

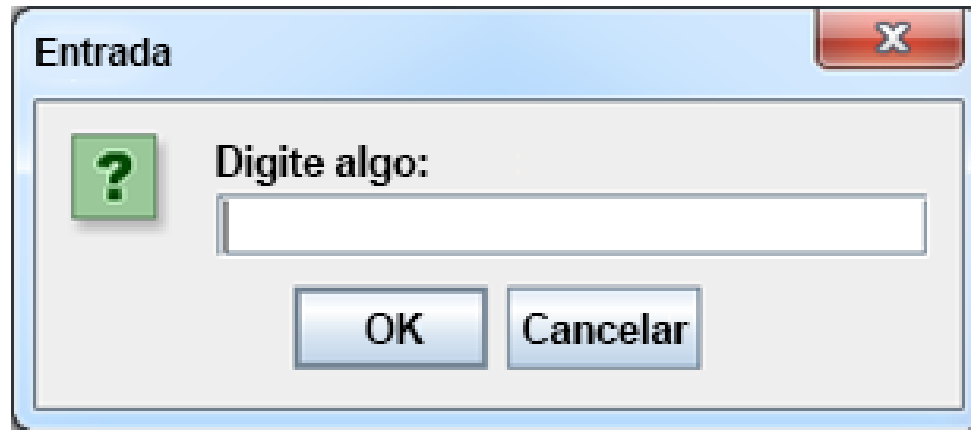
Como Obter a Entrada do Usuário

- Existem várias maneiras de obter a entrada do usuário:
 - Botões (físicos ou virtuais)
 - Discos e mostradores
 - Reconhecimento de voz
 - Caixas de diálogo de texto
 - Arquivos de propriedade
- O Java oferece muitas maneiras de obter entrada do usuário, inclusive...
 - **Swing** `JOptionPane`
 - `Scanner`

JOptionPane

Essa é uma maneira simples de obter entrada dos usuários:

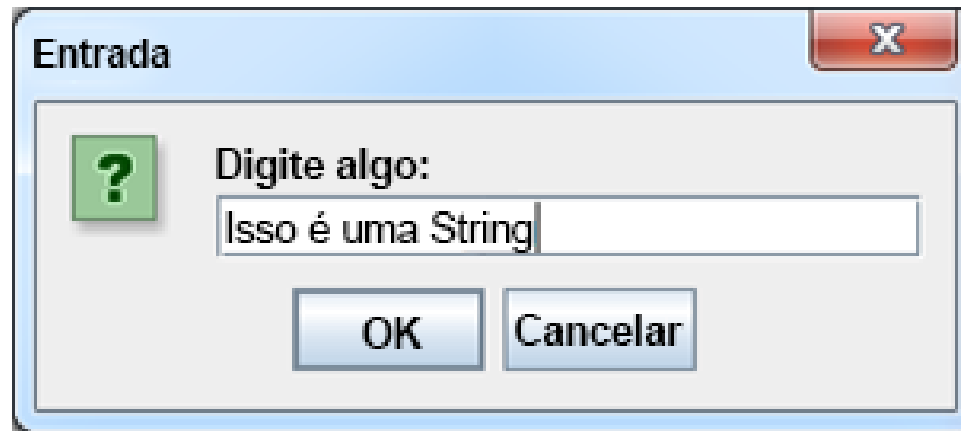
```
JOptionPane.showInputDialog("Digite algo:");
```



O JOptionPane Retorna Strings

- A entrada pode ser armazenada como uma String:

```
String input = JOptionPane.showInputDialog("Digite algo:");
```



- Isso é equivalente a escrever:

```
String input = "Isto é uma String";
```

Código Condensado

- Você poderia propagar sua entrada, fazendo parse e calculando em várias linhas:

```
String inputString =  
JOptionPane.showInputDialog("??");  
int input = Integer.parseInt(inputString);  
input++;
```

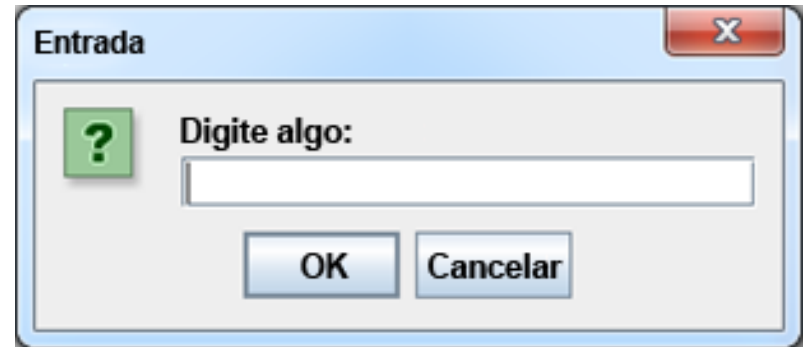
- Ou condensá-la em uma única linha:

```
int input = Integer.parseInt(JOptionPane.showInputDialog("??")) +1;
```

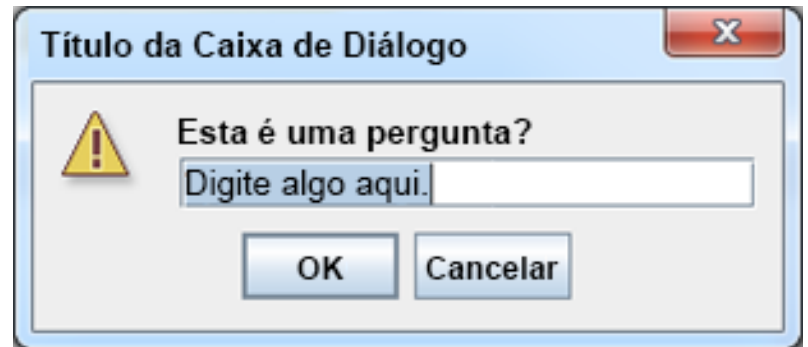
- Essa opção é uma questão de preferência pessoal.
 - Mas, se você precisar fazer referência a determinados valores novamente mais tarde, convém armazená-los em uma variável.

