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 2.11

Por convenção, identificadores de nomes de variáveis iniciam com uma letra minúscula e cada palavra no nome depois da primeira palavra inicia com uma letra maiúscula. Por exemplo, o identificador de nome da variável `firstNumber` *inicia a sua segunda palavra,* `Number`*, com uma letra* `N` *maiúscula.*

- **Prompt**

  Um prompt direciona o usuário a tomar uma ação específica.

- Portanto, `System` é uma classe.

  Parte do pacote **java.lang.**

  A classe `System` não é importada com uma declaração `import` no início do programa.

Java™ COMO PROGRAMAR 8ª edição

Observação de engenharia de software 2.1

Por padrão, o pacote `java.lang` *é importado em cada programa Java; portanto,* `java.lang` *é o único pacote na Java API que não requer uma declaração* `import`.

- Método `ScannernextInt`

```
number1 = input.nextInt(); // lê o primeiro
  número fornecido pelo usuário
```

Obtém um número inteiro do usuário no teclado.

O programa espera que o usuário digite o número e pressione a tecla *Enter* para submeter o número para o programa.

- O resultado da chamada ao método `nextInt` é colocado na variável `number1` usando o **operador de atribuição, =.**

"`number1` obtém o valor de `input.nextInt()`."

O operador = é chamado de **operador binário** — ele tem dois **operandos.**

Tudo à direita do operador de atribuição, =, sempre é avaliado antes de a atribuição ser realizada.

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 2.12

Colocar espaços em qualquer um dos lados de um operador binário faz com que eles se destaquem e torna o programa mais legível.

- Aritmética

```
sum = number1 + number2; // soma os números
```

Instrução de atribuição que calcula a soma das variáveis `number1` e `number2` e, então, atribui o resultado à variável `sum` utilizando o operador de atribuição, =.

"`sum` obtém o valor `number1` + `number2`."

Em geral, os cálculos são realizados em instruções de atribuição.

As partes das instruções que contêm cálculos são chamadas de **expressões.**

De fato, uma expressão é qualquer parte de uma instrução que tem um valor associado com ela.

- Saída formatada como inteiro

```
System.out.printf( "Sum is %d\n", sum );
```

O especificador de formato **%d** é um espaço reservador para um valor `int`

A letra **d** significa “inteiro decimal”.

2.6 Conceitos de memória

- Variáveis

  Toda variável tem um **nome**, um **type**, um **tamanho** (em bytes) e um **valor**.

  Quando o novo valor é colocado em uma variável, ele substitui o valor anterior (se houver algum).

  O valor anterior é perdido.

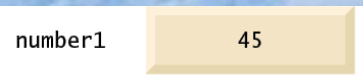


Figura 2.8 | Posição da memória mostrando o nome e valor da variável `number1`.

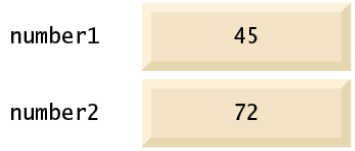


Figura 2.9 | As posições de memória depois de armazenar os valores para `number1` e `number2`.

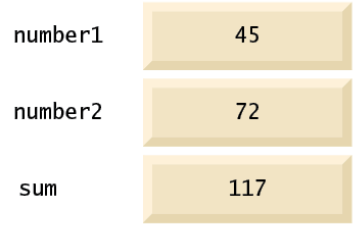


Figura 2.10 | As posições da memória depois de armazenar a soma de `number1` e `number2`.

2.7 Aritmética

- **Os operadores aritméticos** são resumidos na Figura 2.11.
- **O asterisco (*)** indica a multiplicação.
- O sinal de porcentagem (**%**) é o **operador de resto.**
- Os operadores aritméticos são operadores binários porque cada um deles opera em dois operandos.
- A **divisão de inteiros** produz um quociente inteiro.

