

2

Introdução aos aplicativos Java



OBJETIVOS

Neste capítulo, você aprenderá:

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



2.1 Introdução

- **Programação de aplicativo Java**
 - Exibir mensagens.
 - Obter informações do usuário.
 - Cálculos aritméticos.
 - Fundamentos de tomada de decisão.



2.2 Primeiro programa Java: Imprimindo uma linha de texto

- **Aplicativo**
 - Executa quando você utiliza o comando `java` para carregar a Java Virtual Machine (JVM).
- **Programa de exemplo**
 - Exibe uma linha de texto.
 - Ilustra diversos aspectos importantes da linguagem Java.



Resumo

welcome1.java

```
1 // Fig. 2.1: welcome1.java
2 // Programa de impressão de texto.
3
4 public class welcome1
5 {
6     // método main inicia a execução do aplicativo Java
7     public static void main( String args[] )
8     {
9         System.out.println( "welcome to Java Programming!" );
10
11     } // fim do método main
12
13 } // fim da classe welcome1
```

welcome to Java Programming!



2.2 Primeiro programa Java: Imprimindo uma linha de texto (*Continuação*)

```
1 // Fig. 2.1: welcome1.java
```

– Comentários iniciam com: //

- Comentários ignorados durante a execução do programa.
- Documentam e descrevem o código.
- Fornecem legibilidade do código.

– Comentários tradicionais: /* ... */

```
/* Este é um comentário  
   tradicional. Ele pode se estender  
   por várias linhas */
```

```
2 // Programa de impressão de texto.
```

- Outra linha de comentários.
- Nota: os números de linha não fazem parte do programa; eles foram adicionados para referência.



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 da criação de um programa adequado nessa linguagem. 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 não produz um arquivo `.class`. Em vez disso, o compilador emite uma mensagem de erro para ajudar o programador a identificar e corrigir o código incorreto. Erros de sintaxe também são chamados de *erros de compilador*, *erros em tempo de compilação* ou *erros de compilação*, porque o compilador detecta-os durante a fase de compilação. Não será possível executar seu programa até você corrigir todos os erros de sintaxe nele.



Boa prática de programação 2.1

Cada programa deve iniciar com um comentário que explica o propósito do programa, o autor, a data e a hora em que programa foi modificado pela última vez. (Não mostramos o autor, a data e a hora nos programas do *Java: Como programar*, 6ª edição porque essas informações seriam redundantes.)



2.2 Primeiro programa Java: Imprimindo uma linha de texto (*Continuação*)

3

- **Linha em branco.**
 - Torna o programa mais legível.
 - Linhas em branco, espaços e tabulações são caracteres de espaço em branco.

4 `public class` welcome1 4

- **Inicia a declaração de classe para a classe `Welcome1`**
 - Cada programa Java tem pelo menos uma classe definida pelo usuário.
 - **Palavra-chave:** palavras reservadas para uso pelo Java.
 - Palavra-chave `class` seguida pelo nome da classe.
 - **Atribuindo nomes de classes:** coloque a inicial de cada palavra em maiúscula.
 - `ExemploDeNomeDeClasse`



Boa prática de programação 2.2

Utilize linhas em branco e caracteres de espaço em branco para aprimorar a legibilidade do programa.



4 `public class` welcome1

– Identificador Java

- Série de caracteres consistindo em letras, dígitos, sublinhados (`_`) e sinais de cifrão (`$`).
- Não inicia com um dígito, não tem nenhum espaço.
- Exemplos: `cwelcome1`, `$value`, `_value`, `button7t`.
- `7button` não é válido.
- O Java diferencia letras maiúsculas de minúsculas (o uso de letras maiúsculas ou minúsculas importa), e `a1` e `A1` são diferentes.
- Nos capítulos 2 a 7, utilize a classe `public`.
- Certos detalhes não são importantes agora.
- Simulam certos recursos, discutidos mais tarde.



Boa prática de programação 2.3

Por convenção, sempre inicie o identificador do nome de uma classe com uma letra maiúscula e inicie cada palavra subsequente no identificador com uma letra maiúscula.

Programadores em Java sabem que tais identificadores normalmente representam classes Java; portanto, nomear suas classes dessa maneira torna seus programas mais legíveis.



Erro comum de programação 2.2

O Java diferencia letras maiúsculas de minúsculas. Não diferenciar as letras maiúsculas e minúsculas adequadas para um identificador normalmente causa um erro de compilação.



2.2 Primeiro Programa Java: Imprimindo uma linha de texto (*Continuação*)

```
4 public class welcome1
```

– Salvando arquivos:

- O nome do arquivo deve iniciar com a extensão .java.
- welcome1.java

```
5 {
```

– Chave esquerda {:

- Inicia o corpo de toda classe.
- A chave esquerda termina as declarações (linha 13).



Erro comum de programação 2.3

É um erro uma classe `public` ter um nome de arquivo que não é idêntico ao nome de classe (mais a extensão `.java`) tanto em termos de ortografia quanto do uso de letras maiúsculas e minúsculas.



Erro comum de programação 2.4

É um erro não terminar um nome de arquivo com a extensão `.java` no caso de um arquivo contendo uma declaração de classe. Se essa extensão estiver faltando, o compilador Java não será capaz de compilar a declaração de classe.



Boa prática de programação 2.4

Sempre que você digitar uma chave esquerda de abertura, {, em seu programa, digite imediatamente a chave direita, }, de fechamento e, então, reposicione o cursor entre as chaves e crie um recuo para começar a digitar o corpo. Essa prática ajuda a evitar erros em razão da ausência de chaves.



Boa prática de programação 2.5

Recue o corpo inteiro de cada declaração de classe por um ‘nível’ de recuo 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.

Boa prática de programação 2.6

Configure uma convenção para o tamanho de recuo desejado e, então, aplique uniformemente essa convenção.

A tecla *Tab* pode ser utilizada para criar recuos, mas as paradas de tabulação variam entre editores de textos.

Recomendamos utilizar três espaços para formar um nível de recuo.



Erro comum de programação 2.5

É um erro de sintaxe caso as chaves não ocorram em pares correspondentes.

2.2 Primeiro programa Java: imprimindo uma linha de texto (*Continuação*)

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

- Aplicativos começam a executar em `main`.
 - Parênteses indicam que `main` é um método (caps. 3 e 6).
 - Aplicativos Java contêm um ou mais métodos.
- Exatamente um método deve se chamar `main`.
- **Métodos podem realizar tarefas e retornar informações:**
 - `void` significa que `main` não retorna nenhuma informação.
 - Por enquanto, simula a primeira linha de `main`.

```
8 {
```

- A chave esquerda, `{`, inicia o corpo de declaração do método.
- A chave direita, `}`, termina o corpo (linha 11 do programa).