InputDialogs (caixas de diálogo de entrada) Diferentes

Criamos uma InputDialog simples:

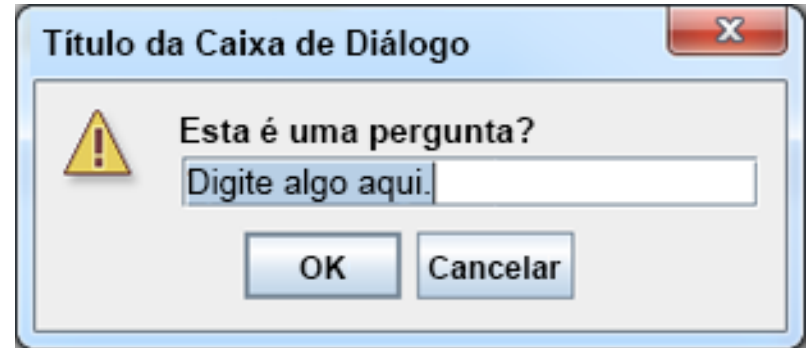


Com um código mais complexo, podemos personalizar mais a InputDialog:



Mais Opções com InputDialogs

- Esta versão de uma InputDialog não retorna uma String.
- O resultado deve ser convertido em uma String para que possa ser utilizado:

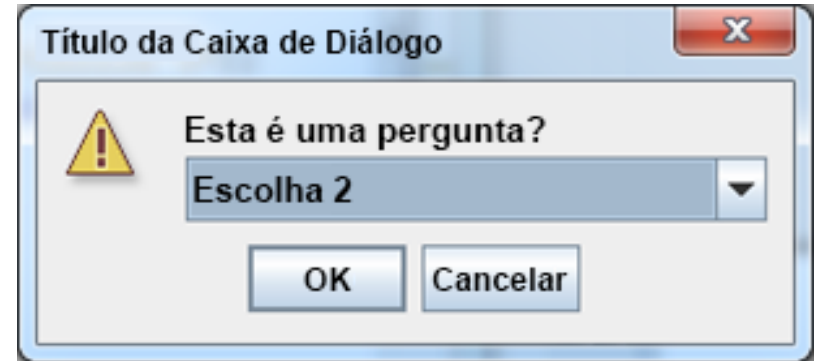


Conversão

```
String input = (String)JOptionPane.showInputDialog(null,  
    "Esta é uma pergunta?",  
    "Título da Caixa de Diálogo",  
    2,  
    null,  
    null,  
    "Digite algo aqui.");
```

Mais Opções com InputDialogs

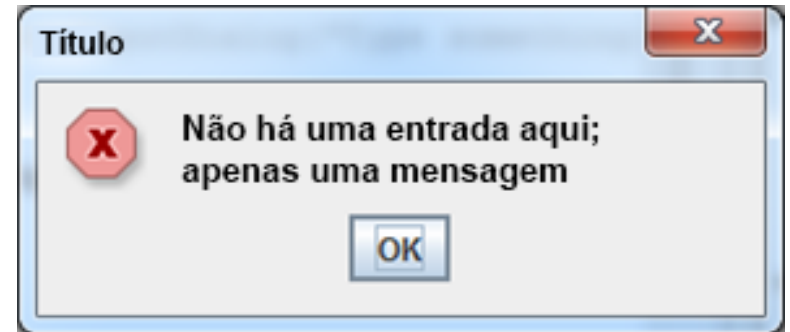
- Para evitar uma entrada indesejada, é possível fornecer somente valores aceitáveis para os usuários.
- Parte dessa sintaxe será discutida em detalhes na Seção 8.



```
String[] acceptableValues = {"Choice 1", "Choice 2", "Choice 3"};
String input2 = (String)JOptionPane.showInputDialog(null,
    "Esta é uma pergunta?",
    "Título da Caixa de Diálogo",
    2,
    null,
    acceptableValues,
    acceptableValues[1]);
```

showMessageDialog

- Uma `showMessageDialog` não fornece um campo de entrada.
- Existem muitas outras variações de `JOptionPane`.



```
JOptionPane.showMessageDialog(null,  
    "Não há uma entrada aqui; apenas uma mensagem",  
    "Título",  
    0);
```

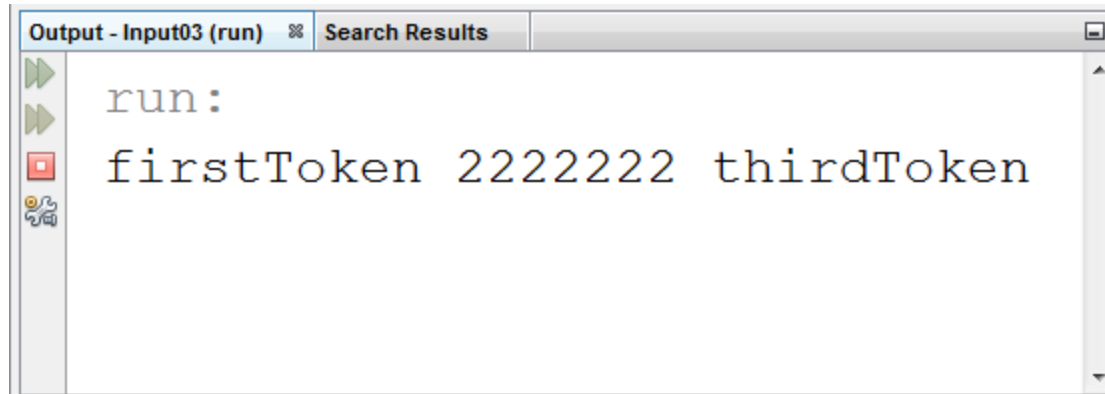

Obtendo Entrada com um Scanner

- Um objeto `Scanner` abre um fluxo para coleta da entrada:
 - `System.in` lê o `Scanner` para coletar entrada do console.
 - Digite sua entrada na janela de saída do NetBeans.
 - Também é possível usar o `Scanner` sem um IDE.
- Uma boa prática é fechar o fluxo do `Scanner` quando terminar.

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
    System.out.println(sc.next());  
    System.out.println(sc.next());  
    sc.close();  
}
```

Lendo a Entrada com um Scanner

- O Scanner procura **tokens**.
- Os tokens são separados por um **delimitador**.
 - O delimitador padrão é um espaço.



The screenshot shows an IDE output window titled "Output - Input03 (run)". The window contains the following text:

```
run:  
firstToken 2222222 thirdToken
```

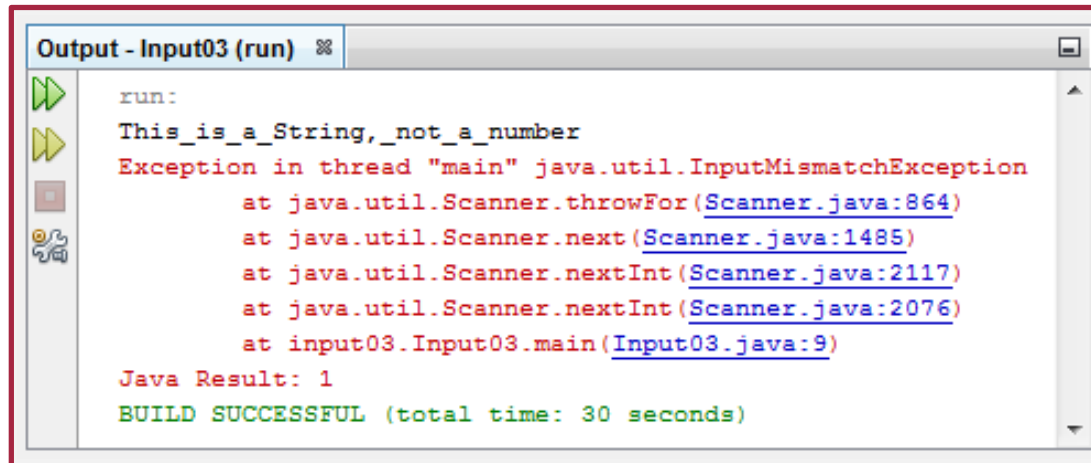
The text is displayed in a monospaced font. The output window has a standard IDE interface with a title bar, a search bar, and a vertical scrollbar on the right.

A Classe Scanner

- A classe Scanner, assim como qualquer outra, tem campos e métodos.
- Alguns métodos úteis do Scanner...
 - `nextInt()` lê o próximo token como um `int`.
 - `nextDouble()` lê o próximo token como um `double`.
 - `next()` lê o próximo token como uma `String`.

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
    int      x = sc.nextInt();  
    double   y = sc.nextDouble();  
    String   z = sc.next();  
    sc.close();  
}
```

Exceções: InputMismatchException

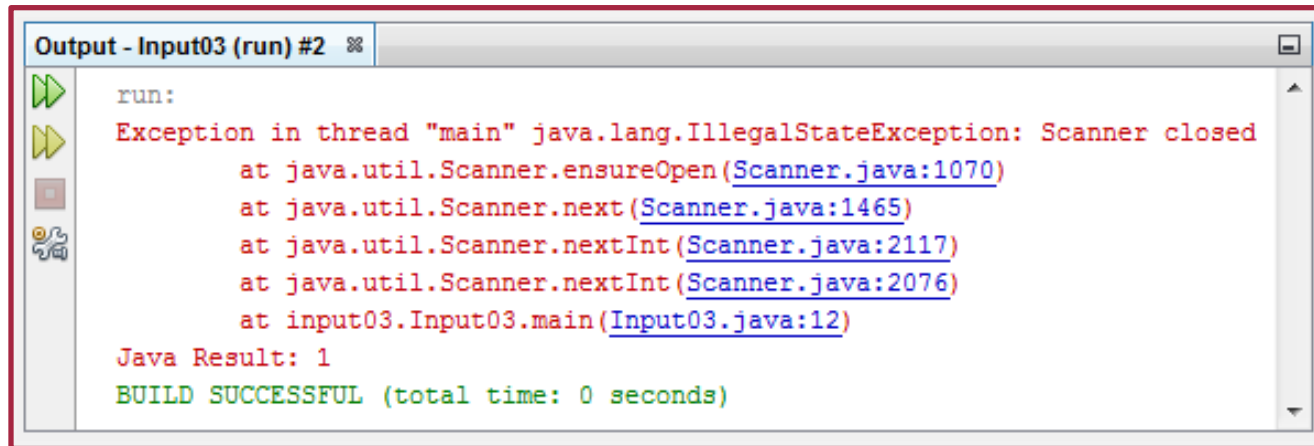
A screenshot of an IDE's output window titled "Output - Input03 (run)". The window shows the execution of a Java program. It starts with "run:" followed by the input "This_is_a_String,_not_a_number". Then, it displays a red error message: "Exception in thread \"main\" java.util.InputMismatchException". Below this, the stack trace is shown in red text, listing the following locations: "at java.util.Scanner.throwFor(Scanner.java:864)", "at java.util.Scanner.next(Scanner.java:1485)", "at java.util.Scanner.nextInt(Scanner.java:2117)", "at java.util.Scanner.nextInt(Scanner.java:2076)", and "at input03.Input03.main(Input03.java:9)". The stack trace lines are underlined. Below the exception, it says "Java Result: 1" and "BUILD SUCCESSFUL (total time: 30 seconds)".

```
run:
This_is_a_String,_not_a_number
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:864)
    at java.util.Scanner.next(Scanner.java:1485)
    at java.util.Scanner.nextInt(Scanner.java:2117)
    at java.util.Scanner.nextInt(Scanner.java:2076)
    at input03.Input03.main(Input03.java:9)
Java Result: 1
BUILD SUCCESSFUL (total time: 30 seconds)
```

Ocorre porque a entrada não pode ser analisada como o tipo esperado:

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println(sc.nextInt());
    sc.close();
}
```

Exceções: IllegalStateException

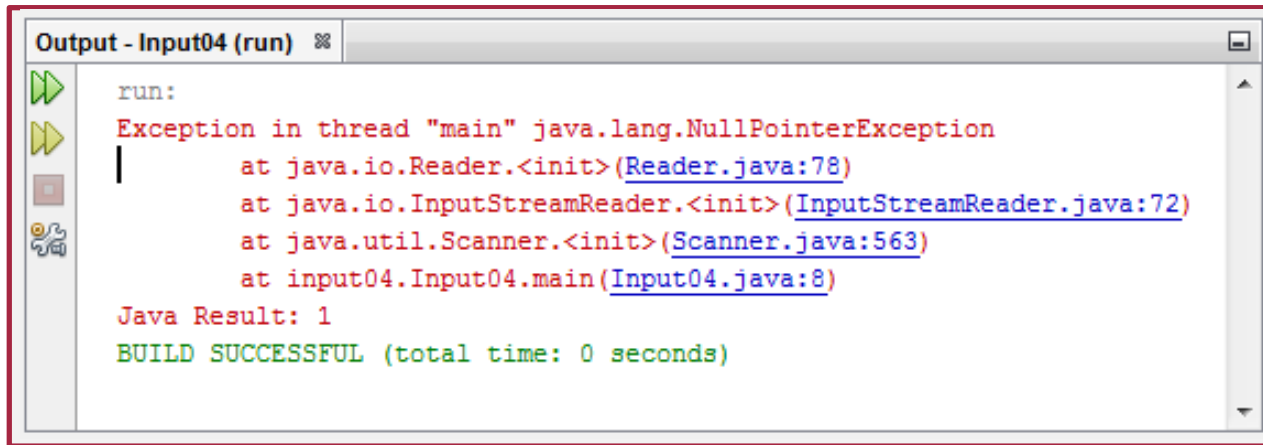
A screenshot of an IDE's output window titled "Output - Input03 (run) #2". The window shows the execution of a Java program. It starts with "run:" followed by a red error message: "Exception in thread \"main\" java.lang.IllegalStateException: Scanner closed". Below this, the stack trace is listed in red text: "at java.util.Scanner.ensureOpen(Scanner.java:1070)", "at java.util.Scanner.next(Scanner.java:1465)", "at java.util.Scanner.nextInt(Scanner.java:2117)", "at java.util.Scanner.nextInt(Scanner.java:2076)", and "at input03.Input03.main(Input03.java:12)". Below the stack trace, it says "Java Result: 1" and "BUILD SUCCESSFUL (total time: 0 seconds)".