  Qualquer parte fracionária na divisão de inteiros é simplesmente descartada (isto é, truncada) — nenhum arredondamento ocorre.
- O operador de módulo, %, produz o resto depois da divisão.

Operação Java	Operador	Expressão algébrica	Expressão Java
Adição	+	$f + 7$	`f + 7`
Subtração	–	$p - c$	`p - c`
Multiplicação	*	bm	`b * m`
Divisão	/	x/y ou $\frac{x}{y}$ ou $x \div y$	`x / y`
Resto	%	$r \bmod s$	`r % s`

Figura 2.11 | Operadores aritméticos.

- Expressões aritméticas em Java devem ser escritas na **forma de linha reta** para facilitar inserir programas no computador.
- Expressões como "a dividido por b" devem ser escritas como `a / b`, de modo que todas as constantes, variáveis e operadores apareçam em uma linha reta.
- Os parênteses são utilizados para agrupar termos em expressões da mesma maneira como em expressões algébricas.
- Se uma expressão contiver **parênteses aninhados**, o conjunto mais interno dentro dos parênteses é avaliado primeiro.

- **Regras de precedência dos operadores**

  Operações de multiplicação, divisão e módulo são aplicadas primeiro.

  Se uma expressão contiver várias dessas operações, elas serão aplicadas da esquerda para a direita.

  Os operadores de multiplicação, divisão e módulo têm o mesmo nível de precedência.

  As operações de adição e subtração são aplicadas em seguida.

  Se uma expressão contiver várias dessas operações, os operadores serão aplicados da esquerda para a direita.

  Os operadores de adição e subtração têm o mesmo nível de precedência.
- Quando dizemos que operadores são aplicados da esquerda para a direita, estamos nos referindo à sua **associatividade**.
- Alguns operadores associam da direita para a esquerda.
- Um gráfico completo de precedência está incluído no Apêndice A.

Operação Java	Operador	Expressão algébrica	Expressão Java
Adição	+	$f + 7$	`f + 7`
Subtração	–	$p - c$	`p - c`
Multiplicação	*	bm	`b * m`
Divisão	/	x/y ou $\frac{x}{y}$ ou $x \div y$	`x / y`
Resto	%	$r \bmod s$	`r % s`

Figura 2.11 | Operadores aritméticos.