Boa prática de programação 2.7

Recue o corpo inteiro de cada declaração de método por um ‘nível’ de recuo entre a chave esquerda, {, e a chave direita, }, que delimitam o corpo da classe. Esse formato faz com que a estrutura do método se destaque, tornando a declaração do método mais fácil de ler.

2.2 Primeiro programa Java: Imprimindo uma linha de texto (*Continuação*)

9

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

– Instrui o computador a realizar uma ação:

- Imprime strings de caracteres.
 - String: série de caracteres entre aspas duplas.
- Espaços em branco em strings não são ignorados pelo compilador.

– `System.out`:

- Objeto de saída-padrão.
- Imprime na janela de comando (isto é, Prompt do MS-DOS).

– Método `System.out.println`:

- Exibe uma linha de texto.

– Isso é conhecido como uma declaração:

Erro comum de programação 2.6

Omitir o ponto-e-vírgula no fim de uma instrução é um erro de sintaxe.



Dica de prevenção de erros 2.1

Ao aprender a programar, às vezes, é útil ‘quebrar’ um programa funcional para você poder se familiarizar 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 no futuro, você terá uma idéia 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.



Dica de prevenção de erros 2.2

Quando o compilador informa um erro de sintaxe, o erro pode não estar no número da linha indicado 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.



2.2 Primeiro programa Java: Imprimindo uma linha de texto (*Continuação*)

```
11    } // fim do método main
```

- Termina a declaração de método.

```
13 } // fim da classe welcome1
```

- Termina a declaração da classe.
- É possível adicionar comentários para monitorar chaves finais.



Boa prática de programação 2.8

Melhora a legibilidade de programa colocar, depois da chave direita de fechamento (}) do corpo da declaração de um método ou classe, um comentário de fim de linha indicando a declaração de método ou classe a que a chave pertence.



2.2 Primeiro programa Java: Imprimindo uma linha de texto (*Continuação*)

- **Compilando um programa:**
 - Abra uma janela de prompt de comando e vá para o diretório onde programa está armazenado.
 - Digite `javac welcome1.java`
 - Se nenhum erro de sintaxe ocorrer, `welcome1.class` será criado.
 - O arquivo `welcome1.class` contém bytecodes Java que representam o aplicativo.
 - Os bytecodes serão executados pela JVM.



Dica de prevenção de erros 2.3

Ao tentar compilar um programa, caso receba uma mensagem como ‘bad command or filename’, ‘javac: command not found’ ou ‘javac’ is not recognized as an internal or external command, operable program or batch file’, então a instalação do seu software Java não foi completada adequadamente.

Se estiver utilizando o J2SE Development Kit, isso indicará que a variável de ambiente *PATH* do sistema não foi configurada adequadamente.

Revise as instruções da instalação do J2SE Development Kit em java.sun.com/j2se/5.0/install.html cuidadosamente.

Em alguns sistemas, depois de corrigir o *PATH*, talvez seja necessário reinicializar seu computador ou abrir uma nova janela de comando para que essas configurações tenham efeito.



Dica de prevenção de erros 2.4

O compilador Java gera mensagens de erro de sintaxe quando a sintaxe de um programa está incorreta.

Cada mensagem de erro contém o nome do arquivo e o 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 informações sobre o erro de sintaxe.



Dica de prevenção de erros 2.5

A mensagem de erro do compilador ‘`Public class NomeDaClasse must be defined in a file called NomeDaClasse.java`’ indica que o nome do arquivo não corresponde exatamente ao nome da classe `public` no arquivo ou que o nome da classe foi digitado incorretamente ao compilar a classe.

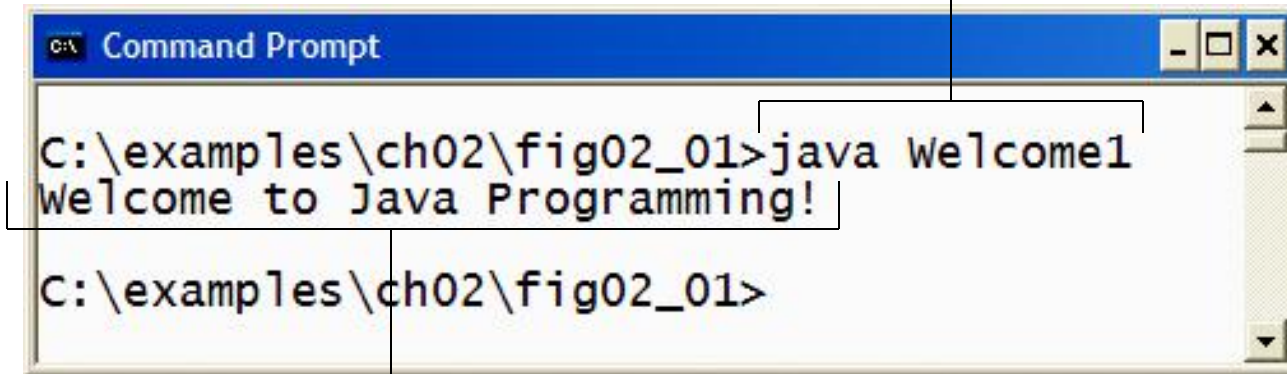


2.2 Primeiro programa Java: imprimindo uma linha de texto (*Continuação*)

- **Executando um programa:**
 - **Digite `java Welcome1`**
 - Isso carrega a JVM.
 - A JVM carrega o arquivo `.class` para a classe `Welcome1`.
 - A extensão `.class` foi omitida no comando.
 - A JVM chama o método `main`.



Você digita esse comando para executar o aplicativo.



The image shows a screenshot of a Windows XP Command Prompt window. The title bar is blue and says "C:\ Command Prompt". The window has standard Windows XP window controls (minimize, maximize, close) on the right. The command prompt shows the following text:

```
C:\examples\ch02\fig02_01>java welcome1
Welcome to Java Programming!
C:\examples\ch02\fig02_01>
```

There are two lines of text in the command prompt. The first line is the command `java welcome1` followed by a prompt character `>`. The second line is the output `Welcome to Java Programming!` followed by a prompt character `>`. A vertical line from the text "Você digita esse comando para executar o aplicativo." points to the command `java welcome1`. Another vertical line from the text "O programa gera a saída Welcome to Java Programming!" points to the output line.

O programa gera a saída
Welcome to Java Programming!

Figura 2.2 | Executando `welcome1` em uma janela de Prompt de Comando do Microsoft Windows XP.