```
run:
Exception in thread "main" java.lang.IllegalStateException: Scanner closed
    at java.util.Scanner.ensureOpen(Scanner.java:1070)
    at java.util.Scanner.next(Scanner.java:1465)
    at java.util.Scanner.nextInt(Scanner.java:2117)
    at java.util.Scanner.nextInt(Scanner.java:2076)
    at input03.Input03.main(Input03.java:12)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

Ocorre porque o fluxo é acessado depois de ter sido fechado:

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    sc.close();
    System.out.println(sc.nextInt());
}
```

Exceções: NullPointerException

A screenshot of an IDE's output window titled "Output - Input04 (run)". It shows the execution of a Java program that has crashed with a NullPointerException. The stack trace indicates the error occurred in the main method of Input04.java at line 8, while initializing a Scanner object. The stack trace lists the following sequence: Input04.main (Input04.java:8), Scanner.<init> (Scanner.java:563), InputStreamReader.<init> (InputStreamReader.java:72), and Reader.<init> (Reader.java:78). The output also shows "Java Result: 1" and "BUILD SUCCESSFUL (total time: 0 seconds)".

```
run:
Exception in thread "main" java.lang.NullPointerException
    at java.io.Reader.<init>(Reader.java:78)
    at java.io.InputStreamReader.<init>(InputStreamReader.java:72)
    at java.util.Scanner.<init>(Scanner.java:563)
    at input04.Input04.main(Input04.java:8)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

Ocorre porque "fakeFile.txt" não existe. Também é um erro comum esquecer a extensão .txt.

```
public static void main(String[] args) {
    Scanner sc = new Scanner(
        Input04.class.getResourceAsStream("fakeFile.txt"));
    sc.close();
}
```

Lembre-se da extensão.

Lendo de um Arquivo

- O Java oferece várias maneiras de ler arquivos.
- Os métodos `Scanner` mais úteis incluem o seguinte:
 - `nextLine()` avança esse scanner de volta à linha atual e retorna a entrada que foi ignorada.
 - `findInLine("String a ser Localizada")` Tenta localizar a próxima ocorrência de um padrão construído com base na String especificada ignorando delimitadores.

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(  
        Input04.class.getResourceAsStream("fakeFile.txt"));  
    int x = sc.nextInt();  
    String entireLine = sc.nextLine();  
    sc.close();  
}
```

Objetivos

- Declarar, inicializar e usar variáveis `booleana`
- Comparar expressões `booleana` usando operadores relacionais
- Criar uma instrução `if`
- Criar construções `if/else`
- Comparar `Strings`






Tomando Decisões

- Suponhamos que você esteja indo para a escola e se depara em um cruzamento.
- Agora você tem que tomar uma decisão lógica:
 - Se virar à esquerda, esse caminho me levará à escola?
 - Se for reto, esse caminho me levará à escola?
 - Se virar à direita, esse caminho me levará à escola?

Tipo de Dados `boolean` do Java

- É basicamente o mesmo que acontece no Java, onde valores `booleanas` informarão ao programa qual é o melhor curso de ação a tomar.
- No Java, os valores do tipo de dados `boolean` são `true` e `false`, em vez de `sim` e `não`.
- Você declara o tipo de dados `boolean` usando a palavra-chave `boolean`.

Usando um Tipo de Dados `boolean` do Java: Exemplo

```
public static void main(String args[]) {  
    boolean passed, largeVenue, grade;  Declarando variáveis booleanas  
    passed = true;  
    largeVenue = false;  
    grade = passed;  Atribuindo valores a variáveis booleanas  
    System.out.println(passed);  
    System.out.println(largeVenue );  
    System.out.println(grade);  Imprimindo valores de variáveis booleanas  
}
```

Expressões e Variáveis

As expressões booleanas podem ser...

- Impressas
- Atribuídas a uma variável booleana

```
System.out.println(x == 5);
```

```
boolean isFive = x == 5;
```

Igualdade e Atribuição

- `==` é um operador relacional.
- Este operador verifica se os dois lados de uma expressão `booleana` são iguais.
- Uma expressão `booleana` retorna um valor `true` ou `false`.

```
x == 5
```

Igualdade e Atribuição

- `=` é um operador de atribuição.
- Esse operador atribui um valor a uma variável.
- É possível atribuir a uma variável `booleana` qualquer valor que uma expressão `booleana` retorne.

```
int x = 4;
```

```
boolean isFive = x == 5;
```

Valores em Expressões booleanas

- Use `==` para testar a igualdade entre valores das primitivas.
- As expressões booleanas podem conter variáveis ou valores submetidos a hard-code.

```
boolean res1 = 24 == 15;  
System.out.println("res1: " + res1);  
  
int valor1 = 15;  
int valor2 = 24;  
boolean res2 = valor1 == valor2;  
System.out.println("res2: " + res1);
```

Valores em Expressões booleanas

As duas expressões abaixo retornam o mesmo valor:

- Se `value1` e `value2` contiverem o mesmo valor, a expressão retornará um resultado `true`.
- Caso contrário, a expressão retornará `false`.