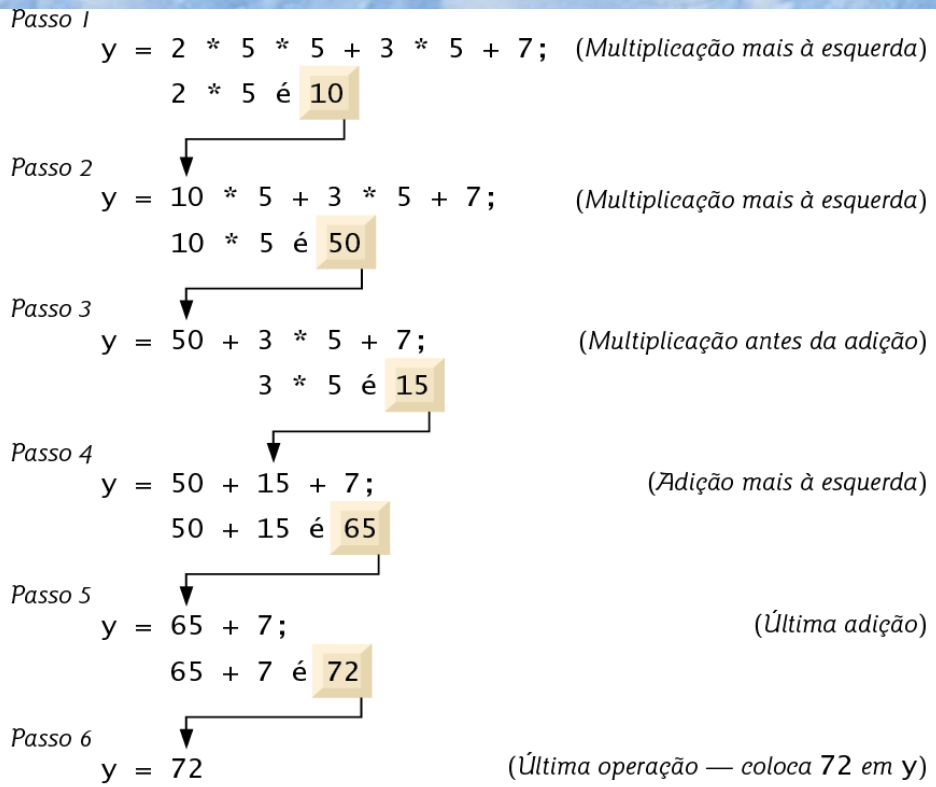


Figura 2.13 | Ordem em que um polinômio de segundo grau é avaliado.

- Como na álgebra, é aceitável colocar **parênteses redundantes** (parênteses desnecessários) em uma expressão para tornar a expressão mais clara.

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 2.13

Utilizar parênteses redundantes em expressões aritméticas complexas pode torná-las mais fáceis de ler.

2.8 Tomada de decisão: operadores de igualdade e operadores relacionais

- **Condição**

  Uma expressão que pode **verdadeira** ou **falsa.**
- **Instrução de seleção if**

  Permite que o programa tome uma **decisão** com base no valor de uma condição.
- **Operadores de igualdade** (**==** e **!=**)
- **Operadores relacionais** (**>, <, >=** e **<=**)
- Os dois operadores de igualdade têm o mesmo nível de precedência, que é mais baixo que o dos operadores relacionais.
- Os operadores de igualdade são associados da esquerda para a direita.
- Todos os operadores relacionais têm o mesmo nível de precedência e também são associados da esquerda para a direita.

Operador de igualdade ou relacional algébrico padrão	Operador de igualdade ou relacional Java	Exemplo de condição em Java	Significado da condição em Java
Operadores de igualdade			
=	==	x == y	x é igual a y
≠	!=	x != y	x é diferente de y
Operadores relacionais			
>	>	x > y	x é maior que y
<	<	x < y	x é menor que y
≥	>=	x >= y	x é maior que ou igual a y
≤	<=	x <= y	x é menor que ou igual a y

Figura 2.14 | Operadores de igualdade e operadores relacionais.


```
1  // Figura 2.15: Comparison.java
2  // Compara inteiros utilizando instruções if, operadores
3  // relacionais e operadores de igualdade.
4  import java.util.Scanner; // programa utiliza a classe Scanner
5
6  public class Comparison
7  {
8     // método principal inicia a execução do aplicativo Java
9     public static void main( String[] args )
10    {
11       // cria Scanner para obter entrada da janela de comando
12       Scanner input = new Scanner( System.in );
13
14       int number1; // primeiro número a comparar
15       int number2; // segundo número a comparar
16
17       System.out.print( "Enter first integer: " ); // prompt
18       number1 = input.nextInt(); // lê o primeiro número fornecido pelo usuário
19
20       System.out.print( "Enter second integer: " ); // prompt
21       number2 = input.nextInt(); // lê o segundo número fornecido pelo usuário
22
```

Figura 2.15 | Compare números inteiros usando instruções `if`, operadores relacionais e operadores de igualdade. (Parte 1 de 3).


```
23          if ( number1 == number2 )
24             System.out.printf( "%d == %d\n", number1, number2 );
25
26          if ( number1 != number2 )
27             System.out.printf( "%d != %d\n", number1, number2 );
28
29          if ( number1 < number2 )
30             System.out.printf( "%d < %d\n", number1, number2 );
31
32          if ( number1 > number2 )
33             System.out.printf( "%d > %d\n", number1, number2 );
34
35          if ( number1 <= number2 )
36             System.out.printf( "%d <= %d\n", number1, number2 );
37
38          if ( number1 >= number2 )
39             System.out.printf( "%d >= %d\n", number1, number2 );
40       } // fim do método main
41    } // fim da classe Comparison
```