Dica de prevenção de erros 2.6

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 adequadamente.

Revise as instruções da instalação do J2SE Development Kit com cuidado. Em alguns sistemas, talvez seja necessário reinicializar seu computador ou abrir uma nova janela de comando depois de configurar o *CLASSPATH*.



2.3 Modificando nosso primeiro programa Java

Modifique o exemplo na Figura 2.1 para imprimir o mesmo conteúdo utilizando código diferente.



2.3 Modificando nosso primeiro programa Java (*Continuação*)

- **Modificando programas:**

- `welcome2.java` (Figura 2.3) produz a mesma saída que `welcome1.java` (Figura 2.1).
- Utilizando código diferente:

```
9      System.out.print( "welcome to " );  
10     System.out.println( "Java Programming!" );
```

- A linha 9 exibe “welcome to ” com o cursor posicionado na linha impressa.
- A linha 10 exibe “Java Programming!” na mesma linha com o cursor na linha seguinte.

Resumo

welcome2.java

1. Comentários

2. Linha em branco

3. Inicia a classe
welcome2

3.1 Método main

4. Método
System.out.print

4.1 Método
System.out.println

5. fim de main,
welcome2

Saída do programa

```
1 // Fig. 2.3: welcome2.java
2 // Imprimindo uma linha de texto com múltiplas instruções.
3
4 public class welcome2
5 {
6     // método main 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
12    } // fim do método main
13
14 } // fim da classe welcome2
```

System.out.print
mantém o cursor na mesma
linha, assim
System.out.println
continua na mesma linha.

Welcome to Java Programming!



2.3 Modificando nosso primeiro programa Java (*Continuação*)

- **Caractere de escape:**
 - Barra invertida (\).
 - Indica que caracteres especiais serão enviados para a saída.
- **Caractere de nova linha (\n):**
 - Interpretado como ‘caractere especial’ pelos métodos `System.out.print` e `System.out.println`.
 - Indica que o cursor deve estar no início da próxima linha.
 - `Welcome3.java` (Figura 2.4).

```
9      System.out.println( "Welcome\nto\nJava\nProgramming!" );
```

- A linha é quebrada em \n.



Resumo

welcome3.java

1. main

2. System.out.println
(usa \n para cada nova
linha)

Saída do programa

```
1 // Fig. 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
11     } // fim do método main
12
13 } // fim da classe welcome3
```

welcome
to
Java
Programming!

Note como é gerada a saída de uma nova linha para cada sequência de escape \n.



Seqüência de escape	Descrição
<code>\n</code>	Nova linha. Posiciona o cursor de tela no início da próxima linha.
<code>\t</code>	Tabulação horizontal. Move o cursor de tela para a próxima parada de tabulação.
<code>\r</code>	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.
<code>\\</code>	Barras invertidas. Utilizadas para imprimir um caractere de barra invertida.
<code>\"</code>	Aspas duplas. Utilizadas para imprimir um caractere de aspas duplas. Por exemplo, <pre>System.out.println("\"entre aspas\"");</pre> exibe <code>"in quotes"</code>

Figura 2.5 | Algumas seqüências de escape comuns.

2.4 Exibindo texto com printf

- **System.out.printf:**
 - Novo recurso do J2SE 5.0.
 - Exibe dados formatados.

```
9      System.out.printf( "%s\n%s\n",  
10         "welcome to", "Java Programming!" );
```

- **String de formato:**
 - Texto fixo.
 - Especificador de formato – marcador de lugar para um valor.
 - Especificador de formato **%S** – marcador de lugar para uma string.



Resumo

welcome4.java

main

printf

System.out.printf
exibe dados formatados.

Saída do programa

```
1 // Fig. 2.6: welcome4.java
2 // Imprimindo múltiplas linhas em uma caixa de diálogo.
3
4 public class welcome4
5 {
6     // método main 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
12     } // fim do método main
13
14 } // fim da classe welcome4
```

```
welcome to
Java Programming!
```



Boa prática de programação 2.9

**Para tornar os programas mais legíveis,
coloque um espaço depois de cada vírgula
(,) em uma lista de argumentos.**

Erro comum de programação 2.7

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



2.5 Outros aplicativos Java: Adicionando inteiros

- **Próximo programa:**
 - **Utilize `Scanner` para ler dois inteiros fornecidos pelo usuário.**
 - **Utilize `printf` para exibir a soma de dois valores.**
 - **Utilize pacotes.**



Resumo

Addition.java

(1 de 2)

declaração import

Scanner

nextInt

```

1 // Fig. 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 main inicia a execução do aplicativo Java
8     public static void main( String args[] )
9     {
10         // cria Scanner para obter entrada da janela de coma
11         Scanner input = new Scanner( System.in );
12
13         int number1; // primeiro número a somar
14         int number2; // segundo número a somar
15         int sum; // soma de number1 e number2
16
17         System.out.print( "Enter first integer: " ); // prompt
18         number1 = input.nextInt(); // lê o primeiro número fornecido pelo usuário
19     }

```

A declaração import importa a classe Scanner do pacote java.util.

Declara e inicializa a variável input, que é um Scanner.

Declara as variáveis number1, number2 e sum.

Lê um inteiro fornecido pelo usuário e o atribui a number1.

Resumo

dition.java

de 2)

4. Adição

5. printf

Lê um inteiro fornecido pelo usuário e o atribui a **number2**.

Calcula a soma das variáveis **number1** e **number2**; atribui o resultado a **sum**.

Exibe a soma utilizando saída formatada.

Dois inteiros inseridos pelo usuário.

```

20 System.out.print( "Enter second integer: " ); // prompt
21 number2 = input.nextInt(); // lê segundo número fornecido pelo usuário
22
23 sum = number1 + number2; // soma os números
24
25 System.out.printf( "Sum is %d\n", sum ); //
26
27 } // fim do método main
28
29 } // fim da classe Addition
  
```

```

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



2.5 Outros aplicativos Java: Adicionando inteiros (*Continuação*)

```
3  import java.util.Scanner;  // programa utiliza a classe Scanner
```

– Declarações `import`:

- Utilizado pelo compilador para identificar e localizar classes utilizadas em programas Java.
- Instrui o compilador a carregar a classe `Scanner` a partir do pacote `java.util`.