```
boolean res1 = 24 == 15;  
System.out.println("res1: " + res1);  
  
int value1 = 15;  
int value2 = 24;  
  
boolean res2 = value1 == value2;  
System.out.println("res2: " + res1);
```


Operadores Relacionais

Use operadores relacionais em expressões booleanas que são utilizados para avaliar instruções `if/else`.

Operadores Relacionais

Condição	Operador	Exemplo
É igual a	==	<code>int i=1; (i == 1)</code>
Não é igual a	!=	<code>int i=2; (i != 1)</code>
É menor que	<	<code>int i=0; (i < 1)</code>
É menor que ou igual a	<=	<code>int i=1; (i <= 1)</code>
É maior que	>	<code>int i=2; (i > 1)</code>
É maior que ou igual a	>=	<code>int i=1; (i >= 1)</code>

Operadores Relacionais: Exemplo

```
public static void main(String args[]) {  
    int a = 10;  
    int b = 20;  
    System.out.println(a == b);  
    System.out.println(a != b);  
    System.out.println(a > b);  
    System.out.println(a < b);  
    System.out.println(b >= a);  
    System.out.println(b <= a);  
}
```

Para os valores das
primitivas, == verifica o
teste de igualdade

Observação: use o sinal de igual (=) para fazer uma atribuição e use o sinal == para fazer uma comparação e retornar um valor `booleano`.

Instruções Condicionais

- As instruções condicionais permitem escolher quais instruções serão executadas a seguir.
- Essas decisões baseiam-se em expressões booleanas (ou condições) que resultam em `true` ou `false`.
- Estas são as instruções condicionais em Java:
 - Instrução `if`
 - Instrução `if/else`
 - Instrução `switch`

Entendendo a Instrução `if`

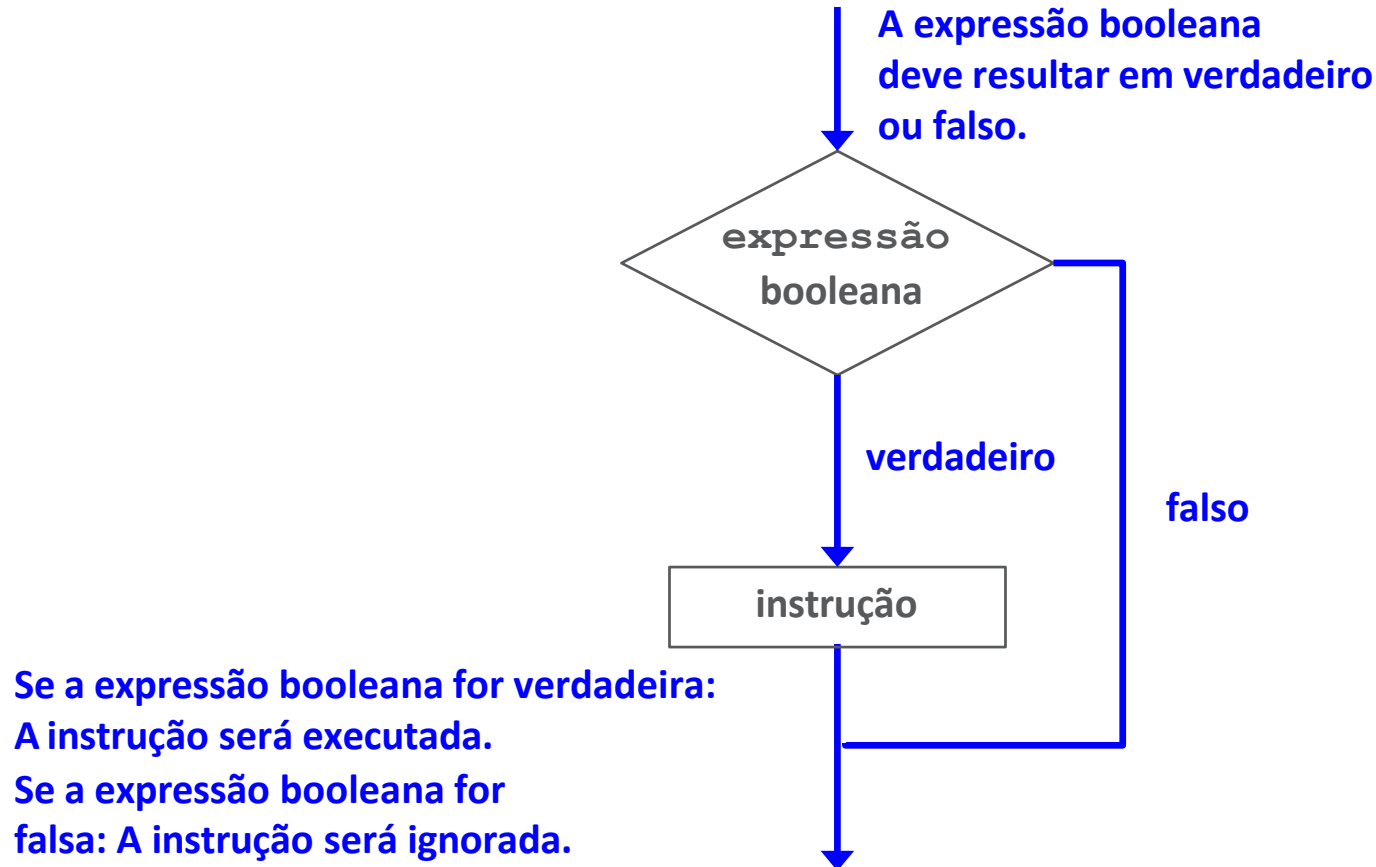
- Uma instrução `if` consiste em uma expressão booleana seguida de uma ou mais instruções.
- Sintaxe:

expressão booleana

```
if ( <alguma condição é verdadeira> ) {
```

```
//Instruções serão executadas se a expressão  
//booleana for verdadeira  
}
```

Entendendo a Instrução `if`



Usando Expressões booleanas em Instruções `if`

```
public static void main(String args[]) {  
    String left = "museu";  
    String straight = "ginástica";  
    String right = "restaurante";  
    if (left == "ginástica") {  
        System.out.println("Virar à Esquerda");  
    }  
  
    if (straight == "ginástica") {  
        System.out.println("Seguir em Frente");  
    }  
  
    if (right == "ginástica") {  
        System.out.println("Virar à Direita");  
    }  
}
```

Este bloco é executado.

Executando um Bloco de Código

- Um bloco de código não é necessário para uma instrução ser executada por uma instrução `if`.
- Veja um exemplo a seguir:

```
daysInFeb = 28;  
if(isLeapYear)  
    daysInFeb = 29;  
    System.out.println(ano + "é um ano bissexto");
```

Só essa instrução é executada.

Executando um Bloco de Código

2


```
daysInFeb = 28;  
if(isLeapYear){  
    daysInFeb = 29;  
    System.out.println(ano + "é um ano bissexto");  
}
```

Este bloco é
executado.

No entanto, sempre é recomendado que você use blocos de código, mesmo que só haja uma instrução para ser executada no bloco.

Instrução `if`: Exemplos