A instrução de saída só executa se os dois números forem iguais

A instrução de saída só executa se os dois números não forem iguais

A instrução de saída só executa se **number1** for menor que **number2**

A instrução de saída só executa se **number1** for maior que **number2**

A instrução de saída só executa se **number1** for menor ou igual a **number2**

A instrução de saída só executa se **number1** for maior ou igual a **number2**

Figura 2.15 | Compare números inteiros usando instruções `if`, operadores relacionais e operadores de igualdade. (Parte 2 de 3.)

```
Enter first integer: 777
Enter second integer: 777
777 == 777
777 <= 777
777 >= 777
```

```
Enter first integer: 1000
Enter second integer: 2000
1000 != 2000
1000 < 1000
1000 <= 2000
```

```
Enter first integer: 2000
Enter second integer: 1000
2000 != 1000
2000 > 1000
2000 >= 1000
```

Figura 2.15 | Compare números inteiros usando instruções `if`, operadores relacionais e operadores de igualdade. (Parte 3 de 3.)

- Uma estrutura `if` sempre inicia com a palavra-chave `if`, seguida por uma condição entre parênteses.

  Espera uma instrução em seu corpo, mas pode conter múltiplas instruções se elas estiverem entre chaves (`{}`).

  O recuo da instrução no corpo mostrado aqui não é exigido, mas melhora a legibilidade do programa enfatizando que as instruções são parte do corpo.
- Observe que não há ponto e vírgula (`;`) no fim da primeira linha de cada instrução `if`.

  Esse ponto e vírgula resultaria em um erro de lógica em tempo de execução.

  Tratado como uma **instrução vazia** — um ponto e vírgula sozinho.

Java™ COMO PROGRAMAR 8ª edição

Erro comum de programação 2.8

Esquecer o parêntese esquerdo e/ou direito para a condição em uma estrutura `if` *é um erro de sintaxe — os parênteses são requeridos.*

Java™ COMO PROGRAMAR 8ª edição

DEITEL®

Erro comum de programação 2.9

Confundir o operador de igualdade, ==, com o operador de atribuição, =, pode causar um erro de lógica ou um erro de sintaxe. O operador de igualdade deve ser lido como "igual a", e o operador de atribuição deve ser lido como "obtém" ou "obtém o valor de". Para evitar confusão, algumas pessoas leem o operador de igualdade como "duplo igual" ou "igual igual".

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 2.14

Colocar apenas uma instrução por linha em um programa aprimora a legibilidade do programa.

Java™ COMO PROGRAMAR 8ª edição

Erro comum de programação 2.10

Colocar um ponto-e-vírgula imediatamente depois do parêntese direito da condição em uma instrução `if` *é normalmente um erro de lógica.*

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 2.15

Uma instrução longa pode se estender por várias linhas. Se uma única instrução deve ser dividida em várias linhas, escolha dividi-la em pontos que fazem sentido, como depois de uma vírgula em uma lista separada por vírgulas ou depois de um operador em uma expressão longa. Se uma instrução for dividida em duas ou mais linhas, recue todas as linhas subsequentes até o fim da instrução.

Operadores	Associatividade	Tipo
* / %	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	atribuição

Figura 2.16 | Precedência e associatividade dos operadores discutidos.

Java™ COMO PROGRAMAR 8ª edição

Boa prática de programação 2.16

Consulte o gráfico de operador de precedência (Apêndice A) ao escrever expressões contendo muitos operadores. Confirme se as operações na expressão são realizados na ordem em que você espera. Se você não tiver certeza sobre a ordem de avaliação em uma expressão complexa, utilize parênteses para forçar a ordem, exatamente como faria em expressões algébricas.