```
5  public class Addition  
6  {
```

– Inicia a classe `public Addition`.

- Lembre-se de que o nome do arquivo deve ser `Addition.java`.

– As linhas 8-9: iniciam `main`.



Erro comum de programação 2.8

Todas as declarações `import` devem aparecer antes da primeira declaração de 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.



Dica de prevenção de erros 2.7

Em geral, esquecer de incluir uma declaração `import` para uma classe utilizada no seu programa resulta em um erro de compilação contendo uma mensagem como `'cannot resolve symbol'`.

Quando isso ocorre, verifique se você forneceu as declarações `import` adequadas e se os nomes nas declarações `import` estão escritos corretamente, incluindo a utilização adequada de letras maiúsculas e minúsculas.



2.5 Outros aplicativos Java: Adicionando inteiros (*Continuação*)

```
10 // cria Scanner para obter entrada a partir da janela de comando
11 Scanner input = new Scanner( System.in );
```

- **Instrução de declaração de variável.**
- **Variáveis:**
 - Localização na memória que armazena um valor.
 - Declare com nome e tipo antes de utilizar.
 - Input é do tipo Scanner .
 - Permite que um programa leia dados para uso.
 - Nome de variável: Qualquer identificador válido.
- **Declarações terminam com um ponto-e-vírgula (;).**
- **Inicializam variáveis em sua declaração:**

- Sinal de igual

2.5 Outros aplicativos Java: Adicionando inteiros (*Continuação*)

```
13      int number1; // primeiro número a somar
14      int number2; // segundo número a somar
15      int sum;      // soma de number1 e number2
```

- Declara as variáveis **number1**, **number2** e **sum** do tipo **int**:
 - **int** armazena valores inteiros: isto é, 0, -4, 97.
 - Os tipos **float** e **double** podem armazenar números decimais.
 - O tipo **char** pode armazenar um único caractere: isto é, x, \$, \n, 7.
 - **int**, **float**, **double** e **char** são tipos primitivos.
- É possível adicionar comentários para descrever o propósito das variáveis.

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

uma declaração.

- Utilize uma lista separada por vírgulas.

Boa prática de programação 2.10

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



Boa prática de programação 2.11

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



Boa prática de programação 2.12

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 de variável `firstNumber` tem um N maiúsculo na sua segunda palavra, `Number`.

2.5 Outros aplicativos Java: Adicionando inteiros (*Continuação*)

```
17      System.out.print( "Enter first integer: " ); // prompt
```

- A mensagem chamou um prompt – instrui o usuário a realizar uma ação.
- Pacote `java.lang`.

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

- Resultado da chamada a `nextInt` dada a `number1` utilizando o operador de atribuição (`=`).
 - Instrução de atribuição.
 - `=` é um operador binário, pois tem dois operandos.
 - A expressão à direita é avaliada e atribuída à variável à esquerda.
 - É lido como: `number1` obtém o valor de `input.nextInt()`.



Observação de engenharia de software 2.1

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



Boa prática de programação 2.13

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



2.5 Outros aplicativos Java: Adicionando inteiros (*Continuação*)

20

```
System.out.print( "Enter second integer: " ); // prompt
```

–Semelhante à instrução anterior:

- Pode para o usuário inserir o segundo inteiro

21

```
number2 = input.nextInt(); // lê o segundo número fornecido pelo usuário
```

–Semelhante à instrução anterior:

Adição de dois inteiros e atribuição ao resultado

23

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

–Instrução de atribuição:

- Calcula a soma de number1 e number2 (lado direito).
- Utiliza o operador de atribuição (=) para atribuir o resultado à variável sum.
- Lido como: sum obtém o valor de number1 + number2.
- number1 e number2 são operandos.



2.5 Outros aplicativos Java: Adicionando inteiros (*Continuação*)

25

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

- Use `System.out.printf` para exibir resultados.
- Especificador de formato `%d`.
 - Marcador de lugar para um valor `int`.