```
public static void main(String args[]) {  
    int grade = 85;  
    if (grade > 88) {  
        System.out.println("Você formou-se com Louvor.");  
    }  
    if (grade <=88) {  
        System.out.println("Você está apto a ensinar.");  
    }  
}
```



Segunda
instrução `if`

Saída:

Você está apto a ensinar.

Escolhendo entre Duas Alternativas

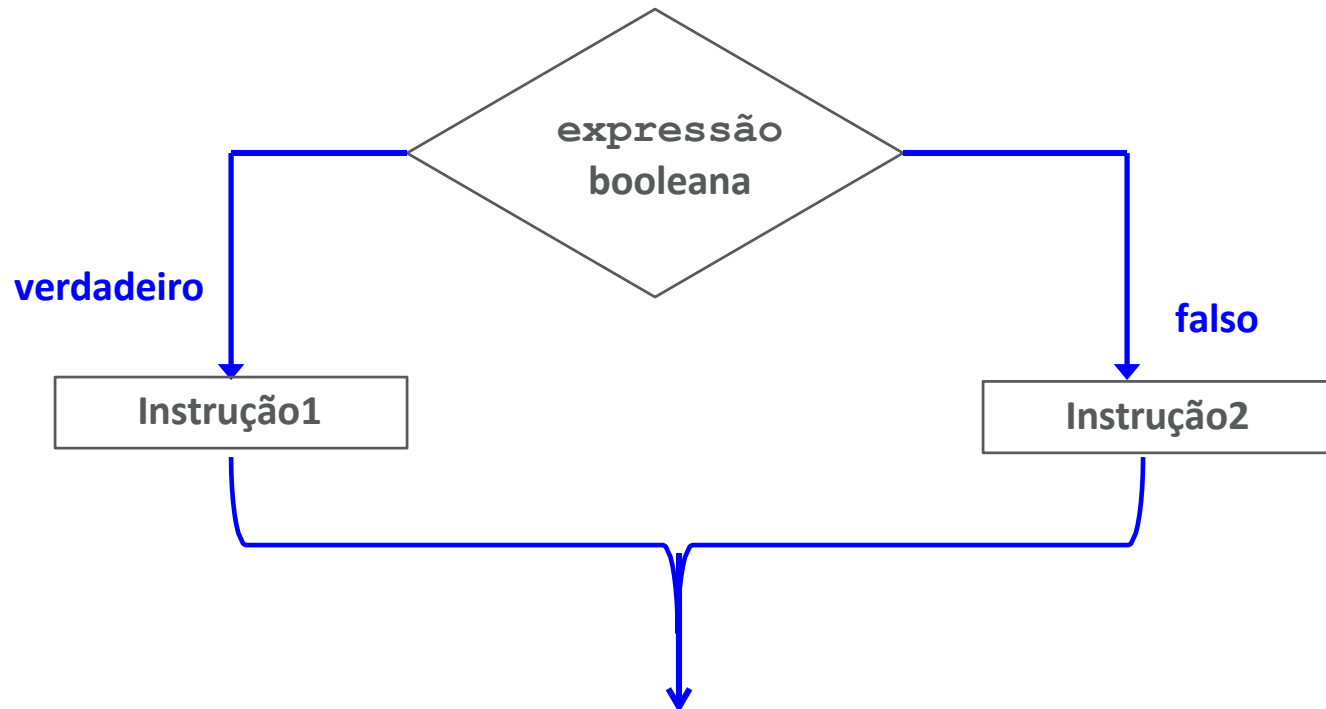
- Se você quiser escolher entre duas alternativas, use a instrução `if/else`.
- Sintaxe:

```
      expressão booleana
if ( <alguma condição for verdadeira> ) {
    // fazer algo
}
else {
    // fazer algo diferente
}
```

bloco if

bloco else

Entendendo Instruções `if/else`



Se a expressão `booleana` for verdadeira: a Instrução1 será executada.
Se a expressão `booleana` for falsa: a Instrução2 será ignorada.

Instruções if/else: Exemplo 1

```
String forecast;  
double temperature = getTemperature();
```

```
if (temperature <= 32.0) {  
    forecast = "NEVE";  
}
```

```
else {  
    forecast = "CHUVA";  
}
```

Este bloco é executado.



30.3 °F

Instruções if/else: Exemplo 2

```
String forecast;  
double temperature = getTemperature();  
  
if (temperature <= 32.0) {  
    forecast = "NEVE";  
}  
else {  
    forecast = "CHUVA";  
}
```

Este bloco é executado.



40.2 °F

Instruções `if/else`: Exemplo 3

```
public static void main(String args[]) {  
    int grade = 85;  
    if (grade > 88) {  
        System.out.println("Você formou-se com Louvor.");  
    }  
    else {  
        System.out.println("Você passou.");  
    }  
}
```

- Você pode substituir as duas instruções `if` por uma instrução `if/else`.
- A instrução `if/else` é mais eficiente porque só uma comparação está sendo feita.

Comparando Variáveis

- Quando compara valores usando expressões `booleana`, você precisa entender as nuances de determinados tipos de dados.
- Operadores relacionais como `==` são...
 - Excelentes para comparar primitivas
 - Ruins para comparar `Strings` (e outros objetos)
- Vamos analisar por quê.

Comparando Primitivas

- O valor `z` é definido como a soma de `x + y`.
- Quando uma expressão `booleana` testa a igualdade entre `z` e a soma de `x + y`, o resultado é `true`.

```
int x = 3;
int y = 2;
int z = x + y;

boolean test = (z == x + y);
System.out.println(test);           //true
```

Comparando Strings

- O valor `z` é definido como sendo a concatenação de `x + y`.
- Quando uma expressão `booleana` testa a igualdade entre `z` e a concatenação de `x + y`, o resultado é `false`.

```
String x = "Ora";  
String y = "cle";  
String z = x + y;  
  
boolean test = (z == x + y);  
System.out.println(test);           //false
```

Por quê?



Por que Há Resultados Contraditórios?

- As primitivas e os objetos são armazenados de forma diferente na memória.
 - As `Strings` recebem um tratamento especial.
 - Isso será discutido mais adiante no curso.
- Como resultado...
 - `==` compara os valores das primitivas.
 - `==` compara os locais dos objetos na memória.
- É mais provável que você precise comparar o conteúdo de `Strings` e não suas localizações na memória.

Como Você Deve Comparar Objetos `String`?

- Você nunca deve comparar `Strings` usando `==`.
- Em vez disso, compare `Strings` usando o método `equals()`.
 - Esse método é parte da classe `String`.
 - Ele aceita um argumento `String`, verifica se o conteúdo das `Strings` é igual e retorna um `booleano`.
 - Também existe um método semelhante: `equalsIgnoreCase()`.