```
System.out.printf( "Sum is %d\n: " , ( number1 + number2 ) );
```

- Cálculos também podem ser realizados dentro de `printf`.
- Parênteses em torno da expressão `number1 + number2` não são necessários.



2.6 Conceitos de memória

- **Variáveis:**

- Cada variável tem um nome, um tipo, um tamanho e um valor.
 - O nome corresponde à posição na memória.
- Quando o novo valor é colocado numa variável, ele substitui (e destrói) o valor anterior.
- Ler variáveis na memória não as modifica.



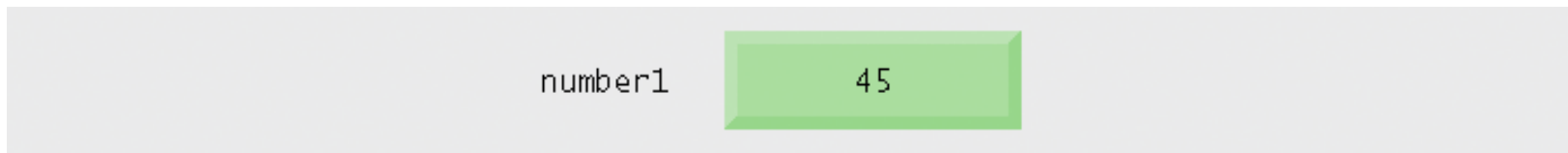


Figura 2.8 | Posição da memória mostrando o nome e o valor da variável number1.

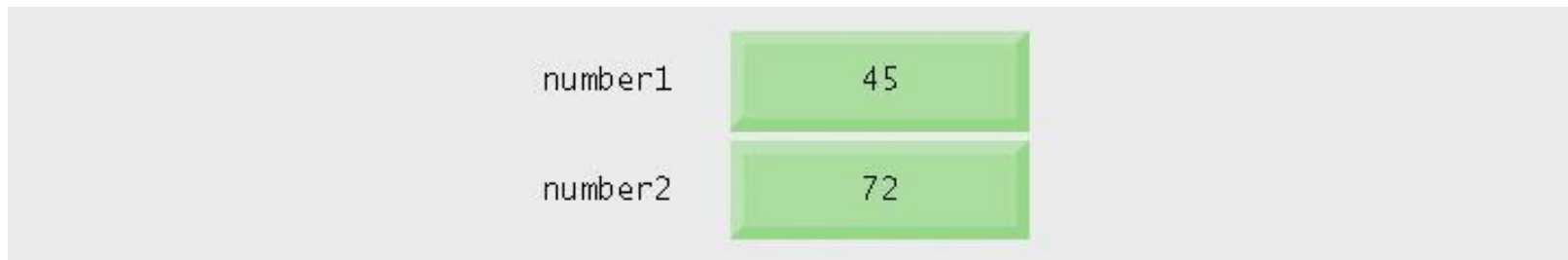


Figura 2.9 | Posições da memória depois de armazenar valores para number1 e number2.

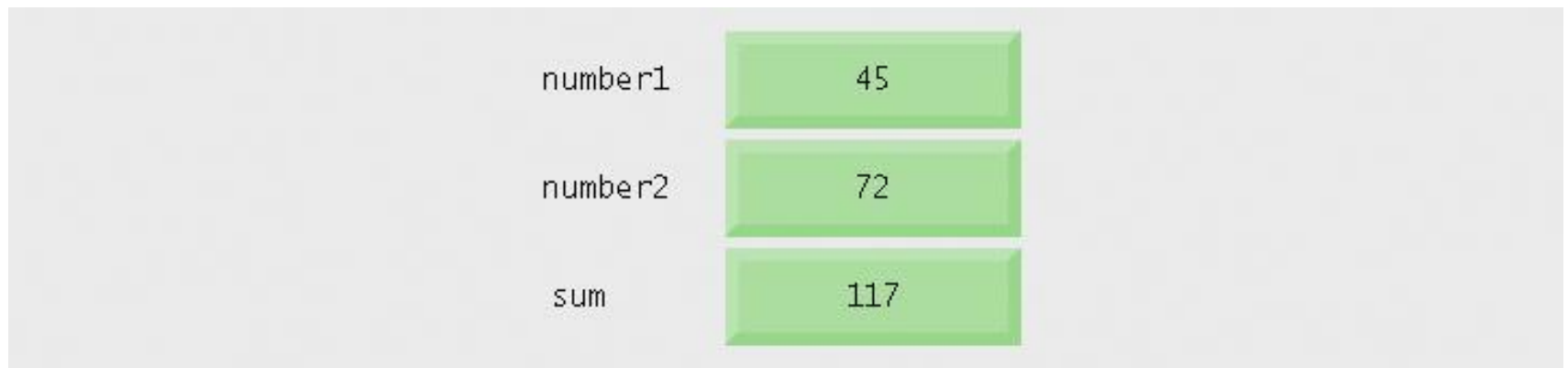


Figura. 2.10 | Posições da memória depois de calcular e armazenar a soma de number1 e number2.

2.7 Aritmética

- **Cálculos aritméticos utilizados na maioria dos programas:**
 - **Utilização:**
 - ***** para multiplicação
 - **/** para divisão
 - **%** para resto
 - **+, -**
 - **A divisão de inteiros trunca o resto:**
7 / 5 é avaliado como 1
 - **O operador de resto (ou módulo) % retorna o resto:**
7 % 5 é avaliado como 2



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

Figura 2.11 | Operadores aritméticos.

2.7 Aritmética (*Continuação*)

- **Precedência de operadores:**
 - **Alguns operadores aritméticos atuam antes de outros (isto é, a multiplicação ocorre antes da adição).**
 - Utilize parênteses quando necessário.
 - **Exemplo: Encontre a média de três variáveis a , b e c .**
 - Não utilize: $a + b + c / 3$
 - Utilize: $(a + b + c) / 3$



Operador(es)	Operação(ões)	Ordem de avaliação (precedência)
* / %	Multiplicação Divisão Resto	Avaliado primeiro. Se houver vários operadores desse tipo, eles são avaliados da esquerda para a direita.
+ -	Adição Subtração	

Figura 2.12 | Precedência de operadores aritméticos.



Boa prática de programação 2.14

Utilizar parênteses para expressões aritméticas complexas – mesmo quando os parênteses não são necessários – pode tornar as expressões aritméticas mais fáceis de ler.

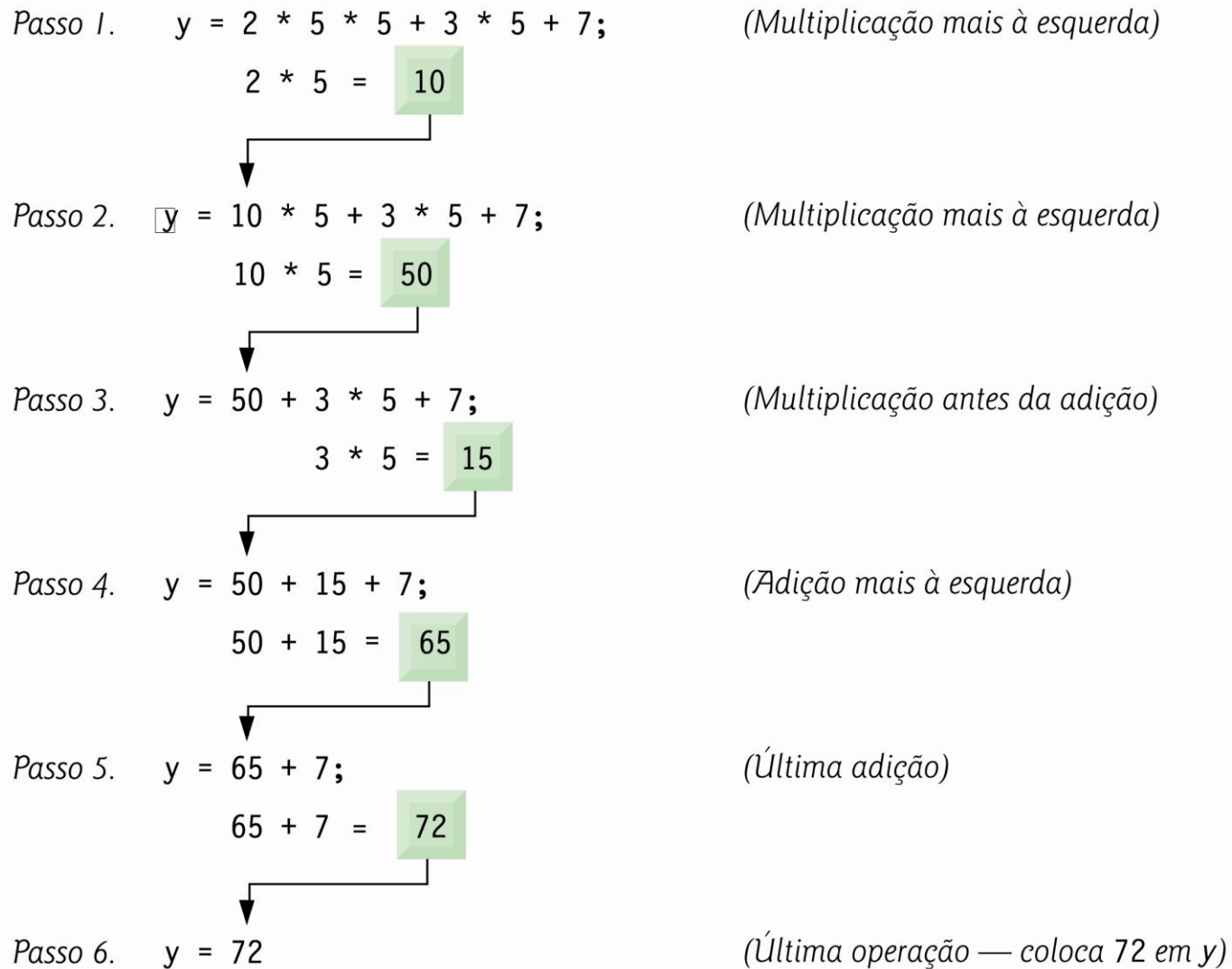


Figura 2.13 | Ordem em que um polinômio de segundo grau é avaliado.

2.8 Tomada de decisão: Operadores de igualdade e operadores relacionais

- **Condição:**

- Uma expressão pode ser `true` ou `false`.

- **instrução `if`:**

- Versão simples nesta seção; mais detalhes posteriormente.
- Se a condição é `true`, então o corpo da instrução `if` é executado.
- O controle sempre é retomado após a instrução `if`.
- Condições em instruções `if` podem ser formadas utilizando operadores de igualdade ou relacionais (*próximo slide*).



Operador algébrico de igualdade ou relacional padrão	Operador de igualdade ou relacional Java	Exemplo de condição no Java	Significado da condição no Java
<i>Operadores de igualdade</i>			
=	==	x == y	x é igual a y
≠	!=	x != y	x não é igual a y
<i>Operadores relacionais</i>			
>	>	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.



Resumo

Comparison.java

(1 de 2)

1. Classe Comparison

1.1 main

1.2 Declarações

1.3 Dados de entrada (nextInt)

1.4 Compara duas entradas utilizando instruções if

```

1 // Fig. 2.15: Comparison.java
2 // Compara inteiros utilizando instruções if, operadores relacionais
3 // e operadores de igualdade.
4 import java.util.Scanner; // programa utiliza a classe Scanner
5
6 public class Comparison
7 {
8     // método main inicia a execução do aplicativo Java
9     public static void main( String args[] )
10    {
11        // cria Scanner para obter entrada a partir 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 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        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 );

```

Testa quanto à igualdade, exibe o resultado utilizando printf.

Compara dois números utilizando o operador relacional <.



Resumo

Comparison.java

(2 de 2)

Compara dois números utilizando os operadores relacionais >, <= and >=.

```

31  if ( number1 > number2 )
32      System.out.printf( "%d > %d\n", number1, number2 );
33
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
41  } // fim do método main
42
43 } // fim da classe Comparison

```

Saída do programa

```

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 < 2000
1000 <= 2000

```

```

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

```



2.8 Tomada de decisão: Operadores de igualdade e operadores relacionais (*Continuação*)

- **Linha 6:** inicia a declaração da classe **Comparison**.
- **Linha 12:** declara a variável de entrada **Scanner** e lhe atribui um **Scanner** que insere dados da entrada-padrão.
- **Linhas 14-15:** declaram as variáveis **int**.
- **Linhas 17-18:** solicitam que o usuário digite o primeiro inteiro e insira o valor.
- **Linhas 20-21:** solicitam que o usuário digite o segundo inteiro e insira o valor.



2.8 Tomada de decisão: operadores de igualdade e operadores relacionais (Continuação)