```
String x = "Ora";  
String y = "cle";  
String z = x + y;  
boolean test = z.equals(x + y);  
System.out.println(test);           //true
```

Exercícios

Crie um programa que leia um número inteiro digitado pelo usuário e classifique-o como positivo, negativo ou zero, exibindo a mensagem correspondente para cada caso.

Escreva um programa que solicite o valor total de uma compra. Se o valor for maior ou igual a R\$ 200, o programa deve aplicar um desconto de 10% e mostrar o valor com desconto. Caso contrário, o valor permanece o mesmo.

Crie um programa que leia um número inteiro digitado pelo usuário e classifique-o como positivo, negativo ou zero, exibindo a mensagem correspondente para cada caso.

Objetivos

Esta lição abrange os seguintes objetivos:

- Descrever a execução condicional
- Descrever operadores lógicos
- Entender a avaliação de operadores lógicos de “curto-circuito”
- Criar construções `if` encadeadas

Quando Várias Condições se Aplicam

- E se determinada ação precisar ser tomada apenas se várias condições forem verdadeiras?
- Considere o cenário em que um aluno estará qualificado a receber uma bolsa de estudos se as duas condições a seguir forem atendidas:
 - Nota ≥ 88 .
 - Número de faltas = 0.

Tratando Várias Condições

- Operadores relacionais são uma boa opção quando você está verificando uma única condição.
- Você pode usar uma sequência de instruções `if` para testar mais de uma condição.

```
if (grade >= 88) {  
    if (numberDaysAbsent == 0) {  
        System.out.println("Você está qualificado para uma  
        bolsa de estudos.");  
    }  
}
```


Tratando Várias Condições Exemplo

Como é demonstrado no exemplo:

- A sequência de instruções `if` é difícil de ser escrita, mais difícil de ser lida e torna-se ainda mais difícil quando você adiciona mais condições.
- Felizmente, o Java tem uma maneira fácil de tratar várias condições: **operadores lógicos**.

Operadores Lógicos Java

Você pode usar três operadores lógicos Java para combinar várias expressões booleanas em uma única expressão `booleana`.

Operador Lógico	Significado
<code>&&</code>	E
<code> </code>	OU
<code>!</code>	NÃO

Três Operadores Lógicos

Operação	Operador	Exemplo
Se uma condição E outra condição	&&	<pre>int i = 2; int j = 8; ((i < 1) && (j > 6))</pre>
Se uma das condições OU as duas condições		<pre>int i = 2; int j = 8; ((i < 1) (j > 10))</pre>
NÃO	!	<pre>int i = 2; (! (i < 3))</pre>

Aplicando Operadores Lógicos

- Você pode escrever o exemplo anterior usando o operador lógico AND como:

```
grade >= 88 && numberDaysAbsent == 0
```

Expressão Booleana 1 Operador Lógico Expressão Booleana 2

- O operador lógico permite que você teste várias condições mais facilmente, e o código é mais legível.

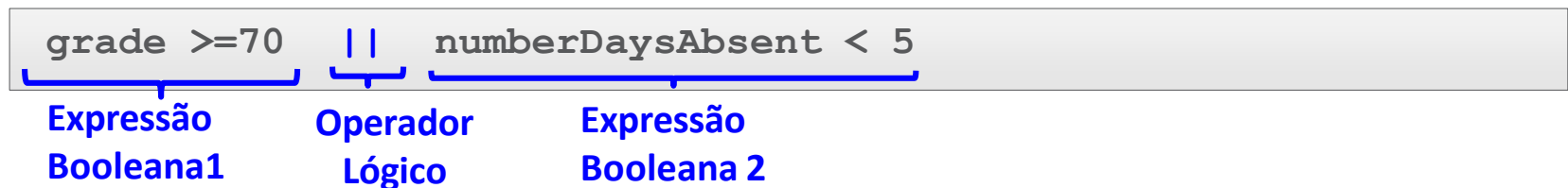
Operador Lógico E: Exemplo

```
public static void main(String[] args) {  
    int numberDaysAbsent = 0;  
    int grade = 95;  
    if (grade >= 88 && numberDaysAbsent == 0) {  
        System.out.println("Você está qualificado para uma bolsa  
        de estudos.");  
    }  
    else {  
        System.out.println("Você não está qualificado para uma bolsa de  
        estudos.");  
    }  
}
```

Será avaliado como verdadeiro se as duas expressões booleanas forem verdadeiras.

Operadores Lógicos OU

- Considere o cenário em que um aluno estará qualificado para participar de uma equipe esportiva se uma destas condições for atendida:
 - Nota ≥ 70
 - Número de faltas < 5
- Neste caso, você pode usar o operador lógico OU para unir as várias expressões booleanas.



Operadores Lógicos OU: Exemplo

```
public static void main(String[] args) {  
    int numberDaysAbsent = 3;  
    int grade = 85;  
    if (grade >= 70 || numberDaysAbsent < 5) {  
        System.out.println("Você está qualificado para uma equipe  
        esportiva");  
    }  
    else {  
        System.out.println("Você não está qualificado para uma  
        equipe esportiva");  
    }  
}
```

Será avaliada como verdadeira se uma das expressões booleanas for verdadeira.

Operadores Lógicos NÃO

- Considere o cenário em que um aluno estará qualificado para monitoria gratuita se as duas condições a seguir forem atendidas:
 - Nota < 88
 - Número de faltas < 3
- Use o operador lógico !.

```
!madeFreeTutor && numberDaysAbsent < 3
```

Operador
Lógico

Expressão
Booleana 1

Expressão
Booleana 2

Operadores Lógicos NÃO

```
public static void main(String args[]) {  
    int numberDaysAbsent = 2;  
  
    int grade = 65;  
  
    boolean madeFreeTutor = grade >= 88;  
  
    if (!madeFreeTutor && numberDaysAbsent < 3) {  
        System.out.println("Você está qualificado a receber  
        monitoria gratuita");  
    }  
}
```

Ignorando o Segundo Teste E

- Os operadores && e || são operadores de curto-circuito.
- Se a primeira expressão (à esquerda) for falsa, não haverá necessidade de calcular a segunda (à direita).