```
23      if ( number1 == number2 )  
24          System.out.printf( "%d == %d\n", number1, number2 );
```

- A instrução `if` compara se há igualdade utilizando (`==`).
- Se as variáveis forem iguais (condição verdadeira), a linha 24 executa.
- Se as variáveis não forem iguais, a instrução é pulada.
- Nenhum ponto-e-vírgula no final da instrução `if`.
- Quando a instrução vazia executa, nenhuma tarefa é realizada.
- As linhas 26-27, 29-30, 32-33, 35-36 e 38-39 comparam `number1` e `number2`, utilizando os operadores `!=`, `<`, `>`, `<=` e `>=`, respectivamente.



Erro comum de programação 2.9

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



Erro comum de programação 2.10

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 lêem o operador de igualdade como ‘duplo igual’ ou ‘igual igual’.



Erro comum de programação 2.11

É um erro de sintaxe os operadores ==, !=, >= e <= conterem espaços entre seus símbolos como em = =, ! =, > = e < =, respectivamente.



Erro comum de programação 2.12

Inverter os operadores \neq , \geq e \leq , como em $=!$, $=>$ e $=<$, é um erro de sintaxe.



Boa prática de programação 2.15

Recue o corpo de uma estrutura `if` para fazer com que ele se destaque e para aprimorar a legibilidade do programa.



Boa prática de programação 2.16

Coloque somente uma instrução por linha em um programa. Esse formato aprimora a legibilidade do programa.



Erro de programação comum 2.13

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



Boa prática de programação 2.17

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.



Boa prática de programação 2.18

Consulte a tabela de precedência de operadores (veja a tabela completa no Apêndice A) ao escrever expressões que contêm vários operadores.

Confirme se as operações na expressão são realizadas na ordem em que você espera. Caso não tenha certeza sobre a ordem de avaliação em uma expressão complexa, utilize parênteses para forçar a ordem, exatamente como faria em expressões algébricas.

Observe que alguns operadores, como de atribuição, =, associam da direita para a esquerda não da esquerda para a direita.

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 das operações discutidas.



2.9 (Opcional) Estudo de caso de engenharia de software: Examinando o documento de requisitos

- **Projeto orientado a objetos (*object-oriented design*, OOD) utilizando UML:**
 - Capítulos 3 a 8, 10.
- **Implementação da programação orientada a objetos (OOP):**
 - Apêndice J.



2.9 (Opcional) Estudo de caso de engenharia de software (*Continuação*)

- **Documento de requisitos:**
 - **Novo caixa automático (*automated teller machine* – ATM):**
 - **Permite transações de financiamento básicas.**
 - **Exibe saldo e permite sacar dinheiro e depositar fundos.**
 - **Interface com o usuário.**
 - **Tela de exibição, teclado, dispensador de cédulas, abertura para depósito.**
 - **Sessão de ATM.**
 - **Autentica o usuário, executa transações financeiras.**

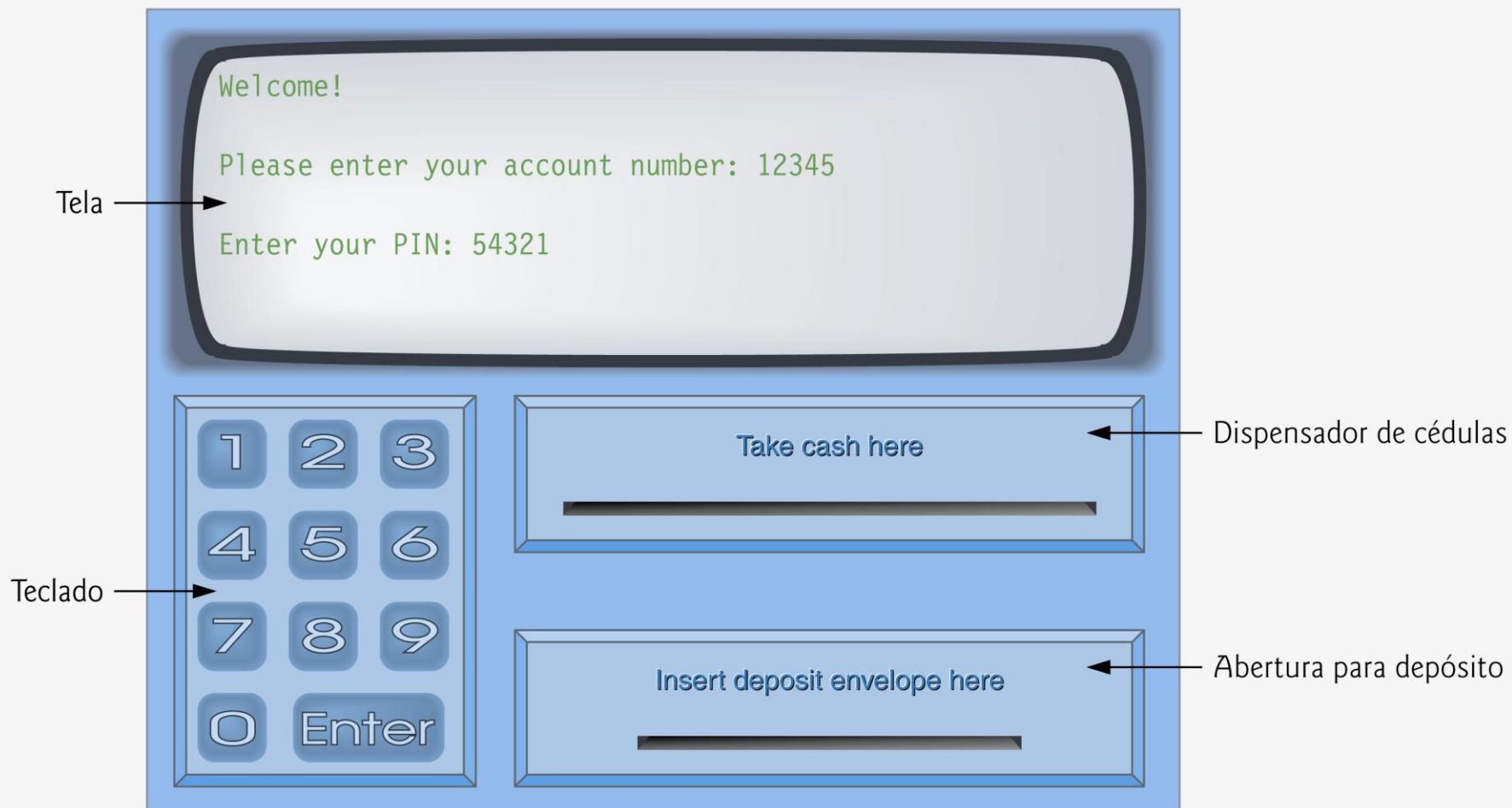


Figura 2.17 | A interface com o usuário do caixa eletrônico.



Figura 2.18 | Menu principal do ATM.



Figura 2.19 | Menu de saque do ATM.

2.9 (Opcional) Estudo de caso de engenharia de software (*Continuação*)

- **Analisando o sistema ATM:**
 - Coleta de requisitos.
 - Ciclo de vida de software:
 - Modelo em cascata.
 - Modelo iterativo.
 - Modelagem de casos de uso.
- **Diagramas de casos de uso:**
 - Modela as interações entre clientes e seus casos de uso.
 - Ator:
 - Entidade externa.



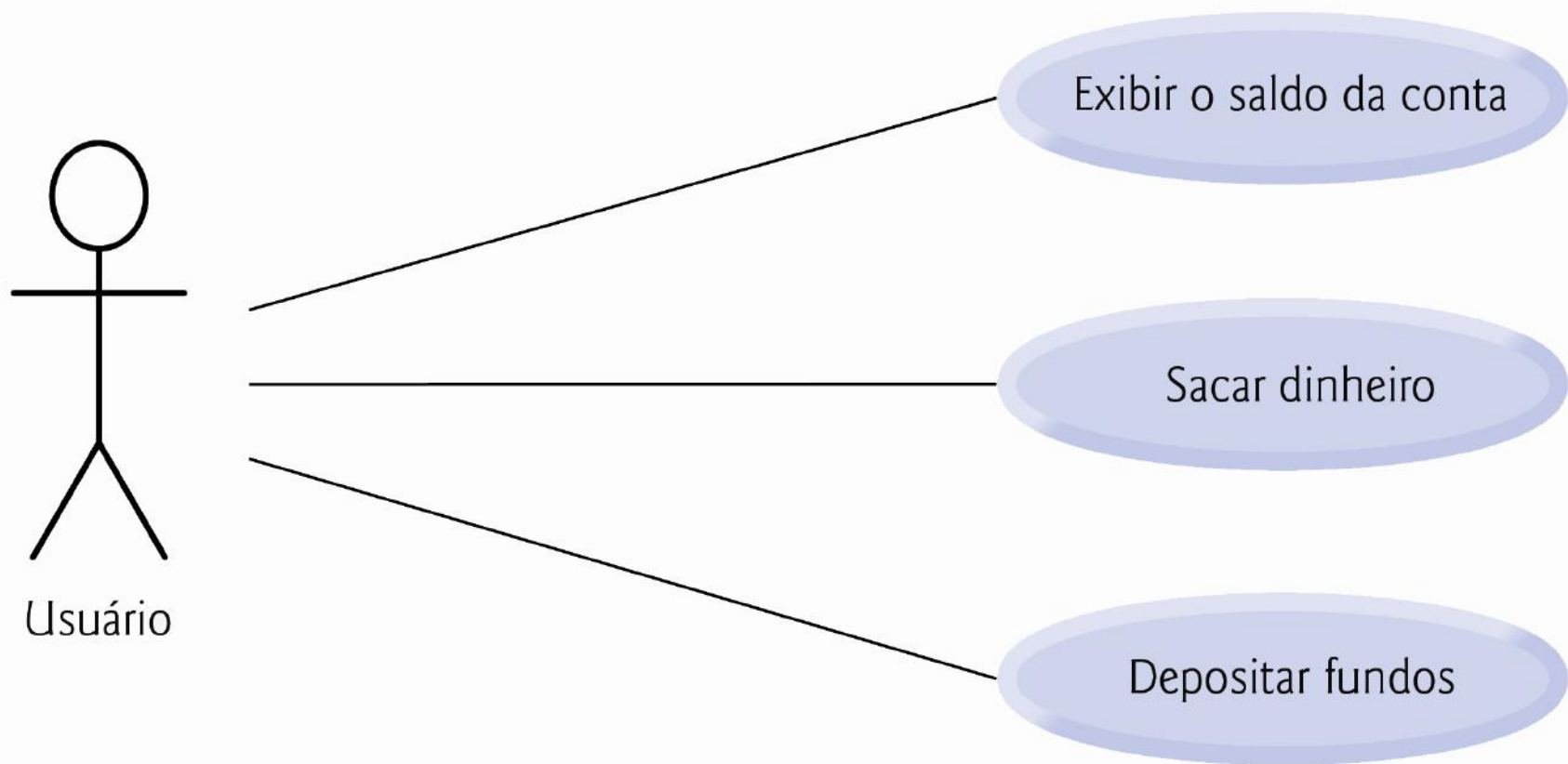


Figura 2.20 | Diagrama de caso de uso para o sistema ATM a partir da perspectiva do usuário.

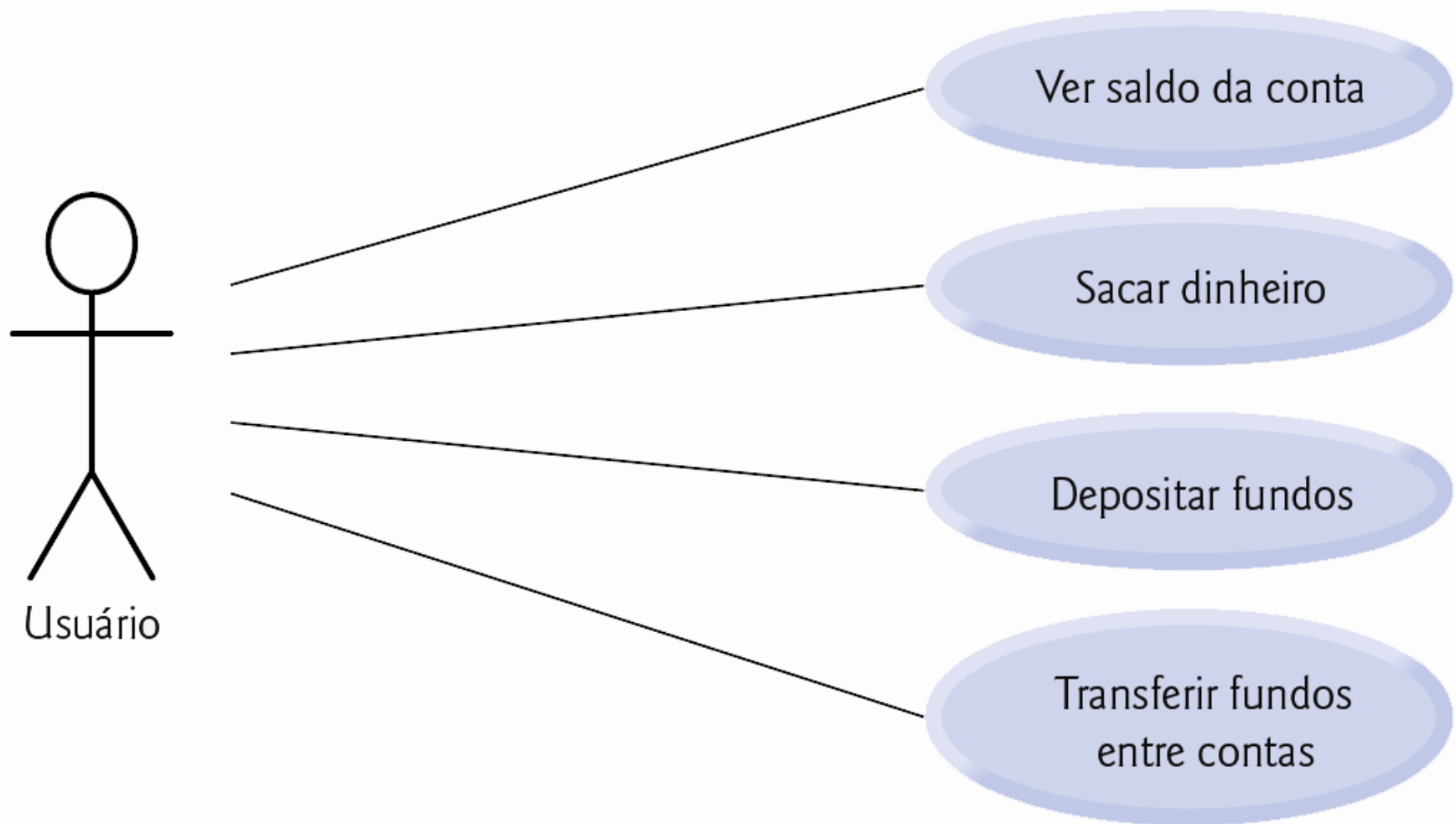


Figura 2.21 | Diagrama de caso de uso para uma versão modificada do nosso sistema ATM que também permite aos usuários transferir dinheiro entre contas.

2.9 (Opcional) Estudo de caso de engenharia de software (*Continuação*)

- **Tipos de diagrama UML:**
 - **Estrutura de sistema modelo:**
 - **Diagrama de classes:**
 - **Modela as classes, ou ‘blocos de construção’ de um sistema.**
 - **Tela de exibição, teclado, dispensador de dinheiro, slot de depósito.**



2.9 (Opcional) Estudo de caso de engenharia de software (*Continuação*)

– **Modela o comportamento do sistema:**

- **Diagramas de casos de uso:**
 - Modela interações entre o usuário e um sistema.
- **Diagramas de máquina de estado:**
 - Modela as maneiras como um objeto muda de estado.
- **Diagramas de atividade:**
 - Modela a atividade de um objeto durante a execução do programa.
- **Diagramas de comunicação (diagramas de colaboração):**
 - Modela as interações entre objetos.
 - Enfatiza quais alterações ocorrem.
- **Diagramas de seqüência:**
 - Modela as interações entre objetos.
 - Enfatiza quando as alterações ocorrem.