```
b = (x != 0) && ((y / x) > 2);
```

Expressão à
Esquerda

Expressão
à Direita

Ignorando o Segundo Teste E

```
b = (x != 0) && ((y / x) > 2);
```

Expressão à
Esquerda

Expressão
à Direita

- Se x for 0, então $(x \neq 0)$ será falso.
- Para o operador $\&\&$, como não importa se $((y/x) > 2)$ é `true` ou `false`, o resultado dessa expressão é `false`.
- Então, o Java não continua a calcular $((y/x) > 2)$.

Ignorando o Segundo Teste OU

- Se a primeira expressão (à esquerda) for `true`, não haverá necessidade de calcular a segunda (à direita).
- Considere este exemplo:

```
boolean b = (x <= 10) || (x > 20);
```

Expressão à
Esquerda

Expressão
à Direita

- Se $(x \leq 10)$ for verdadeiro, $(x > 20)$ não será calculado porque não importa se $(x > 20)$ é verdadeiro ou falso.
- O resultado desta expressão será `true`.

O que é um Operador Condicional Ternário?

Operação	Operador	Exemplo
Se a condição for verdadeira, atribua resultado = valor1. Caso contrário, atribua resultado = valor2. Observação: valor1 e valor2 devem ser do mesmo tipo de dados.	?:	resultado = condição ? valor1 : valor2 Exemplo: int x = 2, y = 5, z = 0; z = (y < x) ? x : y;

Instruções equivalentes

```
z = (y < x) ? x : y;
```

```
if (y < x) {  
    z = x;  
}  
else {  
    z = y;  
}
```

Operador Condicional Ternário: Cenário

Suponha que você esteja jogando futebol e esteja controlando os gols da seguinte forma:

```
public static void main(String args[]) {  
    int numberOfGoals = 5;  
    String s;  
    if (numberOfGoals == 1) {  
        s = "gol";  
    }  
    else {  
        s = "gols";  
    }  
    System.out.println("Marquei " + numberOfGoals + " " + s);  
}
```

Operador Condicional Ternário: Exemplo

Um resultado semelhante é obtido com o operador ternário substituindo toda a instrução `if/else` por uma única linha.

```
int numberOfGoals = 1  
System.out.println("Marquei " +numberOfGoals + " "  
+(numberOfGoals==1 ? "gol" : "gols") );
```

Operador Condicional Ternário: Exemplo

- Vantagem: insere a operação diretamente dentro de uma expressão.

```
int numberOfGoals = 1;  
String s = (numberOfGoals==1 ? "gol" : "gols");  
System.out.println("Marquei " +numberOfGoals + " "+s );
```

- Desvantagem: pode ter apenas dois resultados possíveis.

```
(numberOfGoals==1 ? "gol" : "gols" : "Mais gols");
```

booleano

verdadeiro

falso

???

Tratando Condições Complexas com uma Construção `if` Encadeada

A instrução `if` encadeada:

- Conecta várias condições em uma única construção
- Tende a ser confusa de ser lida e difícil de ser mantida

Encadeando Construções `if/else`

- Você pode encadear construções `if` e `else` juntas para definir vários resultados para diversas expressões diferentes.
- Sintaxe:

```
if (<condition1>) {  
    //code_block1  
}  
else if (<condition2>) {  
    // code_block2  
}  
else {  
    // default_code  
}
```

Encadeando Construções if/else: Exemplo

```
public static void main(String args[]) {  
    double income = 30000, tax;  
    if (income <= 15000) {  
        tax = 0;  
    }  
    else if (income <= 25000) {  
        tax = 0.05 * (income - 15000);  
    }  
    else {  
        tax = 0.05 * (income - (25000 - 15000));  
        tax += 0.10 * (income - 25000);  
    }  
}
```

É Possível Aninhar Instruções `if`?

- Em Java, uma instrução `if` pode estar presente dentro do corpo de outra instrução `if`.

```
if (tvType == "em cores") {  
    if (size == 14) {  
        discPercent = 8;  
    }  
    else {  
        discPercent = 10;  
    }  
}
```

- Neste exemplo, a instrução `else` é combinada com a instrução `if (size==14)`.

Entendendo Instruções `if` Aninhadas

```
if (tvType == "em cores") {  
    if (size == 14) {  
        discPercent = 8;  
    }  
}  
else {  
    discPercent = 10;  
}
```

Neste exemplo, a instrução `else` é combinada com a instrução `if` externa (`TVType=="em cores"`).

Exercícios

Escreva um programa que leia o peso (em kg) e a altura (em metros) de uma pessoa. Calcule o IMC e use if-else encadeado para classificar a pessoa como:

- Abaixo do peso ($\text{IMC} < 18.5$)
- Peso normal (IMC entre 18.5 e 24.9)
- Sobrepeso (IMC entre 25.0 e 29.9)
- Obesidade ($\text{IMC} \geq 30.0$)

Escreva um programa que leia um ano e determine se ele é bissexto. Use if-else para aplicar a regra:

- Um ano é bissexto se é divisível por 4, mas não por 100, a menos que também seja divisível por 400.

Depois, reescreva essa verificação usando um operador ternário para imprimir "Ano bissexto" ou "Ano comum".

Exercícios

Crie um programa que leia a nota de um aluno (de 0 a 10) e informe:

- "Reprovado" se a nota for menor que 5
- "Recuperação" se for entre 5 e 6.9
- "Aprovado" se for 7 ou mais

Use if-else encadeado para implementar a lógica e, ao final, mostre a mesma classificação usando operador ternário (apenas para treinar).

Solicite ao usuário três números inteiros e utilize if-else encadeado para determinar qual é o maior entre eles. Depois, escreva uma segunda versão do programa que utilize operador ternário aninhado para fazer a mesma verificação.

Exercícios

Crie um programa que leia a nota de um aluno (de 0 a 10) e informe:

- "Reprovado" se a nota for menor que 5
- "Recuperação" se for entre 5 e 6.9
- "Aprovado" se for 7 ou mais

Use if-else encadeado para implementar a lógica e, ao final, mostre a mesma classificação usando operador ternário (apenas para treinar).

Solicite ao usuário três números inteiros e utilize if-else encadeado para determinar qual é o maior entre eles. Depois, escreva uma segunda versão do programa que utilize operador ternário aninhado para fazer a mesma verificação.

Desafio prático: Criando um menu inteligente com `switch`

Imagine que você está desenvolvendo um sistema de menu automatizado para um projeto de simulação em sala de aula. O objetivo é permitir que o usuário selecione opções de atendimento como se estivesse ligando para uma central de serviços. Sua missão é descobrir como a estrutura `switch` pode ser usada para lidar com múltiplas opções de forma eficiente e organizada.

Desafio de busca ativa:

Sem receber o código pronto, explore a documentação do Java, vídeos, exemplos ou testes no seu editor para descobrir como usar a instrução `switch`. A partir dessa pesquisa, desenvolva um programa com o seguinte menu:

MENU DE ATENDIMENTO

- 1 - Suporte técnico
- 2 - Financeiro
- 3 - Falar com atendente
- 4 - Cancelar serviço
- 5 - Encerrar atendimento

Quando o usuário digitar um número de 1 a 5, o programa deve exibir uma mensagem correspondente. Se for digitado qualquer outro número, o programa deve responder com: "Opção inválida. Tente novamente."